Package ‘fda.usc’

November 9, 2024

Type Package

Title Functional Data Analysis and Utilities for Statistical Computing

Version 2.2.0

Date 2024-11-04

Depends R (>= 3.5.0), fda, splines, MASS, mgcv, knitr

Imports methods, grDevices, graphics, utils, stats, nlme, doParallel,
parallel, iterators, foreach, kSamples

Suggests rmarkdown

Description Routines for exploratory and descriptive analysis of functional data such as depth measurements, atypical curves detection, regression models, supervised classification, unsupervised classification and functional analysis of variance.

License GPL-2

URL `https://github.com/moviedo5/fda.usc`,
`https://moviedo5.github.io/fda.usc/`,
`https://www.jstatsoft.org/v51/i04/`

BugReports `https://github.com/moviedo5/fda.usc/issues`

LazyLoad yes

Author Manuel Febrero Bande [aut] (<`https://orcid.org/0000-0002-9536-2973`>),
Manuel Oviedo de la Fuente [aut, cre]
(<`https://orcid.org/0000-0001-7360-3249`>),
Pedro Galeano [ctb],
Alicia Nieto [ctb],
Eduardo Garcia-Portugues [ctb]

Maintainer Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

NeedsCompilation yes

Repository CRAN

Encoding UTF-8

RoxygenNote 7.3.2

Date/Publication 2024-11-09 18:40:02 UTC

Contents

`fda.usc-package` *Functional Data Analysis and Utilities for Statistical Computing (fda.usc)*

Description

This devel version carries out exploratory and descriptive analysis of functional data exploring its most important features: such as depth measurements or functional outliers detection, among others. It also helps to explain and model the relationship between a dependent variable and independent (regression models) and make predictions. Methods for supervised or unsupervised classification of a set of functional data regarding a feature of the data are also included. Finally, it can perform analysis of variance model (ANOVA) for functional data.

Details

Sections of fda.usc-package:

A.- Functional Data Representation
B.- Functional Outlier Detection

C.- Functional Regression Model
D.- Functional Supervised Classification
E.- Functional Non-Supervised Classification
F.- Functional ANOVA
G.- Auxiliary functions

A.- Functional Data Representation
The functions included in this section allow to define, transform, manipulate and represent a functional dataset in many ways including derivatives, non-parametric kernel methods or basis representation.

`fdata`
`plot.fdata`
`fdata.deriv`
`CV.S`
`GCV.S`
`optim.np`
`optim.basis`
`S.NW`
`S.LLR`
`S.basis`
`Var.e`
`Var.y`

B.- Functional Depth and Functional Outlier Detection

The functional data depth calculated by the different depth functions implemented that could be use as a measure of centrality or outlyingness.

B.1-Depth methods `Depth`:

`depth.FM`
`depth.mode`
`depth.RP`
`depth.RT`
`depth.RPD`
`Descriptive`

B.2-Functional Outliers detection methods:

`outliers.depth.trim`

outliers.depth.pond
outliers.thres.lrt
outliers.lrt

C.- Functional Regression Models

C.1. Functional explanatory covariate and scalar response
The functions included in this section allow the estimation of different functional regression models with a scalar response and a single functional explicative covariate.

fregre.pc
fregre.pc.cv
fregre.pls
fregre.pls.cv
fregre.basis
fregre.basis.cv
fregre.np
fregre.np.cv

C.2. Test for the functional linear model (FLM) with scalar response.

flm.Ftest, F-test for the FLM with scalar response
flm.test, Goodness-of-fit test for the FLM with scalar response
PCvM.statistic, PCvM statistic for the FLM with scalar response

C.3. Functional and non functional explanatory covariates.
The functions in this section extends those regression models in previous section in several ways.

fregre.plm: Semifunctional Partial Linear Regression (an extension of lm model)
fregre.lm: Functional Linear Regression (an extension of lm model)
fregre.glm: Functional Generalized Linear Regression (an extension of glm model)
fregre.gsam: Functional Generalized Spectral Additive Regression (an extension of gam model)
fregre.gkam: Functional Generalized Kernel Additive Regression (an extension of fregre.np model)

C.4. Functional response model (fregre.basis.fr) allows the estimation of functional regression models with a functional response and a single functional explicative covariate.

C.5. fregre.gls fits functional linear model using generalized least squares. fregre.igls fits iteratively a functional linear model using generalized least squares.

C.6. `fregre.gsam.vs`, Variable Selection using Functional Additive Models

D.- Functional Supervised Classification
This section allows the estimation of the groups in a training set of functional data `fdata` class by different nonparametric methods of supervised classification. Once these classifiers have been trained, they can be used to predict on new functional data.

Package allows the estimation of the groups in a training set of functional data by different methods of supervised classification.

D.1 Univariate predictor (x,y arguments, fdata class)

`classif.knn`
`classif.kernel`

D.2 Multiple predictors (formula,data arguments, ldata class)

`classif.glm`
`classif.gsam`
`classif.gkam`

D.3 Depth classifiers (fdata or ldata class)

`classif.DD`
`classif.depth`

D.4 Functional Classification usign k-fold CV

`classif.kfold`

E.- Functional Non-Supervised Classification
This section allows the estimation of the groups in a functional data set `fdata` class by kmeans method.

`kmeans.fd`

F.- Functional ANOVA

fanova.onefactor
fanova.RPm
fanova.hetero

G.- Utilities and auxiliary functions:

fdata.bootstrap
fdata2fd
fdata2pc
fdata2pls
summary.fdata.comp
cond.F
cond.quantile
cond.mode
FDR
Kernel
Kernel.asymmetric
Kernel.integrate
metric.lp
metric.kl
metric.DTW
metric.hausdorff
metric.dist
semimetric.NPFDA
semimetric.basis

Package:	fda.usc
Type:	Package
Version:	2.0.3
Date:	2021-06-03
License:	GPL-2
LazyLoad:	yes
https://github.com/moviedo5/fda.usc/	

Author(s)

Authors: Manuel Febrero Bande <manuel.febrero@usc.es> and Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

Contributors: Pedro Galeano, Alicia Nieto-Reyes, Eduardo Garcia-Portugues <eduardo.garcia@usc.es> and STAPH group https://www.math.univ-toulouse.fr/~ferraty/

Maintainer: Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. doi:10.18637/jss.v051.i04

`accuracy` *Performance measures for regression and classification models*

Description

`cat2meas` and `tab2meas` calculate the measures for a multiclass classification model.
`pred2meas` calculates the measures for a regression model.

Usage

```
cat2meas(yobs, ypred, measure = "accuracy", cost = rep(1, nlevels(yobs)))

tab2meas(tab, measure = "accuracy", cost = rep(1, nrow(tab)))

pred.MSE(yobs, ypred)

pred.RMSE(yobs, ypred)

pred.MAE(yobs, ypred)

pred2meas(yobs, ypred, measure = "RMSE")
```

Arguments

`yobs`	A vector of the labels, true class or observed response. Can be `numeric`, `character`, or `factor`.
`ypred`	A vector of the predicted labels, predicted class or predicted response. Can be `numeric`, `character`, or `factor`.
`measure`	Type of measure, see `details` section.
`cost`	Cost value by class (only for input factors).
`tab`	Confusion matrix (Contingency table: observed class by rows, predicted class by columns).

Details

- `cat2meas` compute $tab = table(yobs, ypred)$ and calls `tab2meas` function.
- `tab2meas` function computes the following measures (see `measure` argument) for a binary classification model:
  - `accuracy`: Proportion of correct predictions. $\frac{TP+TN}{TP+TN+FP+FN}$
  - `sensitivity`, `TPrate`, `recall`: True Positive Rate or recall. $\frac{TP}{TP+FN}$

- `precision`: Positive Predictive Value. $\frac{TP}{TP+FP}$
- `specificity`, `TNrate`: True Negative Rate. $\frac{TN}{TN+FP}$
- `FPrate`: False Positive Rate. $\frac{FP}{TN+FP}$
- `FNrate`: False Negative Rate. $\frac{FN}{TP+FN}$
- `Fmeasure`: Harmonic mean of precision and recall. $\frac{2}{\frac{1}{\text{recall}}+\frac{1}{\text{precision}}}$
- `Gmean`: Geometric Mean of recall and specificity. $\sqrt{\left(\frac{TP}{TP+FN}\right)\cdot\left(\frac{TN}{TN+FP}\right)}$
- `kappa`: Cohen's Kappa index. $Kappa = \frac{P_o - P_e}{1-P_e}$ where P_o is the proportion of observed agreement, $\frac{TP+TN}{TP+TN+FP+FN}$, and P_e is the proportion of agreement expected by chance, $\frac{(TP+FP)(TP+FN)+(TN+FN)(TN+FP)}{(TP+TN+FP+FN)^2}$.
- `cost`: Weighted accuracy, calculated as $\frac{\sum(\text{diag(tab)/rowSums(tab)}\cdot\text{cost})}{\sum(\text{cost})}$
- `IOU`: Mean Intersection over Union. $\frac{TP}{TP+FN+FP}$
- `IOU4class`: Intersection over Union by class level. $\frac{TP}{TP+FN+FP}$#'

- `pred2meas` function computes the following measures of error, usign the `measure` argument, for observed and predicted vectors:
  - `MSE`: Mean squared error, $\frac{\sum(ypred-yobs)^2}{n}$.
  - `RMSE`: Root mean squared error $\sqrt{\frac{\sum(ypred-yobs)^2}{n}}$.
  - `MAE`: Mean Absolute Error, $\frac{\sum|yobs-ypred|}{n}$.

See Also

Other performance: `weights4class()`

`aemet` *aemet data*

Description

Series of daily summaries of 73 spanish weather stations selected for the period 1980-2009. The dataset contains geographic information of each station and the average for the period 1980-2009 of daily temperature, daily precipitation and daily wind speed.

Format

Elements of aemet:
`..$df`: Data frame with information of each wheather station:

- `ind`: Indicated weather station.
- `name`: Station Name. 36 marked UTF-8 strings.
- `province`:Province (region) of Spain. 36 marked UTF-8 strings
- `altitude`: Altitude of the station (in meters).

- `year.ini`: Start year.
- `year.end`: End year.
- `longitude`: x geographic coordinate of the station (in decimal degrees).
- `latitude`: y geographic coordinate of the station (in decimal degrees).

The functional variables:

- `...$temp`: mean curve of the average daily temperature for the period 1980-2009 (in degrees Celsius, marked with UTF-8 string). In leap years temperatures for February 28 and 29 were averaged.
- `...$wind.speed`: mean curve of the average daily wind speed for the period 1980-2009 (in m/s).
- `...$logprec`: mean curve of the log precipitation for the period 1980-2009 (in log mm). Negligible precipitation (less than 1 tenth of mm) is replaced by `0.05` and no precipitation (0.0 mm) is replaced by `0.01`. Then the logarithm is applied.

Details

Meteorological State Agency of Spain (AEMET), `https://www.aemet.es/es/portada`. Government of Spain.
It marks 36 UTF-8 string of names of stations and 3 UTF-8 string names of provinces through the function `iconv`.

Author(s)

Manuel Febrero Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

Source

The data were obtained from the FTP of AEMET in 2009.

Examples

```
## Not run:
data(aemet)
names(aemet)
names(aemet$df)
lat <- ifelse(aemet$df$latitude<31,"red","blue")
plot(aemet,col=lat)

## End(Not run)
```

classif.DD *DD-Classifier Based on DD-plot*

Description

Fits Nonparametric Classification Procedure Based on DD–plot (depth-versus-depth plot) for G dimensions ($G = g \times h$, g levels and p data depth).

Usage

```
classif.DD(
  group,
  fdataobj,
  depth = "FM",
  classif = "glm",
  w,
  par.classif = list(),
  par.depth = list(),
  control = list(verbose = FALSE, draw = TRUE, col = NULL, alpha = 0.25)
)
```

Arguments

`group`	Factor of length *n* with *g* levels.
`fdataobj`	`data.frame`, `fdata` or `list` with the multivariate, functional or both covariates respectively.
`depth`	Character vector specifying the type of depth functions to use, see `Details`.
`classif`	Character vector specifying the type of classifier method to use, see `Details`.
`w`	Optional case weights, weights for each value of `depth` argument, see `Details`.
`par.classif`	List of parameters for `classif` procedure.
`par.depth`	List of parameters for `depth` function.
`control`	List of parameters for controlling the process. If `verbose=TRUE`, report extra information on progress. If `draw=TRUE` print DD-plot of two samples based on data depth. `col`, the colors for points in DD–plot. `alpha`, the alpha transparency used in the background of DD–plot, a number in [0,1].

Details

Make the group classification of a training dataset using DD-classifier estimation in the following steps.

1. The function computes the selected `depth` measure of the points in `fdataobj` w.r.t. a subsample of each g level group and p data dimension ($G = g \times p$). The user can be specify the parameters for depth function in `par.depth`.

   (i) Type of depth function from functional data, see `Depth`:

   - `"FM"`: Fraiman and Muniz depth.
   - `"mode"`: h-modal depth.
   - `"RT"`: Random Tukey depth.
   - `"RP"`: Random projection depth.
   - `"RPD"`: Double random projection depth.

   (ii) Type of depth function from multivariate functional data, see `depth.mfdata`:

   - `"FMp"`: Fraiman and Muniz depth with common support. Suppose that all p-fdata objects have the same support (same rangevals); see `depth.FMp`.
   - `"modep"`: h-modal depth using a p-dimensional metric; see `depth.modep`.
   - `"RPp"`: Random projection depth using a p-variate depth with the projections; see `depth.RPp`.

   If the procedure requires to compute a distance such as in `"knn"` or `"np"` classifier or `"mode"` depth, the user must use a proper distance function: `metric.lp` for functional data and `metric.dist` for multivariate data.

   (iii) Type of depth function from multivariate data, see `Depth.Multivariate`:

   - `"SD"`: Simplicial depth (for bivariate data).
   - `"HS"`: Half-space depth.
   - `"MhD"`: Mahalanobis depth.
   - `"RD"`: Random projections depth.
   - `"LD"`: Likelihood depth.

2. The function calculates the misclassification rate based on data depth computed in step (1) using the following classifiers.

   - `"MaxD"`: Maximum depth.
   - `"DD1"`: Search for the best separating polynomial of degree 1.
   - `"DD2"`: Search for the best separating polynomial of degree 2.
   - `"DD3"`: Search for the best separating polynomial of degree 3.
   - `"glm"`: Logistic regression computed using Generalized Linear Models; see `classif.glm`.
   - `"gam"`: Logistic regression computed using Generalized Additive Models; see `classif.gsam`.
   - `"lda"`: Linear Discriminant Analysis computed using lda.
   - `"qda"`: Quadratic Discriminant Analysis computed using qda.
   - `"knn"`: k-Nearest Neighbour classification computed using `classif.knn`.
   - `"np"`: Non-parametric Kernel classifier computed using `classif.np`.

   The user can be specify the parameters for classifier function in `par.classif` such as the smoothing parameter `par.classif[["h"]]`, if `classif="np"` or the k-Nearest Neighbour `par.classif[["knn"]]`, if `classif="knn"`.

   In the case of polynomial classifier (`"DD1"`, `"DD2"` and `"DD3"`) uses the original procedure proposed by Li et al. (2012), by defalut rotating the DD-plot (to exchange abscise and ordinate) using in `par.classif` argument `rotate=TRUE`. Notice that the maximum depth classifier can be considered as a particular case of DD1, fixing the slope with a value of 1 (`par.classif=list(pol=1)`).

The number of possible different polynomials depends on the sample size `n` and increases polynomially with order k. In the case of g groups, so the procedure applies some multiple-start optimization scheme to save time:

- Generate all combinations of the elements of \(n \) taken \(k \) at a time: $g \times combn(N, k)$ candidate solutions. When this number exceeds `nmax=10000`, a random sample of `10000` combinations is selected.
- Smooth the empirical loss with the logistic function: $1/(1 + e^{-tx})$. The classification rule is constructed by optimizing the best `noptim` combinations in this random sample (by default, `noptim=1` and `tt=50/range(depth values)`). Note that Li et al. found that the optimization results become stable for $t \in [50, 200]$ when the depth is standardized with an upper bound of 1.

The original procedure (Li et al. (2012)) not need to try many initial polynomials (`nmax=1000`) and that the procedure optimize the best (`noptim=1`), but we recommended to repeat the last step for different solutions, as for example `nmax=250` and `noptim=25`. User can change the parameters `pol`, `rotate`, `nmax`, `noptim` and `tt` in the argument `par.classif`.

The `classif.DD` procedure extends to multi-class problems by incorporating the method of *majority voting* in the case of polynomial classifier and the method *One vs the Rest* in the logistic case (`"glm"` and `"gam"`).

Value

- `group.est`: Estimated vector groups by classified method selected.
- `misclassification`: Probability of misclassification.
- `prob.classification`: Probability of correct classification by group level.
- `dep`: Data frame with the depth of the curves for functional data (or points for multivariate data) in `fdataobj` w.r.t. each `group` level.
- `depth`: Character vector specifying the type of depth functions used.
- `par.depth`: List of parameters for `depth` function.
- `classif`: Type of classifier used.
- `par.classif`: List of parameters for `classif` procedure.
- `w`: Optional case weights.
- `fit`: Fitted object by `classif` method using the depth as covariate.

Author(s)

This version was created by Manuel Oviedo de la Fuente and Manuel Febrero Bande and includes the original version for polynomial classifier created by Jun Li, Juan A. Cuesta-Albertos and Regina Y. Liu.

References

Cuesta-Albertos, J.A., Febrero-Bande, M. and Oviedo de la Fuente, M. *The DDG-classifier in the functional setting*, (2017). Test, 26(1), 119-142. DOI: doi:10.1007/s1174901605026.

See Also

See Also as `predict.classif.DD`

Examples

```
## Not run:
# DD-classif for functional data
data(tecator)
ab <- tecator$absorp.fdata
ab1 <- fdata.deriv(ab, nderiv = 1)
ab2 <- fdata.deriv(ab, nderiv = 2)
gfat <- factor(as.numeric(tecator$y$Fat>=15))

# DD-classif for p=1 functional  data set
out01 <- classif.DD(gfat,ab,depth="mode",classif="np")
out02 <- classif.DD(gfat,ab2,depth="mode",classif="np")
# DD-plot in gray scale
ctrl <- list(draw=T,col=gray(c(0,.5)),alpha=.2)
out02bis <- classif.DD(gfat,ab2,depth="mode",classif="np",control=ctrl)

# 2 depth functions (same curves)
ldat <- mfdata("ab" = ab, "ab2" = ab2)
out03 <- classif.DD(gfat,list(ab2,ab2),depth=c("RP","mode"),classif="np")
# DD-classif for p=2 functional data set
# Weighted version
out04 <- classif.DD(gfat, ldat, depth="mode",
                    classif="np", w=c(0.5,0.5))
# Model version
out05 <- classif.DD(gfat,ldat,depth="mode",classif="np")
# Integrated version (for multivariate functional data)
out06 <- classif.DD(gfat,ldat,depth="modep",classif="np")

# DD-classif for multivariate data
data(iris)
group <- iris[,5]
x <- iris[,1:4]
out07 <- classif.DD(group,x,depth="LD",classif="lda")
summary(out07)
out08 <- classif.DD(group, list(x,x), depth=c("MhD","LD"),
                    classif="lda")
summary(out08)

# DD-classif for functional data: g levels
data(phoneme)
mlearn <- phoneme[["learn"]]
glearn <- as.numeric(phoneme[["classlearn"]])-1
out09 <- classif.DD(glearn,mlearn,depth="FM",classif="glm")
out10 <- classif.DD(glearn,list(mlearn,mlearn),depth=c("FM","RP"),classif="glm")
summary(out09)
summary(out10)

## End(Not run)
```

classif.depth *Classifier from Functional Data*

Description

Classification of functional data using maximum depth.

Usage

```
classif.depth(
  group,
  fdataobj,
  newfdataobj,
  depth = "RP",
  par.depth = list(),
  CV = "none"
)
```

Arguments

`group`	Factor of length *n*
`fdataobj`	`fdata`, `matrix` or `data.frame` class object of train data.
`newfdataobj`	`fdata`, `matrix` or `data.frame` class object of test data.
`depth`	Type of depth function from functional data: • `FM`: Fraiman and Muniz depth. • `mode`: Modal depth. • `RT`: Random Tukey depth. • `RP`: Random project depth. • `RPD`: Double random project depth.
`par.depth`	List of parameters for `depth`.
`CV`	="none" `group.est=group.pred`, =TRUE `group.est` is estimated by cross-validation, =FALSE `group.est` is estimated.

Value

- `group.est`: Vector of classes of train sample data.
- `group.pred`: Vector of classes of test sample data.
- `prob.classification`: Probability of correct classification by group.
- `max.prob`: Highest probability of correct classification.
- `fdataobj`: `fdata` class object.
- `group`: Factor of length *n*.

Author(s)

Febrero-Bande, M. and Oviedo de la Fuente, M.

References

Cuevas, A., Febrero-Bande, M. and Fraiman, R. (2007). *Robust estimation and classification for functional data via projection-based depth notions.* Computational Statistics 22, 3, 481-496.

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme[["learn"]]
mtest<-phoneme[["test"]]
glearn<-phoneme[["classlearn"]]
gtest<-phoneme[["classtest"]]

a1<-classif.depth(glearn,mlearn,depth="RP")
table(a1$group.est,glearn)
a2<-classif.depth(glearn,mlearn,depth="RP",CV=TRUE)
a3<-classif.depth(glearn,mlearn,depth="RP",CV=FALSE)
a4<-classif.depth(glearn,mlearn,mtest,"RP")
a5<-classif.depth(glearn,mlearn,mtest,"RP",CV=TRUE)
table(a5$group.est,glearn)
a6<-classif.depth(glearn,mlearn,mtest,"RP",CV=FALSE)
table(a6$group.est,glearn)

## End(Not run)
```

`classif.gkam` *Classification Fitting Functional Generalized Kernel Additive Models*

Description

Computes functional classification using functional explanatory variables using backfitting algorithm.

Usage

```
classif.gkam(
  formula,
  data,
  weights = "equal",
  family = binomial(),
  par.metric = NULL,
  par.np = NULL,
  offset = NULL,
  prob = 0.5,
  type = "1vsall",
  control = NULL,
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The procedure only considers functional covariates (not implemented for non-functional covariates). The details of model specification are given under `Details`.
`data`	List that containing the variables in the model.
`weights`	Weights: • if `character` string `='equal'` same weights for each observation (by default) and `='inverse'` for inverse-probability of weighting. • if `numeric` vector of length `n`, Weight values of each observation.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`par.metric`	List of arguments by covariable to pass to the `metric` function by covariable.
`par.np`	List of arguments to pass to the `fregre.np.cv` function
`offset`	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
`prob`	probability value used for binary discriminant.
`type`	If type is`"1vsall"` (by default) a maximum probability scheme is applied: requires G binary classifiers. If type is `"majority"` (only for multicalss classification G > 2) a voting scheme is applied: requires G (G - 1) / 2 binary classifiers.
`control`	a list of parameters for controlling the fitting process, by default: maxit, epsilon, trace and inverse.
`...`	Further arguments passed to or from other methods.

Details

The first item in the `data` list is called *"df"* and is a data frame with the response, as `glm`. Functional covariates of class `fdata` are introduced in the following items in the `data` list.

Value

Return `gam` object plus:

- `formula`: formula.
- `data`: List that containing the variables in the model.
- `group`: Factor of length *n*.
- `group.est`: Estimated vector groups.
- `prob.classification`: Probability of correct classification by group.
- `prob.group`: Matrix of predicted class probabilities. For each functional point shows the probability of each possible group membership.
- `max.prob`: Highest probability of correct classification.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande M. and Gonzalez-Manteiga W. (2012). *Generalized Additive Models for Functional Data*. TEST. Springer-Velag. doi:10.1007/s1174901203080

McCullagh and Nelder (1989), *Generalized Linear Models* 2nd ed. Chapman and Hall.

Opsomer J.D. and Ruppert D.(1997). *Fitting a bivariate additive model by local polynomial regression*.Annals of Statistics, 25, 186-211.

See Also

See Also as: `fregre.gkam`.
Alternative method: `classif.glm`.

Examples

```
## Not run:
## Time-consuming: selection of 2 levels
data(phoneme)
mlearn<-phoneme[["learn"]][1:150]
glearn<-factor(phoneme[["classlearn"]][1:150])
dataf<-data.frame(glearn)
dat=list("df"=dataf,"x"=mlearn)
a1<-classif.gkam(glearn~x,data=dat)
summary(a1)
mtest<-phoneme[["test"]][1:150]
gtest<-factor(phoneme[["classtest"]][1:150])
newdat<-list("x"=mtest)
p1<-predict(a1,newdat)
table(gtest,p1)

## End(Not run)
```

`classif.glm` *Classification Fitting Functional Generalized Linear Models*

Description

Computes functional classification using functional (and non functional) explanatory variables by basis representation.

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `glm`.

Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list.
`basis.x` is a list of basis for represent each functional covariate. The basis object can be created by

the function: `create.pc.basis`, pca.fd `create.pc.basis`, `create.fdata.basis` o create.basis. `basis.b` is a list of basis for represent each functional beta parameter. If `basis.x` is a list of functional principal components basis (see `create.pc.basis` or pca.fd) the argument `basis.b` is ignored.

Usage

```
classif.glm(
  formula,
  data,
  family = binomial(),
  weights = "equal",
  basis.x = NULL,
  basis.b = NULL,
  type = "1vsall",
  prob = 0.5,
  CV = FALSE,
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`data`	List that containing the variables in the model.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions).
`weights`	Weights: • if `character` string =`'equal'` same weights for each observation (by default) and =`'inverse'` for inverse-probability of weighting. • if `numeric` vector of length `n`, Weight values of each observation.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for functional beta parameter estimation.
`type`	If type is`"1vsall"` (by default) a maximum probability scheme is applied: requires G binary classifiers. If type is `"majority"` (only for multicalss classification G > 2) a voting scheme is applied: requires G (G - 1) / 2 binary classifiers.
`prob`	probability value used for binari discriminant.
`CV`	=TRUE, Cross-validation (CV) is done.
`...`	Further arguments passed to or from other methods.

Value

Return `glm` object plus:

- `formula`: formula.
- `data`: List that containing the variables in the model.
- `group`: Factor of length *n*.
- `group.est`: Estimated vector groups.
- `prob.classification`: Probability of correct classification by group.
- `prob.group`: Matrix of predicted class probabilities. For each functional point shows the probability of each possible group membership.
- `max.prob`: Highest probability of correct classification.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard `glm` procedure.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

McCullagh and Nelder (1989), *Generalized Linear Models* 2nd ed. Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*, New York: Springer. Regression for R. R News 1(2):20-25

See Also

See Also as: `fregre.glm`.
`classif.gsam` and `classif.gkam`.

Examples

```
## Not run:
data(phoneme)
ldat <- ldata("df" = data.frame(y = phoneme[["classlearn"]]),
             "x" = phoneme[["learn"]])
a1 <- classif.glm(y ~ x, data = ldat)
summary(a1)
newldat <- ldata("df" = data.frame(y = phoneme[["classtest"]]),
                "x" = phoneme[["test"]])
p1 <- predict(a1,newldat)
table(newldat$df$y,p1)
sum(p1==newldat$df$y)/250

## End(Not run)
```

classif.gsam *Classification Fitting Functional Generalized Additive Models*

Description

Computes functional classification using functional (and non functional) explanatory variables by basis representation.

Usage

```
classif.gsam(
  formula,
  data,
  family = binomial(),
  weights = "equal",
  basis.x = NULL,
  CV = FALSE,
  prob = 0.5,
  type = "1vsall",
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`data`	List that containing the variables in the model.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`weights`	Weights: • if `character` string `='equal'` same weights for each observation (by default) and `='inverse'` for inverse-probability of weighting. • if `numeric` vector of length `n`, Weight values of each observation.
`basis.x`	List of basis for functional explanatory data estimation.
`CV`	=TRUE, Cross-validation (CV) is done.
`prob`	probability value used for binari discriminant.
`type`	If type is`"1vsall"` (by default) a maximum probability scheme is applied: requires G binary classifiers. If type is `"majority"` (only for multicalss classification G > 2) a voting scheme is applied: requires G (G - 1) / 2 binary classifiers.
`...`	Further arguments passed to or from other methods.

Details

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `glm`.

Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list. `basis.x` is a list of basis for represent each functional covariate. The basis object can be created by the function: `create.pc.basis`, pca.fd `create.pc.basis`, `create.fdata.basis` o create.basis.

Value

Return `gam` object plus:

- `formula`: formula.
- `data`: List that containing the variables in the model.
- `group`: Factor of length n.
- `group.est`: Estimated vector groups
- `prob.classification`: Probability of correct classification by group.
- `prob.group`: Matrix of predicted class probabilities. For each functional point shows the probability of each possible group membership.
- `max.prob`: Highest probability of correct classification.
- `type`: Type of classification scheme: 1 vs all or majority voting.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard `glm` procedure.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

McCullagh and Nelder (1989), *Generalized Linear Models* 2nd ed. Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*, New York: Springer. Regression for R. R News 1(2):20-25

See Also

See Also as: `fregre.gsam`.
Alternative method: `classif.np`, `classif.glm` and `classif.gkam`.

Examples

```
## Not run:
data(phoneme)
ldat <- ldata("df" = data.frame(y = phoneme[["classlearn"]]),
              "x" = phoneme[["learn"]])
              classifKgroups <- fda.usc:::classifKgroups
a1 <- classif.gsam( y ~ s(x,k=3),data=ldat)
summary(a1)
newldat <- ldata("df" = data.frame(y = phoneme[["classtest"]]),
              "x" = phoneme[["test"]])
p1 <- predict(a1,newldat)
table(newldat$df$y,p1)
sum(p1==newldat$df$y)/250

## End(Not run)
```

`classif.gsam.vs` *Variable Selection in Functional Data Classification*

Description

Computes classification by selecting the functional (and non functional) explanatory variables.

Usage

```
classif.gsam.vs(
  data = list(),
  y,
  x,
  family = binomial(),
  weights = "equal",
  basis.x = NULL,
  basis.b = NULL,
  type = "1vsall",
  prob = 0.5,
  alpha = 0.05,
  dcor.min = 0.01,
  smooth = TRUE,
  measure = "accuracy",
  xydist,
  ...
)
```

Arguments

`data`	List that containing the variables in the model. "df" element is a `data.frame` with the response and scalar covariates (numeric and factors variables are allowed). Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list.

`y`	`caracter` string with the name of the scalar response variable
`x`	`caracter` string vector with the name of the scalar and functional potential covariates.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`weights`	Weights: • if `character` string `='equal'` same weights for each observation (by default) and `='inverse'` for inverse-probability of weighting. • if `numeric` vector of length `n`, Weight values of each observation.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for functional beta parameter estimation.
`type`	`character`, type of scheme classification. `'1vsall'` (by default) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. Other posibility for K-way multiclass problem is the `'majority'` voting scheme (also called one vs one). The procedure trains the $K(K-1)/2$ binary classifiers and predicts the final class label as the class label that has been predicted most frequently.
`prob`	probability value used for binary discriminant.
`alpha`	alpha value to test the null hypothesis for the test of independence among covariate X and residual e. By default is `0.05`.
`dcor.min`	lower threshold for the variable X to be considered. X is discarded if the distance correlation $R(X,e) < dcor.min$ (e is the residual).
`smooth`	if `TRUE`, a smooth estimate is made for all covariates included in the model (less for factors). The model is adjusted with the estimated variable linearly or smoothly. If the models are equivalent, the model is adjusted with the linearly estimated variable.
`measure`	measure related with correct classification (by default accuracy).
`xydist`	list with the matrices of distances of each variable (all potential covariates and the response) with itself.
`...`	Further arguments passed to or from other methods.

Value

Return the final fitted model (same result of the classsification method) plus:

- `dcor`, `matrix` with the values of distance correlation for each pontential covariate (by column) and the residual of the model in each step (by row).
- `i.predictor`, `vector` with 1 if the variable is selected, 0 otherwise.
- `ipredictor`, `vector` with the name of selected variables (in order of selection)

Note

Adapted version from the original method in repression: `fregre.gsam.vs`.

Author(s)

Febrero-Bande, M. and Oviedo de la Fuente, M.

References

Febrero-Bande, M., Gonz\'alez-Manteiga, W. and Oviedo de la Fuente, M. Variable selection in functional additive regression models, (2018). Computational Statistics, 1-19. DOI: doi:10.1007/s0018001808445

See Also

See Also as: `classif.gsam`.

Examples

```
## Not run:
data(tecator)
x <- tecator$absorp.fdata
x1 <- fdata.deriv(x)
x2 <- fdata.deriv(x,nderiv=2)
y <- factor(ifelse(tecator$y$Fat<12,0,1))
xcat0 <- cut(rnorm(length(y)),4)
xcat1 <- cut(tecator$y$Protein,4)
xcat2 <- cut(tecator$y$Water,4)
ind <- 1:129
dat    <- data.frame("Fat"=y, x1$data, xcat1, xcat2)
ldat <- ldata("df"=dat[ind,],"x"=x[ind,],"x1"=x1[ind,],"x2"=x2[ind,])
# 3 functionals (x,x1,x2), 3 factors (xcat0, xcat1, xcat2)
# and 100 scalars (impact poitns of x1)

res.gam <- classif.gsam(Fat~s(x),data=ldat)
summary(res.gam)

# Time consuming
res.gam.vs <- classif.gsam.vs("Fat",data=ldat)
summary(res.gam.vs)
res.gam.vs$i.predictor
res.gam.vs$ipredictor

# Prediction
newldat <- ldata("df"=dat[-ind,],"x"=x[-ind,],
                "x1"=x1[-ind,],"x2"=x2[-ind,])
pred.gam <- predict(res.gam,newldat)
pred.gam.vs <- predict(res.gam.vs,newldat)
cat2meas(newldat$df$Fat, pred.gam)
cat2meas(newldat$df$Fat, pred.gam.vs)

## End(Not run)
```

classif.kfold	*Functional Classification usign k-fold CV*

Description

Computes Functional Classification using k-fold cross-validation

Usage

```
classif.kfold(
  formula,
  data,
  classif = "classif.glm",
  par.classif,
  kfold = 10,
  param.kfold = NULL,
  measure = "accuracy",
  cost,
  models = FALSE,
  verbose = FALSE
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The procedure only considers functional covariates (not implemented for non-functional covariates).
`data`	`list`, it contains the variables in the model.
`classif`	character, name of classification method to be used in fitting the model, see `Details` section.
`par.classif`	`list` of arguments used in the classification method.
`kfold`	`integer`, number of k-fold.
`param.kfold`	`list`, arguments related to number of k-folds for each covariate, see `Details` section.
`measure`	`character`, type of measure of accuracy used, see `cat2meas` function.
`cost`	`numeric`, see `cat2meas` function.
`models`	`logical`. If `TRUE`, return a list of the fitted models used, (k-fold -1) X (number of parameters)
`verbose`	`logical`. If `TRUE`, print some internal results.

Details

Parameters for k-fold cross validation:

1. Number of basis elements:
   - Data-driven basis such as Functional Principal Componetns (PC). No implemented for PLS basis yet.
   - Fixed basis (bspline, fourier, etc.).

   Option used in some classifiers such as `classif.glm`, `classif.gsam`, `classif.svm`, etc.
2. Bandwidth parameter. Option used in non-parametric classificiation models such as `classif.np` and `classif.gkam`.

Value

Best fitted model computed by the k-fold CV using the method indicated in the `classif` argument and also returns:

1. `param.min`, value of parameter (or parameters) selected by k-fold CV.
2. `params.error`, k-fold CV error for each parameter combination.
3. `pred.kfold`, predicted response computed by k-fold CV.
4. `model`, if `TRUE`, list of models for each parameter combination.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

Examples

```
## Not run:
data(tecator)
cutpoint <- 18
tecator$y$class <- factor(ifelse(tecator$y$Fat<cutpoint,0,1))
table(tecator$y$class )
x <- tecator[[1]]
x2 <- fdata.deriv(tecator[[1]],2)
data  <-   list("df"=tecator$y,x=x,x2=x2)
formula  <-   formula(class~x+x2)

# ex: default excution of classifier (no k-fold CV)
classif="classif.glm"
out.default <- classif.kfold(formula, data, classif = classif)
out.default
out.default$param.min
out.default$params.error
summary(out.default)

# ex: Number of PC basis elements selected by 10-fold CV
# Logistic classifier
kfold = 10
param.kfold   <-   list("x"=list("pc"=c(1:8)),"x2"=list("pc"=c(1:8)))
```

```
out.kfold1   <-   classif.kfold(formula, data, classif = classif,
                            kfold = kfold,param.kfold = param.kfold)
out.kfold1$param.min
min(out.kfold1$params.error)
summary(out.kfold1)

# ex: Number of PC basis elements selected by 10-fold CV
# Logistic classifier with inverse weighting
out.kfold2 <- classif.kfold(formula, data, classif = classif,
                            par.classif=list("weights"="inverse"),
                            kfold = kfold,param.kfold = param.kfold)
out.kfold2$param.min
min(out.kfold2$params.error)
summary(out.kfold2)

# ex: Number of fourier  basis elements selected by 10-fold CV
# Logistic classifier
ibase = seq(5,15,by=2)
param.kfold <- list("x"=list("fourier"=ibase),
                    "x2"=list("fourier"=ibase))
out.kfold3 <- classif.kfold(formula, data, classif = classif,
                            kfold = kfold,param.kfold = param.kfold)
out.kfold3$param.min
min(out.kfold3$params.error)
summary(out.kfold3)

# ex: Number of k-nearest neighbors selected by 10-fold CV
# non-parametric classifier  (only for a functional covariate)

output <- classif.kfold( class ~ x, data, classif = "classif.knn",
                       param.kfold= list("x"=list("knn"=c(3,5,9,13))))
output$param.min
output$params.error

output <- classif.kfold( class ~ x2, data, classif = "classif.knn",
                       param.kfold= list("x2"=list("knn"=c(3,5,9,13))))
output$param.min
output$params.error

## End(Not run)
```

`classif.ML` *Functional classification using ML algotithms*

Description

Computes functional classification using functional (and non functional) explanatory variables by rpart, nnet, svm or random forest model

Usage

```
classif.nnet(formula, data, basis.x = NULL, weights = "equal", size, ...)

classif.rpart(
  formula,
  data,
  basis.x = NULL,
  weights = "equal",
  type = "1vsall",
  ...
)

classif.svm(
  formula,
  data,
  basis.x = NULL,
  weights = "equal",
  type = "1vsall",
  ...
)

classif.ksvm(formula, data, basis.x = NULL, weights = "equal", ...)

classif.randomForest(
  formula,
  data,
  basis.x = NULL,
  weights = "equal",
  type = "1vsall",
  ...
)

classif.lda(
  formula,
  data,
  basis.x = NULL,
  weights = "equal",
  type = "1vsall",
  ...
)

classif.qda(
  formula,
  data,
  basis.x = NULL,
  weights = "equal",
  type = "1vsall",
  ...
)

classif.naiveBayes(formula, data, basis.x = NULL, laplace = 0, ...)

classif.cv.glmnet(formula, data, basis.x = NULL, weights = "equal", ...)

classif.gbm(formula, data, basis.x = NULL, weights = "equal", ...)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`data`	List that containing the variables in the model.
`basis.x`	List of basis for functional explanatory data estimation.
`weights`	Weights: • if `character` string ='`equal`' same weights for each observation (by default) and ='`inverse`' for inverse-probability of weighting. • if `numeric` vector of length `n`, Weight values of each observation.
`size`	number of units in the hidden layer. Can be zero if there are skip-layer units.
`...`	Further arguments passed to or from other methods.
`type`	If type is"`1vsall`" (by default) a maximum probability scheme is applied: requires G binary classifiers. If type is "`majority`" (only for multicalss classification G > 2) a voting scheme is applied: requires G (G - 1) / 2 binary classifiers.
`laplace`	value used for Laplace smoothing (additive smoothing). Defaults to 0 (no Laplace smoothing).

Details

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `glm`.

Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list.
`basis.x` is a list of basis for represent each functional covariate. The b object can be created by the function: `create.pc.basis`, pca.fd `create.pc.basis`, `create.fdata.basis` o create.basis.
`basis.b` is a list of basis for represent each functional beta parameter. If `basis.x` is a list of functional principal components basis (see `create.pc.basis` or pca.fd) the argument `basis.b` is ignored.

Value

Return `classif` object plus:

- `formula`: formula.
- `data`: List that containing the variables in the model.
- `group`: Factor of length *n*.

- `group.est`: Estimated vector groups.
- `prob.classification`: Probability of correct classification by group.
- `prob.group`: Matrix of predicted class probabilities. For each functional point shows the probability of each possible group membership.
- `max.prob`: Highest probability of correct classification.
- `type`: Type of classification scheme: 1 vs all or majority voting.
- `fit`: list of binary classification fitted models.

Note

Wrapper versions for multivariate and functional classification:

- `classif.lda`,`classif.qda`: uses `lda` and `qda` functions and requires `MASS` package.
- `classif.nnet`: uses `nnet` function and requires `nnet` package.
- `classif.rpart`: uses `nnet` function and requires `rpart` package.
- `classif.svm`, `classif.naiveBayes`: uses `svm` and `naiveBayes` functions and requires `e1071` package.
- `classif.ksvm`: uses `weighted.ksvm` function and requires `personalized` package.
- `classif.randomForest`: uses `randomForest` function and requires `randomForest` package.
- `classif.cv.glmnet`: uses `cv.glmnet` function and requires `glmnet` package.
- `classif.gbm`: uses `gbm` function and requires `gbm` package.

Author(s)

Febrero-Bande, M. and Oviedo de la Fuente, M.

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

McCullagh and Nelder (1989), *Generalized Linear Models* 2nd ed. Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*, New York: Springer. Regression for R. R News 1(2):20-25

See Also

See Also as: rpart.
Alternative method: `classif.np`, `classif.glm`, `classif.gsam` and `classif.gkam`.

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme[["learn"]]
glearn<-phoneme[["classlearn"]]
mtest<-phoneme[["test"]]
gtest<-phoneme[["classtest"]]
dataf<-data.frame(glearn)
dat=ldata("df"=dataf,"x"=mlearn)
a1<-classif.rpart(glearn~x,data=dat)
a2<-classif.nnet(glearn~x,data=dat)
a3<-classif.gbm(glearn~x,data=dat)
a4<-classif.randomForest(glearn~x,data=dat)
a5<-classif.cv.glmnet(glearn~x,data=dat)
newdat<-list("x"=mtest)
p1<-predict(a1,newdat,type="class")
p2<-predict(a2,newdat,type="class")
p3<-predict(a3,newdat,type="class")
p4<-predict(a4,newdat,type="class")
p5<-predict(a5,newdat,type="class")
mean(p1==gtest);mean(p2==gtest);mean(p3==gtest)
mean(p4==gtest);mean(p5==gtest)

## End(Not run)
```

---

classif.np *Kernel Classifier from Functional Data*

Description

Fits Nonparametric Supervised Classification for Functional Data.

Usage

```
classif.np(
  group,
  fdataobj,
  h = NULL,
  Ker = AKer.norm,
  metric,
  weights = "equal",
  type.S = S.NW,
  par.S = list(),
  ...
)

classif.knn(
  group,
  fdataobj,
  knn = NULL,
  metric,
  weights = "equal",
  par.S = list(),
  ...
)

classif.kernel(
  group,
  fdataobj,
  h = NULL,
  Ker = AKer.norm,
  metric,
  weights = "equal",
  par.S = list(),
  ...
)
```

Arguments

`group`	Factor of length *n*
`fdataobj`	`fdata` class object.
`h`	Vector of smoothing parameter or bandwidth.
`Ker`	Type of kernel used.
`metric`	Metric function, by default `metric.lp`.
`weights`	weights.
`type.S`	Type of smothing matrix `S`. By default `S` is calculated by Nadaraya-Watson kernel estimator (`S.NW`).
`par.S`	List of parameters for `type.S`: `w`, the weights.
`...`	Arguments to be passed for `metric.lp` o other metric function and `Kernel` function.
`knn`	Vector of number of nearest neighbors considered.

Details

Make the group classification of a training dataset using kernel or KNN estimation: `Kernel`.

Different types of metric funtions can be used.

Value

- `fdataobj`: `fdata` class object.
- `group`: Factor of length `n`.
- `group.est`: Estimated vector groups.

- prob.group: Matrix of predicted class probabilities. For each functional point shows the probability of each possible group membership.
- max.prob: Highest probability of correct classification.
- h.opt: Optimal smoothing parameter or bandwidht estimated.
- D: Matrix of distances of the optimal quantile distance hh.opt.
- prob.classification: Probability of correct classification by group.
- misclassification: Vector of probability of misclassification by number of neighbors knn.
- h: Vector of smoothing parameter or bandwidht.
- C: A call of function classif.kernel.

Note

If fdataobj is a data.frame the function considers the case of multivariate covariates.
metric.dist function is used to compute the distances between the rows of a data matrix (as dist function.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Ferraty, F. and Vieu, P. (2006). *NPFDA in practice*. Free access on line at https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/

See Also

See Also as predict.classif

Examples

```
## Not run:
data(phoneme)
mlearn <- phoneme[["learn"]]
glearn <- phoneme[["classlearn"]]
h <- 9:19
out <- classif.np(glearn,mlearn,h=h)
summary(out)
head(round(out$prob.group,4))

## End(Not run)
```

cond.F *Conditional Distribution Function*

Description

Calculate the conditional distribution function of a scalar response with functional data.

Usage

```
cond.F(
  fdata0,
  y0,
  fdataobj,
  y,
  h = 0.15,
  g = 0.15,
  metric = metric.lp,
  Ker = list(AKer = AKer.epa, IKer = IKer.epa),
  ...
)
```

Arguments

`fdata0`	Conditional explanatory functional data of `fdata` class.
`y0`	Vector of conditional response with length `n`.
`fdataobj`	`fdata` class object.
`y`	Vector of scalar response with length `nn`.
`h`	Smoothing parameter or bandwidth of response `y`.
`g`	Smoothing parameter or bandwidth of explanatory functional data `fdataobj`.
`metric`	Metric function, by default `metric.lp`.
`Ker`	List of 2 arguments. The fist argument is a character string that determines the type of asymetric kernel (see `Kernel.asymmetric`). Asymmetric Epanechnikov kernel is selected by default. The second argumentis a string that determines the type of integrated kernel(see `Kernel.integrate`). Integrate Epanechnikov kernel is selected by default. .
`...`	Further arguments passed to or from other methods.

Details

If `x.dist=NULL` the distance matrix between `fdata` objects is calculated by function passed in `metric` argument.

Value

- `Fc`: Conditional distribution function.
- `y0`: Vector of conditional response.
- `g`: Smoothing parameter or bandwidth of explanatory functional data (`fdataobj`).
- `h`: Smoothing parameter or bandwidth of respone, `y`.
- `x.dist`: Distance matrix between curves of `fdataobj` object.
- `xy.dist`: Distance matrix between cuves of `fdataobj` and `fdata0` objects.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

See Also

See Also as: `cond.mode` and `cond.quantile`.

Examples

```
## Not run:
# Read data
n= 500
t= seq(0,1,len=101)
beta = t*sin(2*pi*t)^2
x = matrix(NA, ncol=101, nrow=n)
y=numeric(n)
x0<-rproc2fdata(n,seq(0,1,len=101),sigma="wiener")
x1<-rproc2fdata(n,seq(0,1,len=101),sigma=0.1)
x<-x0*3+x1
fbeta = fdata(beta,t)
y<-inprod.fdata(x,fbeta)+rnorm(n,sd=0.1)
prx=x[1:100];pry=y[1:100]
ind=101;ind2=102:110
pr0=x[ind];pr10=x[ind2,]
ndist=61
gridy=seq(-1.598069,1.598069, len=ndist)

# Conditional Function
res1 = cond.F(pr10, gridy, prx, pry,p=1)
res2 = cond.F(pr10, gridy, prx, pry,h=0.3)
res3 = cond.F(pr10, gridy, prx, pry,g=0.25,h=0.3)

plot(res1$Fc[,1],type="l",ylim=c(0,1))
lines(res2$Fc[,1],type="l",col=2)
lines(res3$Fc[,1],type="l",col=3)
```

```
## End(Not run)
```

cond.mode *Conditional mode*

Description

Computes the mode for conditional distribution function.

Usage

```
cond.mode(Fc, method = "monoH.FC", draw = TRUE)
```

Arguments

`Fc`	Object estimated by `cond.F` function.
`method`	Specifies the type of spline to be used. Possible values are *"diff"*, *"fmm"*, *"natural"*, *"periodic"* and *"monoH.FC"*.
`draw`	=TRUE, plots the conditional distribution and density function.

Details

The conditional mode is calculated as the maximum argument of the derivative of the conditional distribution function (density function `f`).

Value

Return the mode for conditional distribution function.

- `mode.cond`: Conditional mode.
- `x`: A grid of length `n` where the conditional density function is evaluated.
- `f`: The conditional density function evaluated at `x`.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

See Also

See Also as: `cond.F`, `cond.quantile` and splinefun .

Examples

```
## Not run:
n= 500
t= seq(0,1,len=101)
beta = t*sin(2*pi*t)^2
x = matrix(NA, ncol=101, nrow=n)
y=numeric(n)
x0<-rproc2fdata(n,seq(0,1,len=101),sigma="wiener")
x1<-rproc2fdata(n,seq(0,1,len=101),sigma=0.1)
x<-x0*3+x1
fbeta = fdata(beta,t)
y<-inprod.fdata(x,fbeta)+rnorm(n,sd=0.1)
prx=x[1:100];pry=y[1:100]
ind=101;ind2=101:110
pr0=x[ind];pr10=x[ind2]
ndist=161
gridy=seq(-1.598069,1.598069, len=ndist)
# Conditional Function
I=5
# Time consuming
res = cond.F(pr10[I], gridy, prx, pry, h=1)
mcond=cond.mode(res)
mcond2=cond.mode(res,method="diff")

## End(Not run)
```

cond.quantile *Conditional quantile*

Description

Computes the quantile for conditional distribution function.

Usage

```
cond.quantile(
  qua = 0.5,
  fdata0,
  fdataobj,
  y,
  fn,
  a = min(y),
  b = max(y),
  tol = 10^floor(log10(max(y) - min(y)) - 3),
  iter.max = 100,
  ...
)
```

Arguments

qua	Quantile value, by default the median (qua=0.5).
fdata0	Conditional functional explanatory data of `fdata` class object.
fdataobj	Functional explanatory data of `fdata` class object.
y	Scalar Response.
fn	Conditional distribution function.
a	Lower limit.
b	Upper limit.
tol	Tolerance.
iter.max	Maximum iterations allowed, by default `100`.
...	Further arguments passed to or from other methods.

Value

Return the quantile for conditional distribution function.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

See Also

See Also as: `cond.F` and `cond.mode`.

Examples

```
## Not run:
n= 100
t= seq(0,1,len=101)
beta = t*sin(2*pi*t)^2
x = matrix(NA, ncol=101, nrow=n)
y=numeric(n)
x0<-rproc2fdata(n,seq(0,1,len=101),sigma="wiener")
x1<-rproc2fdata(n,seq(0,1,len=101),sigma=0.1)
x<-x0*3+x1
fbeta = fdata(beta,t)
y<-inprod.fdata(x,fbeta)+rnorm(n,sd=0.1)

prx=x[1:50];pry=y[1:50]
ind=50+1;ind2=51:60
pr0=x[ind];pr10=x[ind2]
ndist=161
gridy=seq(-1.598069,1.598069, len=ndist)
```

```
ind4=5
y0 = gridy[ind4]

# Conditional median
med=cond.quantile(qua=0.5,fdata0=pr0,fdataobj=prx,y=pry,fn=cond.F,h=1)

# Conditional CI 95% conditional
lo=cond.quantile(qua=0.025,fdata0=pr0,fdataobj=prx,y=pry,fn=cond.F,h=1)
up=cond.quantile(qua=0.975,fdata0=pr0,fdataobj=prx,y=pry,fn=cond.F,h=1)
print(c(lo,med,up))

## End(Not run)
```

`create.fdata.basis` *Create Basis Set for Functional Data of fdata class*

Description

Compute basis for functional data.

Usage

```
create.fdata.basis(
  fdataobj,
  l = 1:5,
  maxl = max(l),
  type.basis = "bspline",
  rangeval = fdataobj$rangeval,
  class.out = "fd"
)

create.pc.basis(
  fdataobj,
  l = 1:5,
  norm = TRUE,
  basis = NULL,
  lambda = 0,
  P = c(0, 0, 1),
  ...
)

create.pls.basis(
  fdataobj,
  y,
  l = 1:5,
  norm = TRUE,
  lambda = 0,
  P = c(0, 0, 1),
  ...
)

create.raw.fdata(fdataobj, l = 1:nrow(fdataobj))
```

Arguments

`fdataobj`	`fdata` class object.
`l`	Vector of basis index.
`maxl`	maximum number of basis
`type.basis`	Type of basis (see create.basis function).
`rangeval`	A vector of length 2 giving the lower and upper limits of the range of permissible values for the function argument.
`class.out`	=="fd" basisfd class, =="fdata" fdata class.
`norm`	If `TRUE` the norm of eigenvectors `basis` is 1.
`basis`	"fd" basis object.
`lambda`	Amount of penalization. Default value is 0, i.e. no penalization is used.
`P`	If P is a vector: coefficients to define the penalty matrix object. By default P=c(0,0,1) penalize the second derivative (curvature) or acceleration. If P is a matrix: the penalty matrix object.
`...`	Further arguments passed to or from other methods.
`y`	Vector of response (scalar).

Value

- `basis`: basis object.
- `x`: if `TRUE` the value of the rotated data (the centred data multiplied by the basis matrix) is returned.
- `mean`: functional mean of `fdataobj`.
- `df`: degree of freedom.
- `type`: type of basis.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O. and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

N. Kraemer, A.-L. Boulsteix, and G. Tutz (2008). Penalized Partial Least Squares with Applications to B-Spline Transformations and Functional Data. Chemometrics and Intelligent Laboratory Systems, 94, 60 - 69. doi:10.1016/j.chemolab.2008.06.009

See Also

See Also as create.basis and fdata2pc.

Examples

```
## Not run:
data(tecator)
basis.pc<-create.pc.basis(tecator$absorp.fdata,c(1,4,5))
plot(basis.pc$basis,col=1)
summary(basis.pc)
basis.pls<-create.pls.basis(tecator$absorp.fdata,y=tecator$y[,1],c(1,4,5))
summary(basis.pls)
plot(basis.pls$basis,col=2)
summary(basis.pls)

basis.fd<-create.fdata.basis(tecator$absorp.fdata,c(1,4,5),
type.basis="fourier")
plot(basis.pc$basis)
basis.fdata<-create.fdata.basis(tecator$absorp.fdata,c(1,4,5),
type.basis="fourier",class.out="fdata")
plot(basis.fd,col=2,lty=1)
lines(basis.fdata,col=3,lty=1)

## End(Not run)
```

CV.S *The cross-validation (CV) score*

Description

Compute the leave-one-out cross-validation score.

Usage

```
CV.S(y, S, W = NULL, trim = 0, draw = FALSE, metric = metric.lp, ...)
```

Arguments

`y`	Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
`S`	Smoothing matrix, see `S.NW`, `S.LLR` or $S.KNN$.
`W`	Matrix of weights.
`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`metric`	Metric function, by default `metric.lp`.
`...`	Further arguments passed to or from other methods.

Details

A.-If `trim=0`:

$$CV(h) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - r_i(x_i)}{(1 - S_{ii})} \right)^2 w(x_i)$$

S_{ii} is the ith diagonal element of the smoothing matrix S.

B.-If `trim>0`:

$$CV(h) = \frac{1}{l} \sum_{i=1}^{l} \left(\frac{y_i - r_i(x_i)}{(1 - S_{ii})} \right)^2 w(x_i)$$

S_{ii} is the ith diagonal element of the smoothing matrix S and l the index of `(1-trim)` curves with less error.

Value

Returns CV score calculated for input parameters.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

See Also

See Also as `optim.np`
Alternative method: `GCV.S`

Examples

```
## Not run:
data(tecator)
x<-tecator$absorp.fdata
np<-ncol(x)
tt<-1:np
S1 <- S.NW(tt,3,Ker.epa)
S2 <- S.LLR(tt,3,Ker.epa)
S3 <- S.NW(tt,5,Ker.epa)
S4 <- S.LLR(tt,5,Ker.epa)
cv1 <- CV.S(x, S1)
cv2 <- CV.S(x, S2)
cv3 <- CV.S(x, S3)
cv4 <- CV.S(x, S4)
cv5 <- CV.S(x, S4,trim=0.1,draw=TRUE)
cv1;cv2;cv3;cv4;cv5
S6 <- S.KNN(tt,1,Ker.unif,cv=TRUE)
```

```
S7 <- S.KNN(tt,5,Ker.unif,cv=TRUE)
cv6 <- CV.S(x, S6)
cv7 <- CV.S(x, S7)
cv6;cv7

## End(Not run)
```

`dcor.xy` *Distance Correlation Statistic and t-Test*

Description

Distance correlation t-test of multivariate and functional independence (wrapper functions of energy package).

Usage

```
dcor.xy(x, y, test = TRUE, metric.x, metric.y, par.metric.x, par.metric.y, n)

dcor.dist(D1, D2)

bcdcor.dist(D1, D2, n)

dcor.test(D1, D2, n)
```

Arguments

`x`	data (fdata, matrix or data.frame class) of first sample.
`y`	data (fdata, matrix or data.frame class) of second sample.
`test`	if TRUE, compute bias corrected distance correlation statistic and the corresponding t-test, else compute distance correlation statistic.
`metric.x, metric.y`	Name of metric or semi-metric function used for compute the distances of `x` and `y` object respectively. By default, `metric.lp` for functional data and `metric.dist` for multivariate data.
`par.metric.x, par.metric.y`	List of parameters for the corresponding metric function.
`n`	The sample size used in bias corrected version of distance correlation, by default is the number of rows of `x`.
`D1`	Distances of first sample (x data).
`D2`	Distances of second sample (y data).

Details

These wrapper functions extend the functions of the `energy` package for multivariate data to functional data. Distance correlation is a measure of dependence between random vectors introduced by Szekely, Rizzo, and Bakirov (2007). `dcor.xy` performs a nonparametric t-test of multivariate or functional independence in high dimension. The distribution of the test statistic is approximately Student t with $n(n-3)/2-1$ degrees of freedom and for $n \geq 10$ the statistic is approximately distributed as standard normal. Wrapper function of `energy:::dcor.ttest`. The t statistic is a transformation of a bias corrected version of distance correlation (see SR 2013 for details). Large values (upper tail) of the t statistic are significant.
`dcor.test` similar to `dcor.xy` but only for distance matrix. `dcor.dist` compute distance correlation statistic. Wrapper function of `energy::dcor` but only for distance matrix `bcdcor.dist` compute bias corrected distance correlation statistic. Wrapper function of `energy:::bcdcor` but only for distance matrix.

Value

`dcor.test` returns a list with class `htest` containing

- `method`: description of test.
- `statistic`: observed value of the test statistic.
- `parameter`: degrees of freedom.
- `estimate`: bias corrected distance correlation `bcdcor(x,y)`.
- `p.value`: p-value of the t-test.
- `data.name`: description of data.

`dcor.xy` returns the previous list with class `htest` and

- `D1`: the distance matrix of `x`.
- `D2`: the distance matrix of `y`.

`dcor.dist` returns the distance correlation statistic.

`bcdcor.dist` returns the bias corrected distance correlation statistic.

Author(s)

Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`> and Manuel Febrero Bande

References

Szekely, G.J. and Rizzo, M.L. (2013). The distance correlation t-test of independence in high dimension. *Journal of Multivariate Analysis*, Volume 117, pp. 193-213.

Szekely, G.J., Rizzo, M.L., and Bakirov, N.K. (2007), Measuring and Testing Dependence by Correlation of Distances, *Annals of Statistics*, Vol. 35 No. 6, pp. 2769-2794.

See Also

`metric.lp` amd `metric.dist`.

Examples

```
## Not run:
x<-rproc2fdata(100,1:50)
y<-rproc2fdata(100,1:50)
dcor.xy(x, y,test=TRUE)
dx <- metric.lp(x)
dy <- metric.lp(y)
dcor.test(dx, dy)
bcdcor.dist(dx, dy)
dcor.xy(x, y,test=FALSE)
dcor.dist(dx, dy)

## End(Not run)
```

`depth.fdata` *Computation of depth measures for functional data*

Description

Several depth measures can be computed for functional data for descriptive or classification purposes.

Usage

```
depth.mode(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  metric = metric.lp,
  h = NULL,
  scale = FALSE,
  draw = FALSE,
  ...
)

depth.RP(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  nproj = 50,
  proj = "vexponential",
  dfunc = "TD1",
  par.dfunc = list(),
  scale = FALSE,
  draw = FALSE,
  ...
```

```
)

depth.RPD(
  fdataobj,
  fdataori = fdataobj,
  nproj = 20,
  proj = 1,
  deriv = c(0, 1),
  trim = 0.25,
  dfunc2 = mdepth.LD,
  method = "fmm",
  draw = FALSE,
  ...
)

depth.RT(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  nproj = 10,
  proj = 1,
  xeps = 1e-07,
  draw = FALSE,
  ...
)

depth.KFSD(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  h = NULL,
  scale = FALSE,
  draw = FALSE
)

depth.FSD(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  scale = FALSE,
  draw = FALSE
)

depth.FM(
  fdataobj,
  fdataori = fdataobj,
  trim = 0.25,
  scale = FALSE,
  dfunc = "FM1",
  par.dfunc = list(scale = TRUE),
  draw = FALSE
)
```

Arguments

fdataobj	The set of new curves to evaluate the depth. fdata class object.
fdataori	The set of reference curves respect to which the depth is computed. fdata class object.
trim	The alpha of the trimming.
metric	Metric function, by default metric.lp. Distance matrix between fdataobj and fdataori.
h	Bandwidth, h>0. Default argument values are provided as the 15%–quantile of the distance between x and xx.
scale	=TRUE, the depth is scaled respect to depths in fdataori.
draw	=TRUE, draw the curves, the sample median and trimmed mean.
...	Further arguments passed to or from other methods. For depth.mode parameters for metric. For random projection depths, parameters to be included in rproc2fdata not included before.
nproj	The number of projections. Ignored if a fdata class object is provided in proj
proj	if a fdata class, projections provided by the user. Otherwise, it is the sigma parameter of rproc2fdata function.
dfunc	type of univariate depth function used inside depth function: "FM1" refers to the original Fraiman and Muniz univariate depth (default), "TD1" Tukey (Halfspace),"Liu1" for simplical depth, "LD1" for Likelihood depth and "MhD1" for Mahalanobis 1D depth. Also, any user function fulfilling the following pattern FUN.USER(x,xx,...) and returning a dep component can be included.f
par.dfunc	List of parameters for *dfunc*.
deriv	Number of derivatives described in integer vector deriv. =0 means no derivative.
dfunc2	Multivariate depth function (second step depth function) in RPD depth, by default mdepth.LD. Any user function with the pattern FUN.USER(x,xx,...) can be employed.
method	Type of derivative method. See fdata.deriv for more details. • numeric: the procedure considers the argument value as the bandwidth. • NULL: (by default) the bandwidth is provided as the 15%–quantile of the distance among curves of fdataori. • character: a string (like "0.15"), the procedure reads the numeric value and consider it as the quantile of the distance in fdataori (as in the second case).
xeps	Accuracy. The left limit of the empirical distribution function.

Details

Type of depth functions: Fraiman and Muniz (FM) depth, modal depth, random Tukey (RT), random projection (RP) depth and double random projection depth (RPD).

- `depth.FM`: computes the integration of an univariate depth along the axis x (see Fraiman and Muniz 2001). It is also known as Integrated Depth.
- `depth.mode`: implements the modal depth (see Cuevas et al 2007).
- `depth.RT`: implements the Random Tukey depth (see Cuesta–Albertos and Nieto–Reyes 2008).
- `depth.RP`: computes the Random Projection depth (see Cuevas et al. 2007).
- `depth.RPD`: implements a depth measure based on random projections possibly using several derivatives (see Cuevas et al. 2007).
- `depth.FSD`: computes the Functional Spatial Depth (see Sguera et al. 2014).
- `depth.KFSD`: implements the Kernelized Functional Spatial Depth (see Sguera et al. 2014).

- `depth.mode`: calculates the depth of a datum accounting the number of curves in its neighbourhood. By default, the distance is calculated using `metric.lp` function although any other distance could be employed through argument `metric` (with the general pattern `USER.DIST(fdataobj,fdataori)`).
- `depth.RP`: summarizes the random projections through averages whereas the `depth.RT` function uses the minimum of all projections.
- `depth.RPD`: involves the original trajectories and the derivatives of each curve in two steps. It builds random projections for the function and their derivatives (indicated in the parameter `deriv`) and then applies a depth function (by default `depth.mode`) to this set of random projections (by default the Tukey one).
- `depth.FSD` and `depth.KFSD`: are the implementations of the default versions of the functional spatial depths proposed in Sguera et al 2014. At this moment, it is not possible to change the kernel in the second one.

Value

Return a list with:

- `median`: Deepest curve.
- `lmed`: Index deepest element `median`.
- `mtrim`: `fdata` class object with the average from the `(1-trim)%` deepest curves.
- `ltrim`: Indexes of curves that conform the trimmed mean `mtrim`.
- `dep`: Depth of each curve of fdataobj w.r.t. fdataori.
- `dep.ori`: Depth of each curve of fdataori w.r.t. fdataori.
- `proj`: The projection value of each point on the curves.
- `dist`: Distance matrix between curves or functional data.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente `<manuel.oviedo@udc.es>`

References

Cuevas, A., Febrero-Bande, M., Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics* 22, 3, 481-496.

Fraiman R, Muniz G. (2001). Trimmed means for functional data. *Test* 10: 419-440.

Cuesta–Albertos, JA, Nieto–Reyes, A. (2008) The Random Tukey Depth. *Computational Statistics and Data Analysis* Vol. 52, Issue 11, 4979-4988.

Febrero-Bande, M, Oviedo de la Fuente, M. (2012). Statistical Computing in Functional Data Analysis: The R Package fda.usc. *Journal of Statistical Software*, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

Sguera C, Galeano P, Lillo R (2014). Spatial depth based classification for functional data. *TEST* 23(4):725–750.

See Also

See Also as `Descriptive`.

Examples

```
## Not run:
#Ex: CanadianWeather data
tt=1:365
fdataobj<-fdata(t(CanadianWeather$dailyAv[,,1]),tt)
# Fraiman-Muniz Depth
out.FM=depth.FM(fdataobj,trim=0.1,draw=TRUE)
#Modal Depth
out.mode=depth.mode(fdataobj,trim=0.1,draw=TRUE)
out.RP=depth.RP(fdataobj,trim=0.1,draw=TRUE)
out.RT=depth.RT(fdataobj,trim=0.1,draw=TRUE)
out.FSD=depth.FSD(fdataobj,trim=0.1,draw=TRUE)
out.KFSD=depth.KFSD(fdataobj,trim=0.1,draw=TRUE)
## Double Random Projections
out.RPD=depth.RPD(fdataobj,deriv=c(0,1),dfunc2=mdepth.LD,
trim=0.1,draw=TRUE)
out<-c(out.FM$mtrim,out.mode$mtrim,out.RP$mtrim,out.RPD$mtrim)
plot(fdataobj,col="grey")
lines(out)
cdep<-cbind(out.FM$dep,out.mode$dep,out.RP$dep,out.RT$dep,out.FSD$dep,out.KFSD$dep)
colnames(cdep)<-c("FM","mode","RP","RT","FSD","KFSD")
pairs(cdep)
round(cor(cdep),2)

## End(Not run)
```

depth.mdata *Provides the depth measure for multivariate data*

Description

Compute measure of centrality of the multivariate data. Type of depth function: simplicial depth (SD), Mahalanobis depth (MhD), Random Half–Space depth (HS), random projection depth (RP) and Likelihood Depth (LD).

Usage

```
mdepth.LD(x, xx = x, metric = metric.dist, h = NULL, scale = FALSE, ...)

mdepth.HS(x, xx = x, proj = 50, scale = FALSE, xeps = 1e-15, random = FALSE)

mdepth.RP(x, xx = x, proj = 50, scale = FALSE)

mdepth.MhD(x, xx = x, scale = FALSE)

mdepth.KFSD(x, xx = x, trim = 0.25, h = NULL, scale = FALSE, draw = FALSE)

mdepth.FSD(x, xx = x, trim = 0.25, scale = FALSE, draw = FALSE)

mdepth.FM(x, xx = x, scale = FALSE, dfunc = "TD1")

mdepth.TD(x, xx = x, xeps = 1e-15, scale = FALSE)

mdepth.SD(x, xx = NULL, scale = FALSE)
```

Arguments

`x`	is a set of points, a d-column matrix.
`xx`	is a d-dimension multivariate reference sample (a d-column matrix) where `x` points are evaluated.
`metric`	Metric function, by default `metric.dist`. Distance matrix between `x` and `xx` is computed.
`h`	Bandwidth, `h>0`. Default argument values are provided as the 15%–quantile of the distance between `x` and `xx`.
`scale`	=TRUE, scale the depth, see scale.
`...`	Further arguments passed to or from other methods.
`proj`	are the directions for random projections, by default 500 random projections generated from a scaled `runif(500,-1,1)`.
`xeps`	Accuracy. The left limit of the empirical distribution function.
`random`	=TRUE for random projections. =FALSE for deterministic projections.

`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`dfunc`	type of univariate depth function used inside depth function: "FM1" refers to the original Fraiman and Muniz univariate depth (default), "TD1" Tukey (Half-space),"Liu1" for simplical depth, "LD1" for Likelihood depth and "MhD1" for Mahalanobis 1D depth. Also, any user function fulfilling the following pattern `FUN.USER(x,xx,...)` and returning a `dep` component can be included.

Details

Type of depth measures:

- The `mdepth.SD` calculates the simplicial depth (HD) of the points in `x` w.r.t. `xx` (for bivariate data).
- The `mdepth.HS` function calculates the random half–space depth (HS) of the points in `x` w.r.t. `xx` based on random projections `proj`.
- The `mdepth.MhD` function calculates the Mahalanobis depth (MhD) of the points in `x` w.r.t. `xx`.
- The `mdepth.RP` calculates the random' projection depth (RP) of the points in `x` w.r.t. `xx` based on random projections `proj`.
- The `mdepth.LD` calculates the Likelihood depth (LD) of the points in `x` w.r.t. `xx`.
- The `mdepth.TD` function provides the Tukey depth measure for multivariate data.

Value

- `lmed`: Index of the deepest element (median) of `xx`.
- `ltrim`: Index of the set of points `x` with trimmed mean `mtrim`.
- `dep`: Depth of each point in `x` with respect to `xx`.
- `proj`: The projection value of each point onto the set of points.
- `x`: A set of points to be evaluated.
- `xx`: A reference sample.
- `name`: Name of the depth method.

Author(s)

`mdepth.RP`, `mdepth.MhD` and `mdepth.HS` are versions created by Manuel Febrero Bande and Manuel Oviedo de la Fuente of the original version created by Jun Li, Juan A. Cuesta Albertos and Regina Y. Liu for polynomial classifier.

References

Liu, R. Y., Parelius, J. M., and Singh, K. (1999). Multivariate analysis by data depth: descriptive statistics, graphics and inference,(with discussion and a rejoinder by Liu and Singh). *The Annals of Statistics*, 27(3), 783-858.

See Also

Functional depth functions: `depth.FM`, `depth.mode`, `depth.RP`, `depth.RPD` and `depth.RT`.

Examples

```
## Not run:
data(iris)
group<-iris[,5]
x<-iris[,1:2]

MhD<-mdepth.MhD(x)
PD<-mdepth.RP(x)
HD<-mdepth.HS(x)
SD<-mdepth.SD(x)

x.setosa<-x[group=="setosa",]
x.versicolor<-x[group=="versicolor",]
x.virginica<-x[group=="virginica",]
d1<-mdepth.SD(x,x.setosa)$dep
d2<-mdepth.SD(x,x.versicolor)$dep
d3<-mdepth.SD(x,x.virginica)$dep

## End(Not run)
```

`depth.mfdata` *Provides the depth measure for a list of p–functional data objects*

Description

This function computes the depth measure for a list of p–functional data objects. The procedure extends the Fraiman and Muniz (FM), modal, and random project depth functions from 1 functional dataset to p functional datasets.

Usage

```
depth.modep(
  mfdata,
  mfdataref = mfdata,
  h = NULL,
  metric,
  par.metric = list(),
  method = "euclidean",
  scale = FALSE,
  trim = 0.25,
  draw = FALSE,
  ask = FALSE
)

depth.RPp(
  mfdata,
  mfdataref = mfdata,
  nproj = 50,
  proj = "vexponential",
  trim = 0.25,
  dfunc = "mdepth.TD",
  par.dfunc = list(scale = TRUE),
  draw = FALSE,
  ask = FALSE
)

depth.FMp(
  mfdata,
  mfdataref = mfdata,
  trim = 0.25,
  dfunc = "mdepth.MhD",
  par.dfunc = list(scale = FALSE),
  draw = FALSE,
  ask = FALSE,
  ...
)
```

Arguments

`mfdata`	A list of new curves (list of fdata ojects) to evaluate the depth.
`mfdataref`	A set of reference curves (list of fdata ojects) w.r.t. the depth of mfdata is computed.
`h`	Bandwidth, `h>0`. Default argument values are provided as the 15%–quantile of the distance between `fdataobj` and `fdataori`.
`metric`	Metric or semi–metric function used for compute the distance between each element in `ldata` w.r.t. `ldataref`, by default `metric.lp`.
`par.metric`	list of parameters for the metric function.
`method`	Type of the distance measure (by default `euclidean`) to compute the metric between the p-distance matrix computed from the p functional data elements.
`scale`	=TRUE, scale the depth.
`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`ask`	logical. If `TRUE` (and the R session is interactive) the user is asked for input, before a new figure is drawn.
`nproj`	The number projection.
`proj`	if is a character: create the random projection using a covariance matrix by process indicated in the argument (by default, proj=1, sigma=diag(ncol(fdataobj))), else if is a matrix of random projection provided by the user.
`dfunc`	Type of multivariate depth (of order p) function used in Framiman and Muniz depth, `depth.FMp` or in Random Projection depth,`depth.FMp`: : • The `mdepth.SD` function provides the simplicial depth measure for bivariate data.

- The mdepth.LD function provides the Likelihood depth measure based on Nadaraya–Watson estimator of empirical density function.
- The mdepth.HS function implements a half-space depth measure based on random projections.
- The mdepth.TD function implements a Tukey depth measure.
- The mdepth.MhDfunction implements a Mahalanobis depth measure.
- The mdepth.RP function provides the depth measure using random projections for multivariate data.

par.dfunc	list of parameters for the dfunc depth function, see Depth.Multivariate.
...	Further arguments passed to or from other methods.

Details

- depth.FMp, this procedure suposes that each curve of the mfdataobj have the same support [0,T] (same argvals and rangeval). The FMp depth is defined as: $FM_i^p = \int_0^T Z_i^p(t)dt$ where $Z_i^p(t)$ is a p–variate depth of the vector $(x_i^1(t), \ldots, x_i^p(t))$ w.r.t. the sample at t. derivatives. In this case,note solo un dato funcional se reduce depth.FM=depth.FM1
- The depth.RPp function calculates the depth in two steps. It builds random projections for the each curve of the mfdata w.r.t. each curve of the mfdataref object. Then it applyes a multivariate depth function specified in dfunc argument to the set of random projections. This procedure is a generalization of Random Projection with derivatives (RPD) implemented in depth.RPD function. Now, the procedure computes a p-variate depth with the projections using the p functional dataset.
- The modal depth depth.modep function calculates the depth in three steps. First, the function calculates a suitable metrics or semi–metrics $m_1 + \cdots + m_p$ for each curve of the mfdata w.r.t. each curve in the mfdataref object using the metric and par.metric arguments, see metric.lp or semimetric.NPFDA for more details. Second, the function uses the p–dimensional metrics to construct a new metric, specified in method argument, by default if method="euclidean", i.e. $m := \sqrt{m_1^2 + \cdots + m_p^2}$. Finally, the empirical h–depth is computed as:

  $$\hat{f}_h(x_0) = N^{-1} \sum_{i=1}^{N} K(m/h)$$

  where x is dataset with p observed fucntional data, m is a suitable metric or semi–metric, $K(t)$ is an asymmetric kernel function and h is the bandwidth parameter.

Value

- lmed: Index deepest element median.
- ltrim: Index of curves with trimmed mean mtrim.
- dep: Depth of each curve of fdataobj w.r.t. fdataori.
- dfunc: Second depth function used as multivariate depth, see details section.
- par.dfunc: List of parameters for the dfunc depth function.
- proj: The projection value of each point on the curves.
- dist: Distance matrix between curves or functional data.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Cuevas, A., Febrero-Bande, M. and Fraiman, R. (2007). *Robust estimation and classification for functional data via projection-based depth notions.* Computational Statistics 22, 3, 481-496.

See Also

See Also as Descriptive.

Examples

```
## Not run:
data(tecator)
xx<-tecator$absorp
xx1<-fdata.deriv(xx,1)
lx<-list(xx=xx,xx=xx1)
# Fraiman-Muniz Depth
par.df<-list(scale =TRUE)
out.FM1p=depth.FMp(lx,trim=0.1,draw=TRUE, par.dfunc = par.df)
out.FM2p=depth.FMp(lx,trim=0.1,dfunc="mdepth.LD",
par.dfunc = par.df, draw=TRUE)

# Random Project Depth
out.RP1p=depth.RPp(lx,trim=0.1,dfunc="mdepth.TD",
draw=TRUE,par.dfunc = par.df)
out.RP2p=depth.RPp(lx,trim=0.1,dfunc="mdepth.LD",
draw=TRUE,par.dfunc = par.df)

#Modal Depth
out.mode1p=depth.modep(lx,trim=0.1,draw=T,scale=T)
out.mode2p=depth.modep(lx,trim=0.1,method="manhattan",
draw=T,scale=T)

par(mfrow=c(2,3))
plot(out.FM1p$dep,out.FM2p$dep)
plot(out.RP1p$dep,out.RP2p$dep)
plot(out.mode1p$dep,out.mode2p$dep)
plot(out.FM1p$dep,out.RP1p$dep)
plot(out.RP1p$dep,out.mode1p$dep)
plot(out.FM1p$dep,out.mode1p$dep)

## End(Not run)
```

`Descriptive` *Descriptive measures for functional data.*

Description

Central and dispersion measures for functional data.

Usage

```
func.mean(x)

func.var(fdataobj)

func.trim.FM(fdataobj, ...)

func.trim.mode(fdataobj, ...)

func.trim.RP(fdataobj, ...)

func.trim.RT(fdataobj, ...)

func.trim.RPD(fdataobj, ...)

func.med.FM(fdataobj, ...)

func.med.mode(fdataobj, ...)

func.med.RP(fdataobj, ...)

func.med.RT(fdataobj, ...)

func.med.RPD(fdataobj, ...)

func.trimvar.FM(fdataobj, ...)

func.trimvar.mode(fdataobj, ...)

func.trimvar.RP(fdataobj, ...)

func.trimvar.RPD(fdataobj, ...)

func.trim.RT(fdataobj, ...)

func.med.RT(fdataobj, ...)

func.trimvar.RT(fdataobj, ...)
func.mean.formula(formula, data = NULL, ..., drop = FALSE)
```

Arguments

`x`	`fdata` or `ldata` class object.
`fdataobj`	`fdata` class object.
`...`	Further arguments passed to or from other methods. If the argument `p` is passed, it used `metric.lp` function, by default `p=2`. If the argument `trim` (alpha of the trimming) is passed, it used `metric.lp` function. If the argument `deriv` (number of derivatives to use) is passed. This parameter is used in `depth.RPD` function, by default it uses `deriv =(0,1)`.
`formula`	a formula, such as y ~ group, where y is a fdata object to be split into groups according to the grouping variable group (usually a factor).
`data`	List that containing the variables in the formula. The item called *"df"* is a data frame with the grouping variable. The item called *"y"* is a fdata object.
`drop`	logical indicating if levels that do not occur should be dropped (if f is a factor or a list).

Value

`func.mean.formula` The value returned from split is a list of fdata containing the mean curves for the groups. The components of the list are named by the levels of f (after converting to a factor, or if already a factor and drop = TRUE, dropping unused levels).

`func.mean` gives mean curve.
`func.var` gives variance curve.
`func.trim.FM` Returns the average from the `(1-trim)`% deepest curves following FM criteria.
`func.trim.mode` Returns the average from the `(1-trim)`% deepest curves following mode criteria.
`func.trim.RP` Returns the average from the `(1-trim)`% deepest curves following RP criteria.
`func.trim.RT` Returns the average from the `(1-trim)`% deepest curves following RT criteria.
`func.trim.RPD` Returns the average from the `(1-trim)`% deepest curves following RPD criteria.
`func.med.FM` Returns the deepest curve following FM criteria.
`func.med.mode` Returns the deepest curve following mode criteria.
`func.med.RP` Returns the deepest curve following RP criteria.
`func.med.RPD` Returns the deepest curve following RPD criteria.
`func.trimvar.FM` Returns the marginal variance from the deepest curves followinng FM criteria.
`func.trimvar.mode` Returns the marginal variance from the deepest curves followinng mode criteria.
`func.trimvar.RP` Returns the marginal variance from the deepest curves followinng RP criteria.
`func.trimvar.RT` Returns the marginal variance from the deepest curves followinng RT criteria.
`func.trimvar.RPD` Returns the marginal variance from the deepest curves followinng RPD criteria.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

Examples

```
## Not run:
#' # Example with Montreal Daily Temperature (fda-package)
fdataobj<-fdata(MontrealTemp)

# Measures of central tendency by group
fac<-factor(c(rep(1,len=17),rep(2,len=17)))
ldata=list("df"=data.frame(fac),"fdataobj"=fdataobj)
a1<-func.mean.formula(fdataobj~fac,data=ldata)
plot(a1)

# Measures of central tendency
a1<-func.mean(fdataobj)
a2<-func.trim.FM(fdataobj)
a3<-func.trim.mode(fdataobj)
a4<-func.trim.RP(fdataobj)
# a5<-func.trim.RPD(fdataobj,deriv=c(0,1)) # Time-consuming
a6<-func.med.FM(fdataobj)
a7<-func.med.mode(fdataobj)
a8<-func.med.RP(fdataobj)
# a9<-func.med.RPD(fdataobj,deriv=c(0,1)) # Time-consuming
# a10<-func.med.RT(fdataobj)

par(mfrow=c(1,2))
plot(c(a1,a2,a3,a4),ylim=c(-26,29),main="Central tendency: trimmed mean")
plot(c(a1,a6,a7,a8),ylim=c(-26,29),main="Central tendency: median")

## Measures of dispersion
b1<-func.var(fdataobj)
b2<-func.trimvar.FM(fdataobj)
b3<-func.trimvar.FM(fdataobj,trim=0.1)
b4<-func.trimvar.mode(fdataobj)
b5<-func.trimvar.mode(fdataobj,p=1)
b6<-func.trimvar.RP(fdataobj)
b7<-func.trimvar.RPD(fdataobj)
b8<-func.trimvar.RPD(fdataobj)
b9<-func.trimvar.RPD(fdataobj,deriv=c(0,1))
dev.new()
par(mfrow=c(1,2))
plot(c(b1,b2,b3,b4,b5),ylim=c(0,79),main="Measures of dispersion I")
plot(c(b1,b6,b7,b8,b9),ylim=c(0,79),main="Measures of dispersion II")

## End(Not run)
```

`dev.S` *The deviance score*

Description

Returns the deviance of a fitted model object by GCV score.

Usage

```
dev.S(
  y,
  S,
  obs,
  family = gaussian(),
  off,
  offdf,
  criteria = "GCV",
  W = diag(1, ncol = ncol(S), nrow = nrow(S)),
  trim = 0,
  draw = FALSE,
  ...
)
```

Arguments

`y`	Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
`S`	Smoothing matrix.
`obs`	observed response.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`off`	off
`offdf`	off, degrees of freedom
`criteria`	The penalizing function. By default *"Rice"* criteria. Possible values are *"GCV"*, *"AIC"*, *"FPE"*, *"Shibata"*, *"Rice"*.
`W`	Matrix of weights.
`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`...`	Further arguments passed to or from other methods.

Details

Up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.

$$GCV(h) = p(h)\Xi(n^{-1}h^{-1})$$

Where

$$p(h) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - r_i(x_i)\Big)^2 w(x_i)$$

and penalty function

$$\Xi()$$

can be selected from the following criteria:

Generalized Cross-validation (GCV):

$$\Xi_{GCV}(n^{-1}h^{-1}) = (1 - n^{-1}S_{ii})^{-2}$$

Akaike's Information Criterion (AIC):

$$\Xi_{AIC}(n^{-1}h^{-1}) = exp(2n^{-1}S_{ii})$$

Finite Prediction Error (FPE)

$$\Xi_{FPE}(n^{-1}h^{-1}) = \frac{(1 + n^{-1}S_{ii})}{(1 - n^{-1}S_{ii})}$$

Shibata's model selector (Shibata):

$$\Xi_{Shibata}(n^{-1}h^{-1}) = (1 + 2n^{-1}S_{ii})$$

Rice's bandwidth selector (Rice):

$$\Xi_{Rice}(n^{-1}h^{-1}) = (1 - 2n^{-1}S_{ii})^{-1}$$

Value

Returns GCV score calculated for input parameters.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as GCV.S.
Alternative method: CV.S

Examples

```
data(phoneme)
mlearn<-phoneme$learn
np<-ncol(mlearn)
tt<-mlearn[["argvals"]]
S1 <- S.NW(tt,2.5)
gcv1 <- dev.S(mlearn$data[1,],obs=(sample(150)),
S1,off=rep(1,150),offdf=3)
gcv2 <- dev.S(mlearn$data[1,],obs=sort(sample(150)),
S1,off=rep(1,150),offdf=3)
```

`dfv.test` *Delsol, Ferraty and Vieu test for no functional-scalar interaction*

Description

The function `dfv.test` tests the null hypothesis of no interaction between a functional covariate and a scalar response in a general framework. The null hypothesis is

$$H_0 : m(X) = 0,$$

where $m(\cdot)$ denotes the regression function of the functional variate X over the centred scalar response Y ($E[Y] = 0$). The null hypothesis is tested by the smoothed integrated square error of the response (see Details).

Usage

```
dfv.statistic(
  X.fdata,
  Y,
  h = quantile(x = metric.lp(X.fdata), probs = c(0.05, 0.1, 0.15, 0.25, 0.5)),
  K = function(x) 2 * dnorm(abs(x)),
  weights = rep(1, dim(X.fdata$data)[1]),
  d = metric.lp,
  dist = NULL
)

dfv.test(
  X.fdata,
  Y,
  B = 5000,
  h = quantile(x = metric.lp(X.fdata), probs = c(0.05, 0.1, 0.15, 0.25, 0.5)),
  K = function(x) 2 * dnorm(abs(x)),
  weights = rep(1, dim(X.fdata$data)[1]),
  d = metric.lp,
  verbose = TRUE
)
```

Arguments

`X.fdata`	Functional covariate. The object must be in the class `fdata`.
`Y`	Scalar response. Must be a vector with the same number of elements as functions are in `X.fdata`.
`h`	Bandwidth parameter for the kernel smoothing. This is a crucial parameter that affects the power performance of the test. One possibility to choose it is considering the Cross-validatory bandwidth of the nonparametric functional regression, given by the function `fregre.np` (see Examples). Other possibility is to consider a grid of bandwidths. This is the default option, considering the grid given by the quantiles 0.05, 0.10, 0.15, 0.25 and 0.50 of the functional L^2 distances of the data.
`K`	Kernel function. If no specified it is taken to be the rescaled right part of the normal density.
`weights`	A vector of weights for the sample data. The default is the uniform weights `rep(1,dim(X.fdata$data)[1])`.
`d`	Semimetric to use in the kernel smoothers. By default is the L^2 distance given by `metric.lp`.
`dist`	Matrix of distances of the functional data, used to save time in the bootstrap calibration. If not given, the matrix is automatically computed using the semimetric `d`.
`B`	Number of bootstrap replicates to calibrate the distribution of the test statistic. `B=5000` replicates are the recommended for carry out the test, although for exploratory analysis (**not inferential**), an acceptable less time-consuming option is `B=500`.
`verbose`	Either to show or not information about computing progress.

Details

The Delsol, Ferraty and Vieu statistic is defined as

$$T_n = \int \left(\sum_{i=1}^{n} (Y_i - m(X_i)) K\left(\frac{d(X, X_i)}{h} \right) \right)^2 \omega(X) dP_X(X)$$

and in the case of no interaction with centred scalar response (when $H_0 : m(X) = 0$ holds), its sample version is computed from

$$T_n = \frac{1}{n}\sum_{j=1}^{n}\left(\sum_{i=1}^{n} Y_i K\left(\frac{d(X_j, X_i)}{h}\right)\right)^2 \omega(X_j).$$

The sample version implemented here does not consider a splitting of the sample, as the authors comment in their paper. The statistic is computed by the function `dfv.statistic` and, before applying the test, the response Y is centred. The distribution of the test statistic is approximated by a wild bootstrap on the residuals, using the *golden section bootstrap*.

Please note that if a grid of bandwidths is passed, a harmless warning message will prompt at the end of the test (it comes from returning several p-values in the `htest` class).

Value

The value of `dfv.statistic` is a vector of length `length(h)` with the values of the statistic for each bandwidth. The value of `dfv.test` is an object with class `"htest"` whose underlying structure is a list containing the following components:

- `statistic`: The value of the Delsol, Ferraty and Vieu test statistic.
- `boot.statistics`: A vector of length `B` with the values of the bootstrap test statistics.
- `p.value`: The p-value of the test.
- `method`: The character string "Delsol, Ferraty and Vieu test for no functional-scalar interaction".
- `B`: The number of bootstrap replicates used.
- `h`: Bandwidth parameters for the test.
- `K`: Kernel function used.
- `weights`: The weights considered.
- `d`: Matrix of distances of the functional data.
- `data.name`: The character string "Y=0+e".

Note

No NA's are allowed neither in the functional covariate nor in the scalar response.

Author(s)

Eduardo Garcia-Portugues. Please, report bugs and suggestions to
`<eduardo.garcia.portugues@uc3m.es>`

References

Delsol, L., Ferraty, F. and Vieu, P. (2011). Structural test in regression on functional variables. Journal of Multivariate Analysis, 102, 422-447. doi:10.1016/j.jmva.2010.10.003

Delsol, L. (2013). No effect tests in regression on functional variable and some applications to spectrometric studies. Computational Statistics, 28(4), 1775-1811. doi:10.1007/s0018001203781

See Also

rwild, flm.test, flm.Ftest, fregre.np

Examples

```
## Not run:
## Simulated example ##
X=rproc2fdata(n=50,t=seq(0,1,l=101),sigma="OU")

beta0=fdata(mdata=rep(0,length=101)+rnorm(101,sd=0.05),
argvals=seq(0,1,l=101),rangeval=c(0,1))
beta1=fdata(mdata=cos(2*pi*seq(0,1,l=101))-(seq(0,1,l=101)-0.5)^2+
rnorm(101,sd=0.05),argvals=seq(0,1,l=101),rangeval=c(0,1))

# Null hypothesis holds
Y0=drop(inprod.fdata(X,beta0)+rnorm(50,sd=0.1))

# Null hypothesis does not hold
Y1=drop(inprod.fdata(X,beta1)+rnorm(50,sd=0.1))

# We use the CV bandwidth given by fregre.np
# Do not reject H0
dfv.test(X,Y0,h=fregre.np(X,Y0)$h.opt,B=100)
# dfv.test(X,Y0,B=5000)

# Reject H0
dfv.test(X,Y1,B=100)
# dfv.test(X,Y1,B=5000)

## End(Not run)
```

---

dis.cos.cor *Proximities between functional data*

Description

Computes the cosine correlation distance between two functional dataset of class fdata.

Usage

```
dis.cos.cor(fdata1, fdata2 = NULL, as.dis = FALSE)
```

Arguments

fdata1	Functional data 1 or curve 1.
fdata2	Functional data 2 or curve 2.
as.dis	Returns the distance matrix como 1-cor(fdata1,fdata2).

Value

Returns a proximity/distance matrix (depending on `as.dis`) between functional data.

References

Kemmeren P, van Berkum NL, Vilo J, et al. (2002). *Protein Interaction Verification and Functional Annotation by Integrated Analysis of Genome-Scale Data* . Mol Cell. 2002 9(5):1133-43.

See Also

See also `metric.lp` and `semimetric.NPFDA`

Examples

```
## Not run:
 r1<-rnorm(1001,sd=.01)
 r2<-rnorm(1001,sd=.01)
 x<-seq(0,2*pi,length=1001)
 fx<-fdata(sin(x)/sqrt(pi)+r1,x)
 dis.cos.cor(fx,fx)
 dis.cos.cor(c(fx,fx),as.dis=TRUE)
 fx0<-fdata(rep(0,length(x))+r2,x)
 plot(c(fx,fx0))
 dis.cos.cor(c(fx,fx0),as.dis=TRUE)

## End(Not run)
```

`fanova.hetero` *ANOVA for heteroscedastic data*

Description

Univariate ANOVA for heteroscedastic data.

Usage

```
fanova.hetero(object = NULL, formula, pr = FALSE, contrast = NULL, ...)
```

Arguments

`object`	A data frame with dimension (`n x p+1`). In the first column contains the `n` response values and on the following `p` columns the explanatory variables specified in the formula.
`formula`	as formula.
`pr`	If TRUE, print intermediate results.
`contrast`	List of special contrast to be used, by default no special contrasts are used (`contrast=NULL`).
`...`	Further arguments passed to or from other methods.

Details

This function fits a univariate analysis of variance model and allows calculate special contrasts defined by the user. The list of special contrast to be used for some of the factors in the formula. Each matrix of the list has `r` rows and `r-1` columns.

The user can also request special predetermined contrasts, for example using `contr.helmert`, `contr.sum` or `contr.treatment` functions.

Value

Return:

- `ans`: A list with components including: the Beta estimation `Est`, the factor degrees of freedom `df1`, the residual degrees of freedom `df2`, and the `p-value` for each factor.
- `contrast`: List of special contrasts.

Note

anova.hetero deprecated

It only works with categorical variables.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Brunner, E., Dette, H., Munk, A. *Box-Type Approximations in Nonparametric Factorial Designs.* Journal of the American Statistical Association, Vol. 92, No. 440 (Dec., 1997), pp. 1494-1502.

See Also

See Also as: `fanova.RPm`

Examples

```
## Not run:
data(phoneme)
ind=1 # beetwen 1:150
fdataobj=data.frame(phoneme$learn[["data"]][,ind])
n=dim(fdataobj)[1]
group<-factor(phoneme$classlearn)

#ex 1: real factor and random factor
group.rand=as.factor(sample(rep(1:3,n),n))
f=data.frame(group,group.rand)
mm=data.frame(fdataobj,f)
colnames(mm)=c("value","group","group.rand")
out1=fanova.hetero(object=mm[,-2],value~group.rand,pr=FALSE)
out2=fanova.hetero(object=mm[,-3],value~group,pr=FALSE)
out1
```

```
out2

#ex 2: real factor, random factor and  special contrasts
cr5=contr.sum(5)  #each level vs last level
cr3=c(1,0,-1) #first level vs last level
out.contrast=fanova.hetero(object=mm[,-3],value~group,pr=FALSE,
contrast=list(group=cr5))
out.contrast

## End(Not run)
```

`fanova.onefactor` *One–way anova model for functional data*

Description

One–way anova model for k independent samples of functional data. The function contrasts the null hypothesis of equality of mean functions of functional data based on the an asymptotic version of the anova F–test.

$$H_0 : m_1 = \ldots = m_k$$

Usage

```
fanova.onefactor(
  object,
  group,
  nboot = 100,
  plot = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	functional response data. fdata class object with `n` curves.
`group`	a factor specifying the class for each curve.
`nboot`	number of bootstrap samples.
`plot`	if `TRUE`, plot the mean of each factor level and the results of test.
`verbose`	if `TRUE`, print intermediate results.
`...`	further arguments passed to or from other methods.

Details

The function returns the p–value of test using one–way anova model over `nboot` runs.

Value

Returns:

- `p-value`: Probability of rejecting the null hypothesis H0 at a significance level.
- `stat`: Statistic value of the test.
- `wm`: Statistic values of bootstrap resamples.

Note

anova.onefactor deprecated.

Author(s)

Juan A. Cuesta-Albertos, Manuel Febrero-Bande, Manuel Oviedo de la Fuente
`<manuel.oviedo@udc.es>`

References

Cuevas, A., Febrero, M., & Fraiman, R. (2004). *An anova test for functional data.* Computational statistics & data analysis, **47**(1), 111-122.

See Also

See Also as: `fanova.RPm`

Examples

```
## Not run:
data(MCO)
grupo<-MCO$classintact
datos<-MCO$intact
res=fanova.onefactor(datos,grupo,nboot=50,plot=TRUE)
grupo <- MCO$classpermea
datos <- MCO$permea
res=fanova.onefactor(datos,grupo,nboot=50,plot=TRUE)

## End(Not run)
```

`fanova.RPm` *Functional ANOVA with Random Project.*

Description

The procedure is based on the analysis of randomly chosen one-dimensional projections. The function tests ANOVA models for functional data with continuous covariates and perform special contrasts for the factors in the formula.

Usage

```
fanova.RPm(
  object,
  formula,
  data.fac,
  RP = min(30, ncol(object)),
  alpha = 0.95,
  zproj = NULL,
  par.zproj = list(norm = TRUE),
  hetero = TRUE,
  pr = FALSE,
  w = rep(1, ncol(object)),
  nboot = 0,
  contrast = NULL,
  ...
)

## S3 method for class 'fanova.RPm'
summary(object, ndec = NULL, ...)
```

Arguments

`object`	Functional response data. Object with class fdata with `n` curves discretizated in `m` points. For multivariate problems `object` can be a `data.frame` or a `matrix`
`formula`	as formula without response.
`data.fac`	Explanatory variables. Data frame with dimension (`n x p`), where `p` are the number of factors or covariates considered.
`RP`	Vector of number of random projections.
`alpha`	Confidence level, by default `alpha`=0.95.
`zproj`	Function for generating the projections or an object that contains that projections.
`par.zproj`	List of parameters for `zproj` function.
`hetero`	`logical`. If `TRUE` (by default) means heteroskedastic ANOVA.
`pr`	`logical`. If `TRUE` prints intermediate results.
`w`	Vector of weights (only for multivariate problems).
`nboot`	Number of bootstrap samples, by default no bootstrap computations, `nboot`=0.
`contrast`	List of special contrast to be used ; by default no special contrasts are used (`contrast`=`NULL`).
`...`	Further arguments passed to or from other methods.
`ndec`	Number of decimals.

Details

`zproj` allows to change the generator process of the projections. This can be done through the inclusion of a function or a collection of projections generated outside the function. By default, for a functional problem, the function `rproc2fdata` is used. For multivariate problems, if no function is included, the projections are generated by a normalized gaussian process of the same dimension as `object`. Any user function can be included with the only limitation that the two first parameters are:

- `n`: number of projections
- `t`: discretization points for functional problems
- `m`: number of columns for multivariate problems.

That functions must return a `fdata` or `matrix` object respectively.

The function allows user-defined contrasts. The list of contrast to be used for some of the factors in the formula. Each contrast matrix in the list has `r` rows, where `r` is the number of factor levels. The user can also request special predetermined contrasts, for example using the `contr.helmert`, `contr.sum` or `contr.treatment` functions.

The function returns (by default) the significance of the variables using the Bonferroni test and the False Discovery Rate test. Bootstrap procedure provides more precision

Value

An object with the following components:

- `proj`: The projection value of each point on the curves. Matrix with dimensions (`RP` x `m`), where `RP` is the number of projections and `m` is the number of points observed in each projection curve.
- `mins`: Minimum number for each random projection.
- `result`: p-value for each random projection.
- `test.Bonf`: Significance (TRUE or FALSE) for vector of random projections `RP` in columns and factors (and special contrasts) in rows.
- `p.Bonf`: p-value for vector of random projections `RP` in columns and factors (and special contrasts) in rows.
- `test.fdr`: False Discovery Rate (TRUE or FALSE) for vector of random projections `RP` in columns and factors (and special contrasts) in rows.
- `p.fdr`: p-value of False Discovery Rate for vector of random projections `RP` in columns and factors (and special contrasts) in rows.
- `test.Boot`: False Discovery Rate (TRUE or FALSE) for vector of random projections `RP` in columns and factors (and special contrasts) in rows.
- `p.Boot`: p-value of Bootstrap sample for vector of random projections `RP` in columns and factors (and special contrasts) in rows.

Note

anova.RPm deprecated.

If `hetero=TRUE` then all factors must be categorical.

Author(s)

Juan A. Cuesta-Albertos, Manuel Febrero-Bande, Manuel Oviedo de la Fuente
<manuel.oviedo@udc.es>

References

Cuesta-Albertos, J.A., Febrero-Bande, M. *A simple multiway ANOVA for functional data.* TEST 2010, DOI **10.1007/s11749-010-0185-3**.

See Also

See Also as: fanova.onefactor

Examples

```
## Not run:
# ex fanova.hetero
data(phoneme)
names(phoneme)
# A MV matrix obtained from functional data
data=as.data.frame(phoneme$learn$data[,c(1,seq(0,150,10)[-1])])
group=phoneme$classlearn
n=nrow(data)
group.rand=as.factor(sample(rep(1:3,len=n),n))
RP=c(2,5,15,30)

#ex 1: real factor and random factor
m03=data.frame(group,group.rand)
resul1=fanova.RPm(phoneme$learn,~group+group.rand,m03,RP=c(5,30))
summary(resul1)

#ex 2: real factor with special contrast
m0=data.frame(group)
cr5=contr.sum(5)   #each level vs last level
resul03c1=fanova.RPm(data,~group,m0,contrast=list(group=cr5))
summary(resul03c1)

#ex 3: random factor with special contrast. Same projs as ex 2.
m0=data.frame(group.rand)
zz=resul03c1$proj
cr3=contr.sum(3)   #each level vs last level
resul03c1=fanova.RPm(data,~group.rand,m0,contrast=list(group.rand=cr3),zproj=zz)
summary(resul03c1)

## End(Not run)
```

`fda.usc.internal` *fda.usc internal functions*

Description

Internal undocumentation functions for fda.usc package.

Usage

```
trace.matrix(x, na.rm = TRUE)

argvals.equi(tt)

## S3 method for class 'fdata'
fdata1 + fdata2

## S3 method for class 'fdata'
fdata1 - fdata2

## S3 method for class 'fdata'
fdata1 * fdata2

## S3 method for class 'fdata'
fdata1 / fdata2

## S3 method for class 'fdata'
fdataobj[i = TRUE, j = TRUE, drop = FALSE]

## S3 method for class 'fdata'
fdata1 != fdata2

## S3 method for class 'fdata'
fdata1 == fdata2

## S3 method for class 'fdata'
fdataobj ^ pot

## S3 method for class 'fdata'
dim(x)

ncol.fdata(x)

nrow.fdata(x)

## S3 method for class 'fdata'
length(x)
```

```
NROW.fdata(x)

NCOL.fdata(x)

rownames.fdata(x)

colnames.fdata(x)

## S3 method for class 'fdata'
c(...)

argvals(fdataobj)

rangeval(fdataobj)

## S3 method for class 'fdist'
fdataobj[i = TRUE, j = TRUE, drop = FALSE]

## S3 method for class 'fdata'
is.na(x)

## S3 method for class 'fdata'
anyNA(x, recursive = FALSE)

count.na.fdata(x)

unlist_fdata(x, recursive = TRUE, use.names = TRUE)
```

Arguments

`x`	`matrix` or `fdata` class object.
`na.rm`	`logical`. Should missing values (including `NaN`) be removed?
`tt`	Argvals
`fdataobj`, `fdata1`, `fdata2`	`fdata` class object.
`i`, `j`	Indices specifying elements to extract, replace. Indices are numeric or character vectors or empty
`drop`	For `fdata` class object. If TRUE the result is coerced to the lowest possible dimension of element `data`. This only works for extracting elements, not for the replacement.
`pot`	Numeric value for exponentiation.
`...`	`fdata` objects to be concatenated.
`recursive`	should `anyNA` be applied recursively to lists and pairlists? (in anyNA.fdata function) `logical` Should unlisting be applied to list components of `x`? (in unlist_fdata function).
`use.names`	`logical` Should names be preserved?

Details

`argvals.equi` function returns `TRUE` if the argvals are equispaced and `FALSE` in othercase.

Note

In "Ops" functions `"+.fdata"`, `"-.fdata"`, `"*.fdata"` and `"/.fdata"`: The lengths of the objects `fdata1` and `fdata2` may be different because operates recycled into minimum size as necessary.

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. doi:10.18637/jss.v051.i04

`fdata` *Converts raw data or other functional data classes into fdata class.*

Description

Create a functional data object of class `fdata` from (`matrix`, `data.frame`, `numeric`, `integer`, `fd`, `fds`, `fts` or `sfts`) class data.

Usage

```
fdata(mdata, argvals = NULL, rangeval = NULL, names = NULL, fdata2d = FALSE)
```

Arguments

`mdata`	Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
`argvals`	Argvals, by default: `1:m`.
`rangeval`	(optional) Range of discretization points, by default: range(`argvals`).
`names`	(optional) list with tree components: `main` an overall title, `xlab` title for `x` axis and `ylab` title for `y` axis.
`fdata2d`	TRUE class fdata2d, the functional data is observed in at least a two grids (the `argvals` is a list of vectors). By default `fdata2d=FALSE` the functional data is observed in a single grid (the `argvals` is a vector).

Value

Return `fdata` class object with:

- `"data"`: matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve
- `"rangeval"`: the discretizations points values, if not provided: `1:m`
- `"rangeval"`: range of the discretizations points values, by default: range(`argvals`)
- `"names"`: (optional) list with `main` an overall title, `xlab` title for `x` axis and `ylab` title for `y` axis.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as plot.fdata

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme$learn[1:4,1:150]
# Center curves
fdata.c=fdata.cen(mlearn)$Xcen
par(mfrow=c(2,1))
plot(mlearn,type="l")
plot(fdata.c,type="l")

# Convert  from class fda to fdata
bsp1 <- create.bspline.basis(c(1,150),21)
fd1 <- Data2fd(1:150,y=t(mlearn$data),basisobj=bsp1)
fdataobj=fdata(fd1)

# Convert  from class fds, fts or sfts to fdata
#require(fds)
#a=fds(x = 1:20, y = Simulationdata$y, xname = "x",
# yname = "Simulated value")
#b=fts(x = 15:49, y = Australiasmoothfertility$y, xname = "Age",
#    yname = "Fertility rate")
#c=sfts(ts(as.numeric(ElNino_ERSST_region_1and2$y), frequency = 12), xname = "Month",
#yname = "Sea surface temperature")
#class(a);class(b);class(c)
#fdataobj=fdata(b)

## End(Not run)
```

`fdata.bootstrap` *Bootstrap samples of a functional statistic*

Description

provides bootstrap samples for functional data.

Usage

```
fdata.bootstrap(
  fdataobj,
  statistic = func.mean,
  alpha = 0.05,
  nb = 200,
  smo = 0,
  draw = FALSE,
  draw.control = NULL,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`statistic`	Sample statistic. It must be a function that returns an object of class `fdata`. By default, it uses sample mean `func.mean`. See `Descriptive` for other statistics.
`alpha`	Significance value.
`nb`	Number of bootstrap resamples.
`smo`	The smoothing parameter for the bootstrap samples as a proportion of the sample variance matrix.
`draw`	If `TRUE`, plot the bootstrap samples and the statistic.
`draw.control`	List that it specifies the `col`, `lty` and `lwd` for objects: `fdataobj`, `statistic`, `IN` and `OUT`.
`...`	Further arguments passed to or from other methods.

Details

The `fdata.bootstrap` computes a confidence ball using bootstrap in the following way:

- Let $X_1(t),\dots,X_n(t)$ be the original data and $T = T(X_1(t),\dots,X_n(t))$ be the sample statistic.
- Calculate the `nb` bootstrap resamples $\{X_1^*(t),\cdots,X_n^*(t)\}$, using the following scheme: $X_i^*(t) = X_i(t) + Z(t)$, where $Z(t)$ is normally distributed with mean 0 and covariance matrix $\gamma\Sigma_x$, where Σ_x is the covariance matrix of $\{X_1(t),\dots,X_n(t)\}$ and γ is the smoothing parameter.
- Let $T^{*j} = T(X_1^{*j}(t),...,X_n^{*j}(t))$ be the estimate using the j resample.
- Compute $d(T,T^{*j})$, $j = 1,\dots,nb$. Define the bootstrap confidence ball of level $1-\alpha$ as $CB(\alpha) = X \in E$ such that $d(T,X) \leq d_\alpha$ being d_α the quantile $(1-\alpha)$ of the distances between the bootstrap resamples and the sample estimate.

The `fdata.bootstrap` function allows us to define a statistic calculated on the `nb` resamples, control the degree of smoothing by `smo` argument and represent the confidence ball with level $1-\alpha$ as those resamples that fulfill the condition of belonging to $CB(\alpha)$. The `statistic` used by default is the mean (`func.mean`) but also other depth-based functions can be used (see `help(Descriptive)`).

Value

- statistic: fdata class object with the statistic estimate from nb bootstrap samples.
- dband: Bootstrap estimate of (1-alpha)% distance.
- rep.dist: Distance from every replicate.
- resamples: fdata class object with the bootstrap resamples.
- fdataobj: fdata class object.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Cuevas A., Febrero-Bande, M. and Fraiman, R. (2007). *Robust estimation and classification for functional data via projection-based depth notions.* Computational Statistics 22, 3: 481-496.

Cuevas A., Febrero-Bande, M., Fraiman R. 2006. *On the use of bootstrap for estimating functions with functional data.* Computational Statistics and Data Analysis 51: 1063-1074.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as Descriptive

Examples

```
## Not run:
data(tecator)
absorp<-tecator$absorp.fdata
# Time consuming
#Bootstrap for Trimmed Mean with depth mode
out.boot=fdata.bootstrap(absorp,statistic=func.trim.FM,nb=200,draw=TRUE)
names(out.boot)
#Bootstrap for Median with with depth mode
control=list("col"=c("grey","blue","cyan"),"lty"=c(2,1,1),"lwd"=c(1,3,1))
out.boot=fdata.bootstrap(absorp,statistic=func.med.mode,
draw=TRUE,draw.control=control)

## End(Not run)
```

fdata.cen *Functional data centred (subtract the mean of each discretization point)*

Description

The function fdata.cen centres the curves by subtracting the functional mean.

Usage

```
fdata.cen(fdataobj, meanX = func.mean(fdataobj))
```

Arguments

fdataobj	fdata class object.
meanX	The functional mean subtracted in the fdatobj.

Value

Return:
two fdata class objects with:

Xcen	The centered fdata.
meanX	Functional mean substracted.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See Also as fdata

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme[["learn"]][13:15,]
fdata.c=fdata.cen(mlearn)$Xcen
par(mfrow=c(1,2))
plot(mlearn,type="l")
plot(fdata.c,type="l")

## End(Not run)
```

fdata.deriv — *Computes the derivative of functional data object.*

Description

Computes the derivative of functional data.

- If method =*"bspline"*, *"exponential"*, *"fourier"*, *"monomial"* or *"polynomial"*. `fdata.deriv` function creates a basis to represent the functional data. The functional data are converted to class `fd` using the Data2fd function and the basis indicated in the `method`. Finally, the function calculates the derivative of order `nderiv` of curves using deriv.fd function.
- If `method=`*"fmm"*, *"periodic"*, *"natural"* or *"monoH.FC"* is used `splinefun` function.
- If `method=`*"diff"*, raw derivation is applied. Not recommended to use this method when the values are not equally spaced.

Usage

```
fdata.deriv(
  fdataobj,
  nderiv = 1,
  method = "bspline",
  class.out = "fdata",
  nbasis = NULL,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`nderiv`	Order of derivation, by defalult `nderiv`=1.
`method`	Type of derivative method, for more information see **details**.
`class.out`	Class of functional data returned: `fdata` or `fd` class.
`nbasis`	Number of Basis for `fdatataobj\$DATA`. It is only used if method =*"bspline"*, *"exponential"*, *"fourier"*, *"monomial"* or *"polynomial"*
`...`	Further arguments passed to or from other methods.

Value

Returns the derivative of functional data of `fd` class if `class.out=`"*fd*" or `fdata` class if `class.out=`"*fdata*".

See Also

See also deriv.fd , `splinefun` and `fdata`

Examples

```
data(tecator)
absorp=tecator$absorp.fdata
tecator.fd1=fdata2fd(absorp)
tecator.fd2=fdata2fd(absorp,"fourier",9)
tecator.fd3=fdata2fd(absorp,"fourier",nbasis=9,nderiv=1)
#tecator.fd1;tecator.fd2;tecator.fd3
tecator.fdata1=fdata(tecator.fd1)
tecator.fdata2=fdata(tecator.fd2)
tecator.fdata3=fdata(tecator.fd3)
tecator.fdata4=fdata.deriv(absorp,nderiv=1,method="bspline",
class.out='fdata',nbasis=9)
tecator.fd4=fdata.deriv(tecator.fd3,nderiv=0,class.out='fd',nbasis=9)
plot(tecator.fdata4)
plot(fdata.deriv(absorp,nderiv=1,method="bspline",class.out='fd',nbasis=11))
```

`fdata.methods` *fdata S3 Group Generic Functions*

Description

fdata Group generic methods defined for four specified groups of functions, Math, Ops, Summary and Complex.

order.fdata and split.fdata: A wrapper for the order and split function for fdata object.

Usage

```
## S3 method for class 'fdata'
Math(x, ...)

## S3 method for class 'fdata'
Ops(e1, e2 = NULL)

## S3 method for class 'fdata'
Summary(..., na.rm = FALSE)

## S3 method for class 'fdata'
split(x, f, drop = FALSE, ...)

order.fdata(y, fdataobj, na.last = TRUE, decreasing = FALSE)

is.fdata(fdataobj)
```

Arguments

`x`	An `fdata` object containing values to be divided into groups or an `list` of `fdata` objects containing values to be combine by rows in a to be flatten one `fdata` object.
`...`	Further arguments passed to methods.
`e1, e2`	`fdata` class object
`na.rm`	`logical`: should missing values be removed?
`f`	a factor in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
`drop`	`logical` indicating if levels that do not occur should be dropped (if f is a factor or a list).
`y`	A sequence of numeric, complex, character or logical vectors, all of the same length, or a classed R object.
`fdataobj`	`fdata` class object.
`na.last`	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if "keep" they are kept with rank NA.
`decreasing`	`logical` Should the sort order be increasing or decreasing?.

Details

In `order.fdata` the funcional data is ordered w.r.t the sample order of the values of vector.

`split.fdata` divides the data in the fdata object `x` into the groups defined by `f`.

Value

- split.fdata: The value returned from `split` is a list of fdata objects containing the values for the groups. The components of the list are named by the levels of f (after converting to a factor, or if already a factor and drop = TRUE, dropping unused levels).\
- order.fdata: returns the functional data `fdataobj` w.r.t. a permutation which rearranges its first argument into ascending or descending order.

Author(s)

Manuel Febrero Bande and Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

See Also

See Summary and Complex.

Examples

```
## Not run:
data(tecator)
absor<-tecator$absorp.fdata
absor2<-fdata.deriv(absor,1)
```

```
absor<-absor2[1:5,1:4]
absor2<-absor2[1:5,1:4]
sum(absor)
round(absor,4)
log1<-log(absor)

fdataobj<-fdata(MontrealTemp)
fac<-factor(c(rep(1,len=17),rep(2,len=17)))
a1<-split(fdataobj,fac)
dim(a1[[1]]);dim(a1[[2]])

## End(Not run)
```

---

fdata2basis *Compute fucntional coefficients from functional data represented in a base of functions*

Description

Compute fucntional coefficients from functional data (fdata class object) represented in a basis (fixed of data-driven basis).

Usage

```
fdata2basis(fdataobj, basis, method = c("grid", "inprod"))

## S3 method for class 'basis.fdata'
summary(object, draw = TRUE, index = NULL, ...)
```

Arguments

fdataobj	fdata class object.
basis	a functional basis object defining the basis
method	character string, if it is "grid" the fdata object is evaluated in the grid (argvals of fdata), if it is "inprod" the basis representation of functional data is computed by inner product (inprod.fdata(fdataobj,basis)).
object	basis.fdata class object calculated by: fdata2basis
draw	logical, original curves and their basis representation are plotted
index	vector, by default (if NULL) the first n curves are plotted, where n = min(4, length(fdataobj)). Otherwise, index vector indicates taht curvesare plotted.
...	Further arguments passed to or from other methods.

Value

The `fdata2basis` function returns:

- `coef`: A matrix or two-dimensional array of coefficients.
- `basis`: Basis of `fdata` class evaluated on the same grid as `fdataobj`.

And summary function return:

- `R`: a matrix with a measure similar to R-sq for each curve aproximation (by row) and number of basis elements (by column).

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@usc.es`>

See Also

Inverse function: `gridfdata`. Alternative method: `fdata2pc`, `fdata2pls`.

Examples

```
## Not run:
T <- 71
S <- 51
tj <- round(seq(0,1,len=T),3)
si <- round(seq(0,1,len=S),3)
beta1 <- outer(si,tj,function(si,tj){exp(-5*abs((tj-si)/5))})
nbasis.s =7
nbasis.t=11
base.s <- create.fourier.basis(c(0,1),nbasis=nbasis.s)
base.t <- create.fourier.basis(c(0,1),nbasis=nbasis.t)
y1 <- fdata(rbind(log(1+tj),1-5*(tj-0.5)^2),argvals=tj,rangeval=c(0,1))
aa <- fdata2basis(y1,base.t,method="inprod")
summary(aa)
plot(gridfdata(aa$coefs,aa$basis))
lines(y1,lwd=2,col=c(3,4),lty=2)

## End(Not run)
```

`fdata2fd` *Converts fdata class object into fd class object*

Description

Converts `fdata` class object into `fd` class object using `Data2fd` function.

Usage

```
fdata2fd(
  fdataobj,
  type.basis = NULL,
  nbasis = NULL,
  nderiv = 0,
  lambda = NULL,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`type.basis`	Type of basis. A function `create."type.basis".basis` must exists. By default, `bspline` basis is used.
`nbasis`	Number of basis which is used in `create.basis` function.
`nderiv`	Order of derivation which is used in `deriv.fd` function (optional).
`lambda`	Weight on the smoothing operator specified by `nderiv`.
`...`	Further arguments passed to or from other methods.

Value

Return an object of the `fd` class.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as `fdata` and Data2fd

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme$learn[1]
fdata2=fdata2fd(mlearn)
class(mlearn)
class(fdata2)
fdata3=fdata2fd(mlearn,type.basis="fourier",nbasis=7)
plot(mlearn)
```

```
lines(fdata2,col=2)
lines(fdata3,col=3)
fdata5=fdata2fd(mlearn,nderiv=1)

## End(Not run)
```

---

fdata2pc *Principal components for functional data*

Description

Compute (penalized) principal components for functional data.

Usage

```
fdata2pc(fdataobj, ncomp = 2, norm = TRUE, lambda = 0, P = c(0, 0, 1), ...)
```

Arguments

fdataobj	fdata class object.
ncomp	Number of principal components.
norm	=TRUE the norm of eigenvectors (rotation) is 1.
lambda	Amount of penalization. Default value is 0, i.e. no penalization is used.
P	If P is a vector: coefficients to define the penalty matrix object. By default P=c(0,0,1) penalize the second derivative (curvature) or acceleration. If P is a matrix: the penalty matrix object.
...	Further arguments passed to or from other methods.

Details

Smoothing is achieved by penalizing the integral of the square of the derivative of order m over rangeval:

- m = 0 penalizes the squared difference from 0 of the function
- m = 1 penalize the square of the slope or velocity
- m = 2 penalize the squared acceleration
- m = 3 penalize the squared rate of change of acceleration

Value

- d: The standard deviations of the functional principal components.
- rotation: Also known as loadings. A fdata class object whose rows contain the eigenvectors.
- x: Also known as scores. The value of the rotated functional data is returned.
- fdataobj.cen: The centered fdataobj object.

- mean: The functional mean of the fdataobj object.
- l: Vector of indices of principal components.
- C: The matched call.
- lambda: Amount of penalization.
- P: Penalty matrix.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@usc.es>

References

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S*. Springer-Verlag.

N. Kraemer, A.-L. Boulsteix, and G. Tutz (2008). Penalized Partial Least Squares with Applications to B-Spline Transformations and Functional Data. Chemometrics and Intelligent Laboratory Systems, 94, 60 - 69. doi:10.1016/j.chemolab.2008.06.009

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as svd and varimax.

Examples

```
 ## Not run:
 n= 100;tt= seq(0,1,len=51)
 x0<-rproc2fdata(n,tt,sigma="wiener")
 x1<-rproc2fdata(n,tt,sigma=0.1)
 x<-x0*3+x1
 pc=fdata2pc(x,lambda=1)
 summary(pc)

## End(Not run)
```

---

fdata2pls *Partial least squares components for functional data.*

Description

Compute penalized partial least squares (PLS) components for functional data.

Usage

```
fdata2pls(fdataobj, y, ncomp = 2, lambda = 0, P = c(0, 0, 1), norm = TRUE, ...)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`ncomp`	The number of components to include in the model.
`lambda`	Amount of penalization. Default value is 0, i.e. no penalization is used.
`P`	If P is a vector: coefficients to define the penalty matrix object. By default $P = c(0, 0, 1)$ penalizes the second derivative (curvature) or acceleration. If P is a matrix: the penalty matrix object.
`norm`	If `TRUE` the `fdataobj` are centered and scaled.
`...`	Further arguments passed to or from other methods.

Details

If `norm=TRUE`, computes the PLS by `NIPALS` algorithm and the Degrees of Freedom using the Krylov representation of PLS, see Kraemer and Sugiyama (2011).
If `norm=FALSE`, computes the PLS by Orthogonal Scores Algorithm and the Degrees of Freedom are the number of components `ncomp`, see Martens and Naes (1989).

Value

`fdata2pls` function return:

The `fdata2pls` function returns:

- `df`: Degree of freedom.
- `rotation`: `fdata` class object.
- `x`: The value of the rotated data (the centered data multiplied by the rotation matrix) is returned.
- `fdataobj.cen`: The centered `fdataobj` object.
- `mean`: Mean of `fdataobj`.
- `l`: Vector of indices of principal components.
- `C`: The matched call.
- `lambda`: Amount of penalization.
- `P`: Penalty matrix.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@usc.es`>

References

Kraemer, N., Sugiyama M. (2011). *The Degrees of Freedom of Partial Least Squares Regression*. Journal of the American Statistical Association. Volume 106, 697-705.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

Martens, H., Naes, T. (1989) *Multivariate calibration.* Chichester: Wiley.

See Also

Used in: `fregre.pls`, `fregre.pls.cv`. Alternative method: `fdata2pc`.

Examples

```
## Not run:
n= 500;tt= seq(0,1,len=101)
x0<-rproc2fdata(n,tt,sigma="wiener")
x1<-rproc2fdata(n,tt,sigma=0.1)
x<-x0*3+x1
beta = tt*sin(2*pi*tt)^2
fbeta = fdata(beta,tt)
y<-inprod.fdata(x,fbeta)+rnorm(n,sd=0.1)
pls1=fdata2pls(x,y)
pls1$call
summary(pls1)
pls1$l
norm.fdata(pls1$rotation)

## End(Not run)
```

`FDR` *False Discorvery Rate (FDR)*

Description

Compute the False Discovery Rate for a vector of p-values and alpha value.

Usage

```
FDR(pvalues = NULL, alpha = 0.95, dep = 1)

pvalue.FDR(pvalues = NULL, dep = 1)
```

Arguments

`pvalues`	Vector of p-values
`alpha`	Alpha value (level of significance).
`dep`	Parameter dependence test. By default `dep = 1`, direct dependence between tests.

Details

`FDR` method is used for multiple hypothesis testing to correct problems of multiple contrasts.
If `dep = 1`, the tests are positively correlated, for example when many tests are the same contrast.
If `dep < 1` the tests are negatively correlated.

Value

Return:

- `out.FDR=TRUE`: If there are significative differences.
- `pv.FDR`: p-value for False Discovery Rate test.

Author(s)

Febrero-Bande, M. and Oviedo de la Fuente, M.

References

Benjamini, Y., Yekutieli, D. (2001). *The control of the false discovery rate in multiple testing under dependency.* Annals of Statistics. 29 (4): 1165-1188. DOI:10.1214/aos/1013699998.

See Also

Function used in `fanova.RPm`

Examples

```
p=seq(1:50)/1000
FDR(p)
pvalue.FDR(p)
FDR(p,alpha=0.9999)
FDR(p,alpha=0.9)
FDR(p,alpha=0.9,dep=-1)
```

`fEqDistrib.test` *Tests for checking the equality of distributions between two functional populations.*

Description

Three tests for the equality of distributions of two populations are provided. The null hypothesis is that the two populations are the same

Usage

```
XYRP.test(X.fdata, Y.fdata, nproj = 10, npc = 5, test = c("KS", "AD"))

MMD.test(
  X.fdata,
  Y.fdata,
  metric = "metric.lp",
  B = 1000,
  alpha = 0.95,
  kern = "RBF",
  ops.metric = list(lp = 2),
  draw = FALSE
)

MMDA.test(
  X.fdata,
  Y.fdata,
  metric = "metric.lp",
  B = 1000,
  alpha = 0.95,
  kern = "RBF",
  ops.metric = list(lp = 2),
  draw = FALSE
)

fEqDistrib.test(
  X.fdata,
  Y.fdata,
  metric = "metric.lp",
  method = c("Exch", "WildB"),
  B = 5000,
  ops.metric = list(lp = 2),
  iboot = FALSE
)
```

Arguments

`X.fdata`	`fdata` object containing the curves from the first population.
`Y.fdata`	`fdata` object containing the curves from the second population.
`nproj`	Number of projections for `XYRP.test`.
`npc`	The number of principal components employed for generating the random projections.
`test`	For `XYRP.test` "KS" and/or "AD" for computing Kolmogorov-Smirnov or Anderson-Darling p-values in the projections.
`metric`	Character with the metric function for computing distances among curves.
`B`	Number of bootstrap or Monte Carlo replicas.
`alpha`	Confidence level for computing the threshold. By default =0.95.
`kern`	For `MMDA.test` "RBF" or "metric" for indicating the use of Radial Basis Function or directly, the distances.
`ops.metric`	List of parameters to be used with `metric`.
`draw`	By default, FALSE. Plots the density of the bootstrap replicas jointly with the statistic.
`method`	In `fEqDistrib.test` a character indicating the bootstrap method for computing the distribution under H0. "Exch" for Exchangeable bootstrap and "WildB" for Wild Bootstrap. By default, both are provided.
`iboot`	In `fEqDistrib.test` returns the bootstrap replicas.

Details

XYRP.test computes the p-values using random projections. Requires kSamples library. MMD.test computes Maximum Mean Discrepancy p-values using permutations (see Sejdinovic et al, (2013)) and MMDA.test does the same using an asymptotic approximation. fEqDistrib.test checks the equality of distributions using an embedding in a RKHS and two bootstrap approximations for calibration.

Value

A list with the following components by function:

- XYRP.test, FDR.pv: p-value using FDR, proj.pv: Matrix of p-values obtained for projections.
- MMD.test, MMDA.test: stat: Statistic, p.value: p-value, thresh: Threshold at level alpha.
- fEqDistrib.test, result: data.frame with columns Stat and p.value, Boot: data.frame with bootstrap replicas if iboot=TRUE.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.febrero@usc.es>

References

Sejdinovic, D., Sriperumbudur, B., Gretton, A., Fukumizu, K. *Equivalence of distance-based and RKHS-based statistics in Hypothesis Testing* The Annals of Statistics, 2013. DOI **10.1214/13-AOS1140**.

See Also

fmean.test.fdata, cov.test.fdata.

Examples

```
## Not run:
tt=seq(0,1,len=51)
bet=0
mu1=fdata(10*tt*(1-tt)^(1+bet),tt)
mu2=fdata(10*tt^(1+bet)*(1-tt),tt)
fsig=1
X=rproc2fdata(100,tt,mu1,sigma="vexponential",par.list=list(scale=0.2,theta=0.35))
Y=rproc2fdata(100,tt,mu2,sigma="vexponential",par.list=list(scale=0.2*fsig,theta=0.35))
fmean.test.fdata(X,Y,npc=-.98,draw=TRUE)
cov.test.fdata(X,Y,npc=5,draw=TRUE)
bet=0.1
mu1=fdata(10*tt*(1-tt)^(1+bet),tt)
mu2=fdata(10*tt^(1+bet)*(1-tt),tt)
fsig=1.5
X=rproc2fdata(100,tt,mu1,sigma="vexponential",par.list=list(scale=0.2,theta=0.35))
Y=rproc2fdata(100,tt,mu2,sigma="vexponential",par.list=list(scale=0.2*fsig,theta=0.35))
fmean.test.fdata(X,Y,npc=-.98,draw=TRUE)
```

```
cov.test.fdata(X,Y,npc=5,draw=TRUE)
XYRP.test(X,Y,nproj=15)
MMD.test(X,Y,B=1000)
fEqDistrib.test(X,Y,B=1000)

## End(Not run)
```

`fEqMoments.test` *Tests for checking the equality of means and/or covariance between two populations under gaussianity.*

Description

Two tests for the equality of means and covariances of two populations are provided. Both tests are constructed under gaussianity following Horvath & Kokoszka, 2012, Chapter 5.

Usage

```
fmean.test.fdata(
  X.fdata,
  Y.fdata,
  method = c("X2", "Boot"),
  npc = 5,
  alpha = 0.95,
  B = 1000,
  draw = FALSE
)

cov.test.fdata(
  X.fdata,
  Y.fdata,
  method = c("X2", "Boot"),
  npc = 5,
  alpha = 0.95,
  B = 1000,
  draw = FALSE
)
```

Arguments

`X.fdata`	`fdata` object containing the curves from the first population.
`Y.fdata`	`fdata` object containing the curves from the second population.
`method`	c("X2","Boot"). "X2" includes the asymptotic distribution. "Boot" computes the bootstrap approximation.
`npc`	The number of principal components employed. If `npc` is negative and 0<`abs(npc)`<1, the number of components are determined for explaining, at least, `abs(p)`% of variability.

`alpha`	Confidence level. By default =0.95.
`B`	Number of bootstrap replicas when method="Boot".
`draw`	By default, `FALSE`. Plots the density of the bootstrap replicas jointly with the statistic.

Details

`fmean.test.fdata` computes the test for equality of means. `cov.test.fdata` computes the test for equality of covariance operators. Both tests have asymptotic distributions under the null related with chi-square distribution. Also, a parametric bootstrap procedure is implemented in both cases.

Value

Return a list with:

- `stat`: Value of the statistic.
- `pvalue`: P-values for the test.
- `vcrit`: Critical cutoff for rejecting the null hypothesis.
- `p`: Degrees of freedom for X2 statistic.
- `B`: Number of bootstrap replicas.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.febrero@usc.es`>

References

Inference for Functional Data with Applications. Horvath, L and Kokoszka, P. (2012). Springer.

See Also

See Also as `fanova.RPm`, `fanova.onefactor`.

Examples

```
## Not run:
tt=seq(0,1,len=51)
bet=0
mu1=fdata(10*tt*(1-tt)^(1+bet),tt)
mu2=fdata(10*tt^(1+bet)*(1-tt),tt)
fsig=1
X=rproc2fdata(100,tt,mu1,sigma="vexponential",par.list=list(scale=0.2,theta=0.35))
Y=rproc2fdata(100,tt,mu2,sigma="vexponential",par.list=list(scale=0.2*fsig,theta=0.35))
fmean.test.fdata(X,Y,npc=-.98,draw=TRUE)
cov.test.fdata(X,Y,npc=5,draw=TRUE)
bet=0.1
mu1=fdata(10*tt*(1-tt)^(1+bet),tt)
mu2=fdata(10*tt^(1+bet)*(1-tt),tt)
fsig=1.5
X=rproc2fdata(100,tt,mu1,sigma="vexponential",par.list=list(scale=0.2,theta=0.35))
```

```
Y=rproc2fdata(100,tt,mu2,sigma="vexponential",par.list=list(scale=0.2*fsig,theta=0.35))
fmean.test.fdata(X,Y,npc=-.98,draw=TRUE)
cov.test.fdata(X,Y,npc=5,draw=TRUE)

## End(Not run)
```

`flm.Ftest` *F-test for the Functional Linear Model with scalar response*

Description

The function `flm.Ftest` tests the null hypothesis of no interaction between a functional covariate and a scalar response inside the Functional Linear Model (FLM): $Y = \big\langle X, \beta \big\rangle + \epsilon$. The null hypothesis is $H_0 : \beta = 0$ and the alternative is $H_1 : \beta \neq 0$. The null hypothesis is tested by a functional extension of the classical F-test (see Details).

Usage

```
Ftest.statistic(X.fdata, Y)

flm.Ftest(X.fdata, Y, B = 5000, verbose = TRUE)
```

Arguments

`X.fdata`	Functional covariate for the FLM. The object must be in the class `fdata`.
`Y`	Scalar response for the FLM. Must be a vector with the same number of elements as functions are in `X.fdata`.
`B`	Number of bootstrap replicates to calibrate the distribution of the test statistic. `B=5000` replicates are the recommended for carry out the test, although for exploratory analysis (**not inferential**), an acceptable less time-consuming option is `B=500`.
`verbose`	Either to show or not information about computing progress.

Details

The Functional Linear Model with scalar response (FLM), is defined as $Y = \big\langle X, \beta \big\rangle + \epsilon$, for a functional process X such that $E[X(t)] = 0$, $E[X(t)\epsilon] = 0$ for all t and for a scalar variable Y such that $E[Y] = 0$. The *functional F-test* is defined as

$$T_n = \left\| \frac{1}{n} \sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y}) \right\|,$$

where $\bar{X}$ is the functional mean of X, $\bar{Y}$ is the ordinary mean of Y and $\| \cdot \|$ is the L^2 functional norm. The statistic is computed with the function `Ftest.statistic`. The distribution of the test statistic is approximated by a wild bootstrap resampling on the residuals, using the *golden section bootstrap*.

Value

The value for `Ftest.statistic` is simply the F-test statistic. The value for `flm.Ftest` is an object with class `"htest"` whose underlying structure is a list containing the following components:

- `statistic`: The value of the F-test statistic.
- `boot.statistics`: A vector of length `B` with the values of the bootstrap F-test statistics.
- `p.value`: The p-value of the test.
- `method`: The character string "Functional Linear Model F-test".
- `B`: The number of bootstrap replicates used.
- `data.name`: The character string "Y=<X,0>+e".

Note

No NA's are allowed neither in the functional covariate nor in the scalar response.

Author(s)

Eduardo Garcia-Portugues. Please, report bugs and suggestions to
`<eduardo.garcia.portugues@uc3m.es>`

References

Garcia-Portugues, E., Gonzalez-Manteiga, W. and Febrero-Bande, M. (2014). A goodness–of–fit test for the functional linear model with scalar response. Journal of Computational and Graphical Statistics, 23(3), 761-778. doi:10.1080/10618600.2013.812519

Gonzalez-Manteiga, W., Gonzalez-Rodriguez, G., Martinez-Calvo, A. and Garcia-Portugues, E. Bootstrap independence test for functional linear models. arXiv:1210.1072. https://arxiv.org/abs/1210.1072

See Also

`rwild`, `flm.test`, `dfv.test`

Examples

```
## Not run:
## Simulated example ##
X=rproc2fdata(n=50,t=seq(0,1,l=101),sigma="OU")
beta0=fdata(mdata=rep(0,length=101)+rnorm(101,sd=0.05),
argvals=seq(0,1,l=101),rangeval=c(0,1))
beta1=fdata(mdata=cos(2*pi*seq(0,1,l=101))-(seq(0,1,l=101)-0.5)^2+
rnorm(101,sd=0.05),argvals=seq(0,1,l=101),rangeval=c(0,1))

# Null hypothesis holds
Y0=drop(inprod.fdata(X,beta0)+rnorm(50,sd=0.1))
# Null hypothesis does not hold
Y1=drop(inprod.fdata(X,beta1)+rnorm(50,sd=0.1))

# Do not reject H0
```

```
flm.Ftest(X,Y0,B=100)
flm.Ftest(X,Y0,B=5000)

# Reject H0
flm.Ftest(X,Y1,B=100)
flm.Ftest(X,Y1,B=5000)

## End(Not run)
```

`flm.test` *Goodness-of-fit test for the Functional Linear Model with scalar response*

Description

The function `flm.test` tests the composite null hypothesis of a Functional Linear Model with scalar response (FLM),

$$H_0 : Y = \big\langle X, \beta \big\rangle + \epsilon,$$

versus a general alternative. If $\beta = \beta_0$ is provided, then the simple hypothesis $H_0 : Y = \big\langle X, \beta_0 \big\rangle + \epsilon$ is tested. The testing of the null hypothesis is done by a Projected Cramer-von Mises statistic (see Details).

Usage

```
flm.test(
  X.fdata,
  Y,
  beta0.fdata = NULL,
  B = 5000,
  est.method = "pls",
  p = NULL,
  type.basis = "bspline",
  verbose = TRUE,
  plot.it = TRUE,
  B.plot = 100,
  G = 200,
  ...
)
```

Arguments

`X.fdata`	Functional covariate for the FLM. The object must be in the class `fdata`.
`Y`	Scalar response for the FLM. Must be a vector with the same number of elements as functions are in `X.fdata`.

`beta0.fdata` Functional parameter for the simple null hypothesis, in the `fdata` class. Recall that the `argvals` and `rangeval` arguments of `beta0.fdata` must be the same of `X.fdata`. A possibility to do this is to consider, for example for $\beta_0 = 0$ (the simple null hypothesis of no interaction),
`beta0.fdata=fdata(mdata=rep(0,length(X.fdata$argvals)), argvals=X.fdata$argvals,rangeval=X.fdata$rangeval)`.
If `beta0.fdata=NULL` (default), the function will test for the composite null hypothesis.

`B` Number of bootstrap replicates to calibrate the distribution of the test statistic. `B=5000` replicates are the recommended for carry out the test, although for exploratory analysis (**not inferential**), an acceptable less time-consuming option is `B=500`.

`est.method` Estimation method for the unknown parameter β, only used in the composite case. Mainly, there are two options: specify the number of basis elements for the estimated β by `p` or optimally select `p` by a data-driven criteria (see Details section for discussion). Then, it must be one of the following methods:

- `"pc"`: If `p`, the number of basis elements, is given, then β is estimated by `fregre.pc`. Otherwise, an optimum `p` is chosen using `fregre.pc.cv` and the `"SICc"` criteria.
- `"pls"`: If `p` is given, β is estimated by `fregre.pls`. Otherwise, an optimum `p` is chosen using `fregre.pls.cv` and the `"SICc"` criteria. This is the default argument as it has been checked empirically that provides a good balance between the performance of the test and the estimation of β.
- `"basis"`: If `p` is given, β is estimated by `fregre.basis`. Otherwise, an optimum `p` is chosen using `fregre.basis.cv` and the `"GCV.S"` criteria. In these functions, the same basis for the arguments `basis.x` and `basis.b` is considered. The type of basis used will be the given by the argument `type.basis` and must be one of the class of `create.basis`. Further arguments passed to create.basis (not `rangeval` that is taken as the `rangeval` of `X.fdata`), can be passed throughout `...`.

`p` Number of elements of the basis considered. If it is not given, an optimal `p` will be chosen using a specific criteria (see `est.method` and `type.basis` arguments).

`type.basis` Type of basis used to represent the functional process. Depending on the hypothesis, it will have a different interpretation:

- Simple hypothesis. One of these options:
  - `"bspline"`: If `p` is given, the functional process is expressed in a basis of `p` B-splines. If not, an optimal `p` will be chosen by `optim.basis`, using the `"GCV.S"` criteria.
  - `"fourier"`: If `p` is given, the functional process is expressed in a basis of `p` Fourier functions. If not, an optimal `p` will be chosen by `optim.basis`, using the `"GCV.S"` criteria.
  - `"pc"`: `p` must be given. Expresses the functional process in a basis of `p` principal components.
  - `"pls"`: `p` must be given. Expresses the functional process in a basis of `p` partial least squares.

Although other basis types supported by create.basis are possible, `"bspline"` and `"fourier"` are recommended. Other basis types may cause incompatibilities.

- Composite hypothesis. This argument is only used when `est.method="basis"` and, in this case, it specifies the type of basis used in the basis estimation method of the functional parameter. Again, basis `"bspline"` and `"fourier"` are recommended, as other basis types may cause incompatibilities.

`verbose`	Either to show or not information about computing progress.
`plot.it`	Either to show or not a graph of the observed trajectory, and the bootstrap trajectories under the null composite hypothesis, of the process $R_n(\cdot)$ (see Details). Note that if `plot.it=TRUE`, the function takes more time to run.
`B.plot`	Number of bootstrap trajectories to show in the resulting plot of the test. As the trajectories shown are the first `B.plot` of `B`, `B.plot` must be lower or equal to `B`.
`G`	Number of projections used to compute the trajectories of the process $R_n(\cdot)$ by Monte Carlo.
`...`	Further arguments passed to create.basis.

Details

The Functional Linear Model with scalar response (FLM), is defined as $Y = \big\langle X, \beta \big\rangle + \epsilon$, for a functional process X such that $E[X(t)] = 0$, $E[X(t)\epsilon] = 0$ for all t and for a scalar variable Y such that $E[Y] = 0$. Then, the test assumes that `Y` and `X.fdata` are **centred** and will automatically center them. So, bear in mind that when you apply the test for `Y` and `X.fdata`, actually, you are applying it to `Y-mean(Y)` and `fdata.cen(X.fdata)$Xcen`. The test statistic corresponds to the Cramer-von Mises norm of the *Residual Marked empirical Process based on Projections* $R_n(u,\gamma)$ defined in Garcia-Portugues *et al.* (2014). The expression of this process in a p-truncated basis of the space $L^2[0,T]$ leads to the p-multivariate process $R_{n,p}\big(u,\gamma^{(p)}\big)$, whose Cramer-von Mises norm is computed. The choice of an appropriate p to represent the functional process X, in case that is not provided, is done via the estimation of β for the composite hypothesis. For the simple hypothesis, as no estimation of β is done, the choice of p depends only on the functional process X. As the result of the test may change for different p's, we recommend to use an automatic criterion to select p instead of provide a fixed one. The distribution of the test statistic is approximated by a wild bootstrap resampling on the residuals, using the *golden section bootstrap*. Finally, the graph shown if `plot.it=TRUE` represents the observed trajectory, and the bootstrap trajectories under the null, of the process RMPP *integrated on the projections*:

$$R_n(u) \approx \frac{1}{G}\sum_{g=1}^{G} R_n(u,\gamma_g),$$

where γ_g are simulated as Gaussians processes. This gives a graphical idea of how *distant* is the observed trajectory from the null hypothesis.

Value

An object with class `"htest"` whose underlying structure is a list containing the following components:

- `statistic`: The value of the test statistic.
- `boot.statistics`: A vector of length `B` with the values of the bootstrap test statistics.
- `p.value`: The p-value of the test.
- `method`: The method used.
- `B`: The number of bootstrap replicates used.
- `type.basis`: The type of basis used.
- `beta.est`: The estimated functional parameter β in the composite hypothesis. For the simple hypothesis, the given `beta0.fdata`.
- `p`: The number of basis elements passed or automatically chosen.
- `ord`: The optimal order for PC and PLS given by `fregre.pc.cv` and `fregre.pls.cv`. For other methods, it is set to `1:p`.
- `data.name`: The character string "Y=<X,b>+e".

Note

No NA's are allowed neither in the functional covariate nor in the scalar response.

Author(s)

Eduardo Garcia-Portugues. Please, report bugs and suggestions to
`<edgarcia@est-econ.uc3m.es>`

References

Escanciano, J. C. (2006). A consistent diagnostic test for regression models using projections. Econometric Theory, 22, 1030-1051. doi:10.1017/S0266466606060506

Garcia-Portugues, E., Gonzalez-Manteiga, W. and Febrero-Bande, M. (2014). A goodness–of–fit test for the functional linear model with scalar response. Journal of Computational and Graphical Statistics, 23(3), 761-778. doi:10.1080/10618600.2013.812519

See Also

`Adot`, `PCvM.statistic`, `rwild`, `flm.Ftest`, `dfv.test`, `fregre.pc`, `fregre.pls`, `fregre.basis`, `fregre.pc.cv`, `fregre.pls.cv`, `fregre.basis.cv`, `optim.basis`, create.basis

Examples

```
# Simulated example #
X=rproc2fdata(n=100,t=seq(0,1,l=101),sigma="OU")
beta0=fdata(mdata=cos(2*pi*seq(0,1,l=101))-(seq(0,1,l=101)-0.5)^2+
            rnorm(101,sd=0.05),argvals=seq(0,1,l=101),rangeval=c(0,1))
Y=inprod.fdata(X,beta0)+rnorm(100,sd=0.1)

dev.new(width=21,height=7)
par(mfrow=c(1,3))
plot(X,main="X")
plot(beta0,main="beta0")
plot(density(Y),main="Density of Y",xlab="Y",ylab="Density")
rug(Y)

## Not run:
# Composite hypothesis: do not reject FLM
pcvm.sim=flm.test(X,Y,B=50,B.plot=50,G=100,plot.it=TRUE)
pcvm.sim
flm.test(X,Y,B=5000)

# Estimated beta
dev.new()
plot(pcvm.sim$beta.est)

# Simple hypothesis: do not reject beta=beta0
flm.test(X,Y,beta0.fdata=beta0,B=50,B.plot=50,G=100)
flm.test(X,Y,beta0.fdata=beta0,B=5000)

# AEMET dataset #
data(aemet)
# Remove the 5\
dev.new()
res.FM=depth.FM(aemet$temp,draw=TRUE)
qu=quantile(res.FM$dep,prob=0.05)
l=which(res.FM$dep<=qu)
lines(aemet$temp[l],col=3)
aemet$df$name[l]

# Data without outliers
wind.speed=apply(aemet$wind.speed$data,1,mean)[-l]
temp=aemet$temp[-l]
# Exploratory analysis: accept the FLM
pcvm.aemet=flm.test(temp,wind.speed,est.method="pls",B=100,B.plot=50,G=100)
pcvm.aemet

# Estimated beta
dev.new()
plot(pcvm.aemet$beta.est,lwd=2,col=2)
# B=5000 for more precision on calibration of the test: also accept the FLM
flm.test(temp,wind.speed,est.method="pls",B=5000)

# Simple hypothesis: rejection of beta0=0? Limiting p-value...
dat=rep(0,length(temp$argvals))
flm.test(temp,wind.speed, beta0.fdata=fdata(mdata=dat,argvals=temp$argvals,
                                            rangeval=temp$rangeval),B=100)
flm.test(temp,wind.speed, beta0.fdata=fdata(mdata=dat,argvals=temp$argvals,
                                            rangeval=temp$rangeval),B=5000)

# Tecator dataset #
data(tecator)
names(tecator)
absorp=tecator$absorp.fdata
ind=1:129 # or ind=1:215
```

```
x=absorp[ind,]
y=tecator$y$Fat[ind]
tt=absorp[["argvals"]]

# Exploratory analysis for composite hypothesis with automatic choose of p
pcvm.tecat=flm.test(x,y,B=100,B.plot=50,G=100)
pcvm.tecat

# B=5000 for more precision on calibration of the test: also reject the FLM
flm.test(x,y,B=5000)

# Distribution of the PCvM statistic
plot(density(pcvm.tecat$boot.statistics),lwd=2,xlim=c(0,10),
              main="PCvM distribution", xlab="PCvM*",ylab="Density")
rug(pcvm.tecat$boot.statistics)
abline(v=pcvm.tecat$statistic,col=2,lwd=2)
legend("top",legend=c("PCvM observed"),lwd=2,col=2)

# Simple hypothesis: fixed p
dat=rep(0,length(x$argvals))
flm.test(x,y,beta0.fdata=fdata(mdata=dat,argvals=x$argvals,
                               rangeval=x$rangeval),B=100,p=11)

# Simple hypothesis, automatic choose of p
flm.test(x,y,beta0.fdata=fdata(mdata=dat,argvals=x$argvals,
                               rangeval=x$rangeval),B=100)
flm.test(x,y,beta0.fdata=fdata(mdata=dat,argvals=x$argvals,
                               rangeval=x$rangeval),B=5000)

## End(Not run)
```

`fregre.basis` *Functional Regression with scalar response using basis representation.*

Description

Computes functional regression between functional explanatory variable $X(t)$ and scalar response Y using basis representation.

Usage

```
fregre.basis(
  fdataobj,
  y,
  basis.x = NULL,
  basis.b = NULL,
  lambda = 0,
  Lfdobj = vec2Lfd(c(0, 0), rtt),
  weights = rep(1, n),
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`basis.x`	Basis for functional explanatory data `fdataobj`.
`basis.b`	Basis for functional beta parameter.
`lambda`	A roughness penalty. By default, no penalty `lambda=0`.
`Lfdobj`	See eval.penalty.
`weights`	weights
`...`	Further arguments passed to or from other methods.

Details

$$Y = \langle X, \beta \rangle + \epsilon = \int_T X(t)\beta(t)dt + \epsilon$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on L_2 and ϵ are random errors with mean zero, finite variance σ^2 and $E[X(t)\epsilon] = 0$.

The function uses the basis representation proposed by Ramsay and Silverman (2005) to model the relationship between the scalar response and the functional covariate by basis representation of the observed functional data $X(t) \approx \sum_{k=1}^{k_{n1}} c_k \xi_k(t)$ and the unknown functional parameter $\beta(t) \approx \sum_{k=1}^{k_{n2}} b_k \phi_k(t)$.

The functional linear models estimated by the expression:

$$\hat{y} = \langle X, \hat{\beta} \rangle = C^T \psi(t) \phi^T(t) \hat{b} = \tilde{X} \hat{b}$$

where $\tilde{X}(t) = C^T \psi(t) \phi^T(t)$, and $\hat{b} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$ and so, $\hat{y} = \tilde{X}\hat{b} = \tilde{X}(\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y = Hy$ where H is the hat matrix with degrees of freedom: $df = tr(H)$.

If $\lambda > 0$ then `fregre.basis` incorporates a roughness penalty:
$\hat{y} = \tilde{X}\hat{b} = \tilde{X}(\tilde{X}^T \tilde{X} + \lambda R_0)^{-1} \tilde{X}^T y = H_\lambda y$ where R_0 is the penalty matrix.

This function allows covariates of class `fdata`, `matrix`, `data.frame` or directly covariates of class `fd`. The function also gives default values to arguments `basis.x` and `basis.b` for representation on the basis of functional data $X(t)$ and the functional parameter $\beta(t)$, respectively.

If `basis=NULL` creates the `bspline` basis by create.bspline.basis.
If the functional covariate `fdataobj` is a matrix or data.frame, it creates an object of class "fdata" with default attributes, see `fdata`.
If `basis.x$type=``fourier''` and `basis.b$type=``fourier''`, the basis are orthonormal and the function decreases the number of fourier basis elements on the $min(k_{n1}, k_{n2})$, where $k_{n1}$ and $k_{n2}$ are the number of basis element of `basis.x` and `basis.b` respectively.

Value

Return:

- `call`: The matched call.
- `coefficients`: A named vector of coefficients.
- `residuals`: `y` minus `fitted values`.
- `fitted.values`: Estimated scalar response.
- `beta.est`: Estimated beta parameter of class `fd`.
- `weights`: (only for weighted fits) the specified weights.
- `df.residual`: The residual degrees of freedom.
- `r2`: Coefficient of determination.
- `sr2`: Residual variance.
- `Vp`: Estimated covariance matrix for the parameters.
- `H`: Hat matrix.
- `y`: Response.
- `fdataobj`: Functional explanatory data of class `fdata`.
- `a.est`: Intercept parameter estimated.
- `x.fd`: Centered functional explanatory data of class `fd`.
- `basis.b`: Basis used for beta parameter estimation.
- `lambda.opt`: A roughness penalty.
- `Lfdobj`: Order of a derivative or a linear differential operator.
- `P`: Penalty matrix.
- `lm`: Return `lm` object.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as: `fregre.basis.cv`, `summary.fregre.fd` and `predict.fregre.fd`.
Alternative method: `fregre.pc` and `fregre.np`.

Examples

```
## Not run:
# fregre.basis
data(tecator)
names(tecator)
absorp=tecator$absorp.fdata
ind=1:129
x=absorp[ind,]
y=tecator$y$Fat[ind]
tt=absorp[["argvals"]]
res1=fregre.basis(x,y)
summary(res1)
basis1=create.bspline.basis(rangeval=range(tt),nbasis=19)
basis2=create.bspline.basis(rangeval=range(tt),nbasis=9)
res5=fregre.basis(x,y,basis1,basis2)
summary(res5)
x.d2=fdata.deriv(x,nbasis=19,nderiv=1,method="bspline",class.out="fdata")
res7=fregre.basis(x.d2,y,basis1,basis2)
summary(res7)

## End(Not run)
```

`fregre.basis.cv` *Cross-validation Functional Regression with scalar response using basis representation.*

Description

Computes functional regression between functional explanatory variables and scalar response using basis representation.

Usage

```
fregre.basis.cv(
  fdataobj,
  y,
  basis.x = NULL,
  basis.b = NULL,
  type.basis = NULL,
  lambda = 0,
  Lfdobj = vec2Lfd(c(0, 0), rtt),
  type.CV = GCV.S,
  par.CV = list(trim = 0),
  weights = rep(1, n),
  verbose = FALSE,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`basis.x`	Basis for functional explanatory data `fdataobj`.
`basis.b`	Basis for functional beta parameter.
`type.basis`	A vector of character string which determines type of basis. By default *"bspline"*. It is only used when `basis.x` or `basis.b` are a vector of number of basis considered.
`lambda`	A roughness penalty. By default, no penalty `lambda=0`.
`Lfdobj`	See eval.penalty.
`type.CV`	Type of cross-validation. By default generalized cross-validation `GCV.S` method.
`par.CV`	List of parameters for `type.CV`: `trim`, the alpha of the trimming and `draw`.
`weights`	weights
`verbose`	If `TRUE` information about the procedure is printed. Default is `FALSE`.
`...`	Further arguments passed to or from other methods.

Details

The function `fregre.basis.cv()` uses validation criterion defined by argument `type.CV` to estimate the number of basis elements and/or the penalized parameter (`lambda`) that best predicts the response.

If `basis` = `NULL` creates bspline basis.

If the functional covariate `fdataobj` is in a format `raw data`, such as matrix or data.frame, creates an object of class `fdata` with default attributes, see `fdata`.

If `basis.x` is a vector of number of basis elements and `basis.b=NULL`, the function force the same number of elements in the basis of `x` and `beta`.

If `basis.x$type=``fourier''` and `basis.b$type=``fourier''`, the function decreases the number of fourier basis elements on the $min(k_{n1}, k_{n2})$, where $k_{n1}$ and $k_{n2}$ are the number of basis element of `basis.x` and `basis.b` respectively.

Value

Return:

- `fregre.basis`: Fitted regression object by the best parameters (basis elements for data and beta and lambda penalty).
- `basis.x.opt`: Basis used for functional explanatory data estimation `fdata`.
- `basis.b.opt`: Basis used for functional `beta` parameter estimation.
- `lambda.opt`: `lambda` value that minimizes CV or GCV method.
- `gcv.opt`: Minimum value of CV or GCV method.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ramsay, James O. and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as: fregre.basis, summary.fregre.fd and predict.fregre.fd .
Alternative method: fregre.pc.cv and fregre.np.cv.

Examples

```
## Not run:
data(tecator)
x<-tecator$absorp.fdata[1:129]
y=tecator$y$Fat[1:129]
b1<-c(15,21,31)
b2<-c(7,9)
res1=fregre.basis.cv(x,y,basis.x=b1)
res2=fregre.basis.cv(x,y,basis.x=b1,basis.b=b2)
res1$gcv
res2$gcv
l=2^(-4:10)
res3=fregre.basis.cv(x,y,basis.b=b1,type.basis="fourier",
lambda=l,type.CV=GCV.S,par.CV=list(trim=0.15))
res3$gcv

## End(Not run)
```

---

fregre.basis.fr   *Functional Regression with functional response using basis representation.*

Description

Computes functional regression between functional explanatory variable $X(s)$ and functional response $Y(t)$ using basis representation.

Usage

```
fregre.basis.fr(
  x,
  y,
  basis.s = NULL,
  basis.t = NULL,
  lambda.s = 0,
  lambda.t = 0,
  Lfdobj.s = vec2Lfd(c(0, 0), range.s),
  Lfdobj.t = vec2Lfd(c(0, 0), range.t),
  weights = NULL,
  ...
)
```

Arguments

`x`	Functional explanatory variable.
`y`	Functional response variable.
`basis.s`	Basis related with `s` and it is used in the estimation of $\beta(s,t)$.
`basis.t`	Basis related with `t` and it is used in the estimation of $\beta(s,t)$.
`lambda.s`	A roughness penalty with respect to `s` to be applied in the estimation of $\beta(s,t)$. By default, no penalty `lambda.s=0`.
`lambda.t`	A roughness penalty with respect to `t` to be applied in the estimation of $\beta(s,t)$. By default, no penalty `lambda.t=0`.
`Lfdobj.s`	A linear differential operator object with respect to `s` . See eval.penalty.
`Lfdobj.t`	A linear differential operator object with respect to `t`. See eval.penalty.
`weights`	Weights.
`...`	Further arguments passed to or from other methods.

Details

$$Y(t) = \alpha(t) + \int_T X(s)\beta(s,t)ds + \epsilon(t)$$

where $\alpha(t)$ is the intercept function, $\beta(s,t)$ is the bivariate resgression function and $\epsilon(t)$ are the error term with mean zero.

The function is a wrapped of linmod function proposed by Ramsay and Silverman (2005) to model the relationship between the functional response $Y(t)$ and the functional covariate $X(t)$ by basis representation of both.

The unknown bivariate functional parameter $\beta(s,t)$ can be expressed as a double expansion in terms of K basis function ν_k and L basis functions θ_l,

$$\beta(s,t) = \sum_{k=1}^{K}\sum_{l=1}^{L} b_{kl}\nu_k(s)\theta_l(t) = \nu(s)^\top \boldsymbol{B}\theta(t)$$

Then, the model can be re–written in a matrix version as,

$$Y(t) = \alpha(t) + \int_T X(s)\nu(s)^\top \boldsymbol{B}\theta(t)ds + \epsilon(t) = \alpha(t) + \boldsymbol{X}\boldsymbol{B}\theta(t) + \epsilon(t)$$

where $\boldsymbol{X} = \int X(s)\nu^\top(t)ds$

This function allows objects of class `fdata` or directly covariates of class `fd`. If `x` is a `fdata` class, `basis.s` is also the basis used to represent `x` as `fd` class object. If `y` is a `fdata` class, `basis.t` is also the basis used to represent `y` as `fd` class object. The function also gives default values to arguments `basis.s` and `basis.t` for construct the bifd class object used in the estimation of $\beta(s,t)$. If `basis.s=NULL` or `basis.t=NULL` the function creates a `bspline` basis by create.bspline.basis.

`fregre.basis.fr` incorporates a roughness penalty using an appropiate linear differential operator; `lambda.s`, `Lfdobj.s` for penalization of β's variations with respect to s and `lambda.t`, `Lfdobj.t` for penalization of β's variations with respect to t.

Value

Return:

- `call`: The matched call.
- `a.est`: Intercept parameter estimated.
- `coefficients`: The matrix of the coefficients.
- `beta.est`: A bivariate functional data object of class `bifd` with the estimated parameters of $\beta(s,t)$.
- `fitted.values`: Estimated response.
- `residuals`: `y` minus `fitted values`.
- `y`: Functional response.
- `x`: Functional explanatory data.
- `lambda.s`: A roughness penalty with respect to `s`.
- `lambda.t`: A roughness penalty with respect to `t`.
- `Lfdobj.s`: A linear differential operator with respect to `s`.
- `Lfdobj.t`: A linear differential operator with respect to `t`.
- `weights`: Weights.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

See Also

See Also as: `predict.fregre.fr`. Alternative method: linmod.

Examples

```
## Not run:
rtt<-c(0, 365)
basis.alpha  <- create.constant.basis(rtt)
basisx  <- create.bspline.basis(rtt,11)
basisy  <- create.bspline.basis(rtt,11)
basiss  <- create.bspline.basis(rtt,7)
basist  <- create.bspline.basis(rtt,9)

# fd class
dayfd<-Data2fd(day.5,CanadianWeather$dailyAv,basisx)
tempfd<-dayfd[,1]
log10precfd<-dayfd[,3]
res1 <-  fregre.basis.fr(tempfd, log10precfd,
basis.s=basiss,basis.t=basist)

# fdata class
tt<-1:365
tempfdata<-fdata(t(CanadianWeather$dailyAv[,,1]),tt,rtt)
log10precfdata<-fdata(t(CanadianWeather$dailyAv[,,3]),tt,rtt)
res2<-fregre.basis.fr(tempfdata,log10precfdata,
basis.s=basiss,basis.t=basist)

# penalization
Lfdobjt <- Lfdobjs <- vec2Lfd(c(0,0), rtt)
Lfdobjt <- vec2Lfd(c(0,0), rtt)
lambdat<-lambdas <- 100
res1.pen <- fregre.basis.fr(tempfdata,log10precfdata,basis.s=basiss,
basis.t=basist,lambda.s=lambdas,lambda.t=lambdat,
Lfdobj.s=Lfdobjs,Lfdobj.t=Lfdobjt)

res2.pen <- fregre.basis.fr(tempfd, log10precfd,
basis.s=basiss,basis.t=basist,lambda.s=lambdas,
lambda.t=lambdat,Lfdobj.s=Lfdobjs,Lfdobj.t=Lfdobjt)

plot(log10precfd,col=1)
lines(res1$fitted.values,col=2)
plot(res1$residuals)
plot(res1$beta.est,tt,tt)
plot(res1$beta.est,tt,tt,type="persp",theta=45,phi=30)

## End(Not run)
```

`fregre.bootstrap` *Bootstrap regression*

Description

Estimate the beta parameter by wild or smoothed bootstrap procedure

Usage

```
fregre.bootstrap(
  model,
  nb = 500,
  wild = TRUE,
  type.wild = "golden",
  newX = NULL,
  smo = 0.1,
  smoX = 0.05,
  alpha = 0.95,
  kmax.fix = FALSE,
  draw = TRUE,
  ...
)
```

Arguments

`model`	`fregre.pc`, `fregre.pls` or `fregre.basis` object.
`nb`	Number of bootstrap samples.
`wild`	Naive or smoothed bootstrap depending of the `smo` and `smoX` parameters.
`type.wild`	Type of distribution of V in wild bootstrap procedure, see `rwild`.
`newX`	A `fdata` class containing the values of the model covariates at which predictions are required (only for smoothed bootstrap).
`smo`	If > 0, smoothed bootstrap on the residuals (proportion of response variance).
`smoX`	If > 0, smoothed bootstrap on the explanatory functional variable `fdata` (proportion of variance-covariance matrix of `fdata` object.
`alpha`	Significance level used for graphical option, `draw=TRUE`.
`kmax.fix`	The number of maximum components to consider in each bootstrap iteration. =TRUE, the bootstrap procedure considers the same number of components used in the previous fitted model. =FALSE, the bootstrap procedure estimates the best components in each iteration.
`draw`	=TRUE, plot the bootstrap estimated beta, and (optional) the CI for the predicted response values.
`...`	Further arguments passed to or from other methods.

Details

Estimate the beta parameter by wild or smoothed bootstrap procedure using principal components representation `fregre.pc`, Partial least squares components (PLS) representation `fregre.pls` or basis representation `fregre.basis`.
If a new curves are in `newX` argument the bootstrap method estimates the response using the bootstrap resamples.

If the model exhibits heteroskedasticity, the use of wild bootstrap procedure is recommended (by default).

Value

Return:

- `model`: `fregre.pc`, `fregre.pls` or `fregre.basis` object.
- `beta.boot`: Functional beta estimated by the `nb` bootstrap regressions.
- `norm.boot`: Norm of differences between the `nboot` betas estimated by bootstrap and beta estimated by the regression model.
- `coefs.boot`: Matrix with the bootstrap estimated basis coefficients.
- `kn.boot`: Vector or list of length `nb` with index of the basis, PC or PLS factors selected in each bootstrap regression.
- `y.pred`: Predicted response values using `newX` covariates.
- `y.boot`: Matrix of bootstrap predicted response values using `newX` covariates.
- `newX`: A `fdata` class containing the values of the model covariates at which predictions are required (only for smoothed bootstrap).

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Galeano, P. and Gonzalez-Manteiga, W. (2010). *Measures of influence for the functional linear model with scalar response*. Journal of Multivariate Analysis 101, 327-339.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as: `fregre.pc`, `fregre.pls`, `fregre.basis`, .

Examples

```
## Not run:
data(tecator)
iest<-1:165
x=tecator$absorp.fdata[iest]
y=tecator$y$Fat[iest]
nb<-25  ## Time-consuming
res.pc=fregre.pc(x,y,1:6)
# Fix the compontents used in the each regression
res.boot1=fregre.bootstrap(res.pc,nb=nb,wild=FALSE,kmax.fix=TRUE)
# Select the "best" compontents used in the each regression
res.boot2=fregre.bootstrap(res.pc,nb=nb,wild=FALSE,kmax.fix=FALSE)
res.boot3=fregre.bootstrap(res.pc,nb=nb,wild=FALSE,kmax.fix=10)
## predicted responses and bootstrap confidence interval
newx=tecator$absorp.fdata[-iest]
res.boot4=fregre.bootstrap(res.pc,nb=nb,wild=FALSE,newX=newx,draw=TRUE)

## End(Not run)
```

`fregre.gkam` *Fitting Functional Generalized Kernel Additive Models.*

Description

Computes functional regression between functional explanatory variables $(X^1(t_1), ..., X^q(t_q))$ and scalar response Y using backfitting algorithm.

Usage

```
fregre.gkam(
  formula,
  family = gaussian(),
  data,
  weights = rep(1, nobs),
  par.metric = NULL,
  par.np = NULL,
  offset = NULL,
 control = list(maxit = 100, epsilon = 0.001, trace = FALSE, inverse = "solve"),
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The procedure only considers functional covariates (not implemented for non-functional covariates). The details of model specification are given under `Details`.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions).
`data`	List that containing the variables in the model.
`weights`	weights
`par.metric`	List of arguments by covariate to pass to the `metric` function by covariate.
`par.np`	List of arguments to pass to the `fregre.np.cv` function
`offset`	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
`control`	a list of parameters for controlling the fitting process, by default: `maxit`, `epsilon`, `trace` and `inverse`

`...`	Further arguments passed to or from other methods.
`inverse`	="svd" (by default) or ="solve" method.

Details

The smooth functions $f(.)$ are estimated nonparametrically using a iterative local scoring algorithm by applying Nadaraya-Watson weighted kernel smoothers using `fregre.np.cv` in each step, see Febrero-Bande and Gonzalez-Manteiga (2011) for more details.
Consider the fitted response $\hat{Y} = g^{-1}(H_Q y)$, where H_Q is the weighted hat matrix.
Opsomer and Ruppert (1997) solves a system of equations for fit the unknowns $f(\cdot)$ computing the additive smoother matrix H_k such that $\hat{f}_k(X^k) = H_k Y$ and $H_Q = H_1 +, \cdots, +H_q$. The additive model is fitted as follows:

$$\hat{Y} = g^{-1}\Big(\sum_i^q \hat{f}_i(X_i)\Big)$$

Value

- `result`: List of non-parametric estimation by covariate.
- `fitted.values`: Estimated scalar response.
- `residuals`: `y` minus `fitted values`.
- `effects`: The residual degrees of freedom.
- `alpha`: Hat matrix.
- `family`: Coefficient of determination.
- `linear.predictors`: Residual variance.
- `deviance`: Scalar response.
- `aic`: Functional explanatory data.
- `null.deviance`: Non functional explanatory data.
- `iter`: Distance matrix between curves.
- `w`: Beta coefficient estimated.
- `eqrank`: List that containing the variables in the model.
- `prior.weights`: Asymmetric kernel used.
- `y`: Scalar response.
- `H`: Hat matrix, see Opsomer and Ruppert (1997) for more details.
- `converged`: Conv.

Author(s)

Febrero-Bande, M. and Oviedo de la Fuente, M.

References

Febrero-Bande M. and Gonzalez-Manteiga W. (2012). *Generalized Additive Models for Functional Data*. TEST. Springer-Velag. doi:10.1007/s1174901203080

Opsomer J.D. and Ruppert D.(1997). *Fitting a bivariate additive model by local polynomial regression*.Annals of Statistics, 25, 186-211.

See Also

See Also as: fregre.gsam, fregre.glm and fregre.np.cv

Examples

```
## Not run:
data(tecator)
ab=tecator$absorp.fdata[1:100]
ab2=fdata.deriv(ab,2)
yfat=tecator$y[1:100,"Fat"]

# Example 1: # Changing the argument par.np and family
yfat.cat=ifelse(yfat<15,0,1)
xlist=list("df"=data.frame(yfat.cat),"ab"=ab,"ab2"=ab2)
f2<-yfat.cat~ab+ab2

par.NP<-list("ab"=list(Ker=AKer.norm,type.S="S.NW"),
"ab2"=list(Ker=AKer.norm,type.S="S.NW"))
res2=fregre.gkam(f2,family=binomial(),data=xlist,
par.np=par.NP)
res2

# Example 2: Changing the argument par.metric and family link
par.metric=list("ab"=list(metric=semimetric.deriv,nderiv=2,nbasis=15),
"ab2"=list("metric"=semimetric.basis))
res3=fregre.gkam(f2,family=binomial("probit"),data=xlist,
par.metric=par.metric,control=list(maxit=2,trace=FALSE))
summary(res3)

# Example 3: Gaussian family (by default)
# Only 1 iteration (by default maxit=100)
xlist=list("df"=data.frame(yfat),"ab"=ab,"ab2"=ab2)
f<-yfat~ab+ab2
res=fregre.gkam(f,data=xlist,control=list(maxit=1,trace=FALSE))
res

## End(Not run)
```

`fregre.glm` *Fitting Functional Generalized Linear Models*

Description

Computes functional generalized linear model between functional covariate $X^j(t)$ (and non functional covariate Z^j) and scalar response Y using basis representation.

Usage

```
fregre.glm(
  formula,
  family = gaussian(),
  data,
  basis.x = NULL,
  basis.b = NULL,
  subset = NULL,
  weights = NULL,
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`data`	List that containing the variables in the model.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for $\beta(t)$ parameter estimation.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	weights
`...`	Further arguments passed to or from other methods.

Details

This function is an extension of the linear regression models: `fregre.lm` where the $E[Y|X,Z]$ is related to the linear prediction η via a link function $g(.)$.

$$E[Y|X,Z] = \eta = g^{-1}(\alpha + \sum_{j=1}^{p} \beta_j Z^j + \sum_{k=1}^{q} \frac{1}{\sqrt{T_k}} \int_{T_k} X^k(t)\beta_k(t)dt)$$

where $Z = \left[Z^1, \cdots, Z^p\right]$ are the non functional covariates and $X(t) = \left[X^1(t_1), \cdots, X^q(t_q)\right]$ are the functional ones.

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `glm`.

Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list. `basis.x` is a list of basis for represent each functional covariate. The basis object can be created by the function: `create.pc.basis`, pca.fd `create.pc.basis`, `create.fdata.basis` o create.basis.

basis.b is a list of basis for represent each $\beta(t)$ parameter. If basis.x is a list of functional principal components basis (see create.pc.basis or pca.fd) the argument basis.b is ignored.

represent beta lower than the number of basis used to represent the functional data.

Value

Return glm object plus:

- basis.x: Basis used for fdata or fd covariates.
- basis.b: Basis used for beta parameter estimation.
- beta.l: List of estimated beta parameter of functional covariates.
- data: List that contains the variables in the model.
- formula: Formula.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard glm procedure.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

McCullagh and Nelder (1989), *Generalized Linear Models* 2nd ed. Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*, New York: Springer.

See Also

See Also as: predict.fregre.glm and summary.glm.
Alternative method if family=*gaussian*: fregre.lm.

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
y=tecator$y$Fat
tt=x[["argvals"]]
dataf=as.data.frame(tecator$y)
nbasis.x=11
nbasis.b=7
basis1=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.x)
basis2=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.b)
f=Fat~Protein+x
basis.x=list("x"=basis1)
basis.b=list("x"=basis2)
ldata=list("df"=dataf,"x"=x)
res=fregre.glm(f,family=gaussian(),data=ldata,basis.x=basis.x,
basis.b=basis.b)
summary(res)

## End(Not run)
```

`fregre.glm.vs` *Variable Selection using Functional Linear Models*

Description

Computes functional GLM model between functional covariates $(X^1(t_1), \cdots, X^q(t_q))$ and non functional covariates $(Z^1, ..., Z^p)$ with a scalar response Y.

Usage

```
fregre.glm.vs(
  data = list(),
  y,
  include = "all",
  exclude = "none",
  family = gaussian(),
  weights = NULL,
  basis.x = NULL,
  numbasis.opt = FALSE,
  dcor.min = 0.1,
  alpha = 0.05,
  par.model,
  xydist,
  trace = FALSE
)
```

Arguments

`data`	List that containing the variables in the model. "df" element is a data.frame containing the response and scalar covariates (numeric and factors variables are allowed). Functional covariates of class `fdata` or `fd` are included as named components in the `data` list.
`y`	Caracter string with the name of the scalar response variable.
`include`	vector with the name of variables to use. By default `"all"`, all variables are used.
`exclude`	vector with the name of variables to not use. By default `"none"`, no variable is deleted.

family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
weights	weights
basis.x	Basis parameter options • list (recomended) List of basis for functional covariates, see same argument in fregre.glm. By default, the function uses a basis of 3 PC to represent each functional covariate. • vector (by default) Vector with two parameters: 1. Type of basis. By default basis.x[1]="pc", principal component basis is used for each functional covariate included in the model. Other options "pls" and "bspline". 2. Maximum number of basis elements numbasis to be used. By default, basis.x[2]=3.
numbasis.opt	Logical, if FALSE by default, for each functional covariate included in the model, the function uses all basis elements. Otherwise, the function selects the significant coefficients.
dcor.min	Threshold for a variable to be entered into the model. X is discarded if the distance correlation $R(X, e) < dcor.min$ (e is the residual of previous steps).
alpha	Alpha value for testing the independence among covariate X and residual e in previous steps. By default is 0.05.
par.model	Model parameters.
xydist	List with the inner distance matrices of each variable (all potential covariates and the response).
trace	Interactive Tracing and Debugging of Call.

Details

This function is an extension of the functional generalized spectral additive regression models: fregre.glm where the $E[Y|X, Z]$ is related to the linear prediction η via a link function $g(\cdot)$.

$$E[Y|X,Z] = \eta = g^{-1}(\alpha + \sum_{j=1}^{p} \beta_j Z^j + \sum_{k=1}^{q} \frac{1}{\sqrt{T_k}} \int_{T_k} X^k(t)\beta_k(t)dt)$$

where $Z = \left[Z^1, \cdots, Z^p\right]$ are the non functional covariates and $X(t) = \left[X^1(t_1), \cdots, X^q(t_q)\right]$ are the functional ones.

Value

Return an object corresponding to the estimated additive mdoel using the selected variables (ame output as thefregre.glm function) and the following elements:

- gof, the goodness of fit for each step of VS algorithm.
- i.predictor, vector with 1 if the variable is selected, 0 otherwise.
- ipredictor, vector with the name of selected variables (in order of selection)
- dcor, the value of distance correlation for each potential covariate and the residual of the model in each step.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard `glm` procedure.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo-de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Gonz\'alez-Manteiga, W. and Oviedo de la Fuente, M. Variable selection in functional additive regression models, (2018). Computational Statistics, 1-19. DOI: doi:10.1007/s0018001808445

See Also

See Also as: `predict.fregre.glm` and `summary.glm`. Alternative methods: `fregre.glm`, `fregre.glm` and `fregre.gsam.vs`.

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
x1 <- fdata.deriv(x)
x2 <- fdata.deriv(x,nderiv=2)
y=tecator$y$Fat
xcat0 <- cut(rnorm(length(y)),4)
xcat1 <- cut(tecator$y$Protein,4)
xcat2 <- cut(tecator$y$Water,4)
ind <- 1:165
dat <- data.frame("Fat"=y, x1$data, xcat1, xcat2)
ldat <- ldata("df"=dat[ind,],"x"=x[ind,],"x1"=x1[ind,],"x2"=x2[ind,])
# 3 functionals (x,x1,x2), 3 factors (xcat0, xcat1, xcat2)
# and 100 scalars (impact poitns of x1)

# Time consuming
res.glm0 <- fregre.glm.vs(data=ldat,y="Fat",numbasis.opt=T) # All the covariates
summary(res.glm0)
res.glm0$ipredictors
res.glm0$i.predictor

res.glm1 <- fregre.glm.vs(data=ldat,y="Fat") # All the covariates
summary(res.glm1)
res.glm1$ipredictors
covar <- c("xcat0","xcat1","xcat2","x","x1","x2")
res.glm2 <- fregre.glm.vs(data=ldat, y="Fat", include=covar)
summary(res.glm2)
res.glm2$ipredictors
res.glm2$i.predictor
res.glm3 <- fregre.glm.vs(data=ldat,y="Fat",
                            basis.x=c("type.basis"="pc","numbasis"=2))
summary(res.glm3)
res.glm3$ipredictors

res.glm4 <- fregre.glm.vs(data=ldat,y="Fat",include=covar,
basis.x=c("type.basis"="pc","numbasis"=5),numbasis.opt=T)
summary(res.glm4)
res.glm4$ipredictors
lpc <- list("x"=create.pc.basis(ldat$x,1:4)
           ,"x1"=create.pc.basis(ldat$x1,1:3)
           ,"x2"=create.pc.basis(ldat$x2,1:4))
res.glm5 <- fregre.glm.vs(data=ldat,y="Fat",basis.x=lpc)
summary(res.glm5)
res.glm5 <- fregre.glm.vs(data=ldat,y="Fat",basis.x=lpc,numbasis.opt=T)
summary(res.glm5)
bsp <- create.fourier.basis(ldat$x$rangeval,7)
lbsp <- list("x"=bsp,"x1"=bsp,"x2"=bsp)
res.glm6 <- fregre.glm.vs(data=ldat,y="Fat",basis.x=lbsp)
summary(res.glm6)
# Prediction like fregre.glm()
newldat <- ldata("df"=dat[-ind,],"x"=x[-ind,],"x1"=x1[-ind,],
                "x2"=x2[-ind,])
pred.glm1 <- predict(res.glm1,newldat)
pred.glm2 <- predict(res.glm2,newldat)
pred.glm3 <- predict(res.glm3,newldat)
pred.glm4 <- predict(res.glm4,newldat)
pred.glm5 <- predict(res.glm5,newldat)
pred.glm6 <- predict(res.glm6,newldat)
plot(dat[-ind,"Fat"],pred.glm1)
points(dat[-ind,"Fat"],pred.glm2,col=2)
points(dat[-ind,"Fat"],pred.glm3,col=3)
points(dat[-ind,"Fat"],pred.glm4,col=4)
points(dat[-ind,"Fat"],pred.glm5,col=5)
points(dat[-ind,"Fat"],pred.glm6,col=6)
pred2meas(newldat$df$Fat,pred.glm1)
pred2meas(newldat$df$Fat,pred.glm2)
pred2meas(newldat$df$Fat,pred.glm3)
pred2meas(newldat$df$Fat,pred.glm4)
pred2meas(newldat$df$Fat,pred.glm5)
pred2meas(newldat$df$Fat,pred.glm6)

## End(Not run)
```

---

fregre.gls *Fit Functional Linear Model Using Generalized Least Squares*

Description

This function fits a functional linear model using generalized least squares. The errors are allowed to be correlated and/or have unequal variances.

Usage

```
fregre.gls(
  formula,
  data,
  correlation = NULL,
  basis.x = NULL,
  basis.b = NULL,
  rn,
  lambda,
  weights = NULL,
  subset,
  method = c("REML", "ML"),
  control = list(),
  verbose = FALSE,
  criteria = "GCCV1",
  ...
)
```

Arguments

`formula`	a two-sided linear formula object describing the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right.
`data`	an optional data frame containing the variables named in `model`, `correlation`, `weights`, and `subset`. By default the variables are taken from the environment from which `gls` is called.
`correlation`	an optional corStruct object describing the within-group correlation structure. See the documentation of corClasses for a description of the available `corStruct` classes. If a grouping variable is to be used, it must be specified in the `form` argument to the `corStruct` constructor. Defaults to `NULL`, corresponding to uncorrelated errors.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for $\beta(t)$ parameter estimation.
`rn`	List of Ridge parameter.
`lambda`	List of Roughness penalty parameter.
`weights`	an optional varFunc object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to varFixed, corresponding to fixed variance weights. See the documentation on varClasses for a description of the available varFunc classes. Defaults to `NULL`, corresponding to homoscedastic errors.
`subset`	an optional expression indicating which subset of the rows of `data` should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
`method`	a character string. If `"REML"` the model is fit by maximizing the restricted log-likelihood. If `"ML"` the log-likelihood is maximized. Defaults to `"REML"`.

`control`	a list of control values for the estimation algorithm to replace the default values returned by the function glsControl. Defaults to an empty list.
`verbose`	an optional logical value. If `TRUE` information on the evolution of the iterative algorithm is printed. Default is `FALSE`.
`criteria`	GCCV criteria, see `GCCV.S`.
`...`	some methods for this generic require additional arguments. None are used in this method.

Value

An object of class `"gls"` representing the functional linear model fit. Generic functions such as `print`, `plot`, and `summary` have methods to show the results of the fit.
See glsObject for the components of the fit. The functions `resid`, `coef`, and `fitted` can be used to extract some of its components.
Besides, the class(z) is "gls", "lm", and "fregre.lm" with the following objects:

- `sr2`: Residual variance.
- `Vp`: Estimated covariance matrix for the parameters.
- `lambda`: A roughness penalty.
- `basis.x`: Basis used for `fdata` or `fd` covariates.
- `basis.b`: Basis used for beta parameter estimation.
- `beta.l`: List of estimated beta parameter of functional covariates.
- `data`: List containing the variables in the model.
- `formula`: Formula used in the adjusted model.
- `formula.ini`: Formula in call.
- `W`: Inverse of covariance matrix.
- `correlation`: See `glsObject` for the components of the fit.

References

Oviedo de la Fuente, M., Febrero-Bande, M., Pilar Munoz, and Dominguez, A. (2018). Predicting seasonal influenza transmission using functional regression models with temporal dependence. PloS one, 13(4), e0194250. doi:10.1371/journal.pone.0194250

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
x.d2<-fdata.deriv(x,nderiv=)
tt<-x[["argvals"]]
dataf=as.data.frame(tecator$y)

# plot the response
plot(ts(tecator$y$Fat))
```

```
nbasis.x=11;nbasis.b=7
basis1=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.x)
basis2=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.b)
basis.x=list("x.d2"=basis1)
basis.b=list("x.d2"=basis2)
ldata=list("df"=dataf,"x.d2"=x.d2)
res.gls=fregre.gls(Fat~x.d2,data=ldata, correlation=corAR1(),
                   basis.x=basis.x,basis.b=basis.b)
summary(res.gls)

## End(Not run)
```

`fregre.gsam` *Fitting Functional Generalized Spectral Additive Models*

Description

Computes a functional GAM model between a functional covariate $(X^1(t_1), \dots, X^q(t_q))$ and a non-functional covariate $(Z^1, ..., Z^p)$ with a scalar response Y.

This function extends functional generalized linear regression models (fregre.glm) where $E[Y|X, Z]$ is related to the linear predictor η via a link function $g(\cdot)$ with integrated smoothness estimation by the smooth functions $f(\cdot)$.

$$E[Y|X, Z] = \eta = g^{-1}\left(\alpha + \sum_{i=1}^{p} f_i(Z^i) + \sum_{k=1}^{q}\sum_{j=1}^{k_q} f_j^k(\xi_j^k)\right)$$

where ξ_j^k is the coefficient of the basis function expansion of X^k; in PCA analysis, ξ_j^k is the score of the j-functional PC of X^k.

Usage

```
fregre.gsam(
  formula,
  family = gaussian(),
  data = list(),
  weights = NULL,
  basis.x = NULL,
  basis.b = NULL,
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.

`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`data`	List that containing the variables in the model.
`weights`	weights
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for functional beta parameter estimation.
`...`	Further arguments passed to or from other methods.

Details

The smooth functions $f(\cdot)$ can be added to the right-hand side of the formula to specify that the linear predictor depends on smooth functions of predictors using smooth terms `s` and `te` as in `gam` (or linear functionals of these as $Z\beta$ and $\langle X(t), \beta \rangle$ in `fregre.glm`).

The first item in the `data` list is called *"df"* and is a data frame with the response and non-functional explanatory variables, as in `gam`.

Functional covariates of class `fdata` or `fd` are introduced in the following items of the `data` list.
`basis.x` is a list of basis functions for representing each functional covariate. The basis object can be created using functions such as `create.pc.basis`, `pca.fd`, `create.fdata.basis`, or `create.basis`.
`basis.b` is a list of basis functions for representing each functional beta parameter. If `basis.x` is a list of functional principal components basis functions (see `create.pc.basis` or `pca.fd`), the argument `basis.b` is ignored.

Value

Return `gam` object plus:

- `basis.x`: Basis used for `fdata` or `fd` covariates.
- `basis.b`: Basis used for beta parameter estimation.
- `data`: List containing the variables in the model.
- `formula`: Formula used in the model.
- `y.pred`: Predicted response by cross-validation.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard `glm` procedure.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Muller HG and Stadtmuller U. (2005). *Generalized functional linear models.* Ann. Statist.33 774-805.

Wood (2001) *mgcv:GAMs and Generalized Ridge Regression for R*. R News 1(2):20-25.

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*, New York: Springer.

See Also

See Also as: `predict.fregre.gsam` and summary.gam.
Alternative methods: `fregre.glm` and `fregre.gkam`.

Examples

```
## Not run:
data(tecator)
x <- tecator$absorp.fdata
x.d1 <- fdata.deriv(x)
tt <- x[["argvals"]]
dataf <- as.data.frame(tecator$y)
nbasis.x <- 11
nbasis.b <- 5
basis1 <- create.bspline.basis(rangeval=range(tt),nbasis=nbasis.x)
basis2 <- create.bspline.basis(rangeval=range(tt),nbasis=nbasis.b)
f <- Fat ~ s(Protein) + s(x)
basis.x <- list("x"=basis1,"x.d1"=basis1)
basis.b <- list("x"=basis2,"x.d1"=basis2)
ldat <- ldata("df" = dataf, "x" = x , "x.d1" = x.d1)
res <- fregre.gsam(Fat ~ Water + s(Protein) + x + s(x.d1), ldat,
                   family = gaussian(),  basis.x = basis.x,
                   basis.b = basis.b)
summary(res)
pred <- predict(res,ldat)
plot(pred-res$fitted)
pred2 <- predict.gam(res,res$XX)
plot(pred2-res$fitted)
plot(pred2-pred)
res2 <- fregre.gsam(Fat ~ te(Protein, k = 3) + x, data =  ldat,
                     family=gaussian())
summary(res2)

##  dropind basis pc
basis.pc0 <- create.pc.basis(x,c(2,4,7))
basis.pc1 <- create.pc.basis(x.d1,c(1:3))
basis.x <- list("x"=basis.pc0,"x.d1"=basis.pc1)
ldata <- ldata("df"=dataf,"x"=x,"x.d1"=x.d1)
res.pc <- fregre.gsam(f,data=ldata,family=gaussian(),
          basis.x=basis.x,basis.b=basis.b)
summary(res.pc)
```

```
##  Binomial family
ldat$df$FatCat <- factor(ifelse(tecator$y$Fat > 20, 1, 0))
res.bin <- fregre.gsam(FatCat ~ Protein + s(x),ldat,family=binomial())
summary(res.bin)
table(ldat$df$FatCat, ifelse(res.bin$fitted.values < 0.5,0,1))

## End(Not run)
```

`fregre.gsam.vs` *Variable Selection using Functional Additive Models*

Description

Computes functional GAM model between functional covariates $(X^1(t_1), \cdots, X^q(t_q))$ and non functional covariates $(Z^1, ..., Z^p)$ with a scalar response Y.

Usage

```
fregre.gsam.vs(
  data = list(),
  y,
  include = "all",
  exclude = "none",
  family = gaussian(),
  weights = NULL,
  basis.x = NULL,
  numbasis.opt = FALSE,
  kbs,
  dcor.min = 0.1,
  alpha = 0.05,
  par.model,
  xydist,
  trace = FALSE
)
```

Arguments

`data`	List that containing the variables in the model. "df" element is a data.frame containing the response and scalar covariates (numeric and factors variables are allowed). Functional covariates of class `fdata` or `fd` are included as named components in the `data` list.
`y`	Caracter string with the name of the scalar response variable.
`include`	vector with the name of variables to use. By default `"all"`, all variables are used.
`exclude`	vector with the name of variables to not use. By default `"none"`, no variable is deleted.

`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`weights`	weights
`basis.x`	Basis parameter options • `list` (recomended) List of basis for functional covariates, see same argument in `fregre.glm`. By default, the function uses a basis of 3 PC to represent each functional covariate. • `vector` (by default) Vector with two parameters: 1. Type of basis. By default `basis.x[1]="pc"`, principal component basis is used for each functional covariate included in the model. Other options `"pls"` and `"bspline"`. 2. Maximum number of basis elements `numbasis` to be used. By default, `basis.x[2]=3`.
`numbasis.opt`	Logical, if `FALSE` by default, for each functional covariate included in the model, the function uses all basis elements. Otherwise, the function selects the significant coefficients.
`kbs`	The dimension of the basis used to represent the smooth term. The default depends on the number of variables that the smooth is a function of.
`dcor.min`	Threshold for a variable to be entered into the model. X is discarded if the distance correlation $R(X, e) < dcor.min$ (e is the residual of previous steps).
`alpha`	Alpha value for testing the independence among covariate X and residual e in previous steps. By default is `0.05`.
`par.model`	Model parameters.
`xydist`	List with the inner distance matrices of each variable (all potential covariates and the response).
`trace`	Interactive Tracing and Debugging of Call.

Details

This function is an extension of the functional generalized spectral additive regression models: `fregre.gsam` where the $E[Y|X, Z]$ is related to the linear prediction η via a link function $g(\cdot)$ with integrated smoothness estimation by the smooth functions $f(\cdot)$.

$$E[Y|X, Z] = \eta = g^{-1}(\alpha + \sum_{i=1}^{p} f_i(Z^i) + \sum_{k=1}^{q} \sum_{j=1}^{k_q} f_j^k(\xi_j^k))$$

where ξ_j^k is the coefficient of the basis function expansion of X^k, (in PCA analysis, ξ_j^k is the score of the j-functional PC of X^k).

The smooth functions $f(\cdot)$ can be added to the right-hand side of the formula to specify that the linear predictor depends on smooth functions of predictors using smooth terms s and te as in gam (or linear functionals of these as $Z\beta$ and $\langle X(t), \beta(t)\rangle$ in `fregre.glm`).

Value

Return an object corresponding to the estimated additive mdoel using the selected variables (ame output as the`fregre.gsam` function) and the following elements:

- `gof`: the goodness of fit for each step of VS algorithm.
- `i.predictor`: `vector` with 1 if the variable is selected, 0 otherwise.
- `ipredictor`: `vector` with the name of selected variables (in order of selection)
- `dcor`: the value of distance correlation for each potential covariate and the residual of the model in each step.

Note

If the formula only contains a non functional explanatory variables (multivariate covariates), the function compute a standard gam procedure.

Author(s)

Manuel Feb-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Gonz\'alez-Manteiga, W. and Oviedo de la Fuente, M. Variable selection in functional additive regression models, (2018). Computational Statistics, 1-19. DOI: doi:10.1007/s0018001808445

See Also

See Also as: `predict.fregre.gsam` and summary.gam. Alternative methods: `fregre.glm`, `fregre.gsam` and `fregre.gkam`.

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
x1 <- fdata.deriv(x)
x2 <- fdata.deriv(x,nderiv=2)
y=tecator$y$Fat
xcat0 <- cut(rnorm(length(y)),4)
xcat1 <- cut(tecator$y$Protein,4)
xcat2 <- cut(tecator$y$Water,4)
ind <- 1:165
dat <- data.frame("Fat"=y, x1$data, xcat1, xcat2)
ldat <- ldata("df"=dat[ind,],"x"=x[ind,],"x1"=x1[ind,],"x2"=x2[ind,])
# 3 functionals (x,x1,x2), 3 factors (xcat0, xcat1, xcat2)
# and 100 scalars (impact poitns of x1)

# Time consuming
res.gam0 <- fregre.gsam.vs(data=ldat,y="Fat"
            ,exclude="x2",numbasis.opt=T) # All the covariates
```

```
summary(res.gam0)
res.gam0$ipredictors

res.gam1 <- fregre.gsam.vs(data=ldat,y="Fat") # All the covariates
summary(res.gam1)
res.gam1$ipredictors

covar <- c("xcat0","xcat1","xcat2","x","x1","x2")
res.gam2 <- fregre.gsam.vs(data=ldat, y="Fat", include=covar)
summary(res.gam2)
res.gam2$ipredictors
res.gam2$i.predictor

res.gam3 <- fregre.gsam.vs(data=ldat,y="Fat",
            basis.x=c("type.basis"="pc","numbasis"=10))
summary(res.gam3)
res.gam3$ipredictors

res.gam4 <- fregre.gsam.vs(data=ldat,y="Fat",include=c("x","x1","x2"),
basis.x=c("type.basis"="pc","numbasis"=5),numbasis.opt=T)
summary(res.gam4)
res.gam4$ipredictors
lpc <- list("x"=create.pc.basis(ldat$x,1:4)
           ,"x1"=create.pc.basis(ldat$x1,1:3)
           ,"x2"=create.pc.basis(ldat$x2,1:12))
res.gam5 <- fregre.gsam.vs(data=ldat,y="Fat",basis.x=lpc)
summary(res.gam5)
res.gam6 <- fregre.gsam.vs(data=ldat,y="Fat",basis.x=lpc,numbasis.opt=T)
summary(res.gam6)
bsp <- create.fourier.basis(ldat$x$rangeval,7)
lbsp <- list("x"=bsp,"x1"=bsp,"x2"=bsp)
res.gam7 <- fregre.gsam.vs(data=ldat,y="Fat",basis.x=lbsp,kbs=4)
summary(res.gam7)
# Prediction like fregre.gsam()
newldat <- ldata("df"=dat[-ind,],"x"=x[-ind,],"x1"=x1[-ind,],
                "x2"=x2[-ind,])
pred.gam1 <- predict(res.gam1,newldat)
pred.gam2 <- predict(res.gam2,newldat)
pred.gam3 <- predict(res.gam3,newldat)
pred.gam4 <- predict(res.gam4,newldat)
pred.gam5 <- predict(res.gam5,newldat)
pred.gam6 <- predict(res.gam6,newldat)
pred.gam7 <- predict(res.gam7,newldat)
plot(dat[-ind,"Fat"],pred.gam1)
points(dat[-ind,"Fat"],pred.gam2,col=2)
points(dat[-ind,"Fat"],pred.gam3,col=3)
points(dat[-ind,"Fat"],pred.gam4,col=4)
points(dat[-ind,"Fat"],pred.gam5,col=5)
points(dat[-ind,"Fat"],pred.gam6,col=6)
points(dat[-ind,"Fat"],pred.gam7,col=7)
pred2meas(newldat$df$Fat,pred.gam1)
pred2meas(newldat$df$Fat,pred.gam2)
pred2meas(newldat$df$Fat,pred.gam3)
```

```
pred2meas(newldat$df$Fat,pred.gam4)
pred2meas(newldat$df$Fat,pred.gam5)
pred2meas(newldat$df$Fat,pred.gam6)
pred2meas(newldat$df$Fat,pred.gam7)

## End(Not run)
```

fregre.igls *Fit of Functional Generalized Least Squares Model Iteratively*

Description

This function fits iteratively a functional linear model using generalized least squares. The errors are allowed to be correlated and/or have unequal variances.

1. Begin with a preliminary estimation of $\hat{\theta} = \theta_0$ (for instance, $\theta_0 = 0$). Compute $\hat{W}$.
2. Estimate $b_\Sigma = (Z'\hat{W}Z)^{-1}Z'\hat{W}y$
3. Based on the residuals, $\hat{e} = (y - Zb_\Sigma)$, update $\hat{\theta} = \rho\left(\hat{e}\right)$ where ρ depends on the dependence structure chosen.
4. Repeats steps 2 and 3 until convergence (small changes in b_Σ and/or $\hat{\theta}$).

Usage

```
fregre.igls(
  formula,
  data,
  basis.x = NULL,
  basis.b = NULL,
  correlation,
  maxit = 100,
  rn,
  lambda,
  weights = rep(1, n),
  control,
  ...
)
```

Arguments

`formula`	A two-sided linear formula object describing the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right.
`data`	An optional data frame containing the variables named in `model`, `correlation`, `weights`, and `subset`. By default the variables are taken from the environment from which `gls` is called.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for $\beta(t)$ parameter estimation.

`correlation`	List describing the correlation structure. Defaults to `NULL`, corresponding to uncorrelated errors. See the following internal functions for a description and a code example in script file. • `corUnstruc(x)`, fit an unstrutured correlation. • `cor.AR(x, order.max = 8, p=1, method = "lm")` fit an Autoregressive Models to Time Series using `ar` function. • `cor.ARMA(x, p, d = 0, q = 0, method = "lm", order.max = 1)` Fit an ARIMA model to a univariate time series using `arima` function. • `corExpo(xy,range, method = "euclidean",p=2)` Fit an exponential correlation structure.
`maxit`	Number of maximum of interactions.
`rn`	List of Ridge parameter.
`lambda`	List of Roughness penalty parameter.
`weights`	weights
`control`	Control parameters.
`...`	Further arguments passed to or from other methods.

Value

An object of class `fregre.igls` representing the functional linear model fit with temporal dependence errors. Beside, the class(z) is similar to "fregre.lm" plus the following objects:

- `corStruct`: Fitted AR or ARIMA model.

References

Oviedo de la Fuente, M., Febrero-Bande, M., Pilar Munoz, and Dominguez, A. (2018). Predicting seasonal influenza transmission using functional regression models with temporal dependence. PloS one, 13(4), e0194250. doi:10.1371/journal.pone.0194250

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
x.d2<-fdata.deriv(x,nderiv=)
tt<-x[["argvals"]]
dataf=as.data.frame(tecator$y)
# plot the response
plot(ts(tecator$y$Fat))
ldata=list("df"=dataf,"x.d2"=x.d2)
res.gls=fregre.igls(Fat~x.d2,data=ldata,
correlation=list("cor.ARMA"=list()),
control=list("p"=1))
res.gls
res.gls$corStruct

## End(Not run)
```

`fregre.lm` *Fitting Functional Linear Models*

Description

Computes functional regression between functional (and non functional) explanatory variables and scalar response using basis representation.

Usage

```
fregre.lm(
  formula,
  data,
  basis.x = NULL,
  basis.b = NULL,
  lambda = NULL,
  P = NULL,
  weights = rep(1, n),
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`data`	List that containing the variables in the model. Functional covariates are recommended to be of class fdata. Objects of class "fd" can be used at the user's own risk.
`basis.x`	List of basis for functional explanatory data estimation.
`basis.b`	List of basis for functional beta parameter estimation.
`lambda`	List, indexed by the names of the functional covariates, which contains the Roughness penalty parameter.
`P`	List, indexed by the names of the functional covariates, which contains the parameters for the creation of the penalty matrix.
`weights`	weights
`...`	Further arguments passed to or from other methods.

Details

This section is presented as an extension of the linear regression models: `fregre.pc`, `fregre.pls` and `fregre.basis`. Now, the scalar response Y is estimated by more than one functional covariate $X^j(t)$ and also more than one non functional covariate Z^j. The regression model is given by:

$$E[Y|X,Z] = \alpha + \sum_{j=1}^{p} \beta_j Z^j + \sum_{k=1}^{q} \frac{1}{\sqrt{T_k}} \int_{T_k} X^k(t)\beta_k(t)dt$$

where $Z = \left[Z^1, \cdots, Z^p\right]$ are the non functional covariates, $X(t) = \left[X^1(t_1), \cdots, X^q(t_q)\right]$ are the functional ones and ϵ are random errors with mean zero , finite variance σ^2 and $E[X(t)\epsilon] = 0$.

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `lm`. Functional covariates of class `fdata` or `fd` are introduced in the following items in the `data` list.

`basis.x` is a list of basis for represent each functional covariate. The basis object can be created by the function: `create.pc.basis`, pca.fd `create.pc.basis`, `create.fdata.basis` or create.basis. `basis.b` is a list of basis for represent each functional β_k parameter. If `basis.x` is a list of functional principal components basis (see `create.pc.basis` or pca.fd) the argument `basis.b` *(is unnecessary and)* is ignored.

Penalty options are under development, not guaranteed to work properly. The user can penalty the basis elements by: (i) `lambda` is a list of rough penalty values of each functional covariate, see `P.penalty` for more details.

Value

Return `lm` object plus:

- `sr2`: Residual variance.
- `Vp`: Estimated covariance matrix for the parameters.
- `lambda`: A roughness penalty.
- `basis.x`: Basis used for `fdata` or `fd` covariates.
- `basis.b`: Basis used for beta parameter estimation.
- `beta.l`: List of estimated beta parameter of functional covariates.
- `data`: List containing the variables in the model.
- `formula`: Formula used in the model.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@usc.es`>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as: `predict.fregre.lm` and `summary.lm`.
Alternative method: `fregre.glm`.

Examples

```
## Not run:
data(tecator)
x <- tecator$absorp.fdata
y <- tecator$y$Fat
tt <- x[["argvals"]]
dataf <- as.data.frame(tecator$y)

nbasis.x <- 11
nbasis.b <- 5
basis1 <- create.bspline.basis(rangeval=range(tt),nbasis=nbasis.x)
basis2 <- create.bspline.basis(rangeval=range(tt),nbasis=nbasis.b)
basis.x <- list("x"=basis1)
basis.b <- list("x"=basis2)
f <- Fat ~ Protein + x
ldat <- ldata("df"=dataf,"x"=x)
res <- fregre.lm(f,ldat,  basis.b=basis.b)
summary(res)
f2 <- Fat ~ Protein + xd +xd2
xd <- fdata.deriv(x,nderiv=1,class.out='fdata', nbasis=nbasis.x)
xd2 <- fdata.deriv(x,nderiv=2,class.out='fdata', nbasis=nbasis.x)
ldat2 <- list("df"=dataf,"xd"=xd,"x"=x,"xd2"=xd2)
basis.x2 <- NULL#list("xd"=basis1)
basis.b2 <- NULL#list("xd"=basis2)
basis.b2 <- list("xd"=basis2,"xd2"=basis2,"x"=basis2)
res2 <- fregre.lm(f2, ldat2,basis.b=basis.b2)
summary(res2)
par(mfrow=c(2,1))
plot(res$beta.l$x,main="functional beta estimation")
plot(res2$beta.l$xd,col=2)

## End(Not run)
```

`fregre.np` *Functional regression with scalar response using non-parametric kernel estimation*

Description

Computes functional regression between functional explanatory variables and scalar response using kernel estimation.

Usage

```
fregre.np(
  fdataobj,
  y,
  h = NULL,
  Ker = AKer.norm,
  metric = metric.lp,
  type.S = S.NW,
  par.S = list(w = 1),
  ...
)
```

Arguments

fdataobj	fdata class object.
y	Scalar response with length n.
h	Bandwidth, h>0. Default argument values are provided as the 5%–quantile of the distance between fdataobj curves, see h.default.
Ker	Type of asymmetric kernel used, by default asymmetric normal kernel.
metric	Metric function, by default metric.lp.
type.S	Type of smothing matrix S. By default S is calculated by Nadaraya-Watson kernel estimator (S.NW).
par.S	List of parameters for type.S: w, the weights.
...	Arguments to be passed for metric.lp o other metric function.

Details

The non-parametric functional regression model can be written as follows

$$y_i = r(X_i) + \epsilon_i$$

where the unknown smooth real function r is estimated using kernel estimation by means of

$$\hat{r}(X) = \frac{\sum_{i=1}^{n} K(h^{-1}d(X, X_i))y_i}{\sum_{i=1}^{n} K(h^{-1}d(X, X_i))}$$

where K is an kernel function (see Ker argument), h is the smoothing parameter and d is a metric or a semi-metric (see metric argument).

The distance between curves is calculated using the metric.lp although any other semimetric could be used (see semimetric.basis or semimetric.NPFDA functions). The kernel is applied to a metric or semi-metrics that provides non-negative values, so it is common to use asymmetric kernels. Different asymmetric kernels can be used, see Kernel.asymmetric.

Value

Return:

- call: The matched call.
- fitted.values: Estimated scalar response.
- H: Hat matrix.
- residuals: y minus fitted values.
- df.residual: The residual degrees of freedom.

- `r2`: Coefficient of determination.
- `sr2`: Residual variance.
- `y`: Response.
- `fdataobj`: Functional explanatory data.
- `mdist`: Distance matrix between `x` and `newx`.
- `Ker`: Asymmetric kernel used.
- `h.opt`: Smoothing parameter or bandwidth.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

See Also

See Also as: `fregre.np.cv`, `summary.fregre.fd` and `predict.fregre.fd` .
Alternative method: `fregre.basis`,cand `fregre.pc`.

Examples

```
## Not run:
data(tecator)
absorp=tecator$absorp.fdata
ind=1:129
x=absorp[ind,]
y=tecator$y$Fat[ind]

res.np=fregre.np(x,y,Ker=AKer.epa)
summary(res.np)
res.np2=fregre.np(x,y,Ker=AKer.tri)
summary(res.np2)

# with other semimetrics.
res.pca1=fregre.np(x,y,Ker=AKer.tri,metri=semimetric.pca,q=1)
summary(res.pca1)
res.deriv=fregre.np(x,y,metri=semimetric.deriv)
summary(res.deriv)
x.d2=fdata.deriv(x,nderiv=1,method="fmm",class.out='fdata')
res.deriv2=fregre.np(x.d2,y)
```

```
summary(res.deriv2)
x.d3=fdata.deriv(x,nderiv=1,method="bspline",class.out='fdata')
res.deriv3=fregre.np(x.d3,y)
summary(res.deriv3)

## End(Not run)
```

`fregre.np.cv` *Cross-validation functional regression with scalar response using kernel estimation.*

Description

Computes functional regression between functional explanatory variables and scalar response using asymmetric kernel estimation by cross-validation method.

Usage

```
fregre.np.cv(
  fdataobj,
  y,
  h = NULL,
  Ker = AKer.norm,
  metric = metric.lp,
  type.CV = GCV.S,
  type.S = S.NW,
  par.CV = list(trim = 0),
  par.S = list(w = 1),
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`h`	Bandwidth, `h>0`. Default argument values are provided as the sequence of length 25 from 2.5%–quantile to 25%–quantile of the distance between `fdataobj` curves, see `h.default`.
`Ker`	Type of asymmetric kernel used, by default asymmetric normal kernel.
`metric`	Metric function, by default `metric.lp`.
`type.CV`	Type of cross-validation. By default generalized cross-validation `GCV.S` method.
`type.S`	Type of smothing matrix `S`. By default `S` is calculated by Nadaraya-Watson kernel estimator (`S.NW`).
`par.CV`	List of parameters for `type.CV`: `trim`, the alpha of the trimming and `draw=TRUE`.
`par.S`	List of parameters for `type.S`: `w`, the weights.
`...`	Arguments to be passed for `metric.lp` o other metric function.

Details

The non-parametric functional regression model can be written as follows

$$y_i = r(X_i) + \epsilon_i$$

where the unknown smooth real function r is estimated using kernel estimation by means of

$$\hat{r}(X) = \frac{\sum_{i=1}^n K(h^{-1}d(X, X_i))y_i}{\sum_{i=1}^n K(h^{-1}d(X, X_i))}$$

where K is an kernel function (see `Ker` argument), `h` is the smoothing parameter and d is a metric or a semi-metric (see `metric` argument).

The function estimates the value of smoothing parameter (also called bandwidth) `h` through Generalized Cross-validation `GCV` criteria, see `GCV.S` or `CV.S`.

The function estimates the value of smoothing parameter or the bandwidth through the cross validation methods: `GCV.S` or `CV.S`. It computes the distance between curves using the `metric.lp`, although any other semimetric could be used (see `semimetric.basis` or `semimetric.NPFDA` functions). Different asymmetric kernels can be used, see `Kernel.asymmetric`.

Value

Return:

- `call`: The matched call.
- `residuals`: `y` minus `fitted values`.
- `fitted.values`: Estimated scalar response.
- `df.residual`: The residual degrees of freedom.
- `r2`: Coefficient of determination.
- `sr2`: Residual variance.
- `H`: Hat matrix.
- `y`: Response.
- `fdataobj`: Functional explanatory data.
- `mdist`: Distance matrix between `x` and `newx`.
- `Ker`: Asymmetric kernel used.
- `gcv`: CV or GCV values.
- `h.opt`: Smoothing parameter or bandwidth that minimizes CV or GCV method.
- `h`: Vector of smoothing parameter or bandwidth.
- `cv`: List with the fitted values and residuals estimated by CV, without the same curve.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as: `fregre.np`, `summary.fregre.fd` and `predict.fregre.fd` .
Alternative method: `fregre.basis.cv` and `fregre.np.cv`.

Examples

```
## Not run:
data(tecator)
absorp=tecator$absorp.fdata
ind=1:129
x=absorp[ind,]
y=tecator$y$Fat[ind]
Ker=AKer.tri
res.np=fregre.np.cv(x,y,Ker=Ker)
summary(res.np)
res.np2=fregre.np.cv(x,y,type.CV=GCV.S,criteria="Shibata")
summary(res.np2)

## Example with other semimetrics (not run)
res.pca1=fregre.np.cv(x,y,Ker=Ker,metric=semimetric.pca,q=1)
summary(res.pca1)
res.deriv=fregre.np.cv(x,y,Ker=Ker,metric=semimetric.deriv)
summary(res.deriv)

x.d2=fdata.deriv(x,nderiv=1,method="fmm",class.out='fdata')
res.deriv2=fregre.np.cv(x.d2,y,Ker=Ker)
summary(res.deriv2)
x.d3=fdata.deriv(x,nderiv=1,method="bspline",class.out='fdata')
res.deriv3=fregre.np.cv(x.d3,y,Ker=Ker)
summary(res.deriv3)

## End(Not run)
```

`fregre.pc` *Functional Regression with scalar response using Principal Components Analysis*

Description

Computes functional (ridge or penalized) regression between functional explanatory variable $X(t)$ and scalar response Y using Principal Components Analysis.

$$Y = \big\langle X, \beta \big\rangle + \epsilon = \int_T X(t)\beta(t)dt + \epsilon$$

where $\big\langle \cdot, \cdot \big\rangle$ denotes the inner product on L_2 and ϵ are random errors with mean zero , finite variance σ^2 and $E[X(t)\epsilon] = 0$.

Usage

```
fregre.pc(
  fdataobj,
  y,
  l = NULL,
  lambda = 0,
  P = c(0, 0, 1),
  weights = rep(1, len = n),
  ...
)
```

Arguments

`fdataobj`	`fdata` class object or `fdata.comp` class object created by `create.pc.basis` function.
`y`	Scalar response with length `n`.
`l`	Index of components to include in the model.If is null `l` (by default), `l=1:3`.
`lambda`	Amount of penalization. Default value is 0, i.e. no penalization is used.
`P`	If `P` is a vector: `P` are coefficients to define the penalty matrix object, see `P.penalty`. If `P` is a matrix: P is the penalty matrix object.
`weights`	weights
`...`	Further arguments passed to or from other methods.

Details

The function computes the $\{\nu_k\}_{k=1}^{\infty}$ orthonormal basis of functional principal components to represent the functional data as $X_i(t) = \sum_{k=1}^{\infty} \gamma_{ik}\nu_k$ and the functional parameter as $\beta(t) = \sum_{k=1}^{\infty} \beta_k\nu_k$, where $\gamma_{ik} = \Big\langle X_i(t), \nu_k \Big\rangle$ and $\beta_k = \Big\langle \beta, \nu_k \Big\rangle$.
The response can be fitted by:

- $\lambda = 0$, no penalization,
$$\hat{y} = \nu_k^\top (\nu_k^\top \nu_k)^{-1} \nu_k^\top y$$
- Ridge regression, $\lambda > 0$ and $P = 1$,
$$\hat{y} = \nu_k^\top (\nu_k \top \nu_k + \lambda I)^{-1} \nu_k^\top y$$

- Penalized regression, $\lambda > 0$ and $P \neq 0$. For example, $P = c(0,0,1)$ penalizes the second derivative (curvature) by `P=P.penalty(fdataobj["argvals"],P)`,

$$\hat{y} = \nu_k^\top (\nu_k \top \nu_k + \lambda \nu_k^\top \mathbf{P} \nu_k)^{-1} \nu_k^\top y$$

Value

Return:

- `call`: The matched call of `fregre.pc` function.
- `coefficients`: A named vector of coefficients.
- `residuals`: `y` minus `fitted values`.
- `fitted.values`: Estimated scalar response.
- `beta.est`: Beta coefficient estimated of class `fdata`.
- `df.residual`: The residual degrees of freedom. In ridge regression, `df(rn)` is the effective degrees of freedom.
- `r2`: Coefficient of determination.
- `sr2`: Residual variance.
- `Vp`: Estimated covariance matrix for the parameters.
- `H`: Hat matrix.
- `l`: Index of principal components selected.
- `lambda`: Amount of shrinkage.
- `P`: Penalty matrix.
- `fdata.comp`: Fitted object in `fdata2pc` function.
- `lm`: `lm` object.
- `fdataobj`: Functional explanatory data.
- `y`: Scalar response.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Cai TT, Hall P. 2006. *Prediction in functional linear regression*. Annals of Statistics 34: 2159-2179.

Cardot H, Ferraty F, Sarda P. 1999. *Functional linear model*. Statistics and Probability Letters 45: 11-22.

Hall P, Hosseini-Nasab M. 2006. *On properties of functional principal components analysis*. Journal of the Royal Statistical Society B 68: 109-126.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

N. Kraemer, A.-L. Boulsteix, and G. Tutz (2008). *Penalized Partial Least Squares with Applications to B-Spline Transformations and Functional Data*. Chemometrics and Intelligent Laboratory Systems, 94, 60 - 69. doi:10.1016/j.chemolab.2008.06.009

See Also

See Also as: fregre.pc.cv, summary.fregre.fd and predict.fregre.fd.

Alternative method: fregre.basis and fregre.np.

Examples

```
## Not run:
data(tecator)
absorp <- tecator$absorp.fdata
ind <- 1:129
x <- absorp[ind,]
y <- tecator$y$Fat[ind]
res <- fregre.pc(x,y)
summary(res)
res2 <- fregre.pc(x,y,l=c(1,3,4))
summary(res2)
# Functional Ridge Regression
res3 <- fregre.pc(x,y,l=c(1,3,4),lambda=1,P=1)
summary(res3)
# Functional Regression with 2nd derivative penalization
res4 <- fregre.pc(x,y,l=c(1,3,4),lambda=1,P=c(0,0,1))
summary(res4)
betas <- c(res$beta.est,res2$beta.est,
           res3$beta.est,res4$beta.est)
plot(betas)

## End(Not run)
```

---

fregre.pc.cv *Functional penalized PC regression with scalar response using selection of number of PC components*

Description

Functional Regression with scalar response using selection of number of (penalized) principal components PC through cross-validation. The algorithm selects the PC with best estimates the response. The selection is performed by cross-validation (CV) or Model Selection Criteria (MSC). After is computing functional regression using the best selection of principal components.

Usage

```
fregre.pc.cv(
  fdataobj,
  y,
  kmax = 8,
  lambda = 0,
  P = c(0, 0, 1),
  criteria = "SIC",
  weights = rep(1, len = n),
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`kmax`	The number of components to include in the model.
`lambda`	Vector with the amounts of penalization. Default value is 0, i.e. no penalization is used. If `lambda=TRUE` the algorithm computes a sequence of lambda values.
`P`	The vector of coefficients to define the penalty matrix object. For example, if `P=c(1,0,0)`, ridge regresion is computed and if `P=c(0,0,1)`, penalized regression is computed penalizing the second derivative (curvature).
`criteria`	Type of cross-validation (CV) or Model Selection Criteria (MSC) applied. Possible values are *"CV"*, *"AIC"*, *"AICc"*, *"SIC"*, *"SICc"*, *"HQIC"*.
`weights`	weights
`...`	Further arguments passed to `fregre.pc` or `fregre.pls`

Details

The algorithm selects the best principal components `pc.opt` from the first `kmax` PC and (optionally) the best penalized parameter `lambda.opt` from a sequence of non-negative numbers `lambda`.
If `kmax` is a integer (by default and recomended) the procedure is as follows (see example 1):

- Calculate the best principal component (*pc.order[1]*) between `kmax` by `fregre.pc`.
- Calculate the second-best principal component (`pc.order [2]`) between the `(kmax-1)` by `fregre.pc` and calculate the criteria value of the two principal components.
- The process (point 1 and 2) is repeated until `kmax` principal component (*pc.order[kmax]*).
- The proces (point 1, 2 and 3) is repeated for each `lambda` value.
- The method selects the principal components (`pc.opt=pc.order[1:k.min]`) and (optionally) the lambda parameter with minimum MSC criteria.

If `kmax` is a sequence of integer the procedure is as follows (see example 2):

- The method selects the best principal components with minimum MSC criteria by stepwise regression using `fregre.pc` in each step.
- The process (point 1) is repeated for each `lambda` value.
- The method selects the principal components (`pc.opt=pc.order[1:k.min]`) and (optionally) the lambda parameter with minimum MSC criteria.

Finally, is computing functional PC regression between functional explanatory variable $X(t)$ and scalar response Y using the best selection of PC `pc.opt` and ridge parameter `rn.opt`.
The criteria selection is done by cross-validation (CV) or Model Selection Criteria (MSC).

- Predictive Cross-Validation: $PCV(k_n) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_{(-i,k_n)}\right)^2$, `criteria`="CV"
- Model Selection Criteria: $MSC(k_n) = log\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2\right] + p_n\frac{k_n}{n}$

$p_n = \frac{log(n)}{n}$, `criteria`="SIC" (by default)
$p_n = \frac{log(n)}{n-k_n-2}$, `criteria`="SICc"
$p_n = 2$, `criteria`="AIC"
$p_n = \frac{2n}{n-k_n-2}$, `criteria`="AICc"
$p_n = \frac{2log(log(n))}{n}$, `criteria`="HQIC"
where `criteria` is an argument that controls the type of validation used in the selection of the smoothing parameter `kmax`$= k_n$ and penalized parameter `lambda`$= \lambda$.

Value

Return:

- `fregre.pc`: Fitted regression object by the best (`pc.opt`) components.
- `pc.opt`: Index of PC components selected.
- `MSC.min`: Minimum Model Selection Criteria (MSC) value for the (`pc.opt`) components.
- `MSC`: Minimum Model Selection Criteria (MSC) value for `kmax` components.

Note

`criteria=``CV''` is not recommended: time-consuming.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See also as:`fregre.pc` .

Examples

```
## Not run:
data(tecator)
x<-tecator$absorp.fdata[1:129]
y<-tecator$y$Fat[1:129]
# no penalization
 res.pc1=fregre.pc.cv(x,y,8)
# 2nd derivative penalization
```

```
 res.pc2=fregre.pc.cv(x,y,8,lambda=TRUE,P=c(0,0,1))
# Ridge regression
res.pc3=fregre.pc.cv(x,y,1:8,lambda=TRUE,P=1)

## End(Not run)
```

fregre.plm *Semi-functional partially linear model with scalar response.*

Description

Computes functional regression between functional (and non functional) explanatory variables and scalar response using asymmetric kernel estimation.

Usage

```
fregre.plm(
  formula,
  data,
  h = NULL,
  Ker = AKer.norm,
  metric = metric.lp,
  type.CV = GCV.S,
  type.S = S.NW,
  par.CV = list(trim = 0, draw = FALSE),
  par.S = list(w = 1),
  ...
)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under `Details`.
`data`	List that containing the variables in the model.
`h`	Bandwidth, `h>0`. Default argument values are provided as the sequence of length 51 from 2.5%–quantile to 25%–quantile of the distance between the functional data, see `h.default`.
`Ker`	Type of asymmetric kernel used, by default asymmetric normal kernel.
`metric`	Metric function, by default `metric.lp`.
`type.CV`	Type of cross-validation. By default generalized cross-validation `GCV.S` method.
`type.S`	Type of smothing matrix `S`. By default `S` is calculated by Nadaraya-Watson kernel estimator (`S.NW`).
`par.CV`	List of parameters for `type.CV`: `trim`, the alpha of the trimming and `draw=TRUE`.

`par.S`	List of parameters for `type.S`: `w`, the weights.
`...`	Further arguments passed to or from other methods.

Details

An extension of the non-parametric functional regression models is the semi-functional partial linear model proposed in Aneiros-Perez and Vieu (2005). This model uses a non-parametric kernel procedure as that described in `fregre.np`. The output y is scalar. A functional covariate X and a multivariate non functional covariate Z are considered.

$$y = r(X) + \sum_{j=1}^{p} Z_j \beta_j + \epsilon$$

The unknown smooth real function r is estimated by means of

$$\hat{r}_h(X) = \sum_{i=1}^{n} w_{n,h}(X, X_i)(Y_i - Z_i^T \hat{\beta}_h)$$

where W_h is the weight function:
$w_{n,h}(X, X_i) = \frac{K(d(X,X_i)/h)}{\sum_{j=1}^{n} K(d(X,X_j)/h)}$ with smoothing parameter h, an asymmetric kernel K and a metric or semi-metric d. In `fregre.plm()` by default W_h is a functional version of the Nadaraya-Watson-type weights (`type.S=S.NW`) with asymmetric normal kernel (`Ker=AKer.norm`) in L_2 (`metric=metric.lp` with `p=2`). The unknown parameters β_j for the multivariate non functional covariates are estimated by means of $\hat{\beta}_j = (\tilde{Z}_h^T \tilde{Z}_h)^{-1} \tilde{Z}_h^T \tilde{Z}_h$ where $\tilde{Z}_h = (I - W_h)Z$ with the smoothing parameter h. The errors ϵ are independent, with zero mean, finite variance σ^2 and $E[\epsilon | Z_1, \ldots, Z_p, X(t)] = 0$.

The first item in the `data` list is called *"df"* and is a data frame with the response and non functional explanatory variables, as `link{lm}`. If non functional data into the formula then `lm` regression is performed.
Functional variable (`fdata` or `fd` class) is introduced in the second item in the `data` list. If only functional variable into the formula then `fregre.np.cv` is performed.

The function estimates the value of smoothing parameter or the bandwidth `h` through Generalized Cross-validation `GCV` criteria. It computes the distance between curves using the `metric.lp`, although you can also use other metric function.
Different asymmetric kernels can be used, see `Kernel.asymmetric`.

Value

- `call`: The matched call.
- `fitted.values`: Estimated scalar response.
- `residuals`: `y` minus `fitted values`.
- `df.residual`: The residual degrees of freedom.
- `H`: Hat matrix.

- `r2`: Coefficient of determination.
- `sr2`: Residual variance.
- `y`: Scalar response.
- `fdataobj`: Functional explanatory data.
- `XX`: Non functional explanatory data.
- `mdist`: Distance matrix between curves.
- `betah`: beta coefficient estimated.
- `data`: List that containing the variables in the model.
- `Ker`: Asymmetric kernel used.
- `h.opt`: Value that minimizes CV or GCV method.
- `h`: Smoothing parameter or bandwidth.
- `data`: List that containing the variables in the model.
- `gcv`: GCV values.
- `formula`: formula.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Aneiros-Perez G. and Vieu P. (2005). *Semi-functional partial linear regression*. Statistics & Probability Letters, 76:1102-1110.

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as: `predict.fregre.plm` and `summary.fregre.fd`
Alternative methods: `fregre.lm`, `fregre.np` and `fregre.np.cv`

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata[1:129]
dataf=tecator$y[1:129,]

f=Fat~Water+x
ldata=list("df"=dataf,"x"=x)
res.plm=fregre.plm(f,ldata)
```

```
summary(res.plm)

# with 2nd derivative of functional data
x.fd=fdata.deriv(x,nderiv=2)
f2=Fat~Water+x.fd
ldata2=list("df"=dataf,"x.fd"=x.fd)
res.plm2=fregre.plm(f2,ldata2)
summary(res.plm2)

## End(Not run)
```

fregre.pls *Functional Penalized PLS regression with scalar response*

Description

Computes functional linear regression between functional explanatory variable $X(t)$ and scalar response Y using penalized Partial Least Squares (PLS)

$$Y = \langle \tilde{X}, \beta \rangle + \epsilon = \int_T \tilde{X}(t)\beta(t)dt + \epsilon$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on L_2 and ϵ are random errors with mean zero , finite variance σ^2 and $E[\tilde{X}(t)\epsilon] = 0$.
$\{\nu_k\}_{k=1}^{\infty}$ orthonormal basis of PLS to represent the functional data as $X_i(t) = \sum_{k=1}^{\infty} \gamma_{ik}\nu_k$.

Usage

```
fregre.pls(fdataobj, y = NULL, l = NULL, lambda = 0, P = c(0, 0, 1), ...)
```

Arguments

fdataobj	fdata class object.
y	Scalar response with length n.
l	Index of components to include in the model.
lambda	Amount of penalization. Default value is 0, i.e. no penalization is used.
P	If P is a vector: P are coefficients to define the penalty matrix object. By default P=c(0,0,1) penalize the second derivative (curvature) or acceleration. If P is a matrix: P is the penalty matrix object.
...	Further arguments passed to or from other methods.

Details

Functional (FPLS) algorithm maximizes the covariance between $X(t)$ and the scalar response Y via the partial least squares (PLS) components. The functional penalized PLS are calculated in `fdata2pls` by alternative formulation of the NIPALS algorithm proposed by Kraemer and Sugiyama (2011).
Let $\{\tilde{\nu}_k\}_{k=1}^{\infty}$ the functional PLS components and $\tilde{X}_i(t) = \sum_{k=1}^{\infty} \tilde{\gamma}_{ik}\tilde{\nu}_k$ and $\beta(t) = \sum_{k=1}^{\infty} \tilde{\beta}_k\tilde{\nu}_k$. The functional linear model is estimated by:

$$\hat{y} = \big\langle X, \hat{\beta} \big\rangle \approx \sum_{k=1}^{k_n} \tilde{\gamma}_k \tilde{\beta}_k$$

The response can be fitted by:

- $\lambda = 0$, no penalization,
$$\hat{y} = \nu_k^\top (\nu_k^\top \nu_k)^{-1} \nu_k^\top y$$
  - Penalized regression, $\lambda > 0$ and $P \neq 0$. For example, $P = c(0, 0, 1)$ penalizes the second derivative (curvature) by `P=P.penalty(fdataobj["argvals"],P)`,
$$\hat{y} = \nu_k^\top (\nu_k \top \nu_k + \lambda \nu_k^\top \mathbf{P} \nu_k)^{-1} \nu_k^\top y$$

Value

Return:

- `call`: The matched call of `fregre.pls` function.
- `beta.est`: Beta coefficient estimated of class `fdata`.
- `coefficients`: A named vector of coefficients.
- `fitted.values`: Estimated scalar response.
- `residuals`: `y` minus `fitted values`.
- `H`: Hat matrix.
- `df.residual`: The residual degrees of freedom.
- `r2`: Coefficient of determination.
- `GCV`: GCV criterion.
- `sr2`: Residual variance.
- `l`: Index of components to include in the model.
- `lambda`: Amount of shrinkage.
- `fdata.comp`: Fitted object in `fdata2pls` function.
- `lm`: Fitted object in `lm` function.
- `fdataobj`: Functional explanatory data.
- `y`: Scalar response.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente `<manuel.oviedo@udc.es>`

References

Preda C. and Saporta G. *PLS regression on a stochastic process*. Comput. Statist. Data Anal. 48 (2005): 149-158.

N. Kraemer, A.-L. Boulsteix, and G. Tutz (2008). *Penalized Partial Least Squares with Applications to B-Spline Transformations and Functional Data*. Chemometrics and Intelligent Laboratory Systems, 94, 60 - 69. doi:10.1016/j.chemolab.2008.06.009

Martens, H., Naes, T. (1989) *Multivariate calibration.* Chichester: Wiley.

Kraemer, N., Sugiyama M. (2011). *The Degrees of Freedom of Partial Least Squares Regression*. Journal of the American Statistical Association. Volume 106, 697-705.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as: `P.penalty` and `fregre.pls.cv`.
Alternative method: `fregre.pc`.

Examples

```
## Not run:
data(tecator)
x <- tecator$absorp.fdata
y <- tecator$y$Fat
res <- fregre.pls(x,y,c(1:4))
summary(res)
res1 <- fregre.pls(x,y,l=1:4,lambda=100,P=c(1))
res4 <- fregre.pls(x,y,l=1:4,lambda=1,P=c(0,0,1))
summary(res4)#' plot(res$beta.est)
lines(res1$beta.est,col=4)
lines(res4$beta.est,col=2)

## End(Not run)
```

`fregre.pls.cv` *Functional penalized PLS regression with scalar response using selection of number of PLS components*

Description

Functional Regression with scalar response using selection of number of penalized principal componentes PPLS through cross-validation. The algorithm selects the PPLS components with best estimates the response. The selection is performed by cross-validation (CV) or Model Selection Criteria (MSC). After is computing functional regression using the best selection of PPLS components.

Usage

```
fregre.pls.cv(
  fdataobj,
  y,
  kmax = 8,
  lambda = 0,
  P = c(0, 0, 1),
  criteria = "SIC",
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`y`	Scalar response with length `n`.
`kmax`	The number of components to include in the model.
`lambda`	Vector with the amounts of penalization. Default value is 0, i.e. no penalization is used. If `lambda=TRUE` the algorithm computes a sequence of lambda values.
`P`	The vector of coefficients to define the penalty matrix object. For example, if `P=c(0,0,1)`, penalized regression is computed penalizing the second derivative (curvature).
`criteria`	Type of cross-validation (CV) or Model Selection Criteria (MSC) applied. Possible values are *"CV"*, *"AIC"*, *"AICc"*, *"SIC"*, *"SICc"*, *"HQIC"*.
`...`	Further arguments passed to `fregre.pls`.

Details

The algorithm selects the best principal components `pls.opt` from the first `kmax` PLS and (optionally) the best penalized parameter `lambda.opt` from a sequence of non-negative numbers `lambda`.

- The method selects the best principal components with minimum MSC criteria by stepwise regression using `fregre.pls` in each step.
- The process (point 1) is repeated for each `lambda` value.
- The method selects the principal components (`pls.opt=pls.order[1:k.min]`) and (optionally) the lambda parameter with minimum MSC criteria.

Finally, is computing functional PLS regression between functional explanatory variable $X(t)$ and scalar response Y using the best selection of PLS `pls.opt` and ridge parameter `rn.opt`. The criteria selection is done by cross-validation (CV) or Model Selection Criteria (MSC).

- Predictive Cross-Validation: $PCV(k_n) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \hat{y}_{(-i,k_n)}\Big)^2$, `criteria`="CV"
- Model Selection Criteria: $MSC(k_n) = log\left[\frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \hat{y}_i\Big)^2\right] + p_n\frac{k_n}{n}$
  $p_n = \frac{log(n)}{n}$, `criteria`="SIC" (by default)
  $p_n = \frac{log(n)}{n-k_n-2}$, `criteria`="SICc"

$p_n = 2$, `criteria`="AIC"
$p_n = \frac{2n}{n-k_n-2}$, `criteria`="AICc"
$p_n = \frac{2log(log(n))}{n}$, `criteria`="HQIC"
where `criteria` is an argument that controls the type of validation used in the selection of the smoothing parameter `kmax`$= k_n$ and penalized parameter `lambda`$= \lambda$.

Value

Return:

- `fregre.pls`: Fitted regression object by the best (`pls.opt`) components.
- `pls.opt`: Index of PLS components selected.
- `MSC.min`: Minimum Model Selection Criteria (MSC) value for the (`pls.opt`) components.
- `MSC`: Minimum Model Selection Criteria (MSC) value for `kmax` components.

Note

`criteria=``CV''` is not recommended: time-consuming.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Preda C. and Saporta G. *PLS regression on a stochastic process*. Comput. Statist. Data Anal. 48 (2005): 149-158.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See also as:`fregre.pc` .

Examples

```
## Not run:
data(tecator)
x<-tecator$absorp.fdata[1:129]
y<-tecator$y$Fat[1:129]
# no penalization
pls1<- fregre.pls.cv(x,y,8)
# 2nd derivative penalization
pls2<-fregre.pls.cv(x,y,8,lambda=0:5,P=c(0,0,1))

## End(Not run)
```

GCCV.S	*The generalized correlated cross-validation (GCCV) score.*

Description

The generalized correlated cross-validation (GCV) score.

Usage

```
GCCV.S(
  y,
  S,
  criteria = "GCCV1",
  W = NULL,
  trim = 0,
  draw = FALSE,
  metric = metric.lp,
  ...
)
```

Arguments

`y`	Response vectorith length `n` or Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
`S`	Smoothing matrix, see `S.NW`, `S.LLR` or $S.KNN$.
`criteria`	The penalizing function. By default *"Rice"* criteria. "GCCV1","GCCV2","GCCV3","GCV") Possible values are *"GCCV1"*, *"GCCV2"*, *"GCCV3"*, *"GCV"*.
`W`	Matrix of weights.
`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`metric`	Metric function, by default `metric.lp`.
`...`	Further arguments passed to or from other methods.

Details

$$GCCV = \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_{i,b}\right)^2}{(1 - \frac{tr(C)}{n})^2}$$

where $C = 2S\Sigma(\theta) - S\Sigma(\theta)S'$
and Σ is the $n \times n$ covariance matrix with $cor(\epsilon_i, \epsilon_j) = \sigma$.

Here, S is the smoothing matrix, and there are options for C:

- A.- If $C = 2S\Sigma - S\Sigma S$

- B.- If $C = S\Sigma$
- C.- If $C = S\Sigma S'$

with Σ as the $n \times n$ covariance matrix and $cor(\epsilon_i, \epsilon_j) = \sigma$.

Value

Returns GCCV score calculated for input parameters.

Note

Provided that $C = I$ and the smoother matrix S is symmetric and idempotent, as is the case for many linear fitting techniques, the trace term reduces to $n - tr[S]$, which is proportional to the familiar denominator in GCV.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Carmack, P. S., Spence, J. S., and Schucany, W. R. (2012). Generalised correlated cross-validation. Journal of Nonparametric Statistics, 24(2):269–282.

Oviedo de la Fuente, M., Febrero-Bande, M., Munoz, P., and Dominguez, A. Predicting seasonal influenza transmission using Functional Regression Models with Temporal Dependence.https://arxiv.org/abs/1610.08718

See Also

See Also as optim.np.
Alternative method (independent case): GCV.S

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
x.d2<-fdata.deriv(x,nderiv=)
tt<-x[["argvals"]]
dataf=as.data.frame(tecator$y)
y=tecator$y$Fat
# plot the response
plot(ts(tecator$y$Fat))

nbasis.x=11;nbasis.b=7
basis1=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.x)
basis2=create.bspline.basis(rangeval=range(tt),nbasis=nbasis.b)
basis.x=list("x.d2"=basis1)
basis.b=list("x.d2"=basis2)
ldata=list("df"=dataf,"x.d2"=x.d2)
# No correlation
```

```
res.gls=fregre.gls(Fat~x.d2,data=ldata,
                   basis.x=basis.x,basis.b=basis.b)
# AR1 correlation
res.gls=fregre.gls(Fat~x.d2,data=ldata, correlation=corAR1(),
                   basis.x=basis.x,basis.b=basis.b)
GCCV.S(y,res.gls$H,"GCCV1",W=res.gls$W)
res.gls$gcv

## End(Not run)
```

GCV.S *The generalized correlated cross-validation (GCCV) score*

Description

Compute the generalized correlated cross-validation (GCV) score.

Usage

```
GCV.S(
  y,
  S,
  criteria = "GCV",
  W = NULL,
  trim = 0,
  draw = FALSE,
  metric = metric.lp,
  ...
)
```

Arguments

`y`	Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
`S`	Smoothing matrix, see `S.NW`, `S.LLR` or
`criteria`	The penalizing function. By default *"Rice"* criteria. Possible values are *"GCCV1"*, *"GCCV2"*, *"GCCV3"*, *"GCV"*.
`W`	Matrix of weights.
`trim`	The alpha of the trimming.
`draw`	=TRUE, draw the curves, the sample median and trimmed mean.
`metric`	Metric function, by default `metric.lp`.
`...`	Further arguments passed to or from other methods.

Details

A.-If `trim=0`:

$$GCCV = \frac{\sum_{i=1}^{n} y_i - \hat{y}_{i,b}^{2}}{1 - \frac{tr(C)}{n}^{2}}$$

where S is the smoothing matrix S and:
A.-If $C = 2S\Sigma - S\Sigma S$
B.-If $C = S\Sigma$
C.-If $C = S\Sigma S'$
with Σ is the n x n covariance matrix with $cor(\epsilon_i, \epsilon_j) = \sigma$ Note: Provided that $C = I$ and the smoother matrix S is symmetric and idempotent, as is the case for many linear fitting techniques, the trace term reduces to $n - tr[S]$, which is proportional to the familiar denominator in GCV.

Value

Returns GCV score calculated for input parameters.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006. Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994. Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as `optim.np`
Alternative method: `CV.S`

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme$learn
tt<-1:ncol(mlearn)
S1 <- S.NW(tt,2.5)
S2 <- S.LLR(tt,2.5)
gcv1 <- GCV.S(mlearn, S1)
gcv2 <- GCV.S(mlearn, S2)
gcv3 <- GCV.S(mlearn, S1,criteria="AIC")
gcv4 <- GCV.S(mlearn, S2,criteria="AIC")
gcv1; gcv2; gcv3; gcv4

## End(Not run)
```

`h.default` *Calculation of the smoothing parameter (h) for a functional data*

Description

Calculation of the smoothing parameter (h) for a functional data using nonparametric kernel estimation.

Usage

```
h.default(
  fdataobj,
  prob = c(0.025, 0.25),
  len = 51,
  metric = metric.lp,
  type.S = "S.NW",
  Ker = Ker.norm,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`prob`	Vector of probabilities for extracting the quantiles of the distance matrix. If `length(prob)=2` a sequence between `prob[1]` and `prob[2]` of length `len`.
`len`	Vector length of smoothing parameter `h` to return only used when `length(prob)=2`.
`metric`	If is a function: name of the function to calculate the distance matrix between the curves, by default `metric.lp`. If is a matrix: distance matrix between the curves. kernel.
`type.S`	Type of smothing matrix `S`. Possible values are: Nadaraya-Watson estimator *"S.NW"* and K nearest neighbors estimator *"S.KNN"*
`Ker`	Kernel function. By default, *Ker.norm*. Useful for scaling the bandwidth values according to Kernel
`...`	Arguments to be passed for metric argument.

Value

Returns the vector of smoothing parameter or bandwidth `h`.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

See Also

See Also as `metric.lp`, `Kernel` and `S.NW`.
Function used in `fregre.np` and `fregre.np.cv` function.

Examples

```
## Not run:
data(aemet)
h1<-h.default(aemet$temp,prob=c(0.025, 0.25),len=2)
mdist<-metric.lp(aemet$temp)
h2<-h.default(aemet$temp,len=2,metric=mdist)
h3<-h.default(aemet$temp,len=2,metric=semimetric.pca,q=2)
h4<-h.default(aemet$temp,len=2,metric=semimetric.pca,q=4)
h5<-h.default(aemet$temp,prob=c(.2),type.S="S.KNN")
h1;h2;h3;h4;h5

## End(Not run)
```

`influence.fregre.fd` *Functional influence measures*

Description

Once estimated the functional regression model with scalar response, influence.fregre.fd function is used to obtain the functional influence measures.

Usage

```
## S3 method for class 'fregre.fd'
influence(model, ...)
```

Arguments

`model`	`fregre.pc`, `fregre.basis` or `fregre.basis.cv` object.
`...`	Further arguments passed to or from other methods.

Details

Identify influential observations in the functional linear model in which the predictor is functional and the response is scalar. Three statistics are introduced for measuring the influence: Distance Cook Prediction `DCP`, Distance Cook Estimation `DCE` and Distance peña `DP` respectively.

Value

Return:

- `DCP`: Cook's Distance for Prediction.
- `DCE`: Cook's Distance for Estimation.
- `DP`:Peña's Distance.

Note

influence.fdata deprecated.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Galeano, P. and Gonzalez-Manteiga, W. (2010). *Measures of influence for the functional linear model with scalar response*. Journal of Multivariate Analysis 101, 327-339.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as: `fregre.pc`, `fregre.basis`, `influence_quan`

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata[1:129]
y=tecator$y$Fat[1:129]

res1=fregre.pc(x,y,1:5)
# time consuming
res.infl1=influence(res1)
res2=fregre.basis(x,y)
res.infl2=influence(res2)

res<-res1
res.infl<-res.infl1
mat=cbind(y,res$fitted.values,res.infl$DCP,res.infl$DCE,res.infl$DP)
colnames(mat)=c("Resp.","Pred.","DCP","DCE","DP")
pairs(mat)

## End(Not run)
```

`influence_quan` *Quantile for influence measures*

Description

Estimate the quantile of measures of influence for each observation.

Usage

```
influence_quan(model,out.influ,mue.boot=500,
smo=0.1,smoX=0.05,alpha=0.95,kmax.fix=FALSE,...)
```

Arguments

`model`	`fregre.pc`, `fregre.basis` or `fregre.basis.cv` object.
`out.influ`	`inflluence.fd` bject
`mue.boot`	Number of bootstrap samples
`smo`	Smoothing parameter as a proportion of response variance.
`smoX`	Smoothing parameter for `fdata` object as a proportion of variance-covariance matrix of the explanatory functional variable.
`alpha`	Significance level.
`kmax.fix`	The maximum number of principal comoponents or number of basis is fixed by `model` object.
`...`	Further arguments passed to or from other methods.

Details

Compute the quantile of measures of influence estimated in `influence.fregre.fd` for functional regression using principal components representation (`fregre.pc`) or basis representation (`fregre.basis` or `fregre.basis.cv`).

A smoothed bootstrap method is used to estimate the quantiles of the influence measures, which allows to point out which observations have the larger influence on estimation and prediction.

Value

Return:

- `quan.cook.for`: Distance Cook Prediction Quantile.
- `quan.cook.est`: Distance Cook Estimation Quantile.
- `quan.cook.Pena`: Pena Distance Quantile.
- `mues.est`: Sample Cook generated.
- `mues.pena`: Sample Pena generated.
- `beta.boot`: Functional beta estimated by bootstrap method.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Galeano, P. and Gonzalez-Manteiga, W. (2010). *Measures of influence for the functional linear model with scalar response*. Journal of Multivariate Analysis 101, 327-339.

See Also

See Also as: `influence.fregre.fd`, `fregre.basis`, `fregre.pc`.

Examples

```
## Not run:
data(tecator)
x=tecator$absorp.fdata
y=tecator$y$Fat
res=fregre.pc(x,y,1:6)

#time consuming
res.infl=influence.fregre.fd(res)
resquan=influence_quan(res,res.infl,4,0.01,0.05,0.95)
plot(res.infl$betas,type="l",col=2)
lines(res$beta.est,type="l",col=3)
lines(resquan$betas.boot,type="l",col="gray")

res=fregre.basis(x,y)
res.infl=influence.fregre.fd(res)
resquan=influence_quan(res,res.infl,mue.boot=4,kmax.fix=T)
plot(resquan$betas.boot,type="l",col=4)
lines(res.infl$betas,type="l",col=2)
lines(resquan$betas.boot,type="l",col="gray")

## End(Not run)
```

---

inprod.fdata *Inner products of Functional Data Objects o class (fdata)*

Description

Computes a inner products of functional data objects of class fdata.

Usage

```
inprod.fdata(fdata1, fdata2 = NULL, w = 1, ...)
```

Arguments

`fdata1`	Functional data 1 or curve 1. `fdata1$data` with dimension (`n1` x `m`), where `n1` is the number of curves and `m` are the points observed in each curve.
`fdata2`	Functional data 2 or curve 2. `fdata2$data` with dimension (`n2` x `m`), where `n2` is the number of curves and `m` are the points observed in each curve.
`w`	Vector of weights with length `m`, If `w` = 1 approximates the metric Lp by Simpson's rule. By default it uses `w` = 1
`...`	Further arguments passed to or from other methods.

Details

By default it uses weights `w=1`.

$$\langle fdata1, fdata2\rangle = \frac{1}{\int_a^b w(x)dx}\int_a^b fdata1(x) * fdata2(x)w(x)dx$$

The observed points on each curve are equally spaced (by default) or not.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See also inprod and `norm.fdata`

Examples

```
## Not run:
x<-seq(0,2*pi,length=1001)
fx1<-sin(x)/sqrt(pi)
fx2<-cos(x)/sqrt(pi)
argv<-seq(0,2*pi,len=1001)
fdat0<-fdata(rep(0,len=1001),argv,range(argv))
fdat1<-fdata(fx1,x,range(x))
inprod.fdata(fdat1,fdat1)
inprod.fdata(fdat1,fdat1)
metric.lp(fdat1)
metric.lp(fdat1,fdat0)
norm.fdata(fdat1)
# The same
integrate(function(x){(abs(sin(x)/sqrt(pi))^2)},0,2*pi)
integrate(function(x){(abs(cos(x)/sqrt(pi))^2)},0,2*pi)

## End(Not run)
```

`int.simpson` *Simpson integration*

Description

Computes the integral of `fdataobj$data` with respect to `fdataobj$argvals` using simpson or trapezoid rule integration.

Usage

```
int.simpson(fdataobj, method = NULL)

int.simpson2(x, y, equi = TRUE, method = NULL)
```

Arguments

`fdataobj`	fdata objtect.
`method`	Method for numerical integration, see details.
`x`	Sorted vector of x-axis values: `argvals`.
`y`	Vector of y-axis values.
`equi`	=TRUE, the observed points on each curve are equally spaced (by default).

Details

Posible values for `method` are:

- `"TRAPZ"`: Trapezoid rule integration.
- `"CSR"`: Composite Simpson's rule integration.
- `"ESR"`: Extended Simpson's rule integration.

If `method=NULL` (default), the value of `par.fda.usc$int.method` is used.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

See Also

See also `integrate`.

Examples

```
## Not run:
x<-seq(0,2*pi,length=1001)
fx<-fdata(sin(x)/sqrt(pi),x)
fx0<-fdata(rep(0,length(x)),x)
int.simpson(fx0)
int.simpson(fx)

## End(Not run)
```

`Kernel` *Symmetric Smoothing Kernels.*

Description

Represent symmetric smoothing kernels:: normal, cosine, triweight, quartic and uniform.

Usage

```
Kernel(u, type.Ker = "Ker.norm")
```

Arguments

`u`	Data.
`type.Ker`	Type of Kernel. By default normal kernel.

Details

Ker.norm=dnorm(u)
Ker.cos=ifelse(abs(u)<=1,pi/4*(cos(pi*u/2)),0)
Ker.epa=ifelse(abs(u)<=1,3/4*(1-u^2),0)
Ker.tri=ifelse(abs(u)<=1,35/32*(1-u^2)^3,0)
Ker.quar=ifelse(abs(u)<=1,15/16*(1-u^2)^2,0)
Ker.unif=ifelse(abs(u)<=1,1/2,0)

Type of kernel:

Normal Kernel: `Ker.norm`
Cosine Kernel: `Ker.cos`
Epanechnikov Kernel: `Ker.epa`
Triweight Kernel: `Ker.tri`
Quartic Kernel: `Ker.quar`
Uniform Kernel: `Ker.unif`

Value

Returns symmetric kernel.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Examples

```
y=qnorm(seq(.1,.9,len=100))
a<-Kernel(u=y)
b<-Kernel(type.Ker="Ker.tri",u=y)
c=Ker.cos(y)
```

`Kernel.asymmetric` *Asymmetric Smoothing Kernel*

Description

Represent Asymmetric Smoothing Kernels: normal, cosine, triweight, quartic and uniform.

AKer.norm=ifelse(u>=0,2*dnorm(u),0)
AKer.cos=ifelse(u>=0,pi/2*(cos(pi*u/2)),0)
AKer.epa=ifelse(u>=0 & u<=1,3/2*(1-u^2),0)
AKer.tri=ifelse(u>=0 & u<=1,35/16*(1-u^2)^3,0)
AKer.quar=ifelse(u>=0 & u<=1,15/8*(1-u^2)^2,0)
AKer.unif=ifelse(u>=0 & u<=1,1,0)

Usage

```
Kernel.asymmetric(u, type.Ker = "AKer.norm")
```

Arguments

`u`	Data.
`type.Ker`	Type of asymmetric metric kernel, by default asymmetric normal kernel.

Details

Type of Asymmetric kernel:

Asymmetric Normal Kernel: `AKer.norm`
Asymmetric Cosine Kernel: `AKer.cos`
Asymmetric Epanechnikov Kernel: `AKer.epa`
Asymmetric Triweight Kernel: `AKer.tri`
Asymmetric Quartic Kernel: `AKer.quar`
Asymmetric Uniform Kernel: `AKer.unif`

Value

Returns asymmetric kernel.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Examples

```
y=qnorm(seq(.1,.9,len=100))
a<-Kernel.asymmetric(u=y)
b<-Kernel.asymmetric(type.Ker="AKer.tri",u=y)
c=AKer.cos(y)
```

Kernel.integrate *Integrate Smoothing Kernels.*

Description

Represent integrate kernels: normal, cosine, triweight, quartic and uniform.

Usage

```
Kernel.integrate(u, Ker = Ker.norm, a = -1)
```

Arguments

u	data
Ker	Type of Kernel. By default normal kernel.
a	Lower limit of integration.

Details

Type of integrate kernel:

Integrate Normal Kernel: `IKer.norm`
Integrate Cosine Kernel: `IKer.cos`
Integrate Epanechnikov Kernel: `IKer.epa`
Integrate Triweight Kernel: `IKer.tri`
Integrate Quartic Kernel: `IKer.quar`
Integrate Uniform Kernel: `IKer.unif`

Value

Returns integrate kernel.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

See Also

See Also as: Kernel and integrate.

Examples

```
y=qnorm(seq(.1,.9,len=100))
d=IKer.tri(y)
e=IKer.cos(y)
e2=Kernel.integrate(u=y,Ker=Ker.cos)
e-e2
f=IKer.epa(y)
f2=Kernel.integrate(u=y,Ker=Ker.epa)
f-f2
plot(d,type="l",ylab="Integrate Kernel")
lines(e,col=2,type="l")
lines(f,col=4,type="l")
```

---

kmeans.center.ini *K-Means Clustering for functional data*

Description

Perform k-means clustering on functional data.

Usage

```
kmeans.center.ini(
  fdataobj,
  ncl = 2,
  metric = metric.lp,
  draw = TRUE,
  method = "sample",
  max.iter = 100,
  max.comb = 1e+06,
  par.metric = NULL,
  ...
)

kmeans.fd(
  fdataobj,
  ncl = 2,
  metric = metric.lp,
  dfunc = func.trim.FM,
  max.iter = 100,
  par.metric = NULL,
  par.dfunc = list(trim = 0.05),
  method = "sample",
  cluster.size = 5,
  draw = TRUE,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`ncl`	See details section.
`metric`	Metric function, by default `metric.lp`.
`draw`	=TRUE, draw the curves in the color of the centers.
`method`	Method for selecting initial centers. If `method=`*"Sample"* (by default) takes `n` times a random selection by the `ncl` centers. The `ncl` curves with greater distance are the initial centers. If `method=`*"Exact"* calculated all combinations (if < 1e+6) of `ncl` centers. The `ncl` curves with greater distance are the initial centers (this method may be too slow).
`max.iter`	Maximum number of iterations for the detection of centers.
`max.comb`	Maximum number of initial selection of centers (only used when `method="exact"`).
`par.metric`	List of arguments to pass to the `metric` function.
`...`	Further arguments passed to or from other methods.
`dfunc`	Type of depth measure, by default FM depth.
`par.dfunc`	List of arguments to pass to the `dfunc` function .
`cluster.size`	Minimum cluster size (by default is 5). If a cluster has fewer curves, it is eliminated and the process is continued with a less cluster.

Details

The method searches the locations around which are grouped data (for a predetermined number of groups).

If `ncl=NULL`, randomizes the initial centers, `ncl=2` using `kmeans.center.ini` function.
If `ncl` is an integer, indicating the number of groups to classify,
are selected `ncl` initial centers using `kmeans.center.ini` function.
If `ncl` is a vector of integers, indicating the position of the initial centers with `length(ncl)` equal to number of groups.
If `ncl` is a `fdata` class objecct, `ncl` are the initial centers curves with `nrow(ncl)` number of groups.

Value

Return:

- `cluster`: Indexes of groups assigned.
- `centers`: Curves centers.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Hartigan, J. A. and Wong, M. A. (1979). *A K-means clustering algorithm.* Applied Statistics 28, 100 \-108.

See Also

See Also generic kmeans function.

Examples

```
## Not run:
library(fda.usc)
data(phoneme)
mlearn <- phoneme$learn[c(1:50,101:150,201:250),]
# Unsupervised classification
out.fd1 <- kmeans.fd(mlearn,ncl=3,draw=TRUE)
out.fd2 <- kmeans.fd(mlearn,ncl=3,draw=TRUE,method="exact")
# Different Depth function
ind <- c(17,77,126)
out.fd3 <- kmeans.fd(mlearn,ncl=mlearn[ind,],draw=FALSE,
dfunc <- func.trim.FM,par.dfunc=list(trim=0.1))
out.fd4 <- kmeans.fd(mlearn, ncl=mlearn[ind,], draw=FALSE,
dfunc = func.med.FM)
group <- c(rep(1,50), rep(2,50),rep(3,50))
table(out.fd4$cluster, group)

## End(Not run)
```

`ldata` *ldata class definition and utilities*

Description

ldata is a list with two type of objects:

- `df` is a data frame with the multivariate data with n rows.
- `...` fdata objects of class `fdata` with n rows.

Usage

```
ldata(df, ..., mfdata)

## S3 method for class 'ldata'
names(x)

is.ldata(x)

## S3 method for class 'ldata'
x[i, row = FALSE]

## S3 method for class 'ldata'
subset(x, subset, ...)

## S3 method for class 'ldata'
plot(x, ask = FALSE, color, var.name, ...)
```

Arguments

`df`	data frame
`...`	Further arguments passed to methods.
`mfdata`	list of fdata objects
`i`	index
`row`	logical If `FALSE` (by default), `i` index selects the variables. If `TRUE`, `i` index selects the observations.
`subset`	subset
`ask`	logilcal If TRUE (and the R session is interactive) the user is asked for input, before a new figure is drawn.
`color`	colors to interpolate; must be a valid argument to `colorRampPalette`.
`var.name`	name of continuous univariate variable used in `color` argument
`ldata`, `x`	object of class `ldata`

Examples

```
data(tecator)
ab0 <- tecator$absorp.fdata
ab1 <- fdata.deriv(ab0)
ab2 <- fdata.deriv(ab0,nderiv=2)
ldat<-ldata(tecator$y,ab1=ab1,ab2=ab2)
is.ldata(ldat)
class(ldat)
plot(ldat[[1]])
plot(ldat[[2]])
# plot(ldat)
# plot(ldat,var.name="Fat")
```

LMDC.select *Impact points selection of functional predictor and regression using local maxima distance correlation (LMDC)*

Description

LMDC.select function selects impact points of functional predictior using local maxima distance correlation (LMDC) for a scalar response given.
LMDC.regre function fits a multivariate regression method using the selected impact points like covariates for a scalar response.

Usage

```
LMDC.select(
  y,
  covar,
  data,
  tol = 0.06,
  pvalue = 0.05,
  plot = FALSE,
  local.dc = TRUE,
  smo = FALSE,
  verbose = FALSE
)

LMDC.regre(
  y,
  covar,
  data,
  newdata,
  pvalue = 0.05,
  method = "lm",
  par.method = NULL,
  plot = FALSE,
  verbose = FALSE
)
```

Arguments

`y`	name of the response variable.
`covar`	vector with the names of the covaviables (or points of impact) with length `p`.
`data`	data frame with length n rows and at least p + 1 columns, containing the scalar response and the potencial p covaviables (or points of impact) in the model.
`tol`	Tolerance value for distance correlation and imapct point.
`pvalue`	pvalue of bias corrected distance correlation t-test.

`plot`	logical value, if TRUE plots the distance correlation curve for each covariate in multivariate case and in each discretization points (argvals) in the functional case.
`local.dc`	Compute local distance correlation.
`smo`	logical. If TRUE, the curve of distance correlation computed in the impact points is smoothed using B-spline representation with a suitable number of basis elements.
`verbose`	print iterative and relevant steps of the procedure.
`newdata`	An optional data frame in which to look for variables with which to predict.
`method`	Name of regression method used, see details. This argument is used in do.call function like "what" argument.
`par.method`	List of parameters used to call the method. This argument is used in do.call function like "args" argument.

Details

String of characters corresponding to the name of the regression method called. Model available options:

- `"lm"`: Step-wise lm regression model (uses lm function, stats package). Recommended for linear models, test linearity using. `flm.test` function.
- `"gam"`: Step-wise gam regression model (uses gam function, mgcv package). Recommended for non-linear models.

Models that use the indicated function of the required package:

- `"svm"`: Support vector machine (svm function, e1071 package).
- `"knn"`: k-nearest neighbor regression (knnn.reg function, FNN package).
- `"lars"`: Least Angle Regression using Lasso (lars function, lars package).
- `"glmnet"`: Lasso and Elastic-Net Regularized Generalized Linear Models (glmnet and cv.glmnet function, glmnet package).
- `"rpart"`: Recursive partitioning for regression a (rpart function, rpart package).
- `"flam"`: Fit the Fused Lasso Additive Model for a Sequence of Tuning Parameters (flam function, flam package).
- `"novas"`: NOnparametric VAriable Selection (code available in `https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NOVAS/novas-routines.R`).
- `"cosso"`: Fit Regularized Nonparametric Regression Models Using COSSO Penalty (cosso function, cosso package).
- `"npreg"`: kernel regression estimate of a one (1) dimensional dependent variable on p-variate explanatory data (npreg function, np package).
- `"mars"`: Multivariate adaptive regression splines (mars function, mda package).
- `"nnet"`: Fit Neural Networks (nnet function, nnet package).
- `"lars"`: Fits Least Angle Regression, Lasso and Infinitesimal Forward Stagewise regression models (lars function, lars package).

Value

`LMDC.select`	function returns a list of two elements:
`cor`	the value of distance correlation for each covariate.
`maxLocal`	index or locations of local maxima distance correlations.
`LMDC.regre`	function returns a list of folowing elements:
`model`	object corresponding to the estimated method using the selected variables.
`xvar`	names of selected variables (impact points).
`edf`	effective degrees of freedom.
`nvar`	number of selected variables (impact points).

Author(s)

Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ordonez, C., Oviedo de la Fuente, M., Roca-Pardinas, J., Rodriguez-Perez, J. R. (2018). Determining optimum wavelengths for leaf water content estimation from reflectance: A distance correlation approach. *Chemometrics and Intelligent Laboratory Systems*. 173,41-50 doi:10.1016/j.chemolab.2017.12.001.

See Also

See Also as: lm, gam, `dcor.xy`.

Examples

```
## Not run:
data(tecator)
absorp=fdata.deriv(tecator$absorp.fdata,2)
ind=1:129
x=absorp[ind,]
y=tecator$y$Fat[ind]
newx=absorp[-ind,]
newy=tecator$y$Fat[-ind]

## Functional PC regression
res.pc=fregre.pc(x,y,1:6)
pred.pc=predict(res.pc,newx)

# Functional regression with basis representation
res.basis=fregre.basis.cv(x,y)
pred.basis=predict(res.basis[[1]],newx)

# Functional nonparametric regression
res.np=fregre.np.cv(x,y)
pred.np=predict(res.np,newx)
```

```
dat    <- data.frame("y"=y,x$data)
newdat <- data.frame("y"=newy,newx$data)

res.gam=fregre.gsam(y~s(x),data=list("df"=dat,"x"=x))
pred.gam=predict(res.gam,list("x"=newx))

dc.raw <- LMDC.select("y",data=dat, tol = 0.05, pvalue= 0.05,
                      plot=F, smo=T,verbose=F)
covar <- paste("X",dc.raw$maxLocal,sep="")
# Preselected design/impact points
covar
ftest<-flm.test(dat[,-1],dat[,"y"], B=500, verbose=F,
    plot.it=F,type.basis="pc",est.method="pc",p=4,G=50)

if (ftest$p.value>0.05) {
  # Linear relationship, step-wise lm is recommended
  out <- LMDC.regre("y",covar,dat,newdat,pvalue=.05,
              method ="lm",plot=F,verbose=F)
} else {
 # Non-Linear relationship, step-wise gam is recommended
  out <- LMDC.regre("y",covar,dat,newdat,pvalue=.05,
              method ="gam",plot=F,verbose=F) }

# Final  design/impact points
out$xvar

# Predictions
mean((newy-pred.pc)^2)
mean((newy-pred.basis)^2)
mean((newy-pred.np)^2)
mean((newy-pred.gam)^2)
mean((newy-out$pred)^2)

## End(Not run)
```

MCO *Mithochondiral calcium overload (MCO) data set*

Description

The mithochondiral calcium overload (MCO) was measured in two groups (control and treatment) every 10 seconds during an hour in isolated mouse cardiac cells. In fact, due to technical reasons, the original experiment [see Ruiz-Meana et al. (2000)] was performed twice, using both the "intact", original cells and "permeabilized" cells (a condition related to the mitochondrial membrane).

Format

Elements of MCO:
`..$intact`: `fdata` class object with “intact cells”curves,

- `"data"`: Matrix of class `fdata` with 89 intact cells curves (rows) measured every 10 seconds during an hour in isolated mouse cardiac cell.
- `"argvals"`, 360 discretization points from seond 0 to 3590.
- `"rangeval"`: range(`"argvals"`).
- `"names"` list with: `main` an overall title "Control Intact Treatment", `xlab` title for `x` axis "seconds" and `ylab` title for `y` axis "Ca".

`..$classintact`: Factor levels of “intact cells” curves: "1" control group and "2" treatment group.

`..$permea`: `fdata` class object with “permeabilized cells” curves (whose membrane has been removed),

- `"data"`: Matrix of class `fdata` with 90 permeabilizzed cells curves (rows) measured every 10 seconds during an hour in isolated mouse cardiac cell.
- `"argvals"`, 360 discretization points from seond 0 to 3590.
- `"rangeval"`: range(`"argvals"`).
- `"names"` list with: `main` an overall title "Control Intact Treatment", `xlab` title for `x` axis "seconds" and `ylab` title for `y` axis "Ca".

`..$classpermea`: Factor levels of “permeabilized cells” curves: "1" control group and "2" treatment group.

Note

The structure of the curves during the initial period (first 180 seconds) of the experiment shows a erratic behavior (not very relevant in the experiment context) during this period.

References

Ruiz–Meana M, Garcia-Dorado D, Pina P, Inserte J, Agullo L, Soler–Soler J. Cariporide preserves mitochondrial proton gradient and delays ATP depletion in cardiomyocytes during ischemic conditions. *American Journal Physiology Heart Circulatori Physiology*. 2003 Sep;285(3):H999–1006.

Examples

```
data(MCO)
names(MCO)
par(mfrow=c(1,2))
plot.fdata(MCO$intact, col=MCO$classintact)
plot.fdata(MCO$permea, col=MCO$classintact)
```

`metric.dist` *Distance Matrix Computation*

Description

This function computes the distances between the rows of a data matrix by using the specified distance measure.

This function returns a distance matrix by using `dist` function.
The matrix dimension is (`n1` x `n1`) if `y=NULL`, (`n1` x `n2`) otherwise.

Usage

```
metric.dist(x, y = NULL, method = "euclidean", p = 2, dscale = 1, ...)
```

Arguments

`x`	Data frame 1. The dimension is (`n1` x `m`).
`y`	Data frame 2. The dimension is (`n2` x `m`).
`method`	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
`p`	The power of the Minkowski distance.
`dscale`	If scale is a numeric, the distance matrix is divided by the scale value. If scale is a function (as the mean for example) the distance matrix is divided by the corresponding value from the output of the function.
`...`	Further arguments passed to `dist` function.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

See Also

See also `dist` for multivariate date case and `metric.lp` `for functional data case`

Examples

```
## Not run:
data(iris)
d<-metric.dist(iris[,1:4])
matplot(d,type="l",col=as.numeric(iris[,5]))

## End(Not run)
```

metric.DTW *DTW: Dynamic time warping*

Description

Computes distances time warping for functional data

Usage

```
metric.DTW(fdata1, fdata2 = NULL, p = 2, w = min(ncol(fdata1), ncol(fdata2)))

metric.WDTW(
  fdata1,
  fdata2 = NULL,
  p = 2,
  w = min(ncol(fdata1), ncol(fdata2)),
  wmax = 1,
  g = 0.05
)

metric.TWED(fdata1, fdata2 = NULL, p = 2, lambda = 1, nu = 0.05)
```

Arguments

`fdata1`	Functional data 1 or curve 1. If `fdata` class, the dimension of `fdata1$data` object is (`n1` x `m`), where `n1` is the number of curves and `m` are the points observed in each curve.
`fdata2`	Functional data 2 or curve 2. If `fdata` class, the dimension of `fdata2$data` object is (`n2` x `m`), where `n2` is the number of curves and `m` are the points observed in each curve.
`p`	Lp norm, by default it uses `p = 2`
`w`	Vector of weights with length `m`, If `w = 1` approximates the metric Lp by Simpson's rule. By default it uses `w = 1`
`wmax`	`numeric` maximum value of weight, (1 by default)
`g`	`numeric` `g=0` (constant), `0.05` (linear) by default, 0.25 `sigmoid`, 3 two weight values
`lambda`	`numeric` lambda value (0 by default)
`nu`	`numeric` constant value, (0 by default)

Details

Three optins:

- DTW: Dynamic time warping
- WDTW: Weight Dynamic time warping
- TWED: twed

Value

DTW matrix

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Jeong, Y. S., Jeong, M. K., & Omitaomu, O. A. (2011). Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9), 2231-2240

See Also

See also semimetric.basis and semimetric.NPFDA

Examples

```
## Not run:
data(tecator)
metric.DTW(tecator$absorp.fdata[1:4,])
ab=tecator[[1]]
D1=fda.usc:::DTW(ab$data[1,],ab$data[2,],p=2)
aa1=fda.usc:::findPath(D1$D)
D2=fda.usc:::DTW(ab$data[1,],ab$data[2,],p=2,w=5)
aa2=fda.usc:::findPath(D2$D)
D3=fda.usc:::WDTW(ab$data[1,],ab$data[2,],p=2,g=0.05)
aa3=fda.usc:::findPath(D3$D)
D4=fda.usc:::TWED(ab$data[1,],ab$data[2,],p=2,lambda=0,nu=0)
aa4=fda.usc:::findPath(D4$D)
par(mfrow=c(2,2))
plot(c(ab[1:2]))
segments(ab$argvals[aa1[,1]],ab[1]$data[aa1[,1]],ab$argvals[aa1[,2]],ab[2]$data[aa1[,2]])
plot(c(ab[1:2]))
segments(ab$argvals[aa2[,1]],ab[1]$data[aa2[,1]],ab$argvals[aa2[,2]],ab[2]$data[aa2[,2]],col=2)
plot(c(ab[1:2]))
segments(ab$argvals[aa3[,1]],ab[1]$data[aa3[,1]],ab$argvals[aa3[,2]],ab[2]$data[aa3[,2]],col=3)
plot(c(ab[1:2]))
segments(ab$argvals[aa4[,1]],ab[1]$data[aa4[,1]],ab$argvals[aa4[,2]],ab[2]$data[aa4[,2]],col=4)

## End(Not run)
```

metric.hausdorff *Compute the Hausdorff distances between two curves.*

Description

Hausdorff distance is the greatest of all the distances from a point in one curve to the closest point in the other curve (been closest the euclidean distance).

Usage

```
metric.hausdorff(fdata1, fdata2 = fdata1)
```

Arguments

`fdata1`	Curves 1 of `fdata` class. The dimension of `fdata1` object is (`n1` x `m`), where `n1` is the number of points observed in `t` coordinates with lenght `m`.
`fdata2`	Curves 2 of `fdata` class. The dimension of `fdata2` object is (`n2` x `m`), where `n2` is the number of points observed in `t` coordinates with lenght `m`.

Details

Let $G(X) = \left\{(t, X(t)) \in R^2\right\}$ and $G(Y) = \left\{(t, Y(t)) \in R^2\right\}$ be two graphs of the considered curves X and Y respectively, the Hausdorff distance $d_H(X, Y)$ is defined as,

$$d_H(X,Y) = max\left\{sup_{x\in G(X)} inf_{y\in G(Y)} d_2(x,y), sup_{y\in G(Y)} inf_{x\in G(X)} d_2(x,y)\right\},$$

where $d_2(x, y)$ is the euclidean distance, see `metric.lp`.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

Examples

```
## Not run:
data(poblenou)
nox<-poblenou$nox[1:6]
# Hausdorff vs maximum distance
out1<-metric.hausdorff(nox)
out2<-metric.lp(nox,lp=0)
out1
out2
par(mfrow=c(1,3))
plot(nox)
plot(hclust(as.dist(out1)))
plot(hclust(as.dist(out2)))

## End(Not run)
```

`metric.kl` *Kullback–Leibler distance*

Description

Measures the proximity between two groups of densities (of class `fdata`) by computing the Kullback–Leibler distance.

Usage

```
metric.kl(fdata1, fdata2 = NULL, symm = TRUE, base = exp(1), eps = 1e-10, ...)
```

Arguments

`fdata1`	Functional data 1 (`fdata` class) with the densities. The dimension of `fdata1` object is (`n1 x m`), where `n1` is the number of densities and `m` is the number of coordinates of the points where the density is observed.
`fdata2`	Functional data 2 (`fdata` class) with the densities. The dimension of `fdata2` object is (`n2 x m`).
`symm`	If `TRUE` the symmetric K–L distance is computed, see details section.
`base`	The logarithm base used to compute the distance.
`eps`	Tolerance value.
`...`	Further arguments passed to or from other methods.

Details

Kullback–Leibler distance between $f(t)$ and $g(t)$ is

$$metric.kl(f(t),g(t)) = \int_a^b f(t)log\left(\frac{f(t)}{g(t)}\right)dt$$

where t are the `m` coordinates of the points where the density is observed (the `argvals` of the `fdata` object).

The Kullback–Leibler distance is asymmetric,

$$metric.kl(f(t),g(t)) \neq metric.kl(g(t),f(t))$$

A symmetry version of K–L distance (by default) can be obtained by

$$0.5\left(metric.kl(f(t),g(t)) + metric.kl(g(t),f(t))\right)$$

If $(f_i(t) = 0 \;\&\; g_j(t) = 0) \Longrightarrow metric.kl(f(t),g(t)) = 0$.

If $|f_i(t) - g_i(t)| \leq \epsilon \Longrightarrow f_i(t) = f_i(t) + \epsilon$, where ϵ is the tolerance value (by default `eps=1e-10`).

The coordinates of the points where the density is observed (discretization points t) can be equally spaced (by default) or not.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente `<manuel.oviedo@udc.es>`

References

Kullback, S., Leibler, R.A. (1951). *On information and sufficiency.* Annals of Mathematical Statistics, 22: 79-86

See Also

See also `metric.lp` and `fdata`

Examples

```
## Not run:
n<-201
tt01<-seq(0,1,len=n)
rtt01<-c(0,1)
x1<-dbeta(tt01,20,5)
x2<-dbeta(tt01,21,5)
y1<-dbeta(tt01,5,20)
y2<-dbeta(tt01,5,21)
xy<-fdata(rbind(x1,x2,y1,y2),tt01,rtt01)
plot(xy)
round(metric.kl(xy,xy,eps=1e-5),6)
round(metric.kl(xy,eps=1e-5),6)
round(metric.kl(xy,eps=1e-6),6)
round(metric.kl(xy,xy,symm=FALSE,eps=1e-5),6)
round(metric.kl(xy,symm=FALSE,eps=1e-5),6)

plot(c(fdata(y1[1:101]),fdata(y2[1:101])))
metric.kl(fdata(x1))
metric.kl(fdata(x1),fdata(x2),eps=1e-5,symm=F)
metric.kl(fdata(x1),fdata(x2),eps=1e-6,symm=F)
metric.kl(fdata(y1[1:101]),fdata(y2[1:101]),eps=1e-13,symm=F)
metric.kl(fdata(y1[1:101]),fdata(y2[1:101]),eps=1e-14,symm=F)

## End(Not run)
```

`metric.ldata` *Distance Matrix Computation for ldata and mfdata class object*

Description

This function computes the distances between the list elements

Usage

```
metric.ldata(
  ldata1,
  ldata2 = NULL,
  include = "all",
  exclude = "none",
  metric,
  par.metric = NULL,
  w,
  method = "none"
)
```

Arguments

`ldata1`	List with of fdata objects and a data.frame object calle 'df'.
`ldata2`	List with of fdata objects and a data.frame object calle 'df'.
`include`	vector with the name of variables to use
`exclude`	vector with the name of variables to not use
`metric`	Type of metric to combine, if 'none', the function no combine and return a list o distances for each variable included
`par.metric`	List of metric parameters for each variable included
`w`	weights to combine the metric (if metric is not 'none')
`method`	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".

Details

This function returns a distance matrix by using `metric.lp` function for `fdata` objects and `metric.dist` function for `vector` and `matric` objects.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manul.oviedo@usc.es`>

See Also

See also `dist` for multivariate date case and `metric.lp` `for functional data case`

Examples

```
## Not run:
data(tecator)
names(tecator)[2] <- "df"
# Example 1 (list of distances)
ldist <- metric.ldata(tecator,method="none")
lapply(ldist,names)
# Example 2 (combined metric)
mdist <- metric.ldata(tecator,method="euclidean")
dim(mdist)

## End(Not run)
```

metric.lp — *Approximates Lp-metric distances for functional data.*

Description

Measures the proximity between the functional data and curves approximating Lp-metric. If `w = 1` approximates the Lp-metric by Simpson's rule. By default it uses `lp = 2` and weights `w = 1`.

Usage

```
metric.lp(fdata1, fdata2 = NULL, lp = 2, w = 1, dscale = 1, ...)
```

Arguments

`fdata1`	Functional data 1 or curve 1. If `fdata` class, the dimension of `fdata1$data` object is (`n1 x m`), where `n1` is the number of curves and `m` are the points observed in each curve.
`fdata2`	Functional data 2 or curve 2. If `fdata` class, the dimension of `fdata2$data` object is (`n2 x m`), where `n2` is the number of curves and `m` are the points observed in each curve.
`lp`	Lp norm, by default it uses `lp = 2`
`w`	Vector of weights with length `m`, If `w = 1` approximates the metric Lp by Simpson's rule. By default it uses `w = 1`
`dscale`	If scale is a numeric, the distance matrix is divided by the scale value. If scale is a function (as the mean for example) the distance matrix is divided by the corresponding value from the output of the function.
`...`	Further arguments passed to or from other methods.

Details

By default it uses the L2-norm with `lp = 2`.

$$Let\ \ f(x) = fdata1(x) - fdata2(x)$$

$$\|f\|_p = \left(\frac{1}{\int_a^b w(x)dx} \int_a^b |f(x)|^p \, w(x)dx \right)^{1/p}$$

The observed points on each curve are equally spaced (by default) or not.

The L∞-norm is computed with `lp = 0`.

$$d(fdata1(x), fdata2(x))_{\infty} = sup\,|fdata1(x) - fdata2(x)|$$

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See also semimetric.basis and semimetric.NPFDA

Examples

```
## Not run:
# INFERENCE PHONDAT
data(phoneme)
mlearn<-phoneme$learn[1:100]
mtest<-phoneme$test[1:100]
glearn<-phoneme$classlearn[1:100]
gtest<-phoneme$classtest[1:100]
# Matrix of distances of curves of DATA1
mdist1<-metric.lp(mlearn)

# Matrix of distances between curves of DATA1 and curves of DATA2
mdist2<-metric.lp(mlearn,mtest,lp=2)
# mdist with L1 norm and weigth=v
v=dnorm(seq(-3,3,len=dim(mlearn)[2]))
mdist3<-metric.lp(mlearn,mtest,lp=1,w=v)
plot(1:100,mdist2[1,],type="l",ylim=c(1,max(mdist3[1,])))
lines(mdist3[1,],type="l",col="2")

# mdist with mlearn with different discretization points.
# mlearn2=mlearn
# mlearn2[["argvals"]]=seq(0,1,len=150)
# mdist5<-metric.lp(mlearn,mlearn2)
# mdist6<-metric.lp(mlearn2,mlearn)
# sum(mdist5-mdist6)
# sum(mdist1-mdist6)

x<-seq(0,2*pi,length=1001)
fx<-fdata(sin(x)/sqrt(pi),x)
fx0<-fdata(rep(0,length(x)),x)
metric.lp(fx,fx0)
# The same
integrate(function(x){(abs(sin(x)/sqrt(pi))^2)},0,2*pi)

## End(Not run)
```

mfdata — *mfdata class definition and utilities*

Description

mfdata is a list with fdata type of object:

- `...` fdata objects of class `fdata` with n rows.

Usage

```
mfdata(...)

## S3 method for class 'mfdata'
names(x)

## S3 method for class 'mfdata'
subset(x, subset, ...)
```

Arguments

`...`	Further arguments passed to methods.
`x`	object of class `mfdata`
`subset`	subset

Examples

```
data(tecator)
ab0 <- tecator$absorp.fdata
ab1 <- fdata.deriv(ab0)
ab2 <- fdata.deriv(ab0,nderiv=2)
mdat <- mfdata(ab0, ab1, ab2)
is.ldata(mdat)
class(mdat)
plot(mdat[[1]])
plot(mdat[[2]])
plot(mdat)
```

na.omit.fdata — *A wrapper for the na.omit and na.fail function for fdata object*

Description

`na.fail` returns the object if it does not contain any missing values, and signals an error otherwise. `na.omit` returns the object with incomplete cases removed. If `na.omit.fdata` removes cases, the row numbers of the cases form the `"na.action"` attribute of the result, of class `"omit"`, see generic function `na.omit`.

Usage

```
## S3 method for class 'fdata'
na.omit(object, ...)

## S3 method for class 'fdata'
na.fail(object, ...)
```

Arguments

`object`	an `fdata` object.
`...`	further potential arguments passed to methods.

Value

The value returned from `omit` is a `fdata` object with incomplete cases removed.

Author(s)

Manuel Febrero Bande and Manuel Oviedo

Examples

```
## Not run:
fdataobj<-fdata(MontrealTemp)
fdataobj$data[3,3]<-NA
fdataobj$data[10,]<-NA
fdastaobj2<-na.omit(fdataobj)

## End(Not run)
```

`norm.fdata` *Approximates Lp-norm for functional data.*

Description

Approximates Lp-norm for functional data (fdata) object using metric or semimetric functions. Norm for functional data using by default Lp-metric.

Usage

```
norm.fdata(fdataobj, metric = metric.lp, ...)

norm.fd(fdobj)
```

Arguments

fdataobj	fdata class object.
metric	Metric function, by default metric.lp.
...	Further arguments passed to or from other methods.
fdobj	Functional data or curves of fd class.

Details

By default it computes the L2-norm with `p = 2` and weights `w` with length=`(m-1)`.

$$Let\ \ f(x) = fdataobj(x)$$

$$\|f\|_p = \left(\frac{1}{\int_a^b w(x)dx} \int_a^b |f(x)|^p \, w(x)dx \right)^{1/p}$$

The observed points on each curve are equally spaced (by default) or not.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See also metric.lp and norm
Alternative method: inprod of fda-package

Examples

```
## Not run:
x<-seq(0,2*pi,length=1001)
fx1<-sin(x)/sqrt(pi)
fx2<-cos(x)/sqrt(pi)
argv<-seq(0,2*pi,len=1001)
fdat0<-fdata(rep(0,len=1001),argv,range(argv))
fdat1<-fdata(fx1,x,range(x))
metric.lp(fdat1)
metric.lp(fdat1,fdat0)
norm.fdata(fdat1)
# The same
integrate(function(x){(abs(sin(x)/sqrt(pi))^2)},0,2*pi)
integrate(function(x){(abs(cos(x)/sqrt(pi))^2)},0,2*pi)

bspl1<- create.bspline.basis(c(0,2*pi),21)
fd.bspl1 <- fd(basisobj=bspl1)
fd.bspl2<-fdata2fd(fdat1,nbasis=21)
norm.fd(fd.bspl1)
norm.fd(fd.bspl2)

## End(Not run)
```

ops.fda.usc *ops.fda.usc Options Settings*

Description

Set or query graphical and prompt output parameters. Allow the user to set and examine a variety of global or local options which affect the way in which fda.usc functions computes and displays its results.

Usage

```
ops.fda.usc(
  verbose = FALSE,
  trace = FALSE,
  warning = FALSE,
  ncores = NULL,
  int.method = "TRAPZ",
  reset = FALSE,
  eps = as.double(.Machine[[1]] * 10)
)
```

Arguments

`verbose`	`logical`. Should R report extra information on progress? Set to `TRUE` by the command-line option –verbose.
`trace`	`logical`. Show internal information of procedure.
`warning`	`logical`: If true, warnings are shown.
`ncores`	integer. Number of CPU cores on the current host.
`int.method`	see `method` argument in `int.simpson` function.
`reset`	`logical`. If `TRUE` creates a new Parallel Socket Cluster (ncores>1) or a sequential parallel backend (ncores=1). It is useful when worker initialization failed or after a crush.
`eps`	epsilon parameter.

Author(s)

Manuel Oviedo de la Fuente (<`manuel.oviedo@udc.es`>).

Examples

```
## Not run:
# If worker initialization failed, please execute this code
 ncores <- max(parallel::detectCores() -1,1)
 if (ncores==1) {
     foreach::registerDoSEQ()
 }  else{
```

```
 cl <-  suppressWarnings(parallel::makePSOCKcluster(ncores ))
 doParallel::registerDoParallel(cl)
 }
 ops.fda.usc()

## End(Not run)
```

`optim.basis` *Select the number of basis using GCV method.*

Description

Functional data estimation via basis representation using cross-validation (CV) or generalized cross-validation (GCV) method with a roughness penalty.

Usage

```
optim.basis(
  fdataobj,
  type.CV = GCV.S,
  W = NULL,
  lambda = 0,
  numbasis = floor(seq(ncol(fdataobj)/16, ncol(fdataobj)/2, len = 10)),
  type.basis = "bspline",
  par.CV = list(trim = 0, draw = FALSE),
  verbose = FALSE,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`type.CV`	Type of cross-validation. By default generalized cross-validation (GCV) method.
`W`	Matrix of weights.
`lambda`	A roughness penalty. By default, no penalty `lambda=0`.
`numbasis`	Number of basis to use.
`type.basis`	Character string which determines type of basis. By default *"bspline"*.
`par.CV`	List of parameters for type.CV: trim, the alpha of the trimming and `draw=TRUE`.
`verbose`	If `TRUE` information about GCV values and input parameters is printed. Default is `FALSE`.
`...`	Further arguments passed to or from other methods. Arguments to be passed by default to create.basis.

Details

Provides the least GCV for functional data for a list of number of basis `num.basis` and lambda values `lambda`. You can define the type of CV to use with the `type.CV`, the default is used `GCV.S`.

Smoothing matrix is performed by `S.basis`. `W` is the matrix of weights of the discretization points.

Value

- `gcv`: Returns GCV values calculated for input parameters.
- `fdataobj`: Matrix of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve.
- `fdata.est`: Estimated `fdata` class object.
- `numbasis.opt`: `numbasis` value that minimizes CV or GCV method.
- `lambda.opt`: `lambda` value that minimizes CV or GCV method.
- `basis.opt`: `basis` for the minimum CV or GCV method.
- `S.opt`: Smoothing matrix for the minimum CV or GCV method.
- `gcv.opt`: Minimum of CV or GCV method.
- `lambda`: A roughness penalty. By default, no penalty `lambda=0`.
- `numbasis`: Number of basis to use.
- `verbose`: If `TRUE` information about GCV values and input parameters is printed. Default is `FALSE`.

#' @author Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

Note

min.basis deprecated.

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as `S.basis`.
Alternative method: `optim.np`

Examples

```
## Not run:
a1<-seq(0,1,by=.01)
a2=rnorm(length(a1),sd=0.2)
f1<-(sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
nc<-50
np<-length(f1)
tt=1:101
S<-S.NW(tt,2)
mdata<-matrix(NA,ncol=np,nrow=50)
for (i in 1:50) mdata[i,]<- (sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
mdata<-fdata(mdata)
nb<-floor(seq(5,29,len=5))
l<-2^(-5:15)
out<-optim.basis(mdata,lambda=l,numbasis=nb,type.basis="fourier")
matplot(t(out$gcv),type="l",main="GCV with fourier basis")

# out1<-optim.basis(mdata,type.CV = CV.S,lambda=l,numbasis=nb)
# out2<-optim.basis(mdata,lambda=l,numbasis=nb)

# variance calculations
y<-mdata
i<-3
z=qnorm(0.025/np)
fdata.est<-out$fdata.est
var.e<-Var.e(mdata,out$S.opt)
var.y<-Var.y(mdata,out$S.opt)
var.y2<-Var.y(mdata,out$S.opt,var.e)

# estimated fdata and point confidence interval
upper.var.e<-out$fdata.est[["data"]][i,]-z*sqrt(diag(var.e))
lower.var.e<-out$fdata.est[["data"]][i,]+z*sqrt(diag(var.e))
dev.new()
plot(y[i,],lwd=1,ylim=c(min(lower.var.e),max(upper.var.e)))
lines(out$fdata.est[["data"]][i,],col=gray(.1),lwd=1)
lines(out$fdata.est[["data"]][i,]+z*sqrt(diag(var.y)),col=gray(0.7),lwd=2)
lines(out$fdata.est[["data"]][i,]-z*sqrt(diag(var.y)),col=gray(0.7),lwd=2)
lines(upper.var.e,col=gray(.3),lwd=2,lty=2)
lines(lower.var.e,col=gray(.3),lwd=2,lty=2)
legend("top",legend=c("Var.y","Var.error"), col = c(gray(0.7),
gray(0.3)),lty=c(1,2))

## End(Not run)
```

`optim.np` *Smoothing of functional data using nonparametric kernel estimation*

Description

Smoothing of functional data using nonparametric kernel estimation with cross-validation (CV) or generalized cross-validation (GCV) methods.

Usage

```
optim.np(
  fdataobj,
  h = NULL,
  W = NULL,
  Ker = Ker.norm,
  type.CV = GCV.S,
  type.S = S.NW,
  par.CV = list(trim = 0, draw = FALSE),
  par.S = list(),
  correl = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`fdataobj`	`fdata` class object.
`h`	Smoothing parameter or bandwidth.
`W`	Matrix of weights.
`Ker`	Type of kernel used, by default normal kernel.
`type.CV`	Type of cross-validation. By default generalized cross-validation (GCV) method. Possible values are *GCV.S* and *CV.S*
`type.S`	Type of smothing matrix `S`. By default `S` is calculated by Nadaraya-Watson kernel estimator (`S.NW`). Possible values are `S.KNN`, `S.LLR`, `S.LPR` and `S.LCR`.
`par.CV`	List of parameters for type.CV: `trim`, the alpha of the trimming and `draw=TRUE`
`par.S`	List of parameters for `type.S`: `tt` for argvals, `h` for bandwidth, `Ker` for kernel, etc.
`correl`	logical. If `TRUE` the bandwidth parameter `h` is computed following the procedure described for De Brabanter et al. (2018). (option avalaible since v1.6.0 version)
`verbose`	If `TRUE` information about GCV values and input parameters is printed. Default is `FALSE`.
`...`	Further arguments passed to or from other methods. Arguments to be passed for kernel method.

Details

Calculate the minimum GCV for a vector of values of the smoothing parameter `h`. Nonparametric smoothing is performed by the kernel function. The type of kernel to use with the parameter `Ker` and the type of smothing matrix `S` to use with the parameter `type.S` can be selected by the user, see function `Kernel`. W is the matrix of weights of the discretization points.

Value

Returns GCV or CV values calculated for input parameters.

- `gcv`: GCV or CV for a vector of values of the smoothing parameter `h`.
- `fdataobj`: `fdata` class object.
- `fdata.est`: Estimated `fdata` class object.
- `h.opt`: `h` value that minimizes CV or GCV method.
- `S.opt`: Smoothing matrix for the minimum CV or GCV method.
- `gcv.opt`: Minimum of CV or GCV method.
- `h`: Smoothing parameter or bandwidth.

Note

min.np deprecated.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

De Brabanter, K., Cao, F., Gijbels, I., Opsomer, J. (2018). Local polynomial regression with correlated errors in random design and unknown correlation structure. *Biometrika*, 105(3), 681-69.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). Statistical Computing in Functional Data Analysis: The R Package fda.usc. *Journal of Statistical Software*, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

Alternative method: `optim.basis`

Examples

```
## Not run:
# Exemple, phoneme DATA
data(phoneme)
mlearn<-phoneme$learn[1:100]

out1<-optim.np(mlearn,type.CV=CV.S,type.S=S.NW)
np<-ncol(mlearn)
# variance calculations
y<-mlearn
out<-out1
```

```
i<-1
z=qnorm(0.025/np)
fdata.est<-out$fdata.est
tt<-y[["argvals"]]
var.e<-Var.e(y,out$S.opt)
var.y<-Var.y(y,out$S.opt)
var.y2<-Var.y(y,out$S.opt,var.e)

# plot estimated fdata and point confidence interval
upper.var.e<-fdata.est[i,]-z*sqrt(diag(var.e))
lower.var.e<-fdata.est[i,]+z*sqrt(diag(var.e))
dev.new()
plot(y[i,],lwd=1,
ylim=c(min(lower.var.e$data),max(upper.var.e$data)),xlab="t")
lines(fdata.est[i,],col=gray(.1),lwd=1)
lines(fdata.est[i,]+z*sqrt(diag(var.y)),col=gray(0.7),lwd=2)
lines(fdata.est[i,]-z*sqrt(diag(var.y)),col=gray(0.7),lwd=2)
lines(upper.var.e,col=gray(.3),lwd=2,lty=2)
lines(lower.var.e,col=gray(.3),lwd=2,lty=2)
legend("bottom",legend=c("Var.y","Var.error"),
col = c(gray(0.7),gray(0.3)),lty=c(1,2))

## End(Not run)
```

Outliers.fdata *outliers for functional dataset*

Description

Procedure for detecting funcitonal outliers.

Usage

```
outliers.depth.pond(
  fdataobj,
  nb = 200,
  smo = 0.05,
  quan = 0.5,
  dfunc = depth.mode,
  ...
)

outliers.depth.trim(
  fdataobj,
  nb = 200,
  smo = 0.05,
  trim = 0.01,
  quan = 0.5,
  dfunc = depth.mode,
  ...
)

outliers.lrt(fdataobj, nb = 200, smo = 0.05, trim = 0.1, ...)

outliers.thres.lrt(fdataobj, nb = 200, smo = 0.05, trim = 0.1, ...)
```

Arguments

`fdataobj`	`fdata` class object.
`nb`	The number of bootstrap samples.
`smo`	The smoothing parameter for the bootstrap samples.
`quan`	Quantile to determine the cutoff from the Bootstrap procedure (by default=0.5)
`dfunc`	Type of depth measure, by default `depth.mode`.
`...`	Further arguments passed to or from other methods.
`trim`	The alpha of the trimming.

Details

Outlier detection in functional data by likelihood ratio test (`outliers.lrt`). The threshold for outlier detection is given by the `outliers.thres.lrt`. Outlier detection in functional data by depth measures:

- `outliers.depth.pond`: function weights the data according to depth.
- `outliers.depth.trim`: function uses trimmed data.

`quantile_outliers_pond` and `quantile_outliers_trim` functions provides the quantiles of the bootstrap samples for functional outlier detection by, respectively, weigthed and trimmed procedures. Bootstrap smoothing function (`fdata.bootstrap` with `nb` resamples) is applied to these weighted or trimmed data. If `smo=0` smoothed bootstrap is not performed. The function returns a vector of size `1xnb` with bootstrap replicas of the quantile.

Value

`outliers`	Indexes of functional outlier.
`dep.out`	Depth value of functional outlier.
`dep.out`	Iteration in which the functional outlier is detected.
`quantile`	Threshold for outlier detection.
`dep`	Depth value of functional data.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Cuevas A, Febrero M, Fraiman R. 2006. *On the use of bootstrap for estimating functions with functional data*. Computational Statistics and Data Analysis 51: 1063-1074.

Febrero-Bande, M., Galeano, P., and Gonzalez-Manteiga, W. (2008). *Outlier detection in functional data by depth measures with application to identify abnormal NOx levels*. Environmetrics 19, 4, 331–345.

Febrero-Bande, M., Galeano, P. and Gonzalez-Manteiga, W. (2007). *A functional analysis of NOx levels: location and scale estimation and outlier detection*. Computational Statistics 22, 3, 411-427.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also: `fdata.bootstrap`, `Depth`.

Examples

```
## Not run:
data(aemet)
nb=20 # Time consuming
out.trim<-outliers.depth.trim(aemet$temp,dfunc=depth.FM,nb=nb)
plot(aemet$temp,col=1,lty=1)
lines(aemet$temp[out.trim[[1]]],col=2)

## End(Not run)
```

`P.penalty` *Penalty matrix for higher order differences*

Description

This function computes the matrix that penalizes the higher order differences.

Usage

```
P.penalty(tt, P = c(0, 0, 1))
```

Arguments

`tt`	vector of the `n` discretization points or argvals.
`P`	vector of coefficients with the order of the differences. Default value `P`=c(0,0,1) penalizes the second order difference.

Details

For example, if P=c(0,1,2), the function return the penalty matrix the second order difference of a vector tt. That is

$$v^T P_j tt = \sum_{i=3}^{n} (\Delta tt_i)^2$$

where

$$\Delta tt_i = tt_i - 2tt_{i-1} + tt_{i-2}$$

is the second order difference. More details can be found in Kraemer, Boulesteix, and Tutz (2008).

Value

penalty matrix of size `sum(n)` x `sum(n)`

Note

The discretization points can be equidistant or not.

Author(s)

This version is created by Manuel Oviedo de la Fuente modified the original version created by Nicole Kramer in `ppls` package.

References

N. Kraemer, A.-L. Boulsteix, and G. Tutz (2008). *Penalized Partial Least Squares with Applications to B-Spline Transformations and Functional Data*. Chemometrics and Intelligent Laboratory Systems, 94, 60 - 69. doi:10.1016/j.chemolab.2008.06.009

See Also

`fdata2pls`

Examples

```
P.penalty((1:10)/10,P=c(0,0,1))
# a more detailed example can be found under script file
```

PCvM.statistic *PCvM statistic for the Functional Linear Model with scalar response*

Description

Projected Cramer-von Mises statistic (PCvM) for the Functional Linear Model with scalar response (FLM): $Y = \big\langle X, \beta \big\rangle + \varepsilon$.

Usage

```
Adot(X, inpr)

PCvM.statistic(X, residuals, p, Adot.vec)
```

Arguments

X	Functional covariate for the FLM. The object must be either in the class `fdata` or in the class fd. It is used to compute the matrix of inner products.
inpr	Matrix of inner products of `X`. Computed if not given.
residuals	Residuals of the estimated FLM.
p	Number of elements of the functional basis where the functional covariate is represented.
Adot.vec	Output from the `Adot` function (see Details). Computed if not given.

Details

In order to optimize the computation of the statistic, the critical parts of these two functions are coded in FORTRAN. The hardest part corresponds to the function `Adot`, which involves the computation of a symmetric matrix of dimension $n \times n$ where each entry is a sum of n elements. As this matrix is symmetric, the order of the method can be reduced from $O(n^3)$ to $O\big(\frac{n^3-n^2}{2}\big)$. The memory requirement can also be reduced to $O\big(\frac{n^2-n+2}{2}\big)$. The value of `Adot` is a vector of length $\frac{n^2-n+2}{2}$ where the first element is the common diagonal element and the rest are the lower triangle entries of the matrix, sorted by rows (see Examples).

Value

For `PCvM.statistic`, the value of the statistic. For `Adot`, a suitable output to be used in the argument `Adot.vec`.

Note

No NA's are allowed in the functional covariate.

Author(s)

Eduardo Garcia-Portugues. Please, report bugs and suggestions to
`<eduardo.garcia.portugues@uc3m.es>`

References

Escanciano, J. C. (2006). A consistent diagnostic test for regression models using projections. Econometric Theory, 22, 1030-1051. doi:10.1017/S0266466606060506

Garcia-Portugues, E., Gonzalez-Manteiga, W. and Febrero-Bande, M. (2014). A goodness–of–fit test for the functional linear model with scalar response. Journal of Computational and Graphical Statistics, 23(3), 761-778. doi:10.1080/10618600.2013.812519

See Also

`flm.test`

Examples

```
# Functional process
X=rproc2fdata(n=10,t=seq(0,1,l=101))
# Adot
Adot.vec=Adot(X)

# Obtain the entire matrix Adot
Ad=diag(rep(Adot.vec[1],dim(X$data)[1]))
Ad[upper.tri(Ad,diag=FALSE)]=Adot.vec[-1]
Ad=t(Ad)
Ad=Ad+t(Ad)-diag(diag(Ad))
Ad
# Statistic
PCvM.statistic(X,residuals=rnorm(10),p=5)
```

`phoneme` *phoneme data*

Description

Phoneme curves

Format

Elements of phoneme:
`..$learn`: learning sample of curves. `fdata` class object with: i.- `"data"`: Matrix of class `fdata` with 250 curves (rows) discretized in 150 points or argvals (columns).
, ii.- `"argvals"`, iii.- `"rangeval"`: range(`"argvals"`), iv.- `"names"` list with: `main` an overall title "Phoneme learn", `xlab` title for x axis "frequencies" and `ylab` title for y axis "log-periodograms".

`..$test`: testing sample of curves. `fdata` class object with: i.- `"data"`: Matrix of class `fdata` with 250 curves (rows) discretized in 150 points or argvals (columns).
, ii.- `"argvals"`, iii.- `"rangeval"`: range(`"argvals"`), iv.- `"names"` list with: `main` an overall title "Phoneme learn", `xlab` title for x axis "frequencies" and `ylab` title for y axis "log-periodograms".

`..$classlearn`:learning class numbers (as factor). Factor levels: "sh" 1, "iy" 2, "dcl" 3, "aa" 4 and "ao" 5.

`..$classtest`: testing class numbers (as factor). Factor levels: "sh" 1, "iy" 2, "dcl" 3, "aa" 4 and "ao" 5.

Details

The following instructions have been used file:
`https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/npfda-phondiscRS.txt`
of `Phoneme dataset` file.

Author(s)

Manuel Febrero-Bande and Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

Source

`https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/npfda-datasets.html`

References

Ferraty, F. and Vieu, P. (2006). *NPFDA in practice*. Free access on line at `https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/`

Examples

```
data(phoneme)
names(phoneme)
names(phoneme$learn)
class(phoneme$learn)
dim(phoneme$learn)
table(phoneme$classlearn)
```

`plot.fdata` *Plot functional data: fdata class object*

Description

Plot object of class `fdata`.

Usage

```
## S3 method for class 'fdata'
plot(x, type, main, xlab, ylab, lty = 1, mfrow = c(1, 1), time = 1, ...)

## S3 method for class 'fdata'
lines(x, ...)

title.fdata(x, main = NULL, xlab = NULL, ylab = NULL, rownames = NULL)

## S3 method for class 'mdepth'
plot(x, trim, levgray = 0.9, ...)

## S3 method for class 'depth'
plot(x, trim, levgray = 0.9, ...)

## S3 method for class 'bifd'
plot(x, argvals.s, argvals.t, ...)

## S3 method for class 'lfdata'
plot(x, ask = FALSE, col, ...)
```

Arguments

`x`	`fdata` class object with: • `"data"`: For `fdata` class object as curve (1d), `"data"` is a `matrix` (by default), `data.frame` or `array` of set cases with dimension (`n` x `m`), where `n` is the number of curves and `m` are the points observed in each curve over the x–axe. For `fdata2d` class object as surface (2d). `"data"` is a `array` of set cases with dimension (`n` x `m1` x `m2`), where `n` is the number of functional data and `m1` and `m2` are the points observed over the x–y plane. • `"argvals"`: vector or list of vectors with the discretizations points values. • `"rangeval"`: vector or list of vectors with the range of the discretizations points values, by default range(`argvals`). • `"names"`: (optional) list with `main` an overall title, `xlab` title for `x` axis and `ylab` title for `y` axis. or a two-argument functional data object, see bifd.
`type`	1-character string giving the type of plot desired. The following values are possible for `fdata` class object: "l" for lines (by default),"p" for points, , "o" for overplotted points and lines, "b", "c" for (empty if "c") points joined by lines, "s" and "S" for stair steps and "h" for histogram-like vertical lines. Finally, "n" does not produce any points or lines. The following values are possible for `fdata2d` class object: "image.contour" (by default) to display three-dimensional data and add the contour lines, "image" to display three-dimensional data, "contour" to display a contour plot, "persp" to display a perspective plots of a surface over the x-y plane and "filled.contour" to display a contour plot with the areas between the contours filled in solid color.

main	an overall title for the plot: see title.
xlab	xlab title for x axis, as in plot.
ylab	ylab title for y axis, as in plot.
lty	a vector of line types, see par.
mfrow	A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by rows (mfrow).
time	The time interval to suspend plot execution for, in seconds, see Sys.sleep.
...	Further arguments passed to matplot function (for fdata class) or image, contour, persp or filled.contour (for fdata2d class).
rownames	Row names.
trim	The alpha of the trimming.
levgray	A vector of desired gray levels between 0 and 1; zero indicates "black" and one indicates "white".
argvals.s	a vector of argument values for the first argument s of the functional data object to be evaluated.
argvals.t	a vector of argument values for the second argument t of the functional data object to be evaluated.
ask	Logical. If TRUE, prompts for user interaction with the current graphics device.
col	The colors for curves, lines and points.

Author(s)

Manuel Febrero Bande and Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See Also as fdata

Examples

```
## Not run:
# Example for fdata class of 1 dimension (curve)
a1<-seq(0,1,by=.01)
a2=rnorm(length(a1),sd=0.2)
f1<-(sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
nc<-10
np<-length(f1)
tt=seq(0,1,len=101)
mdata<-matrix(NA,ncol=np,nrow=nc)
for (i in 1:nc) mdata[i,]<- (sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
fdataobj<-fdata(mdata,tt)
res=plot.fdata(fdataobj,type="l",col=gray(1:nrow(mdata)/nrow(mdata)))
lines(func.mean(fdataobj),col=3,lwd=2) #original curve

# example for fdata2d class of 2 dimension (surface)
t1 <- seq(0, 1, length= 51)
```

```
t2 <- seq(0, 1, length= 31)
z<-array(NA,dim=c(4,51,31))
for (i in 1:4) z[i,,] <- outer(t1, t2, function(a, b) (i*a)*(b)^i)
z.fdata<-fdata(z,list(t1,t2))
plot(z.fdata,time=2)
plot(z.fdata,mfrow=c(2,2),type="persp",theta=30)

## End(Not run)
```

`poblenou` *poblenou data*

Description

NOx levels measured every hour by a control station in Poblenou in Barcelona (Spain).

Format

The format is:
`..$nox`: `fdata` class object with:
i.- `"data"`: Matrix with 115 curves (rows) discretized in 24 points or argvals (columns).
ii.- `"argvals"`: `0:23`
iii.- `"rangeval"=(0,23)`: range(`"argvals"`),
iv.- `"names"` list with: `main` an overall title "NOx data set", `xlab` title for `x` axis "Hours" and `ylab` title for `y` axis "NOx (mglm^3)".

`..$df`: Data Frame with (115x3) dimension.
"date" in the first column.
Second column ("day.week"). Factor levels: "Monday" 1, "Tuesday" 2, "Wednesday" 3, "Thursday" 4, "Friday" 5, "Saturday" 6 and "Sunday" 7.
Third column "day.festive". Factor levels: "non festive day" 0 and "festive day" 1.

Details

The dataset starts on 23 February and ends on 26 June, in 2005. We split the whole sample of hourly measures in a dataset of functional trajectories of 24 h observations (each curve represents the evolution of the levels in 1 day).
Twelve curves that contained missing data were eliminated.

Author(s)

Febrero-Bande, M and Oviedo de la Fuente, Manuel

Source

`https://mediambient.gencat.cat/ca/05_ambits_dactuacio/`

References

Febrero-Bande, M., Galeano, P., and Gonzalez-Manteiga, W. (2008). *Outlier detection in functional data by depth measures with application to identify abnormal NOx levels*. Environmetrics 19, 4, 331-345.

Examples

```
data(poblenou)
names(poblenou)
names(poblenou$nox)
nox<-poblenou$nox
class(nox)
ind.weekend<-as.integer(poblenou$df[,"day.week"])>5
plot(nox,col=ind.weekend+1)
```

---

predict.classif *Predicts from a fitted classif object.*

Description

Classifier of functional data by kernel method using functional data object of class `classif`. Returns the predicted classes using a previously trained model.

Usage

```
## S3 method for class 'classif'
predict(object, new.fdataobj = NULL, type = "class", ...)
```

Arguments

`object`	Object `object` estimated by: k nearest neighbors method `classif.knn`, kernel method `classif.kernel`.
`new.fdataobj`	New functional explanatory data of `fdata` class.
`type`	Type of prediction ("class or probability of each group membership").
`...`	Further arguments passed to or from other methods.

Value

If type="class", produces a vector of predictions. If type="probs", a list with the following components is returned:

- `group.pred` the vector of predictions.
- `prob.group` the matrix of predicted probability by factor level.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametricc functional data analysis.* Springer Series in Statistics, New York.

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

See Also

See also `classif.np` `classif.glm`, `classif.gsam` and `classif.gkam` .

Examples

```
## Not run:
data(phoneme)
mlearn <- phoneme[["learn"]][1:100]
glearn <- phoneme[["classlearn"]][1:100]

# ESTIMATION knn
out1 <- classif.knn(glearn, mlearn, knn = 3)
summary(out1)

# PREDICTION knn
mtest <- phoneme[["test"]][1:100]
gtest <- phoneme[["classtest"]][1:100]
pred1 <- predict(out1, mtest)
table(pred1, gtest)

# ESTIMATION kernel
h <- 2^(0:5)
# using metric distances computed in classif.knn
out2 <- classif.kernel(glearn, mlearn, h = h, metric = out1$mdist)
summary(out2)
# PREDICTION kernel
pred2 <- predict(out2,mtest)
table(pred2,gtest)

## End(Not run)
```

`predict.classif.DD` *Predicts from a fitted classif.DD object.*

Description

Classifier of functional (and multivariate) data by DD–classifier.

Usage

```
## S3 method for class 'classif.DD'
predict(object, new.fdataobj = NULL, type = "class", ...)
```

Arguments

`object`	Object `object` estimated by `classif.DD`.
`new.fdataobj`	By default, new p functional explanatory dataset or new mulitvariate data of `data.frame` class
`type`	!="predictive", for each row of data shows the probability of each group membership.
`...`	Further arguments passed to or from other methods.

Details

Returns the groups or classes predicted using a previously trained model.

Value

- `group.pred`:Vector of groups or classes predicted
- `prob.group`: For each functional data shows the probability of each group membership.

Author(s)

Febrero-Bande, M., and Oviedo de la Fuente, M.

References

Li, J., P.C., Cuesta-Albertos, J.A. and Liu, R. *DD–Classifier: Nonparametric Classification Procedure Based on DD-plot.* Journal of the American Statistical Association (2012), Vol. 107, 737–753.

See Also

See also `classif.DD` .

Examples

```
## Not run:
# DD-classif for multivariate data
data(iris)
iris<-iris[1:100,]
ii<-sample(1:100,80)
group.train<-factor(iris[ii,5])
x.train<-iris[ii,1:4]
out1=classif.DD(group.train,x.train,depth="MhD",classif="lda")
out2=classif.DD(group.train,x.train,depth="MhD",classif="glm")
summary(out1)
summary(out2)
x.test<-iris[-ii,1:4]
pred1=predict(out1,x.test)
pred2=predict(out2,x.test)
group.test<-iris[-ii,5]
table(pred1,group.test)
table(pred2,group.test)
```

```
# DD-classif for Functional data
data(phoneme)
mlearn<-phoneme[["learn"]]
glearn<-phoneme[["classlearn"]]

# ESTIMATION
out1=classif.DD(glearn,mlearn,depth="FM",classif="glm")
summary(out1)
# PREDICTION
mtest<-phoneme[["test"]]
gtest<-phoneme[["classtest"]]
pred1=predict(out1,mtest)
table(pred1,gtest)

## End(Not run)
```

`predict.fregre.fd` *Predict method for functional linear model (fregre.fd class)*

Description

Computes predictions for regression between functional explanatory variables and scalar response using: basis representation, Principal Components Analysis, Partial least squares or nonparametric kernel estimation.

Predicts from a fitted `fregre.basis` object,see `fregre.basis` or `fregre.basis.cv`
Predicts from a fitted `fregre.pc` object,see `fregre.pc` or `fregre.pc.cv`
Predicts from a fitted `fregre.pls` object,see `fregre.pls` or `fregre.pls.cv`
Predicts from a fitted `fregre.np` object, see `fregre.np` or `fregre.np.cv`.

Usage

```
## S3 method for class 'fregre.fd'
predict(
  object,
  new.fdataobj = NULL,
  se.fit = FALSE,
  scale = NULL,
  df = df,
  interval = "none",
  level = 0.95,
  weights = 1,
  pred.var = res.var/weights,
  ...
)
```

Arguments

`object`	`fregre.fd` object.
`new.fdataobj`	New functional explanatory data of `fdata` class.
`se.fit`	=TRUE (not default) standard error estimates are returned for each prediction.
`scale`	Scale parameter for std.err. calculation.
`df`	Degrees of freedom for scale.
`interval`	Type of interval calculation.
`level`	Tolerance/confidence level.
`weights`	variance weights for prediction. This can be a numeric vector or a one-sided model formula. In the latter case, it is interpreted as an expression evaluated in newdata
`pred.var`	the variance(s) for future observations to be assumed for prediction intervals. See `link{predict.lm}` for more details.
`...`	Further arguments passed to or from other methods.

Value

If `se.fit = FALSE`, a vector of predictions of scalar response is returned or a matrix of predictions and bounds with column names fit, lwr, and upr if interval is set. If `se.fit =TRUE` a list with the following components is returned:

- `fit`: A vector of predictions or a matrix of predictions and bounds as above.
- `se.fit`: Associated standard error estimates of predictions.
- `residual.scale`: Residual standard deviations.
- `df`: Degrees of freedom for residual.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Cai TT, Hall P. 2006. *Prediction in functional linear regression*. Annals of Statistics 34: 2159-2179.

Cardot H, Ferraty F, Sarda P. 1999. *Functional linear model*. Statistics and Probability Letters 45: 11-22.

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Hall P, Hosseini-Nasab M. 2006. *On properties of functional principal components analysis*. Journal of the Royal Statistical Society B 68: 109-126.

Hardle, W. *Applied Nonparametric Regression*. Cambridge University Press, 1994.

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. https://www.jstatsoft.org/v51/i04/

See Also

See Also as: fregre.basis, fregre.basis.cv, fregre.np, fregre.np.cv, fregre.pc, fregre.pc.cv, fregre.pls, fregre.pls.cv and summary.fregre.fd.

Examples

```
## Not run:
data(tecator)
absorp=tecator$absorp.fdata
ind=1:129
x=absorp[ind,]
y=tecator$y$Fat[ind]
newx=absorp[-ind,]
newy=matrix(tecator$y$Fat[-ind],ncol=1)
## Functional PC regression
res.pc=fregre.pc(x,y,1:6)
pred.pc=predict(res.pc,newx)
# Functional PLS regression
res.pls=fregre.pls(x,y,1:6)
pred.pls=predict(res.pls,newx)
# Functional nonparametric regression
res.np=fregre.np(x,y,Ker=AKer.tri,metric=semimetric.deriv)
pred.np=predict(res.np,newx)
# Functional regression with basis representation
res.basis=fregre.basis.cv(x,y)
pred.basis=predict(res.basis[[1]],newx)

dev.new()
plot(pred.pc-newy)
points(pred.pls-newy,col=2,pch=2)
points(pred.np-newy,col=3,pch=3)
points(pred.basis-newy,col=4,pch=4)
sum((pred.pc-newy)^2,na.rm=TRUE)/sum((newy-mean(newy))^2,na.rm=TRUE)
sum((pred.pls-newy)^2,na.rm=TRUE)/sum((newy-mean(newy))^2,na.rm=TRUE)
sum((pred.np-newy)^2,na.rm=TRUE)/sum((newy-mean(newy))^2,na.rm=TRUE)
sum((pred.basis-newy)^2,na.rm=TRUE)/sum((newy-mean(newy))^2,na.rm=TRUE)

## End(Not run)
```

---

predict.fregre.fr *Predict method for functional response model*

Description

Computes predictions for regression between functional explanatory variables and functional response.

Usage

```
## S3 method for class 'fregre.fr'
predict(object, new.fdataobj = NULL, ...)
```

Arguments

`object`	`fregre.fr` object.
`new.fdataobj`	New functional explanatory data of `fdata` class.
`...`	Further arguments passed to or from other methods.

Value

Return the predicted functional data.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See Also as: `fregre.basis.fr`

Examples

```
## Not run:
# CV prediction for CandianWeather data
rtt<-c(0, 365)
basiss  <- create.bspline.basis(rtt,7)
basist  <- create.bspline.basis(rtt,9)
nam<-dimnames(CanadianWeather$dailyAv)[[2]]

# fdata class (raw data)
tt<-1:365
tempfdata<-fdata(t(CanadianWeather$dailyAv[,,1]),tt,rtt)
log10precfdata<-fdata(t(CanadianWeather$dailyAv[,,3]),tt,rtt)
rng<-range(log10precfdata)
for (ind in 1:35){
 res1<-  fregre.basis.fr(tempfdata[-ind], log10precfdata[-ind],
 basis.s=basiss,basis.t=basist)
 pred1<-predict(res1,tempfdata[ind])
 plot( log10precfdata[ind],col=1,ylim=rng,main=nam[ind])
 lines(pred1,lty=2,col=2)
 Sys.sleep(1)
}

# fd class  (smooth data)
basis.alpha  <- create.constant.basis(rtt)
basisx  <- create.bspline.basis(rtt,65)

dayfd<-Data2fd(day.5,CanadianWeather$dailyAv,basisx)
tempfd<-dayfd[,1]
```

```
log10precfd<-dayfd[,3]
for (ind in 1:35){
 res2 <-  fregre.basis.fr(tempfd[-ind], log10precfd[-ind],
 basis.s=basiss,basis.t=basist)
 pred2<-predict(res2,tempfd[ind])
 plot(log10precfd[ind],col=1,ylim=range(log10precfd$coef),main=nam[ind])
 lines(pred2,lty=2,col=2)
 Sys.sleep(.5)
}

## End(Not run)
```

`predict.fregre.gkam` *Predict method for functional linear model*

Description

Computes predictions for regression between functional (and non functional) explanatory variables and scalar response.

- `predict.fregre.lm`, Predict method for functional linear model of `fregre.lm` fits object using basis or principal component representation.
- `predict.fregre.plm`, Predict method for semi-functional linear regression model of `fregre.plm` fits object using using asymmetric kernel estimation.
- `predict.fregre.glm`, Predict method for functional generalized linear model of `fregre.glm` fits object using basis or principal component representation.
- `predict.fregre.gsam`, Predict method for functional generalized spectral additive model of `fregre.gsam` fits object using basis or principal component representation.
- `predict.fregre.gkam`, Predict method for functional generalized kernel additive model of `fregre.gkam` fits object using backfitting algorithm.

These functions use the model fitting function `lm`, `glm` or `gam` properties.
If using functional data derived, is recommended to use a number of bases to represent beta lower than the number of bases used to represent the functional data.
The first item in the `data` list of `newx` argument is called *"df"* and is a data frame with the response and non functional explanatory variables, as `lm`, `glm` or `gam`. Functional variables (`fdata` and `fd` class) are introduced in the following items in the `data` list of `newx` argument.

Usage

```
## S3 method for class 'fregre.gkam'
predict(object, newx = NULL, type = "response", ...)

## S3 method for class 'fregre.glm'
predict(object, newx = NULL, type = "response", ...)
```

```
## S3 method for class 'fregre.gsam'
predict(object, newx = NULL, type = "response", ...)

## S3 method for class 'fregre.lm'
predict(
  object,
  newx = NULL,
  type = "response",
  se.fit = FALSE,
  scale = NULL,
  df = df,
  interval = "none",
  level = 0.95,
  weights = 1,
  pred.var = res.var/weights,
  ...
)

## S3 method for class 'fregre.plm'
predict(object, newx = NULL, ...)
```

Arguments

`object`	`fregre.lm`, `fregre.plm`, `fregre.glm`, `fregre.gsam` or `fregre.gkam` object.
`newx`	An optional data list in which to look for variables with which to predict. If omitted, the fitted values are used. List of new explanatory data.
`type`	a character vector, Type of prediction: (`response`, `terms` for model terms or `effects` for model terms where the partial effects are summarized for each functional variable.
`...`	Further arguments passed to or from other methods.
`se.fit`	=TRUE (not default) standard error estimates are returned for each prediction.
`scale`	Scale parameter for std.err. calculation.
`df`	Degrees of freedom for scale.
`interval`	Type of interval calculation.
`level`	Tolerance/confidence level.
`weights`	variance weights for prediction. This can be a numeric vector or a one-sided model formula. In the latter case, it is interpreted as an expression evaluated in newdata
`pred.var`	the variance(s) for future observations to be assumed for prediction intervals. See `link{predict.lm}` for more details.

Value

Return the predicted values and optionally:

- `predict.lm,predict.glm,predict.gam`: produces a vector of predictions or a matrix of predictions and bounds with column names fit, lwr, and upr if interval is set. If se.fit is TRUE, a list with the following components is returned: fit vector or matrix as above.
- `se.fit`: standard error of predicted means.
- `residual.scale`: residual standard deviations.
- `df`: degrees of freedom for residual.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Febrero-Bande, M., Oviedo de la Fuente, M. (2012). *Statistical Computing in Functional Data Analysis: The R Package fda.usc.* Journal of Statistical Software, 51(4), 1-28. `https://www.jstatsoft.org/v51/i04/`

See Also

See Also as: `fregre.lm`, `fregre.plm`, `fregre.glm`, `fregre.gsam` and `fregre.gkam`.

Examples

```
## Not run:
data(tecator)
ind <- 1:129
x <- tecator$absorp.fdata
x.d2 <- fdata.deriv(x,nderiv=2)
tt <- x[["argvals"]]
dataf <- as.data.frame(tecator$y)
ldat <- ldata("df" = dataf[ind,], "x.d2" = x.d2[ind])
basis.x <- list("x.d2" = create.pc.basis(ldat$x.d2))
res <- fregre.gsam(Fat ~  s(x.d2,k=3),
                   data=ldat, family = gaussian(),
                   basis.x = basis.x)
newldat <- ldata("df" = dataf[-ind,], "x.d2" = x.d2[-ind])
pred <- predict(res, newldat)
plot(pred,tecator$y$Fat[-ind])
res.glm <- fregre.glm(Fat  ~  x.d2, data = ldat,
                  family = gaussian(),basis.x = basis.x)
pred.glm <- predict(res.glm, newldat)
newy <- tecator$y$Fat[-ind]
points(pred.glm,tecator$y$Fat[-ind],col=2)

# Time-consuming
res.gkam <- fregre.gkam(Fat ~ x.d2, data = ldat)
pred.gkam <- predict(res.gkam, newldat)
points(pred.gkam,tecator$y$Fat[-ind],col = 4)

((1/length(newy)) * sum((drop(newy)-pred)^2)) / var(newy)
((1/length(newy)) * sum((newy-pred.glm)^2)) / var(newy)
```

```
((1/length(newy)) * sum((newy-pred.gkam)^2)) / var(newy)

## End(Not run)
```

`predict.fregre.gls` *Predictions from a functional gls object*

Description

The predictions for the functional generalized least squares fitted linear model represented by `object` are obtained at the covariate values defined in `newx`.

Usage

```
## S3 method for class 'fregre.gls'
predict(
  object,
  newx = NULL,
  type = "response",
  se.fit = FALSE,
  scale = NULL,
  df,
  interval = "none",
  ...
)

## S3 method for class 'fregre.igls'
predict(
  object,
  newx = NULL,
  data,
  df = df,
  weights = 1,
  pred.var,
  n.ahead = 1L,
  ...
)
```

Arguments

`object`	`fregre.gls` object.
`newx`	An optional data list in which to look for variables with which to predict. If omitted, the fitted values are used. List of new explanatory data.
`type`	Type of prediction (response or model term).
`se.fit`	=TRUE (not default) standard error estimates are returned for each prediction.
`scale`	Scale parameter for std.err. calculation.

`df`	Degrees of freedom for scale.
`interval`	Type of interval calculation.
`...`	Further arguments passed to or from other methods.
`data`	Data frame with the time or spatinal index
`weights`	variance weights for prediction. This can be a numeric vector or a one-sided model formula. In the latter case, it is interpreted as an expression evaluated in newdata
`pred.var`	the variance(s) for future observations to be assumed for prediction intervals. See `link{predict.lm}` for more details.
`n.ahead`	number of steps ahead at which to predict.

Value

a vector with the predicted values.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Oviedo de la Fuente, M., Febrero-Bande, M., Pilar Munoz, and Dominguez, A. Predicting seasonal influenza transmission using Functional Regression Models with Temporal Dependence. arXiv:1610.08718. `https://arxiv.org/abs/1610.08718`

See Also

`fregre.gls`

Examples

```
## Not run:
data(tecator)
ind<-1:190
x <-fdata.deriv(tecator$absorp.fdata,nderiv=1)
dataf=as.data.frame(tecator$y)
dataf$itime <- 1:nrow(x)
ldat=list("df"=dataf[ind,],"x"=x[ind])
newldat=list("df"=dataf[-ind,],"x"=x[-ind])
newy <- tecator$y$Fat[-ind]
ff <- Fat ~ x
res.lm <- fregre.lm(ff,data=ldat)
summary(res.lm)
res.gls <- fregre.gls(ff,data=ldat, correlation=corAR1())
summary(res.gls)
par.cor <- list("cor.ARMA"=list("p"=1))
par.cor <- list("cor.ARMA"=list("index"=c("itime"),"p"=1))
res.igls <- fregre.igls(ff,data=ldat,correlation=par.cor)
pred.lm <- predict(res.lm,newldat)
pred.gls <- predict(res.gls,newldat)
pred.igls <- predict(res.igls,newldat)
mean((pred.lm-newldat$df$Fat)^2)
mean((pred.gls-newldat$df$Fat)^2)
mean((pred.igls-newldat$df$Fat)^2)

## End(Not run)
```

r.ou *Ornstein-Uhlenbeck process*

Description

Sampling of paths of the Ornstein-Uhlenbeck process.

Usage

```
r.ou(
  n,
  t = seq(0, 1, len = 201),
  mu = 0,
  alpha = 1,
  sigma = 1,
  x0 = rnorm(n, mean = mu, sd = sigma/sqrt(2 * alpha))
)
```

Arguments

`n`	number of curves.
`t`	discretization points.
`mu`	mean of the process.
`alpha`	strength of the drift.
`sigma`	diffusion coefficient.
`x0`	a number or a vector of length `n` giving the initial value(s) of the Ornstein-Uhlenbeck process. By default, `n` points are sampled from the stationary distribution.

Value

Functional sample, an `fdata` object of length `n`.

Author(s)

Eduardo Garcia-Portugues (<`edgarcia@est-econ.uc3m.es`>).

Examples

```
plot(r.ou(n = 100))
plot(r.ou(n = 100, alpha = 2, sigma = 4, x0 = 1:100))
```

`rcombfdata` *Utils for generate functional data*

Description

`gridfdata` generates `n` curves as lineal combination of the original curves `fdataobj` plus a functional trend `mu`.

Usage

```
rcombfdata(n = 10, fdataobj, mu, sdarg = rep(1, nrow(fdataobj)), norm = 1)

gridfdata(coef, fdataobj, mu)
```

Arguments

`n`	Number of curves to be generated
`fdataobj`	`fdata` class object.
`mu`	Functional trend, by default `mu`=$\mu(t) = 0$. An object of class `fdata`. `t`=`argvals(mu)`.
`sdarg`	Standard deviation of the coefficients.
`norm`	Norm of the coefficients. The norm is adjusted before the transformation for `sdarg` is performed.
`coef`	Coefficients of the combination. A matrix with number of columns equal to number of curves in `fdataobj`

Details

`rcombfdata` generates `n` random linear combinations of the `fdataobj` curves plus a functional trend `mu`. The coefficients of the combinations follows a normal distribution with zero mean and standard deviation `sdarg`.

Value

Return the functional trajectories as a `fdata` class object.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See Also as `rproc2fdata`

Examples

```
## Not run:
tt=seq(0,1,len=51)
fou3=create.fourier.basis(c(0,1),nbasis=3)
fdataobj=fdata(t(eval.basis(tt,fou3)),argvals=tt)

coef=expand.grid(0,seq(-1,1,len=11),seq(-1,1,len=11))
grid=gridfdata(coef,fdataobj)
plot(grid,lty=1)

rcomb=rcombfdata(n=51,fdataobj,mu=fdata(30*tt*(1-tt),tt))
plot(rcomb,lty=1)

## End(Not run)
```

---

rdir.pc *Data-driven sampling of random directions guided by sample of functional data*

Description

Generation of random directions based on the principal components $\hat{e}_1, \ldots, \hat{e}_k$ of a sample of functional data $X_1, \ldots, X_n$. The random directions are sampled as

$$h = \sum_{j=1}^{k} h_j \hat{e}_j,$$

with $h_j \sim \mathcal{N}(0, \sigma_j^2)$, $j = 1, \ldots, k$. Useful for sampling non-orthogonal random directions h such that they are non-orthogonal for the random sample.

Usage

```
rdir.pc(
  n,
  X.fdata,
  ncomp = 0.95,
 fdata2pc.obj = fdata2pc(X.fdata, ncomp = min(length(X.fdata$argvals), nrow(X.fdata))),
  sd = 0,
  zero.mean = TRUE,
  norm = FALSE
)
```

Arguments

n	number of curves to be generated.
X.fdata	an fdata object used to compute the functional principal components.

`ncomp`	if an integer vector is provided, the index for the principal components to be considered. If a threshold between `0` and `1` is given, the number of components k is determined automatically as the minimum number that explains at least the `ncomp` proportion of the total variance of `X.fdata`.
`fdata2pc.obj`	output of `fdata2pc` containing as many components as the ones to be selected by `ncomp`. Otherwise, it is computed internally.
`sd`	if `0`, the standard deviations σ_j are estimated by the standard deviations of the scores for e_j. If not, the σ_j's are set to `sd`.
`zero.mean`	whether the projections should have zero mean. If not, the mean is set to the mean of `X.fdata`.
`norm`	whether the samples should be L2-normalized or not.

Value

A `fdata` object with the sampled directions.

Author(s)

Eduardo Garcia-Portugues (<`edgarcia@est-econ.uc3m.es`>) and Manuel Febrero-Bande (<`manuel.febrero@usc.es`>).

Examples

```
## Not run:
# Simulate some data
set.seed(345673)
X.fdata <- r.ou(n = 200, mu = 0, alpha = 1, sigma = 2, t = seq(0, 1, l = 201),
                x0 = rep(0, 200))
pc <- fdata2pc(X.fdata, ncomp = 20)

# Samples
set.seed(34567)
rdir.pc(n = 5, X.fdata = X.fdata, zero.mean = FALSE)$data[, 1:5]
set.seed(34567)
rdir.pc(n = 5, X.fdata = X.fdata, fdata2pc.obj = pc)$data[, 1:5]

# Comparison for the variance type
set.seed(456732)
n.proj <- 100
set.seed(456732)
samp1 <- rdir.pc(n = n.proj, X.fdata = X.fdata, sd = 1, norm = FALSE, ncomp = 0.99)
set.seed(456732)
samp2 <- rdir.pc(n = n.proj, X.fdata = X.fdata, sd = 0, norm = FALSE, ncomp = 0.99)
set.seed(456732)
samp3 <- rdir.pc(n = n.proj, X.fdata = X.fdata, sd = 1, norm = TRUE, ncomp = 0.99)
set.seed(456732)
samp4 <- rdir.pc(n = n.proj, X.fdata = X.fdata, sd = 0, norm = TRUE, ncomp = 0.99)
par(mfrow = c(1, 2))
plot(X.fdata, col = gray(0.85), lty = 1)
lines(samp1[1:10], col = 2, lty = 1)
lines(samp2[1:10], col = 4, lty = 1)
```

```
legend("topleft", legend = c("Data", "Different variances", "Equal variances"),
       col = c(gray(0.85), 2, 4), lwd = 2)
plot(X.fdata, col = gray(0.85), lty = 1)
lines(samp3[1:10], col = 5, lty = 1)
lines(samp4[1:10], col = 6, lty = 1)
legend("topleft", legend = c("Data", "Different variances, normalized",
       "Equal variances, normalized"), col = c(gray(0.85), 5:6), lwd = 2)

# Correlations (stronger with different variances and unnormalized;
# stronger with lower ncomp)
ind <- lower.tri(matrix(nrow = n.proj, ncol = n.proj))
median(abs(cor(sapply(1:n.proj, function(i) inprod.fdata(X.fdata, samp1[i]))))[ind])
median(abs(cor(sapply(1:n.proj, function(i) inprod.fdata(X.fdata, samp2[i]))))[ind])
median(abs(cor(sapply(1:n.proj, function(i) inprod.fdata(X.fdata, samp3[i]))))[ind])
median(abs(cor(sapply(1:n.proj, function(i) inprod.fdata(X.fdata, samp4[i]))))[ind])

# Comparison for the threshold
samp1 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.25, fdata2pc.obj = pc)
samp2 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.50, fdata2pc.obj = pc)
samp3 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.90, fdata2pc.obj = pc)
samp4 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.95, fdata2pc.obj = pc)
samp5 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.99, fdata2pc.obj = pc)
cols <- rainbow(5, alpha = 0.25)
par(mfrow = c(3, 2))
plot(X.fdata, col = gray(0.75), lty = 1, main = "Data")
plot(samp1, col = cols[1], lty = 1, main = "Threshold = 0.25")
plot(samp2, col = cols[2], lty = 1, main = "Threshold = 0.50")
plot(samp3, col = cols[3], lty = 1, main = "Threshold = 0.90")
plot(samp4, col = cols[4], lty = 1, main = "Threshold = 0.95")
plot(samp5, col = cols[5], lty = 1, main = "Threshold = 0.99")

# Normalizing
samp1 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.50, fdata2pc.obj = pc,
                 norm = TRUE)
samp2 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.90, fdata2pc.obj = pc,
                 norm = TRUE)
samp3 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.95, fdata2pc.obj = pc,
                 norm = TRUE)
samp4 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.99, fdata2pc.obj = pc,
                 norm = TRUE)
samp5 <- rdir.pc(n = 100, X.fdata = X.fdata, ncomp = 0.999, fdata2pc.obj = pc,
                 norm = TRUE)
cols <- rainbow(5, alpha = 0.25)
par(mfrow = c(3, 2))
plot(X.fdata, col = gray(0.75), lty = 1, main = "Data")
plot(samp1, col = cols[1], lty = 1, main = "Threshold = 0.50")
plot(samp2, col = cols[2], lty = 1, main = "Threshold = 0.90")
plot(samp3, col = cols[3], lty = 1, main = "Threshold = 0.95")
plot(samp4, col = cols[4], lty = 1, main = "Threshold = 0.99")
plot(samp5, col = cols[5], lty = 1, main = "Threshold = 0.999")

## End(Not run)
```

rp.flm.statistic *Statistics for testing the functional linear model using random projections*

Description

Computes the Cramer-von Mises (CvM) and Kolmogorv-Smirnov (kS) statistics on the projected process

$$T_{n,h}(u) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \langle X_i, \hat{\beta}\rangle)1_{\{\langle X_i, h\rangle \le u\}},$$

designed to test the goodness-of-fit of a functional linear model with scalar response. `NA`'s are not allowed neither in the functional covariate nor in the scalar response.

Usage

```
rp.flm.statistic(proj.X, residuals, proj.X.ord = NULL, F.code = TRUE)
```

Arguments

`proj.X`	matrix of size `c(n, n.proj)` containing, for each column, the projections of the functional data $X_1, \ldots, X_n$ into a random direction h. Not required if `proj.X.ord` is provided.
`residuals`	the residuals of the fitted funtional linear model, $Y_i - \langle X_i, \hat{\beta}\rangle$. Either a vector of length `n` (same residuals for all projections) or a matrix of size `c(n.proj, n)` (each projection has an associated set residuals).
`proj.X.ord`	matrix containing the row permutations of `proj.X` which rearranges them increasingly, for each column. So, for example `proj.X[proj.X.ord[, 1], 1]` equals `sort(proj.X[, 1])`. If not provided, it is computed internally.
`F.code`	whether to use faster `FORTRAN` code or `R` code.

Value

A list containing:

- `list("statistic")`: a matrix of size `c(n.proj, 2)` with the the CvM (first column) and KS (second) statistics, for the `n.proj` different projections.
- `list("proj.X.ord")`: the computed row permutations of `proj.X`, useful for recycling in subsequent calls to `rp.flm.statistic` with the same projections but different residuals.

Author(s)

Eduardo Garcia-Portugues (<`edgarcia@est-econ.uc3m.es`>) and Manuel Febrero-Bande (<`manuel.febrero@usc.es`>).

References

Cuesta-Albertos, J.A., Garcia-Portugues, E., Febrero-Bande, M. and Gonzalez-Manteiga, W. (2017). Goodness-of-fit tests for the functional linear model based on randomly projected empirical processes. arXiv:1701.08363. https://arxiv.org/abs/1701.08363

Examples

```
## Not run:
# Simulated example
set.seed(345678)
t <- seq(0, 1, l = 101)
n <- 100
X <- r.ou(n = n, t = t)
beta0 <- fdata(mdata = cos(2 * pi * t) - (t - 0.5)^2, argvals = t,
               rangeval = c(0,1))
Y <- inprod.fdata(X, beta0) + rnorm(n, sd = 0.1)

# Linear model
mod <- fregre.pc(fdataobj = X, y = Y, l = 1:3)

# Projections
proj.X1 <- inprod.fdata(X, r.ou(n = 1, t = t))
proj.X2 <- inprod.fdata(X, r.ou(n = 1, t = t))
proj.X12 <- cbind(proj.X1, proj.X2)

# Statistics
t1 <- rp.flm.statistic(proj.X = proj.X1, residuals = mod$residuals)
t2 <- rp.flm.statistic(proj.X = proj.X2, residuals = mod$residuals)
t12 <- rp.flm.statistic(proj.X = proj.X12, residuals = mod$residuals)
t1$statistic
t2$statistic
t12$statistic

# Recycling proj.X.ord
rp.flm.statistic(proj.X.ord = t1$proj.X.ord, residuals = mod$residuals)$statistic
t1$statistic

# Sort in the columns
cbind(proj.X12[t12$proj.X.ord[, 1], 1], proj.X12[t12$proj.X.ord[, 2], 2]) -
apply(proj.X12, 2, sort)

# FORTRAN and R code
rp.flm.statistic(proj.X = proj.X1, residuals = mod$residuals)$statistic -
rp.flm.statistic(proj.X = proj.X1, residuals = mod$residuals,
                 F.code = FALSE)$statistic

# Matrix and vector residuals
rp.flm.statistic(proj.X = proj.X12, residuals = mod$residuals)$statistic
rp.flm.statistic(proj.X = proj.X12,
                 residuals = rbind(mod$residuals, mod$residuals * 2))$statistic

## End(Not run)
```

rp.flm.test *Goodness-of fit test for the functional linear model using random projections*

Description

Tests the composite null hypothesis of a Functional Linear Model with scalar response (FLM),

$$H_0 : Y = \langle X, \beta \rangle + \epsilon \quad \text{vs} \quad H_1 : Y \neq \langle X, \beta \rangle + \epsilon.$$

If $\beta = \beta_0$ is provided, then the simple hypothesis $H_0 : Y = \langle X, \beta_0 \rangle + \epsilon$ is tested. The way of testing the null hypothesis is via a norm (Cramer-von Mises or Kolmogorov-Smirnov) in the empirical process indexed by the projections.

No NA's are allowed neither in the functional covariate nor in the scalar response.

Usage

```
rp.flm.test(
  X.fdata,
  Y,
  beta0.fdata = NULL,
  B = 1000,
  n.proj = 10,
  est.method = "pc",
  p = NULL,
  p.criterion = "SICc",
  pmax = 20,
  type.basis = "bspline",
  projs = 0.95,
  verbose = TRUE,
  same.rwild = FALSE,
  seed = NULL,
  ...
)
```

Arguments

`X.fdata`	functional observations in the class `fdata`.
`Y`	scalar responses for the FLM. Must be a vector with the same number of elements as functions are in `X.fdata`.
`beta0.fdata`	functional parameter for the simple null hypothesis, in the `fdata` class. The `argvals` and `rangeval` arguments of `beta0.fdata` must be the same of `X.fdata`. If `beta0.fdata=NULL` (default), the function will test for the composite null hypothesis.

`B`	number of bootstrap replicates to calibrate the distribution of the test statistic.
`n.proj`	vector with the number of projections to consider.
`est.method`	Estimation method for β, only used in the composite case. There are three methods: • `"pc"`: if `p` is given, then β is estimated by `fregre.pc`. Otherwise, `p` is chosen using `fregre.pc.cv` and the `p.criterion` criterion. • `"pls"`: if `p` is given, β is estimated by `fregre.pls`. Otherwise, `p` is chosen using `fregre.pls.cv` and the `p.criterion` criterion. • `"basis"`: if `p` is given, β is estimated by `fregre.basis`. Otherwise, `p` is chosen using `fregre.basis.cv` and the `p.criterion` criterion. Both in `fregre.basis` and `fregre.basis.cv`, the same basis for `basis.x` and `basis.b` is considered.
`p`	number of elements for the basis representation of `beta0.fdata` and `X.fdata` with the `est.method` (only composite hypothesis). If not supplied, it is estimated from the data.
`p.criterion`	for `est.method` equal to `"pc"` or `"pls"`, either `"SIC"`, `"SICc"` or one of the criterions described in `fregre.pc.cv`. For `"basis"` a value for `type.CV` in `fregre.basis.cv` such as `GCV.S`.
`pmax`	maximum size of the basis expansion to consider in when using `p.criterion`.
`type.basis`	type of basis if `est.method` = `"basis"`.
`projs`	a `fdata` object containing the random directions employed to project `X.fdata`. If numeric, the convenient value for `ncomp` in `rdir.pc`.
`verbose`	whether to show or not information about the testing progress.
`same.rwild`	wether to employ the same wild bootstrap residuals for different projections or not.
`seed`	seed to be employed with `set.seed` for initializing random projections.
`...`	further arguments passed to create.basis (not `rangeval` that is taken as the `rangeval` of `X.fdata`).

Value

An object with class `"htest"` whose underlying structure is a list containing the following components:

- `p.values.fdr`: a matrix of size `c(n.proj, 2)`, containing in each row the FDR p-values of the CvM and KS tests up to that projection.
- `proj.statistics`: a matrix of size `c(max(n.proj), 2)` with the value of the test statistic on each projection.
- `boot.proj.statistics`: an array of size `c(max(n.proj), 2,B)` with the values of the bootstrap test statistics for each projection.
- `proj.p.values`: a matrix of size `c(max(n.proj), 2)`.
- `method`: information about the test performed and the kind of estimation performed.
- `B`: number of bootstrap replicates used.

- `n.proj`: number of projections specified.
- `projs`: random directions employed to project `X.fdata`.
- `type.basis`: type of basis for `est.method` = `"basis"`.
- `beta.est`: estimated functional parameter $\hat{\beta}$ in the composite hypothesis. For the simple hypothesis, `beta0.fdata`.
- `p`: number of basis elements considered for estimation of β.
- `p.criterion`: criterion employed for selecting `p`.
- `data.name`: the character string "Y = <X, b> + e".

Author(s)

Eduardo Garcia-Portugues (`<edgarcia@est-econ.uc3m.es>`) and Manuel Febrero-Bande (`<manuel.febrero@usc.es>`).

References

Cuesta-Albertos, J.A., Garcia-Portugues, E., Febrero-Bande, M. and Gonzalez-Manteiga, W. (2017). Goodness-of-fit tests for the functional linear model based on randomly projected empirical processes. arXiv:1701.08363. `https://arxiv.org/abs/1701.08363`

Garcia-Portugues, E., Gonzalez-Manteiga, W. and Febrero-Bande, M. (2014). A goodness-of-fit test for the functional linear model with scalar response. Journal of Computational and Graphical Statistics, 23(3), 761–778. doi:10.1080/10618600.2013.812519

Examples

```
## Not run:
# Simulated example

set.seed(345678)
t <- seq(0, 1, l = 101)
n <- 100
X <- r.ou(n = n, t = t, alpha = 2, sigma = 0.5)
beta0 <- fdata(mdata = cos(2 * pi * t) - (t - 0.5)^2, argvals = t,
               rangeval = c(0,1))
Y <- inprod.fdata(X, beta0) + rnorm(n, sd = 0.1)

# Test all cases
rp.flm.test(X.fdata = X, Y = Y, est.method = "pc")
rp.flm.test(X.fdata = X, Y = Y, est.method = "pls")
rp.flm.test(X.fdata = X, Y = Y, est.method = "basis",
            p.criterion = fda.usc.devel::GCV.S)
rp.flm.test(X.fdata = X, Y = Y, est.method = "pc", p = 5)
rp.flm.test(X.fdata = X, Y = Y, est.method = "pls", p = 5)
rp.flm.test(X.fdata = X, Y = Y, est.method = "basis", p = 5)
rp.flm.test(X.fdata = X, Y = Y, beta0.fdata = beta0)

# Composite hypothesis: do not reject FLM
rp.test <- rp.flm.test(X.fdata = X, Y = Y, est.method = "pc")
rp.test$p.values.fdr
pcvm.test <- flm.test(X.fdata = X, Y = Y, est.method = "pc", B = 1e3,
                      plot.it = FALSE)
pcvm.test

# Estimation of beta
par(mfrow = c(1, 3))
plot(X, main = "X")
plot(beta0, main = "beta")
lines(rp.test$beta.est, col = 2)
lines(pcvm.test$beta.est, col = 3)
plot(density(Y), main = "Density of Y", xlab = "Y", ylab = "Density")
rug(Y)

# Simple hypothesis: do not reject beta = beta0
rp.flm.test(X.fdata = X, Y = Y, beta0.fdata = beta0)$p.values.fdr
flm.test(X.fdata = X, Y = Y, beta0.fdata = beta0, B = 1e3, plot.it = FALSE)

# Simple hypothesis: reject beta = beta0^2
rp.flm.test(X.fdata = X, Y = Y, beta0.fdata = beta0^2)$p.values.fdr
flm.test(X.fdata = X, Y = Y, beta0.fdata = beta0^2, B = 1e3, plot.it = FALSE)

# Tecator dataset

# Load data
data(tecator)
absorp <- tecator$absorp.fdata
ind <- 1:129 # or ind <- 1:215
x <- absorp[ind, ]
y <- tecator$y$Fat[ind]

# Composite hypothesis
rp.tecat <- rp.flm.test(X.fdata = x, Y = y, est.method = "pc")
pcvm.tecat <- flm.test(X.fdata = x, Y = y, est.method = "pc", B = 1e3,
                       plot.it = FALSE)
rp.tecat$p.values.fdr[c(5, 10), ]
pcvm.tecat

# Simple hypothesis
zero <- fdata(mdata = rep(0, length(x$argvals)), argvals = x$argvals,
              rangeval = x$rangeval)
rp.flm.test(X.fdata = x, Y = y, beta0.fdata = zero)
flm.test(X.fdata = x, Y = y, beta0.fdata = zero, B = 1e3)

# With derivatives
rp.tecat <- rp.flm.test(X.fdata = fdata.deriv(x, 1), Y = y, est.method = "pc")
rp.tecat$p.values.fdr
rp.tecat <- rp.flm.test(X.fdata = fdata.deriv(x, 2), Y = y, est.method = "pc")
rp.tecat$p.values.fdr

# AEMET dataset

# Load data
data(aemet)
wind.speed <- apply(aemet$wind.speed$data, 1, mean)
```

```
temp <- aemet$temp

# Remove the 5% of the curves with less depth (i.e. 4 curves)
par(mfrow = c(1, 1))
res.FM <- depth.FM(temp, draw = TRUE)
qu <- quantile(res.FM$dep, prob = 0.05)
l <- which(res.FM$dep <= qu)
lines(aemet$temp[l], col = 3)

# Data without outliers
wind.speed <- wind.speed[-l]
temp <- temp[-l]

# Composite hypothesis
rp.aemet <- rp.flm.test(X.fdata = temp, Y = wind.speed, est.method = "pc")
pcvm.aemet <- flm.test(X.fdata = temp, Y = wind.speed, B = 1e3,
                       est.method = "pc", plot.it = FALSE)
rp.aemet$p.values.fdr
apply(rp.aemet$p.values.fdr, 2, range)
pcvm.aemet

# Simple hypothesis
zero <- fdata(mdata = rep(0, length(temp$argvals)), argvals = temp$argvals,
              rangeval = temp$rangeval)
flm.test(X.fdata = temp, Y = wind.speed, beta0.fdata = zero, B = 1e3,
         plot.it = FALSE)
rp.flm.test(X.fdata = temp, Y = wind.speed, beta0.fdata = zero)

## End(Not run)
```

rproc2fdata *Simulate several random processes.*

Description

Simulate Functional Data from different processes: Ornstein Uhlenbeck, Brownian, Fractional Brownian, Gaussian or Exponential variogram.

Usage

```
rproc2fdata(
  n,
  t = NULL,
  mu = rep(0, length(t)),
  sigma = 1,
  par.list = list(scale = 1, theta = 0.2 * diff(rtt), H = 0.5),
  norm = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`n`	Number of functional curves to be generated.
`t`	Discretization points.
`mu`	`vector` which specifies the trend values at the discretization points, by default `mu`=$\mu(t) = 0$. If `mu` is a `fdata` class object, `t`=`argvals(mu)`.
`sigma`	A positive-definite symmetric matrix, $\Sigma_{s,t}$, specifying the covariance matrix among grid points. If `sigma` is a `scalar`, creates a random Gaussian process with $\Sigma_{s,t}$ =`sigmaI` (by default `sigma=1`). If `sigma` is a `vector`, creates a random Gaussian process with $\Sigma_{s,t}$ =`diag(sigma)`. If `sigma` is a character: create a random process using the covariance matrix $\Sigma_{s,t}$ indicated in the argument, • `"OU"` or `"OrnsteinUhlenbeck"`, creates a random Ornstein Uhlenbeck process with $\Sigma_{s,t} = \frac{\sigma^2}{2\theta} e^{-\theta(s+t)} \left(e^{2\theta(s+t)} - 1\right)$, by default $\theta = 1/(3range(t))$, $\sigma^2 = 1$. • `"brownian"` or `"wiener"`, creates a random Wiener process with $\Sigma_{s,t} = \sigma^2 min(s,t)$, by default $\sigma^2 = 1$. • `"fbrownian"`, creates a random fractional brownian process with $\Sigma_{s,t} = \sigma^{2H}/2|s|^{2H} + |t|^{2H} - |s-t|^{2H}$, by default $\sigma^2 = 1$ and $H = 0.5$ (brownian process). • `"vexponential"`, creates a random gaussian process with exponential variogram $\Sigma_{s,t} = \sigma^2 e^{\left(-\frac{|s-t|}{\theta}\right)}$, by default $\theta = 0.2range(t)$, $\sigma^2 = 1$.
`par.list`	List of parameter to process, by default `"scale"` $\sigma^2 = 1$, `"theta"` $\theta = 0.2range(t)$ and `"H"`=0.5.
`norm`	If `TRUE` the norm of random projection is 1. Default is `FALSE`
`verbose`	If `TRUE`, information about procedure is printed. Default is `FALSE`.
`...`	Further arguments passed to or from other methods.

Value

Return the functional random processes as a `fdata` class object.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente `<manuel.oviedo@udc.es>`

Examples

```
## Not run:
par(mfrow=c(3,2))
lent<-30
tt<-seq(0,1,len=lent)
mu<-fdata(rep(0,lent),tt)
plot(rproc2fdata(200,t=tt,sigma="OU",par.list=list("scale"=1)))
plot(rproc2fdata(200,mu=mu,sigma="OU",par.list=list("scale"=1)))
plot(rproc2fdata(200,t=tt,sigma="vexponential"))
plot(rproc2fdata(200,t=tt,sigma=1:lent))
plot(rproc2fdata(200,t=tt,sigma="brownian"))
plot(rproc2fdata(200,t=tt,sigma="wiener"))
#plot(rproc2fdata(200,seq(0,1,len=30),sigma="oo")) # this is an error

## End(Not run)
```

---

rwild *Wild bootstrap residuals*

Description

The wild bootstrap residuals are computed as $residuals * V$, where V is a sampling from a random variable (see details section).

Usage

```
rwild(residuals, type = "golden")
```

Arguments

`residuals`	residuals
`type`	Type of distribution of V.

Details

For the construction of wild bootstrap residuals, sampling from a random variable V such that $E[V^2] = 0$ and $E[V] = 0$ is needed. A simple and suitable V is obtained with a discrete variable of the form:

- "golden", Sampling from golden section bootstrap values suggested by Mammen (1993).

  $$P\left\{V = \frac{1-\sqrt{5}}{2}\right\} = \frac{5+\sqrt{5}}{10}\, and\, P\left\{V = \frac{1+\sqrt{5}}{2}\right\} = \frac{5-\sqrt{5}}{10},$$

  which leads to the *golden section bootstrap*.
- "Rademacher", Sampling from Rademacher distribution values $\big\{-1,\,1\big\}$ with probabilities $\big\{\frac{1}{2},\,\frac{1}{2}\big\}$, respectively.
- "normal", Sampling from a standard normal distribution.

Value

The wild bootstrap residuals computed using a sample of the random variable V.

Author(s)

Eduardo Garcia-Portugues, Manuel Febrero-Bande and Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>.

References

Mammen, E. (1993). *Bootstrap and wild bootstrap for high dimensional linear models*. Annals of Statistics 21, 255 285. Davidson, R. and E. Flachaire (2001). *The wild bootstrap, tamed at last*. working paper IER1000, Queens University.

See Also

`flm.test`, `flm.Ftest`, `dfv.test`, `fregre.bootstrap`

Examples

```
n<-100
# For golden wild bootstrap variable
e.boot0=rwild(rep(1,len=n),"golden")
# Construction of wild bootstrap residuals
e=rnorm(n)
e.boot1=rwild(e,"golden")
e.boot2=rwild(e,"Rademacher")
e.boot3=rwild(e,"normal")
summary(e.boot1)
summary(e.boot2)
summary(e.boot3)
```

`S.basis` *Smoothing matrix with roughness penalties by basis representation.*

Description

Provides the smoothing matrix `S` with roughness penalties.

Usage

```
S.basis(tt, basis, lambda = 0, Lfdobj = vec2Lfd(c(0, 0)), w = NULL, ...)
```

Arguments

`tt`	Discretization points.
`basis`	Basis to use. See create.basis.
`lambda`	A roughness penalty. By default, no penalty `lambda`=0.
`Lfdobj`	See eval.penalty.
`w`	Optional case weights.
`...`	Further arguments passed to or from other methods. Arguments to be passed by default to create.basis

Details

Provides the smoothing matrix S for the discretization points `tt` and `bbasis` with roughness penalties. If `lambda=0` is not used penalty, else a basis roughness penalty matrix is caluclated using getbasispenalty.

Value

Return the smoothing matrix `S`.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ramsay, James O. and Silverman, Bernard W. (2006). *Functional Data Analysis*, 2nd ed., Springer, New York.

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

See Also

See Also as `S.np`

Examples

```
## Not run:
np=101
tt=seq(0,1,len=np)

nbasis=11
base1 <- create.bspline.basis(c(0, np), nbasis)
base2 <- create.fourier.basis(c(0, np), nbasis)

S1<-S.basis(tt,basis=base1,lambda=3)
image(S1)
S2<-S.basis(tt,basis=base2,lambda=3)
image(S2)

## End(Not run)
```

`S.np` *Smoothing matrix by nonparametric methods*

Description

Provides the smoothing matrix `S` for the discretization points `tt`

Usage

```
S.LLR(tt, h, Ker = Ker.norm, w = NULL, cv = FALSE)

S.LPR(tt, h, p = 1, Ker = Ker.norm, w = NULL, cv = FALSE)

S.LCR(tt, h, Ker = Ker.norm, w = NULL, cv = FALSE)

S.KNN(tt, h = NULL, Ker = Ker.unif, w = NULL, cv = FALSE)

S.NW(tt, h = NULL, Ker = Ker.norm, w = NULL, cv = FALSE)
```

Arguments

`tt`	Vector of discretization points or distance matrix `mdist`
`h`	Smoothing parameter or bandwidth. In S.KNN, number of k-nearest neighbors.
`Ker`	Type of kernel used, by default normal kernel.
`w`	Optional case weights.
`cv`	If `TRUE`, cross-validation is done.
`p`	Polynomial degree. be passed by default to create.basis

Details

Options:

- `S.NW`: Nadaraya-Watson kernel estimator with bandwidth parameter `h`.
- `S.LLR`: Local Linear Smoothing with bandwidth parameter `h`.
- `S.KNN`: K nearest neighbors estimator with parameter `knn`.
- `S.LPR`: Polynomial Local Regression Estimator with parameter of polynomial `p` and of kernel `Ker`.
- `S.LCR`: Local Cubic Regression Estimator with kernel `Ker`.

Value

Return the smoothing matrix `S`.

- `S.LLR`: Local Linear Smoothing.
- `S.NW`: Nadaraya-Watson kernel estimator.
- `S.KNN`: k nearest neighbors estimator.
- `S.LPR`: Local Polynomial Regression Estimator.
- `S.LCR`: Cubic Polynomial Regression.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Wasserman, L. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2006.

Opsomer, J. D., and Ruppert, D. (1997). Fitting a bivariate additive model by local polynomial regression. *The Annals of Statistics*, 25(1), 186-211.

See Also

See Also as `S.basis`

Examples

```
## Not run:
  tt <- 1:101
  S <- S.LLR(tt,h=5)
  S2 <- S.LLR(tt,h=10,Ker=Ker.tri)
  S3 <- S.NW(tt,h=10,Ker=Ker.tri)
  S4 <- S.KNN(tt,h=5,Ker=Ker.tri)
  par(mfrow=c(2,3))
  image(S)
  image(S2)
  image(S3)
  image(S4)
  S5 <- S.LPR(tt,h=10,p=1, Ker=Ker.tri)
  S6 <- S.LCR(tt,h=10,Ker=Ker.tri)
  image(S5)
  image(S6)

## End(Not run)
```

`semimetric.basis` *Proximities between functional data*

Description

Approximates semi-metric distances for functional data of class `fdata` or `fd`.

Usage

```
semimetric.basis(
  fdata1,
  fdata2 = fdata1,
  nderiv = 0,
  type.basis1 = NULL,
  nbasis1 = NULL,
  type.basis2 = type.basis1,
  nbasis2 = NULL,
    ...
)
```

Arguments

`fdata1`	Functional data 1 or curve 1.
`fdata2`	Functional data 2 or curve 2.
`nderiv`	Order of derivation, used in `deriv.fd`
`type.basis1`	Type of Basis for `fdata1`.
`nbasis1`	Number of Basis for `fdata1`.
`type.basis2`	Type of Basis for `fdata2`.
`nbasis2`	Number of Basis for `fdata2`.
`...`	Further arguments passed to or from other methods.

Details

Approximates semi-metric distances for functional data of two `fd` class objects. If functional data are not functional `fd` class, the `semimetric.basis` function creates a basis to represent the functional data, by default is used create.bspline.basis and the `fdata` class object is converted to `fd` class using the Data2fd.
The function calculates distances between the derivative of order `nderiv` of curves using deriv.fd function.

Value

Returns a proximities matrix between functional data.

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

See Also

See also `metric.lp`, `semimetric.NPFDA` and deriv.fd

Examples

```
## Not run:
data(phoneme)
DATA1<-phoneme$learn[c(30:50,210:230)]
DATA2<-phoneme$test[231:250]
a1=semimetric.basis(DATA1,DATA2)
a2=semimetric.basis(DATA1,DATA2,type.basis1="fourier",
nbasis1=11, type.basis2="fourier",nbasis2=11)
fd1 <- fdata2fd(DATA1)
fd2 <- fdata2fd(DATA2)
a3=semimetric.basis(fd1,fd2)
a4=semimetric.basis(fd1,fd2,nderiv=1)
```

```
## End(Not run)
```

semimetric.NPFDA *Proximities between functional data (semi-metrics)*

Description

Computes semi-metric distances of functional data based on Ferraty F and Vieu, P. (2006).

Usage

```
semimetric.deriv(
  fdata1,
  fdata2 = fdata1,
  nderiv = 1,
  nknot = ifelse(floor(ncol(DATA1)/3) > floor((ncol(DATA1) - nderiv - 4)/2),
    floor((ncol(DATA1) - nderiv - 4)/2), floor(ncol(DATA1)/3)),
  ...
)

semimetric.fourier(
  fdata1,
  fdata2 = fdata1,
  nderiv = 0,
  nbasis = ifelse(floor(ncol(DATA1)/3) > floor((ncol(DATA1) - nderiv - 4)/2),
    floor((ncol(DATA1) - nderiv - 4)/2), floor(ncol(DATA1)/3)),
  period = NULL,
  ...
)

semimetric.hshift(fdata1, fdata2 = fdata1, t = 1:ncol(DATA1), ...)

semimetric.mplsr(fdata1, fdata2 = fdata1, q = 2, class1, ...)

semimetric.pca(fdata1, fdata2 = fdata1, q = 1, ...)
```

Arguments

`fdata1`	Functional data 1 or curve 1. `DATA1` with dimension (`n1` x `m`), where `n1` is the number of curves and `m` are the points observed in each curve.
`fdata2`	Functional data 2 or curve 2. `DATA1` with dimension (`n2` x `m`), where `n2` is the number of curves and `m` are the points observed in each curve.
`nderiv`	Order of derivation, used in `semimetric.deriv` and `semimetric.fourier`

`nknot`	semimetric.deriv argument: number of interior knots (needed for defining the B-spline basis).
`...`	Further arguments passed to or from other methods.
`nbasis`	`semimetric.fourier`: size of the basis.
`period`	`semimetric.fourier`:allows to select the period for the fourier expansion.
`t`	`semimetric.hshift`: vector which defines `t` (one can choose `1,2,...,nbt` where `nbt` is the number of points of the discretization)
`q`	If `semimetric.pca`: the retained number of principal components. If `semimetric.mplsr`: the retained number of factors.
`class1`	`semimetric.mplsr`: vector containing a categorical response which corresponds to class number for units stored in `DATA1`.

Details

`semimetric.deriv`: approximates L_2 metric between derivatives of the curves based on ther B-spline representation. The derivatives set with the argument `nderiv`.
`semimetric.fourier`: approximates L_2 metric between the curves based on ther B-spline representation. The derivatives set with the argument `nderiv`.
`semimetric.hshift`: computes distance between curves taking into account an horizontal shift effect.
`semimetric.mplsr`: computes distance between curves based on the partial least squares method.
`semimetric.pca`: computes distance between curves based on the functional principal components analysis method.

In the next semi-metric functions the functional data X is approximated by k_n elements of the Fourier, B–spline, PC or PLS basis using, $\hat{X_i}=\sum_{k=1}^{k_n}\nu_{k,i}\xi_k$, where ν_k are the coefficient of the expansion on the basis function $\{\xi_k\}_{k=1}^{\infty}$.
The distances between the q-order derivatives of two curves X_1 and X_2 is,

$$d_2^{(q)}\left(X_1,X_2\right)_{k_n}=\sqrt{\frac{1}{T}\int_T\left(X_1^{(q)}(t)-X_2^{(q)}(t)\right)^2dt}$$

where $X_i^{(q)}\left(t\right)$ denot the q derivative of X_i.

`semimetric.deriv` and `semimetric.fourier` function use a B-spline and Fourier approximation respectively for each curve and the derivatives are directly computed by differentiating several times their analytic form, by default `q=1` and `q=0` respectively. `semimetric.pca` and `semimetric.mprls` function compute proximities between curves based on the functional principal components analysis (FPCA) and the functional partial least square analysis (FPLS), respectively. The FPC and FPLS reduce the functional data in a reduced dimensional space (q components). `semimetric.mprls` function requires a scalar response.

$$d_2^{(q)}\left(X_1,X_2\right)_{k_n}\approx\sqrt{\sum_{k=1}^{k_n}\left(\nu_{k,1}-\nu_{k,2}\right)^2\left\|\xi_k^{(q)}\right\|dt}$$

`semimetric.hshift` computes proximities between curves taking into account an horizontal shift effect.

$$d_{hshift}\left(X_1, X_2\right) = \min_{h \in [-mh, mh]} d_2(X_1(t), X_2(t+h))$$

where mh is the maximum horizontal shifted allowed.

Value

Returns a proximities matrix between two functional datasets.

Source

https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/

References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis.* Springer Series in Statistics, New York.

Ferraty, F. and Vieu, P. (2006). *NPFDA in practice*. Free access on line at https://www.math.univ-toulouse.fr/~ferraty/SOFTWARES/NPFDA/

See Also

See also metric.lp and semimetric.basis

Examples

```
## Not run:
# INFERENCE PHONDAT
data(phoneme)
ind=1:100 # 2 groups
mlearn<-phoneme$learn[ind,]
mtest<-phoneme$test[ind,]
n=nrow(mlearn[["data"]])
np=ncol(mlearn[["data"]])
mdist1=semimetric.pca(mlearn,mtest)
mdist2=semimetric.pca(mlearn,mtest,q=2)
mdist3=semimetric.deriv(mlearn,mtest,nderiv=0)
mdist4=semimetric.fourier(mlearn,mtest,nderiv=2,nbasis=21)
#uses hshift function
#mdist5=semimetric.hshift(mlearn,mtest) #takes a lot
glearn<-phoneme$classlearn[ind]
#uses mplsr function
mdist6=semimetric.mplsr(mlearn,mtest,5,glearn)
mdist0=metric.lp(mlearn,mtest)
b=as.dist(mdist6)
c2=hclust(b)
plot(c2)
memb <- cutree(c2, k = 2)
table(memb,phoneme$classlearn[ind])

## End(Not run)
```

subset.fdata *Subsetting*

Description

Return subsets of fdata which meet conditions.

Usage

```
## S3 method for class 'fdata'
subset(x, subset, select, drop = TRUE, ...)
```

Arguments

x	object to be subsetted (fdata class).
subset	logical expression indicating elements or rows to keep.
select	logical expression indicating points or columns to keep.
drop	passed on to [indexing operator.
...	Further arguments passed to or from other methods.

Value

An object similar to x contain just the selected elements.

See Also

See subset and fdata.

summary.classif *Summarizes information from kernel classification methods.*

Description

Summary function for classif.knn or classif.kernel.

Usage

```
## S3 method for class 'classif'
summary(object, ...)

## S3 method for class 'classif'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

object	Estimated by kernel classification.
...	Further arguments passed to or from other methods.
x	Estimated by kernel classification.
digits	how many significant digits are to be used for numeric and complex x.

Details

object from classif.knn or classif.kernel

Value

Shows:

- -Probability of correct classification by group prob.classification.
- -Confusion matrix between the theoretical groups and estimated groups.
- -Highest probability of correct classification max.prob.

If the object is returned from the function classif.knn

- -Vector of probability of correct classification by number of neighbors knn.
- -Optimal number of neighbors: knn.opt.

If the object is returned from the function: classif.kernel

- -Vector of probability of correct classification by banwidth h.
- -Functional measure of closeness (optimal distance, h.opt).

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

See Also as: classif.knn, classif.kernel
and summary.classif

Examples

```
## Not run:
data(phoneme)
mlearn<-phoneme[["learn"]]
glearn<-phoneme[["classlearn"]]
out=classif.knn(glearn,mlearn,knn=c(3,5,7))
summary(out)
out2=classif.kernel(glearn,mlearn,h=2^(0:5))
summary(out2)

## End(Not run)
```

`summary.fdata.comp` *Correlation for functional data by Principal Component Analysis*

Description

Summary of functional principal components

Usage

```
## S3 method for class 'fdata.comp'
summary(object, biplot = TRUE, ...)
```

Arguments

`object`	fdata.comp class object calculated by: `fdata2pc`, `fdata2pls`, `fregre.pc` or `fregre.pls`.
`biplot`	=TRUE draw the biplot and PC (or PLS) components.
`...`	Further arguments passed to or from other methods.

Value

If `corplot`=TRUE, are displaying the biplot between the PC (or PLS) components.
If `ask`=TRUE, draw each graph in a window, waiting to confirm the change of page with a click of the mouse or pressing ENTER. If `ask`=FALSE draw graphs in one window.

Author(s)

Manuel Febrero-Bande and Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S*. Springer-Verlag.

See Also

See Also as `fdata2pc`, `fdata2pls` and `cor`

Examples

```
## Not run:
library(fda.usc)
n <- 200
tt <- seq(0,1,len=101)
x0 <- rproc2fdata(n,tt,sigma="wiener")
x1 <- rproc2fdata(n,tt,sigma=0.1)
x <- x0*3+x1
```

```
beta <- tt*sin(2*pi*tt)^2
fbeta <- fdata(beta,tt)
pc1 <- fdata2pc(x,3)
summary.fdata.comp(pc1)
y <- inprod.fdata(x,fbeta) #+ rnorm(n,sd=0.1)
pls1 <- fdata2pls(x,y,2)
summary(pls1)

## End(Not run)
```

`summary.fregre.fd` *Summarizes information from fregre.fd objects.*

Description

Summary function for `fregre.pc`, `fregre.basis`, `fregre.pls`, `fregre.np` and `fregre.plm` functions.

Shows:

-Call.
-R squared.
-Residual variance.
-Index of possible atypical curves or possible outliers.
-Index of possible influence curves.

If the `fregre.fd` object comes from the `fregre.pc` then shows:

-Variability of explicative variables explained by Principal Components.
-Variability for each principal components -PC-.

If draw=TRUE plot:

-y vs y fitted values.
-Residuals vs fitted values.
-Standarized residuals vs fitted values.
-Levarage.
-Residual boxplot.
-Quantile-Quantile Plot (qqnorm).

If `ask`=FALSE draw graphs in one window, by default. If `ask`=TRUE, draw each graph in a window, waiting to confirm.

Usage

```
## S3 method for class 'fregre.fd'
summary(object, times.influ = 3, times.sigma = 3, draw = TRUE, ...)
```

Arguments

`object`	Estimated by functional regression, `fregre.fd` object.
`times.influ`	Limit for detect possible infuence curves.
`times.sigma`	Limit for detect possible oultiers or atypical curves.
`draw`	=TRUE draw estimation and residuals graphics.
`...`	Further arguments passed to or from other methods.

Value

- `Influence`: Vector of influence measures.
- `i.influence`: Index of possible influence curves.
- `i.atypical`: Index of possible atypical curves or possible outliers.

Author(s)

Manuel Febrero-Bande and Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

See Also

Summary function for `fregre.pc`, `fregre.basis`, `fregre.pls`,
`fregre.np` and `fregre.plm`.

Examples

```
## Not run:
# Ex 1. Simulated data
n= 200;tt= seq(0,1,len=101)
x0<-rproc2fdata(n,tt,sigma="wiener")
x1<-rproc2fdata(n,tt,sigma=0.1)
x<-x0*3+x1
beta = tt*sin(2*pi*tt)^2
fbeta = fdata(beta,tt)
y<-inprod.fdata(x,fbeta)+rnorm(n,sd=0.1)

# Functional regression
res=fregre.pc(x,y,l=c(1:5))
summary(res,3,ask=TRUE)

res2=fregre.pls(x,y,l=c(1:4))
summary(res2)

res3=fregre.pls(x,y)
summary(res3)
```

```
## End(Not run)
```

`summary.fregre.gkam` *Summarizes information from fregre.gkam objects.*

Description

Summary function for `fregre.gkam` function.

-Family used.
-Number or iteration of algorithm and if it has converged.
-Residual and null deviance.
-Number of data.

Produces a list of summary information for a fitted fregre.np object for each functional covariate.

-Call.
-R squared.
-Residual variance.
-Index of possible atypical curves or possible outliers.
-Index of possible influence curves.

If draw=TRUE plot:

-y vs y fitted values.
-Residuals vs fitted values.
-Residual boxplot.
-Quantile-Quantile Plot (qqnorm).
-Plot for a each single model term.

If ask=FALSE draw graphs in one window, by default. If ask=TRUE, draw each graph in a window, waiting to confirm.

Usage

```
## S3 method for class 'fregre.gkam'
summary(object, draw = TRUE, selec = NULL, times.influ = 3, ...)
```

Arguments

`object`	Estimated by functional regression, `fregre.fd` object.
`draw`	=TRUE draw estimation and residuals graphics.
`selec`	Allows the plot for a single model term to be selected for printing. e.g. if you just want the plot for the second smooth term set selec=2. .
`times.influ`	Limit for detect possible infuence curves.
`...`	Further arguments passed to or from other methods.

Author(s)

Manuel Febrero-Bande and Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

See Also

Summary function for `fregre.gkam`.

Examples

```
## Not run:
# Time consuming
data(tecator)
ind<-1:129
ab=tecator$absorp.fdata[ind]
ab2=fdata.deriv(ab,2)
yfat=as.integer(cut(tecator$y[ind,"Fat"],c(0,15,100)))-1
xlist=list("df"=data.frame(yfat),"ab2"=ab2,"ab"=ab)
f<-yfat~ab+ab2
res=fregre.gkam(f, data = xlist, family = binomial("logit"),
                control = list(maxit = 2))
summary(res)


## End(Not run)
```

`tecator` *tecator data*

Description

Water, Fat and Protein content of meat samples

Format

The format is:
`..$absorp.fdata`: absorbance data. `fdata` class object with:

- `"data"`: Matrix of class `fdata` with 215 curves (rows) discretized in 100 points or argvals (columns).
- `"argvals"`: 100 discretization points from 850 to 1050nm
- `"rangeval"`=(850,1050): range(`"argvals"`)
- `"names"` list with: `main` an overall title "Tecator data set", `xlab` title for `x` axis "Wavelength (nm)" and `ylab` title for `y` axis "Absorbances".

`..$y`: the percentages of Fat, Water and Protein. The three contents are determined by analytic chemistry.

Details

`absorp.fdata` absorbance data for 215 samples. The first 129 were originally used as a training set endpoints the percentages of Fat, Water and Protein.
for more details see tecator package

Author(s)

Manuel Febrero-Bande and Manuel Oviedo de la Fuente <`manuel.oviedo@udc.es`>

Examples

```
data(tecator)
names(tecator)
names(tecator$absorp.fdata)
names(tecator$y)
names(tecator$y)
class(tecator$absorp.fdata)
class(tecator$y)
dim(tecator$absorp.fdata)
dim(tecator$y)
```

`Var.y` *Sampling Variance estimates*

Description

Sampling variance or error variance estimates for regression estimates.

Usage

```
Var.y(y, S, Var.e = NULL)
```

Arguments

y	fdata class object.
S	Smoothing matrix calculated by S.basis or S.NW functions.
Var.e	Error Variance Estimates. If Var.e=NULL, Var.e is calculated.

Value

Var.y: returns the sampling variance of the functional data. Var.e: returns the sampling error variance of the functional data.

Author(s)

Manuel Febrero-Bande, Manuel Oviedo de la Fuente <manuel.oviedo@udc.es>

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

See Also

See Also as Var.e

Examples

```
a1<-seq(0,1,by=.01)
a2=rnorm(length(a1),sd=0.2)
f1<-(sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
nc<-50
np<-length(f1)
tt=1:101
mdata<-matrix(NA,ncol=np,nrow=nc)
for (i in 1:nc) mdata[i,]<- (sin(2*pi*a1))+rnorm(length(a1),sd=0.2)
mdata<-fdata(mdata,tt)
S=S.NW(tt,h=0.15)
var.e<-Var.e(mdata,S)
var.y<-Var.y(mdata,S)
var.y2<-Var.y(mdata,S,var.e) #the same
```

`weights4class` *Weighting tools*

Description

computes inverse probability weighting.

Usage

```
weights4class(x, type = c("equal", "inverse"))
```

Arguments

`x`	A vector of the labels, true class or observed response. Can be `numeric`, `character`, or `factor`
`type`	Type of weights.

See Also

Other performance: `accuracy`

Index