Package ‘fdapace’

July 18, 2019

Type Package

Title Functional Data Analysis and Empirical Dynamics

URL https://github.com/functionaldata/tPACE

BugReports https://github.com/functionaldata/tPACE/issues

Version 0.5.0

Date 2019-03-20,

Maintainer Pantelis Z. Hadjipantelis <pantelis@ucdavis.edu>

Description Provides implementation of various methods of Functional Data Analysis (FDA) and Empirical Dynamics. The core of this package is Functional Principal Component Analysis (FPCA), a key technique for functional data analysis, for sparsely or densely sampled random trajectories and time courses, via the Principal Analysis by Conditional Estimation (PACE) algorithm or numerical integration. PACE is useful for the analysis of data that have been generated by a sample of underlying (but usually not fully observed) random trajectories. It does not rely on pre-smoothing of trajectories, which is problematic if functional data are sparsely sampled. PACE provides options for functional regression and correlation, for Longitudinal Data Analysis, the analysis of stochastic processes from samples of realized trajectories, and for the analysis of underlying dynamics. The core computational algorithms are implemented using the 'Eigen' C++ library for numerical linear algebra and 'RcppEigen' ``glue''.

License BSD_3_clause + file LICENSE

LazyData false

Imports Rcpp (>= 0.11.5), Hmisc, MASS, Matrix, pracma, numDeriv

LinkingTo Rcpp, RcppEigen

Suggests plot3D, rgl, aplpack, mgcv, ks, gtools, knitr, EMCluster, minqa, testthat

NeedsCompilation yes

RoxygenNote 6.1.1.9000

VignetteBuilder knitr

Author Xiongtao Dai [aut],
Pantelis Z. Hadjipantelis [aut, cre],
Kynghee Han [aut],
Hao Ji [aut],

Shu-Chin Lin [ctb],
Hans-Georg Mueller [cph, ths],
Jane-Ling Wang [cph, ths]

R topics documented:

BwNN . . . 3
CheckData . . . 4
CheckOptions . . . 4
ConvertSupport . . . 5
CreateBasis . . . 5
CreateBWPlot . . . 6
CreateCovPlot . . . 6
CreateDesignPlot . . . 7
CreateDiagnosticsPlot . . . 8
CreateFuncBoxPlot . . . 9
CreateModeOfVarPlot . . . 10
CreateOutliersPlot . . . 10
CreatePathPlot . . . 12
CreateScreePlot . . . 13
CreateStringingPlot . . . 13
cumtrapzRcpp . . . 14
DynCorr . . . 14
Dyn_test . . . 15
FAM . . . 16
FCCor . . . 19
FClust . . . 21
FCReg . . . 22
fdapace . . . 24
fitted.FPCA . . . 24
fitted.FPCAder . . . 26
FOptDes . . . 27
FPCA . . . 28
FPCAder . . . 31
FPCquantile . . . 33
FSVD . . . 34
FVPA . . . 36
GetCrCorYX . . . 37
GetCrCorYZ . . . 37
GetCrCovYX . . . 38
GetCrCovYZ . . . 39
GetNormalisedSample . . . 40
kCFC . . . 41
Lwls1D . . . 42
Lwls2D . . . 43
Lwls2DDeriv . . . 44
MakeBWtoZscore02y . . . 45
MakeFPCAInputs . . . 45
MakeGPFunctionalData . . . 46
MakeHCtoZscore02y . . . 46
MakeLNtoZscore02y . . . 47
MakeSparseGP . . . 47

medfly25 48
MultiFAM 48
NormCurvToArea 52
predict.FPCA 53
print.FPCA 54
print.FSVD 54
print.WFDA 55
SBFitting 55
SelectK 57
SetOptions 58
Sparsify 59
str.FPCA 59
Stringing 60
trapzRcpp 61
TVAM 61
VCAM 63
WFDA 66
Wiener 68

Index **69**

BwNN *Minimum bandwidth based on kNN criterion.*

Description

Input a list of time points Lt, and the number of unique neighbors k and get the minimum bandwidth garanteeing k unique neighbours.

Usage

```
BwNN(Lt, k = 3, onlyMean = FALSE, onlyCov = FALSE)
```

Arguments

Lt	n-by-1 list of vectors
k	number of unique neighbors for cov and mu (default = 3)
onlyMean	Indicator to return only the minimum bandwidth for the mean
onlyCov	Indicator to return only the minium bandwidth for the covariance

Examples

```
tinyGrid = list(c(1,7), c(2,3),  6,  c(2,4), c(4,5))
BwNN(tinyGrid, k = 2) # c(3,2)
```

CheckData — *Check data format*

Description

Check if there are problems with the form and basic structure of the functional data 'y' and the recorded times 't'.

Usage

```
CheckData(y, t)
```

Arguments

y	is a n-by-1 list of vectors
t	is a n-by-1 list of vectors

CheckOptions — *Check option format*

Description

Check if the options structure is valid and set the ones that are NULL

Usage

```
CheckOptions(t, optns, n)
```

Arguments

t	is a n-by-1 list of vectors
optns	is an initialized option list
n	is a total number of sample curves

`ConvertSupport` *Convert support of a mu/phi/cov etc. to and from obsGrid and workGrid*

Description

Convert the support of a given function 1-D or 2-D function from 'fromGrd' to 'toGrid'. Both grids need to be sorted. This is a interpolation/convenience function.

Usage

```
ConvertSupport(fromGrid, toGrid, mu = NULL, Cov = NULL, phi = NULL,
  isCrossCov = FALSE)
```

Arguments

`fromGrid`	vector of points with input grid to interpolate from
`toGrid`	vector of points with the target grid to interpolate on
`mu`	any vector of function to be interpolated
`Cov`	a square matrix supported on fromGrid * fromGrid, to be interpolated to toGrid * toGrid.
`phi`	any matrix, each column containing a function to be interpolated
`isCrossCov`	logical, indicating whether the input covariance is a cross-covariance. If so then the output is not made symmetric.

`CreateBasis` *Create an orthogonal basis of K functions in [0, 1], with nGrid points.*

Description

Create an orthogonal basis of K functions in [0, 1], with nGrid points.

Usage

```
CreateBasis(K, pts = seq(0, 1, length.out = 50), type = c("cos", "sin",
  "fourier", "legendre01", "poly"))
```

Arguments

`K`	A positive integer specifying the number of eigenfunctions to generate.
`pts`	A vector specifying the time points to evaluate the basis functions.
`type`	A string for the type of orthogonal basis.

Value

A K by nGrid matrix, each column containing an basis function.

Examples

```
basis <- CreateBasis(3, type='fourier')
head(basis)
```

`CreateBWPlot` *Functional Principal Component Analysis Bandwidth Diagnostics plot*

Description

This function by default creates the mean and first principal modes of variation plots for 50 If provided with a derivative options object (?FPCAder) it will return the differentiated mean and first two principal modes of variations for 50

Usage

```
CreateBWPlot(fpcaObj, derOptns = NULL, bwMultipliers = NULL)
```

Arguments

`fpcaObj`	An FPCA class object returned by FPCA().
`derOptns`	A list of options to control the derivation parameters; see ?FPCAder. If NULL standard diagnostics are returned
`bwMultipliers`	A vector of multipliers that the original 'bwMu' and 'bwCov' will be multiplied by. (default: c(0.50, 0.75, 1.00, 1.25, 1.50)) - default: NULL

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res1 <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=FALSE))
CreateBWPlot(res1)
```

`CreateCovPlot` *Create the covariance surface plot based on the results from FPCA() or FPCder().*

Description

This function will open a new device if not instructed otherwise.

Usage

```
CreateCovPlot(fpcaObj, covPlotType = "Fitted", isInteractive = FALSE,
  colSpectrum = NULL, ...)
```

Arguments

`fpcaObj`	returned object from FPCA().
`covPlotType`	a string specifying the type of covariance surface to be plotted: 'Smoothed': plot the smoothed cov surface 'Fitted': plot the fitted cov surface
`isInteractive`	an option for interactive plot: TRUE: interactive plot; FALSE: printable plot
`colSpectrum`	character vector to be use as input in the 'colorRampPalette' function defining the colouring scheme (default: c('blue','red'))
`...`	other arguments passed into persp3d, persp3D, plot3d or points3D for plotting options

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
CreateCovPlot(res)
```

`CreateDesignPlot` *Create the design plot of the functional data.*

Description

This function will open a new device if not instructed otherwise.

Usage

```
CreateDesignPlot(Lt, obsGrid = NULL, isColorPlot = TRUE,
  noDiagonal = TRUE, addLegend = TRUE, ...)
```

Arguments

`Lt`	a list of observed time points for functional data
`obsGrid`	a vector of sorted observed time points. Default to the unique time points in Lt.
`isColorPlot`	an option for colorful plot: TRUE: create color plot with color indicating counts FALSE: create black and white plot with dots indicating observed time pairs
`noDiagonal`	an option specifying plotting the diagonal design points: TRUE: remove diagonal time pairs FALSE: do not remove diagonal time pairs
`addLegend`	Logical, default TRUE
`...`	Other arguments passed into `plot()`.

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
CreateDesignPlot(sampWiener$Lt, sort(unique(unlist(sampWiener$Lt))))
```

CreateDiagnosticsPlot *Functional Principal Component Analysis Diagnostics plot*

Description

Deprecated. Use `plot.FPCA` instead.

This function plot the results for an FPCA. It prints the design plot, mean function, scree-plot and the first three eigenfunctions of a sample. If provided with a derivative options object (?FPCAder) it will return the differentiated mean and first two principal modes of variations for 50%, 75%, 100%, 125% and 150% of the defined bandwidth choice.

Usage

```
CreateDiagnosticsPlot(...)

## S3 method for class 'FPCA'
plot(x, openNewDev = FALSE, addLegend = TRUE, ...)
```

Arguments

`...`	passed into `plot.FPCA`.
`x`	An FPCA class object returned by FPCA().
`openNewDev`	A logical specifying if a new device should be opened - default: FALSE
`addLegend`	A logical specifying whether to add legend.

Details

The black, red, and green curves stand for the first, second, and third eigenfunctions, respectively. `plot.FPCA` is currently implemented only for the original function, but not a derivative FPCA object.

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res1 <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=FALSE))
plot(res1)
```

`CreateFuncBoxPlot` *Create functional boxplot using 'bagplot', 'KDE' or 'pointwise' methodology*

Description

Using an FPCA object create a functional box-plot based on the function scores. The green line corresponds to the functional median, the dark grey area to the area spanned by the curves within the 25th and 75-th percentile and the light gret to the area spanned by the curves within the 2.5th and 97.5-th percentile.

Usage

```
CreateFuncBoxPlot(fpcaObj, optns = list(), ...)
```

Arguments

`fpcaObj`	An object of class FPCA returned by the function FPCA().
`optns`	A list of options control parameters specified by `list(name=value)`. See 'Details'.
`...`	Additional arguments for the 'plot' function.

Details

Available control options are

ifactor inflation ifactor for the bag-plot defining the loop of bag-plot or multiplying ifactor the KDE pilot bandwidth matrix. (see ?aplpack::compute.bagplot; ?ks::Hpi respectively; default: 2.58; 2 respectively).

variant string defining the method used ('KDE', 'pointwise' or 'bagplot') (default: 'bagplot')

unimodal logical specifying if the KDE estimate should be unimodal (default: FALSE, relavant only for variant='KDE')

addIndx vector of indeces corresponding to which samples one should overlay (Default: NULL)

K integer number of the first K components used for the representation. (default: length(fpcaObj$lambda))

References

P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999): The bagplot: a bivariate boxplot, The American Statistician, vol. 53, no. 4, 382-387

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
CreateFuncBoxPlot(res, list(addIndx=c(1:3)) )
```

`CreateModeOfVarPlot` *Functional Principal Component Analysis mode of variation plot*

Description

Create the k-th mode of variation plot around the mean. The red-line is the functional mean, the grey shaded areas show the range of variations around the mean: $\pm Q\sqrt{\lambda_k}\phi_k$ for the dark grey area Q = 1, and for the light grey are Q = 2. In the case of 'rainbowPlot' the blue edge corresponds to Q = -3, the green edge to Q = +3 and the red-line to Q = 0 (the mean).

Usage

```
CreateModeOfVarPlot(fpcaObj, k = 1, rainbowPlot = FALSE,
  colSpectrum = NULL, ...)
```

Arguments

`fpcaObj`	An FPCA class object returned by FPCA().
`k`	The k-th mode of variation to plot (default k = 1)
`rainbowPlot`	Indicator to create a rainbow-plot instead of a shaded plot (default: FALSE)
`colSpectrum`	Character vector to be use as input in the 'colorRampPalette' function defining the outliers colours (default: c("blue","red", "green"), relavant only for rainbowPlot=TRUE)
`...`	Additional arguments for the `plot` function.

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
CreateModeOfVarPlot(res)
```

`CreateOutliersPlot` *Functional Principal Component or Functional Singular Value Decomposition Scores Plot using 'bagplot' or 'KDE' methodology*

Description

This function will create, using the first components scores, a set of convex hulls of the scores based on 'bagplot' or 'KDE' methodology.

Usage

```
CreateOutliersPlot(fObj, optns = NULL, ...)
```

Arguments

`fObj`	A class object returned by FPCA() or FSVD().
`optns`	A list of options control parameters specified by `list(name=value)`. See 'Details'.
`...`	Additional arguments for the 'plot' function.

Details

Available control options are

ifactor inflation ifactor for the bag-plot defining the loop of bag-plot or multiplying ifactor the KDE pilot bandwidth matrix. (see ?aplpack::compute.bagplot; ?ks::Hpi respectively; default: 2.58; 2 respectively).

variant string defining the outlier method used ('KDE', 'NN' or 'bagplot') (default: 'KDE')

unimodal logical specifying if the KDE estimate should be unimodal (default: FALSE, relavant only for variant='KDE')

maxVar logical specifying if during slicing we should used the directions of maximum variance (default: FALSE for FPCA, TRUE for FSVD)

nSlices integer between 3 and 16, denoting the number of slices to be used (default: 4, relavant only for groupingType='slice')

showSlices logical specifying if during slicing we should show the outline of the slice (default: FALSE)

colSpectrum character vector to be use as input in the 'colorRampPalette' function defining the outliers colours (default: c("red", "yellow", 'blue'), relavant only for groupingType='slice')

groupingType string specifying if a slice grouping ('slice') or a standard percentile/bagplot grouping ('standard') should be returned (default: 'standard')

fIndeces a two-component vector with the index of the mode of variation to consider (default: c(1,2) for FPCA and c(1,1) for FSVD)

Value

An (temporarily) invisible copy of a list containing the labels associated with each of sample curves.

References

P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999): The bagplot: a bivariate boxplot, The American Statistician, vol. 53, no. 4, 382-387 R. J. Hyndman and H. L. Shang. (2010) Rainbow plots, bagplots, and boxplots for functional data, Journal of Computational and Graphical Statistics, 19(1), 29-45

Examples

```
## Not run:
set.seed(1)
n <- 420
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
CreateOutliersPlot(res)
```

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

CreatePathPlot *Create the fitted sample path plot based on the results from FPCA().*

Description

Create the fitted sample path plot based on the results from FPCA().

Usage

```
CreatePathPlot(fpcaObj, subset, K = NULL,
  inputData = fpcaObj[["inputData"]], showObs = !is.null(inputData),
  obsOnly = FALSE, showMean = FALSE, derOptns = list(p = 0), ...)
```

Arguments

`fpcaObj`	Returned object from FPCA().
`subset`	A vector of indices or a logical vector for subsetting the observations.
`K`	The number of components to reconstruct the fitted sample paths.
`inputData`	A list of length 2 containing the sparse/dense (unsupported yet) observations. `inputData` needs to contain two fields: `Lt` for a list of time points and `Ly` for a list of observations. Default to the ‘inputData‘ field within ‘fpcaObj‘.
`showObs`	Whether to plot the original observations for each subject.
`obsOnly`	Whether to show only the original curves.
`showMean`	Whether to plot the mean function as a bold solid curve.
`derOptns`	A list of options to control derivation parameters; see ‘fitted.FPCA’. (default = NULL)
`...`	other arguments passed into matplot for plotting options

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan',
            verbose=TRUE))
CreatePathPlot(res, subset=1:5)

# CreatePathPlot has a lot of usages:
## Not run:
CreatePathPlot(res)
CreatePathPlot(res, 1:20)
CreatePathPlot(res, 1:20, showObs=FALSE)
CreatePathPlot(res, 1:20, showMean=TRUE, showObs=FALSE)
CreatePathPlot(res, 1:20, obsOnly=TRUE)
```

```
CreatePathPlot(res, 1:20, obsOnly=TRUE, showObs=FALSE)
CreatePathPlot(inputData=sampWiener, subset=1:20, obsOnly=TRUE)
## End(Not run)
```

`CreateScreePlot` *Create the scree plot for the fitted eigenvalues*

Description

This function will open a new device if not instructed otherwise.

Usage

```
CreateScreePlot(fpcaObj, ...)
```

Arguments

`fpcaObj`	A object of class FPCA returned by the function FPCA().
`...`	Additional arguments for the 'plot' function.

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
CreateScreePlot(res)
```

`CreateStringingPlot` *Create plots for observed and stringed high dimensional data*

Description

The function produces the following three plots: 1) A plot of predictors (standardized if specified so during stringing) in original order for a subset of observations; 2) A plot of predictors in stringed order for the same subset of observations; 3) A plot of the stringing function, which is the stringed order vs. the original order.

Usage

```
CreateStringingPlot(stringingObj, subset, ...)
```

Arguments

`stringingObj`	A stringing object of class "Stringing", returned by the function Stringing.
`subset`	A vector of indices or a logical vector for subsetting the observations. If missing, first min(n,50) observations will be plotted where n is the sample size.
`...`	Other arguments passed into matplot for plotting options

Examples

```
set.seed(1)
n <- 50
wiener = Wiener(n = n)[,-1]
p = ncol(wiener)
rdmorder = sample(size = p, x=1:p, replace = FALSE)
stringingfit = Stringing(X = wiener[,rdmorder], disOptns = "correlation")
diff_norev = sum(abs(rdmorder[stringingfit$StringingOrder] - 1:p))
diff_rev = sum(abs(rdmorder[stringingfit$StringingOrder] - p:1))
if(diff_rev <= diff_norev){
  stringingfit$StringingOrder = rev(stringingfit$StringingOrder)
  stringingfit$Ly = lapply(stringingfit$Ly, rev)
}
CreateStringingPlot(stringingfit, 1:20)
```

`cumtrapzRcpp` *Cumulative Trapezoid Rule Numerical Integration*

Description

Cumulative Trapezoid Rule Numerical Integration using Rcpp

Usage

```
cumtrapzRcpp(X, Y)
```

Arguments

`X`	Sorted vector of X values
`Y`	Vector of Y values.

`DynCorr` *Dynamical Correlation*

Description

Calculate Dynamical Correlation for 2 paired dense regular functional data observed on the same grid.

Usage

```
DynCorr(x, y, t)
```

Arguments

`x`	a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.
`y`	a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.
`t`	a length m vector of time points where x,y are observed.

Value

A length m vector of individual dynamic correlations

References

Dubin J A, M\"uller H G. Dynamical correlation for multivariate longitudinal data[J]. Journal of the American Statistical Association, 2005, 100(471): 872-881. Liu S, Zhou Y, Palumbo R, et al. Dynamical correlation: A new method for quantifying synchrony with multivariate intensive longitudinal data[J]. Psychological methods, 2016, 21(3): 291.

Examples

```
set.seed(10)
n=200             # sample size
t=seq(0,1,length.out=100)       # length of data
mu_quad_x=8*t^2-4*t+5
mu_quad_y=8*t^2-12*t+6
fun=rbind(rep(1,length(t)),-t,t^2)
z1=matrix(0,n,3)
z1[,1]=rnorm(n,0,2)
z1[,2]=rnorm(n,0,16/3)
z1[,3]=rnorm(n,0,4)
x1_quad_error=y1_quad_error=matrix(0,nrow=n,ncol=length(t))
for (i in 1:n){
  x1_quad_error[i,]=mu_quad_x+z1[i,]%*%fun+rnorm(length(t),0,0.01)
  y1_quad_error[i,]=mu_quad_y+2*z1[i,]%*%fun +rnorm(length(t),0,0.01)
}
dyn1_quad=DynCorr(x1_quad_error,y1_quad_error,t)
```

Dyn_test *Bootstrap test of Dynamic correlation*

Description

Perform one sample (H0: Dynamic correlation = 0) or two sample (H0:Dynamic_correlation_1 = Dynamic_correlation_2) bootstrap test of Dynamical Correlation.

Usage

```
Dyn_test(x1, y1, t1, x2, y2, t2, B = 1000)
```

Arguments

x1	a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.
y1	a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.
t1	a vector of time points where x1,y1 are observed.
x2	(optional if missing will be one sample test) a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.

y2	(optional if missing will be one sample test) a n by m matrix where rows representing subjects and columns representing measurements, missings are allowed.
t2	(optional if missing will be one sample test) a vector of time points where x2,y2 are observed.
B	number of bootstrap samples.

Value

a list of the following

stats: test statistics.

pval: p-value of the test.

References

Dubin J A, M\"uller H G. Dynamical correlation for multivariate longitudinal data[J]. Journal of the American Statistical Association, 2005, 100(471): 872-881.

Liu S, Zhou Y, Palumbo R, et al. Dynamical correlation: A new method for quantifying synchrony with multivariate intensive longitudinal data[J]. Psychological methods, 2016, 21(3): 291.

Examples

```
n=200             # sample size
t=seq(0,1,length.out=100)       # length of data
mu_quad_x=8*t^2-4*t+5
mu_quad_y=8*t^2-12*t+6
fun=rbind(rep(1,length(t)),-t,t^2)
z1=matrix(0,n,3)
z1[,1]=rnorm(n,0,2)
z1[,2]=rnorm(n,0,16/3)
z1[,3]=rnorm(n,0,4)   # covariance matrix of random effects
x1_quad_error=y1_quad_error=matrix(0,nrow=n,ncol=length(t))
for (i in 1:n){
  x1_quad_error[i,]=mu_quad_x+z1[i,]%*%fun+rnorm(length(t),0,0.01)
  y1_quad_error[i,]=mu_quad_y+2*z1[i,]%*%fun +rnorm(length(t),0,0.01)
}
bt_DC=Dyn_test(x1_quad_error,y1_quad_error,t,B=1000)
```

FAM   *Functional Additive Models*

Description

Functional additive models with a single predictor process

Usage

```
FAM(Y, Lx, Lt, nEval = 51, newLx = NULL, newLt = NULL,
  bwMethod = 0, alpha = 0.7, supp = c(-2, 2), optns = NULL)
```

Arguments

Y	An n-dimensional vector whose elements consist of scalar responses.
Lx	A list of n vectors containing the observed values for each individual. See FPCA for detail.
Lt	A list of n vectors containing the observation time points for each individual. Each vector should be sorted in ascending order. See FPCA for detail.
nEval	The number of evaluation grid points for kernel smoothing (default is 51. If it is specified as 0, then estimated FPC scores in the training set are used for evaluation grid instead of equal grid).
newLx	A list of the observed values for test set. See predict.FPCA for detail.
newLt	A list of the observed time points for test set. See predict.FPCA for detail.
bwMethod	The method of bandwidth selection for kernel smoothing, a positive value for designating K-fold cross-validtaion and zero for GCV (default is 50)
alpha	The shrinkage factor (positive number) for bandwidth selection. See Han et al. (2016) (default is 0.7).
supp	The lower and upper limits of kernel smoothing domain for studentized FPC scores, which FPC scores are divided by the square roots of eigenvalues (default is [-2,2]).
optns	A list of options control parameters specified by list(name=value). See FPCA.

Details

FAM fits functional additive models for a scalar response and single predictor process proposed by Mueller and Yao (2007) that

$$E(Y|\mathbf{X}) = \sum_{k=1}^{K} g_k(\xi_k),$$

where ξ_k stand for the k-th FPC score of the the predictor process.

Value

A list containing the following fields:

mu	Mean estimator of EY
fam	A N by K matrix whose column vectors consist of the component function estimators at the given estimation points.
xi	An N by K matrix whose column vectors consist of N vectors of estimation points for each component function.
bw	A K-dimensional bandwidth vector.
lambda	A K-dimensional vector containing eigenvalues.
phi	An *nWorkGrid* by K matrix containing eigenfunctions, supported by WorkGrid. See FPCA.
workGrid	An *nWorkGrid* by *K_j* working grid, the internal regular grid on which the eigen analysis is carried on. See FPCA.

References

Mueller, H.-G. and Yao, F. (2005), "Functional additive models", JASA, Vol.103, No.484, p.1534-1544.

Examples

```
set.seed(1000)

library(MASS)

f1 <- function(t) 0.5*t
f2 <- function(t) 2*cos(2*pi*t/4)
f3 <- function(t) 1.5*sin(2*pi*t/4)
f4 <- function(t) 2*atan(2*pi*t/4)

n<-250
N<-500

sig <- diag(c(4.0,2.0,1.5,1.2))

scoreX <- mvrnorm(n,mu=rep(0,4),Sigma=sig)
scoreXTest <- mvrnorm(N,mu=rep(0,4),Sigma=sig)

Y <- f1(scoreX[,1]) + f2(scoreX[,2]) + f3(scoreX[,3]) + f4(scoreX[,4]) + rnorm(n,0,0.1)
YTest <- f1(scoreXTest[,1]) + f2(scoreXTest[,2]) +
  f3(scoreXTest[,3]) + f4(scoreXTest[,4]) + rnorm(N,0,0.1)

phi1 <- function(t) sqrt(2)*sin(2*pi*t)
phi2 <- function(t) sqrt(2)*sin(4*pi*t)
phi3 <- function(t) sqrt(2)*cos(2*pi*t)
phi4 <- function(t) sqrt(2)*cos(4*pi*t)

grid <- seq(0,1,length.out=21)
Lt <- Lx <- list()
for (i in 1:n) {
  Lt[[i]] <- grid
  Lx[[i]] <- scoreX[i,1]*phi1(grid) + scoreX[i,2]*phi2(grid) +
    scoreX[i,3]*phi3(grid) + scoreX[i,4]*phi4(grid) + rnorm(1,0,0.01)
}

LtTest <- LxTest <- list()
for (i in 1:N) {
  LtTest[[i]] <- grid
  LxTest[[i]] <- scoreXTest[i,1]*phi1(grid) + scoreXTest[i,2]*phi2(grid) +
    scoreXTest[i,3]*phi3(grid) + scoreXTest[i,4]*phi4(grid) + rnorm(1,0,0.01)
}


# estimation
fit <- FAM(Y=Y,Lx=Lx,Lt=Lt)

xi <- fit$xi

par(mfrow=c(2,2))
j <- 1
g1 <- f1(sort(xi[,j]))
tmpSgn <- sign(sum(g1*fit$fam[,j]))
plot(sort(xi[,j]),g1,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi1')
points(sort(xi[,j]),tmpSgn*fit$fam[order(xi[,j]),j],type='l')

j <- 2
```

```
g2 <- f2(sort(xi[,j]))
tmpSgn <- sign(sum(g2*fit$fam[,j]))
plot(sort(xi[,j]),g2,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi2')
points(sort(xi[,j]),tmpSgn*fit$fam[order(xi[,j]),j],type='l')

j <- 3
g3 <- f3(sort(xi[,j]))
tmpSgn <- sign(sum(g3*fit$fam[,j]))
plot(sort(xi[,j]),g3,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi3')
points(sort(xi[,j]),tmpSgn*fit$fam[order(xi[,j]),j],type='l')

j <- 4
g4 <- f4(sort(xi[,j]))
tmpSgn <- sign(sum(g4*fit$fam[,j]))
plot(sort(xi[,j]),g4,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi4')
points(sort(xi[,j]),tmpSgn*fit$fam[order(xi[,j]),j],type='l')


# fitting
fit <- FAM(Y=Y,Lx=Lx,Lt=Lt,nEval=0)
yHat <- fit$mu+apply(fit$fam,1,'sum')
par(mfrow=c(1,1))
plot(yHat,Y)
abline(coef=c(0,1),col=2)


# R^2
R2 <- 1-sum((Y-yHat)^2)/sum((Y-mean(Y))^2)
R2


# prediction
fit <- FAM(Y=Y,Lx=Lx,Lt=Lt,newLx=LxTest,newLt=LtTest)
yHat <- fit$mu+apply(fit$fam,1,'sum')
par(mfrow=c(1,1))
plot(yHat,YTest,xlim=c(-10,10))
abline(coef=c(0,1),col=2)
```

`FCCor` *Calculate functional correlation between two simultaneously observed processes.*

Description

Calculate functional correlation between two simultaneously observed processes.

Usage

```
FCCor(x, y, Lt, bw = stop("bw missing"), kern = "epan",
  Tout = sort(unique(unlist(Lt))))
```

Arguments

x	A list of function values corresponding to the first process.

y	A list of function values corresponding to the second process.
Lt	A list of time points for both x and y.
bw	A numeric vector for bandwidth of length either 5 or 1, specifying the bandwidths for E(X), E(Y), var(X), var(Y), and cov(X, Y). If bw is a scalar then all five bandwidths are chosen to be the same.
kern	Smoothing kernel for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" (default: "gauss")
Tout	Output time points. Default to the sorted unique time points.

Details

FCCor calculate only the concurrent correlation corr(X(t), Y(t)) (note that the time points t are the same). It assumes no measurement error in the observed values.

Value

A list with the following components:

corr	A vector of the correlation corr(X(t), Y(t)) evaluated at Tout.
Tout	Same as the input Tout.
bw	The bandwidths used for E(X), E(Y), var(X), var(Y), and cov(X, Y).

Examples

```
set.seed(1)
n <- 200
nGridIn <- 50
sparsity <- 1:5 # must have length > 1
bw <- 0.2
kern <- 'epan'
T <- matrix(seq(0.5, 1, length.out=nGridIn))

## Corr(X(t), Y(t)) = 1/2
A <- Wiener(n, T)
B <- Wiener(n, T)
C <- Wiener(n, T) + matrix((1:nGridIn) , n, nGridIn, byrow=TRUE)
X <- A + B
Y <- A + C
indEach <- lapply(1:n, function(x) sort(sample(nGridIn, sample(sparsity, 1))))
tAll <- lapply(1:n, function(i) T[indEach[[i]]])
Xsp <- lapply(1:n, function(i) X[i, indEach[[i]]])
Ysp <- lapply(1:n, function(i) Y[i, indEach[[i]]])

plot(T, FCCor(Xsp, Ysp, tAll, bw)[['corr']], ylim=c(-1, 1))
abline(h=0.5)
```

`FClust` *Functional clustering and identifying substructures of longitudinal data*

Description

By default the function will cluster the data using the functional principal component (FPC) scores from the data's FPC analysis using EMCluster (Chen and Maitra, 2015) or directly clustering the functional data using kCFC (Chiou and Li, 2007).

Usage

```
FClust(Ly, Lt, k = 3, cmethod = "EMCluster", optnsFPCA = NULL,
  optnsCS = NULL)
```

Arguments

`Ly`	A list of *n* vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
`Lt`	A list of *n* vectors containing the observation time points for each individual corresponding to y.
`k`	A scalar defining the number of clusters to define; default 3.
`cmethod`	A string specifying the clusterig method to use ('EMCluster' or 'kCFC'); default: 'EMCluster'.
`optnsFPCA`	A list of options control parameters specified by `list(name=value)` to be used for by FPCA on the sample y; by default: "list(methodMuCovEst ='smooth', FVEthreshold= 0.90, methodBwCov = 'GCV', methodBwMu = 'GCV')". See 'Details in ?FPCA'.
`optnsCS`	A list of options control parameters specified by `list(name=value)` to be used for cluster-specific FPCA from kCFC; by default: "list(methodMuCovEst ='smooth', FVEthreshold= 0.70, methodBwCov = 'GCV', methodBwMu = 'GCV')". See 'Details in ?FPCA' and '?kCFC'.

Details

Within EMCluster we examine the model initiated "EMCluster::em.EM" and return the optimal model based on 'EMCluster::emcluster'. See ?EMCluster::emcluster for details.

Value

A list containing the following fields:

`cluster`	A vector of levels 1:k, indicating the cluster to which each curve is allocated.
`fpca`	An FPCA object derived from the sample used by Rmixmod, otherwise NULL.
`clusterObj`	Either a EMCluster object or kCFC object.

References

Wei-Chen Chen and Ranjan Maitra, "EMCluster: EM Algorithm for Model-Based Clusttering of Finite Mixture Gaussian Distribution". (2015)

Julien Jacques and Cristian Preda, "Funclust: A curves clustering method using functional random variables density approximation". Neurocomputing 112 (2013): 164-171

Jeng-Min Chiou and Pai-Ling Li, "Functional clustering and identifying substructures of longitudinal data". Journal of the Royal Statistical Society B 69 (2007): 679-699

Examples

```
## Not run:
data(medfly25)
Flies <- MakeFPCAInputs(medfly25$ID, medfly25$Days, medfly25$nEggs)
newClust <- FClust(Flies$Ly, Flies$Lt, k = 2, optnsFPCA =
                    list(methodMuCovEst = 'smooth', userBwCov = 2, FVEthreshold = 0.90))

# We denote as 'veryLowCount' the group of flies that lay less
# than twenty-five eggs during the 25-day period examined.

veryLowCount = ifelse( sapply( unique(medfly25$ID), function(u)
                   sum( medfly25$nEggs[medfly25$ID == u] )) < 25, 0, 1)
N <- length(unique(medfly25$ID))
(correctRate <- sum( (1 + veryLowCount) ==  newClust$cluster) / N) # 99.6%

## End(Not run)
```

`FCReg` *Functional Concurrent Regression by 2D smoothing method.*

Description

Functional concurrent regression with dense or sparse functional data for scalar or functional dependent variable.

Usage

```
FCReg(vars, userBwMu, userBwCov, outGrid, kern = "gauss",
  measurementError = TRUE, diag1D = "none", useGAM = FALSE,
  returnCov = TRUE)
```

Arguments

`vars`	A list of input functional/scalar covariates. Each field corresponds to a functional (a list) or scalar (a vector) covariate. The last entry is assumed to be the response if no entry is names 'Y'. If a field corresponds to a functional covariate, it should have two fields: 'Lt', a list of time points, and 'Ly', a list of function values.
`userBwMu`	A scalar with bandwidth used for smoothing the mean
`userBwCov`	A scalar with bandwidth used for smoothing the auto- and cross-covariances
`outGrid`	A vector with the output time points

`kern`	Smoothing kernel choice, common for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" (default: "gauss")
`measurementError`	Indicator measurement errors on the functional observations should be assumed. If TRUE the diagonal raw covariance will be removed when smoothing. (default: TRUE)
`diag1D`	A string specifying whether to use 1D smoothing for the diagonal line of the covariance. 'none': don't use 1D smoothing; 'cross': use 1D only for cross-covariances; 'all': use 1D for both auto- and cross-covariances. (default : 'none')
`useGAM`	Indicator to use gam smoothing instead of local-linear smoothing (semi-parametric option) (default: FALSE)
`returnCov`	Indicator to return the covariance surfaces, which is a four dimensional array. The first two dimensions correspond to outGrid and the last two correspond to the covariates and the response, i.e. (i, j, k, l) entry being Cov(X_k(t_i), X_l(t_j)) (default: FALSE)
`...`	Additional arguments

Details

If measurement error is assumed, the diagonal elements of the raw covariance will be removed. This could result in highly unstable estimate if the design is very sparse, or strong seasonality presents.

References

Yao, F., Mueller, H.G., Wang, J.L. "Functional Linear Regression Analysis for Longitudinal Data." Annals of Statistics 33, (2005): 2873-2903.(Dense data)

Senturk, D., Nguyen, D.V. "Varying Coefficient Models for Sparse Noise-contaminated Longitudinal Data", Statistica Sinica 21(4), (2011): 1831-1856. (Sparse data)

Examples

```
# Y(t) = \beta_0(t) + \beta_1(t) X_1(t) + \beta_2(t) Z_2 + \epsilon

# Settings
set.seed(1)
n <- 75
nGridIn <- 150
sparsity <- 5:10 # Sparse data sparsity
T <- round(seq(0, 1, length.out=nGridIn), 4) # Functional data support
bw <- 0.1
outGrid <- round(seq(min(T), 1, by=0.05), 2)

# Simulate functional data
mu <- T * 2 # mean function for X_1
sigma <- 1

beta_0 <- 0
beta_1 <- 1
beta_2 <- 1

Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(mu, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
```

```
Y <- matrix(NA, n, nGridIn)
for (i in seq_len(n)) {
  Y[i, ] <- beta_0 + beta_1 * X_1[i, ] + beta_2 * Z[i, 2] + epsilon[i]
}

# Sparsify functional data
set.seed(1)
X_1sp <- Sparsify(X_1, T, sparsity)
set.seed(1)
Ysp <- Sparsify(Y, T, sparsity)
vars <- list(X_1=X_1sp, Z_2=Z[, 2], Y=Ysp)
withError2D <- FCReg(vars, bw, bw, outGrid)
```

`fdapace` *PACE: Principal Analysis by Conditional Expectation*

Description

PACE package for Functional Data Analysis and Empirical Dynamics.

Details

PACE is a versatile package that provides implementation of various methods of Functional Data Analysis (FDA) and Empirical Dynamics. The core of this package is Functional Principal Component Analysis (FPCA), a key technique for functional data analysis, for sparsely or densely sampled random trajectories and time courses, via the Principal Analysis by Conditional Estimation (PACE) algorithm. PACE is useful for the analysis of data that have been generated by a sample of underlying (but usually not fully observed) random trajectories. It does not rely on pre-smoothing of trajectories, which is problematic if functional data are sparsely sampled. PACE provides options for functional regression and correlation, for Longitudinal Data Analysis, the analysis of stochastic processes from samples of realized trajectories, and for the analysis of underlying dynamics.

Author(s)

Xiongtao Dai `<dai@ucdavis.edu>`, Pantelis Z. Hadjipantelis `<pantelis@ucdavis.edu>`, Hao Ji `<haoji@ucdavis.edu>` Hans-Georg Mueller `<hgmueller@ucdavis.edu>` Jane-Ling Wang `<janelwang@ucdavis.edu>`

Maintainer: Pantelis Z. Hadjipantelis `<pantelis@ucdavis.edu>`

`fitted.FPCA` *Fitted functional sample from FPCA object*

Description

Combine the zero-meaned fitted values and the interpolated mean to get the fitted values for the trajectories or the derivatives of these trajectories. Estimates are given on the work-grid, not on the observation grid. Use ConvertSupport to map the estimates to your desired domain. `100*(1-alpha)`-percentage coverage intervals, or bands, for trajectory estimates (not derivatives) are provided. See details in example.

Usage

```
## S3 method for class 'FPCA'
fitted(object, K = NULL, derOptns = list(p = 0),
  ciOptns = list(alpha = NULL, cvgMethod = NULL), ...)
```

Arguments

`object`	A object of class FPCA returned by the function FPCA().
`K`	The integer number of the first K components used for the representation. (default: length(fpcaObj$lambda))
`derOptns`	A list of options to control the derivation parameters specified by `list(name=value)`. See 'Details'. (default = NULL)
`ciOptns`	A list of options to control the confidence interval/band specified by `list(name=value)`. See 'Details'. (default = NULL)
`...`	Additional arguments

Details

Available derivation control options are

p The order of the derivatives returned (default: 0, max: 2)

method The method used to produce the sample of derivatives ('FPC' (default) or 'QUO'). See Liu and Mueller (2009) for more details

bw Bandwidth for smoothing the derivatives (default: p * 0.10 * S)

kernelType Smoothing kernel choice; same available types are FPCA(). default('epan')

Available confidence interval/band control options are

alpha Significant level for confidence interval/band for trajectory coverage. default=0.05 (currently only work when p=0)

cvgMethod Option for trajectory coverage method beween 'interval' and 'band'. default='band'

Value

If `alpha` is `NULL`, `p>1` or functional observations are dense, an `n` by `length(workGrid)` matrix, each row of which contains a sample. Otherwise, it returns a list which consists of the following items:

`workGrid`	An evaluation grid for fitted values.
`fitted`	An n by length(workGrid) matrix, each row of which contains a sample.
`cvgUpper`	An n by length(workGrid) matrix, each row of which contains the upper `alpha`-coverage limit
`cvgLower`	An n by length(workGrid) matrix, each row of which contains the lower `alpha`-coverage limit

References

Yao, F., Mueller, H.-G. and Wang, J.-L. "Functional data analysis for sparse longitudinal data", Journal of the American Statistical Association, vol.100, No. 470 (2005): 577-590.

Liu, Bitao, and Hans-Georg Mueller. "Estimating derivatives for samples of sparsely observed functions, with application to online auction dynamics." Journal of the American Statistical Association 104, no. 486 (2009): 704-717. (Sparse data FPCA)

Examples

```
set.seed(1)
n <- 100
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 5:10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
fittedY <- fitted(res, ciOptns = list(alpha=0.05))

workGrid <- res$workGrid
cvgUpper <- fittedY$cvgUpper
cvgLower <- fittedY$cvgLower

par(mfrow=c(2,3))
ind <- sample(1:n,6)
for (i in 1:6) {
 j <- ind[i]
 plot(workGrid,cvgUpper[j,],type='l',ylim=c(min(cvgLower[j,]),max(cvgUpper[j,])),col=4,lty=2,
   xlab='t', ylab='X(t)', main=paste(j,'-th subject',sep=''))
 points(workGrid,cvgLower[j,],type='l',col=4,lty=2)
 points(res$inputData$Lt[[j]],res$inputData$Ly[[j]])
}
```

`fitted.FPCAder` *Fitted functional sample from FPCAder object*

Description

Combine the zero-meaned fitted values and the mean derivative to get the fitted values for the derivatives trajectories Estimates are given on the work-grid, not on the observation grid. Use ConvertSupport to map the estimates to your desired domain.

Usage

```
## S3 method for class 'FPCAder'
fitted(object, K = NULL, ...)
```

Arguments

`object`	A object of class FPCA returned by the function FPCA().
`K`	The integer number of the first K components used for the representation. (default: length(derObj$lambda))
`...`	Additional arguments

Value

An `n` by `length(workGrid)` matrix, each row of which contains a sample.

References

Liu, Bitao, and Hans-Georg Mueller. "Estimating derivatives for samples of sparsely observed functions, with application to online auction dynamics." Journal of the American Statistical Association 104, no. 486 (2009): 704-717. (Sparse data FPCA)

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
```

FOptDes — *Optimal Designs for Functional and Longitudinal Data for Trajectory Recovery or Scalar Response Prediction*

Description

Optimal Designs for Functional and Longitudinal Data for Trajectory Recovery or Scalar Response Prediction

Usage

```
FOptDes(Ly, Lt, Resp, p = 3, optns = list(),
  isRegression = !missing(Resp), isSequential = FALSE,
  RidgeCand = NULL)
```

Arguments

`Ly`	A list of n vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
`Lt`	A list of n vectors containing the observation time points for each individual corresponding to y. Each vector should be sorted in ascending order.
`Resp`	A vector of response values, keep void for trajectory recovery, only necessary for scalar response prediction task.
`p`	A fixed positive integer indicating the number of optimal design points requested, with default: 3.
`optns`	A list of options control parameters specified by `list(name=value)` for FPCA, with default: list().
`isRegression`	A logical argument, indicating the purpose of the optimal designs: TRUE for scalar response prediction, FALSE for trajectory recovery, with default value !missing(Resp).
`isSequential`	A logical argument, indicating whether to use the sequential optimization procedure for faster computation, recommended for relatively large p (default: FALSE).
`RidgeCand`	A vector of positive numbers as ridge penalty candidates for regularization. The final value is selected via cross validation. If only 1 ridge parameter is specified, CV procedure is skipped.

Details

To select a proper RidgeCand, check with the returned optimal ridge parameter. If the selected parameter is the maximum/minimum values in the candidates, it is possible that the selected one is too small/big.

Value

A list containing the following fields:

OptDes	The vector of optimal design points of the regular time grid of the observed data.
R2	Coefficient of determination. (Check the paper for details.)
R2adj	Adjusted coefficient of determination.
OptRidge	The selected ridge parameter.

References

Ji, H., Mueller, H.G. (2016) "Optimal Designs for Longitudinal and Functional Data" Journal of the Royal Statistical Society: Series B (Statistical Methodology)

Examples

```
set.seed(1)
n <- 50
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- MakeFPCAInputs(IDs = rep(1:n, each=length(pts)),
                             tVec = rep(pts, times = n),
                             yVec = t(sampWiener))
res <- FOptDes(Ly=sampWiener$Ly, Lt=sampWiener$Lt, p=2,
               isSequential=FALSE, RidgeCand = seq(0.02,0.2,0.02))
```

FPCA *Functional Principal Component Analysis*

Description

FPCA for dense or sparse functional data.

Usage

```
FPCA(Ly, Lt, optns = list())
```

Arguments

Ly	A list of *n* vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
Lt	A list of *n* vectors containing the observation time points for each individual corresponding to y. Each vector should be sorted in ascending order.
optns	A list of options control parameters specified by `list(name=value)`. See 'Details'.

Details

If the input is sparse data, make sure you check the design plot is dense and the 2D domain is well covered, using `plot` or `CreateDesignPlot`. Some study design such as snippet data (each subject is observed only on a sub-interval of the period of study) will have an ill-covered design plot, for which the covariance estimate will be unreliable.

Available control options are

userBwCov The bandwidth value for the smoothed covariance function; positive numeric - default: determine automatically based on 'methodBwCov'

methodBwCov The bandwidth choice method for the smoothed covariance function; 'GMeanAndGCV' (the geometric mean of the GCV bandwidth and the minimum bandwidth),'CV','GCV' - default: 10% of the support

userBwMu The bandwidth value for the smoothed mean function (using 'CV' or 'GCV'); positive numeric - default: determine automatically based on 'methodBwMu'

methodBwMu The bandwidth choice method for the mean function; 'GMeanAndGCV' (the geometric mean of the GCV bandwidth and the minimum bandwidth),'CV','GCV' - default: 5% of the support

dataType The type of design we have (usually distinguishing between sparse or dense functional data); 'Sparse', 'Dense', 'DenseWithMV', 'p»n' - default: determine automatically based on 'IsRegular'

diagnosticsPlot Deprecated. Same as the option 'plot'

plot Plot FPCA results (design plot, mean, scree plot and first K (<=3) eigenfunctions); logical - default: FALSE

error Assume measurement error in the dataset; logical - default: TRUE

fitEigenValues Whether also to obtain a regression fit of the eigenvalues - default: FALSE

FVEthreshold Fraction-of-Variance-Explained threshold used during the SVD of the fitted covar. function; numeric (0,1] - default: 0.9999

kernel Smoothing kernel choice, common for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" - default: "gauss"; dense data are assumed noise-less so no smoothing is performed.

kFoldMuCov The number of folds to be used for mean and covariance smoothing. Default: 10

lean If TRUE the 'inputData' field in the output list is empty. Default: FALSE

maxK The maximum number of principal components to consider - default: min(20, N-1), N:# of curves

methodXi The method to estimate the PC scores; 'CE' (Condit. Expectation), 'IN' (Numerical Integration) - default: 'CE' for sparse data and dense data with missing values, 'IN' for dense data.

methodMuCovEst The method to estimate the mean and covariance in the case of dense functional data; 'cross-sectional', 'smooth' - default: 'cross-sectional'

nRegGrid The number of support points in each direction of covariance surface; numeric - default: 51

numBins The number of bins to bin the data into; positive integer > 10, default: NULL

methodSelectK The method of choosing the number of principal components K; 'FVE','AIC','BIC', or a positive integer as specified number of components: default 'FVE')

shrink Whether to use shrinkage method to estimate the scores in the dense case (see Yao et al 2003) - default FALSE

outPercent A 2-element vector in [0,1] indicating the outPercent data in the boundary - default (0,1)

rho The truncation threshold for the iterative residual. 'cv': choose rho by leave-one-observation out cross-validation; 'no': no regularization - default "cv" if error == TRUE, and "no" if error == FALSE.

rotationCut The 2-element vector in [0,1] indicating the percent of data truncated during sigma^2 estimation; default (0.25, 0.75))

useBinnedData Should the data be binned? 'FORCE' (Enforce the # of bins), 'AUTO' (Select the # of bins automatically), 'OFF' (Do not bin) - default: 'AUTO'

useBinnedCov Whether to use the binned raw covariance for smoothing; logical - default:TRUE

userCov The user-defined smoothed covariance function; list of two elements: numerical vector 't' and matrix 'cov', 't' must cover the support defined by 'Ly' - default: NULL

userMu The user-defined smoothed mean function; list of two numerical vector 't' and 'mu' of equal size, 't' must cover the support defined 'Ly' - default: NULL

userSigma2 The user-defined measurement error variance. A positive scalar. If specified then no regularization is used (rho is set to 'no', unless specified otherwise). Default to 'NULL'

userRho The user-defined measurement truncation threshold used for the calculation of functional principal components scores. A positive scalar. Default to 'NULL'

useBW1SE Pick the largest bandwidth such that CV-error is within one Standard Error from the minimum CV-error, relevant only if methodBwMu ='CV' and/or methodBwCov ='CV'; logical - default: FALSE

verbose Display diagnostic messages; logical - default: FALSE

Value

A list containing the following fields:

`sigma2`	Variance for measure error.
`lambda`	A vector of length K containing eigenvalues.
`phi`	An nWorkGrid by K matrix containing eigenfunctions, supported on workGrid.
`xiEst`	A n by K matrix containing the FPC estimates.
`xiVar`	A list of length n, each entry containing the variance estimates for the FPC estimates.
`obsGrid`	The (sorted) grid points where all observation points are pooled.
`mu`	A vector of length nWorkGrid containing the mean function estimate.
`workGrid`	A vector of length nWorkGrid. The internal regular grid on which the eigen analysis is carried on.
`smoothedCov`	A nWorkGrid by nWorkGrid matrix of the smoothed covariance surface.
`fittedCov`	A nWorkGrid by nWorkGrid matrix of the fitted covariance surface, which is guaranteed to be non-negative definite.
`optns`	A list of actually used options.
`timings`	A vector with execution times for the basic parts of the FPCA call.
`bwMu`	The selected (or user specified) bandwidth for smoothing the mean function.
`bwCov`	The selected (or user specified) bandwidth for smoothing the covariance function.
`rho`	A regularizing scalar for the measurement error variance estimate.

cumFVE	A vector with the percentages of the total variance explained by each FPC. Increase to almost 1.
FVE	A percentage indicating the total variance explained by chosen FPCs with corresponding 'FVEthreshold'.
criterionValue	A scalar specifying the criterion value obtained by the selected number of components with specific methodSelectK: FVE,AIC,BIC values or NULL for fixedK.
inputData	A list containting the original 'Ly' and 'Lt' lists used as inputs to FPCA. NULL if 'lean' was specified to be TRUE.

References

Yao, F., Mueller, H.G., Clifford, A.J., Dueker, S.R., Follett, J., Lin, Y., Buchholz, B., Vogel, J.S. (2003). "Shrinkage estimation for functional principal component scores, with application to the population kinetics of plasma folate." Biometrics 59, 676-685. (Shrinkage estimates for dense data)

Yao, Fang, Hans-Georg Mueller, and Jane-Ling Wang. "Functional data analysis for sparse longitudinal data." Journal of the American Statistical Association 100, no. 470 (2005): 577-590. (Sparse data FPCA)

Liu, Bitao, and Hans-Georg Mueller. "Estimating derivatives for samples of sparsely observed functions, with application to online auction dynamics." Journal of the American Statistical Association 104, no. 486 (2009): 704-717. (Sparse data FPCA)

Castro, P. E., W. H. Lawton, and E. A. Sylvestre. "Principal modes of variation for processes with continuous sample curves." Technometrics 28, no. 4 (1986): 329-337. (Dense data FPCA)

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt,
            list(dataType='Sparse', error=FALSE, kernel='epan', verbose=TRUE))
plot(res) # The design plot covers [0, 1] * [0, 1] well.
CreateCovPlot(res, 'Fitted')
```

FPCAder *Take derivative of an FPCA object*

Description

Take derivative of an FPCA object

Usage

```
FPCAder(fpcaObj, derOptns = list(p = 1))
```

Arguments

fpcaObj	A object of class FPCA returned by the function FPCA().
derOptns	A list of options to control the derivation parameters specified by `list(name=value)`. See 'Details'. (default = NULL)

Details

Available derivation control options are

method The method used for obtaining the derivatives (default: 'FPC'). 'DPC': derivatives principal component, with G^(1,1) estimated by first kernel local smoothing G^(1,0), and then apply a 1D smoother on the second direction; 'FPC': functional principal component, based on smoothing the eigenfunctions; 'FPC1': functional principal component, based on smoothing G^(1,0). May produce better estimate than 'FPC' but is slower.

p The order of the derivatives returned (default: 1, max: 2).

bw Bandwidth for the 1D and the 2D smoothers (default: p * 0.1 * S).

kernelType Smoothing kernel choice; same available types are FPCA(). default('epan')

References

Dai, Xiongtao, Hans-Georg Mueller, and Wenwen Tao. "Derivative Principal Component Analysis for Representing the Time Dynamics of Longitudinal and Functional Data." Submitted (DPC) Liu, Bitao, and Hans-Georg Mueller. "Estimating derivatives for samples of sparsely observed functions, with application to online auction dynamics." Journal of the American Statistical Association 104, no. 486 (2009): 704-717. (FPC)

Examples

```
bw <- 0.2
kern <- 'epan'
set.seed(1)
n <- 100
M <- 40
pts <- seq(0, 1, length.out=M)
lambdaTrue <- c(1, 0.8, 0.1)^2
sigma2 <- 0.1

samp2 <- MakeGPFunctionalData(n, M, pts, K=length(lambdaTrue),
                          lambda=lambdaTrue, sigma=sqrt(sigma2), basisType='legendre01')
samp2 <- c(samp2, MakeFPCAInputs(tVec=pts, yVec=samp2$Yn))
fpcaObj <- FPCA(samp2$Ly, samp2$Lt, list(methodMuCovEst='smooth',
                userBwCov=bw, userBwMu=bw, kernel=kern, error=TRUE))
CreatePathPlot(fpcaObj, showObs=FALSE)

FPCoptn <- list(bw=bw, kernelType=kern, method='FPC')
DPCoptn <- list(bw=bw, kernelType=kern, method='DPC')
FPC <- FPCAder(fpcaObj, FPCoptn)
DPC <- FPCAder(fpcaObj, DPCoptn)

CreatePathPlot(FPC, ylim=c(-5, 10))
CreatePathPlot(DPC, ylim=c(-5, 10))

# Get the true derivatives
phi <-  CreateBasis(K=3, type='legendre01', pts=pts)
basisDerMat <- apply(phi, 2, function(x)
                     ConvertSupport(seq(0, 1, length.out=M - 1), pts, diff(x) * (M - 1)))
trueDer <- matrix(1, n, M, byrow=TRUE) + tcrossprod(samp2$xi, basisDerMat)
matplot(t(trueDer), type='l', ylim=c(-5, 10))

# DPC is slightly better in terms of RMSE
```

```
mean((fitted(FPC) - trueDer)^2)
mean((fitted(DPC) - trueDer)^2)
```

`FPCquantile` *Conditional Quantile estimation with functional covariates*

Description

Main function to implement conditional Quantile estimation with functional covariates and scalar response. The method includes 3 steps: 1) FPCA using the PACE method for X(t_x) 2) Computation of the conditional distribution function through a functional generalized linear model. 3) Prediction of quantiles for given predictor values

Usage

```
FPCquantile(Lx, Lt_x, y, outQ = c(0.1, 0.25, 0.5, 0.75, 0.9),
  optns_x = NULL, isNewsub = NULL)
```

Arguments

`Lx`	A length n list of predictor function where x[[i]] is the row vector of measurements for ith subject, i=1,...,n
`Lt_x`	A length n list where the observations of x are taken, t_x[[i]] is a row vector of time points where x[[i]] are observed, i=1,...,n
`y`	A 1*n vector for scalar response y. y[i] is the response value for the ith subject, i = 1,...,n.
`outQ`	A vector of desired quantile levels with default value outQ = c(0.1, 0.25, 0.5, 0.75, 0.9).
`optns_x`	A list of options for predictor x with control parameters specified by list(name=value) with default NA. See function FPCA for details.
`isNewSub`	A 1*n vector of 0s or 1s, where n is the total count of subjects. 0 denotes the corresponding subject is only used for training and 1 denotes the corresponding subject is only used for prediction. (default: 0's)

Value

A list of the following

`pred_quantile`: a `matrix` of `n*length(outQ)` where the the `first` `nn` (`number` of `0s` in `isNewSub`) `rows` `contain`

`pred_CDF`: a `matrix` of `n*100`. The `ith` `row` contains the `fitted` or `predicted` `conditional` `distribution` `func`

`b`: a `matrix` of `50*(K+1)` `contains` `the` `coefficient` `functions`, `defined` `as` $F(y|X) = g(\sum_(k = 0)^K b_k(y)\xi_k)$, s

References

Chen, K., M\"uller, H.G. (2011). Conditional quantile analysis when covariates are functions, with application to growth data. J. Royal Statistical Society B.

Examples

```
set.seed(10)

n = 200
npred = 50
m = 50
xi <- Wiener(n, 0:m/m)

x=list()
t_x=list()
y=numeric(n)
for(i in 1:n){
 t_x = c(t_x,list(0:m/m))
 x = c(x,list(xi[i,]))
 y[i] = 5*rnorm(1)+2*sum(xi[i,])
}

outQ = c(0.1,0.25,0.5,0.75,0.9,0.95)
isNewsub = c(rep(0,150),rep(1,50))
qtreg = FPCquantile(x, t_x, y, outQ,optns_x = NULL,isNewsub)
```

FSVD *Functional Singular Value Decomposition*

Description

FSVD for a pair of dense or sparse functional data.

Usage

```
FSVD(Ly1, Lt1, Ly2, Lt2, FPCAoptns1 = NULL, FPCAoptns2 = NULL,
  SVDoptns = list())
```

Arguments

`Ly1`	A list of *n* vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
`Lt1`	A list of *n* vectors containing the observation time points for each individual corresponding to y. Each vector should be sorted in ascending order.
`Ly2`	A list of *n* vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
`Lt2`	A list of *n* vectors containing the observation time points for each individual corresponding to y. Each vector should be sorted in ascending order.
`FPCAoptns1`	A list of options control parameters specified by `list(name=value)` for the FPC analysis of sample 1. See '?FPCA'.
`FPCAoptns2`	A list of options control parameters specified by `list(name=value)` for the FPC analysis of sample 2. See '?FPCA'.
`SVDoptns`	A list of options control parameters specified by `list(name=value)` for the FSVD analysis of samples 1 & 2. See 'Details'.

Details

Available control options for SVDoptns are:

bw1 The bandwidth value for the smoothed cross-covariance function across the direction of sample 1; positive numeric - default: determine automatically based on 'methodBwCov'

bw2 The bandwidth value for the smoothed cross-covariance function across the direction of sample 2; positive numeric - default: determine automatically based on 'methodBwCov'

methodBwCov The bandwidth choice method for the smoothed covariance function; 'GMeanAndGCV' (the geometric mean of the GCV bandwidth and the minimum bandwidth),'CV','GCV' - default: 10% of the support

userMu1 The user defined mean of sample 1 used to centre it prior to the cross-covariance estimation. - default: determine automatically based by the FPCA of sample 1

userMu2 The user defined mean of sample 2 used to centre it prior to the cross-covariance estimation. - default: determine automatically based by the FPCA of sample 2

maxK The maximum number of singular components to consider; default: min(20, N-1), N:# of curves.

kernel Smoothing kernel choice, common for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" - default: "gauss"; dense data are assumed noise-less so no smoothing is performed.

rmDiag Logical describing if the routine should remove diagonal raw cov for cross cov estimation (default: FALSE)

noScores Logical describing if the routine should return functional singular scores or not (default: TRUE)

regulRS String describing if the regularisation of the compositie cross-covariance matrix should be done using 'sigma1' or 'rho' (see ?FPCA for details) (default: 'sigma2')

bwRoutine String specifying the routine used to find the optimal bandwidth 'grid-search', 'bobyqa', 'l-bfgs-b' (default: 'l-bfgs-b')

flip Logical describing if the routine should flip the sign of the singular components functions or not after the SVD of the cross-covariance matrix. (default: FALSE)

useGAM Indicator to use gam smoothing instead of local-linear smoothing (semi-parametric option) (default: FALSE)

dataType1 The type of design we have for sample 1 (usually distinguishing between sparse or dense functional data); 'Sparse', 'Dense', 'DenseWithMV' - default: determine automatically based on 'IsRegular'

dataType2 The type of design we have for sample 2 (usually distinguishing between sparse or dense functional data); 'Sparse', 'Dense', 'DenseWithMV' - default: determine automatically based on 'IsRegular'

Value

A list containing the following fields:

`bw1`	The selected (or user specified) bandwidth for smoothing the cross-covariance function across the support of sample 1.
`bw2`	The selected (or user specified) bandwidth for smoothing the cross-covariance function across the support of sample 2.
`CrCov`	The smoothed cross-covariance between samples 1 & 2.

sValues	A list of length *nsvd*, each entry containing the singuar value estimates for the FSC estimates.
nsvd	The number of singular componentes used.
canCorr	The canonical correlations for each dimension.
FVE	A percentage indicating the total variance explained by chosen FSCs with corresponding 'FVEthreshold'.
sFun1	An nWorkGrid by K matrix containing the estimated singular functions for sample 1.
sFun2	An nWorkGrid by K matrix containing the estimated singular functions for sample 2.
grid1	A vector of length nWorkGrid1. The internal regular grid on which the singular analysis is carried on the support of sample 1.
grid2	A vector of length nWorkGrid2. The internal regular grid on which the singular analysis is carried on the support of sample 2.
sScores1	A n by K matrix containing the singular scores for sample 1.
sScores2	A n by K matrix containing the singular scores for sample 2.
optns	A list of options used by the SVD and the FPCA's procedures.

FVPA *Functional Variance Process Analysis for dense functional data*

Description

Functional Variance Process Analysis for dense functional data

Usage

```
FVPA(y, t, q = 0.1, optns = list(error = TRUE, FVEthreshold = 0.9))
```

Arguments

y	A list of n vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
t	A list of n vectors containing the observation time points for each individual corresponding to y.
q	A scalar defining the percentile of the pooled sample residual sample used for adjustment before taking log (default: 0.1).
optns	A list of options control parameters specified by `list(name=value)`; by default: 'error' has to be TRUE, 'FVEthreshold' is set to 0.90. See 'Details in ?FPCA'.

Value

A list containing the following fields:

sigma2	Variance estimator of the functional variance process.
fpcaObjY	FPCA object for the original data.
fpcaObjR	FPCA object for the functional variance process associated with the original data.

References

Hans-Georg Mueller, Ulrich Stadtmuller and Fang Yao, "Functional variance processes." Journal of the American Statistical Association 101 (2006): 1007-1018

Examples

```
set.seed(1)
n <- 25
pts <- seq(0, 1, by=0.01)
sampWiener <- Wiener(n, pts)
# Data have to dense for FVPA to be relevant!
sampWiener <- Sparsify(sampWiener, pts, 101)
fvpaObj <- FVPA(sampWiener$Ly, sampWiener$Lt)
```

GetCrCorYX — *Make cross-correlation matrix from auto- and cross-covariance matrix*

Description

Make cross-correlation matrix from auto- andcross-covariance matrix

Usage

```
GetCrCorYX(ccXY, ccXX, ccYY)
```

Arguments

ccXY	The cross-covariance matrix between variables X and Y.
ccXX	The auto-covariance matrix of variable X or the diagonal of that matrix.
ccYY	The auto-covariance matrix of variable Y or the diagonal of that matrix.

Value

A cross-correlation matrix between variables X and Y.

GetCrCorYZ — *Make cross-correlation matrix from auto- and cross-covariance matrix*

Description

Make cross-correlation matrix from auto- andcross-covariance matrix

Usage

```
GetCrCorYZ(ccYZ, acYY, covZ)
```

Arguments

ccYZ	The cross-covariance vector between variables Y and Z (n-by-1).
acYY	The auto-covariance n-by-n matrix of variable Y or the (n-by-1) diagonal of that matrix.
covZ	The (scalar) covariance of variable Z.

Value

A cross-correlation matrix between variables Y (functional) and Z (scalar).

GetCrCovYX *Functional Cross Covariance between longitudinal variable Y and longitudinal variable X*

Description

Calculate the raw and the smoothed cross-covariance between functional predictors using bandwidth bw or estimate that bw using GCV.

Usage

```
GetCrCovYX(bw1 = NULL, bw2 = NULL, Ly1, Lt1 = NULL, Ymu1 = NULL,
  Ly2, Lt2 = NULL, Ymu2 = NULL, useGAM = FALSE, rmDiag = FALSE,
  kern = "gauss", bwRoutine = "l-bfgs-b")
```

Arguments

bw1	Scalar bandwidth for smoothing the cross-covariance function (if NULL it will be automatically estimated) (Y)
bw2	Scalar bandwidth for smoothing the cross-covariance function (if NULL it will be automatically estimated) (X)
Ly1	List of N vectors with amplitude information (Y)
Lt1	List of N vectors with timing information (Y)
Ymu1	Vector Q-1 Vector of length nObsGrid containing the mean function estimate (Y)
Ly2	List of N vectors with amplitude information (X)
Lt2	List of N vectors with timing information (X)
Ymu2	Vector Q-1 Vector of length nObsGrid containing the mean function estimate (X)
useGAM	Indicator to use gam smoothing instead of local-linear smoothing (semi-parametric option) (default: FALSE)
rmDiag	Indicator to remove the diagonal element when smoothing (default: FALSE)
kern	String specifying the kernel type (default: FALSE; see ?FPCA for details)
bwRoutine	String specifying the routine used to find the optimal bandwidth 'grid-search', 'bobyqa', 'l-bfgs-b' (default: 'l-bfgs-b') If the variables Ly1 and Ly2 are in matrix form the data are assumed dense and only the raw cross-covariance is returned. One can obtain Ymu1 and Ymu2 from FPCA and ConvertSupport.

Value

A list containing:

smoothedCC	The smoothed cross-covariance as a matrix (currently only 51 by 51)
rawCC	The raw cross-covariance as a list
bw	The bandwidth used for smoohting as a vector of lengh 2
score	The GCV score associated with the scalar used
smoothGrid	The grid over which the smoothed cross-covariance is evaluated

References

Yang, Wenjing, Hans-Georg Mueller, and Ulrich Stadtmueller. "Functional singular component analysis." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 73.3 (2011): 303-324

Examples

```
Ly1= list( rep(2.1,7), rep(2.1,3),2.1 );
Lt1 = list(1:7,1:3, 1);
Ly2 = list( rep(1.1,7), rep(1.1,3),1.1);
Lt2 = list(1:7,1:3, 1);
Ymu1 = rep(55,7);
Ymu2 = rep(1.1,7);
AA<-GetCrCovYX(Ly1 = Ly1, Ly2= Ly2, Lt1=Lt1, Lt2=Lt2, Ymu1=Ymu1, Ymu2=Ymu2)
```

GetCrCovYZ — *Functional Cross Covariance between longitudinal variable Y and scalar variable Z*

Description

Calculate the raw and the smoothed cross-covariance between functional and scalar predictors using bandwidth bw or estimate that bw using GCV

Usage

```
GetCrCovYZ(bw = NULL, Z, Zmu = NULL, Ly, Lt = NULL, Ymu = NULL,
  support = NULL, kern = "gauss")
```

Arguments

bw	Scalar bandwidth for smoothing the cross-covariance function (if NULL it will be automatically estimated)
Z	Vector N-1 Vector of length N with the scalar function values
Zmu	Scalar with the mean of Z (if NULL it will be automaticall estimated)
Ly	List of N vectors with amplitude information
Lt	List of N vectors with timing information
Ymu	Vector Q-1 Vector of length nObsGrid containing the mean function estimate

support	Vector of unique and sorted values for the support of the smoothed cross-covariance function (if NULL it will be automatically estimated)
kern	Kernel type to be used. See ?FPCA for more details. (defult: 'gauss') If the variables Ly1 is in matrix form the data are assumed dense and only the raw cross-covariance is returned. One can obtain Ymu1 from `FPCA` and `ConvertSupport`.

Value

A list containing:

smoothedCC	The smoothed cross-covariance as a vector
rawCC	The raw cross-covariance as a vector
bw	The bandwidth used for smoohting as a scalar
score	The GCV score associated with the scalar used

References

Yang, Wenjing, Hans-Georg Mueller, and Ulrich Stadtmueller. "Functional singular component analysis." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 73.3 (2011): 303-324

Examples

```
Ly <- list( runif(5),  c(1:3), c(2:4), c(4))
Lt <- list( c(1:5), c(1:3), c(1:3), 4)
Z = rep(4,4) # Constant vector so the covariance has to be zero.
sccObj = GetCrCovYZ(bw=1, Z= Z, Ly=Ly, Lt=Lt, Ymu=rep(4,5))
```

GetNormalisedSample *Normalise sparse functional sample*

Description

Normalise sparse functional sample given in an FPCA object

Usage

```
GetNormalisedSample(fpcaObj, errorSigma = FALSE)

GetNormalizedSample(...)
```

Arguments

fpcaObj	An FPCA object.
errorSigma	Indicator to use sigma^2 error variance when normalising the data (default: FALSE)
...	Passed into GetNormalisedSample

Value

A list containing the normalised sample 'y' at times 't'

References

Chiou, Jeng-Min and Chen, Yu-Ting and Yang, Ya-Fang. "Multivariate Functional Principal Component Analysis: A Normalization Approach" Statistica Sinica 24 (2014): 1571-1596

Examples

```
set.seed(1)
n <- 100
M <- 51
pts <- seq(0, 1, length.out=M)
mu <- rep(0, length(pts))
sampDense <- MakeGPFunctionalData(n, M, mu, K=1, basisType='sin', sigma=0.01)
samp4 <- MakeFPCAInputs(tVec=sampDense$pts, yVec=sampDense$Yn)
res4E <- FPCA(samp4$Ly, samp4$Lt, list(error=TRUE))
sampN <- GetNormalisedSample(res4E, errorSigma=TRUE)

CreatePathPlot(subset=1:20, inputData=samp4, obsOnly=TRUE, showObs=FALSE)
CreatePathPlot(subset=1:20, inputData=sampN, obsOnly=TRUE, showObs=FALSE)
```

`kCFC` *Functional clustering and identifying substructures of longitudinal data using kCFC.*

Description

Functional clustering and identifying substructures of longitudinal data using kCFC.

Usage

```
kCFC(y, t, k = 3, kSeed = 123, maxIter = 125,
  optnsSW = list(methodMuCovEst = "smooth", FVEthreshold = 0.9,
  methodBwCov = "GCV", methodBwMu = "GCV"), optnsCS = list(methodMuCovEst
  = "smooth", FVEthreshold = 0.7, methodBwCov = "GCV", methodBwMu = "GCV"))
```

Arguments

`y`	A list of n vectors containing the observed values for each individual. Missing values specified by `NA`s are supported for dense case (`dataType='dense'`).
`t`	A list of n vectors containing the observation time points for each individual corresponding to y.
`k`	A scalar defining the number of clusters to define; default 3. Values that define very small clusters (eg.cluster size <=3) will potentiall err.
`kSeed`	A scalar valid seed number to ensure replication; default: 123
`maxIter`	A scalar defining the maximum number of iterations allowed; default 20, common for both the simple kmeans initially and the subsequent k-centres
`optnsSW`	A list of options control parameters specified by `list(name=value)` to be used for sample-wide FPCA; by default: "list(methodMuCovEst ='smooth', FVEthreshold= 0.90, methodBwCov = 'GCV', methodBwMu = 'GCV')". See 'Details in ?FPCA'.

`optnsCS`	A list of options control parameters specified by `list(name=value)` to be used for cluster-specific FPCA; by default: "list(methodMuCovEst ='smooth', FVEthreshold= 0.70, methodBwCov = 'GCV', methodBwMu = 'GCV')". See 'Details in ?FPCA'.

Value

A list containing the following fields:

`cluster`	A vector of levels 1:k, indicating the cluster to which each curve is allocated.
`fpcaList`	A list with the fpcaObj for each separate cluster.
`iterToConv`	A number indicating how many iterations where required until convergence.

References

Jeng-Min Chiou, Pai-Ling Li, "Functional clustering and identifying substructures of longitudinal data." Journal of the Royal Statistical Society 69 (2007): 679-699

Examples

```
data(medfly25)
Flies <- MakeFPCAInputs(medfly25$ID, medfly25$Days, medfly25$nEggs)
kcfcObj <- kCFC(Flies$Ly[1:250], Flies$Lt[1:250], # using only 250 for speed consideration
             optnsSW = list(methodMuCovEst = 'smooth', userBwCov = 2, FVEthreshold = 0.90),
             optnsCS = list(methodMuCovEst = 'smooth', userBwCov = 2, FVEthreshold = 0.70))
```

`Lwls1D` *One dimensional local linear kernel smoother*

Description

One dimensional local linear kernel smoother for longitudinal data.

Usage

```
Lwls1D(bw, kernel_type, win = rep(1L, length(xin)), xin, yin, xout,
  npoly = 1L, nder = 0L)
```

Arguments

`bw`	Scalar holding the bandwidth
`kernel_type`	Character holding the kernel type (see ?FPCA for supported kernels)
`win`	Vector of length N with weights
`xin`	Vector of length N with measurement points
`yin`	Vector of length N with measurement values
`xout`	Vector of length M with output measurement points
`npoly`	Scalar (integer) degree of polynomial fitted (default 1)
`nder`	Scalar (integer) degree of derivative fitted (default 0)

Value

Vector of length M with measurement values at the the point speficied by 'xout'

Lwls2D *Two dimensional local linear kernel smoother.*

Description

Two dimensional local weighted least squares smoother. Only local linear smoother for estimating the original curve is available (no higher order, no derivative).

Usage

```
Lwls2D(bw, kern = "epan", xin, yin, win = NULL, xout1 = NULL,
  xout2 = NULL, xout = NULL, subset = NULL, crosscov = FALSE,
  method = ifelse(kern == "gauss", "plain", "sort2"))
```

Arguments

bw	A scalar or a vector of length 2 specifying the bandwidth.
kern	Kernel used: 'gauss', 'rect', 'gausvar', 'epan' (default), 'quar'.
xin	An n by 2 data frame or matrix of x-coordinate.
yin	A vector of y-coordinate.
win	A vector of weights on the observations.
xout1	a p1-vector of first output coordinate grid. Defaults to the input gridpoints if left unspecified.
xout2	a p2-vector of second output coordinate grid. Defaults to the input gridpoints if left unspecified.
xout	alternative to xout1 and xout2. A matrix of p by 2 specifying the output points (may be inefficient if the size of `xout` is small).
subset	a vector with the indices of x-/y-/w-in to be used (Default: NULL)
crosscov	using function for cross-covariance estimation (Default: FALSE)
method	should one try to sort the values xin and yin before using the lwls smoother? if yes ('sort2' - default for non-Gaussian kernels), if no ('plain' - fully stable; de)

Value

a p1 by p2 matrix of fitted values if xout is not specified. Otherwise a vector of length p corresponding to the rows of xout.

`Lwls2DDeriv` *Two dimensional local linear kernel smoother with derivatives.*

Description

Two dimensional local weighted least squares smoother. Only local linear smoother for estimating the original curve is available (no higher order, no derivative).

Usage

```
Lwls2DDeriv(bw, kern = "epan", xin, yin, win = NULL, xout1 = NULL,
  xout2 = NULL, xout = NULL, npoly = 1L, nder1 = 0L, nder2 = 0L,
  subset = NULL, crosscov = TRUE, method = "sort2")
```

Arguments

`bw`	A scalar or a vector of length 2 specifying the bandwidth.
`kern`	Kernel used: 'gauss', 'rect', 'gausvar', 'epan' (default), 'quar'.
`xin`	An n by 2 data frame or matrix of x-coordinate.
`yin`	A vector of y-coordinate.
`win`	A vector of weights on the observations.
`xout1`	a p1-vector of first output coordinate grid. Defaults to the input gridpoints if left unspecified.
`xout2`	a p2-vector of second output coordinate grid. Defaults to the input gridpoints if left unspecified.
`xout`	alternative to xout1 and xout2. A matrix of p by 2 specifying the output points (may be inefficient if the size of `xout` is small).
`npoly`	The degree of polynomials (include all $x^a y^b$ terms where $a + b <= npoly$)
`nder1`	Order of derivative in the first direction
`nder2`	Order of derivative in the second direction
`subset`	a vector with the indices of x-/y-/w-in to be used (Default: NULL)
`crosscov`	using function for cross-covariance estimation (Default: TRUE)
`method`	should one try to sort the values xin and yin before using the lwls smoother? if yes ('sort2' - default for non-Gaussian kernels), if no ('plain' - fully stable; de)

Value

a p1 by p2 matrix of fitted values if xout is not specified. Otherwise a vector of length p corresponding to the rows of xout.

MakeBWtoZscore02y — *Z-score body-weight for age 0 to 24 months based on WHO standards*

Description

Make vector of age and body-weight to z-scores based on WHO standards using LMS

Usage

```
MakeBWtoZscore02y(sex, age, bw)
```

Arguments

sex	A character ’M’ or ’F’ indicating the sex of the child.
age	A vector of time points of size Q.
bw	A vector of body-weight readings of size Q.

Value

A vector of Z-scores of size Q.

MakeFPCAInputs — *Format FPCA input*

Description

Turn vector inputs to the list so they can be used in FPCA

Usage

```
MakeFPCAInputs(IDs = NULL, tVec, yVec, na.rm = FALSE, sort = FALSE)
```

Arguments

IDs	n-by-1 vector of subject IDs (Default: NULL)
tVec	Either an n-by-1 vector of measurement times, or a p-by-1 vector corresponding to the common time support
yVec	n-by-1 vector of measurements from the variable of interest, or a n-by-p matrix with each row corresponding to the dense observations.
na.rm	logical indicating if NA should be omitted (Default: FALSE)
sort	logical indicating if the returned lists Lt and Ly should be ensured to be sorted (Default: FALSE)

Value

L list containing 3 lists each of length ’m’, ’m’ being the number of unique subject IDs

MakeGPFunctionalData *Make Gaussian Process Dense Functional Data sample*

Description

Make a Gaussian process dense functional data sample of size n over a [0,1] support.

Usage

```
MakeGPFunctionalData(n, M = 100, mu = rep(0, M), K = 2,
  lambda = rep(1, K), sigma = 0, basisType = "cos")
```

Arguments

n	number of samples to generate
M	number of equidistant readings per sample (default: 100)
mu	vector of size M specifying the mean (default: rep(0,M))
K	scalar specifying the number of basis to be used (default: 2)
lambda	vector of size K specifying the variance of each components (default: rep(1,K))
sigma	The standard deviation of the Gaussian noise added to each observation points.
basisType	string specifiying the basis type used; possible options are: 'sin', 'cos' and 'fourier' (default: 'cos') (See code of 'CreateBasis' for implementation details.)

Value

Y: X(t_j), Yn: noisy observations

MakeHCtoZscore02y *Z-score head-circumference for age 0 to 24 months based on WHO standards*

Description

Make vector of age and height measurement to z-scores based on WHO standards using mu and sigma (not LMS)

Usage

```
MakeHCtoZscore02y(sex, age, hc)
```

Arguments

sex	A character 'M' or 'F' indicating the sex of the child.
age	A vector of time points of size Q.
hc	A vector of head circumference readings of size Q (in cm).

Value

A vector of Z-scores of size Q.

`MakeLNtoZscore02y` *Z-score height for age 0 to 24 months based on WHO standards*

Description

Make vector of age and height measurement to z-scores based on WHO standards using mu and sigma (not LMS)

Usage

```
MakeLNtoZscore02y(sex, age, ln)
```

Arguments

`sex`	A character 'M' or 'F' indicating the sex of the child.
`age`	A vector of time points of size Q.
`ln`	A vector of body-length readings of size Q (in cm).

Value

A vector of Z-scores of size Q.

`MakeSparseGP` *Make Gaussian Process Sparse Functional Data sample*

Description

Make a Gaussian process sparse functional data sample of size n

Usage

```
MakeSparseGP(n, rdist = runif, sparsity = 2:9, muFun = function(x)
  rep(0, length(x)), K = 2, lambda = rep(1, K), sigma = 0,
  basisType = "cos", CovFun = NULL)
```

Arguments

`n`	number of samples to generate.
`rdist`	a sampler for generating the random design time points within [0, 1].
`sparsity`	A vector of integers. The number of observation per sample is chosen to be one of the elements in sparsity with equal chance.
`muFun`	a function that takes a vector input and output a vector of the corresponding mean (default: zero function).
`K`	scalar specifying the number of basis to be used (default: 2).
`lambda`	vector of size K specifying the variance of each components (default: rep(1,K)).
`sigma`	The standard deviation of the Gaussian noise added to each observation points.
`basisType`	string specifiying the basis type used; possible options are: 'sin', 'cos' and 'fourier' (default: 'cos') (See code of 'CreateBasis' for implementation details.)
`CovFun`	an alternative specification of the covariance structure.

Value

TODO

`medfly25` *Number of eggs laid daily from medflies*

Description

A dataset containing the eggs laid from 789 medflies (Mediterranean fruit flies, Ceratitis capitata) during the first 25 days of their lives. This is a subset of the dataset used by Carey at al. (1998); only flies having lived at least 25 days are shown. At the end of the recording period all flies were still alive.

Format

A data frame with 19725 rows and 3 variables:

ID : Medfly ID according to the orignal dataset

Days : Day of measurement

nEggs : Number of eggs laid at that particular day

nEggsRemain : Remaining total number of eggs laid

Source

http://anson.ucdavis.edu/~mueller/data/medfly1000.html

References

Carey, J.R., Liedo, P., Mueller, H.G., Wang, J.L., Chiou, J.M. (1998). Relationship of age patterns of fecundity to mortality, longevity, and lifetime reproduction in a large cohort of Mediterranean fruit fly females. J. of Gerontology –Biological Sciences 53, 245-251.

`MultiFAM` *Functional Additive Models with Multiple Predictor Processes*

Description

Smooth backfitting procedure for functional additive models with multiple predictor processes

Usage

```
MultiFAM(Y, X, ker = "epan", nEval = 51, XTest = NULL,
  bwMethod = 0, alpha = 0.7, supp = c(-2, 2), optnsList = NULL)
```

Arguments

`Y` An *n*-dimensional vector whose elements consist of scalar responses.

`X` A *d*-dimensional list whose components consist of two lists of *Ly* and *Lt* containing obervation times and functional covariate values for each predictor component, respectively. For details of *Ly* and *Lt*, see `FPCA` for detail.

`ker` A `function` object representing the base kernel to be used in the smooth backfitting algorithm (default is 'epan' which is the only option supported currently).

`nEval` The number of evaluation grid points for kernel smoothing (default is 51. If it is specified as 0, then estimated FPC scores in the training set are used for evaluation grid instead of equal grid).

`XTest` A *d*-dimensional list for test set of functional predictors (default is NULL). If `XTest` is specified, then estimated FPC scores in the test set are used for evaluation grid.

`bwMethod` The method of initial bandwidth selection for kernel smoothing, a positive value for designating K-fold cross-validtaion and zero for GCV (default is 0)

`alpha` The shrinkage factor (positive number) for bandwidth selection. See Han et al. (2016) (default is 0.7).

`supp` The lower and upper limits of kernel smoothing domain for studentized FPC scores, which FPC scores are divided by the square roots of eigenvalues (default is [-2,2]).

`optnsList` A *d*-dimensional list whose components consist of `optns` for each predictor component, respectively. (default is NULL which assigns the same default `optns` for all components as in `FPCA`).

Details

`MultiFAM` fits functional additive models for a scalar response and multiple predictor processes based on the smooth backfitting algorithm proposed by Han et al. (2016) that

$$E(Y|\mathbf{X}) = \sum_{j=1}^{d} \sum_{k=1}^{K_j} g_{jk}(\xi_{jk}),$$

where ξ_{jk} stand for the k-th FPC score of the j-th predictor process. `MultiFAM` only focuses on mutiple predictor processes case. In fact, the case of univariate predictor is the same with functional additive model proposed by Mueller and Yao (2008). Especially in this development, one can designate an estimation support of additive models when the additive modeling is only allowed over restricted intervals or one is interested in the modeling over the support (see Han et al., 2016).

Value

A list containing the following fields:

`mu` A scalar of centered part of the regression model.

`SBFit` An *N* by $(K_1 + ... + K_d)$ matrix whose column vectors consist of the smooth backfitting component function estimators at the given *N* estimation points.

`xi` An *N* by $(K_1 + ... + K_d)$ matrix whose column vectors consist of FPC score grid vectors that each additive component functional is evluated.

`bw` A $(K_1 + ... + K_d)$-dimensional bandwidth vector.

`lambda` A $(K_1 + ... + K_d)$-dimensional vector containing eigenvalues.

phi — A *d*-dimensional list whose components consist of an *nWorkGrid* by *K_j* matrix containing eigenfunctions, supported by `WorkGrid`. See `FPCA`.

workGrid — A *d*-dimensional list whose components consist of an *nWorkGrid* by *K_j* working grid, the internal regular grid on which the eigen analysis is carried on. See `FPCA`.

References

Mammen, E., Linton, O. and Nielsen, J. (1999), "The existence and asymptotic properties of a backfitting projection algorithm under weak conditions", Annals of Statistics, Vol.27, No.5, p.1443-1490.

Mammen, E. and Park, B. U. (2006), "A simple smooth backfitting method for additive models", Annals of Statistics, Vol.34, No.5, p.2252-2271.

Mueller, H.-G. and Yao, F. (2008), "Functional additive models", Journal of the Americal Statistical Association, Vol.103, No.484, p.1534-1544.

Han, K., Mueller, H.-G. and Park, B. U. (2016), "Smooth backfitting for additive modeling with small errors-in-variables, with an application to additive functional regression for multiple predictor functions", Bernoulli (accepted).

Examples

```
set.seed(1000)

library(MASS)

f11 <- function(t) t
f12 <- function(t) 2*cos(2*pi*t/4)
f21 <- function(t) 1.5*sin(2*pi*t/4)
f22 <- function(t) 1.5*atan(2*pi*t/4)

n<-100
N<-200

sig <- matrix(c(2.0, 0.0, 0.5, -.2,
                0.0, 1.2, -.2, 0.3,
                0.5, -.2, 1.7, 0.0,
                -.2, 0.3, 0.0, 1.0),
              nrow=4,ncol=4)

scoreX <- mvrnorm(n,mu=rep(0,4),Sigma=sig)
scoreXTest <- mvrnorm(N,mu=rep(0,4),Sigma=sig)

Y <- f11(scoreX[,1]) + f12(scoreX[,2]) + f21(scoreX[,3]) + f22(scoreX[,4]) + rnorm(n,0,0.5)
YTest <- f11(scoreXTest[,1]) + f12(scoreXTest[,2]) +
f21(scoreXTest[,3]) + f22(scoreXTest[,4]) + rnorm(N,0,0.5)

phi11 <- function(t) sqrt(2)*sin(2*pi*t)
phi12 <- function(t) sqrt(2)*sin(4*pi*t)
phi21 <- function(t) sqrt(2)*cos(2*pi*t)
phi22 <- function(t) sqrt(2)*cos(4*pi*t)

grid <- seq(0,1,length.out=21)
Lt <- Lx1 <- Lx2 <- list()
for (i in 1:n) {
```

```
  Lt[[i]] <- grid
  Lx1[[i]] <- scoreX[i,1]*phi11(grid) + scoreX[i,2]*phi12(grid) + rnorm(1,0,0.01)
  Lx2[[i]] <- scoreX[i,3]*phi21(grid) + scoreX[i,4]*phi22(grid) + rnorm(1,0,0.01)
}

LtTest <- Lx1Test <- Lx2Test <- list()
for (i in 1:N) {
  LtTest[[i]] <- grid
 Lx1Test[[i]] <- scoreXTest[i,1]*phi11(grid) + scoreXTest[i,2]*phi12(grid) + rnorm(1,0,0.01)
 Lx2Test[[i]] <- scoreXTest[i,3]*phi21(grid) + scoreXTest[i,4]*phi22(grid) + rnorm(1,0,0.01)
}

X1 <- list(Ly=Lx1, Lt=Lt)
X2 <- list(Ly=Lx2, Lt=Lt)

X1Test <- list(Ly=Lx1Test, Lt=LtTest)
X2Test <- list(Ly=Lx2Test, Lt=LtTest)

X <- list(X1, X2)
XTest <- list(X1Test, X2Test)

# estimation
sbf <- MultiFAM(Y=Y,X=X)

xi <- sbf$xi

par(mfrow=c(2,2))
j <- 1
p0 <- trapzRcpp(sort(xi[,j]),dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))
g11 <- f11(sort(xi[,j])) -
trapzRcpp(sort(xi[,j]),f11(sort(xi[,j]))*dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))/p0
tmpSgn <- sign(sum(g11*sbf$SBFit[,j]))
plot(sort(xi[,j]),g11,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi11')
points(sort(xi[,j]),tmpSgn*sbf$SBFit[order(xi[,j]),j],type='l')
legend('top',c('true','SBF'),col=c(2,1),lwd=2,bty='n',horiz=TRUE)

j <- 2
p0 <- trapzRcpp(sort(xi[,j]),dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))
g12 <- f12(sort(xi[,j])) -
trapzRcpp(sort(xi[,j]),f12(sort(xi[,j]))*dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))/p0
tmpSgn <- sign(sum(g12*sbf$SBFit[,j]))
plot(sort(xi[,j]),g12,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi12')
points(sort(xi[,j]),tmpSgn*sbf$SBFit[order(xi[,j]),j],type='l')
legend('top',c('true','SBF'),col=c(2,1),lwd=2,bty='n',horiz=TRUE)

j <- 3
p0 <- trapzRcpp(sort(xi[,j]),dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))
g21 <- f21(sort(xi[,j])) -
trapzRcpp(sort(xi[,j]),f21(sort(xi[,j]))*dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))/p0
tmpSgn <- sign(sum(g21*sbf$SBFit[,j]))
plot(sort(xi[,j]),g21,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi21')
points(sort(xi[,j]),tmpSgn*sbf$SBFit[order(xi[,j]),j],type='l')
legend('top',c('true','SBF'),col=c(2,1),lwd=2,bty='n',horiz=TRUE)

j <- 4
p0 <- trapzRcpp(sort(xi[,j]),dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))
g22 <- f22(sort(xi[,j])) -
```

```
trapzRcpp(sort(xi[,j]),f22(sort(xi[,j]))*dnorm(sort(xi[,j]),0,sqrt(sig[j,j])))/p0
tmpSgn <- sign(sum(g22*sbf$SBFit[,j]))
plot(sort(xi[,j]),g22,type='l',col=2,ylim=c(-2.5,2.5),xlab='xi22')
points(sort(xi[,j]),tmpSgn*sbf$SBFit[order(xi[,j]),j],type='l')
legend('top',c('true','SBF'),col=c(2,1),lwd=2,bty='n',horiz=TRUE)

## Not run:
# fitting
sbf <- MultiFAM(Y=Y,X=X,nEval=0)
yHat <- sbf$mu+apply(sbf$SBFit,1,'sum')
par(mfrow=c(1,1))
plot(yHat,Y)
abline(coef=c(0,1),col=2)


# R^2
R2 <- 1-sum((Y-yHat)^2)/sum((Y-mean(Y))^2)
R2


# prediction
sbf <- MultiFAM(Y=Y,X=X,XTest=XTest)
yHat <- sbf$mu+apply(sbf$SBFit,1,'sum')
par(mfrow=c(1,1))
plot(yHat,YTest)
abline(coef=c(0,1),col=2)

## End(Not run)
```

`NormCurvToArea` *Normalise a curve to a particular area.*

Description

Normalise a curve such that \intyNewdx = area (according to trapezoid integration)

Usage

```
NormCurvToArea(y, x = seq(0, 1, length.out = length(y)), area = 1)
```

Arguments

`y`	values of curve at time-points x
`x`	design time-points (default: seq(0,1, length.out=length(y)))
`area`	value to normalise the curve onto (default: 1)

Value

yNew values of curve at times x such that [\intyNewdx = area]

`predict.FPCA` *Predict FPC scores for a new sample given an FPCA object*

Description

Return a matrix with the first k FPC scores according to conditional expectation or numerical integration.

Usage

```
## S3 method for class 'FPCA'
predict(object, newLy, newLt, sigma2 = NULL, K = 1,
  xiMethod = "CE", ...)
```

Arguments

`object`	An FPCA object.
`newLy`	A list of n vectors containing the observed values for each individual.
`newLt`	A list of n vectors containing the observation time points for each individual corresponding to y.
`sigma2`	The user-defined measurement error variance. A positive scalar. (default: rho if applicable, otherwise sigma2 if applicable, otherwise 0 if no error.)
`K`	The scalar defining the number of clusters to define; (default: 1).
`xiMethod`	The integration method used to calculate the functional principal component scores (standard numerical integration 'IN' or conditional expectation 'CE'); default: 'CE'.
`...`	Not used.

Value

A matrix of size n-by-k

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt)
res
```

print.FPCA *Print an FPCA object*

Description

Print a simple description of an FPCA object

Usage

```
## S3 method for class 'FPCA'
print(x, ...)
```

Arguments

`x`	An FPCA object.
`...`	Not used.

Examples

```
set.seed(1)
n <- 20
pts <- seq(0, 1, by=0.05)
sampWiener <- Wiener(n, pts)
sampWiener <- Sparsify(sampWiener, pts, 10)
res <- FPCA(sampWiener$Ly, sampWiener$Lt)
res
```

print.FSVD *Print an FSVD object*

Description

Print a simple description of an FSVD object

Usage

```
## S3 method for class 'FSVD'
print(x, ...)
```

Arguments

`x`	An FSVD object.
`...`	Not used.

`print.WFDA` *Print a WFDA object*

Description

Print a simple description of a WFDA object

Usage

```
## S3 method for class 'WFDA'
print(x, ...)
```

Arguments

`x`	A WFDA object.
`...`	Not used.

`SBFitting` *Iterative Smooth Backfitting Algorithm*

Description

Smooth backfitting procedure for nonparametric additive models

Usage

```
SBFitting(Y, x, X, h = NULL, K = "epan", supp = NULL)
```

Arguments

`Y`	An n-dimensional vector whose elements consist of scalar responses.
`x`	An N by d matrix whose column vectors consist of N vectors of estimation points for each component function.
`X`	An n by d matrix whose row vectors consist of multivariate predictors.
`h`	A d vector of bandwidths for kernel smoothing to estimate each component function.
`K`	A `function` object representing the kernel to be used in the smooth backfitting (default is 'epan', the the Epanechnikov kernel.).
`supp`	A d by 2 matrix whose row vectors consist of the lower and upper limits of estimation intervals for each component function (default is the d-dimensional unit rectangle of *[0,1]*).

Details

`SBFitting` fits component functions of additive models for a scalar response and a multivariate predictor based on the smooth backfitting algorithm proposed by Mammen et al. (1999) and intensively studied by Mammen and Park (2006), Yu et al. (2008), Lee et al. (2010, 2012) and so on. `SBFitting` only focuses on the local constant smooth backfitting estimator with multivariate predictor case. In fact, the case of univariate predictor is the same with the local constant kernel regression estimator (Nadaraya-Watson estimator) and the local polynomial version can be extended similarly, so that those are omitted in the development. Support of the multivariate predictor is assumed to be a product of closed intervals. Especially in this development, one can designate an estimation support of additive models when the additive modeling is only allowed over restricted intervals or one is interested in the modeling over the support (see Han et al., 2016). If one puts `X` on the argument of estimation points `x`, `SBFitting` returns estimated values of conditional mean responses given observed predictors.

Value

A list containing the following fields:

`SBFit`	An N by d matrix whose column vectors consist of the smooth backfitting component function estimators at the given estimation points.
`mY`	A scalar of centered part of the regression model.
`NW`	An N by d matrix whose column vectors consist of the Nadaraya-Watson marginal regression function estimators for each predictor component at the given estimation points.
`mgnDens`	An N by d matrix whose column vectors consist of the marginal kernel density estimators for each predictor component at the given estimation points.
`jntDens`	An N by N by d by d array representing the 2-dimensional joint kernel density estimators for all pairs of predictor components at the given estimation grid. For example, `[,,j,k]` of the object provides the 2-dimensional joint kernel density estimator of the `(j,k)`-component of predictor components at the corresponding N by N matrix of estimation grid.
`itemNum`	The iteration number that the smooth backfitting algorithm has stopped.
`itemErr`	The iteration error of the smooth backfitting algorithm that represents the maximum L2 distance among component functions in the last successive updates.

References

Mammen, E., Linton, O. and Nielsen, J. (1999), "The existence and asymptotic properties of a backfitting projection algorithm under weak conditions", Annals of Statistics, Vol.27, No.5, p.1443-1490.

Mammen, E. and Park, B. U. (2006), "A simple smooth backfitting method for additive models", Annals of Statistics, Vol.34, No.5, p.2252-2271.

Yu, K., Park, B. U. and Mammen, E. (2008), "Smooth backfitting in generalized additive models", Annals of Statistics, Vol.36, No.1, p.228-260.

Lee, Y. K., Mammen, E. and Park., B. U. (2010), "backfitting and smooth backfitting for additive quantile models", Vol.38, No.5, p.2857-2883.

Lee, Y. K., Mammen, E. and Park., B. U. (2012), "Flexible generalized varying coefficient regression models", Annals of Statistics, Vol.40, No.3, p.1906-1933.

Han, K., Mueller, H.-G. and Park, B. U. (2016), "Smooth backfitting for additive modeling with small errors-in-variables, with an application to additive functional regression for multiple predictor functions", Bernoulli (accepted).

Examples

```
set.seed(100)

n <- 100
d <- 2
X <- pnorm(matrix(rnorm(n*d),nrow=n,ncol=d)%*%matrix(c(1,0.6,0.6,1),nrow=2,ncol=2))

f1 <- function(t) 2*(t-0.5)
f2 <- function(t) sin(2*pi*t)

Y <- f1(X[,1])+f2(X[,2])+rnorm(n,0,0.1)

# component function estimation
N <- 101
x <- matrix(rep(seq(0,1,length.out=N),d),nrow=N,ncol=d)
h <- c(0.12,0.08)

sbfEst <- SBFitting(Y,x,X,h)
fFit <- sbfEst$SBFit

par(mfrow=c(1,2))
plot(x[,1],f1(x[,1]),type='l',lwd=2,col=2,lty=4,xlab='X1',ylab='Y')
points(x[,1],fFit[,1],type='l',lwd=2,col=1)
points(X[,1],Y,cex=0.3,col=8)
legend('topleft',legend=c('SBF','true'),col=c(1,2),lwd=2,lty=c(1,4),horiz=FALSE,bty='n')
abline(h=0,col=8)

plot(x[,2],f2(x[,2]),type='l',lwd=2,col=2,lty=4,xlab='X2',ylab='Y')
points(x[,2],fFit[,2],type='l',lwd=2,col=1)
points(X[,2],Y,cex=0.3,col=8)
legend('topright',legend=c('SBF','true'),col=c(1,2),lwd=2,lty=c(1,4),horiz=FALSE,bty='n')
abline(h=0,col=8)

# prediction
x <- X
h <- c(0.12,0.08)

sbfPred <- SBFitting(Y,X,X,h)
fPred <- sbfPred$mY+apply(sbfPred$SBFit,1,'sum')

par(mfrow=c(1,1))
plot(fPred,Y,cex=0.5,xlab='SBFitted values',ylab='Y')
abline(coef=c(0,1),col=8)
```

SelectK — *Selects number of functional principal components for given FPCA output and selection criteria*

Description

Selects number of functional principal components for given FPCA output and selection criteria

Usage

```
SelectK(fpcaObj, criterion = "FVE", FVEthreshold = 0.95, Ly = NULL,
  Lt = NULL)
```

Arguments

`fpcaObj`	A list containing FPCA related subjects returned by MakeFPCAResults().
`criterion`	A string or positive integer specifying selection criterion for number of functional principal components, available options: 'FVE', 'AIC', 'BIC', or the specified number of components - default: 'FVE'
`FVEthreshold`	A threshold percentage specified by user when using "FVE" as selection criterion: (0,1] - default: NULL
`Ly`	A list of n vectors containing the observed values for each individual - default: NULL
`Lt`	A list of n vectors containing the observation time points for each individual corresponding to Ly - default: NULL

Value

A list including the following two fields:

`K`	An integer indicating the selected number of components based on given criterion.
`criterion`	The calculated criterion value for the selected number of components, i.e. FVE, AIC or BIC value, NULL for fixedK criterion.

`SetOptions` *Set the PCA option list*

Description

Set the PCA option list

Usage

```
SetOptions(y, t, optns)
```

Arguments

`y`	A list of n vectors containing the observed values for each individual.
`t`	A list of n vectors containing the observation time points for each individual corresponding to y.
`optns`	A list of options control parameters specified by `list(name=value)`. See 'Details'. See '?FPCAfor more details. Usually users are not supposed to use this function directly.

Sparsify *Sparsify densely observed functional data*

Description

Given a matrix of densely observed functional data, make a sparsified sample.

Usage

```
Sparsify(samp, pts, sparsity, aggressive = FALSE, fragment = FALSE)
```

Arguments

samp	A matrix of densely observed functional data, with each row containing one sample.
pts	A vector of grid points corresponding to the columns of samp.
sparsity	A vector of integers. The number of observation per sample is chosen to be one of the elements in sparsity with equal chance.
aggressive	Sparsify in an "aggressive" manner making sure that near-by readings are excluded.
fragment	Sparsify the observations into fragments, which are (almost) uniformly distributed in the time domain. Default to FALSE as not fragmenting. Otherwise a positive number specifying the approximate length of each fragment.

Value

A list of length 2, containing the following fields:

Lt	A list of observation time points for each sample.
Ly	A list of values for each sample, corresponding to the time points.

str.FPCA *Compactly display the structure of an FPCA object*

Description

Compactly display the structure of an FPCA object

Usage

```
## S3 method for class 'FPCA'
str(object, ...)
```

Arguments

object	An FPCA object
...	Not used

`Stringing` *Stringing for High-Dimensional data*

Description

Stringing for High-Dimensional data

Usage

```
Stringing(X, Y = NULL, standardize = FALSE, disOptns = "euclidean",
  disMat = NA)
```

Arguments

`X`	A matrix (n by p) of data, where X[i,] is the row vector of measurements for the ith subject.
`Y`	A vector (n by 1), where Y[i] is the reponse associated with X[i,]
`standardize`	A logical variable indicating whether standardization of the input data matrix is required, with default: FALSE.
`disOptns`	A distance metric to be used, one of the following: "euclidean" (default), "maximum", "manhattan", "canberra", "binary", "minkowski", "correlation", "spearman", "hamming", "xycor", or "user". If specified as "xycor", the absolute difference of correlation between predictor and response is used. If specified as "user", a dissimilarity matrix for the argument "disMat" must be provided.
`disMat`	A user-specified dissimilarity matrix, only necessary when "disOptns" is "user".

Value

A list containing the following fields:

`Ly`	A list of n vectors, which are the random trajectories for all subjects identified by the Stringing method.
`Lt`	A list of n time points vectors, at which corresponding measurements Ly are taken.
`StringingOrder`	A vector representing the order of the stringing, s.t. using as column index on `X` yields recovery of the underlying process.
`Xin`	A matrix, corresponding to the input data matrix.
`Xstd`	A matrix, corresponding to the standardized input data matrix. It is NULL if standardize is FALSE.

References

Chen, K., Chen, K., Mueller, H. G., and Wang, J. L. (2011). "Stringing high-dimensional data for functional analysis." Journal of the American Statistical Association, 106(493), 275-284.

Examples

```
set.seed(1)
n <- 50
wiener = Wiener(n = n)[,-1]
p = ncol(wiener)
rdmorder = sample(size = p, x=1:p, replace = FALSE)
stringingfit = Stringing(X = wiener[,rdmorder], disOptns = "correlation")
diff_norev = sum(abs(rdmorder[stringingfit$StringingOrder] - 1:p))
diff_rev = sum(abs(rdmorder[stringingfit$StringedOrder] - p:1))
if(diff_rev <= diff_norev){
  stringingfit$StringingOrder = rev(stringingfit$StringingOrder)
  stringingfit$Ly = lapply(stringingfit$Ly, rev)
}
plot(1:p, rdmorder[stringingfit$StringingOrder], pch=18); abline(a=0,b=1)
```

---

trapzRcpp *Trapezoid Rule Numerical Integration*

Description

Trapezoid Rule Numerical Integration using Rcpp

Usage

```
trapzRcpp(X, Y)
```

Arguments

X	Sorted vector of X values
Y	Vector of Y values.

TVAM *Iterative Smooth Backfitting Algorithm*

Description

Smooth backfitting procedure for time-varying additive models

Usage

```
TVAM(Lt, Ly, LLx, gridT = NULL, x = NULL, ht = NULL, hx = NULL,
  K = "epan", suppT = NULL, suppX = NULL)
```

Arguments

`Lt` An *n*-dimensional list of *N_i*-dimensional vector whose elements consist of longitudial time points for each *i*-th subject.

`Ly` An *n*-dimensional list of *N_i*-dimensional vector whose elements consist of longitudial response observations of each *i*-subject corresponding to *Lt*.

`LLx` A tuple of *d*-lists, where each list respresents longitudinal covariate observations of the *j*-th component corresponding to *Lt* and *Ly*.

`gridT` An *M*-dimensional sequence of evaluation time points for additive surface estimators. (Must be sorted in increasing orders.)

`x` An *N* by *d* matrix whose column vectors consist of *N* vectors of evaluation points for additive surface component estimators at each covariate value.

`ht` A bandwidth for kernel smoothing in time component.

`hx` A *d* vector of bandwidths for kernel smoothing covariate components, respectively.

`K` A `function` object representing the kernel to be used in the smooth backfitting (default is 'epan', the the Epanechnikov kernel.).

`suppT` A 2-dimensional vector consists of the lower and upper limits of estimation intervals for time component (default is *[0,1]*).

`suppX` A *d* by 2 matrix whose row vectors consist of the lower and upper limits of estimation intervals for each component function (default is the *d*-dimensional unit rectangle of *[0,1]*).

Details

`TVAM` estimates component surfaces of time-varying additive models for londitudinal observations based on the smooth backfitting algorithm proposed by Zhang et al. (2013). `TVAM` only focuses on the local constant smooth backfitting in contrast to the original development as in Zhang et al. (2013). However, the local polynomial version can be extended similarly, so that those are omitted in the development. Especially in this development, one can designate an estimation support of additive surfaces when the additive modeling is only allowed over restricted intervals or one is interested in the modeling over the support (see Han et al., 2016).

Value

A list containing the following fields:

`tvamComp` A tuple of *d*-lists, where each list is given by *M* by *N* matrix whose elements represents the smooth backfitting surface estimator of the *j*-component evaluated at `gridT` and the *j*-th column of `x`.

`tvamMean` An *M*-dimensional vector whose elelments consist of the marginal time regression function estimated at `gridT`.

References

Zhang, X., Park, B. U. and Wang, J.-L. (2013), "Time-varying additive models for longitudinal data", Journal of the American Statistical Association, Vol.108, No.503, p.983-998.

Han, K., Mueller, H.-G. and Park, B. U. (2018), "Smooth backfitting for additive modeling with small errors-in-variables, with an application to additive functional regression for multiple predictor functions", Bernoulli, Vol.24, No.2, p.1233-1265.

Examples

```
set.seed(1000)

n <- 100
Lt <- list()
Ly <- list()
Lx1 <- list()
Lx2 <- list()

for (i in 1:n) {
  Ni <- sample(10:15,1)

  Lt[[i]] <- sort(runif(Ni,0,1))
  Lx1[[i]] <- runif(Ni,0,1)
  Lx2[[i]] <- runif(Ni,0,1)
  Ly[[i]] <- Lt[[i]]*(cos(2*pi*Lx1[[i]]) + sin(2*pi*Lx2[[i]])) + rnorm(Ni,0,0.1)

}

LLx <- list(Lx1,Lx2)

gridT <- seq(0,1,length.out=41)
x0 <- seq(0,1,length.out=51)
x <- cbind(x0,x0)

ht <- 0.1
hx <- c(0.1,0.1)

tvam <- TVAM(Lt,Ly,LLx,gridT=gridT,x=x,ht=ht,hx=hx,K='epan')

g0Sbf <- tvam$tvamMean
gjSbf <- tvam$tvamComp

par(mfrow=c(1,2))
par(mar=c(1,1,1,1)+0.1)
persp(gridT,x0,gjSbf[[1]],theta=60,phi=30,
      xlab='time',ylab='x1',zlab='g1(t, x1)')
persp(gridT,x0,gjSbf[[2]],theta=60,phi=30,
      xlab='time',ylab='x2',zlab='g1(t, x2)')
```

VCAM *Sieve estimation*

Description

B-spline based estimation procedure for time-varying additive models

Usage

```
VCAM(Lt, Ly, X, optnAdd = list(), optnVc = list())
```

Arguments

`Lt`	An *n*-dimensional list of *N_i*-dimensional vectors whose elements consist of longitudial time points for each *i*-th subject.
`Ly`	An *n*-dimensional list of *N_i*-dimensional vectors whose elements consist of longitudial response observations of each *i*-subject corresponding to *Lt*.
`X`	An *n* by *d* matrix whose row vectors consist of covariate vector of additive components for each subject.
`optnAdd`	A list of options controls B-spline parameters for additive components, specified by list(name=value). See 'Details'.
`optnVc`	A list of options controls B-spline parameters for varying-coefficient components, specified by list(name=value). See 'Details'.

Details

`VCAM` provides a simple algorithm based on B-spline basis to estimate its nonparametric additive and varying-coefficient components.

Available control options for *optnAdd* are

nKnot A *d*-dimensional vector which designates the number of knots for each additive component function estimation (default=10).

order A *d*-dimensional vector which designates the order of B-spline basis for each additive component function estimation (default=3).

grid A *N* by *d* matrix whose column vector consist of evaluation grid points for each component function estimation.

and control options for *optnVc* are

nKnot A *(d+1)*-dimensional vector which designates the number of knots for overall mean function and each varying-coefficient component function estimation (default=10).

order A *(d+1)*-dimensional vector which designates the order of B-spline basis for overall mean function and each varying-coefficient component function estimation (default=3).

grid A *M* by *(d+1)* matrix whose column vectors consist of evaluation grid points for overall mean function and each varying-coefficient component function estimation.

Value

A list containing the following fields:

`Lt`	The same with input given by *Lt*
`LyHat`	Fitted values corresponding to *Ly*
`phiEst`	An *N* by *d* matrix whose column vectors consist of esimates for each additive component function evaluated at *gridX*.
`beta0Est`	An *M*-dimensional vector for overall mean function estimates evalueated at *gridT*.
`betaEst`	An *M* by *d* matrix whose column vectors consist of esimates for each varying-coefficient components evalueated at *gridT*.
`gridX`	The same with input given by *optnAdd$grid*
`gridT`	The same with input given by *optnVc$grid*

References

Zhang, X. and Wang, J.-L. (2015), "Varying-coefficient additive models for functional data", Biometrika, Vol.102, No.1, p.15-32.

Examples

```
library(MASS)

set.seed(100)

n <- 100
d <- 2

Lt <- list()
Ly <- list()

m <- rep(0,2)
S <- matrix(c(1,0.5,0.5,1),nrow=2,ncol=2)
X <- pnorm(mvrnorm(n,m,S))

beta0 <- function(t) 1.5*sin(3*pi*(t+0.5))
beta1 <- function(t) 3*(1-t)^2
beta2 <- function(t) 4*t^3

phi1 <- function(x) sin(2*pi*x)
phi2 <- function(x) 4*x^3-1

for (i in 1:n) {
  Ni <- sample(10:20,1)

  Lt[[i]] <- sort(runif(Ni,0,1))
 Ly[[i]] <- beta0(Lt[[i]]) + beta1(Lt[[i]])*phi1(X[i,1]) + beta2(Lt[[i]])*phi2(X[i,2]) + rnorm(Ni,0,0.1)

}


vcam <- VCAM(Lt,Ly,X)

par(mfrow=c(1,1))
plot(unlist(vcam$LyHat),unlist(Ly),xlab='observed Y',ylab='fitted Y')
abline(coef=c(0,1),col=8)

par(mfrow=c(1,2))
plot(vcam$gridX[,1],vcam$phiEst[,1],type='l',ylim=c(-1,1),xlab='x1',ylab='phi1')
points(vcam$gridX[,1],phi1(vcam$gridX[,1]),col=2,type='l',lty=2,lwd=2)
legend('topright',c('true','est'),lwd=2,lty=c(1,2),col=c(1,2))

plot(vcam$gridX[,2],vcam$phiEst[,2],type='l',ylim=c(-1,3),xlab='x2',ylab='phi2')
points(vcam$gridX[,2],phi2(vcam$gridX[,2]),col=2,type='l',lty=2,lwd=2)
legend('topleft',c('true','est'),lwd=2,lty=c(1,2),col=c(1,2))

par(mfrow=c(1,3))
plot(vcam$gridT,vcam$beta0Est,type='l',xlab='t',ylab='beta0')
points(vcam$gridT,beta0(vcam$gridT),col=2,type='l',lty=2,lwd=2)
legend('topright',c('true','est'),lwd=2,lty=c(1,2),col=c(1,2))
```

```
plot(vcam$gridT,vcam$betaEst[,1],type='l',xlab='t',ylab='beta1')
points(vcam$gridT,beta1(vcam$gridT),col=2,type='l',lty=2,lwd=2)
legend('topright',c('true','est'),lwd=2,lty=c(1,2),col=c(1,2))

plot(vcam$gridT,vcam$betaEst[,2],type='l',xlab='t',ylab='beta2')
points(vcam$gridT,beta2(vcam$gridT),col=2,type='l',lty=2,lwd=2)
legend('topright',c('true','est'),lwd=2,lty=c(1,2),col=c(1,2))
```

`WFDA` *Warped Functional Data Analysis*

Description

Pairwise curve synchronization for functional data

Usage

```
WFDA(Ly, Lt, optns = list())
```

Arguments

`Ly`	A list of *n* vectors containing the observed values for each individual.
`Lt`	A list of *n* vectors containing the observation time points for each individual corresponding to y. Each vector should be sorted in ascending order.
`optns`	A list of options control parameters specified by `list(name=value)`. See 'Details'.

Details

WFDA uses a pairwise warping method to obtain the desired alignment (registration) of the random trajectories. The data has to be regular. The routine returns the aligned curves and the associated warping function.

Available control options are

choice Choice of estimating the warping functions ('weighted' or 'truncated'). If 'weighted' then weighted averages of pairwise warping functions are computed; the weighting is based on the inverse pairwise distances. If 'truncated' the pairs with the top 10% largest distances are truncated and the simple average of the remaining pairwise distances are used - default: 'truncated'

subsetProp Pairwise warping functions are determined by using a subset of the whole sample; numeric (0,1] - default: 0.50

lambda Penalty parameter used for estimating pairwise warping functions; numeric - default : V*10^-4, where V is the average L2 norm of y-mean(y).

nknots Number of knots used for estimating the piece-wise linear pairwise warping functions; numeric - default: 2

isPWL Indicator if the resulting warping functions should piece-wise linear, if FALSE 'nknots' is ignored and the resulting warping functions are simply monotonic; logical - default: TRUE (significantly larger computation time.)

seed Random seed for the selection of the subset of warping functions; numeric - default: 666

verbose Indicator if the progress of the pairwise warping procedure should be displayed; logical - default: FALSE

Value

A list containing the following fields:

`optns`	Control options used.
`lambda`	Penalty parameter used.
`aligned`	Aligned curves evaluated at time 't'
`h`	Warping functions for 't'
`hInv`	Inverse warping functions evaluated at 't'
`costs`	The mean cost associated with each curve
`timing`	The time required by time-warping.

References

Tang, R. and Mueller, H.G. (2008). "Pairwise curve synchronization for functional data." Biometrika 95, 875-889

Tang, R. and Mueller, H.G. (2009) "Time-synchronized clustering of gene expression trajectories." Biostatistics 10, 32-45

Examples

```
N = 44;
eps = 0.123;
M = 41;
set.seed(123)
Tfinal = 3
me <- function(t) exp(-Tfinal*(((t/Tfinal^2)-0.5))^2);
T = seq(0,Tfinal,length.out = M)
recondingTimesMat = matrix(nrow = N, ncol = M)
yMat = matrix(nrow = N, ncol = M)

for (i in 1:N){
  peak = runif(min = 0.2,max =  0.8,1) * Tfinal
  recondingTimesMat[i,] = sort( unique(c( seq(0.0 , peak, length.out = round((M+1)/2)),
                            seq( peak, Tfinal, length.out = round((M+1)/2))))) * Tfinal
  yMat[i,] = me(recondingTimesMat[i,]) * rnorm(1, mean=4.0, sd=  eps)
                                       + rnorm(M, mean=0.0, sd=  eps)
}

Y = as.list(as.data.frame(t(yMat)))
X = rep(list(T),N)

sss =  WFDA(Ly = Y, Lt = X, list( choice = 'weighted' ))
par(mfrow=c(1,2))
matplot(x= T, t(yMat), t='l', main = 'Raw', ylab = 'Y'); grid()
matplot(x= T, t(sss$aligned), t='l', main = 'Aligned', ylab='Y'); grid()
```

`Wiener` *Simulate standard Wiener processes (Brownian motions)*

Description

Simulate `n` standard Wiener processes on [0, 1], possibly sparsifying the results.

Usage

```
Wiener(n = 1, pts = seq(0, 1, length = 50), sparsify = NULL,
  K = 50)
```

Arguments

`n`	Sample size.
`pts`	A vector of points in [0, 1] specifying the support of the processes.
`sparsify`	A vector of integers. The number of observations per curve will be uniform distribution on sparsify.
`K`	The number of components.

Details

The algorithm is based on Karhunen-Loeve expansion.

Value

If `sparsify` is not specified, a matrix with `n` rows corresponding to the samples are returned. Otherwise the sparsified sample is returned.

See Also

Sparsify

Index

BwNN, 3

CheckData, 4
CheckOptions, 4
ConvertSupport, 5
CreateBasis, 5
CreateBWPlot, 6
CreateCovPlot, 6
CreateDesignPlot, 7
CreateDiagnosticsPlot, 8
CreateFuncBoxPlot, 9
CreateModeOfVarPlot, 10
CreateOutliersPlot, 10
CreatePathPlot, 12
CreateScreePlot, 13
CreateStringingPlot, 13
cumtrapzRcpp, 14

Dyn_test, 15
DynCorr, 14

FAM, 16
FCCor, 19
FClust, 21
FCReg, 22
fdapace, 24
fdapace-package (fdapace), 24
fitted.FPCA, 24
fitted.FPCAder, 26
FOptDes, 27
FPCA, 28
FPCAder, 31
FPCquantile, 33
FSVD, 34
FVPA, 36

GetCrCorYX, 37
GetCrCorYZ, 37
GetCrCovYX, 38
GetCrCovYZ, 39
GetNormalisedSample, 40
GetNormalizedSample
(GetNormalisedSample), 40

kCFC, 41

Lwls1D, 42
Lwls2D, 43
Lwls2DDeriv, 44

MakeBWtoZscore02y, 45
MakeFPCAInputs, 45
MakeGPFunctionalData, 46
MakeHCtoZscore02y, 46
MakeLNtoZscore02y, 47
MakeSparseGP, 47
medfly25, 48
MultiFAM, 48

NormCurvToArea, 52

plot.FPCA (CreateDiagnosticsPlot), 8
predict.FPCA, 53
print.FPCA, 54
print.FSVD, 54
print.WFDA, 55

SBFitting, 55
SelectK, 57
SetOptions, 58
Sparsify, 59
str.FPCA, 59
Stringing, 60

trapzRcpp, 61
TVAM, 61

VCAM, 63

WFDA, 66
Wiener, 68