Package 'roahd'

August 24, 2018

Type Package

Title Robust Analysis of High Dimensional Data

Version 1.4.1

Date 2018-08-18

Author Nicholas Tarabelloni [aut, cre],
Ana Arribas-Gil [aut],
Francesca Ieva [aut],
Anna Maria Paganoni [aut],
Juan Romo [aut],
Francesco Palma [ctb]

Maintainer Nicholas Tarabelloni <nicholas.tarabelloni@polimi.it>

Description A collection of methods for the robust analysis of univariate and multivariate functional data, possibly in high-dimensional cases, and hence with attention to computational efficiency and simplicity of use.

Depends R (>= 2.10)

License GPL-3

LazyData yes

Suggests testthat, knitr, rmarkdown

Imports scales, robustbase, magrittr, dplyr

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-08-23 22:30:03 UTC

R topics documented:

`append_fData` *Append two compatible univariate functional datasets*

Description

This is a convenience function that simplifies the task of appending univariate functional observations of two datasets to a unique univariate functional dataset.

Usage

```
append_fData(fD1, fD2)
```

Arguments

`fD1`	is the first functional dataset, stored into an `fData` object.
`fD2`	is the second functional dataset, stored into an `fData` object.

Details

The two original datasets must be compatible, i.e. must be defined on the same grid. If we denote with $X_1, \ldots, X_n$ the first dataset, defined over the grid $I = t_1, \ldots, t_P$, and with $Y_1, \ldots, Y_m$ the second functional dataset, defined on the same grid, the method returns the union dataset obtained by taking all the $n + m$ observations together.

Value

The function returns an `fData` object containing the union of `fD1` and `fD2`

See Also

`append_mfData`, `fData`

Examples

```
# Creating two simple univariate datasets

grid = seq(0, 2 * pi, length.out = 100)

values1 = matrix( c(sin(grid),
                    sin(2 * grid)), nrow = 2, ncol = length(grid),
                  byrow=TRUE)
```

```
values2 = matrix( c(cos(grid),
                    cos(2 * grid)), nrow = 2, ncol = length(grid),
                   byrow=TRUE)

fD1 = fData( grid, values1 )
fD2 = fData( grid, values2 )

# Appending them to a unique dataset
append_fData(fD1, fD2)
```

---

append_mfData *Append two compatible multivariate functional datasets*

Description

This is a convenience function that simplifies the task of appending multivariate functional observations of two datasets to a unique multivariate functional dataset.

Usage

```
append_mfData(mfD1, mfD2)
```

Arguments

mfD1	is the first multivariate functional dataset, stored into an mfData object.
mfD2	is the second multivariate functional dataset, stored into an mfData object.

Details

The two original datasets must be compatible, i.e. must have same number of components (dimensions) and must be defined on the same grid. If we denote with $X_1^(i), \ldots, X_n^(i)$, $i = 0, \ldots, L$ the first dataset, defined over the grid $I = t_1, \ldots, t_P$, and with $Y_1^(i), \ldots, Y_m^(i)$, $i = 0, \ldots, L$ the second functional dataset, the method returns the union dataset obtained by taking all the $n + m$ observations together.

Value

The function returns a mfData object containing the union of mfD1 and mfD2

See Also

append_fData, mfData

Examples

```
# Creating two simple bivariate datasets

grid = seq(0, 2 * pi, length.out = 100)

values11 = matrix( c(sin(grid),
                     sin(2 * grid)), nrow = 2, ncol = length(grid),
                   byrow=TRUE)
values12 = matrix( c(sin(3 * grid),
                     sin(4 * grid)), nrow = 2, ncol = length(grid),
                   byrow=TRUE)
values21 = matrix( c(cos(grid),
                     cos(2 * grid)), nrow = 2, ncol = length(grid),
                   byrow=TRUE)
values22 = matrix( c(cos(3 * grid),
                     cos(4 * grid)), nrow = 2, ncol = length(grid),
                   byrow=TRUE)

mfD1 = mfData( grid, list(values11, values12) )
mfD2 = mfData( grid, list(values21, values22) )

# Appending them to a unique dataset
append_mfData(mfD1, mfD2)
```

---

area_ordered *Area-under-curve order relation between univariate functional data*

Description

This function implements an order relation between univariate functional data based on the area-under-curve relation, that is to say a pre-order relation obtained by comparing the area-under-curve of two different functional data.

Usage

```
area_ordered(fData, gData)
```

Arguments

fData	the first univariate functional dataset containing elements to be compared, in form of fData object.
gData	the second univariate functional dataset containing elements to be compared , in form of fData object.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$ and another functional dataset $Y_1(t)$, $Y_2(t), \ldots, Y_M(t)$ defined over the same compact interval $I = [a, b]$, the function computes the area-under-curve (namely, the integral) in both the datasets, and checks whether the first ones are lower or equal than the second ones.

By default the function tries to compare each $X_i(t)$ with the corresponding $Y_i(t)$, thus assuming $N = M$, but when either $N = 1$ or $M = 1$, the comparison is carried out cycling over the dataset with fewer elements. In all the other cases ($N \neq M$, and either $N \neq 1$ or $M \neq 1$) the function stops.

Value

The function returns a logical vector of length $\max(N, M)$ containing the value of the predicate for all the corresponding elements.

References

Valencia, D., Romo, J. and Lillo, R. (2015). A Kendall correlation coefficient for functional dependence, *Universidad Carlos III de Madrid technical report*, http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133228

See Also

area_under_curve, fData

Examples

```
P = 1e3

grid = seq( 0, 1, length.out = P )

Data_1 = matrix( c( 1 * grid,
                    2 *  grid ),
                 nrow = 2, ncol = P, byrow = TRUE )

Data_2 = matrix( 3 * ( 0.5 - abs( grid - 0.5 ) ),
                 nrow = 1, byrow = TRUE )

Data_3 = rbind( Data_1, Data_1 )


fD_1 = fData( grid, Data_1 )
fD_2 = fData( grid, Data_2 )
fD_3 = fData( grid, Data_3 )

area_ordered( fD_1, fD_2 )

area_ordered( fD_2, fD_3 )
```

area_under_curve *Area under curve of elements of univariate functional data*

Description

This method computes the (signed) area under the curve of elements of a univariate functional dataset, namely, their integral.

Usage

```
area_under_curve(fData)
```

Arguments

fData	the functional dataset containing elements whose areas under the curve have to be computed, in form of fData object.

Value

The function returns a numeric vector containing the values of areas under the curve for all the elements of the functional dataset fData.

See Also

area_ordered, fData

Examples

```
P = 1e3
grid = seq( 0, 1, length.out = P )

fD = fData( grid,
            matrix( c( sin( 2 * pi * grid ),
                       cos( 2 * pi * grid ),
                       4 * grid * ( 1 - grid ) ),
                    nrow = 3, ncol = P, byrow = TRUE ) )
plot( fD )

area_under_curve( fD )
```

as.mfData *Converting object to* mfData *class*

Description

This S3 method provides a way to convert some objects to the class mfData, thus obtaining a multivariate functional dataset.

Usage

```
as.mfData(x, ...)

## S3 method for class 'list'
as.mfData(x, ...)
```

Arguments

x	a list of univariate functional datasets, provided in form of fData objects.
...	additional parameters.

Value

The function returns a mfData object, obtained starting from argument x.

Examples

```
grid = seq( 0, 1, length.out = 100 )

fD_1 = fData( grid, sin( 2 * pi * grid ) )
fD_2 = fData( grid, cos( 2 * pi * grid ) )

plot( as.mfData( list( fD_1, fD_2 ) ) )
```

BCIntervalSpearman *Bootstrap Confidence Interval on Spearman's Correlation Coefficient between Univariate Functional Datasets*

Description

This function computes the bootstrap confidence interval of coverage probability $1 - \alpha$ for the Spearman correlation coefficient between two univariate functional samples.

Usage

```
BCIntervalSpearman(fD1, fD2, ordering = "MEI", bootstrap_iterations = 1000,
  alpha = 0.05, verbose = FALSE)
```

Arguments

fD1	is the first univariate functional sample in form of an fData object.
fD2	is the first univariate functional sample in form of an fData object.
ordering	is either MEI (default) or MHI, and indicates the ordering relation to be used during in the Spearman's coefficient computation.
bootstrap_iterations	is the number of bootstrap iterations to use in order to estimate the confidence interval (default is 1000).
alpha	controls the coverage probability (1-alpha).
verbose	whether to log information on the progression of bootstrap iterations.

Details

The function takes two samples of compatible functional data (i.e., they must be defined over the same grid and have same number of observations) and computes a bootstrap confidence interval for their Spearman correlation coefficient.

Value

The function returns a list of two elements, lower and upper, representing the lower and upper end of the bootstrap confidence interval.

See Also

cor_spearman, cor_spearman_accuracy, fData, mfData, BCIntervalSpearmanMultivariate

Examples

```
set.seed(1)

N = 2e2
P = 1e2
grid = seq( 0, 1, length.out = P )

# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating (independent) gaussian functional data with given center and
# covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )

# Using the simulated data as (independent) components of a bivariate functional
```

```
# dataset
mfD = mfData( grid, list( Data_1, Data_2 ) )
## Not run:
BCIntervalSpearman(mfD$fDList[[1]], mfD$fDList[[2]], ordering = 'MEI')

BCIntervalSpearman(mfD$fDList[[1]], mfD$fDList[[2]], ordering = 'MHI')

## End(Not run)
# BC intervals contain zero since the functional samples are uncorrelated.
```

`BCIntervalSpearmanMultivariate` *Bootstrap Confidence Interval on Spearman's Correlation Coefficient of a Multivariate Functional Dataset*

Description

This function computes the bootstrap confidence intervals of coverage probability $1 - \alpha$ for the Spearman correlation coefficients within a multivariate functional dataset.

Usage

```
BCIntervalSpearmanMultivariate(mfD, ordering = "MEI",
  bootstrap_iterations = 1000, alpha = 0.05, verbose = FALSE)
```

Arguments

`mfD`	is the multivariate functional sample in form of `mfData` object.
`ordering`	is either `MEI` (default) or `MHI`, and indicates the ordering relation to be used during in the Spearman's coefficient computation.
`bootstrap_iterations`	is the number of bootstrap iterations to use in order to estimate the confidence intervals (default is 1000).
`alpha`	controls the coverage probability (1-`alpha`).
`verbose`	whether to log information on the progression of bootstrap iterations.

Details

The function takes a multivariate functional dataset and computes a matrix of bootstrap confidence intervals for its Spearman correlation coefficients.

Value

The function returns a list of two elements, `lower` and `upper`, representing the matrices of lower and upper ends of the bootstrap confidence intervals for each pair of components. The elements on the main diagonal are set to 1.

See Also

cor_spearman, cor_spearman_accuracy, fData, mfData, BCIntervalSpearman

Examples

```
set.seed(1)

N = 2e2
P = 1e2
grid = seq( 0, 1, length.out = P )

# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating (independent) gaussian functional data with given center and
# covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )
Data_2 = generate_gauss_fdata( N, centerline = sin( 4 * pi * grid ), Cov = Cov )
Data_3 = generate_gauss_fdata( N, centerline = sin( 6 * pi * grid ), Cov = Cov )

# Using the simulated data as (independent) components of a multivariate functional
# dataset
mfD = mfData( grid, list( Data_1, Data_2, Data_3 ) )

## Not run:
BCIntervalSpearmanMultivariate(mfD, ordering = 'MEI')

## End(Not run)
# BC intervals contain zero since the functional samples are uncorrelated.
```

BD *Band Depth for univariate functional data*

Description

This function computes the Band Depth (BD) of elements of a functional dataset.

Usage

```
BD(Data)

## S3 method for class 'fData'
BD(Data)

## Default S3 method:
BD(Data)
```

Arguments

`Data`	either an object of class `fData` or a matrix-like dataset of functional data (e.g. `fData$values`), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, this function computes the sample BD of each element with respect to the other elements of the dataset, i.e.:

$$BD(X(t)) = \binom{N}{2}^{-1} \sum_{1 \leq i_1 < i_2 \leq N} I(G(X) \subset B(X_{i_1}, X_{i_2})),$$

where $G(X)$ is the graphic of $X(t)$, $B(X_{i_1}, X_{i_2})$ is the envelope of $X_{i_1}(t)$ and $X_{i_2}(t)$, and $X \in \{X_1, \ldots, X_N\}$.

See the References section for more details.

Value

The function returns a vector containing the values of BD for the given dataset.

References

Lopez-Pintado, S. and Romo, J. (2009). On the Concept of Depth for Functional Data, *Journal of the American Statistical Association*, 104, 718-734.

Lopez-Pintado, S. and Romo. J. (2007). Depth-based inference for functional data, *Computational Statistics & Data Analysis* 51, 4957-4968.

See Also

`MBD`, `BD_relative`, `MBD_relative`, `fData`

Examples

```
grid = seq( 0, 1, length.out = 1e2 )


D = matrix( c( 1 + sin( 2 * pi * grid ),
               0 + sin( 4 * pi * grid ),
               1 - sin( pi * ( grid - 0.2 ) ),
               0.1 + cos( 2 * pi * grid ),
               0.5 + sin( 3 * pi + grid ),
               -2 + sin( pi * grid ) ),
            nrow = 6, ncol = length( grid ), byrow = TRUE )

fD = fData( grid, D )

BD( fD )
```

```
BD( D )
```

`BD_relative` *Relative Band Depth of functions in a univariate functional dataset*

Description

This function computes Band Depth (BD) of elements of a univariate functional dataset with respect to another univariate functional dataset.

Usage

```
BD_relative(Data_target, Data_reference)

## S3 method for class 'fData'
BD_relative(Data_target, Data_reference)

## Default S3 method:
BD_relative(Data_target, Data_reference)
```

Arguments

`Data_target`	is the univariate functional dataset, provided either as a `fData` object or in matrix form (N observations as rows and P measurements as columns), whose BD have to be computed with respect to the reference dataset.
`Data_reference`	is the dataset, provided either as a `fData` object or in matrix form (N observations as rows and P measurements as columns), containing the reference univariate functional data that must be used to compute the band depths of elements in `Data_target`. If `Data_target` is `fData`, it must be of class `fData`

Details

Given a univariate functional dataset of elements $X_1(t), X_2(t), \ldots, X_N(t)$, and another univariate functional dataset of elements $Y_1(t), Y_2(t) \ldots, Y_M(t)$, this function computes the BD of elements of the former with respect to elements of the latter, i.e.:

$$BD(X_i(t)) = \binom{M}{2}^{-1} \sum_{1 \leq i_1 < i_2 \leq M} I(G(X_i) \subset B(Y_{i_1}, Y_{i_2})), \quad \forall i = 1, \ldots, N,$$

where $G(X_i)$ is the graphic of $X_i(t)$ and $B(Y_{i_1}, Y_{i_2})$ is the envelope of $Y_{i_1}(t)$ and $Y_{i_2}(t)$.

Value

The function returns a vector containing the BD of elements in `Data_target` with respect to elements in `Data_reference`.

See Also

BD, MBD, MBD_relative, fData

Examples

```
grid = seq( 0, 1, length.out = 1e2 )

Data_ref = matrix( c( 0  + sin( 2 * pi * grid ),
                      1  + sin( 2 * pi * grid ),
                      -1 + sin( 2 * pi * grid ) ),
                   nrow = 3, ncol = length( grid ), byrow = TRUE )

Data_test_1 = matrix( c( 0.6 + sin( 2 * pi * grid ) ),
                      nrow = 1, ncol = length( grid ), byrow = TRUE )

Data_test_2 = matrix( c( 0.6 + sin( 2 * pi * grid ) ),
                      nrow = length( grid ), ncol = 1, byrow = TRUE )

Data_test_3 = 0.6 + sin( 2 * pi * grid )

Data_test_4 = array( 0.6 + sin( 2 * pi * grid ), dim = length( grid ) )

Data_test_5 = array( 0.6 + sin( 2 * pi * grid ), dim = c( 1, length( grid ) ) )

Data_test_6 = array( 0.6 + sin( 2 * pi * grid ), dim = c( length( grid ), 1 ) )

Data_test_7 = matrix( c( 0.5  + sin( 2 * pi * grid ),
                         -0.5 + sin( 2 * pi * grid ),
                         1.1 + sin( 2 * pi * grid ) ),
                      nrow = 3, ncol = length( grid ), byrow = TRUE )

fD_ref = fData( grid, Data_ref )
fD_test_1 = fData( grid, Data_test_1 )
fD_test_2 = fData( grid, Data_test_2 )
fD_test_3 = fData( grid, Data_test_3 )
fD_test_4 = fData( grid, Data_test_4 )
fD_test_5 = fData( grid, Data_test_5 )
fD_test_6 = fData( grid, Data_test_6 )
fD_test_7 = fData( grid, Data_test_7 )

BD_relative( fD_test_1, fD_ref )
BD_relative( Data_test_1, Data_ref )

BD_relative( fD_test_2, fD_ref )
BD_relative( Data_test_2, Data_ref )

BD_relative( fD_test_3, fD_ref )
BD_relative( Data_test_3, Data_ref )

BD_relative( fD_test_4, fD_ref )
BD_relative( Data_test_4, Data_ref )
```

```
BD_relative( fD_test_5, fD_ref )
BD_relative( Data_test_5, Data_ref )

BD_relative( fD_test_6, fD_ref )
BD_relative( Data_test_6, Data_ref )

BD_relative( fD_test_7, fD_ref )
BD_relative( Data_test_7, Data_ref )
```

`BTestSpearman` *Bootstrap Hypothesis Test on Spearman Correlation Coefficients for Multivariate Functional Data*

Description

This function performs a bootstrap test that checks whether the Spearman correlation structures (e.g. matrices) of two populations of compatible multivariate functional data are equal or not.

Usage

```
BTestSpearman(mfD1, mfD2, bootstrap_iterations = 1000, ordering = "MEI",
  normtype = "f", verbose = FALSE)
```

Arguments

`mfD1`	is the first functional dataset, specified in form of `mfData` object; it must be compatible with `mfD2`.
`mfD2`	is the second functional dataset, specified in form of `mfData` object; it must be compatible with `mfD1`.
`bootstrap_iterations`	is the number of bootstrap iterations to be performed.
`ordering`	is the kind of ordering to be used in the computation of Spearman's correlation coefficeint (default is `MEI`).
`normtype`	is the norm to be used when comparing the Spearman correlation matrices of the two functional datasets (default is Frobenius, allowed values are the same as for parameter `type` in the base function `norm`).
`verbose`	a boolean flag specifying whether to print the progress of bootstrap iterations or not (default is FALSE).

Details

Given a first multivariate functional population, $X_1^(i), \ldots, X_n^(i)$ with $i = 1, \ldots, L$, defined on the grid $I = t_1, \ldots, t_P$, and a second multivariate functional population, $Y_1^(i), \ldots, Y_m^(i)$ with $i = 1, \ldots, L$, defined on the same grid I, the function performs a bootstrap test to check the hypothesis:

$$H_0 : R_X = R_Y$$

$$H_1 : R_X \neq R_Y,$$

where R_X and R_Y denote the L x L matrices of Spearman correlation coefficients of the two populations.

- The two functional samples must have the same number of components and must be defined over the same discrete interval $t_1, \ldots, t_P$.
- The test is performed through a bootstrap argument, so a number of bootstrap iterations must be specified as well. A high value for this parameter may result in slow performances of the test (you may consider setting `verbose` to `TRUE` to get hints on the process).

Value

The function returns the estimates of the test's p-value and statistics.

See Also

BCIntervalSpearman, BCIntervalSpearmanMultivariate, mfData

Examples

```
set.seed(1)
N = 2e2
P = 1e2
L = 2
grid = seq( 0, 1, length.out = P )
# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )



# Simulating two populations of bivariate functional data
#
# The first population has very high correlation between first and second component
centerline_1 = matrix(rep(sin(2 * pi * grid)), nrow = 2, ncol=P, byrow=TRUE)
values1 = generate_gauss_mfdata( N, L, correlations = 0.9,
                                 centerline = centerline_1, listCov = list(Cov, Cov)  )
mfD1 = mfData(grid, values1)

# Pointwise estimate
cor_spearman(mfD1)

# The second population has zero correlation between first and second component
centerline_2 = matrix(rep(cos(2 * pi * grid)), nrow = 2, ncol=P, byrow=TRUE)
values2 = generate_gauss_mfdata( N, L, correlations = 0,
                                 centerline = centerline_1, listCov = list(Cov, Cov)  )
mfD2 = mfData(grid, values2)
```

```
# Pointwise estimate
cor_spearman(mfD2)

# Applying the test
## Not run:
BTestSpearman(mfD1, mfD2)

## End(Not run)
```

`cor_kendall` *Kendall's tau correlation coefficient for bivariate functional data*

Description

This function computes the Kendall's tau correlation coefficient for a bivariate functional dataset, with either a max or area-under-curve order order relation between univariate functional elements (components).

Usage

```
cor_kendall(mfD, ordering = "max")
```

Arguments

`mfD`	a bivariate functional dataset whose Kendall's tau coefficient must be computed, in form of bivariate `mfData` object (`mfD$L=2`).
`ordering`	the ordering relation to use on functional observations, either `"max"` for the maximum relation or `"area"` for the area under the curve relation (default is `"max"`).

Details

Given a bivariate functional dataset, with first components $X_1(t), X_2(t), \ldots, X_N(t)$ and second components $Y_1(t), Y_2(t), \ldots, Y_N(t)$, the function exploits either the order relation based on the maxima or the area-under-curve relation to compare data and produce concordances and discordances, that are then used to compute the tau coefficient.

See the references for more details.

Value

The function returns the Kendall's tau correlation coefficient for the bivariate dataset provided with `mfData`.

References

Valencia, D., Romo, J. and Lillo, R. (2015). A Kendall correlation coefficient for functional dependence, *Universidad Carlos III de Madrid technical report*, `http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133228`

See Also

mfData, area_ordered, max_ordered

Examples

```
#### TOTALLY INDEPENDENT COMPONENTS
N = 2e2
P = 1e3

grid = seq( 0, 1, length.out = P )

# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating (independent) gaussian functional data with given center and
# covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )

# Using the simulated data as (independent) components of a bivariate functional
# dataset
mfD = mfData( grid, list( Data_1, Data_2 ) )

# Correlation approx. zero (components were created independently)
cor_kendall( mfD, ordering = 'max' )

# Correlation approx. zero (components were created independently)
cor_kendall( mfD, ordering = 'area' )

#### TOTALLY DEPENDENT COMPONENTS

# Nonlinear transform of first component
Data_3 = t( apply( Data_1, 1, exp ) )

# Creating bivariate dataset starting from nonlinearly-dependent components
mfD = mfData( grid, list( Data_1, Data_3 ) )

# Correlation very high (components are nonlinearly dependent)
cor_kendall( mfD, ordering = 'max' )

# Correlation very high (components are nonlinearly dependent)
cor_kendall( mfD, ordering = 'area' )
```

---

cor_spearman *Spearman's correlation coefficient for multivariate functional data*

Description

This function computes the Spearman's correlation coefficient for a multivariate functional dataset, with either a Modified Epigraph Index (MEI) or Modified Hypograph Index (MHI) ranking of univariate elments of data components.

Usage

```
cor_spearman(mfD, ordering = "MEI")
```

Arguments

`mfD`	a multivariate functional dataset whose Spearman's correlation coefficient must be computed, in form of multivariate `mfData` object.
`ordering`	the ordering relation to use on functional observations, either `"MEI"` for MEI or `"MHI"` for MHI (default is `"MEI"`).

Details

Given a multivariate functional dataset, with first components $X_1^1(t), X_2^1(t), \ldots, X_N^1(t)$, second components $X_1^2(t), X_2^2(t), \ldots, X_N^2(t)$, etc., the function exploits either the MEI or MHI to compute the matrix of Spearman's correlation coefficients. Such matrix is symmetrical and has ones on the diagonal. The entry (i, j) represents the Spearman correlation coefficient between curves of component i and j.

See the references for more details.

Value

If the original dataset is bivariate, the function returns only the scalar value of the correlation coefficient for the two components. When the number of components is L >2, it returns the whole matrix of Spearman's correlation coefficients for all the components.

References

Valencia, D., Romo, J. and Lillo, R. (2015). Spearman coefficient for functions, *Universidad Carlos III de Madrid technical report*, `http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133329`.

See Also

`mfData`, `MEI`, `MHI`

Examples

```
#### TOTALLY INDEPENDENT COMPONENTS

N = 2e2
P = 1e3

grid = seq( 0, 1, length.out = P )
```

```
# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating (independent) gaussian functional data with given center and
# covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )

# Using the simulated data as (independent) components of a bivariate functional
# dataset
mfD = mfData( grid, list( Data_1, Data_2 ) )

# Correlation approx. zero (components were created independently)
cor_spearman( mfD, ordering = 'MEI' )

# Correlation approx. zero (components were created independently)
cor_spearman( mfD, ordering = 'MHI' )

#### TOTALLY DEPENDENT COMPONENTS

# Nonlinear transform of first component
Data_3 = t( apply( Data_1, 1, exp ) )

# Creating bivariate dataset starting from nonlinearly-dependent components
mfD = mfData( grid, list( Data_1, Data_3 ) )

# Correlation very high (components are nonlinearly dependent)
cor_spearman( mfD, ordering = 'MEI' )

# Correlation very high (components are nonlinearly dependent)
cor_spearman( mfD, ordering = 'MHI' )
```

`cor_spearman_accuracy` *Bootstrap Spearman's correlation coefficient for multivariate functional data*

Description

This function computes the bootstrap estimates of standard error and bias of the Spearman's correlation coefficient for a multivariate functional dataset.

Usage

```
cor_spearman_accuracy(mfD, ordering = "MEI", bootstrap_iterations = 1000,
  verbose = FALSE)
```

Arguments

mfD	a multivariate functional dataset whose Spearman's correlation coefficient must be computed, in form of multivariate mfData object.
ordering	the ordering relation to use on functional observations, either "MEI" for MEI or "MHI" for MHI (default is "MEI").
bootstrap_iterations	the number of bootstrap iterations to be used for estimation of bias and standard error.
verbose	a logical flag specifying whether to log information on the estimation progress.

Details

Given a multivariate functional dataset $X_1^(i), \ldots, X_n^(i)$, $i = 0, \ldots, L$ defined over the grid $I = t_0, \ldots, t_P$, having components $i = 1, \ldots, L$, and a chosen ordering strategy (MEI or MHI), the function computes the matrix of Speraman's correlation indexes of the dataset's components, as well as their bias and standard deviation estimates through a specified number of bootstrap iterations (bias and standard error are updated with on-line formulas).

Value

a list of three elements: mean, the mean of the matrix of correlation coefficients; bias, a matrix containing the estimated bias (mean - point estimate of correlation coefficients); sd, a matrix containing the estiated standard deviation of the coefficients' matrix. In case the multivariate functional dataset has only two components, the return type is scalar and not matrix.

See Also

cor_spearman, mfData

Examples

```
N = 2e2
P = 1e2
grid = seq( 0, 1, length.out = P )

# Creating an exponential covariance function to simulate guassian data
Cov = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating (independent) gaussian functional data with given center and covariance function

Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = Cov )

# Using the simulated data as (independent) components of a bivariate functional dataset
mfD = mfData( grid, list( Data_1, Data_2 ) )
## Not run:
cor_spearman_accuracy(mfD, ordering='MEI')

cor_spearman_accuracy(mfD, ordering='MHI')
```

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

---

cov_fun *Covariance function for functional data*

Description

S3 method to compute the sample covariance and cross-covariance functions for a set of functional data.

Usage

```
cov_fun(X, Y = NULL)

## S3 method for class 'fData'
cov_fun(X, Y = NULL)

## S3 method for class 'mfData'
cov_fun(X, Y = NULL)
```

Arguments

X	is the (eventually first) functional dataset, i.e. either an object of class fData or an object of class mfData;
Y	is the (optional) second functional dataset to be used to compute the cross-covariance function, either NULL or an fData or mfData object (see the Value section for details).

Details

Given a univariate random function X, defined over the grid $I = [a, b]$, the covariance function is defined as:

$$C(s,t) = Cov(X(s), X(t)), \qquad s,t \in I.$$

Given another random function, Y, defined over the same grid as X, the cross- covariance function of X and Y is:

$$C^{X,Y}(s,t) = Cov(X(s), Y(t)), \qquad s,t \in I.$$

For a generic L-dimensional random function X, i.e. an L-dimensional multivariate functional datum, the covariance function is defined as the set of blocks:

$$C_{i,j}(s,t) = Cov(X_i(s), X_j(t)), i,j = 1, ..., L, s,t \in I,$$

while the cross-covariance function is defined by the blocks:

$$C_{i,j}^{X,Y}(s,t) = Cov(X_i(s), Y_j(t))$$

The method cov_fun provides the sample estimator of the covariance or cross-covariance functions for univariate or multivariate functional datasets.

The class of X (fData or mfData) is used to dispatch the correct implementation of the method.

Value

The following cases are given:

- if X is of class fData and Y is NULL, then the covariance function of X is returned;
- if X is of class fData and Y is of class fData, the cross-covariance function of the two datasets is returned;
- if X is of class mfData and Y is NULL, the upper-triangular blocks of the covariance function of X are returned (in form of list and by row, i.e. in the squence 1_1, 1_2, ..., 1_L, 2_2, ... - have a look at the labels of the list with str);
- if X is of class mfData and Y is of class fData, the cross-covariances of X's components and Y are returned (in form of list);
- if X is of class mfData and Y is of class mfData, the upper-triangular blocks of the cross-covariance of X's and Y's components are returned (in form of list and by row, i.e. in the squence 1_1, 1_2, ..., 1_L, 2_2, ... - have a look at the labels of the list with str));

In any case, the return type is either an instance of the S3 class Cov or a list of instances of such class (for the case of multivariate functional data).

See Also

fData, mfData, plot.Cov

Examples

```
# Generating a univariate functional dataset
N = 1e2

P = 1e2
t0 = 0
t1 = 1

time_grid = seq( t0, t1, length.out = P )

Cov = exp_cov_function( time_grid, alpha = 0.3, beta = 0.4 )

D1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * time_grid ), Cov = Cov )
D2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * time_grid ), Cov = Cov )

fD1 = fData( time_grid, D1 )
```

```
fD2 = fData( time_grid, D2 )

# Computing the covariance function of fD1

C = cov_fun( fD1 )
str( C )

# Computing the cross-covariance function of fD1 and fD2
CC = cov_fun( fD1, fD2 )
str( CC )

# Generating a multivariate functional dataset
L = 3

C1 = exp_cov_function( time_grid, alpha = 0.1, beta = 0.2 )
C2 = exp_cov_function( time_grid, alpha = 0.2, beta = 0.5 )
C3 = exp_cov_function( time_grid, alpha = 0.3, beta = 1 )

centerline = matrix( c( sin( 2 * pi * time_grid ),
                        sqrt( time_grid ),
                        10 * ( time_grid - 0.5 ) * time_grid ),
                     nrow = 3, byrow = TRUE )

D3 = generate_gauss_mfdata( N, L, centerline,
                       correlations = c( 0.5, 0.5, 0.5 ),
                       listCov = list( C1, C2, C3 ) )

# adding names for better readability of BC3's labels
names( D3 ) = c( 'comp1', 'comp2', 'comp3' )
mfD3 = mfData( time_grid, D3 )

D1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * time_grid ),
                              Cov = Cov )
fD1 = fData( time_grid, D1 )

# Computing the block covariance function of mfD3
BC3 = cov_fun( mfD3 )
str( BC3 )

# computing cross-covariance between mfData and fData objects
CC = cov_fun( mfD3, fD1 )
str( CC )
```

EI *Epigraph Index of univariate functional dataset*

Description

This function computes the Epigraphic Index (EI) of elements of a univariate functional dataset.

Usage

```
EI(Data)

## S3 method for class 'fData'
EI(Data)

## Default S3 method:
EI(Data)
```

Arguments

Data	either an fData object or a matrix-like dataset of functional data (e.g. fData$values), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the EI, i.e.:

$$EI(X(t)) = \frac{1}{N}\sum_{i=1}^{N} I(G(X_i(t)) \subset epi(X(t))) = \frac{1}{N}\sum_{i=1}^{N} I(X_i(t) \geq X(t), \ \ \forall t \in I),$$

where $G(X_i(t))$ indicates the graph of $X_i(t)$, $epi(X(t))$ indicates the epigraph of $X(t)$.

Value

The function returns a vector containing the values of EI for each element of the functional dataset provided in Data.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

MEI, HI, MHI, fData

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )
```

```
Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )
fD = fData( grid, Data )

EI( fD )

EI( Data )
```

`exp_cov_function` *Exponential covariance function over a grid*

Description

This function computes the discretisation of an exponential covariance function of the form:

$$C(s,t) = \alpha e^{-\beta|s-t|}$$

over a 1D grid $[t_0, t_1, \ldots, t_{P-1}]$, thus obtaining the $P \times P$ matrix of values:

$$C_{i,j} = C(t_i, t_j) = \alpha e^{-\beta|t_i - t_j|}.$$

Usage

```
exp_cov_function(grid, alpha, beta)
```

Arguments

`grid`	a vector of time points.
`alpha`	the alpha parameter in the exponential covariance formula.
`beta`	the beta parameter in the exponential covariance formula.

See Also

`generate_gauss_fdata`, `generate_gauss_mfdata`

Examples

```
grid = seq( 0, 1, length.out = 5e2 )

alpha = 0.2
beta = 0.3

dev.new()
image( exp_cov_function( grid, alpha, beta ),
       main = 'Exponential covariance function',
       xlab = 'grid', ylab = 'grid')
```

`fbplot` *Functional boxplot of univariate and multivariate functional data*

Description

This function can be used to perform the functional boxplot of univariate or multivariate functional data.

Usage

```
fbplot(Data, Depths = "MBD", Fvalue = 1.5, adjust = FALSE,
  display = TRUE, xlab = NULL, ylab = NULL, main = NULL, ...)

## S3 method for class 'fData'
fbplot(Data, Depths = "MBD", Fvalue = 1.5, adjust = FALSE,
  display = TRUE, xlab = NULL, ylab = NULL, main = NULL, ...)

## S3 method for class 'mfData'
fbplot(Data, Depths = list(def = "MBD", weights = "uniform"),
  Fvalue = 1.5, adjust = FALSE, display = TRUE, xlab = NULL,
  ylab = NULL, main = NULL, ...)
```

Arguments

`Data` the univariate or multivariate functional dataset whose functional boxplot must be determined, in form of `fData` or `mfData` object.

`Depths` either a vector containing the depths for each element of the dataset, or:

- "*univariate case*": a string containing the name of the method you want to use to compute it. The default is `'MBD'`;
- "*multivariate case*": a list with elements `def`, containing the name of the depth notion to be used to compute depths (`BD` or `MBD`), and `weights`, containing the value of parameter `weights` to be passed to the depth function. Default is `list( def = 'MBD', weights = 'uniform' )` .

In both cases the name of the functions to compute depths must be available in the caller's environment.

`Fvalue` the value of the inflation factor F, default is `F = 1.5`.

`adjust` either `FALSE` if you would like the default value for the inflation factor, $F = 1.5$, to be used, or (for now **only in the univariate functional case**) a list specifying the parameters required by the adjustment:

- "`N_trials`": the number of repetitions of the adujustment procedure based on the simulation of a gaussisan population of functional data, each one producing an adjusted value of F, which will lead to the averaged adjusted value $\bar{F}$. Default is 20;

- "trial_size": the number of elements in the gaussian population of functional data that will be simulated at each repetition of the adjustment procedure. Default is 8 * Data$N;
- "TPR": the True Positive Rate of outliers, i.e. the proportion of observations in a dataset without amplitude outliers that have to be considered outliers. Default is 2 * pnorm(4 * qnorm(0.25));
- "F_min": the minimum value of F, defining the left boundary for the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to Data, the optimal value of F. Default is 0.5;
- "F_max": the maximum value of F, defining the right boundary for the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to Data, the optimal value of F. Default is 5;
- "tol": the tolerance to be used in the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to Data, the optimal value of F. Default is 1e-3;
- "maxiter": the maximum number of iterations to solve the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to Data, the optimal value of F. Default is 100;
- "VERBOSE": a parameter controlling the verbosity of the adjustment process;

display — either a logical value indicating wether you want the outliergram to be displayed, or the number of the graphical device where you want the outliergram to be displayed.

xlab — the label to use on the x axis when displaying the functional boxplot.

ylab — the label (or list of labels for the multivariate functional case) to use on the y axis when displaying the functional boxplot.

main — the main title (or list of titles for the multivariate functional case) to be used when displaying the functional boxplot.

... — additional graphical parameters to be used in plotting functions.

Value

Even when used in graphical way to plot the functional boxplot, the function returns a list of three elements: the first, Depths, contains the depths of each element of the functional dataset; the second, Fvalue, is the value of F used to obtain the outliers, and the third, ID_out, contains the vector of indices of dataset's elements flagged as outliers (if any).

Adjustment

In the **univariate functional case**, when the adjustment option is selected, the value of F is optimised for the univariate functional dataset provided with Data.

In practice, a number adjust$N_trials of times a synthetic population (of size adjust$tiral_size with the same covariance (robustly estimated from data) and centerline as fData is simulated without outliers and each time an optimised value F_i is computed so that a given proportion (adjust$TPR) of observations is flagged as outliers. The final value of F for the functional boxplot is determined as an average of $F_1, F_2, \ldots, F_{N_{trials}}$. At each time step the optimisation problem is solved using stats::uniroot (Brent's method.

References

Sun, Y., & Genton, M. G. (2012). Functional boxplots. Journal of Computational and Graphical Statistics.

Sun, Y., & Genton, M. G. (2012). Adjusted functional boxplots for spatio- temporal data visualization and outlier detection. Environmetrics, 23(1), 54-64.

See Also

fData, MBD, BD, mfData, multiMBD, multiBD

Examples

```
# UNIVARIATE FUNCTIONAL BOXPLOT - NO ADJUSTMENT

set.seed(1)

N = 2 * 100 + 1
P = 2e2

grid = seq( 0, 1, length.out = P )

D = 10 * matrix( sin( 2 * pi * grid ), nrow = N, ncol = P, byrow = TRUE )

D = D + rexp(N, rate = 0.05)


# c( 0, 1 : (( N - 1 )/2), -( ( ( N - 1 ) / 2 ) : 1 ) )^4


fD = fData( grid, D )

dev.new()
par( mfrow = c(1,3) )
plot( fD, lwd = 2, main = 'Functional dataset',
      xlab = 'time', ylab = 'values' )

fbplot( fD, main = 'Functional boxplot', xlab = 'time', ylab = 'values', Fvalue = 1.5 )

boxplot(fD$values[,1], ylim = range(fD$values), main = 'Boxplot of functional dataset at t_0 ' )

# UNIVARIATE FUNCTIONAL BOXPLOT - WITH ADJUSTMENT


set.seed( 161803 )

P = 2e2
grid = seq( 0, 1, length.out = P )

N = 1e2
```

```
# Generating a univariate synthetic gaussian dataset
Data = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ),
                             Cov = exp_cov_function( grid,
                                                     alpha = 0.3,
                                                     beta  = 0.4 ) )
fD = fData( grid, Data )

dev.new()
## Not run:
fbplot( fD, adjust = list( N_trials = 10,
                           trial_size = 5 * N,
                           VERBOSE = TRUE ),
                    xlab = 'time', ylab = 'Values',
                    main = 'My adjusted functional boxplot' )

## End(Not run)

# MULTIVARIATE FUNCTIONAL BOXPLOT - NO ADJUSTMENT

set.seed( 1618033 )

P = 1e2
N = 1e2
L = 2

grid = seq( 0, 1, length.out = 1e2 )

C1 = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )
C2 = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Generating a bivariate functional dataset of gaussian data with partially
# correlated components
Data = generate_gauss_mfdata( N, L,
                              centerline = matrix( sin( 2 * pi * grid ),
                                                   nrow = 2, ncol = P,
                                                   byrow = TRUE ),
                              correlations = rep( 0.5, 1 ),
                              listCov = list( C1, C2 ) )

mfD = mfData( grid, Data )

dev.new()
fbplot( mfD, Fvalue = 2.5, xlab = 'time', ylab = list( 'Values 1',
                                                       'Values 2' ),
        main = list( 'First component', 'Second component' ) )
```

---

fData S3 *Class for univariate functional datasets.*

Description

This function implements a constructor for elements of S3 class fData, aimed at implementing a representation of a functional dataset.

Usage

```
fData(grid, values)
```

Arguments

grid	the evenly spaced grid over which the functional observations are measured. It must be a numeric vector of length P.
values	the values of the observations in the functional dataset, prodived in form of a 2D data structure (e.g. matrix or array) having as rows the observations and as columns their measurements over the 1D grid of length P specified in grid.

Details

The functional dataset is represented as a collection of measurement of the observations on an evenly spaced, 1D grid of discrete points (representing, e.g. time), namely, for functional data defined over a grid $[t_0, t_1, \ldots, t_{P-1}]$:

$$f_{i,j} = f_i(t_0 + jh), \quad h = \frac{t_P - t_0}{N}, \quad \forall j = 1, \ldots, P, \quad \forall i = 1, \ldots N.$$

Value

The function returns a S3 object of class fData, containing the following elements:

- "N": the number of elements in the dataset;
- "P": the number of points in the 1D grid over which elements are measured;
- "t0": the starting point of the 1D grid;
- "tP": the ending point of the 1D grid;
- "values": the matrix of measurements of the functional observations on the 1D grid provided with grid.

See Also

generate_gauss_fdata, sub-.fData

Examples

```
# Defining parameters
N = 20
P = 1e2

# One dimensional grid
grid = seq( 0, 1, length.out = P )
```

```
# Generating an exponential covariance function (see related help for more
# information )
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Generating a synthetic dataset with a gaussian distribution and
# required mean and covariance function:
values = generate_gauss_fdata( N,
                               centerline = sin( 2 * pi * grid ),
                               Cov = C )

fD = fData( grid, values )
```

`fDColorPalette` *A set of fancy color to plot functional datasets*

Description

This function can be used to generate a palette of colors useful to plot functional datasets with the `plot` methods.

Usage

```
fDColorPalette(N, hue_range = c(0, 360), alpha = 0.8, ...)
```

Arguments

`N`	number of different colors (ideally, functional observations).
`hue_range`	the range of hues in the HCL scheme.
`alpha`	the alpha channel parameter(s) of the colors (transparency).
`...`	additional parameters to be passed to `scales::hue_pal`

Details

The function, built around `scales::hue_pal`, allows to set up the HCL parameters of the set of colors desired, and besides to set up the alpha channel value.

See Also

`plot.fData`, `plot.mfData`

Examples

```
N = 1e2
angular_grid = seq( 0, 359, length.out = N )

dev.new()
plot( angular_grid, angular_grid,
      col = fDColorPalette( N, hue_range = c( 0, 359 ), alpha = 1 ),
      pch = 16, cex = 3 )
```

`generate_gauss_fdata` *Generation of gaussian univariate functional data*

Description

`generate_gauss_fdata` generates a dataset of univariate functional data with a desired mean and covariance function.

Usage

```
generate_gauss_fdata(N, centerline, Cov = NULL, CholCov = NULL)
```

Arguments

`N`	the number of distinct functional observations to generate.
`centerline`	the centerline of the distribution, represented as a one- dimensional data structure of length P containing the measurement of the centerline on grid points.
`Cov`	the covariance operator (provided in form of a $P \times P$ matrix) that has to be used in the generation of $\epsilon(t)$. At least one argument between `Cov` and `CholCov` should be different from `NULL`.
`CholCov`	the Cholesky factor of the covariance operator (provided in form of a $P \times P$ matrix) that has to be used in the generation of observations from the process $\epsilon(t)$. At least one argument between `Cov` and `CholCov` should be different from `NULL`.

Details

In particular, the following model is considered for the generation of data:

$$X(t) = m(t) + \epsilon(t), \quad t \in I = [a, b]$$

where $m(t)$ is the center and $\epsilon(t)$ is a centered gaussian process with covariance function C_i. That is to say:

$$Cov(\epsilon(s), \epsilon(t)) = C(s, t), \quad \forall s, t \in I$$

All the functions are supposed to be observed on an evenly-spaced, one- dimensional grid of P points: $[a = t_0, t_1, \ldots, t_{P-1} = b] \subset I$.

Value

The function returns a matrix containing the discretized values of the generated observations (in form of an $N \times P$ matrix).

See Also

exp_cov_function, fData, generate_gauss_mfdata

Examples

```
N = 30
P = 1e2

t0 = 0
tP = 1

time_grid = seq( t0, tP, length.out = P )

C = exp_cov_function( time_grid, alpha = 0.1, beta = 0.2 )

CholC = chol( C )

centerline = sin( 2 * pi * time_grid )

invisible(generate_gauss_fdata( N, centerline, Cov = C ))

invisible(generate_gauss_fdata( N, centerline, CholCov = CholC ))
```

---

generate_gauss_mfdata *Generation of gaussian multivariate functional data*

Description

generate_gauss_mfdata generates a dataset of multivariate functional data with a desired mean and covariance function in each dimension and a desired correlation structure among components.

Usage

```
generate_gauss_mfdata(N, L, centerline, correlations, listCov = NULL,
  listCholCov = NULL)
```

Arguments

N	the number of distinct functional observations to generate.
L	the number of components of the multivariate functional data.

`centerline` the centerline of the distribution, represented as a 2-dimensional data structure with L rows (one for each dimension) having the measurements along the grid as columns.

`correlations` is the vector containing the $1/2L(L-1)$ correlation coefficients ρ_{ij} in the model generating data. They have to be provided in the following order:

$$(\rho_{1,2},\dots,\rho_{1,L},\rho_{2,3},\dots,\rho_{2,L},\dots,\rho_{L,L-1}),$$

that is to say, the row-wise, upper triangular part of the correlation matrix without the diagonal.

`listCov` a list containing the L covariance operators (provided in form of a $P\times P$ matrix), one for each component of the multivariate functional random vairable, that have to be used in the generation of the processes $\epsilon_1(t),\dots,\epsilon_L(t)$. At least one argument between `listCov` and `listCholCov` must be different from `NULL`.

`listCholCov` the Cholesky factor of the L covariance operators (in $P\times P$ matrix form), one for each component of the multivariate functional random vairable, that have to be used in the generation of the processes $\epsilon_1(t),\dots,\epsilon_L(t)$. At least one argument between `listCov` and `listCholCov` must be different from `NULL`.

Details

In particular, the following model is considered for the generation of data:

$$X(t) = (m_1(t)+\epsilon_1(t),\dots,m_L(t)+\epsilon_L(t)), \quad t\in I=[a,b]$$

where L is the number of components of the multivariate functional random variable, $m_i(t)$ is the $i-$th component of the center and $\epsilon_i(t)$ is a centered gaussian process with covariance function C_i. That is to say:

$$Cov(\epsilon_i(s),\epsilon_i(t)) = C(s,t), \quad \forall i=1,\dots,L, \quad \forall s,t\in I$$

A correlation structure among $\epsilon_1(t),\dots,\epsilon_L(t)$ is allowed in the following way:

$$Cor(\epsilon_i(t),\epsilon_j(t)) = \rho_{i,j}, \quad \forall i\neq j, \quad \forall t\in I.$$

All the functions are supposed to be observed on an evenly-spaced, one- dimensional grid of P points: $[a=t_0,t_1,\dots,t_{P-1}=b]\subset I$.

Value

The function returns a list of L matrices, one for each component of the multivariate functional random variable, containing the discretized values of the generated observations (in form of $N\times P$ matrices).

See Also

`exp_cov_function`, `mfData`, `generate_gauss_fdata`

Examples

```
N = 30
P = 1e2
L = 3

time_grid = seq( 0, 1, length.out = P )

C1 = exp_cov_function( time_grid, alpha = 0.1, beta = 0.2 )
C2 = exp_cov_function( time_grid, alpha = 0.2, beta = 0.5 )
C3 = exp_cov_function( time_grid, alpha = 0.3, beta = 1 )


centerline = matrix( c( sin( 2 * pi * time_grid ),
                        sqrt( time_grid ),
                        10 * ( time_grid - 0.5 ) * time_grid ),
                     nrow = 3, byrow = TRUE )

generate_gauss_mfdata( N, L, centerline,
                       correlations = c( 0.5, 0.5, 0.5 ),
                       listCov = list( C1, C2, C3 ) )

CholC1 = chol( C1 )
CholC2 = chol( C2 )
CholC3 = chol( C3 )

generate_gauss_mfdata( N, L, centerline,
                       correlations = c( 0.5, 0.5, 0.5 ),
                      listCholCov = list( CholC1, CholC2, CholC3 ) )
```

HI *Hypograph Index of univariate functional dataset*

Description

This function computes the Hypograph Index (HI) of elements of a univariate functional dataset.

Usage

```
HI(Data)

## S3 method for class 'fData'
HI(Data)

## Default S3 method:
HI(Data)
```

Arguments

Data	either an fData object or a matrix-like dataset of functional data (e.g. fData$values), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the HI, i.e.:

$$HI(X(t)) = \frac{1}{N}\sum_{i=1}^{N} I(G(X_i(t)) \subset hyp(X(t))) = \frac{1}{N}\sum_{i=1}^{N} I(X_i(t) \leq X(t), \ \ \forall t \in I),$$

where $G(X_i(t))$ indicates the graph of $X_i(t)$, $hyp(X(t))$ indicates the hypograph of $X_i(t)$.

Value

The function returns a vector containing the values of HI for each element of the functional dataset provided in Data.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

MHI, EI, MEI, fData

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )
fD = fData( grid, Data )

HI( fD )

HI( Data )
```

HRD *Half-Region Depth for univariate functional data*

Description

This function computes the Half-Region Depth (HRD) of elements of a univariate functional dataset.

Usage

```
HRD(Data)

## S3 method for class 'fData'
HRD(Data)

## Default S3 method:
HRD(Data)
```

Arguments

`Data`	either an `fData` object or a matrix-like dataset of functional data (e.g. `fData$values`), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the HRD of its elements, i.e.:

$$HRD(X(t)) = \min(EI(X(t)), HI(X(t))),$$

where $EI(X(t))$ indicates the Epigraph Index (EI) of $X(t)$ with respect to the dataset, and $HI(X(t))$ indicates the Hypograph Index of $X(t)$ with respect to the dataset.

Value

The function returns a vector containing the values of HRD for each element of the functional dataset provided in `Data`.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

`MHRD`, `EI`, `HI`, `fData`

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )

fD = fData( grid, Data )

HRD( fD )

HRD( Data )
```

`maxima` *Maxima of a univariate functional dataset*

Description

This function computes the maximum value of each element of a univariate functional dataset, optionally returing also the value of the grid where they are fulfilled.

Usage

```
maxima(fData, ..., which = FALSE)
```

Arguments

`fData`	the functional dataset containing elements whose maxima have to be computed, in form of `fData` object.
`...`	additional parameters.
`which`	logical flag specifying whether the grid values where maxima are fulfilled have to be returned too.

Value

If `which = FALSE`, the function returns a vector containing the maxima for each element of the functional dataset; if `which = TRUE`, the function returns a `data.frame` whose field `value` contains the values of maxima, and `grid` contains the grid points where maxima are reached.

See Also

`minima`

Examples

```
P = 1e3

grid = seq( 0, 1, length.out = P )

Data = matrix( c( 1 * grid,
                  2 *  grid,
                  3 * ( 0.5 - abs( grid - 0.5 ) ) ),
               nrow = 3, ncol = P, byrow = TRUE )

fD = fData( grid, Data )

maxima( fD, which = TRUE )
```

`max_ordered` *Maximum order relation between univariate functional data*

Description

This function implements an order relation between univariate functional data based on the maximum relation, that is to say a pre-order relation obtained by comparing the maxima of two different functional data.

Usage

```
max_ordered(fData, gData)
```

Arguments

`fData`	the first univariate functional dataset containing elements to be compared, in form of `fData` object.
`gData`	the second univariate functional dataset containing elements to be compared, in form of `fData` object.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$ and another functional dataset $Y_1(t)$, $Y_2(t), \ldots, Y_M(t)$ defined over the same compact interval $I = [a, b]$, the function computes the maxima in both the datasets, and checks whether the first ones are lower or equal than the second ones.

By default the function tries to compare each $X_i(t)$ with the corresponding $Y_i(t)$, thus assuming $N = M$, but when either $N = 1$ or $M = 1$, the comparison is carried out cycling over the dataset with fewer elements. In all the other cases ($N \neq M$, and either $N \neq 1$ or $M \neq 1$) the function stops.

Value

The function returns a logical vector of length $\max(N, M)$ containing the value of the predicate for all the corresponding elements.

References

Valencia, D., Romo, J. and Lillo, R. (2015). A Kendall correlation coefficient for functional dependence, *Universidad Carlos III de Madrid technical report*, http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133228

See Also

maxima, minima, fData, area_ordered

Examples

```
P = 1e2

grid = seq( 0, 1, length.out = P )

Data_1 = matrix( c( 1 * grid,
                    2 *  grid ),
                 nrow = 2, ncol = P, byrow = TRUE )

Data_2 = matrix( 3 * ( 0.5 - abs( grid - 0.5 ) ),
                 nrow = 1, byrow = TRUE )

Data_3 = rbind( Data_1, Data_1 )


fD_1 = fData( grid, Data_1 )
fD_2 = fData( grid, Data_2 )
fD_3 = fData( grid, Data_3 )

max_ordered( fD_1, fD_2 )

max_ordered( fD_2, fD_3 )
```

MBD *Modified Band Depth for univariate functional data*

Description

This function computes the Modified Band Depth (MBD) of elements of a functional dataset.

Usage

```
MBD(Data, manage_ties = FALSE)

## S3 method for class 'fData'
MBD(Data, manage_ties = FALSE)

## Default S3 method:
MBD(Data, manage_ties = FALSE)
```

Arguments

`Data`	either a `fData` object or a matrix-like dataset of functional data (e.g. `fData$values`), with observations as rows and measurements over grid points as columns.
`manage_ties`	a logical flag specifying whether a check for ties and relative treatment must be carried out or not (default is `FALSE`).

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the sample MBD of each element with respect to the other elements of the dataset, i.e.:

$$MBD(X(t)) = \binom{N}{2}^{-1} \sum_{1 \le i_1 < i_2 \le N} \tilde{\lambda}\big(t : \min(X_{i_1}(t), X_{i_2}(t)) \le X(t) \le \max(X_{i_1}(t), X_{i_2}(t))\big),$$

where $\tilde{\lambda}(\cdot)$ is the normalised Lebesgue measure over $I = [a, b]$, that is $\tilde{\lambda(A)} = \lambda(A)/(b - a)$.

See the References section for more details.

Value

The function returns a vector containing the values of MBD for the given dataset.

References

Lopez-Pintado, S. and Romo, J. (2009). On the Concept of Depth for Functional Data, *Journal of the American Statistical Association*, 104, 718-734.

Lopez-Pintado, S. and Romo. J. (2007). Depth-based inference for functional data, *Computational Statistics & Data Analysis* 51, 4957-4968.

See Also

`BD`, `MBD_relative`, `BD_relative`, `fData`

Examples

```
grid = seq( 0, 1, length.out = 1e2 )


D = matrix( c( 1 + sin( 2 * pi * grid ),
               0 + sin( 4 * pi * grid ),
               1 - sin( pi * ( grid - 0.2 ) ),
               0.1 + cos( 2 * pi * grid ),
               0.5 + sin( 3 * pi + grid ),
               -2 + sin( pi * grid ) ),
            nrow = 6, ncol = length( grid ), byrow = TRUE )

fD = fData( grid, D )

MBD( fD )

MBD( D )
```

`MBD_relative` *Relative Modified Band Depth of functions in a univariate functional dataset*

Description

This function computes Modified Band Depth (BD) of elements of a univariate functional dataset with respect to another univariate functional dataset.

Usage

```
MBD_relative(Data_target, Data_reference)

## S3 method for class 'fData'
MBD_relative(Data_target, Data_reference)

## Default S3 method:
MBD_relative(Data_target, Data_reference)
```

Arguments

`Data_target`	is the univariate functional dataset, provided either as an `fData` object or in matrix form (N observations as rows and P measurements as columns), whose MBD have to be computed with respect to the reference dataset.
`Data_reference`	is the dataset, provided either as an `fData` object or in matrix form (N observations as rows and P measurements as columns), containing the reference univariate functional data that must be used to compute the MBD of elements in `Data_target`. If `Data_target` is `fData`, it must be of class `fData`.

Details

Given a univariate functional dataset of elements $X_1(t), X_2(t), \ldots, X_N(t)$, and another univariate functional dataset of elements $Y_1(t), Y_2(t) \ldots, Y_M(t)$, defined over the same compact interval $I = [a, b]$, this function computes the MBD of elements of the former with respect to elements of the latter, i.e.:

$$MBD(X_i(t)) = \binom{M}{2}^{-1} \sum_{1 \leq i_1 < i_2 \leq M} \tilde{\lambda}\big(t : \min(Y_{i_1}(t), Y_{i_2}(t)) \leq X_i(t) \leq \max(Y_{i_1}(t), Y_{i_2}(t))\big),$$

$\forall i = 1, \ldots, N$, where $\tilde{\lambda}(\cdot)$ is the normalised Lebesgue measure over $I = [a, b]$, that is $\tilde{\lambda(A)} = \lambda(A)/(b - a)$.

Value

The function returns a vector containing the MBD of elements in `Data_target` with respect to elements in `Data_reference`.

See Also

MBD, BD, BD_relative, fData

Examples

```
grid = seq( 0, 1, length.out = 1e2 )

Data_ref = matrix( c( 0  + sin( 2 * pi * grid ),
                      1  + sin( 2 * pi * grid ),
                      -1 + sin( 2 * pi * grid ) ),
                   nrow = 3, ncol = length( grid ), byrow = TRUE )

Data_test_1 = matrix( c( 0.6 + sin( 2 * pi * grid ) ),
                      nrow = 1, ncol = length( grid ), byrow = TRUE )

Data_test_2 = matrix( c( 0.6 + sin( 2 * pi * grid ) ),
                      nrow = length( grid ), ncol = 1, byrow = TRUE )

Data_test_3 = 0.6 + sin( 2 * pi * grid )

Data_test_4 = array( 0.6 + sin( 2 * pi * grid ), dim = length( grid ) )

Data_test_5 = array( 0.6 + sin( 2 * pi * grid ), dim = c( 1, length( grid ) ) )

Data_test_6 = array( 0.6 + sin( 2 * pi * grid ), dim = c( length( grid ), 1 ) )

Data_test_7 = matrix( c( 0.5  + sin( 2 * pi * grid ),
                         -0.5 + sin( 2 * pi * grid ),
                         1.1 + sin( 2 * pi * grid ) ),
                      nrow = 3, ncol = length( grid ), byrow = TRUE )
```

```
fD_ref = fData( grid, Data_ref )
fD_test_1 = fData( grid, Data_test_1 )
fD_test_2 = fData( grid, Data_test_2 )
fD_test_3 = fData( grid, Data_test_3 )
fD_test_4 = fData( grid, Data_test_4 )
fD_test_5 = fData( grid, Data_test_5 )
fD_test_6 = fData( grid, Data_test_6 )
fD_test_7 = fData( grid, Data_test_7 )

MBD_relative( fD_test_1, fD_ref )
MBD_relative( Data_test_1, Data_ref )

MBD_relative( fD_test_2, fD_ref )
MBD_relative( Data_test_2, Data_ref )

MBD_relative( fD_test_3, fD_ref )
MBD_relative( Data_test_3, Data_ref )

MBD_relative( fD_test_4, fD_ref )
MBD_relative( Data_test_4, Data_ref )

MBD_relative( fD_test_5, fD_ref )
MBD_relative( Data_test_5, Data_ref )

MBD_relative( fD_test_6, fD_ref )
MBD_relative( Data_test_6, Data_ref )

MBD_relative( fD_test_7, fD_ref )
MBD_relative( Data_test_7, Data_ref )
```

`mean.fData` *Cross-sectional mean of of a fData object.*

Description

This `S3` method implements the **cross-sectional** mean of a univariate functional dataset stored in a `fData` object, i.e. the mean computed point-by-point along the grid over which the dataset is defined.

Usage

```
## S3 method for class 'fData'
mean(x, ...)
```

Arguments

`x`	the univariate functional dataset whose cross-sectional mean must be computed, in form of `fData` object.

... possible additional parameters. This argument is kept for compatibility with the S3 definition of mean, but it is not actually used.

Value

The function returns a fData object with one observation defined on the same grid as the argument x's representing the desired cross-sectional mean.

See Also

fData

Examples

```
N = 1e2
P = 1e2
grid = seq( 0, 1, length.out = P )

# Generating a gaussian functional sample with desired mean
target_mean = sin( 2 * pi * grid )
C = exp_cov_function( grid, alpha = 0.2, beta = 0.2 )
fD = fData( grid, generate_gauss_fdata( N,
                                        centerline = target_mean,
                                         Cov = C ) )

# Graphical representation of the mean
plot( fD )
plot( mean( fD ), col = 'black', lwd = 2, lty = 2, add = TRUE )
```

---

mean.mfData *Cross-sectional mean of of a mfData object.*

Description

This S3 method implements the **cross-sectional** mean of a multivariate functional dataset stored in a mfData object, i.e. the mean computed point-by-point along the grid over which the dataset is defined.

Usage

```
## S3 method for class 'mfData'
mean(x, ...)
```

Arguments

x	the multivariate functional dataset whose cross-sectional mean must be computed, in form of mfData object.
...	possible additional parameters. This argument is kept for compatibility with the S3 definition of mean, but it is not actually used.

Value

The function returns a mfData object with one observation defined on the same grid as the argument x's representing the desired cross-sectional mean.

See Also

mfData

Examples

```
N = 1e2
L = 3
P = 1e2
grid = seq( 0, 1, length.out = P )

# Generating a gaussian functional sample with desired mean
target_mean = sin( 2 * pi * grid )
C = exp_cov_function( grid, alpha = 0.2, beta = 0.2 )
# Independent components
correlations = c( 0, 0, 0 )
mfD = mfData( grid,
              generate_gauss_mfdata( N, L,
                                     correlations = correlations,
                                     centerline = matrix( target_mean,
                                                          nrow = 3,
                                                          ncol = P,
                                                          byrow = TRUE ),
                                     listCov = list( C, C, C ) )
)

# Graphical representation of the mean
par( mfrow = c( 1, 3 ) )

for( iL in 1 : L )
{
  plot( mfD$fDList[[ 1 ]] )
  plot( mean( mfD )$fDList[[ 1 ]], col = 'black',
        lwd = 2, lty = 2, add = TRUE )
}
```

median_fData *Median of a univariate functional dataset*

Description

This method computes the sample median of a univariate functional dataset based on a definition of depth for univariate functional data.

Usage

```
median_fData(fData, type = "MBD", ...)
```

Arguments

fData	the univariate functional dataset whose median is required, in form of fData object.
type	a string specifying the name of the function defining the depth for univariate data to be used. It must be a valid name of a function defined in the current environment, default is MBD.
...	additional parameters to be used in the function specified by argument type.

Details

Provided a definition of functional depth for univariate data, the corresponding median (i.e. the deepest element of the sample) is returned as the desired median. This method does **not** coincide with the computation of the cross-sectional median of the sample of the point-by-point measurements on the grid. Hence, the sample median is a member of the dataset provided.

Value

The function returns a fData object containing the desired sample median.

See Also

fData, mean.fData, median_mfData

Examples

```
N = 1e2
P = 1e2
grid = seq( 0, 1, length.out = P )

# Generating a gaussian functional sample with desired mean
# Being the distribution symmetric, the sample mean and median are coincident
target_median = sin( 2 * pi * grid )
C = exp_cov_function( grid, alpha = 0.2, beta = 0.2 )
fD = fData( grid, generate_gauss_fdata( N,
```

```
                                        centerline = target_median,
                                         Cov = C ) )

# Graphical representation of the mean
plot( fD )
plot( median_fData( fD ), col = 'black', lwd = 2, lty = 2, add = TRUE )
```

`median_mfData` *Median of a multivariate functional dataset*

Description

This method computes the sample median of a multivariate functional dataset based on a definition of depth for multivariate functional data.

Usage

```
median_mfData(mfData, type = "multiMBD", ...)
```

Arguments

`mfData`	the multivariate functional dataset whose median is required, in form of `mfData` object.
`type`	a string specifying the name of the function defining the depth for multivariate data to be used. It must be a valid name of a function defined in the current environment, default is `multiMBD`.
`...`	additional parameters to be used in the function specified by argument `type`.

Details

Provided a definition of functional depth for multivariate data, the corresponding median (i.e. the deepest element of the sample) is returned as the desired median. This method does **not** coincide with the computation of the cross-sectional median of the sample of the point-by-point measurements on the grid. Hence, the sample median is a member of the dataset provided.

Value

The function returns a `mfData` object containing the desired sample median.

See Also

`mfData`, `mean.mfData`, `median_fData`

Examples

```
N = 1e2
L = 3
P = 1e2
grid = seq( 0, 1, length.out = P )

# Generating a gaussian functional sample with desired mean
# Being the distribution symmetric, the sample mean and median are coincident
target_median = sin( 2 * pi * grid )
C = exp_cov_function( grid, alpha = 0.2, beta = 0.2 )

# Strongly dependent components
correlations = c( 0.9, 0.9, 0.9 )
mfD = mfData( grid,
              generate_gauss_mfdata( N, L,
                                     correlations = correlations,
                                     centerline = matrix( target_median,
                                                          nrow = 3,
                                                          ncol = P,
                                                          byrow = TRUE ),
                                     listCov = list( C, C, C ) )
)

med_mfD = median_mfData( mfD, type = 'multiMBD', weights = 'uniform' )

# Graphical representation of the mean
par( mfrow = c( 1, 3 ) )

for( iL in 1 : L )
{
  plot( mfD$fDList[[ 1 ]] )
  plot( med_mfD$fDList[[ 1 ]], col = 'black',
        lwd = 2, lty = 2, add = TRUE )
}
```

MEI *Modified Epigraph Index of univariate functional dataset*

Description

This function computes the Modified Epigraphic Index (MEI) of elements of a univariate functional dataset.

Usage

```
MEI(Data)

## S3 method for class 'fData'
```

```
MEI(Data)

## Default S3 method:
MEI(Data)
```

Arguments

`Data`	either an `fData` object or a matrix-like dataset of functional data (e.g. `fData$values`), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the MEI, i.e.:

$$MEI(X(t)) = \frac{1}{N} \sum_{i=1}^{N} \tilde{\lambda}(X(t) \leq X_i(t)),$$

where $\tilde{\lambda}(\cdot)$ is the normalised Lebesgue measure over $I = [a, b]$, that is $\tilde{\lambda(A)} = \lambda(A)/(b - a)$.

Value

The function returns a vector containing the values of MEI for each element of the functional dataset provided in `Data`.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

`EI`, `MHI`, `HI`, `fData`

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )
```

```
fD = fData( grid, Data )

MEI( fD )

MEI( Data )
```

`mfData` `S3` *class for multivariate functional datasets*

Description

This function implements a constructor for elements of `S3` class `mfData`, aimed at implementing a representation of a multivariate functional dataset.

Usage

```
mfData(grid, Data_list)
```

Arguments

`grid`	the (evenly spaced) grid over which the functional dataset is defined.
`Data_list`	a `list` containing the `L` components of the multivariate functional dataset, defined as 2D data structures (e.g. matrix or array) having as rows the `N` observations and as columns the `P` measurements on the grid provided by `grid`.

Details

The functional dataset is represented as a collection of `L` components, each one an object of class `fData`. Each component must contain elements defined on the same grid as the others, and must contain the same number of elements (`N`).

Value

The function returns a `S3` object of class `mfData`, containing the following elements:

- "`N`": the number of elements in the dataset;
- "`L`": the number of components of the functional dataset;
- "`P`": the number of points in the 1D grid over which elements are measured;
- "`t0`": the starting point of the 1D grid;
- "`tP`": the ending point of the 1D grid;
- "`fDList`": the list of `fData` objects representing the `L` components as corresponding unviariate functional datasets.

See Also

fData, generate_gauss_fdata, generate_gauss_mfdata

Examples

```
# Defining parameters
N = 1e2

P = 1e3

t0 = 0
t1 = 1

# Defining the measurement grid
grid = seq( t0, t1, length.out = P )

# Generating an exponential covariance matrix to be used in the simulation of
# the functional datasets (see the related help for details)
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating the measurements of two univariate functional datasets with
# required center and covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )

# Building the mfData object
mfData( grid, list( Data_1, Data_2 ) )
```

---

mfD_healthy *ECG trace of healthy subjects*

Description

A dataset containing the 8-Lead ECG traces of 50 healthy subjects. They can be used to compare the signals of pathological subjects stored in mfD_LBBB and mfD_RBBB objects.

Usage

```
mfD_healthy
```

Format

A mfData object.

Details

The 8 leads are, in order, V1, V2, V3, V4, V5, D1 and D2. The signals have been registered and smoothed over an evenly spaced grid of 1024 time points at 1kHz.

mfD_LBBB — *ECG trace of subjects suffering from Left-Bundle-Branch-Block (LBBB)*

Description

A dataset containing the 8-Lead ECG traces of 50 subjects suffering from Left-Bundle-Branch-Block (LBBB), a cardiac pathology affecting the conduction process and resulting in some peculiar distortions of the ECG.

Usage

```
mfD_LBBB
```

Format

A mfData object.

Details

The 8 leads are, in order, V1, V2, V3, V4, V5, D1 and D2. The signals have been registered and smoothed over an evenly spaced grid of 1024 time points at 1kHz.

MHI — *Modified Hypograph Index of univariate functional dataset*

Description

This function computes the Modified Hypograph Index (MEI) of elements of a univariate functional dataset.

Usage

```
MHI(Data)

## S3 method for class 'fData'
MHI(Data)

## Default S3 method:
MHI(Data)
```

Arguments

Data	either an fData object or a matrix-like dataset of functional data (e.g. fData$values), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the MHI, i.e.:

$$MHI(X(t)) = \frac{1}{N} \sum_{i=1}^{N} \tilde{\lambda}(X(t) \geq X_i(t)),$$

where $\tilde{\lambda}(\cdot)$ is the normalised Lebesgue measure over $I = [a, b]$, that is $\tilde{\lambda(A)} = \lambda(A)/(b - a)$.

Value

The function returns a vector containing the values of MHI for each element of the functional dataset provided in `Data`.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

HI, MEI, EI, fData

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )
fD = fData( grid, Data )

MHI( fD )

MHI( Data )
```

MHRD *Modified Half-Region Depth for univariate functional data*

Description

This function computes the Modified Half-Region Depth (MHRD) of elements of a univariate functional dataset.

Usage

```
MHRD(Data)

## S3 method for class 'fData'
MHRD(Data)

## Default S3 method:
MHRD(Data)
```

Arguments

Data	either an fData object or a matrix-like dataset of functional data (e.g. fData$values), with observations as rows and measurements over grid points as columns.

Details

Given a univariate functional dataset, $X_1(t), X_2(t), \ldots, X_N(t)$, defined over a compact interval $I = [a, b]$, this function computes the MHRD of its elements, i.e.:

$$MHRD(X(t)) = \min(MEI(X(t)), MHI(X(t))),$$

where $MEI(X(t))$ indicates the Modified Epigraph Index (MEI) of $X(t)$ with respect to the dataset, and $MHI(X(t))$ indicates the Modified Hypograph Index of $X(t)$ with respect to the dataset.

Value

The function returns a vector containing the values of MHRD for each element of the functional dataset provided in Data.

References

Lopez-Pintado, S. and Romo, J. (2012). A half-region depth for functional data, *Computational Statistics and Data Analysis*, 55, 1679-1695.

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

HRD, MEI, MHI

Examples

```
N = 20
P = 1e2

grid = seq( 0, 1, length.out = P )

C = exp_cov_function( grid, alpha = 0.2, beta = 0.3 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 2 * pi * grid ),
                             C )
fD = fData( grid, Data )

MHRD( fD )

MHRD( Data )
```

---

minima *Minima of a univariate functional dataset*

Description

This function computes computes the minimum value of each element of a univariate functional dataset, optionally returing also the value of the grid where they are fulfilled.

Usage

```
minima(fData, ..., which = FALSE)
```

Arguments

fData	the functional dataset containing elements whose minima have to be computed, in form of fData object.
...	additional parameters.
which	logical flag specifying whether the grid values where minima are fulfilled have to be returned too.

Value

If which = FALSE, the function returns a vector containing the minima for each element of the functional dataset; if which = TRUE, the function returns a data.frame whose field value contains the values of minima, and grid contains the grid points where minima are reached.

See Also

maxima

Examples

```
P = 1e3

grid = seq( 0, 1, length.out = P )

Data = matrix( c( 1 * grid,
                  2 *  grid,
                  3 * ( 0.5 - abs( grid - 0.5 ) ) ),
               nrow = 3, ncol = P, byrow = TRUE )

fD = fData( grid, Data )

minima( fD, which = TRUE )
```

---

multiMBD *(Modified) Band Depth for multivariate functional data*

Description

These functions compute the Band Depth (BD) and Modified Band Depth (MBD) of elements of a multivariate functional dataset.

Usage

```
multiMBD(Data, weights = "uniform", manage_ties = FALSE)

## S3 method for class 'mfData'
multiMBD(Data, weights = "uniform", manage_ties = FALSE)

## Default S3 method:
multiMBD(Data, weights = "uniform", manage_ties = FALSE)

multiBD(Data, weights = "uniform")

## S3 method for class 'mfData'
multiBD(Data, weights = "uniform")

## Default S3 method:
multiBD(Data, weights = "uniform")
```

Arguments

`Data`	specifies the the multivariate functional dataset. It is either an object of class `mfData` or a list of 2-dimensional matrices having as rows the elements of that component and as columns the measurements of the functional data over the grid.
`weights`	either a set of weights (of the same length of `Data`) or the string `"uniform"` specifying that a set of uniform weights (of value $1/L$, where L is the number of dimensions of the functional dataset and thus the length of `Data`) is to be used.
`manage_ties`	a logical flag specifying whether the check for ties and the relative treatment is to be carried out while computing the MBDs in each dimension. It is directly passed to `MBD`.

Details

Given a multivariate functional dataset composed of N elements with L components each, $\mathbf{X_1} = (X_1^1(t), X_1^2(t), \ldots, X_1^L(t))$, and a set of L non-negative weights,

$$w_1, w_2, \ldots, w_L, \qquad \sum_{i=1}^{L} w_i = 1,$$

these functions compute the BD and MBD of each element of the functional dataset, namely:

$$BD(\mathbf{X_j}) = \sum_{i=1}^{L} w_i BD(X_j^i), \quad \forall j = 1, \ldots N.$$

$$MBD(\mathbf{X_j}) = \sum_{i=1}^{L} w_i MBD(X_j^i), \quad \forall j = 1, \ldots N.$$

Value

The function returns a vector containing the depths of each element of the multivariate functional dataset.

References

Ieva, F. and Paganoni, A. M. (2013). Depth measures for multivariate functional data, *Communications in Statistics: Theory and Methods*, 41, 1265-1276.

Tarabelloni, N., Ieva, F., Biasi, R. and Paganoni, A. M. (2015). Use of Depth Measure for Multivariate Functional Data in Disease Prediction: An Application to Electrocardiograph Signals, *International Journal of Biostatistics*, 11.2, 189-201.

See Also

`MBD`, `BD`, `toListOfValues`, `mfData`

Examples

```
N = 20
P = 1e3

grid = seq( 0, 10, length.out = P )

# Generating an exponential covariance function to be used to simulate gaussian
# functional data
Cov = exp_cov_function( grid, alpha = 0.2, beta = 0.8 )

# First component of the multivariate guassian functional dataset
Data_1 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )

# First component of the multivariate guassian functional dataset
Data_2 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )

mfD = mfData( grid, list( Data_1, Data_2 ) )

multiBD( mfD, weights = 'uniform' )
multiMBD( mfD, weights = 'uniform', manage_ties = TRUE )

multiBD( mfD, weights = c( 1/3, 2/3 ))
multiMBD( mfD, weights = c( 1/3, 2/3 ), manage_ties = FALSE )

multiBD( list( Data_1, Data_2 ), weights = 'uniform')
multiMBD( list( Data_1, Data_2 ), weights = 'uniform', manage_ties = TRUE )

multiBD( list( Data_1, Data_2 ), weights = c( 1/3, 2/3 ))
multiMBD( list( Data_1, Data_2 ), weights = c( 1/3, 2/3 ), manage_ties = FALSE )
```

`multiMEI` *Modified Epigraph Index for multivariate functional data*

Description

These functions compute the Modified Epigraph Index of elements of a multivariate functional dataset.

Usage

```
multiMEI(Data, weights = "uniform")

## S3 method for class 'mfData'
multiMEI(Data, weights = "uniform")

## Default S3 method:
multiMEI(Data, weights = "uniform")
```

Arguments

`Data`	specifies the the multivariate functional dataset. It is either an object of class `mfData` or a list of 2-dimensional matrices having as rows the elements of that component and as columns the measurements of the functional data over the grid.
`weights`	either a set of weights (of the same length of `Data`) or the string `"uniform"` specifying that a set of uniform weights (of value $1/L$, where L is the number of dimensions of the functional dataset and thus the length of `Data`) is to be used.

Details

Given a multivariate functional dataset composed of N elements with L components each, $\mathbf{X_1} = (X_1^1(t), X_1^2(t), \ldots, X_1^L(t))$, and a set of L non-negative weights,

$$w_1, w_2, \ldots, w_L, \qquad \sum_{i=1}^{L} w_i = 1,$$

these functions compute the MEI of each element of the functional dataset, namely:

$$MEI(\mathbf{X_j}) = \sum_{i=1}^{L} w_i MEI(X_j^i), \quad \forall j = 1, \ldots N.$$

Value

The function returns a vector containing the values of MEI of each element of the multivariate functional dataset.

See Also

`mfData`, `MEI`, `MHI`, `multiMHI`

Examples

```
N = 20
P = 1e3

grid = seq( 0, 10, length.out = P )

# Generating an exponential covariance function to be used to simulate gaussian
# functional data
Cov = exp_cov_function( grid, alpha = 0.2, beta = 0.8 )

# First component of the multivariate guassian functional dataset
Data_1 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )

# First component of the multivariate guassian functional dataset
Data_2 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )
```

```
mfD = mfData( grid, list( Data_1, Data_2 ) )

# Uniform weights
multiMEI( mfD, weights = 'uniform' )

# Non-uniform, custom weights
multiMEI( mfD, weights = c(2/3, 1/3) )
```

`multiMHI` *Modified Hypograph Index for multivariate functional data*

Description

These functions compute the Modified Hypograph Index of elements of a multivariate functional dataset.

Usage

```
multiMHI(Data, weights = "uniform")

## S3 method for class 'mfData'
multiMHI(Data, weights = "uniform")

## Default S3 method:
multiMHI(Data, weights = "uniform")
```

Arguments

`Data`	specifies the the multivariate functional dataset. It is either an object of class `mfData` or a list of 2-dimensional matrices having as rows the elements of that component and as columns the measurements of the functional data over the grid.
`weights`	either a set of weights (of the same length of `Data`) or the string `"uniform"` specifying that a set of uniform weights (of value $1/L$, where L is the number of dimensions of the functional dataset and thus the length of `Data`) is to be used.

Details

Given a multivariate functional dataset composed of N elements with L components each, $\mathbf{X_1} = (X_1^1(t), X_1^2(t), \ldots, X_1^L(t))$, and a set of L non-negative weights,

$$w_1, w_2, \ldots, w_L, \qquad \sum_{i=1}^{L} w_i = 1,$$

these functions compute the MHI of each element of the functional dataset, namely:

$$MHI(\mathbf{X_j}) = \sum_{i=1}^{L} w_i MHI(X_j^i), \quad \forall j = 1, \dots N.$$

Value

The function returns a vector containing the values of MHI of each element of the multivariate functional dataset.

See Also

mfData, MHI, MEI, multiMEI

Examples

```
N = 20
P = 1e3

grid = seq( 0, 10, length.out = P )

# Generating an exponential covariance function to be used to simulate gaussian
# functional data
Cov = exp_cov_function( grid, alpha = 0.2, beta = 0.8 )

# First component of the multivariate guassian functional dataset
Data_1 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )

# First component of the multivariate guassian functional dataset
Data_2 = generate_gauss_fdata( N, centerline = rep( 0, P ), Cov = Cov )

mfD = mfData( grid, list( Data_1, Data_2 ) )

# Uniform weights
multiMHI( mfD, weights = 'uniform' )

# Non-uniform, custom weights
multiMHI( mfD, weights = c(2/3, 1/3) )
```

---

multivariate_outliergram

Outliergram for multivariate functional datasets

Description

This function performs the outliergram of a multivariate functional dataset.

Usage

```
multivariate_outliergram(mfData, MBD_data = NULL, MEI_data = NULL,
  weights = "uniform", p_check = 0.05, Fvalue = 1.5, shift = TRUE,
  display = TRUE, xlab = NULL, ylab = NULL, main = NULL)
```

Arguments

mfData	the multivariate functional dataset whose outliergram has to be determined;
MBD_data	a vector containing the MBD for each element of the dataset; If missing, MBDs are computed with the specified choice of weights;
MEI_data	a vector containing the MEI for each element of the dataset. If not not provided, MEIs are computed;
weights	the weights choice to be used to compute multivariate MBDs and MEIs;
p_check	percentage of observations with either low or high MEI to be checked for outliers in the secondary step (shift towards the center of the dataset).
Fvalue	the F value to be used in the procedure that finds the shape outliers by looking at the lower parabolic limit in the outliergram. Default is 1.5;
shift	whether to apply the shifting algorithm to properly manage observations having low or high MEI. Default is TRUE.
display	either a logical value indicating wether you want the outliergram to be displayed, or the number of the graphical device where you want the outliergram to be displayed;
xlab	the label to use on the x axis in the outliergram plot;
ylab	the label to use on the x axis in the outliergram plot;
main	the title to use in the outliergram;

Details

The method applies the extension of the univariate outliergram to the case of multivariate functional datasets. Differently from the function for the univariate case, only the outliergram plot is displayed.

Adjustment

Differently from the case of univariate functional data, in this case the function does not apply an automatic tuning of the F parameter, since the related procedure would become computationally too heavy for general datasets. If a good value of F is sought, it is recommended to run several trials of the outliergram and manually select the best value.

References

Ieva, F. & Paganoni, A.M. Stat Papers (2017). https://doi.org/10.1007/s00362-017-0953-1.

See Also

outliergram, mfData, MBD, MEI

Examples

```
N = 2e2
P = 1e2

t0 = 0
t1 = 1

set.seed(1)

# Defining the measurement grid
grid = seq( t0, t1, length.out = P )

# Generating an exponential covariance matrix to be used in the simulation of
# the functional datasets (see the related help for details)
C = exp_cov_function( grid, alpha = 0.3, beta = 0.2)

# Simulating the measurements of two univariate functional datasets with
# required center and covariance function
f1 = function(x) x * ( 1 - x )
f2 = function(x) x^3
Data = generate_gauss_mfdata( N, L = 2,
                              centerline = matrix(c(sin(2 * pi * grid),
                                                    cos(2 * pi * grid)), nrow=2, byrow=TRUE),
                              listCov = list(C, C), correlations = 0.1 )

# Building the mfData object
mfD = mfData( grid, Data )


dev.new()
out = multivariate_outliergram(mfD, Fvalue = 2., shift=TRUE)
col_non_outlying = scales::hue_pal( h = c( 180, 270 ),
                                    l = 60 )( N - length( out$ID_outliers ) )
col_non_outlying = set_alpha( col_non_outlying, 0.5 )
col_outlying = scales::hue_pal( h = c( - 90, 180  ),
                                c = 150 )( length( out$ID_outliers ) )
colors = rep('black', N)
colors[out$ID_outliers] = col_outlying
colors[colors == 'black'] = col_non_outlying

lwd = rep(1, N)
lwd[out$ID_outliers] = 2

dev.new()
plot(mfD, col=colors, lwd=lwd)
```

`outliergram` *Outliergram for univariate functional datasets*

Description

This function performs the outliergram of a univariate functional dataset, possibly with an adjustment of the true positive rate of outliers discovered under assumption of gaussianity.

Usage

```
outliergram(fData, MBD_data = NULL, MEI_data = NULL, p_check = 0.05,
  Fvalue = 1.5, adjust = FALSE, display = TRUE, xlab = NULL,
  ylab = NULL, main = NULL, ...)
```

Arguments

`fData`	the univariate functional dataset whose outliergram has to be determined.
`MBD_data`	a vector containing the MBD for each element of the dataset. If missing, MBDs are computed.
`MEI_data`	a vector containing the MEI for each element of the dataset. If not not provided, MEIs are computed.
`p_check`	percentage of observations with either low or high MEI to be checked for outliers in the secondary step (shift towards the center of the dataset).
`Fvalue`	the F value to be used in the procedure that finds the shape outliers by looking at the lower parabolic limit in the outliergram. Default is `1.5`. You can also leave the default value and, by providing the parameter `adjust`, specify that you want `Fvalue` to be adjusted for the dataset provided in `fData`.
`adjust`	either `FALSE` if you would like the default value for the inflation factor, $F = 1.5$, to be used, or a list specifying the parameters required by the adjustment.

- `"N_trials"`: the number of repetitions of the adujustment procedure based on the simulation of a gaussisan population of functional data, each one producing an adjusted value of F, which will lead to the averaged adjusted value $\bar{F}$. Default is 20;
- `"trial_size"`: the number of elements in the gaussian population of functional data that will be simulated at each repetition of the adjustment procedure. Default is `5 * fData$N`;
- `"TPR"`: the True Positive Rate of outleirs, i.e. the proportion of observations in a dataset without shape outliers that have to be considered outliers. Default is `2 * pnorm( 4 * qnorm( 0.25 ) )`;
- `"F_min"`: the minimum value of F, defining the left boundary for the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to `fData`, the optimal value of F. Default is 0.5;
- `"F_max"`: the maximum value of F, defining the right boundary for the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to `fData`, the optimal value of F. Default is 20;
- `"tol"`: the tolerance to be used in the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to `fData`, the optimal value of F. Default is `1e-3`;

- "maxiter": the maximum number of iterations to solve the optimisation problem aimed at finding, for a given dataset of simulated gaussian data associated to fData, the optimal value of F. Default is 100;
- "VERBOSE": a parameter controlling the verbosity of the adjustment process;

display — either a logical value indicating wether you want the outliergram to be displayed, or the number of the graphical device where you want the outliergram to be displayed.

xlab — a list of two labels to use on the x axis when displaying the functional dataset and the outliergram

ylab — a list of two labels to use on the y axis when displaying the functional dataset and the outliergram;

main — a list of two titles to be used on the plot of the functional dataset and the outliergram;

... — additional graphical parameters to be used *only* in the plot of the functional dataset

Value

Even when used graphically to plot the outliergram, the function returns a list containing:

- Fvalue: the value of the parameter F used;
- d: the vector of values of the parameter d for each observation (distance to the parabolic border of the outliergram);
- ID_outliers: the vector of observations id corresponding to outliers.

Adjustment

When the adjustment option is selected, the value of F is optimised for the univariate functional dataset provided with fData. In practice, a number adjust$N_trials of times a synthetic population (of size adjust$trial_size with the same covariance (robustly estimated from data) and centerline as fData is simulated without outliers and each time an optimised value F_i is computed so that a given proportion (adjust$TPR) of observations is flagged as outliers. The final value of F for the outliergram is determined as an average of $F_1, F_2, \ldots, F_{N_{trials}}$. At each time step the optimisation problem is solved using stats::uniroot (Brent's method).

References

Arribas-Gil, A., and Romo, J. (2014). Shape outlier detection and visualization for functional data: the outliergram, *Biostatistics*, 15(4), 603-619.

See Also

fData, MEI, MBD, fbplot

Examples

```
set.seed( 1618 )

N = 200
P = 200
N_extra = 4

grid = seq( 0, 1, length.out = P )

Cov = exp_cov_function( grid, alpha = 0.2, beta = 0.8 )

Data = generate_gauss_fdata( N,
                             centerline = sin( 4 * pi * grid ),
                             Cov = Cov )

Data_extra = array( 0, dim = c( N_extra, P ) )

Data_extra[ 1, ] = generate_gauss_fdata( 1,
                                         sin( 4 * pi * grid + pi / 2 ),
                                         Cov = Cov )

Data_extra[ 2, ] = generate_gauss_fdata( 1,
                                         sin( 4 * pi * grid - pi / 2 ),
                                         Cov = Cov )

Data_extra[ 3, ] = generate_gauss_fdata( 1,
                                         sin( 4 * pi * grid + pi/ 3 ),
                                         Cov = Cov )

Data_extra[ 4, ] = generate_gauss_fdata( 1,
                                         sin( 4 * pi * grid - pi / 3),
                                         Cov = Cov )
Data = rbind( Data, Data_extra )

fD = fData( grid, Data )

outliergram( fD, display = TRUE )

outliergram( fD, Fvalue = 2.5, display = TRUE )
## Not run:
outliergram( fD,
             adjust = list( N_trials = 10,
                            trial_size = 5 * nrow( Data ),
                            TPR = 0.01,
                            VERBOSE = FALSE ),
             display = TRUE )

## End(Not run)
```

plot.Cov *Specialised method to plot* Cov *objects*

Description

This function performs the plot of an object of class Cov, i.e. a covariance or cross-covaraince function.

Usage

```
## S3 method for class 'Cov'
plot(x, ...)
```

Arguments

x	the covariance or cross-covariance function of class Cov.
...	additional graphical parameters to be used in plotting functions

Details

It builds above the function graphics::image, therefore any additional parameter suitable for graphics::image will also be suitable as ... argument to plot.Cov.

See Also

cov_fun

Examples

```
# Generating a univariate functional dataset
N = 1e2

P = 1e2
t0 = 0
t1 = 1

time_grid = seq( t0, t1, length.out = P )

Cov = exp_cov_function( time_grid, alpha = 0.3, beta = 0.4 )

D1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * time_grid ), Cov = Cov )

fD1 = fData( time_grid, D1 )

# Computing the covariance function of fD1

plot( cov_fun( fD1 ), main = 'Covariance function', xlab = 'time', ylab = 'time' )
```

plot.fData *Specialised method to plot* fData *objects*

Description

This function performs the plot of a functional univariate dataset stored in an object of class fData. It is able to accept all the usual customisable graphical parameters, otherwise it will use the default ones.

Usage

```
## S3 method for class 'fData'
plot(x, ...)
```

Arguments

x	the univariate functional dataset in form of fData object.
...	additional graphical parameters to be used in plotting functions

See Also

fData

Examples

```
N = 20
P = 1e2

# One dimensional grid
grid = seq( 0, 1, length.out = P )

# Generating an exponential covariance function (see related help for more
# information )
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Generating a synthetic dataset with a gaussian distribution and
# required mean and covariance function:
values = generate_gauss_fdata( N,
                               centerline = sin( 2 * pi * grid ),
                               Cov = C )

fD = fData( grid, values )

plot( fD )
```

plot.mfData *Specialised method to plot* mfData *objects*

Description

This function performs the plot of a functional multivariate dataset stored in an object of class mfData. It is able to accept all the usual customisable graphical parameters, otherwise it will use the default ones.

Usage

```
## S3 method for class 'mfData'
plot(x, ...)
```

Arguments

x	the multivariate functional dataset in form of mfData object.
...	additional graphical parameters to be used in plotting functions (see Details for the use of ylab and main).

Details

The current active graphical device is split into a number of sub-figures, each one meant to contain the plot of the corresponding dimension of the mfData object. In particular, they are arranged in a rectangular lattice with a number of rows equal to $\lfloor\sqrt{L}\rfloor$ and a number of columns equal to $\lceil L/\lfloor\sqrt{L}\rfloor\rceil$.

A special use of the graphical parameters allows to set up y-labels and titles for all the sub-figures in the graphical window. In particular, parameters ylab and main can take as argument either a single string, that are repeatedly used for all the sub-graphics, or a list of different strings (one for each of the L dimensions) that have to be used in the corresponding graphic.

See Also

mfData, fData, plot.fData

Examples

```
N = 1e2

P = 1e3

t0 = 0
t1 = 1

# Defining the measurement grid
grid = seq( t0, t1, length.out = P )

# Generating an exponential covariance matrix to be used in the simulation of
```

```
# the functional datasets (see the related help for details)
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating the measurements of two univariate functional datasets with
# required center and covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )

# Building the mfData object and plotting tt
plot( mfData( grid, list( Data_1, Data_2 ) ),
      xlab = 'time', ylab = list( '1st dim.', '2nd dim.' ),
      main = list( 'An important plot here', 'And another one here' ) )
```

`plus-.fData` *Operator* `+` *and* `-` *for* `fData` *objects*

Description

These methods provide operators `+` and `-` to perform sums or differences between an `fData` object and either another `fData` object or other compliant data structures, like matrices or vectors or arrays, representing the pointwise measurements of the second term of the sum.

Usage

```
## S3 method for class 'fData'
fD + A

## S3 method for class 'fData'
fD - A
```

Arguments

`fD`	the univariate functional dataset in form of `fData` object.
`A`	either an `fData` object, defined on the very same grid of `fD`, or a 1D data structure (such as 1D array or raw numeric vector), or a 2D data structure (such as 2D array or raw numeric matrix), that specifies the second term of the sum. In case of a 1D data structure, the sum is performed element-wise between each element of `fD` and `A`, and `A` must have length `P`, size of `fD`'s grid. In case of a 2D data structure, the sum is performed element-wise between corresponding elements of `fD` and `A`'s rows. In this case, `A` must have `P` columns, as the size of `fD`'s grid.

Details

If the second term of the operation is an `fData` object, it must be defined over the same grid as the first.

Value

The function returns an `fData` object, whose function values have undergone the sum/difference.

Examples

```
fD = fData( seq( 0, 1, length.out = 10 ),
            values = matrix( seq( 1, 10 ),
                             nrow = 21, ncol = 10, byrow = TRUE ) )
fD + 1 : 10

fD + array( 1, dim = c( 1, 10 ) )

fD + fD

fD = fData( seq( 0, 1, length.out = 10 ),
            values = matrix( seq( 1, 10 ),
                             nrow = 21, ncol = 10, byrow = TRUE ) )
fD - 2 : 11

fD - array( 1, dim = c( 1, 10 ) )

fD - fD
```

---

roahd *roahd: RObust Analysis for High dimensional Data.*

Description

A package meant to collect and provide methods for the analysis of unviariate and multivariate functional datasets through the use of robust methods, with special focus on computation of depths and outlier detection. Special care was devoted to the efficient implementation of robust methods, so that they can be employed also in high-dimensional datasets,

---

set_alpha *Function to setup alpha value for a set of colours*

Description

`set_alpha` manipulates a vector of colour representations in order to setup the alpha value, and get the desired transparency level.

Usage

```
set_alpha(col, alpha)
```

Arguments

col	a vector of colours
alpha	the value(s) of alpha for (each of) the colors.

See Also

fDColorPalette

Examples

```
original_col = c( 'blue', 'red', 'green', 'yellow' )

alpha_col = set_alpha( original_col, 0.5 )

alpha_col = set_alpha( original_col, c(0.5, 0.5, 0.2, 0.1 ) )

dev.new()
par( mfrow = c( 1, 2 ) )

plot( seq_along( original_col ),
      seq_along( original_col ),
      col = original_col,
      pch = 16,
      cex = 2,
      main = 'Original colours' )

plot( seq_along( alpha_col ),
      seq_along( alpha_col ),
      col = alpha_col,
      pch = 16,
      cex = 2,
      main = 'Alpha colours' )
```

---

sub-.fData *Operator* sub-.fData *to subset* fData *obejcts*

Description

This method provides an easy and natural way to subset a functional dataset stored in a fData object, without having to deal with the inner representation of fData class.

Usage

```
## S3 method for class 'fData'
fD[i, j, as_fData = TRUE]
```

Arguments

fD	the univariate functional dataset in form of fData object.
i	a valid expression to subset rows (observations) of the univariate functional dataset.
j	a valid expression to subset columns (measurements over the grid) of the univariate functional dataset (must be contiguous).
as_fData	logical flag to specify whether the output should be returned as an fData object containing the required subset or as a matrix of values, default is TRUE.

Value

The method returns either an fData object (if as_fData = TRUE) or a matrix (if as_fData = FALSE) containing the required subset (both in terms of observations and measurement points) of the univariate functional dataset.

See Also

fData

Examples

```
N = 20
P = 1e2

# One dimensional grid
grid = seq( 0, 1, length.out = P )

# Generating an exponential covariance function (see related help for more
# information )
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Generating a synthetic dataset with a gaussian distribution and
# required mean and covariance function:
fD = fData( grid,
            generate_gauss_fdata( N,
                                  centerline = sin( 2 * pi * grid ),
                                  Cov = C ) )

dev.new()
par( mfrow = c( 2, 2 ) )

# Original data
plot( fD )

# Subsetting observations
plot( fD[ c(1,2,3), , as_fData = TRUE ] )

# Subsetting measurements
plot( fD[ , 1 : 30 ] )
```

```
# Subsetting both observations and measurements
plot( fD[ 1 : 10, 50 : P ] )

# Subsetting both observations and measurements but returning a matrix
fD[ 1 : 10, 50 : P, as_fData = FALSE ]
```

`sub-.mfData` *Operator* `sub-.mfData` *to subset* `mfData` *obejcts*

Description

This method provides an easy and natural way to subset a multivariate functional dataset stored in a `mfData` object, without having to deal with the inner representation of `mfData` class.

Usage

```
## S3 method for class 'mfData'
mfD[i, j]
```

Arguments

`mfD`	the multivariate functional dataset in form of `mfData` object.
`i`	a valid expression to subset rows (observations) of the univariate functional dataset.
`j`	a valid expression to subset columns (measurements over the grid) of the univariate functional dataset (must be contiguous).

Value

The method returns and `mfData` object containing the required subset (both in terms of observations and measurement points) of the multivariate functional dataset.

See Also

`mfData`

Examples

```
# Defining parameters
N = 1e2

P = 1e3

t0 = 0
t1 = 1
```

```
# Defining the measurement grid
grid = seq( t0, t1, length.out = P )

# Generating an exponential covariance matrix to be used in the simulation of
# the functional datasets (see the related help for details)
C = exp_cov_function( grid, alpha = 0.3, beta = 0.4 )

# Simulating the measurements of two univariate functional datasets with
# required center and covariance function
Data_1 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )
Data_2 = generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )

# Building the mfData object
mfD = mfData( grid, list( Data_1, Data_2 ) )

# Subsetting the first 10 elements and 10 time points
mfD[1:10, 1:10]

# Subsetting only observations
mfD[1:10,]

# Subsetting only time points (contiguously)
mfD[,1:10]
```

`times-.fData` *Operator* `*` *and* `/` *for* `fData` *objects*

Description

These methods provide operators `*` and `/` to perform products or divisions between an `fData` object and either a number or a compliant 1D data structure, like numeric vector, array or matrix. The operation is computed by performing the element-wise product or division between `fD`'s observations and the provided value(s).

Usage

```
## S3 method for class 'fData'
fD * a

## S3 method for class 'fData'
fD / a
```

Arguments

`fD`	the univariate functional dataset in form of `fData` object.
`a`	either a single number or a 1D data structure (such as numeric raw vector, matrix or array) specifying the factor(s) to use in the multiplication/division of `fD` elements' values. In the latter case, each factor is used with the corresponding element in `fD`, hence a must have length `N`, number of observations in `fD`.

Details

If the second argument is a 1D data structure, it must have length N equal to the number of observations in fD.

Value

The function returns an fData object, whose function values have undergone the product/division.

Examples

```
N = 11
fD = fData( seq( 0, 1, length.out = 10 ),
            values = matrix( seq( 1, 10 ),
                             nrow = N, ncol = 10, byrow = TRUE ) )
fD * 2

fD * seq( 1, N )


N = 11
fD = fData( seq( 0, 1, length.out = 10 ),
            values = matrix( seq( 1, 10 ),
                             nrow = N, ncol = 10, byrow = TRUE ) )
fD / 2

fD / rep( 10, N )
```

---

toListOfValues *Manipulation of* mfData *list of values*

Description

This utility function manipulates a mfData object in order to extract from the list of its fData objects (namely, mfData$fDList) the measurement values of each component and stores them into a list.

Usage

```
toListOfValues(mfData)
```

Arguments

mfData	the multivariate functional dataset in form of mfData object.

Details

Given a mfData of L components, the function is equivalent to `list( mfData$fDList[[ 1 ]]$values, ..., mfData$fDList[[ L ]]$values )` .

Value

The function returns the list of values of each `fData` object representing the components of `mfData`.

See Also

`mfData`

Examples

```
grid = seq( 0, 1, length.out = 5 )

D_1 = matrix( 1 : 5, nrow = 10, ncol = 5, byrow = TRUE )
D_2 = 2 * D_1
D_3 = 3 * D_1

mfD = mfData( grid, list( D_1, D_2, D_3 ) )
mfD

toListOfValues( mfD )
```

---

toRowMatrixForm *Conversion of vector/array/matrix to row-matrix form*

Description

This function manipulates a numeric data structure of vector/array/matrix type in order to obtain a matrix representation. For 1D data structures and column/row arrays and matrices the output is turned in a matrix format with just one row. If the input structure is rectangular, instead, it is only converted in matrix format.

Usage

```
toRowMatrixForm(D)
```

Arguments

`D`	a generic array, matrix or vector to be converted in row-matrix format.

Warning

The function is **not** supposed to work with arbitrary N-dimensional arrays.

Examples

```
toRowMatrixForm( 1 : 10 )

toRowMatrixForm( array( 1 : 10, dim = c(1,10 ) ) )

toRowMatrixForm( array( 1 : 10, dim = c( 10, 1 ) ) )

toRowMatrixForm( matrix( 1 : 10, ncol = 10, nrow = 1 ) )

toRowMatrixForm( matrix( 1 : 10, ncol = 1, nrow = 10 ) )

toRowMatrixForm( matrix( 1 : 12, ncol = 3, nrow = 4 ) )

toRowMatrixForm( matrix( 1 : 12, ncol = 4, nrow = 3 ) )
```

---

unfold *Unfolding a univariate functional dataset*

Description

This function operates on a univariate functional dataset and transforms its observations unfolding their values and turning them into monotone functions.

Usage

```
unfold(fData)
```

Arguments

fData	the unvariate functional dataset in form of fData object.

Details

Each function of the fData object is transformed into a nonmonotone function into a monotone function by "unfolding" it at any of its maxima. For more details about the definition of the transform, see the reference.

Value

The function returns an fData object whose observations are the unfolded version of the corresponding observations in the argument fData.

References

Arribas-Gil, A. and Romo, J. (2012). Robust depth-based estimation in the time warping model, *Biostatistics*, 13 (3), 398–414.

See Also

fData, warp

Examples

```
P = 1e3

time_grid = seq( 0, 1, length.out = P )

D = matrix( c( sin( 2 * pi * time_grid ),
               cos( 2 * pi * time_grid ),
               sin( 10 * pi * time_grid ) * time_grid + 2 ),
            ncol = P, nrow = 3, byrow = TRUE )

# Functional dataset
fD = fData( time_grid, D )

# Unfolded version
fD_unfold = unfold( fD )

dev.new()
par( mfrow = c( 1, 2 ) )
plot( fD, main = 'Original data' )
plot( fD_unfold, main = 'Unfolded data' )
```

---

warp *Warp elements of a univariate functional dataset*

Description

This function carries out the warping of elements of a univariate functional dataset by using a set of pre-computed warping functions.

Usage

```
warp(fData, warpings)
```

Arguments

fData	the functional dataset whose observations must be warped in form of fData object.
warpings	the warping functions $H_1, \ldots, H_N$, in form of fData object, defined over the registered/warped grid.

Details

Given a univariate functional dataset $X_1(t), \ldots, X_N(t)$ and a set of warping functions $H_1(t), \ldots, H_N(t)$, such that:

$$H_i : s \longrightarrow t = H_i(s), \quad \forall i = 1, \ldots, N,$$

where s spans the warped (or registered) grid and t spans the original grid, the function computes the warping given by the following composition:

$$X_1 \circ H_1(t), \ldots, X_N \circ H_N(t).$$

Value

The function returns the univariate functional dataset of warped functions, in form of `fData` object.

See Also

`fData`

Examples

```
set.seed( 1618033 )

N = 30

t0 = 0
t1 = 1
P = 1e3 + 1

time_grid = seq( t0, t1, length.out = P )

means = round( runif( N,
                      t0 + (t1 - t0) / 8,
                      t1 - (t1 - t0) / 8  ), 3 )

Data = matrix( sapply( means,
                       function( m )( dnorm( time_grid, mean = m, sd = 0.05 ) ) ),
               ncol = P, nrow = N, byrow = TRUE )

fD = fData( time_grid, Data )

# Piecewise linear warpings
template_warping = function( m )( c( time_grid[ time_grid <= 0.5 ] * m / 0.5,
                                     ( time_grid[ time_grid > 0.5 ]
                                       - 0.5 ) * (1 - m ) / 0.5 + m ) )


warpings = matrix( sapply( means, template_warping ),
                   ncol = P,
                   nrow = N, byrow = TRUE )

wfD = fData( time_grid, warpings )
```

```
fD_warped = warp( fD, wfD )

dev.new()
par( mfrow = c( 1, 3 ) )
plot( fD,
     main = 'Unregistered functions', xlab = 'actual grid', ylab = 'values'  )
plot( wfD,
     main = 'Warping functions', xlab = 'registered grid',
     ylab = 'actual grid' )
plot( fD_warped,
     main = 'Warped functions', xlab = 'registered grid',
     ylab = 'values' )
```

Index