Package ‘fChange’

December 27, 2017

Type Package

Title Change Point Analysis in Functional Data

Version 0.2.0

Author Ozan Sonmez, Alexander Aue, Gregory Rice

Maintainer Ozan Sonmez <osonmez@ucdavis.edu>

Description
Change point estimation and detection methods for functional data are implemented using dimension reduction via functional principal component analysis and a fully-functional (norm-based) method. Detecting and dating structural breaks for both dependent and independent functional samples is illustrated along with some basic functional data generating processes.

License MIT + file LICENSE

Encoding UTF-8

LazyData TRUE

Depends R (>= 2.10)

Imports fda, sde, stats, sandwich, reshape2, lattice

Suggests knitr

RoxygenNote 6.0.1

R topics documented:

`Australian_Temp` *Australian Climate Data*

Description

Australian daily minimum temperature climate data for 8 different stations are provided. The data is taken from Australian Government Bureau of Meteorology.

Usage

```
Australian_Temp
```

Format

An object of class `data.table` or `data.frame`.

Value

Australian daily minimum temperature climate data for 8 different stations:

`Sydney`	Sydney (Observatory Hill), taken from 1859 to 2012
`Melbourne`	Melbourne (Regional Office), taken from 1855 to 2012
`Boulia`	Boulia Airport, taken from 1888 to 2012
`Cape_Otway`	Cape Otway Lighthouse, taken from 1864 to 2012
`Gayndah`	Gayndah Post Office, taken from 1893 to 2009
`Gunnedah`	Gunnedah Pool, taken from 1876 to 2011
`Hobart`	Hobart (Ellerslie Road), taken from 1882 to 2012
`Robe`	Robe Comparison, taken from 1884 to 2012

Source

The daily observations are available from http://www.bom.gov.au/climate/data. Copyright Commonwealth of Australia 2010, Bureau of Meteorology. Definitions adapted from http://www.bom.gov.au/climate/dwo/ID

References

Australian Government Bureau of Meteorology

Examples

```
library(fda)
library(reshape2)
fun_data_S = Australian_Temp$Sydney
D = 21
basis = create.fourier.basis(rangeval = c(0, 1), nbasis = D)
nas = which(is.na(fun_data_S$Days.of.accumulation.of.minimum.temperature))
fun_data_S = fun_data_S[-nas, ]
yy = unique(fun_data_S$Year)
mat.S = matrix(0, D, length(yy))
for (i in 1:length(yy)){
 aa = subset(fun_data_S, Year==yy[i])
 cc = aa$Minimum.temperature..Degree.C.
 a = which(is.na(cc))
 if (length(a)>0){
 cc = cc[-which(is.na(cc))]
 }else{
   cc = cc
 }
 f_Obs = Data2fd(argvals=seq(0, 1, length = length(cc)) , cc, basisobj = basis)
 mat.S[, i] = f_Obs$coefs
}
fdata = fd(mat.S, basis)
# note that the last year, has data only up to 6 months
# therefore we remove it
fdata = fdata[-length(yy)]
plot(fdata)
```

`center_data` *Center Functional Data With Change*

Description

This function centers the functional data by subtracting the pointwise mean from each of the functions in a functional data by taking into account a potential change in the mean function. If there is a change in the mean function, the location of the change is estimated using a fully functional estimator implemented in `change_FF`, and the mean before and after the change is computed and subtracted from the respective part of the functional data.

Usage

```
center_data(fdobj, change = TRUE)
```

Arguments

`fdobj`	A functional data object
`change`	If `TRUE` centering is done by considering the mean change, if `FALSE`, the global mean function is subtracted from the functional data. The default is `change=TRUE`.

Value

Centered functional data sample (class fd) containing:

coefs	The coefficient array
basis	A basis object
fdnames	A list containing names for the arguments, function values and variables

See Also

center.fd

Examples

```
# Generate FAR(1) process with change in the mean
f_AR = fun_AR(n=100, nbasis=21, kappa=0.9)
f_AR_change = insert_change(f_AR, k=20, change_location = 0.5, SNR=5)
fdata = f_AR_change$fundata
c_fdata = center_data(fdata)
par(mfrow=c(1,2))
plot(fdata, main="Functional Data")
plot(c_fdata, main="Centered Functional Data")
```

---

change_FF — *Change Point Analysis Of Functional Data Without Dimension Reduction (Fully Functional)*

Description

This function tests whether there is a significant change in the mean function of the functional data, and it will give an estimate for the location of the change. The procedure is based on the standard L-2 norm and hence does not depend on any dimension reduction technique such as fPCA.

Usage

```
change_FF(fdobj, M = 1000, h = 0, plot = FALSE, ...)
```

Arguments

fdobj	A functional data object of class 'fd'
M	Number of monte carlo simulations used to get the critical values. The default value is M=1000
h	The window parameter for the estimation of the long run covariance kernel. The default value is h=0, i.e., it assumes iid data
plot	If TRUE plot of the functional data before and after the estimated change and plot of the estimated change function is given
...	Further arguments to pass

Details

This function dates and detects changes in the mean function of functional data using a fully functional technique that does not dependent on dimension reduction. For more details, see Aue, Rice, Sonmez (2017+)

Value

pvalue	Approximate p value for testing whether there is a significant change in the mean function
change	Estimated change location
DataBefore	Data before the estimated change
DataAfter	Data after the estimated change
MeanBefore	Mean function before the estimated change
MeanAfter	Mean function after the estimated change
change_fun	Estimated change function

References

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

change_fPCA

Examples

```
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
# insert an artifiical change
data_c = insert_change(fdata, k=21, change_location = 0.5, SNR=1)$fundata
change_FF(data_c)$change
```

---

change_fPCA *Change Point Analysis Of Functional Data Via Dimension Reduction*

Description

This function tests whether there is a significant change in the mean function of functional data, and it gives an estimate of the location of the change. The procedure will reduce the dimension of the functional data using functional principal component analysis and will use d leading principal curves to carry out the change point analysis. The projection dimension d can be chosen via total variation explained (TVE) using the function pick_dim.

Usage

```
change_fPCA(fdobj, d, M = 1000, h = 0, plot = FALSE, ...)
```

Arguments

`fdobj`	A functional data object of class 'fd'
`d`	Number of principal components
`M`	Number of monte carlo simulations to get the critical values. The default value is `M=1000`
`h`	The window parameter for the estimation of the long run covariance kernel. The default value is `h=0`, i.e., it assumes iid data
`plot`	If `TRUE` plot of the functional data before and after the estimated change and plot of the estimated change function is given
`...`	Further arguments to pass

Details

This functions performs structural break analysis for the functional data using an fPCA based initial dimension reduction. It is recommended that the dimension of the subspace, `d`, that the functional observations are projected onto should be selected based on TVE using `pick_dim`.

Value

`pvalue`	An approximate p value for testing whether there is a significant change in the mean function
`change`	Estimated change location
`DataBefore`	Data before the estimated change
`DataAfter`	Data after the estimated change
`MeanBefore`	Mean function before the estimated change
`MeanAfter`	Mean function after the estimated change
`change_fun`	Estimated change function

References

Berkes, I., Gabrys, R.,Hovarth, L. & P. Kokoszka (2009)., *Detecting changes in the mean of functional observations* Journal of the Royal Statistical Society, Series B 71, 927–946

Aue, A., Gabrys, R.,Hovarth, L. & P. Kokoszka (2009)., *Estimation of a change-point in the mean function of functional data*Journal of Multivariate Analysis 100, 2254–2269.

See Also

`change_FF`

Examples

```
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
# insert an artifiical change
data_c = insert_change(fdata, k=21, change_location = 0.5, SNR=1)$fundata
d.hat = pick_dim(data_c, 0.9)$d
change_fPCA(data_c, d=d.hat)$change
```

Conf_int *Confidence Intervals For Change Point Estimator*

Description

A change point in the mean function of functional data is estimated, and for a user selected level aconfidence interval is computed using the fully functional procedure introduced in Aue, Rice and Sonmez (2017+).

Usage

```
Conf_int(fdobj, h = 0, kern_type = "BT", alpha = 0.05, ...)
```

Arguments

fdobj	Functional data object, class of "fd"
h	Bandwidth parameter to estimate the long run covariance operator.
kern_type	Kernel to be used for the estimation of the long run covariance function
alpha	Nominal level used to construct the confidence intervals
...	Further arguments to pass

Value

Lower CI	Lower Confidence Band
Estimate	Estimated change point location
Upper CI	Upper Confidence Band

References

Aue, A., Rice, G. & O. Sonmez (2017+), *Detecting and dating structural breaks ¨ in functional data without dimension reduction*, (https://arxiv.org/pdf/1511.04020.pdf)

See Also

LongRun opt_bandwidth change_FF

Examples

```
fdata1 = fun_AR(n=100, nbasis=21, order=1, kappa=0.5)
Conf_int(fdata1, h=2)
```

`Cov_test` *Testing the Equality of Covariance Operators in Functional Samples*

Description

This function tests for the equality of the covariance structures in two functional samples.

Usage

```
Cov_test(fdobj1, fdobj2, d)
```

Arguments

`fdobj1`	A functional data object of class 'fd'
`fdobj2`	A functional data object of class 'fd'
`d`	Level ofdimension reduction needed to represent the data. One can use `pick_dim`

Details

test for the equality of the covariance structures in two functional samples. The test statistic has a chi-square asymptotic distribution with a known number of degrees of freedom, which depends on the level of dimension reduction needed, d, to represent the data.

Value

`pvalue`	Approximate p value for testing equality of the covariance structures in two functional samples

References

Fremdt S., Horvath L., Kokoszka P., Steinebach J. (2017+), *Testing the Equality of Covariance Operators in Functional Samples* Scandinavian Journal of Statistics, 2013.

See Also

`change_fPCA`

Examples

```
# generate functional data
fdata1 = fun_IID(n=100, nbasis=21)
fdata2 = fun_IID(n=150, nbasis=21)
Cov_test(fdata1, fdata2, d=5)
```

`eval_component` *Detecting Changes in the Eigenvalues of the Covariance Operator of the Functional Data*

Description

This function tests and detects changes in the specific eigenvalue of the covariance operator.

Usage

```
eval_component(fdobj, component, h = 2, mean_change = FALSE, delta = 0.1,
  M = 1000)
```

Arguments

`fdobj`	A functional data object of class '`fd`'
`component`	The eigenvalue that the componentwise test is applied to.
`h`	The window parameter for the estimation of the long run covariance matrix. The default value is `h=2`.
`mean_change`	If `TRUE` then the data is centered considering the change in the mean function.
`delta`	Trimming parameter to estimate the covariance function using partial sum estimates.
`M`	Number of monte carlo simulations used to get the critical values. The default value is `M=1000`

Details

This function dates and detects changes in the defined eigenvalue of the covariance function. The critical values are approximated via `M` Monte Carlo simulations.

Value

`pvalue`	Approximate p value for testing whether there is a significant change in the desired eigenvalue of the covariance operator
`change`	Estimated change location

Examples

```
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
eval_component(fdata, 2)
```

eval_joint	*Detecting Changes Jointly in the Eigenvalues of the Covariance Operator of the Functional Data*

Description

This function tests and detects changes jointly in the eigenvalue of the covariance operator.

Usage

```
eval_joint(fdobj, d, h = 2, mean_change = FALSE, delta = 0.1, M = 1000)
```

Arguments

fdobj	A functional data object of class 'fd'
d	Number of eigenvalues to include in testing.
h	The window parameter for the estimation of the long run covariance matrix. The default value is h=2.
mean_change	If TRUE then the data is centered considering the change in the mean function.
delta	Trimming parameter to estimate the covariance function using partial sum estimates.
M	Number of monte carlo simulations used to get the critical values. The default value is M=1000

Details

This function dates and detects changes in the joint eigenvalues that is defined by d of the covariance function. The critical values are approximated via M Monte Carlo simulations.

Value

pvalue	Approximate p value for testing whether there is a significant change in the desired eigenvalue of the covariance operator
change	Estimated change location
eval_before	Estimated eigenvalues before the change
eval_after	Estimated eigenvalues after the change

Examples

```
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
eval_joint(fdata, 2)
```

fun_AR — Simulate Functional Auto-Regressive Process

Description

This generates a functional Auto-Regressive, FAR, process of sample size n with a specific number of basis functions where the eigenvalue decay of the covariance operator is given by the defined vector Sigma. The norm of the FAR operators are defined by the vector `kappa`. The generic function uses Fourier basis in [0,1], however one can define a different basis and different range values. If the order or kappa is not defined then the function generates iid functional data by default.

Usage

```
fun_AR(n, nbasis, order = NULL, kappa = NULL, Sigma = NULL,
  basis = NULL, rangeval = c(0, 1), ...)
```

Arguments

`n`	Sample size of generated functional data. A strictly positive integer
`nbasis`	Number of basis functions used to represent functional observations
`order`	Order of the FAR process
`kappa`	Vector of norm of the FAR operators. The length of this vector must be same as the FAR `order`
`Sigma`	Eigen value decay of the covariance operator of the functional data. The eigenvalues of the covariance operator of the generated functional sample are given by `Sigma`. The length of `Sigma` must match number of basis. By default it is set as `(1:nbasis)^-1`
`basis`	A functional basis object defining the basis. It can be the class of `basisfd`, `fd`, `fdPar`. As a default it is set to be a Fourier basis
`rangeval`	A vector of length 2 containing the initial and final values of the interval over which the functional data object can be evaluated. As a default it is set to be [0,1].
`...`	Further arguments to pass

Details

This function should be used for a simple FAR data generation for a desired eigenvalue decay of covariance operator. The j-th FAR operator $\Psi[j]$ is generated by $\Psi[j] = \kappa[j]\Psi$, where Ψ has a unit norm with $\Psi[i,j] = N(0, \sigma[i]\sigma[j])$. For more details see Aue A., Rice G., Sonmez O. (2017+).

Value

Functional Auto-Regressive data sample (class `fd`) containing:

`coefs`	The coefficient array
`basis`	A basis object
`fdnames`	A list containing names for the arguments, function values and variables

References

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

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

Data2fd, fun_IID, fun_MA

Examples

```
# FAR(1) data with 21 fourier basis with a geometric eigenvalue decay
fun_AR(n=100, nbasis=21, order=1, kappa=0.8)


# Define eigenvalue decay
Sigma1 = 2^-(1:21)
# Then generate FAR(2) data
fun_AR(n=100, nbasis=21, order=2, kappa= c(0.5, 0.3), Sigma=Sigma1)

# Define eigenvalue decay, and basis function
library(fda)
basis1 = create.bspline.basis(rangeval = c(0,1), nbasis=21)
Sigma1 = 2^-(1:21)
# Then generate FAR(1)
fun_AR(n=100, nbasis=21, order=1, kappa= 0.3,Sigma=Sigma1, basis=basis1)

# Not defining order will result in generating IID functions
fun_AR(n=100, nbasis=21) # same as fun_IID(n=100, nbasis=21)
```

---

fun_heavy_tailed *Simulate Heavy Tailed Independent Functional Data*

Description

It generates a heavy tail independent functional observations of sample size n

Usage

```
fun_heavy_tailed(n, nbasis, df = 3, basis = NULL, rangeval = c(0, 1), ...)
```

Arguments

n	Sample size of generated functional data. A strictly positive integer
nbasis	Number of basis functions used to represent functional observations
df	Degrees of freedom of the T-distribution to construct the functional observations. The default value is 3
basis	A functional basis object defining the basis. It can be the class of basisfd, fd, fdPar. As a default it is set to be a Fourier basis

`rangeval` A vector of length 2 containing the initial and final values of the interval over which the functional data object can be evaluated. As a default it is set to be [0,1].

`...` Further arguments to pass

Details

The implementation of this function is very similar to `fun_IID`. The heavy tail functional observations are generated based on a linear combination of basis functions where the i-th linear combination coefficient has a t-distribution with `df` - degrees of freedom.

Value

An independent functional data sample (class `fd`) containing:

`coefs` The coefficient array

`basis` A basis object

`fdnames` A list containing names for the arguments, function values and variables

References

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

Aue A., Rice G., Sonmez O. (2017), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

`Data2fd`, `fun_IID`, `fun_AR`, `fun_MA`

Examples

```
fdata1 = fun_heavy_tailed(n=100, nbasis=25)
fdata2 = fun_heavy_tailed(n=100, nbasis=25, df=4)
```

`fun_IID` *Simulate Independent Functional Data*

Description

This function generates independent functional observations of sample size n with a desired eigenvalue decay structure of the covariance operator.

Usage

```
fun_IID(n, nbasis, Sigma = NULL, basis = NULL, rangeval = c(0, 1), ...)
```

Arguments

n	Sample size of generated functional data. A strictly positive integer
nbasis	Number of basis functions used to represent functional observations
Sigma	Eigen value decay of the covariance operator of the functional data. The eigenvalues of the covariance operator of the generated functional sample are given by Sigma. The length of Sigma must match number of basis. By default it is set as (1:nbasis)^-1
basis	A functional basis object defining the basis. It can be the class of "basisfd, fd, fdPar". As a default it is set to be a Fourier basis
rangeval	A vector of length 2 containing the initial and final values of the interval over which the functional data object can be evaluated. As a default it is set to be [0,1].
...	Further arguments to pass

Details

Independent functional sample is generated based on a linear combination of basis functions where the i-th linear combination coefficient is normally distributed with mean zero and standard deviation $\sigma[i]$.

Value

An independent functional data sample (class fd) containing:

coefs	The coefficient array
basis	A basis object
fdnames	A list containing names for the arguments, function values and variables

References

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

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

Data2fd

Examples

```
# Functional data with 21 fourier basis with a geometric eigenvalue decay
fun_IID(n=100, nbasis=21)

# Define eigenvalue decay
Sigma1=2^-(1:21)
# Then generate functional data
fun_IID(n=100, nbasis=21, Sigma=Sigma1)

# Define eigenvalue decay, and basis function
library(fda)
basis1 = create.bspline.basis(rangeval = c(0,1), nbasis=21)
```

```
Sigma1=2^-(1:21)
# Then generate functional data
fun_IID(n=100, nbasis=21, Sigma=Sigma1, basis=basis1)
```

---

fun_MA *Simulate Functional Moving Average Process*

Description

It generates functional Moving Average, FMA, process of sample size n with a specific number of basis functions where the eigenvalue decay of the covariance operator is given by the defined vector Sigma. The norm of the functional moving average operators are defined by the vector `kappa`. The generic function uses Fourier basis in [0,1], however one can define a different basis and different range values. If the order or kappa is not defined then the function generates iid functional data by default.

Usage

```
fun_MA(n, nbasis, order = NULL, kappa = NULL, Sigma = NULL,
  basis = NULL, rangeval = c(0, 1), ...)
```

Arguments

`n`	Sample size of generated functional data. A strictly positive integer
`nbasis`	Number of basis functions used to represent functional observations
`order`	Order of the FMA process
`kappa`	Vector of norm of the FMA operators. The length of this vector must be same as the FMA `order`
`Sigma`	Eigen value decay of the covariance operator of the functional data. The eigenvalues of the covariance operator of the generated functional sample will mimic the behavior of `Sigma`. The length of `Sigma` must match number of basis. By default it is set as `(1:nbasis)^-1`
`basis`	A functional basis object defining the basis. It can be the class of `basisfd`, `fd`, `fdPar`. As a default it is set to be a Fourier basis
`rangeval`	A vector of length 2 containing the initial and final values of the interval over which the functional data object can be evaluated. As a default it is set to be [0,1].
`...`	Further arguments to pass

Details

This function should be used for a simple functional moving average data generation for a desired eigenvalue decay of covariance operator. The j-th FMA operator $\Theta[j]$ is generated by $\Theta[j] = \kappa[j]\Theta$, where Θ has a unit norm with $\Theta[i,j] = N(0, \sigma[i]\sigma[j])$. For more details see Aue A., Rice G., Sonmez O. (2017+).

Value

Functional Moving Average data sample (class fd) containing:

coefs	The coefficient array
basis	A basis object
fdnames	A list containing names for the arguments, function values and variables

References

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

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

Data2fd, fun_IID

Examples

```
# FMA(1) data with 21 fourier basis with a geometric eigenvalue decay
fun_MA(n=100, nbasis=21, order=1, kappa=0.8)


# Define eigenvalue decay
Sigma1 = 2^-(1:21)
# Then generate FMA(2) data
fun_MA(n=100, nbasis=21, order=2, kappa= c(0.5, 0.3), Sigma=Sigma1)

# Define eigenvalue decay, and basis function
library(fda)
basis1 = create.bspline.basis(rangeval = c(0,1), nbasis=21)
Sigma1 = 2^-(1:21)
# Then generate FMA(1)
fun_MA(n=100, nbasis=21, order=1, kappa= 0.3,Sigma=Sigma1, basis=basis1)

# Not defining order will result in generating IID functions
fun_MA(n=100, nbasis=21) # same as fun_IID(n=100, nbasis=21)
```

---

insert_change *Insert A Change In the Mean Function Of Functional Data*

Description

This function inserts a change in the mean function to a given functional data sample. The change function can either be directly defined by the user or it can be generated based on the sum of first k basis functions defined by the fdobj. Once the change function is defined the change is inserted at the defined change location with a signal magnitude defined by signal to noise ratio, SNR. For more details on how these quantities are defined. See Aue, Rice and Sonmez (2017+).

Usage

```
insert_change(fdobj, change_fun = NULL, k = NULL, change_location, SNR,
  plot = TRUE, ...)
```

Arguments

fdobj	Functional data object of class 'fd'
change_fun	Self defined change function. It has to be a functional data object having the same number of basis functions.
k	Number of basis functions to be summed to construct the change function. It should be used when change_fun is not defined. It has to be less than number of basis functions.
change_location	Location of the change to be inserted. It is scaled to be in [0,1].
SNR	Signal to Noise Ratio to determine the magnitude of the change function that is being inserted.
plot	Plots the functional data before (blue) and after (red) the change.
...	Further information to pass

Details

This function should only be used to artificially insert a change function to the mean of the functional data set either by defining a specific change function or generating the change function based from the basis functions.

Value

fundata: functional data with an inserted change in the mean function

change_fun: inserted change function

plot: of the functional data with inserted change

References

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

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

See Also

Data2fd, fun_IID, fun_MA, fun_AR

Examples

```
####################################
#first generate FAR(1) process
fdata = fun_AR(n=100, nbasis=25, Sigma=2^-(1:25))
# insert the change which is the sum of first 3 basis functions
# in the middle of the data with SNR=2
insert_change(fdata, k=3, change_location=0.5, SNR=2)
```

```
#####################################
#first generate FAR(1) process
fdata = fun_AR(n=100, nbasis=25, Sigma=2^-(1:25))
# insert the change which is the 20th onservation
# in the middle of the data with SNR=2
insert_change(fdata, change_fun = fdata[20], change_location=0.5, SNR=2)
```

LongRun *Long Run Covariance Operator Estimation for Functional Time Series*

Description

This function estimates the long run covariance operator of a given functional data sample and its estimated eigenelements.

Usage

```
LongRun(fdobj, h, kern_type = "BT", is_change = TRUE, ...)
```

Arguments

`fdobj`	A functional data object
`h`	The bandwidth parameter. It is strictly non-zero. Choosing the bandwidth parameter to be zero is identical to estimating covariance operator assuming iid data.
`kern_type`	Kernel function to be used for the estimation of the long run covariance function. The choices are `c("BT", "PR", "SP", "FT")` which are respectively, bartlett, parzen, simple and flat-top kernels. By default the function uses a `"barlett"` kernel.
`is_change`	If `TRUE` then the data is centered considering the change in the mean function.
`...`	Further arguments to pass

Value

`e_fun`	Eigenfunctions of the estimated long run covariance function
`e_val`	Eigenvalues of the estimated long run covariance function
`covm`	Coefficient matrix of the estimated long run covariance operator.
`contour_plot`	The estimated covariance function $C(t, s)$ surface plot if `plot=TRUE`

References

Aue A., Rice G., Sonmez O. (2017+), *Detecting and dating structural breaks in functional data without dimension reduction* (https://arxiv.org/pdf/1511.04020.pdf)

Rice G. and Shang H. L. (2017), *A plug-in bandwidth selection procedure for long run covariance estimation with stationary functional time series*, Journal of Time Series Analysis, 38(4), 591-609

See Also

`opt_bandwidth`

Examples

```
# Generate FAR(1) process
fdata = fun_AR(n=100, nbasis=31, order=1, kappa=0.9)
# Estimate the Long run covrariance
C_hat = LongRun(fdata, h=2)
C_hat$e_fun # eigenfunctions of Long Run Cov
C_hat$e_val # eigenvalues of Long Run Cov
C_hat$covm # Estimated covariance matrix
```

`LongRunCovMatrix` *Long Run Covariance Matrix Estimation for Multivariate Time Series*

Description

This function estimates the long run covariance matrix of a given multivariate data sample.

Usage

```
LongRunCovMatrix(mdobj, h = 0, kern_type = "bartlett")
```

Arguments

`mdobj`	A multivariate data object
`h`	The bandwidth parameter. It is strictly non-zero. Choosing the bandwidth parameter to be zero is identical to estimating covariance matrix assuming iid data.
`kern_type`	Kernel function to be used for the estimation of the long run covariance matrix. The choices are `c("BT", "PR", "SP", "FT")` which are respectively, bartlett, parzen, simple and flat-top kernels. By default the function uses a `"barlett"` kernel.
`...`	Further arguments to pass

Value

Estimated long run covariance matrix.

See Also

`LongRun`

Examples

```
# Generate FAR(1) process
fdata = fun_AR(n=100, nbasis=31, order=1, kappa=0.9)
```

opt_bandwidth *Optimal Bandwidth Selection for the Long Run Covariance Estimation*

Description

This function estimates an optimal window parameter for long run covariance operator estimation in functional time series using the method of Rice G. and Shang H. L. (2017)

Usage

```
opt_bandwidth(fdobj, kern_type, kern_type_ini, is_change = TRUE, ...)
```

Arguments

`fdobj`	Functional data object, class of `"fd"`
`kern_type`	Kernel that is used for the long run covariance estimation. The available options are `c("BT", "PR", "TH", "QS")` where `"BT"` is Bartlett, `"PR"` is Parzen, `"TH"` is Tukey-Hanning, and `"QS"` is Quadratic Spectral kernel.
`kern_type_ini`	Initial Kernel function to start the optimal bandwidth search
`is_change`	If `TRUE` then the data is centered considering the change in the mean function
`...`	Further arguments to pass

Value

`hat_h_opt`	Estimated optimal bandwidth
`C_0_est`	Estimated Long run covariance kernel using the optimal bandwidth `hat_h_opt`

References

Rice G. and Shang H. L. (2017), *A plug-in bandwidth selection procedure for long run covariance estimation with stationary functional time series*, Journal of Time Series Analysis, 38(4), 591-609

See Also

LongRun

Examples

```
fdata1 = fun_AR(n=100, nbasis=21, order=1, kappa=0.8)
opt_bandwidth(fdata1, "PR", "BT")
```

partial_cov *Partial Sample Estimates of the Covariance Function in Functional Data Analysis*

Description

This function computes the partial sum estimate of the covariance function in functional data analysis. It also generate the eigenvalues and eigenfunctions of the parial sum estimate, along with the coefficient matrix of the estimated covariance function.

Usage

```
partial_cov(fdobj, x = NULL)
```

Arguments

fdobj	A functional data object of class 'fd'
x	Fraction of the sample size that the partial sum is computed. This input must be in (0,1]. The default is x=1 which corresponds to the regular covariance function estimate in functional data analysis.

Details

This function simply estimates the covariance function based on the partial sum of the centered functional observations. The length of the sum is determined by x, and when x=1 this estimate corresponds to the regular covariance function estimates using the whole sample.

Value

eigen_val	Eigenvalues of the partial sum estimate of the covariance function
eigen_fun	Eigenfunctions of the partial sum estimate of the covariance function
coef_matrix	Coefficient matrix of the partial sum estimate of the covariance function

See Also

pca.fd

Examples

```
library(fda)
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
# Estimated eigenvalues
e1 = partial_cov(fdata)$eigen_val
e2 = pca.fd(fdata, nharm = 21, centerfns = TRUE)$values
# e1 and e2 will both estimate the eigenvalues of the covariance
# operator based on the whole sample
# estimates using only 90% of the data
Cov = partial_cov(fdata, 0.9)
```

pick_dim *Number Of Principal Component Selection Based On Variation*

Description

This function selects number of principal components based on the total variation explained (TVE). The functional data is projected onto a smaller number of principal curves via functional Principal Component Analysis (fPCA). This function picks the dimension of the projection space based on the desired total variation explained.

Usage

```
pick_dim(fdobj, TVE)
```

Arguments

fdobj	Functional data object, class of "fd"
TVE	Total Variation Explained. It must be in [0,1].

Details

This function is used to determine the dimension of the space that the functional data is projected onto based on the variation. One of the common treatments of the functional data is to transform it to multivariate objects using so called score vectors and utilize the multivariate techniques. This function will enable users to pick the dimension of the score vectors.

Value

d	Minimum number of Principle components needed in order to reach desired TVE
TVEs	Vector of TVEs. This has the same length of number of basis that the functional data is represented

References

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

See Also

pca.fd

Examples

```
fdata1 = fun_IID(n=100, nbasis=21)
pick_dim(fdata1, 0.95)
fdata2 = fun_IID(n=100, nbasis=21, Sigma=3^-(1:21))
pick_dim(fdata2, 0.95)
```

Stock — *Intra-Day Stock Price Data Sets*

Description

Stock prices in one minute resolution are recorded for Bank of America, Walmart, Disney, Chevron, IBM, Microsoft, CocaCola, Exon mobile, McDonalds and Microsoft. The time period that the observations span is given in the column names of each stock data. During each day, stock price values were recorded in one-minute intervals from 9:30 AM to 4:00 PM EST. The columns display the day, and the rows display the time at which the stock price is recorded.

Usage

```
Stock
```

Format

An object of class `data.table` or `data.frame`.

Value

Several Stock Price Data Sets:

`Walmart`	Walmart Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`Disney`	Disney Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`ExonMobile`	Exon Mobile Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`IBM`	IBM Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`CitiGroup`	Citi Group Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`McDonalds`	McDonalds Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`Microsoft`	Microsoft Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`BankOfAmerica`	Bank Of America Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`Chevron`	Chevron Stock Price Data Set taken from 06/15/2006 to 04/02/2007
`CocaCola`	Coca Cola Stock Price Data Set taken from 06/15/2006 to 04/02/2007

Source

Google Finance

Examples

```
 library(fda)
 # transform first 100 days of Chevron data into functional data object
 data = as.matrix(Stock$Chevron[,1:100])
 # First need to transform the data to obtain the log returns
 temp1=data
 for(j in c(1:dim(data)[1])){
  for(k in c(1:dim(data)[2])){
    data[j,k]=100*(log(temp1[j,k])-log(temp1[1,k]))
  }
}
```

```
# Transform the Log return data into functional data
#using 21 bspline basis functions on [0,1]
D=21
basis =  create.bspline.basis(rangeval = c(0, 1), nbasis = D)
Domain = seq(0, 1, length = nrow(data))
f_data = Data2fd(argvals = Domain , data, basisobj = basis)
plot(f_data)
```

`trace_change` *Testing changes in the trace of the covariance operator in functional data*

Description

This function tests and detects changes in the trace of the covariance operator.

Usage

```
trace_change(fdobj, mean_change = FALSE, delta = 0.1, M = 1000)
```

Arguments

`fdobj`	A functional data object of class 'fd'
`mean_change`	If `TRUE` then the data is centered considering the change in the mean function.
`delta`	Trimming parameter to estimate the covariance function using partial sum estimates.
`M`	Number of monte carlo simulations used to get the critical values. The default value is `M=1000` value is h=2.

Details

This function dates and detects changes in trace of the covariance function. This can be interpreted as the changes in the total variation of the the functional data. Trace is defined as the infinite sum of the eigenvalues of the covariance operator and for the sake of implementation purpose, the sum is truncated up to the total number of basis functions that defines the functional data at hand. The critical values are approximated via `M` Monte Carlo simulations.

Value

`pvalue`	Approximate p value for testing whether there is a significant change in the desired eigenvalue of the covariance operator
`change`	Estimated change location
`trace_before`	Estimated trace before the change
`trace_after`	Estimated trace after the change

Examples

```
# generate functional data
fdata = fun_IID(n=100, nbasis=21)
trace_change(fdata)
```

Index