Package ‘Rfssa’

September 13, 2019

Type Package

Title Functional Singular Spectrum Analysis

Version 1.0.0

Maintainer Hossein Haghbin <haghbinh@gmail.com>

URL https://github.com/haghbinh/Rfssa.git

Description Methods and tools for implementing functional singular spectrum analysis for functional time series
as described in Haghbin H., Najibi, S.M., Mahmoudvand R., Trinka J., Maadooliat M. (2019). Functional singular spectrum Analysis. Manuscript submitted for publication.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports Rcpp, fda, lattice, plotly, shiny, Rssa, markdown,

LinkingTo Rcpp, RcppArmadillo,

Suggests knitr,

Depends R (>= 2.10), dplyr

NeedsCompilation yes

Author Hossein Haghbin [aut, cre],
Seyed Morteza Najibi [aut],
Jordan Trinka [aut],
Mehdi Maadooliat [aut]

Repository CRAN

Date/Publication 2019-09-12 22:40:20 UTC

R topics documented:

`*.fts` *Multiplication of Functional Time Series*

Description

A method that lets you multiply functional time series (`fts`) and perform scalar multiplication of functional time series.

Usage

```
## S3 method for class 'fts'
Y1 * Y2
```

Arguments

`Y1`	an object of class `fts` or scalar
`Y2`	an object of class `fts` or scalar

Value

an object of class `fts`

See Also

`fts`

Examples

```
## Not run:
require(fda)
require(Rfssa)
data(Callcenter) # Read data
u=seq(0,1,length.out=240) # Define domain of functional data
d=12 # number of basis elements
basis=create.bspline.basis(rangeval = c(0,1),nbasis = d) # create basis object
smooth.calls=smooth.basis(u, matrix(nrow=240,ncol=365,Callcenter$calls), basis)
Y=fts(smooth.calls$fd) # create functional time series
plot(Y)
Ytimes=Y*Y # elementwise multiplication of the functional time series with itself
plot(Ytimes)
Ytimes2=2*Y # multiply 2 with every term in the functional time series
plot(Ytimes2)

## End(Not run)
```

`+.fts` *Addition of Functional Time Series*

Description

A method that lets you perform functional time series (`fts`) addition and scalar addition.

Usage

```
## S3 method for class 'fts'
Y1 + Y2
```

Arguments

Y1	an object of class `fts` or scalar
Y2	an object of class `fts` or scalar

Value

an object of class `fts`.

See Also

`fts`

Examples

```
## Not run:
require(fda)
require(Rfssa)
data(Callcenter) # Read data
u=seq(0,1,length.out=240) # Define domain of functional data
d=12 # number of basis elements
basis=create.bspline.basis(rangeval = c(0,1),nbasis = d) # create basis object
smooth.calls=smooth.basis(u, matrix(nrow=240,ncol=365,Callcenter$calls), basis)
Y=fts(smooth.calls$fd) # create functional time series
plot(Y)
Yplus=Y+Y # add the functional time series to itself
plot(Yplus)
Yplus2=Y+2 # add 2 to every term in the functional time series
plot(Yplus2)

## End(Not run)
```

-.fts *Subtraction of Functional Time Series*

Description

A method that lets you perform functional time series (fts) subtraction and scalar subtraction.

Usage

```
## S3 method for class 'fts'
Y1 - Y2
```

Arguments

Y1	an object of class fts or scalar
Y2	an object of class fts or scalar

Value

an object of class fts.

See Also

fts

Examples

```
## Not run:
require(fda)
require(Rfssa)
data(Callcenter) # Read data
u=seq(0,1,length.out=240) # Define domain of functional data
d=12 # number of basis elements
basis=create.bspline.basis(rangeval = c(0,1),nbasis = d) # create basis object
smooth.calls=smooth.basis(u, matrix(nrow=240,ncol=365,Callcenter$calls), basis)
Y=fts(smooth.calls$fd) # create functional time series
plot(Y)
Yminus=Y[4:8]-Y[14:18] # subtract elements of the functional time series from each other
plot(Yminus)
Yminus2=Y-2 # add 2 to every term in the functional time series
plot(Yminus2)

## End(Not run)
```

Callcenter *Number of Calls for a Bank.*

Description

This dataset is a small call center for an anonymous bank (Brown et al., 2005). This dataset provides the exact time of the calls that were connected to the center from January 1 to December 31 in the year 1999. The data are aggregated into time intervals to obtain a data matrix. More precisely, the *(i,j)*'th element of the data matrix contains the call volume during the *j*th time interval on day *i*. This dataset has been analyzed in several prior studies; e.g. Brown et al. (2005), Shen and Huang (2005), Huang et al. (2008), and Maadooliat et al. (2015). Here, the data are aggregated into time intervals 6 minutes.

Usage

```
Callcenter
```

Format

A dataframe with 87600 rows and 5 variables:

calls The number of calls in 6 minutes aggregated interval.

u a numeric vector to show the aggregated interval.

Date Date time when the calls counts are recorded.

Day Weekday associated with Date.

Month Month associated with Date.

Source

http://iew3.technion.ac.il/serveng/callcenterdata/index.html

References

1. Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., & Zhao, L. (2005). Statistical analysis of a telephone call center: A queueing-science perspective. *Journal of the American statistical association*, **100**(469), 36-50.
2. Shen, H., & Huang, J. Z. (2005). Analysis of call center arrival data using singular value decomposition. Applied Stochastic Models in Business and Industry, 21(3), 251-263.
3. Huang, J. Z., Shen, H., & Buja, A. (2008). Functional principal components analysis via penalized rank one approximation. *Electronic Journal of Statistics*, **2**, 678-695.
4. Maadooliat, M., Huang, J. Z., & Hu, J. (2015). Integrating data transformation in principal components analysis. *Journal of Computational and Graphical Statistics*, **24**(1), 84-103.

See Also

fssa

---

cor.fts *Correlation for Functional Time Series Objects*

Description

This function finds the correlation between univarite or multivariate functional time series (fts) objects.

Usage

```
cor.fts(Y1, Y2)
```

Arguments

Y1	an object of class fts
Y2	an object of class fts

Value

a scalar that is the correlation between fts objects

See Also

fts

Examples

```
## Not run:
require(fda)
require(Rfssa)
## Raw image data
NDVI=Jambi$NDVI
EVI=Jambi$EVI
## Kernel density estimation of pixel intensity
D0_NDVI <- matrix(NA,nrow = 512, ncol = 448)
D0_EVI <- matrix(NA,nrow =512, ncol = 448)
for(i in 1:448){
  D0_NDVI[,i] <- density(NDVI[,,i],from=0,to=1)$y
  D0_EVI[,i] <- density(EVI[,,i],from=0,to=1)$y
}
## Define functional objects
d <- 11
basis <- create.bspline.basis(c(0,1),d)
u <- seq(0,1,length.out = 512)
y_NDVI <- smooth.basis(u,as.matrix(D0_NDVI),basis)$fd
y_EVI <- smooth.basis(u,as.matrix(D0_EVI),basis)$fd
Y_NDVI <- fts(y_NDVI)
Y_EVI <- fts(y_EVI)
cor.fts(Y_NDVI,Y_EVI)

## End(Not run)
```

---

freconstruct — *Reconstruction Stage of Functional Singular Spectrum Analysis*

Description

This is a function for reconstructing functional time series (fts) objects from functional singular spectrum analysis (fssa) objects (including Grouping and Hankelization steps). The function performs the reconstruction step for univariate functional singular spectrum analysis (ufssa) or multivariate functional singular spectrum analysis (mfssa) depending on whether or not the input is an fssa object from ufssa or mfssa.

Usage

```
freconstruct(U, group = as.list(1L:10L))
```

Arguments

U	an object of class fssa
group	a list of numeric vectors, each vector includes indices of elementary components of a group used for reconstruction

Value

a named list of objects of class `fts` that are reconstructed as according to the specified groups and a numeric vector of eigenvalues

Note

refer to `fssa` for an example on how to run this function starting from `fssa` objects

See Also

`fssa`, `fts`,

`fssa` *Functional Singular Spectrum Analysis*

Description

This is a function which performs the decomposition (including embedding and functional SVD steps) stage for univariate functional singular spectrum analysis (ufssa) or multivariate functional singular spectrum analysis (mfssa) depending on whether the supplied input is a univariate or multivariate functional time series (`fts`) object.

Usage

```
fssa(Y, L = NA, type = "fssa")
```

Arguments

`Y`	an object of class `fts`
`L`	window length
`type`	type of FSSA with options of `type = "ufssa"` or `type = "mfssa"`

Value

An object of class `fssa`, which is a list of multivariate functional objects and the following components:

`values`	a numeric vector of eigenvalues
`L`	window length
`N`	length of the functional time series
`Y`	the original functional time series

Examples

```
## Not run:
## Univariate FSSA Example on Callcenter data
data("Callcenter")
require(fda)
require(Rfssa)
## Define functional objects
D <- matrix(sqrt(Callcenter$calls),nrow = 240)
N <- ncol(D)
time <- seq(ISOdate(1999,1,1), ISOdate(1999,12,31), by="day")
K <- nrow(D)
u <- seq(0,K,length.out =K)
d <- 22 #Optimal Number of basis elements
basis <- create.bspline.basis(c(min(u),max(u)),d)
Ysmooth <- smooth.basis(u,D,basis)
## Define functional time series
Y <- fts(Ysmooth$fd,time = time)
plot(Y,ylab = "Sqrt of Callcenter", xlab = "Intraday intervals")

## Univariate functional singular spectrum analysis
L <- 28
U <- fssa(Y,L)
plot(U,d=13)
plot(U,d=9,type="lheats")
plot(U,d=9,type="lcurves")
plot(U,d=9,type="vectors")
plot(U,d=10,type="periodogram")
plot(U,d=10,type="paired")
plot(U,d=10,type="wcor")
gr <- list(1,2:3,4:5,6:7,8:20)
Q <- freconstruct(U, gr)
plot(Y,main="Call Numbers(Observed)")
plot(Q[[1]],main="1st Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[2]],main="2nd Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[3]],main="3rd Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[4]],main="4th Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[5]],main="5th Component(Noise)",ylab = " ", xlab = "Intraday intervals")

## Other visiualisation types for object of class "fts":

plot(Q[[1]], type="3Dsurface", main="1st Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[2]][1:60], type="heatmap", main="2nd Component",ylab = " ", xlab = "Intraday intervals")
plot(Q[[3]][1:60], type = "3Dline", main="3rd Component",ylab = " ", xlab = "Intraday intervals")

## Multivariate FSSA Example on Bivariate Satelite Image Data
require(fda)
require(Rfssa)
## Raw image data
NDVI=Jambi$NDVI
EVI=Jambi$EVI
time <- Jambi$Date
```

```
## Kernel density estimation of pixel intensity
D0_NDVI <- matrix(NA,nrow = 512, ncol = 448)
D0_EVI <- matrix(NA,nrow =512, ncol = 448)
for(i in 1:448){
  D0_NDVI[,i] <- density(NDVI[,,i],from=0,to=1)$y
  D0_EVI[,i] <- density(EVI[,,i],from=0,to=1)$y
}
## Define functional objects
d <- 11
basis <- create.bspline.basis(c(0,1),d)
u <- seq(0,1,length.out = 512)
y_NDVI <- smooth.basis(u,as.matrix(D0_NDVI),basis)$fd
y_EVI <- smooth.basis(u,as.matrix(D0_EVI),basis)$fd
y=list(y_NDVI,y_EVI)
## Define functional time series
Y <- fts(y,time=time)
plot(Y[1:100],ylab = c("NDVI","EVI"),main = "Probability Kernel Density")
plot(Y, type = '3Dsurface', var=1,ylab = c("NDVI"),main = "Probability Kernel Density")
plot(Y, type = '3Dline', var=2,ylab = c("EVI"),main = "Probability Kernel Density")
plot(Y, type = 'heatmap',ylab = c("NDVI","EVI"),main = "Probability Kernel Density")
L=45
## Multivariate functional singular spectrum analysis
U=fssa(Y,L)
plot(U,d=10,type='values')
plot(U,d=10,type='paired')
plot(U,d=10,type='lheats', var = 1)
plot(U,d=10,type='lcurves',var = 1)
plot(U,d=10,type='lheats', var = 2)
plot(U,d=10,type='lcurves',var = 2)
plot(U,d=10,type='wcor')
plot(U,d=10,type='periodogram')
plot(U,d=10,type='vectors')
recon <- freconstruct(U = U, group = list(c(1),c(2,3),c(4)))
plot(recon[[1]],type = '3Dsurface',var=1, ylab = "NDVI")
plot(recon[[2]],type = '3Dsurface',var=1, ylab = "NDVI")
plot(recon[[3]],type = '3Dsurface',var=1, ylab = "NDVI")
plot(recon[[1]],type = '3Dsurface',var=2, ylab = "EVI")
plot(recon[[2]],type = '3Dsurface',var=2, ylab = "EVI")
plot(recon[[3]],type = '3Dsurface',var=2, ylab = "EVI")


## End(Not run)
```

`fts` *Functional Time Series Class*

Description

This function is used to create functional time series objects from functional data (fd) objects.

Usage

```
fts(Y, time = NULL)
```

Arguments

Y	an object of class fd or a list of objects of class fd
time	the vector of times at which a time series was sampled

Note

refer to fssa for an example on how to run this function starting from fd objects

See Also

fssa

---

fwcor — *Weighted Correlation Matrix*

Description

This function returns the weighted correlation (w-correlation) matrix for functional time series (fts) objects that were reconstructed from functional singular spectrum analysis (fssa) objects.

Usage

```
fwcor(U, group)
```

Arguments

U	an object of class fssa
group	a list or vector of indices which determines the grouping used for the reconstruction in pairwise w-correlations matrix

Value

a square matrix of w-correlation values for the reconstructed fts objects that were built from fssa components

See Also

fssa, freconstruct, fts, wplot

Examples

```
## Not run:

## Univariate W-Correlation Example on Callcenter data
data("Callcenter")
require(fda)
require(Rfssa)
## Define functional objects
D <- matrix(sqrt(Callcenter$calls),nrow = 240)
N <- ncol(D)
time <- 1:N
K <- nrow(D)
u <- seq(0,K,length.out =K)
d <- 22 #Optimal Number of basis elements
basis <- create.bspline.basis(c(min(u),max(u)),d)
Ysmooth <- smooth.basis(u,D,basis)
## Define functional time series
Y <- fts(Ysmooth$fd)
## Decomposition stage of univariate functional singular spectrum analysis
L <- 28
U <- fssa(Y,L)
ufwcor=fwcor(U = U,group = list(1,2,3))
wplot(W=ufwcor)

## Multivariate W-Correlation Example on Bivariate Satelite Image Data
require(fda)
require(Rfssa)
## Raw image data
NDVI=Jambi$NDVI
EVI=Jambi$EVI
## Kernel density estimation of pixel intensity
D0_NDVI <- matrix(NA,nrow = 512, ncol = 448)
D0_EVI <- matrix(NA,nrow =512, ncol = 448)
for(i in 1:448){
  D0_NDVI[,i] <- density(NDVI[,,i],from=0,to=1)$y
  D0_EVI[,i] <- density(EVI[,,i],from=0,to=1)$y
}
## Define functional objects
d <- 11
basis <- create.bspline.basis(c(0,1),d)
u <- seq(0,1,length.out = 512)
y_NDVI <- smooth.basis(u,as.matrix(D0_NDVI),basis)$fd
y_EVI <- smooth.basis(u,as.matrix(D0_EVI),basis)$fd
y=list(y_NDVI,y_EVI)
## Define functional time series
Y=fts(y)
plot(Y)
L=45
## Decomposition stage of multivariate functional singular spectrum analysis
U=fssa(Y,L)
mfwcor=fwcor(U = U,group = list(1,2,3,4))
```

```
wplot(W=mfwcor)

## End(Not run)
```

Jambi *Jambi MODIS Data*

Description

This data set contains the normalized difference vegetation index (NDVI) and enhanced vegetation index (EVI) image data from NASA's MODerate-resolution Imaging Spectroradiometer (MODIS) with global coverage at 250 m^2. The goal of the study is to collect raw image data of the Jambi Province, Indonesia. Indonesia manages various forested land utilizations such as natural forest and plantations which, in the past, have been exploited throughout the country. Greater criticisms on forest exploitation lead to a moratorium which needs to be monitored frequently. Assessment of woody vegetation could be taken using field surveys or remote sensing. It was found that season is probably the most intriguing factor in vegetative land cover, especially in long-term land cover changes (Lambin, 1999). The data was gathered starting in 2000-02-18 and ending in 2019-07-28 every 16 days.

Usage

```
Jambi
```

Format

A list which contains two 33 by 33 by 448 arrays where one array is for NDVI image data and the other is for EVI image data. The list also contains a date vector of length 448 which specifies upon which date was each image 33 by 33 image taken.

Days 1 - 448 Pixel intensity with values between zero and one

@references

1. Lambin, E., Geist, H., Lepers, E. (1999). Dynamics of Land-Use and Land-Cover Change in Tropical Regions *Annual Review of Environment and Resources*, 205-244.

Source

https://lpdaac.usgs.gov/products/mod13q1v006/

See Also

fssa

launchApp *Launch the Shiny Application Demonstration*

Description

This function launches an app that can be used to help an individual better understand univariate or multivariate functional singular spectrum analysis (`fssa`). The app allows the user to run univariate or multivariate functional singular spectrum analysis (depending on the entered type of parameter) on a variety of data types including simulated and real data available through the server. The app also has functionality that allows the user to upload their own data. The app allows the user to compare different methods simultaneously such as multivariate singular spectrum analysis versus univariate functional singular spectrum analysis. It also allows the user to choose the number and types of basis elements used to estimate functional time series (`fts`) objects. The app supports `fts` plots and `fssa` plots.

Usage

```
launchApp(type = "ufssa")
```

Arguments

type	type of FSSA with options of `type = "ufssa"` or `type = "mfssa"`

Value

a shiny application object

Examples

```
## Not run:

launchApp()


## End(Not run)
```

plot.fssa *Plot Functional Singular Spectrum Analysis Objects*

Description

This is a plotting method for objects of class functional singular spectrum analysis (`fssa`). The method is designed to help the user make decisions on how to do the grouping stage of univariate or multivariate functional singular spectrum analysis.

Usage

```
## S3 method for class 'fssa'
plot(x, d = length(x$values), idx = 1:d, idy = idx +
  1, contrib = TRUE, groups = as.list(1:d), type = "values",
  var = 1L, ylab = NA, ...)
```

Arguments

- `x` an object of class `fssa`
- `d` an integer which is the number of elementary components in the plot
- `idx` a vector of indices of eigen elements to plot
- `idy` a second vector of indices of eigen elements to plot (for type="paired")
- `contrib` a logical where if the value is 'TRUE' (the default), the contribution of the component to the total variance is displayed
- `groups` a list or vector of indices determines grouping used for the decomposition(for type="wcor")
- `type` the type of plot to be displayed where possible types are:
  - `"values"` plot the square-root of singular values (default)
  - `"paired"` plot the pairs of eigenfunction's coefficients (useful for the detection of periodic components)
  - `"wcor"` plot the W-correlation matrix for the reconstructed objects
  - `"vectors"` plot the eigenfunction's coefficients (useful for the detection of period length)
  - `"lcurves"` plot of the eigenfunctions (useful for the detection of period length)
  - `"lheats"` heatmap plot the eigenfunctions (useful for the detection of meaningful patterns)
  - `"periodogram"` periodogram plot (useful for the detecting the frequencies of oscillations in functional data)
- `var` an integer specifying the variable number
- `ylab` the character vector of name of variables
- `...` arguments to be passed to methods, such as graphical parameters

Note

for a multivariate example, see the examples in `fssa`

See Also

`fssa`, `plot.fts`

Examples

```
## Not run:
## Simulated Data Example
require(Rfssa)
require(fda)
n <- 50 # Number of points in each function.
d <- 9
N <- 60
sigma <- 0.5
set.seed(110)
E <- matrix(rnorm(N*d,0,sigma/sqrt(d)),ncol = N, nrow = d)
basis <- create.fourier.basis(c(0, 1), d)
Eps <- fd(E,basis)
om1 <- 1/10
om2 <- 1/4
f0 <- function(tau, t) 2*exp(-tau*t/10)
f1 <- function(tau, t) 0.2*exp(-tau^3) * cos(2 * pi * t * om1)
f2 <- function(tau, t) -0.2*exp(-tau^2) * cos(2 * pi * t * om2)
tau <- seq(0, 1, length = n)
t <- 1:N
f0_mat <- outer(tau, t, FUN = f0)
f0_fd <- smooth.basis(tau, f0_mat, basis)$fd
f1_mat <- outer(tau, t, FUN = f1)
f1_fd <- smooth.basis(tau, f1_mat, basis)$fd
f2_mat <- outer(tau, t, FUN = f2)
f2_fd <- smooth.basis(tau, f2_mat, basis)$fd
Y_fd <- f0_fd+f1_fd+f2_fd
L <-10
U <- fssa(Y_fd,L)
plot(U)
plot(U,d=4,type="lcurves")
plot(U,d=4,type="vectors")
plot(U,d=5,type="paired")
plot(U,d=5,type="wcor")
plot(U,d=5,type="lheats")
plot(U,d=5,type="periodogram")

## End(Not run)
```

`plot.fts` *Functional Time Series Visualization Tools Using Plotly*

Description

This is a plotting method for univariate or multivariate functional time series (fts). This method is designed to help the user visualize fts data using a variety of techniques that use plotly.

Usage

```
## S3 method for class 'fts'
plot(x, npts = 100, type = "line", main = NULL,
  ylab = NULL, xlab = NULL, tlab = NULL, var = NULL, ...)
```

Arguments

`x`	an object of class `fts`
`npts`	number of points to evaluate functional object at
`type`	the type of plot to be displayed where possible types are: • `"line"` plot the `fts` elements in a line plot (default) • `"heatmap"` plot the `fts` elements in a heat map • `"3Dsurface"` plot the `fts` elements as a surface • `"3Dline"` plot the `fts` elements in a three-dimensional line plot
`main`	the main title
`ylab`	the y-axis label
`xlab`	the x-axis label
`tlab`	the time-axis label
`var`	an integer specifying the variable number to plot if `type="3Dsurface"` or `type="3Dline"`
`...`	arguments to be passed to methods, such as graphical parameters.

Note

for a multivariate example, see the examples in `fssa`

Examples

```
## Not run:
require(fda)
require(Rfssa)
data(Callcenter) # Read data
u=seq(0,1,length.out=240) # Define domain of functional data
d=12 # number of basis elements
basis=create.bspline.basis(rangeval = c(0,1),nbasis = d) # create basis object
smooth.calls=smooth.basis(u, matrix(nrow=240,ncol=365,Callcenter$calls), basis)
Y=fts(smooth.calls$fd) # create functional time series
plot(Y,type = "heatmap")
plot(Y,type = "line",var = 1)
plot(Y,type = "3Dsurface",var = 1)
plot(Y,type = "3Dline",var = 1)

## End(Not run)
```

Rfssa *Rfssa: A Package for Functional Singular Spectrum Analysis.*

Description

The Rfssa package provides the collection of necessary functions to implement functional singular spectrum analysis (FSSA) for analysing functional time series (FTS) data. FSSA is a novel non-parametric method to perform decomposition and reconstruction of FTS.

Details

The use of the package starts with the decomposition of functional time series (`fts`) objects using `fssa`. Then a suitable grouping of the principal compomnents is required for reconstruction which can be done heuristically by looking at the plots of the decomposition (`plot`). Alternatively, one can examine the weighted correlation (w-correlation) matrix (`fwcor`). The final step is the reconstruction of the principal components into additive `fts` objects whose sum approximates the original univariate or multivariate functional time series (`freconstruct`).

This version of the Rfssa package includes updates to existing functions including `fssa`, `plot`, `wplot`, and `freconstruct`. Multivariate functional singular spectrum analysis (mfssa) was added to the package in `fssa` to allow the user to perform embedding and decomposition of a multivariate FTS. The reconstruction stage in `freconstruct` was also updated to allow for reconstruction (including Hankelization) of multivariate FTS objects using multivariate FSSA objects that come from mfssa. Plotting options for FSSA objects in `plot` were also updated so that the user can now plot left singular functions, right singular vectors, left singular function heat diagrams, and periodograms. FSSA plotting options also allow the user to specify which particular components they want to plot. For example, a user can specify that they want to see a paired-plot of only the third and fourth component. The 'meanvectors' and 'meanpaired' options were removed as these are satisfied with 'paired' and 'vectors' options. The 'efunctions' and 'efunctions2' options were also removed in lieu of the addition of the left singular function heat map option. The user can also specify the 'cuts' parameter in `wplot` to make visualization of the w-correlation matrix easier.

This version of the Rfssa package also includes new functions for converting functional data (FD) objects to FTS objects, arithmetic, indexing, correlation, and plotting of FTS data. The user is able to convert an FD object to an FTS object using `fts`. The user can also perform addition, subtraction, and multiplication of FTS objects with other FTS objects or FTS objects with scalars by using '+', '-', and '*' respectively. The package also allows for indexing of FTS objects by using '['. The user can also measure the unweighted correlation between FTS objects by using `cor.fts`. The plotting of FTS objects can be performed using `plot` which uses the plotly package for visualization.

The package update also includes a new shiny app (`launchApp`) that can be used for demonstrations of univariate or multivariate FSSA depending on the type that is specified. The app allows the user to explore FSSA with simulated data, data that is provided on the server, or data that the user provides. It allows the user to change parameters as they please, gives visual results of the methods, and also allows the user to compare FSSA results to other spectrum analysis methods such as multivariate singular spectrum analysis. The tool is easy to use and can act as a nice starting point for a user that wishes to perform FSSA as a part of their data analysis.

References

Haghbin H., Najibi, S.M., Mahmoudvand R., Trinka J., Maadooliat M. (2019). Functional singular spectrum Analysis. Manuscript submitted for publication.

See Also

`fssa`, `freconstruct`, `fwcor`, `wplot`, `fts`, `plot.fts`, `plot.fssa`, `cor.fts`, `launchApp`

`wplot` *Weighted-Correlations Plot*

Description

This function displays a plot of the weighted-correlation (w-correlation) matrix of functional time series (`fts`) objects that were reconstructed from functional singular spectrum analysis (`fssa`) objects.

Usage

```
wplot(W, cuts = 20)
```

Arguments

`W`	a w-correlation matrix
`cuts`	an integer that is the number of levels the range of w-correlation values will be divided into

Note

refer to `fwcor` for an example on how to run this function starting from a w-correlation matrix

See Also

`fssa`, `freconstruct`, `fts`, `fwcor`

`[.fts` *Indexing into Functional Time Series*

Description

A method that lets you index into a functional time series (`fts`).

Usage

```
## S3 method for class 'fts'
Y[i = "index"]
```

Arguments

`Y`	an object of class `fts`
`i`	index

Value

an object of class `fts`

Note

can use ':' as an operator to specify a range of indices

See Also

`fts`

Examples

```
## Not run:
require(fda)
require(Rfssa)
data(Callcenter) # Read data
u=seq(0,1,length.out=240) # Define domain of functional data
d=12 # number of basis elements
basis=create.bspline.basis(rangeval = c(0,1),nbasis = d) # create basis object
smooth.calls=smooth.basis(u, matrix(nrow=240,ncol=365,Callcenter$calls), basis)
Y=fts(smooth.calls$fd) # create functional time series
Yind=Y[4:8] # take only the 4th through 8th functions
plot(Yind)
Yminus=Y[4:8]-Y[14:18] # subtract functions from each other
plot(Yminus)

## End(Not run)
```

Index