Package 'Rfssa'

October 12, 2022

Type Package

Title Functional Singular Spectrum Analysis

Version 2.1.0

Maintainer Hossein Haghbin <haghbin@pgu.ac.ir>

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

Description
Methods and tools for implementing functional singular spectrum analysis and related techniques.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.0

Imports Rcpp, fda, lattice, plotly, shiny, Rssa, ggplot2, tibble, methods, RSpectra, httr, markdown

LinkingTo Rcpp, RcppArmadillo, RcppEigen,

Suggests testthat (>= 3.0.0), knitr

Depends R (>= 4.0.0), dplyr

Config/testthat/edition 3

NeedsCompilation yes

Author Hossein Haghbin [aut, cre] (<https://orcid.org/0000-0001-8416-2354>),
Jordan Trinka [aut],
Seyed Morteza Najibi [aut],
Mehdi Maadooliat [aut] (<https://orcid.org/0000-0002-5408-2676>)

Repository CRAN

Date/Publication 2022-09-08 21:12:54 UTC

R topics documented:

*.fts 2
+.fts 3
-.fts 4
Callcenter 5

cor.fts . . . 6
eval.fts . . . 7
fforecast . . . 9
freconstruct . . . 12
fssa . . . 13
fts . . . 17
fwcor . . . 18
Jambi . . . 19
launchApp . . . 20
load_github_data . . . 21
Montana . . . 22
plot.fssa . . . 22
plot.fts . . . 24
Rfssa . . . 26
wplot . . . 27
[.fts . . . 28

Index **30**

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

Description

A method for functional time series (`fts`) pointwise multiplication and fts-scalar multiplication. Note that if the fts is multivariate then a vector of numerics may be provided allowing for multiplication of different variables by different scalars. For example, multivariate fts-numeric multiplication follows the form of `Y*c(1,2)` if `Y` is a bivariate fts.

Usage

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

Arguments

`Y1`	An object of class `fts` or numeric.
`Y2`	An object of class `fts` or numeric.

Value

An object of class `fts`.

See Also

`fts`

Examples

```
## Not run:
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
u <- seq(0, 1, length.out = 240) # Define domain of functional data
d <- 22 # number of basis elements
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u))
plot(Y)
Ytimes <- Y * Y # multiply the functional time series by itself
plot(Ytimes)
Ytimes2 <- Y * 2 # multiply every term in the fts by 2
plot(Ytimes2)

## End(Not run)
```

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

Description

A method for functional time series (`fts`) addition and fts-scalar addition. Note that if the fts is multivariate then a vector of numerics may be provided allowing for addition of different scalars to different variables. For example, multivariate fts-numeric addition follows the form of `Y+c(1,2)` if `Y` is a bivariate fts.

Usage

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

Arguments

`Y1`	An object of class `fts` or numeric.
`Y2`	An object of class `fts` or numeric.

Value

An object of class `fts`.

See Also

`fts`

Examples

```
## Not run:
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
u <- seq(0, 1, length.out = 240) # Define domain of functional data
d <- 22 # number of basis elements
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u))
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 for functional time series (`fts`) subtraction and fts-scalar subtraction. Note that if the fts is multivariate then a vector of numerics may be provided allowing for subtraction of different scalars from different variables. For example, multivariate fts-numeric subtraction follows the form of `Y-c(1,2)` if `Y` is a bivariate fts.

Usage

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

Arguments

`Y1`	An object of class `fts` or numeric.
`Y2`	An object of class `fts` or numeric.

Value

An object of class `fts`.

See Also

`fts`

Examples

```
## Not run:
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
u <- seq(0, 1, length.out = 240) # Define domain of functional data
d <- 22 # number of basis elements
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u))
plot(Y)
Yminus <- Y[1:4] - Y[5:8] # subtract the functional time series to itself
plot(Yminus)
Yminus2 <- Y - 2 # subtract 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 of 6 minutes. The data is hosted on GitHub and `load_github_data` may be used to load the data.

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.

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 univariate 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(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Jambi.RData")
## 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
}
d <- 11
u <- seq(0, 1, length.out = 512)
Y_1 <- Rfssa::fts(list(D0_NDVI), list(list(d, "bspline")), list(u))
Y_2 <- Rfssa::fts(list(D0_EVI), list(list(d, "bspline")), list(u))
out <- cor.fts(Y_1, Y_2)
print(out)

## End(Not run)
```

`eval.fts` *Functional Time Series Evaluation*

Description

This is a method that can be used to evaluate a functional time series (`fts`) object at new, specified grid points.

Usage

```
eval.fts(Y, grid, ud_basis = NULL)
```

Arguments

`Y`	An object of class `fts` to be evaluated.
`grid`	A list of length p where each entry is a numeric, matrix, list, or `NULL` specifying the grid to evaluate the variable over. If the entry is `NULL`, then the variable is evaluated over the original grid.
`ud_basis`	A list of length p where each entry is a matrix specifying the user-defined basis to be used for the evaluation of a variable. Note that the specified basis should match with the basis used to estimate the `fts` object in type and dimension.

Value

A list of length p of matrices or three-dimensional arrays where each list entry type depends on whether the variable is observed over a one or two-dimensional domain.

Examples

```
## Not run:
## Evaluate a univariate FTS at less grid points
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
## Define functional objects
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
N <- ncol(D)
```

```
time <- substr(seq(ISOdate(1999, 1, 1), ISOdate(1999, 12, 31), by = "day"),1,10)
K <- nrow(D)
u <- seq(0, K, length.out = K)
d <- 22 # Optimal Number of basis elements
## Define functional time series
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u),time)
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
ylabels = "Sqrt of Call Numbers",type="line",
xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
 list(c(1,60,120,180,240)))
u <- seq(0,K,length.out = 48)
D = eval.fts(Y=Y,grid = list(u))
Y <- Rfssa::fts(D, list(list(d, "bspline")), list(u))
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
ylabels = "Sqrt of Call Numbers",type="line",
xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
 list(c(1,12,24,36,48)))

## Evaluate a multivariate FTS at more grid points

require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Montana.RData")
Temp <- Montana$Temp
NDVI <- Montana$NDVI
d_temp <- 11
d_NDVI <- 13
## Define functional time series
Y <- Rfssa::fts(
  list(Temp / sd(Temp), NDVI), list(
    list(d_temp, "bspline"),
    list(d_NDVI, d_NDVI, "bspline", "bspline")
  ),
  list(c(0, 23), list(c(1, 33), c(1, 33))),
time=colnames(Temp))
# Plot the first 100 observations
plot(Y[1:100],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Standardized Temperature (\u00B0C)", "Latitude"),
     zlabels = c("", "NDVI"),
     mains = c("Temperature Curves", "NDVI Images"), color_palette = "RdYlGn",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
     c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
       list(c(1,6,12,18,24),c(1,33)),
       yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
       list(NA,c(1,33))
)

grid_temp = seq(0,23,length.out = 100)
u_NDVI = seq(1,33,length.out = 100)
v_NDVI = seq(1,33,length.out = 100)
grid_NDVI = list(u_NDVI,v_NDVI)
```

```
D = eval.fts(Y = Y, grid = list(grid_temp, grid_NDVI))

Y <- Rfssa::fts(
  D, list(
    list(d_temp, "bspline"),
    list(d_NDVI,d_NDVI,"bspline","bspline")
  ),
  list(grid_temp, grid_NDVI)
)

plot(Y[1:100],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Standardized Temperature (\u00B0C)", "Latitude"),
     zlabels = c("", "NDVI"),
     mains = c("Temperature Curves", "NDVI Images"), color_palette = "RdYlGn",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
     c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
       list(c(1,24,48,72,100),c(1,33)),
       yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
       list(NA,c(1,33))
)


## End(Not run)
```

`fforecast` *Functional Singular Spectrum Analysis Recurrent Forecasting and Vector Forecasting*

Description

This function performs functional singular spectrum analysis (FSSA) recurrent forecasting (FSSA R-forecasting) or vector forecasting (FSSA V-forecasting) of univariate or multivariate functional time series (`fts`) observed over a one-dimensional domain.

Usage

```
fforecast(U, groups = list(c(1)), h = 1, method = "recurrent", tol = 10^-3)
```

Arguments

`U`	An object of class `fssa` that holds the decomposition.
`groups`	A list of numeric vectors where each vector includes indices of elementary components of a group used for reconstruction and forecasting.
`h`	An integer that specifies the forecast horizon.
`method`	A character string specifying the type of forecasting to perform either `"recurrent"` or `"vector"`.

`tol` A double specifying the amount of tolerated error in the approximation of the matrix that corresponds with the operator formed using a Neumann series leveraged in both forecasting algorithms; see Trinka et. al. (2021) for more details.

Value

A list of objects of class `fts` where each fts corresponds to a forecasted group.

Examples

```
## Not run:
# FSSA Forecasting
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
## Define functional objects
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
N <- ncol(D)
time <- substr(seq(ISOdate(1999, 1, 1), ISOdate(1999, 12, 31), by = "day"),1,10)
K <- nrow(D)
u <- seq(0, K, length.out = K)
d <- 22
## Define functional time series
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u),time)
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
ylabels = "Sqrt of Call Numbers",type="line",
xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
 list(c(1,60,120,180,240)))
## Perform FSSA decomposition
L <- 28
U <- fssa(Y, L)
groups <- list(1, 2:3, 4:5, 6:7, 1:7)
## Perform FSSA R-forecast and FSSA V-forecast
pr_R <- fforecast(U = U, groups = groups, h = 30,
method = "recurrent", tol = 10^-3)
plot(pr_R[[1]], mains = "Call Center Mean Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_R[[2]], mains = "Call Center First Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_R[[3]], mains = "Call Center Second Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_R[[4]], mains = "Call Center Third Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
```

```
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_R[[5]], mains = "Call Center Extracted Signal Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))

pr_V <- fforecast(U = U, groups = groups, h = 30, method = "vector", tol = 10^-3)
plot(pr_V[[1]], mains = "Call Center Mean Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_V[[2]], mains = "Call Center First Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_V[[3]], mains = "Call Center Second Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_V[[4]], mains = "Call Center Third Periodic Component Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(pr_V[[5]], mains = "Call Center Extracted Signal Forecast",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))

# MFSSA Forecasting
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Jambi.RData")
## 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
D <- list(D0_NDVI, D0_EVI)
B <- list(list(d, "bspline"),list(d + 4, "fourier"))
U <- list(c(0, 1), c(0, 1))
Y <- Rfssa::fts(D, B, U,time)
plot(Y,mains = c("NDVI","EVI"),xlabels = c("NDVI","EVI"),
     ylabels = c("NDVI Density","EVI Density"),type="line",
     xticklabels = list(c("0","1"),c("0","1")),xticklocs =
       list(c(0,512),c(0,512)))
U <- fssa(Y = Y, L = 45)
groups <- list(c(1:4), c(1), c(2:3), c(4))
pr_R <- fforecast(U = U, groups = groups, h = 1, method = "recurrent")
plot(pr_R[[1]])
plot(pr_R[[2]])
plot(pr_R[[3]])
plot(pr_R[[4]])

pr_V <- fforecast(U = U, groups = groups, h = 1, method = "vector")
plot(pr_V[[1]])
plot(pr_V[[2]])
plot(pr_V[[3]])
plot(pr_V[[4]])

## End(Not run)
```

`freconstruct` *Reconstruction Stage of Functional Singular Spectrum Analysis*

Description

This is a function for reconstructing univariate or multivariate 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, groups = as.list(1L:10L))
```

Arguments

U	An object of class fssa.
groups	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). The algorithm (ufssa or mfssa) is chosen based on whether the supplied input is a univariate or multivariate functional time series (fts) object. The type parameter can also be set to "mfssa" if the user wishes to perform ufssa of a univariate fts object using the mfssa code. Also note that the variables of the fts maybe observed over different dimensional domains where the maximum dimension currently supported is two.

Usage

```
fssa(Y, L = NA, ntriples = 20, type = "ufssa")
```

Arguments

Y	An object of class fts.
L	A positive integer giving the window length.
ntriples	A positive integer specifying the number of eigentriples to calculate in the decomposition.
type	A string indicating which type of fssa to perform. Use type="ufssa" to perform univariate fssa (default for univariate fts). Use type="mfssa" to perform multivariate fssa (default for multivariate fts).

Value

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

`values`	A numeric vector of eigenvalues.
`L`	The specified window length.
`N`	The length of the functional time series.
`Y`	The original functional time series.

Examples

```
## Not run:
## Univariate FSSA Example on Callcenter data
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
## Define functional objects
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
N <- ncol(D)
time <- substr(seq(ISOdate(1999, 1, 1), ISOdate(1999, 12, 31), by = "day"),1,10)
K <- nrow(D)
u <- seq(0, K, length.out = K)
d <- 22 # Optimal Number of basis elements
## Define functional time series
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u),time)
Y
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
ylabels = "Sqrt of Call Numbers",type="line",
xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
 list(c(1,60,120,180,240)))
## 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, 1:7)
Q <- freconstruct(U, gr)
plot(Q[[1]], mains = "Call Center Mean Component",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Q[[2]], mains = "Call Center First Periodic Component",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
```

```
plot(Q[[3]], mains = "Call Center Second Periodic Component",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Q[[4]], mains = "Call Center Third Periodic Component",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Y, mains = c("Call Center Data Line Plot"),
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Q[[5]], mains = "Call Center Extracted Signal",
xlabels = "Time (6 minutes aggregated)",
     ylabels = "Sqrt of Call Numbers",type="line",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))

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

plot(Q[[1]],xlabels = "Time (6 minutes aggregated)",
     zlabels = "Sqrt of Call Numbers",type="3Dsurface", tlabels = "Date",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Q[[2]], mains = "Call Center First Periodic Component",
xlabels = "Time (6 minutes aggregated)",
     zlabels = "Sqrt of Call Numbers",type="heatmap",
     tlabels = "Date",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))
plot(Q[[3]],xlabels = "Time (6 minutes aggregated)",
     zlabels = "Sqrt of Call Numbers",type="3Dline",
     tlabels = "Date",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00")),xticklocs =
       list(c(1,60,120,180,240)))

## Multivariate FSSA Example on bivariate intraday
## temperature curves and smoothed images of vegetation
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Montana.RData")
Temp <- Montana$Temp
NDVI <- Montana$NDVI
d_temp <- 11
d_NDVI <- 13
## Define functional time series
Y <- Rfssa::fts(
  list(Temp / sd(Temp), NDVI), list(
    list(d_temp, "bspline"),
    list(d_NDVI, d_NDVI, "bspline", "bspline")
  ),
```

```
  list(c(0, 23), list(c(1, 33), c(1, 33))),
colnames(Temp))
# Plot the first 100 observations
plot(Y[1:100],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Normalized Temperature (\u00B0C)", "Latitude"),
     zlabels = c("", "NDVI"),
     mains = c("Temperature Curves", "NDVI Images"), color_palette = "RdYlGn",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
     c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
       list(c(1,6,12,18,24),c(1,33)),
       yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
       list(NA,c(1,33))
)
plot(Y, types = c("3Dline", "heatmap"), vars = c(1, 1))
plot(Y, types = "heatmap", vars = 2)
plot(Y, vars = c(2, 1))
L <- 45
## Multivariate functional singular spectrum analysis
U <- fssa(Y, L)
plot(U, type = "values", d = 10)
plot(U, type = "vectors", d = 4)
plot(U, type = "lheats", d = 4)
plot(U, type = "lcurves", d = 4, vars = c(1))
plot(U, type = "paired", d = 6)
plot(U, type = "wcor", d = 10)
plot(U, type = "periodogram", d = 4)
# Reconstruction of multivariate fts observed over different dimensional domains
Q <- freconstruct(U = U, groups = list(c(1), c(2:3), c(4)))
# Plotting reconstructions to show accuracy
plot(Q[[1]],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Normalized Temperature (\u00B0C)", "Latitude"),
     zlabels = c("", "NDVI"),
     mains = c("Temperature Curves Mean", "NDVI Images Mean"), color_palette = "RdYlGn",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
     c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
       list(c(1,6,12,18,24),c(1,33)),
       yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
       list(NA,c(1,33))) # mean
plot(Q[[2]],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Normalized Temperature (\u00B0C)", "Latitude"),
     zlabels = c("", "NDVI"),
    mains = c("Temperature Curves Periodic", "NDVI Images Periodic"), color_palette = "RdYlGn",
     xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
     c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
       list(c(1,6,12,18,24),c(1,33)),
       yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
       list(NA,c(1,33))) # periodic
plot(Q[[3]],
     xlabels = c("Time", "Longitude"),
     ylabels = c("Normalized Temperature (\u00B0C)", "Latitude"),
```

```
      zlabels = c("", "NDVI"),
     mains = c("Temperature Curves Trend", "NDVI Images Trend"), color_palette = "RdYlGn",
      xticklabels = list(c("00:00","06:00","12:00","18:00","24:00"),
      c("113.40\u00B0 W", "113.30\u00B0 W")),xticklocs =
        list(c(1,6,12,18,24),c(1,33)),
        yticklabels = list(NA,c("48.70\u00B0 N", "48.77\u00B0 N")),yticklocs =
        list(NA,c(1,33))) # trend

## End(Not run)
```

---

fts *Functional Time Series Class*

Description

This function is used to create functional time series (fts) objects from lists of discretely sampled data, basis specifications, and grid elements which provide the domain that each variable is observed over. Each variable is assumed to be observed over a regular and equidistant grid. In addition, each variable in the fts is assumed to be observed over a one or two-dimensional domain.

Usage

```
fts(X, B, grid, time = NULL)
```

Arguments

X	A list of length p where p is a positive integer specifying the number of variables observed in the fts. Each entry in the list should be a matrix or an array.
B	A list of length p. Each entry in the list should be either a matrix specifying the basis for each variable or each list entry should be a list specifying the number of basis elements and desired basis type to be used in the smoothing process.
grid	A list of length p. Each entry in the list should either be a numeric or a list of numeric elements depending on the dimension of the domain the variable is observed over.
time	A character vector where each entry specifies the time an observation is made.

Note

Refer to fssa for examples on how to run this function.

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, groups)
```

Arguments

U	An object of class fssa.
groups	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 FSSA Example on Callcenter data
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
## Define functional objects
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
N <- ncol(D)
time <- substr(seq(ISOdate(1999, 1, 1), ISOdate(1999, 12, 31), by = "day"),1,10)
K <- nrow(D)
u <- seq(0, K, length.out = K)
d <- 22 # Optimal Number of basis elements
## Define functional time series
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u),time)
Y
plot(Y, mains = c("Sqrt of Call Center Data"))
## Univariate functional singular spectrum analysis
L <- 28
U <- fssa(Y, L)
ufwcor <- fwcor(U = U, groups = list(1, 2, 3))
```

```
wplot(W = ufwcor)

## Multivariate W-Correlation Example on Bivariate Satelite Image Data
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Jambi.RData")
## 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
D <- list(D0_NDVI, D0_EVI)
B <- list(list(d, "bspline"), list(d + 4, "fourier"))
U <- list(c(0, 1), c(0, 1))
Y <- Rfssa::fts(D, B, U, time)
plot(Y)
U <- fssa(Y = Y, L = 45)
L <- 45
mfwcor <- fwcor(U = U, groups = 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. The data is hosted on GitHub and `load_github_data` may be used to load the data.

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)
```

---

load_github_data *Load Data from GitHub Repositories*

Description

This function was found in https://stackoverflow.com/questions/24846120/importing-data-into-r-rdata-from-github and can be used to load .RData files from GitHub repositories. This function can be used to load the Callcenter, Jambi, and Montana datasets from the Rfssa package hosted by GitHub at https://github.com/haghbinh/Rfssa. It was found that hosting such datasets on GitHub and not including them in the Rfssa R package saved a significant amount of space.

Usage

```
load_github_data(github_data_url)
```

Arguments

github_data_url	The GitHub url of the dataset.

Value

A dataset specified by a GitHub url.

Examples

```
## Not run:
# Loading different datasets from the Rfssa repository hosted by GitHub.
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Jambi.RData")
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Montana.RData")

## End(Not run)
```

Montana *Montana Intraday Temperature Curves and NDVI Images Data Set*

Description

This data set contains the intraday hourly temperature curves measured in degrees celcius and normalized difference vegetation index (NDVI) image data where both types of data are recorded near Saint Mary, Montana, USA. The NDVI images are taken of a region located between longitudes of 113.30 degrees West and 113.56 degrees west and latitudes of 48.71 degrees North and 48.78 degrees North. The intraday temperature curves are available for download from Diamond et al. (2013) and the NDVI images were attained leveraging resources provided by Tuck et al. (2014). For each recorded intraday temperature curve, an NDVI image was recorded on the same day, every 16 days, starting January 1, 2008 and ending September 30, 2013. With the the threat of global warming damaging various ecosystems, the goal of the study was to analyze trends in the temperature and to investigate how changes in temperature effects the amount of vegetation in the region. We discovered that leveraging both types of variables in a multivariate analysis revealed a stronger signal extraction result and more informative patterns. The data is hosted on GitHub and `load_github_data` may be used to load the data.

Format

A list which contains a 24 by 133 matrix of discrete samplings of intraday hourly temperature curves and an array that is 33 by 33 by 133 where one 33 by 33 slice of the array is an NDVI image.

References

1. Diamond, H. J., Karl, T., Palecki, M. A., Baker, C. B., Bell, J. E., Leeper, R. D., Easterling, D. R., Lawrimore, J. H., Meyers, T. P., Helfert, M. R., Goodge, G., and Thorne, P.W. (2013). U.S. climate reference network after one decade of operations: status and assessment. https://www.ncdc.noaa.gov/crn/qcdatasets.html. Last accessed April 2020.
2. Tuck, S. L., Phillips, H. R., Hintzen, R. E., Scharlemann, J. P., Purvis, A., and Hudson, L. N. (2014). MODISTools – downloading and processing MODIS remotely sensed data in R. Ecology and Evolution, 4(24):4658–4668.

See Also

`fssa`

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",
  vars = NULL,
  ylab = NA,
  main = NA,
  color_palette = "RdYlBu",
  reverse_color_palette = FALSE,
  ...
)
```

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 of the eigenfunctions which can be used for fts variables observed over one or two-dimensional domains (useful for the detection of meaningful patterns) • "periodogram" - periodogram plot (useful for the detecting the frequencies of oscillations in functional data).
vars	A numeric specifying the variable number (can be used in plotting MFSSA "lheats" or "lcurves").
ylab	The character vector of name of variables.

main	The main plot title
color_palette	A string specifying the color palette that is offered by the ggplot2 package to be used when plotting left singular functions corresponding with fts variables observed over two-dimensional domains.
reverse_color_palette	A boolean specifying if the color palette scale should be reversed.
...	Arguments to be passed to methods, such as graphical parameters.

Note

See fssa examples.

See Also

fssa, plot.fts

---

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,
  vars = NULL,
  types = NULL,
  subplot = TRUE,
  mains = NULL,
  ylabels = NULL,
  xlabels = NULL,
  tlabels = NULL,
  zlabels = NULL,
  xticklabels = NULL,
  xticklocs = NULL,
  yticklabels = NULL,
  yticklocs = NULL,
  color_palette = "RdYlBu",
  reverse_color_palette = FALSE,
  ...
)
```

Arguments

x	An object of class fts.
vars	A numeric specifying which variables in the fts to plot. The default is to plot all variables in succession. Note as well that variable indices may be repeated.
types	A tuple of strings specifying the types of plots to be displayed where possible types for fts variables observed over a one-dimensional domain are: • "line" - plot the fts elements in a line plot (default) • "heatmap" - plot the fts elements in a heat map which can be used for variables observed over one or two-dimensional domains • "3Dsurface" - plot the fts elements as a surface • "3Dline" - plot the fts elements in a three-dimensional line plot. The current plot type supported for fts variables observed over a two-dimensional domain is "heatmap". Also note that the same variable may be plotted several times using many different type options.
subplot	A logical specifying whether or not line plots should be plotted in a subplot or not. The default is TRUE and if any other plot type is provided, the value is switched to FALSE.
mains	A tuple of strings providing the the main titles of each plot.
ylabels	A tuple of strings providing the the y-axis titles of each plot.
xlabels	A tuple of strings providing the the x-axis titles of each plot.
tlabels	A tuple of strings providing the the time-axis titles of each plot.
zlabels	A tuple of strings providing the the z-axis titles of each plot.
xticklabels	A list of character vectors where each entry specifies the tick labels for the domain of the functions.
xticklocs	A list of numerics where each entry specifies the position of the tick labels for the domain of the functions.
yticklabels	A list of character vectors where each entry specifies the tick labels for the domain of the functions.
yticklocs	A list of numerics where each entry specifies the position of the tick labels for the domain of the functions.
color_palette	A string specifying the color palette that is offered by the ggplot2 package to be used when plotting fts variables observed over two-dimensional domains.
reverse_color_palette	A boolean specifying if the color palette scale should be reversed.
...	arguments to be passed to methods, such as graphical parameters.

Note

For examples, see fssa

Rfssa *Rfssa: A Package for Functional Singular Spectrum Analysis and Related Methods.*

Description

The Rfssa package provides the collection of necessary functions to implement functional singular spectrum analysis (FSSA)-based methods for analyzing univariate and multivariate functional time series (FTS). Univariate and multivariate FSSA are novel, non-parametric methods used to perform decomposition and reconstruction of univariate and multivariate FTS respectively. In addition, the FSSA-based routines may be performed on FTS whose variables are observed over a one or two-dimensional domain. Finally, one may perform FSSA recurrent or FSSA vector forecasting of univariate or multivariate FTS observed over one-dimensional domains. Forecasting of FTS whose variables are observed over domains of dimension greater than one is under development.

Details

The use of the package begins by defining an `fts` object by providing the constructor with the raw data, basis specifications, and grid specifications. We note that the FTS object may be univariate or multivariate and variables may be observed over one (curves) or two-dimensional (images) domains. Validity checking of the S4 object constructor inputs is included to help guide the user. The user may leverage the `plot.fts` method to visualize the `fts` object. A variety of plotting options are available for variables observed over a one-dimensional domain and a visuanimation is offered for variables observed over a two-dimensional domain. Next, the user provides the `fts` object and a chosen lag parameter to the FSSA routine (`fssa`) to obtain the decomposition. We note that the decomposition function leverages the RSpectra and RcppEigen R packages, and the Eigen C++ package to speed up the routine. The `plot.fssa` method may be used to visualize the results of the decomposition and to choose an appropriate grouping of the eigentriples for reconstruction (`freconstruct`) or forecasting (`fforecast`). The `freconstruct` routine can be used to reconstruct a list of `fts` objects specified by the grouping while the `fforecast` function returns a list of `fts` objects that contain predictions of the signals specified by the grouping. We note that when forecasting is performed, usually the user specifies one group that captures the assumed deterministic, extracted signal that is found within the FTS and all other modes of variation are excluded. We also note that currently, forecasting only supports FTS whose variables are observed over a one-dimensional domain with two-dimensional domain forecasting to be added in the future.

Other functionalities offered by the package include:

- FTS arithmetic - Allows the user to perform FTS-FTS arithmetic and FTS-scalar arithmetic (such as addition, subtraction, etc.).
- `eval.fts` - Allows the user to evaluate the FTS object over a new specified grid.
- `load_github_data` - Allows the user to load any .RData file hosted on GitHub including the `Callcenter`, `Jambi`, and `Montana` datasets.
- `fwcor` - Returns the weighted correlation matrix corresponding to the decomposition of an FTS.
- `cor.fts` - Returns the correlation between two `fts` objects.

- launchApp - Launches the built-in R Shiny app that can be used to interactively explore the FSSA-based routines on various datasets.

The first update we include in this version of the Rfssa R package, is the eval.fts method used to evaluate an fts object over a new, specified grid. We updated the plot.fts method to allow for custom tick labels and new choices in visuanimation colors (for variables observed over two-dimensional domains) that are offered by the ggplot2 package. We have also updated the plot.fssa method to allow for new choices in visuanimation colors offered by the ggplot2 package when plotting left singular functions that correspond with variables observed over two-dimensional domains. A user may now specify a character vector that contains the time when each observation is made when building an fts object and we improved various plot fonts for readability. Finally, we include many other small updates that further improve plotting quality, code readability, documentation improvements, and other details that add to the professionalism of the package.

References

Haghbin, H., Morteza Najibi, S., Mahmoudvand, R., Trinka, J., and Maadooliat, M. (2021). Functional singular spectrum analysis. Stat. e330 STAT-20-0240.R1.

Trinka J., Haghbin H., Maadooliat M. (Accepted) Multivariate Functional Singular Spectrum Analysis: A Nonparametric Approach for Analyzing Multivariate Functional Time Series. In: Bekker A., Ferreira, J., Arashi M., Chen D. (eds) Innovations in Multivariate Statistical Modeling: Navigating Theoretical and Multidisciplinary Domains. Emerging Topics in Statistics and Biostatistics. Springer, Cham.

Trinka J. (2021) Functional Singular Spectrum Analysis: Nonparametric Decomposition and Forecasting Approaches for Functional Time Series [Doctoral dissertation, Marquette University]. ProQuest Dissertations Publishing.

Trinka, J., Haghbin, H., and Maadooliat, M. (2021). Functional time series forecasting: Functional singular spectrum analysis approaches. Version 4 retrieved from https://arxiv.org/abs/2011. 13077.

See Also

fssa, freconstruct, fforecast 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, main = NA)
```

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.
main	A character that is the main title of the plot.

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

An indexing method for functional time series (fts).

Usage

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

Arguments

Y	An object of class fts.
i	The index value.

Value

An object of class fts.

See Also

fts

Examples

```
## Not run:
require(Rfssa)
load_github_data("https://github.com/haghbinh/Rfssa/blob/master/data/Callcenter.RData")
D <- matrix(sqrt(Callcenter$calls), nrow = 240)
u <- seq(0, 1, length.out = 240) # Define domain of functional data
d <- 22 # number of basis elements
Y <- Rfssa::fts(list(D), list(list(d, "bspline")), list(u))
plot(Y)
plot(Y[10:15])

## End(Not run)
```

Index

*.fts, 2
+.fts, 3
-.fts, 4
[.fts, 28

Callcenter, 5, *26*
cor.fts, 6, *26, 27*

eval.fts, 7, *26, 27*

fforecast, 9, *26, 27*
freconstruct, 12, *18*, *26–28*
fssa, *6*, *9*, *12, 13*, 13, *17, 18*, *20*, *22–28*
fts, *2–4*, *6, 7*, *9, 10*, *12, 13*, *17*, 17, *18*, *20*, *23–28*
fwcor, 18, *26–28*

Jambi, 19, *26*

launchApp, 20, *27*
load_github_data, *5*, *19*, 21, *22*, *26*

Montana, 22, *26*

plot.fssa, 22, *26, 27*
plot.fts, *24*, 24, *26, 27*

Rfssa, 26

wplot, *18*, *27*, 27