Package ‘Rfssa’

January 10, 2024

Type Package
Title Functional Singular Spectrum Analysis
Version 3.1.0
Maintainer Hossein Haghbin <haghbin@pgu.ac.ir>
URL https://github.com/haghbin/Rfssa
Description Methods and tools for implementing functional singular spectrum analysis and related techniques.
License GPL-3
Encoding UTF-8
LazyData true
LazyLoad true
RoxygenNote 7.2.3
Imports Rcpp, fda, lattice, plotly, shiny, Rssa, ggplot2, tibble,
RSpectra, rainbow, ftsa, dplyr, markdown
LinkingTo Rcpp, RcppArmadillo, RcppEigen,
Suggests knitr
Depends R (>= 4.0.0)
Repository CRAN
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>)
Date/Publication 2024-01-10 17:13:12 UTC

R topics documented:

*.funts ................................................................. 2
+.funts .................................................................... 3
Scalar Multiplication of Functional Time Series (funts)

Description

Perform scalar multiplication of a Functional Time Series (funts) object by either another funts object or a scalar value.

Usage

```r
## S3 method for class 'funts'
obj1 * obj2
```

Arguments

- `obj1` an object of class `funts` or a scalar value.
- `obj2` an object of class `funts` or a scalar value.
Details

This function allows element-wise multiplication of a Functional Time Series (funts) object by another funts object or scalar value.

Value

An object of class funts representing the result of scalar multiplication.

See Also

funts

Description

A method for functional time series (funts) addition and funts-scalar addition.

Usage

```r
# S3 method for class 'funts'
obj1 + obj2 = NULL
```

Arguments

- `obj1`: an object of class funts or numeric.
- `obj2`: an object of class funts or numeric.

Value

- an object of class funts.

See Also

funts
### -.funts

*Subtract 'funts' Objects or a 'funts' Object and a Scalar*

#### Description

This function allows subtraction between two ‘funts’ (functional time series) objects or between a ‘funts’ object and a scalar. It returns the resulting difference.

#### Usage

```r
## S3 method for class 'funts'
obj1 - obj2 = NULL
```

#### Arguments

- `obj1`: an object of class ‘funts’, representing the first ‘funts’ object.
- `obj2`: an object of class ‘funts’ or a scalar.

#### See Also

- `funts`

#### Examples

```r
## Not run:
data("Callcenter")
y <- Callcenter
print(1 - y)
## End(Not run)
```

### as.funts

*Convert Object to a funts*

#### Description

This function allows you to convert various types of objects into a functional time series (`funts`) object.

#### Usage

```r
as.funts(obj, basis = NULL)
```
Arguments

**obj**
the object to be converted. It can be an object of class `fd` (functional data) of the package `fda`, `fts` (functional time series) of the package `rainbow` types.

**basis**
an optional argument specifying the basis to be used for the resulting `funts` object when converting from `fts` objects. If not provided, a B-spline basis will be created by default.

Value

An object of class `funts`.

Note

Only objects of class `fd` (functional data) and `fts` (functional time series) can be converted to a `funts` object. Other types will result in an error.

See Also

`funts`, `create.bspline.basis`

Examples

```r
require(rainbow)
class(Australiasmoothfertility)
x_funts1 <- as.funts(Australiasmoothfertility)
plot(x_funts1, main = "Australians Fertility")

require(fda)
bs <- create.bspline.basis(rangeval = c(15, 49), nbasis = 13)
fd_obj <- smooth.basis(argvals = Australiasmoothfertility$x, Australiasmoothfertility$y, bs)$fd

x_funts <- as.funts(fd_obj)
plotly_funts(x_funts,
             main = "Australians Fertility",
             ylab = "Fertility rate",
             xlab = "Age")
```

Callcenter

**Callcenter Dataset: Number of Calls for a Bank**

Description

This dataset represents a small call center for an anonymous bank (Brown et al., 2005). It provides detailed information about the exact times of calls that were connected to the center throughout the year 1999, from January 1 to December 31.
Format

A functional time series object of class ‘funts’ with the following fields:

- **time** the time index indicating when the calls occurred.
- **coefs** the coefficients corresponding to the B-spline basis functions.
- **basisobj** the basis functions used for the functional representation.
- **dimSupp** the dimension support of the functional data.

Details

The data have been converted into a functional time series using a B-spline basis system with 22 basis functions. The resulting dataset is stored as a functional time series object of class ‘funts’. You can load the raw data using the function `loadCallcenterData`. See `funts` for more details.

References


See Also

`loadCallcenterData`, `funts`

Examples

```r
## Not run:
# Load the Callcenter dataset
data("Callcenter")
## End(Not run)
```

---

**eval.funts**

Evaluate a Functional Time Series (funts) Object on a Given Grid

Description

This function allows you to evaluate a Functional Time Series (funts) object on a specified grid of argument values. The result is a list of matrices, each matrix corresponding to one dimension of the functional data.

Usage

`eval.funts(argvals, obj)`
Arguments

argvals a list or numeric vector specifying the grid points at which to evaluate the functional time series. For multivariate functional data, provide a list of grids corresponding to each dimension.

obj an object of class funts to be evaluated.

Details

The argvals argument can be a list of grids for multivariate functional data. The function handles both functional basis and empirical basis cases for evaluation. For empirical basis with irregular grids, a warning is issued as this feature is under development.

Value

A list of matrices, where each matrix represents the evaluated values of the functional data on the specified grid.

See Also

funts

Examples

data("Montana")
y <- Montana
u <- seq(0, 23, len = 4)
v <- seq(1, 33, len = 3)
grid <- list(u, list(v, v))
eval.funts(grid, y)

Description

Perform functional singular spectrum analysis (FSSA) recurrent forecasting (FSSA R-forecasting) or vector forecasting (FSSA V-forecasting) on univariate or multivariate functional time series (funts) observed over a one-dimensional domain.

Usage

fforecast(  
  U,  
  groups = as.list(1L:10L),  
  len = 1,  
  method = "recurrent",  
)
only.new = TRUE,
tol = NULL
)

Arguments

U an object of class fssa holding the decomposition.
groups a list of numeric vectors where each vector includes indices of elementary components of a group used for reconstruction and forecasting.
len integer, the desired length of the forecasted FTS.
method a character string specifying the type of forecasting to perform: - "recurrent" for FSSA R-forecasting. - "vector" for FSSA V-forecasting.
only.new logical, if 'TRUE' then only forecasted FTS are returned, whole FTS otherwise.
tol a double specifying the tolerated error in the approximation of the matrix used in forecasting algorithms.

Value

An object of class ‘fforecast’ which is a list of objects of class funts, where each one corresponds to a forecasted group.

Examples

## Not run:
data("Callcenter")
U <- fssa(Callcenter, L = 28)
groups <- list(1, 1:7)

## Perform FSSA R-forecast
pr_R <- fforecast(
  U = U, groups = groups, only.new = FALSE,
  len = 30, method = "recurrent"
)
plot(pr_R, group_index = 1)

plotly_funts(pr_R[[2]], main = "group = '1:7'")

## Perform FSSA V-forecast
pr_V <- fforecast(U = U, groups = groups, len= 30, method = "vector")
plot(pr_V, group_index = 1)

plotly_funts(pr_V[[2]], type = "3Dline", main = "group = '1:7'")

# Multivariate forecasting example:
data("Montana")
time <- Montana$time
grid <- list(0:23, list(1:33, 1:33))
montana <- eval.funts(Montana, argvals = grid)
montana[[2]] <- array(
  scale(montana[[2]][, , ]),
  center = min(montana[[2]][, , ]),
  scale = max(montana[[2]][, , ]) - min(montana[[2]][, , ])
),
  dim = c(33, 33, 133)
)
## Kernel density estimation of pixel intensity
NDVI <- matrix(NA, nrow = 512, ncol = 133)
for (i in 1:133) NDVI[, i] <- (density(montana[[2]][, , i], from = 0, to = 1)$y)
## Define functional objects
bs1 <- Montana$basis[[1]]

require(fda)
bs2 <- create.bspline.basis(nbasis = 15)
Y <- funts(X = list(montana[[1]], NDVI), basisobj = list(bs1, bs2),
           vnames = c("Temperature", "NDVI Density"),
           dnames = c("Time", "NDVI"),
           tname = "Date")

plotly_funts(Y,
             main = c("Temperature", "NDVI"),
             xticklocs = list(c(0, 6, 12, 18, 23), seq(1, 512, len = 9)),
             xticklabels = list(c(0, 6, 12, 18, 23), seq(0, 1, len = 9)))

U <- fssa(Y = Y, L = 45)
plotly_funts(U$lsingf[[1]])
plot(U$lsingf[[2]])

groups <- list(1, 1:3)
pr_R <- fforecast(U = U, groups = groups,
                  only.new = FALSE, len = 10, method = "recurrent")
plot(pr_R)
plotly_funts(pr_R[[2]], main = "Recurrent method, group = '1:3'")

pr_V <- fforecast(U = U, groups = groups, len = 10, method = "vector")
plot(pr_V, group_index = 1)
plotly_funts(pr_V[[2]], main = "Vector method, group = '1:3'")
## End(Not run)
Description

Calculate the bootstrap prediction interval for functional singular spectrum analysis (FSSA) forecasting predictions of univariate functional time series (`funts`) observed over a one-dimensional domain.

Usage

```r
fpredinterval(
  Y,
  O = floor(Y$N * 0.7),
  L = floor((Y$N * 0.7)/12),
  ntriples = 10,
  Bt = 100,
  h = 1,
  alpha = 0.05,
  method = "recurrent",
  tol = 10^-3
)
```

Arguments

- `Y`: an object of class `funts`.
- `O`: a positive integer specifying the training set size.
- `L`: a positive integer specifying the window length.
- `ntriples`: the number of eigentriples to use for forecasts.
- `Bt`: a positive integer specifying the number of bootstrap samples.
- `h`: an integer specifying the forecast horizon.
- `alpha`: a double (0 < alpha < 1) specifying the significance level.
- `method`: a character string: "recurrent" or "vector" forecasting.
- `tol`: a double specifying tolerated error in the approximation.

Value

a list of numeric vectors: point forecast, lower, and upper bounds.

Examples

```r
## Not run:
data("Callcenter")
pred_interval <- fpredinterval(
  Y = Callcenter, O = 310,
  L = 28, ntriples = 7, Bt = 10000, h = 3
)
# Plot the forecast and prediction interval using ggplot
df <- data.frame(
  x = 1:240,
)```
y = pred_interval$forecast,
lower = pred_interval$lower,
upper = pred_interval$upper
)
require(ggplot2)
# Create the ggplot
ggplot(df, aes(x = x, y = y)) +
geom_line(linewidth = 1.2) +
scale_x_continuous(
  name = "Time",
  breaks = c(1, 60, 120, 180, 240),
  labels = c("00:00", "06:00", "12:00", "18:00", "24:00"),
) +
scale_y_continuous(name = "Sqrt of Call Numbers") +
ggtitle("Prediction Intervals for Jan. 3, 2000") +
geom_ribbon(aes(ymin = lower, ymax = upper), fill = "darkolivegreen3", alpha = 0.3) +
theme_minimal()
## End(Not run)

---

**freconstruct**

*Reconstruction Stage of Functional Singular Spectrum Analysis*

**Description**

Reconstruct univariate or multivariate functional time series (funts) objects from functional singular spectrum analysis (fssa) objects, including Grouping and Hankelization steps. This function performs the reconstruction step for either univariate functional singular spectrum analysis (ufssa) or multivariate functional singular spectrum analysis (mfssa), depending on the input.

**Usage**

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

**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 funts that are reconstructed 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, funts

Examples

data("Callcenter")
L <- 28
U <- fssa(Callcenter, L)

# FSSA Reconstruction step:
gr <- list(1, 2:3, 4:5, 6:7, 1:7)
Q <- freconstruct(U, gr)
plot(Q[[1]],
     main = "Call Center Mean Component")
plot(Q[[2]],
     main = "Call Center First Periodic Component")

#--------------- Multivariate FSSA Example on bivariate -----------------------------
## temperature curves and smoothed images of vegetation
## Not run:
data("Montana")
L <- 45
U <- fssa(Montana, L)

# MFSSA Reconstruction step:
Q <- freconstruct(U = U, groups = list(1, 2, 3))
plotly_funts(Q[[1]],
             main = 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
plotly_funts(Q[[2]],
             main = 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]],
Description

This function performs the decomposition (embedding and functional SVD steps) for univariate (ufssa) or multivariate (mfssa) functional singular spectrum analysis based on the input data type. The input can be a univariate or multivariate functional time series (funts) object.

Usage

fssa(Y, L = Y$N/2, ntriples = 20, type = "ufssa")

Arguments

Y
an object of class funts.
L
a positive integer, the window length, the default is half of FTS length.
ntriples
a positive integer, the number of eigentriples for the decomposition.
type
a string indicating the type of FSSA: "ufssa" (default for univariate FTS) or "mfssa" (default for multivariate FTS).

Value

An object of class fssa, containing functional objects, eigenvalues, window length, and original data.

Examples

data("Callcenter")

# FSSA Decomposition step:
L <- 28
U <- fssa(Callcenter, L)
plot(U, type = "values", d = 10)
plot(U, type = "vectors", d = 4)
plot(U, type = "paired", d = 6)
plot(U, type = "lcurves", d = 4, vars = 1)
plot(U, type = "lheats", d = 4)
plot(U, type = "wcor", d = 10)
plotly_funts(U$Lsingf[[1]])
plot(U$Lsingf[[2]])
## Not run:
#------------------ Multivariate FSSA Example on bivariate ------------------
## temperature curves and smoothed images of vegetation
data("Montana")

# MFSSA Decomposition step:
L <- 45
U <- fssa(Montana, 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 = 1)
plot(U, type = "paired", d = 6)
plot(U, type = "periodogram", d = 4)
plot(U, type = "wcor", d = 10)
plotly_funts(U$Lsingf[[1]])
plot(U$Lsingf[[2]])

## End(Not run)

---

`funts`  
*Functional Time Series (funts) Class*

---

**Description**

The `funts` class is designed to encapsulate functional time series objects, including both univariate (FTS) and multivariate (MFTS) forms. It provides a versatile framework for creating and manipulating `funts` objects, accommodating various basis systems and dimensions.

**Usage**

```
funts(
  X,
  basisobj, 
  argval = NULL, 
  method = "data", 
  start = 1, 
  end = NULL, 
  vnames = NULL, 
  dnames = NULL, 
  tname = NULL
)
```

**Arguments**

- **X**: A matrix, three-dimensional array, or a list of matrix or array objects. When `method="data"`, it represents the observed curve values at discrete sampling
is.funts

Description

Check if an object is of class 'funts'

Usage

is.funts(obj)
launchApp

Arguments

obj The object to check.

Value

TRUE if the object is of class 'funts', FALSE otherwise.

Examples

data("Callcenter")
is.funts(Callcenter)

Description

This function launches a Shiny app to facilitate the understanding of univariate or multivariate Functional Singular Spectrum Analysis (fssa). The app enables users to perform univariate or multivariate FSSA on various data types, including simulated and real data provided by the server. Users can also upload their own data. The app supports simultaneous comparisons of different methods, such as multivariate vs. univariate FSSA, and allows users to select the number and types of basis elements used for estimating Functional Time Series (funts) objects. It offers plotting capabilities for both funts and fssa.

Usage

launchApp(type = "ufssa")

Arguments

Type of FSSA with options of type = "ufssa" or type = "mfssa".

Value

A shiny application object.

Examples

## Not run:
launchApp()

## End(Not run)
### length.funts 

**Length of Functional Time Series**

**Description**

Returns the length of a "funts" object.

**Usage**

```r
## S3 method for class 'funts'
length(x)
```

**Arguments**

- `x`: an object of class "funts".

**Examples**

```r
data("Callcenter")
length(Callcenter)
```

### loadAustinData 

**Load Austin Temperature Data from GitHub Repository**

**Description**

This function retrieves the Austin Temperature dataset from a GitHub repository hosted at https://github.com/haghbinh/dataset/Rfssa_dataset. Hosting datasets on GitHub rather than including them in the Rfssa R package conserves storage space. The Austin Temperature dataset contains intraday hourly temperature curves measured in degrees Celsius from January 1973 to July 2023, recorded once per month. The returned object is a raw dataset in 'list' format;

**Usage**

```r
loadAustinData()
```

**Format**

Containing two matrix objects:

- **Austin Temperature Data** A 24 by 607 matrix of discrete samplings of intraday hourly temperature curves, one day per month.

- **Utqiagvik Temperature Data** A 24 by 607 matrix of discrete samplings of intraday hourly temperature curves, one day per month.
References


Examples

```r
Austin_raw <- loadAustinData()
str(Austin_raw)
```

---

**loadCallcenterData**  
*Load Callcenter Data from GitHub Repository*

**Description**

This function retrieves the Callcenter dataset from the Rfssa_dataset repository on GitHub (https://github.com/haghbinh/dataset/Rfssa_dataset). The Callcenter dataset represents a small call center for an anonymous bank. It provides precise call timing data from January 1 to December 31, 1999. The data is aggregated into 6-minute intervals on each day. The returned object is a raw dataset in dataframe format; it is not a 'funts' class object. This raw data can then be further processed and converted into a 'funts' object named 'Callcenter'. See funts for more details on working with functional time series of class 'funts'.

**Usage**

```r
loadCallcenterData()
```

**Format**

A dataframe with 87,600 rows and 5 variables:

- **calls**: number of calls in a 6-minute aggregated interval.
- **u**: numeric vector indicating the aggregated interval.
- **Date**: date and time of call count recording.
- **Day**: weekday associated with Date.
- **Month**: month associated with Date.

**References**

See Also

funts

Examples

```r
require(fda)
# Load Callcenter data
Call_data <- loadCallcenterData()
D <- matrix(sqrt(Call_data$calls), nrow = 240)

# Define basis functions
bs1 <- create.bspline.basis(c(0, 23), 22)

Y <- funts(X = D, basisobj = bs1)
```

Description

This function retrieves the ‘Jambi’ dataset from a GitHub repository hosted at https://github.com/haghbinh/dataset/Rfssa_dataset.

The Jambi dataset contains normalized difference vegetation index (NDVI) and enhanced vegetation index (EVI) image data from NASA's MODerate-resolution Imaging Spectroradiometer (MODIS) with global coverage at a 250 m^2 resolution. The dataset covers the Jambi Province, Indonesia, known for various forested land uses, including natural forests and plantations. Monitoring land cover changes is crucial, especially in the context of forest exploitation and conservation efforts. Seasonal variations significantly impact long-term land cover changes. Data collection began on February 18, 2000, and continued until July 28, 2019, with data recorded every 16 days. This dataset is valuable for studying vegetative land cover changes in the region. The returned object is a raw dataset in 'list' format.

Usage

`loadJambiData()`

Format

A list containing two arrays, each with dimensions 33 by 33 by 448. One array represents NDVI image data, and the other represents EVI image data. The list also contains a date vector of length 448, specifying the capture date for each 33 by 33 image.

Source

[MODIS Product Information](https://lpdaac.usgs.gov/products/mod13q1v006/)
References


See Also

- The dataset object loaded by this function.

Examples

```r
Jambi_raw <- loadJambiData()
str(Jambi_raw)
```

loadMontanaData

Load Montana Data from GitHub Repository

Description

This function retrieves the Montana dataset from a GitHub repository hosted at https://github.com/haghbinh/dataset/Rfssa_dataset. Hosting datasets on GitHub rather than including them in the Rfssa R package conserves storage space. The Montana dataset contains intraday hourly temperature curves measured in degrees Celsius and normalized difference vegetation index (NDVI) image data. Both types of data are recorded near Saint Mary, Montana, USA. The NDVI images cover 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. For each recorded intraday temperature curve, an NDVI image was captured on the same day every 16 days, starting from January 1, 2008, and ending on September 30, 2013. The dataset is valuable for environmental analysis, especially in the context of studying the impact of temperature changes on vegetation. Combining both temperature and NDVI data can reveal more informative patterns and insights. The returned object is a raw dataset in ‘list’ format; This raw data can then be further processed and converted into a ‘funts’ object named ‘Montana’. See funts for more details on working with functional time series of class ‘funts’.

Usage

```r
loadMontanaData()
```

Format

A list containing two components:

**Temperature Data** A 24 by 133 matrix of discrete samplings of intraday hourly temperature curves.

**NDVI Images** An array with dimensions 33 by 33 by 133, where each 33 by 33 slice represents an NDVI image.
References


See Also

funts

Examples

```r
require(fda)
# Load Montana data
montana_data <- loadMontanaData()

# Extract variables
Temp <- montana_data$Temp
NDVI <- montana_data$NDVI

# Create a list for Montana data
Montana_Data <- list(Temp / sd(Temp), NDVI)

# Define basis functions
bs1 <- create.bspline.basis(c(0, 23), 11)
bs2 <- create.bspline.basis(c(1, 33), 13)
bs2d <- list(bs2, bs2)
bsmv <- list(bs1, bs2d)

# Convert to funts object
Y <- funts(X = Montana_Data, basisobj = bsmv,
            start = as.Date("2008-01-01"),
            end = as.Date("2013-09-30"),
            vnames = c("Normalized Temperature (°C)", "NDVI"),
            dnames = list("Time", c("Latitude", "Longitude")),
            tname = "Date"
)
```
Description

This function retrieves the Utqiagvik Temperature dataset from a GitHub repository hosted at https://github.com/haghbinh/dataset/Rfssa_dataset. Hosting datasets on GitHub rather than including them in the Rfssa R package conserves storage space. The Utqiagvik Temperature dataset contains intraday hourly temperature curves measured in degrees Celsius from January 1973 to July 2023, recorded once per month. The returned object is a raw dataset in ‘list’ format;

Usage

loadUtqiagvikData()

Format

Containing two matrix objects:

**Austin Temperature Data** A 24 by 607 matrix of discrete samplings of intraday hourly temperature curves, one day per month.

**Utqiagvik Temperature Data** A 24 by 607 matrix of discrete samplings of intraday hourly temperature curves, one day per month.

References


Examples

Utqiagvik_raw <- loadUtqiagvikData()
str(Utqiagvik_raw)

Montana Intraday Temperature Curves and NDVI Images Data Set

Description

This dataset includes intraday hourly temperature curves measured in degrees Celsius and normalized difference vegetation index (NDVI) image data. The observations were recorded near Saint Mary, Montana, USA. The NDVI images cover a geographical region with longitudes ranging from 113.30 degrees West to 113.56 degrees West and latitudes from 48.71 degrees North to 48.78 degrees North. The intraday temperature curves are sourced from Diamond et al. (2013), while the NDVI images were obtained from resources provided by Tuck et al. (2014). For each recorded intraday temperature curve, an NDVI image was captured on the same day every 16 days. Data collection started on January 1, 2008, and concluded on September 30, 2013. The primary goal of this dataset is to facilitate the analysis of temperature trends and investigate how temperature changes impact vegetation in the region. A multivariate analysis leveraging both temperature and NDVI variables can reveal more informative patterns and yield stronger signal extraction results.
The dataset is hosted on GitHub, and you can load it using the function `loadMontanaData`. The dataset has been converted into a multivariate functional time series using two B-spline basis function systems with 11 and 13 members, respectively. The resulting dataset is stored as a functional time series object of class `funts`. You can load the raw data using the function `loadMontanaData`. See `funts` for more details.

References


See Also

`loadMontanaData` - Function to load the Montana dataset.

### plot.fforecast

Plot Method for FSSA Forecast (fforecast) Class

#### Description

Create visualizations of FSSA Forecast (fforecast) class. This function supports plotting ‘fforecast’ data with one-dimensional or two-dimensional domains.

#### Usage

```R
## S3 method for class 'fforecast'
plot(
x, 
group_index = NULL, 
ask = TRUE, 
npts = 100, 
obs = 1, 
main = NULL, 
col = NULL, 
ori_col = NULL, 
type = "l", 
lty = 1, 
...)
```

```
Arguments

- **x**: an object of class `fforecast`.
- **group_index**: an integer specifying the group index for the plot.
- **ask**: logical: If `TRUE`, and `group_index` be `NULL`, after printing the first grouping graphic, it will pause when the user asks for the next group graphic and wait.
- **npts**: number of grid points for the plots.
- **obs**: observation number (for two-dimensional domains).
- **main**: main title for the plot.
- **col**: specify the predicted FTS color; if it is `NULL`, it will be set as the default.
- **ori_col**: specify the original FTS color; if it is `NULL`, it will be set as the default.
- **type**: type of plot ("l" for line, "p" for points, etc.).
- **lty**: line type (1 for solid, 2 for dashed, etc.).
- **...**: additional graphical parameters passed to plotting functions.

See Also

- `fforecast`

Examples

```r
# Example with one-dimensional domain
data("Callcenter")
# FSSA Decomposition step:
fssa_results <- fssa(Callcenter, L = 28)

# Perform FSSA R-forecasting
pr_V <- fforecast(U = fssa_results, groups = list(1,1:7),
  len = 14, method = "vector", only.new = FALSE)
plot(pr_V)
```

---

**plot.fssa**

*Plot Functional Singular Spectrum Analysis Objects*

Description

This function is a plotting method for objects of class functional singular spectrum analysis (`fssa`). It aids users in making decisions during the grouping stage of univariate or multivariate functional singular spectrum analysis.
Usage

```r
## S3 method for class 'fssa'
plot(
  x,
  d = length(x$values),
  idx = 1:d,
  idy = idx + 1,
  contrib = TRUE,
  groups = as.list(1:d),
  lwd = 2,
  type = "values",
  vars = NULL,
  ylab = NA,
  main = NA,
  ...
)
```

Arguments

- **x**: An object of class `fssa`.
- **d**: An integer representing the number of elementary components to plot.
- **idx**: A vector of indices specifying which eigen elements to plot.
- **idy**: A second vector of indices of eigen elements to plot (for `type="paired"`).
- **contrib**: A logical value. If `TRUE` (default), it displays the component’s contribution to the total variance.
- **groups**: A list or vector of indices determining the grouping used for decomposition (for `type="wcor"`).
- **lwd**: A vector of line widths.
- **type**: The type of plot to be displayed. Possible types include:
  - "values" - Plot the square-root of eigen values (default).
  - "paired" - Plot pairs of right singular function’s coefficients (useful for detecting periodic components).
  - "wcor" - Plot the W-correlation matrix for the reconstructed objects.
  - "vectors" - Plot the right singular vectors (useful for detecting period length).
  - "lcurves" - Plot left singular functions (useful for detecting period length).
  - "lheats" - Heatmap plot of eigenfunctions, usable for `funts` variables observed over one or two-dimensional domains (useful for detecting meaningful patterns).
  - "periodogram" - Periodogram plot right singular vectors (useful for detecting the frequencies of oscillations in functional data).
- **vars**: A numeric value specifying the variable number (used in plotting MFSSA "lheats" or "lcurves").
- **ylab**: A character vector representing the names of variables.
- **main**: The main plot title.
- **...**: Additional arguments to be passed to methods, such as graphical parameters.
See Also

fssa, plotly_funts

Examples

data("Callcenter")
L <- 28
U <- fssa(Callcenter, L)
plot(U, type = "values", d = 10)
plot(U, type = "vectors", d = 4)
plot(U, type = "paired", d = 6)
plot(U, type = "lcurves", d = 4, vars = 1)
plot(U, type = "lheats", d = 4)
plot(U, type = "wcor", d = 10)

plot.funts

Plot Functional Time Series (funts) Data

Description

Create visualizations of Functional Time Series (funts) data, supporting both one-dimensional and two-dimensional domains.

Usage

## S3 method for class 'funts'
plot(
  x,
  npts = 100,
  obs = 1,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  type = "l",
  lty = 1,
  ...
)

Arguments

x An object of class funts.
npts Number of grid points for the plots.
obs Observation number (for two-dimensional domains).
xlab X-axis label.
ylab Y-axis label.
### plotly_funts

**Description**

Visualize univariate or multivariate Functional Time Series (funts) using Plotly-based plots.

**Usage**

```r
plotly_funts(
  x,
  vars = NULL,
  types = NULL,
  subplot = TRUE,
  main = NULL,
  ylab = NULL,
  xlab = NULL,
  tlab = NULL,
  zlab = NULL,
  xticklabels = NULL,
)```

**Details**

This function enables the creation of visualizations for Functional Time Series (funts) data. It supports both one-dimensional and two-dimensional domains.

For one-dimensional domains, line plots are used, while for two-dimensional domains, image plots are employed.

**See Also**

`funts, Callcenter, Montana`

**Examples**

```r
# Example with one-dimensional domain
data("Callcenter")
plot(Callcenter, lwd = 2, col = "deepskyblue4", main = "Call Center Data")

# Example with two-dimensional domain
data("Montana")
plot(Montana, obs = 2, main = c("Temperature Curves", "NDVI Images,"))
```
xticklocs = NULL,
yticklabels = NULL,
yticklocs = NULL,
color_palette = "RdYlBu",
reverse_color_palette = FALSE,
... )

Arguments

x an object of class funts.
vars numeric vector specifying which variables in the FTS to plot (default: all).
types tuple of strings specifying plot types for each variable.
subplot logical for subplotting line plots.
main titles for each plot.
ylab y-axis titles.
xlab x-axis titles.
tlab time-axis titles.
zlab z-axis titles.
xticklabels tick labels for the domain of the functions.
xticklocs positions of tick labels for the domain of the functions.
yticklabels tick labels for the domain of the functions.
yticklocs positions of tick labels for the domain of the functions.
color_palette color palette for two-dimensional FTS plots.
reverse_color_palette reverse the color palette scale.
... additional arguments to pass to Plotly methods.

Details

Supported plot types for one-dimensional domain variables: - "line": line plots (default). - "heatmap": heatmaps. - "3Dsurface": 3D surface plots. - "3Dline": 3D line plots.

Supported plot type for two-dimensional domain variables: - "heatmap"

Each variable can be plotted multiple times with different types.

See Also

funts, Callcenter, Montana
print.ffecond

Examples

## Not run:

```r
data("Callcenter") # Univariate FTS example

plotly_funts(Callcenter)

plotly_funts(Callcenter,
  main = "Call Center Data Line Plot",
  xticklabels = list(c("00:00", "06:00", "12:00", "18:00", "24:00")),
  xticklocs = list(1, 60, 120, 180, 240))
)

plotly_funts(Callcenter, type = "3Dline", main = "Callcenter Data")

plotly_funts(Callcenter, type = "3Dsurface", main = "Callcenter Data")

plotly_funts(Callcenter, type = "heatmap", main = "Callcenter Data")

data("Montana") # Multivariate FTS example

plotly_funts(Montana[1:100],
  main = 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))
)

## End(Not run)
```

print.ffecond

Custom Print Method for FSSA Forecast (fforecast) class

Description

This custom print method is designed for objects of the FSSA Forecast (fforecast) class. It provides a summary of the fforecast object.

Usage

```r
## S3 method for class 'fforecast'
print(x, ...)
```
Arguments

x  an object of class "fforecast" to be printed.
...

further arguments passed to or from other methods.

Examples

# Example with one-dimensional domain
data("Callcenter")
# FSSA Decomposition step:
fssa_results <- fssa(Callcenter, L = 28)

# Perform FSSA R-forecasting
pr_R <- fforecast(U = fssa_results,
groups = c(1:3),
len = 14,
method = "recurrent")
print(pr_R)
Indexing into Functional Time Series

Description

An indexing method for functional time series (funts) objects.

Usage

```r
## S3 method for class 'funts'
obj[i = NULL, j = NULL]
```

Arguments

- `obj` an object of class funts.
- `i` an index or indices specifying the subsets of times to extract.
- `j` an index or indices specifying the subsets of variables to extract.

Details

This function allows you to extract specific subsets of a functional time series based on the provided indices. You can specify which subsets you want to extract from the functional time series.

Value

an `funts` object containing the specified subsets

See Also

funts
Index

*.funts, 2
+.funts, 3
-.funts, 4
[].funts, 31

as.funts, 4

Callcenter, 5, 27, 28
create.bspine.basis, 5

eval.funts, 6

fd, 5
fforecast, 7, 24
fpredinterval, 9
freconstruct, 11
fssa, 8, 11, 12, 13, 15, 16, 24–26
fts, 5
funts, 2–8, 10–13, 14, 16, 18–21, 23, 25, 27, 28, 31

is.funts, 15

launchApp, 16
length.funts, 17
loadAustinData, 17
loadCallcenterData, 6, 18
loadJambiData, 19
loadMontanaData, 20, 23
loadUtqiagvikData, 21

Montana, 22, 27, 28

plot.fforecast, 23
plot.fssa, 24
plot.funts, 26
plotly_funts, 26, 27
print.fforecast, 29
print.funts, 30