Package ‘ftsa’

September 25, 2020

Type Package
Title Functional Time Series Analysis
Version 5.9
Date 2020-9-25
Depends R (>= 3.5.0), forecast, rainbow, sde
Suggests fds, R2jags, vars, meboot
Imports colorspace, MASS, pcaPP, fda, pdfCluster, ecp, strucchange, e1071
LazyLoad yes
LazyData yes
ByteCompile TRUE
Maintainer Han Lin Shang <hanlin.shang@mq.edu.au>
Description Functions for visualizing, modeling, forecasting and hypothesis testing of functional time series.
License GPL-3
NeedsCompilation no
Author Rob Hyndman [aut] (<https://orcid.org/0000-0002-2140-5352>),
    Han Lin Shang [aut, cre, cph] (<https://orcid.org/0000-0003-1769-6430>)
Repository CRAN
Date/Publication 2020-09-25 14:00:09 UTC

R topics documented:

ftsa-package .................................................. 3
centre .......................................................... 5
diff.fts .......................................................... 6
dmfpcap ............................................................ 7
dynamic_FLR .................................................... 8
dynupdate ....................................................... 10
error ............................................................ 13
### R topics documented:

- `ER_GR` ................................................................. 15
- `extract` .......................................................... 16
- `facf` ............................................................... 17
- `farforecast` .......................................................... 18
- `fbootstrap` ............................................................ 19
- `forecast.ftsm` ......................................................... 21
- `forecast.hdfpca` ....................................................... 23
- `forecastfplsr` .......................................................... 25
- `fplsr` ................................................................. 26
- `ftsm` ................................................................. 29
- `ftsriterativeforecasts` .................................................. 32
- `ftsmweightselect` ....................................................... 33
- `hdfpca` ............................................................... 34
- `hd_data` ............................................................... 36
- `is.fts` ................................................................. 37
- `isfe.fts` .............................................................. 38
- `long_run_covariance_estimation` ......................................... 39
- `MAF_multivariate` ..................................................... 40
- `mean.fts` ............................................................. 41
- `median.fts` .......................................................... 43
- `MFDM` ................................................................. 44
- `MFPCA` ............................................................... 46
- `mftsc` ............................................................... 48
- `pcscorebootstrapdata` ................................................. 49
- `plot.fm` .............................................................. 51
- `plot.fmres` .......................................................... 52
- `plot.ftsf` ............................................................ 53
- `plot.ftsm` ............................................................ 55
- `plotfplsr` ............................................................ 56
- `pm_10_GR` ............................................................ 57
- `quantile` ............................................................. 58
- `quantile.fts` .......................................................... 59
- `residuals.fm` ......................................................... 59
- `sd` ................................................................. 60
- `sd.fts` ............................................................... 61
- `sim_ex_cluster` ...................................................... 63
- `stop_time_detect` ..................................................... 64
- `stop_time_sim_data` ................................................... 65
- `summary.fm` .......................................................... 65
- `T_stationary` ........................................................ 66
- `var` ................................................................. 67
- `var.fts` ............................................................. 68

**Index** 71
Description

This package presents descriptive statistics of functional data; implements principal component regression and partial least squares regression to provide point and distributional forecasts for functional data; utilizes functional linear regression, ordinary least squares, penalized least squares, ridge regression, and moving block approaches to dynamically update point and distributional forecasts when partial data points in the most recent curve are observed; performs stationarity test for a functional time series; estimates a long-run covariance function by kernel sandwich estimator.

Author(s)

Rob J Hyndman and Han Lin Shang
Maintainer: Han Lin Shang <hanlin.shang@anu.edu.au>

References

H. L. Shang (2016) "Mortality and life expectancy forecasting for a group of populations in developed countries: A robust multilevel functional data method", in C. Agostinelli, A. Basu, P.


H. L. Shang, A. Wisniowski, J. Bijak, P. W. F. Smith and J. Raymer (2014) "Bayesian functional models for population forecasting", in M. Marsili and G. Capacci (eds), Proceedings of the Sixth


centre

*Mean function, variance function, median function, trim mean function of functional data*

**Description**

Mean function, variance function, median function, trim mean function of functional data

**Usage**

centre(x, type)

**Arguments**

- **x**: An object of class `matrix`.
- **type**: Mean, variance, median or trim mean?

**Value**

Return mean function, variance function, median function or trim mean function.
diff.fts

Author(s)
Han Lin Shang

See Also
diff.fts, mean.fts, median.fts, sd.fts, var.fts

Examples

# mean function is often removed in the functional principal component analysis.
# trimmed mean function is sometimes employed for robustness in the presence of outliers.
# In calculating trimmed mean function, several functional depth measures were employed.

centre(x = ElNino_ERSST_region_land2$y, type = "mean")
centre(x = ElNino_ERSST_region_land2$y, type = "var")
centre(x = ElNino_ERSST_region_land2$y, type = "median")
centre(x = ElNino_ERSST_region_land2$y, type = "trimmed")

diff.fts

Differences of a functional time series

Description
Computes differences of a fts object at each variable.

Usage

## S3 method for class 'fts'
diff(x, lag = 1, differences = 1, ...)

Arguments

x       An object of class fts.
lag     An integer indicating which lag to use.
differences     An integer indicating the order of the difference.
...     Other arguments.

Value
An object of class fts.

Author(s)
Rob J Hyndman

Examples

# ElNino is an object of sliced functional time series.
# Differencing is sometimes used to achieve stationarity.
diff(x = ElNino_ERSST_region_land2)
dmfpca  Dynamic multilevel functional principal component analysis

Description

Functional principal component analysis is used to decompose multiple functional time series. This function uses a functional panel data model to reduce dimensions for multiple functional time series.

Usage

dmfpca(y, M = NULL, J = NULL, N = NULL, tstart = 0, tlength = 1)

Arguments

y  A data matrix containing functional responses. Each row contains measurements from a function at a set of grid points, and each column contains measurements of all functions at a particular grid point

M  Number of fts objects

J  Number of functions in each object

N  Number of grid points per function

tstart  Start point of the grid points

tlength  Length of the interval that the functions are evaluated at

Value

K1  Number of components for the common time-trend

K2  Number of components for the residual component

lambda1  A vector containing all common time-trend eigenvalues in non-increasing order

lambda2  A vector containing all residual component eigenvalues in non-increasing order

phi1  A matrix containing all common time-trend eigenfunctions. Each row contains an eigenfunction evaluated at the same set of grid points as the input data. The eigenfunctions are in the same order as the corresponding eigenvalues

phi2  A matrix containing all residual component eigenfunctions. Each row contains an eigenfunction evaluated at the same set of grid points as the input data. The eigenfunctions are in the same order as the corresponding eigenvalues

scores1  A matrix containing estimated common time-trend principal component scores. Each row corresponding to the common time-trend scores for a particular subject in a cluster. The number of rows is the same as that of the input matrix y. Each column contains the scores for a common time-trend component for all subjects.

scores2  A matrix containing estimated residual component principal component scores. Each row corresponding to the level 2 scores for a particular subject in a cluster. The number of rows is the same as that of the input matrix y. Each column contains the scores for a residual component for all subjects.
Dynamic updates via functional linear regression

Description

A functional linear regression is used to address the problem of dynamic updating, when partial data in the most recent curve are observed.

Usage

dynamic_FLR(dat, newdata, holdoutdata, order_k_percent = 0.9, order_m_percent = 0.9, pcd_method = c("classical", "M"), robust_lambda = 2.33, bootrep = 100, pointfore, level = 80)

mu
A vector containing the overall mean function.

etta
A matrix containing the deviation from overall mean function to country specific mean function. The number of rows is the number of countries.

Author(s)

Chen Tang and Han Lin Shang

References


See Also

mftsc

Examples

## The following takes about 10 seconds to run ##
## Not run:
y <- do.call(rbind, sim_ex_cluster)
MFPCA.sim <- dmfpca(y, M = length(sim_ex_cluster), J = nrow(sim_ex_cluster[[1]]),
N = ncol(sim_ex_cluster[[1]]), tlength = 1)
## End(Not run)
Arguments

dat An object of class sfts.
newdata A data vector of newly arrived observations.
holdoutdata A data vector of holdout sample to evaluate point forecast accuracy.
order_k_percent Select the number of components that explains at least 90 percent of the total variation.
order_m_percent Select the number of components that explains at least 90 percent of the total variation.
pcd_method Method to use for principal components decomposition. Possibilities are "M", "rapca" and "classical".
robust_lambda Tuning parameter in the two-step robust functional principal component analysis, when pcd_method = "M".
bootstrap Number of bootstrap samples.
pointfore If pointfore = TRUE, point forecasts are produced.
level Nominal coverage probability.

Details

This function is designed to dynamically update point and interval forecasts, when partial data in the most recent curve are observed.

Value

update_forecast Updated forecasts.
holdoutdata Holdout sample.
err Forecast errors.
order_k Number of principal components in the first block of functions.
order_m Number of principal components in the second block of functions.
update_comb Bootstrapped forecasts for the dynamically updating time period.
update_comb_lb_ub By taking corresponding quantiles, obtain lower and upper prediction bounds.
err_boot Bootstrapped in-sample forecast error for the dynamically updating time period.

Author(s)

Han Lin Shang
**References**


**See Also**

dynupdate

**Examples**

dynamic_FLR_point = dynupdate(dat = ElNino_ERSST_region_1and2$y[,1:68],
newdata = ElNino_ERSST_region_1and2$y[1:4,69],
holdoutdata = ElNino_ERSST_region_1and2$y[5:12,69], pointfore = TRUE)

dynamic_FLR_interval = dynupdate(dat = ElNino_ERSST_region_1and2$y[,1:68],
newdata = ElNino_ERSST_region_1and2$y[1:4,69],
holdoutdata = ElNino_ERSST_region_1and2$y[5:12,69], pointfore = FALSE)

dynupdate

*Dynamic updates via BM, OLS, RR and PLS methods*

**Description**

Four methods, namely block moving (BM), ordinary least squares (OLS) regression, ridge regression (RR), penalized least squares (PLS) regression, were proposed to address the problem of dynamic updating, when partial data in the most recent curve are observed.

**Usage**

dynupdate(data, newdata = NULL, holdoutdata, method = c("ts", "block",
"ols", "pls", "ridge"), fmethod = c("arima", "ar", "ets", "ets.na",
"rwdrift", "rw"), pcdmethod = c("classical", "H", "rapca"),
ngrid = max(1000, ncol(data$y)), order = 6,
robust_lambda = 2.33, lambda = 0.01, value = FALSE, interval = FALSE, level = 80, pimethod = c("parametric", "nonparametric"), B = 1000)

Arguments

data          An object of class sfts.
newdata        A data vector of newly arrived observations.
holdoutdata    A data vector of holdout sample to evaluate point forecast accuracy.
method         Forecasting methods. The latter four can dynamically update point forecasts.
fmethod        Univariate time series forecasting methods used in method = "ts" or method = "block".
pcdmethod      Method to use for principal components decomposition. Possibilities are "M", "rapca" and "classical".
ngrid          Number of grid points to use in calculations. Set to maximum of 1000 and ncol(data$y).
order          Number of principal components to fit.
robust_lambda  Tuning parameter in the two-step robust functional principal component analysis, when pcdmethod = "M".
lambda         Penalty parameter used in method = "pls" or method = "ridge".
value          When value = TRUE, returns forecasts or when value = FALSE, returns forecast errors.
interval       When interval = TRUE, produces distributional forecasts.
level          Nominal coverage probability.
pimethod       Parametric or nonparametric method to construct prediction intervals.
B              Number of bootstrap samples.

Details

This function is designed to dynamically update point and interval forecasts, when partial data in the most recent curve are observed.

If method = "classical", then standard functional principal component decomposition is used, as described by Ramsay and Dalzell (1991).

If method = "rapca", then the robust principal component algorithm of Hubert, Rousseeuw and Verboven (2002) is used.

If method = "M", then the hybrid algorithm of Hyndman and Ullah (2005) is used.

Value

forecasts      An object of class fts containing the dynamic updated point forecasts.
bootsamp       An object of class fts containing the bootstrapped point forecasts, which are updated by the PLS method.
low            An object of class fts containing the lower bound of prediction intervals.
up             An object of class fts containing the upper bound of prediction intervals.
Author(s)

Han Lin Shang

References


See Also

*ftsm, forecast.ftsm, plot.fm, residuals.fm, summary.fm*

Examples

```r
# ElNino is an object of sliced functional time series, constructed from a univariate time series.
# When we observe some newly arrived information in the most recent time period, this function
# allows us to update the point and interval forecasts for the remaining time period.
dynupdate(data = ElNino_ERSST_region_land2, newdata = ElNino_ERSST_region_land2$y[1:4,69],
          holdoutdata = ElNino_ERSST_region_land2$y[5:12,57], method = "block", interval = FALSE)
```
Description

Computes the forecast error measure.

Usage

```
```

Arguments

- **forecast**: Out-of-sample forecasted values.
- **forecastbench**: Forecasted values using a benchmark method, such as random walk.
- **true**: Out-of-sample holdout values.
- **insampletrue**: Insample values.
- **method**: Method of forecast error measure.
- **giveall**: If `giveall = TRUE`, all error measures are provided.

Details

**Bias measure:**

If `method = "me"`, the forecast error measure is mean error.

If `method = "mpe"`, the forecast error measure is mean percentage error.

**Forecast accuracy error measure:**

If `method = "mae"`, the forecast error measure is mean absolute error.

If `method = "mse"`, the forecast error measure is mean square error.

If `method = "sse"`, the forecast error measure is sum square error.

If `method = "rmse"`, the forecast error measure is root mean square error.

If `method = "mdae"`, the forecast error measure is median absolute error.

If `method = "mdape"`, the forecast error measure is median absolute percentage error.

If `method = "rmspe"`, the forecast error measure is root mean square percentage error.

If `method = "rmdspe"`, the forecast error measure is root median square percentage error.

**Forecast accuracy symmetric error measure:**

If `method = "smape"`, the forecast error measure is symmetric mean absolute percentage error.

If `method = "smdape"`, the forecast error measure is symmetric median absolute percentage error.
**Forecast accuracy relative error measure:**

If method = "mrae", the forecast error measure is mean relative absolute error.

If method = "mdrae", the forecast error measure is median relative absolute error.

If method = "gmrae", the forecast error measure is geometric mean relative absolute error.

If method = "relmae", the forecast error measure is relative mean absolute error.

If method = "relmse", the forecast error measure is relative mean square error.

**Forecast accuracy scaled error measure:**

If method = "mase", the forecast error measure is mean absolute scaled error.

If method = "mdase", the forecast error measure is median absolute scaled error.

If method = "rmsse", the forecast error measure is root mean square scaled error.

**Value**

A numeric value.

**Author(s)**

Han Lin Shang

**References**


**Examples**

# Forecast error measures can be categorized into three groups: (1) scale-dependent,
# (2) scale-independent but with possible zero denominator,
# (3) scale-independent with non-zero denominator.
e = error(forecast = 1:2, true = 3:4, method = "mae")
e = error(forecast = 1:5, forecastbench = 6:10, true = 11:15, method = "mrae")
e = error(forecast = 1:5, forecastbench = 6:10, true = 11:15, insampletrue = 16:20, giveall = TRUE)
### Description

Eigenvalue ratio and growth ratio

### Usage

```r
ER_GR(data)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An n by p matrix, where n denotes sample size and p denotes the number of discretized data points in a curve</td>
</tr>
</tbody>
</table>

### Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_ER</td>
<td>The number of components selected by the eigenvalue ratio</td>
</tr>
<tr>
<td>k_GR</td>
<td>The number of components selected by the growth ratio</td>
</tr>
</tbody>
</table>

### Author(s)

Han Lin Shang

### References


### See Also

`ftsm`

### Examples

```r
ER_GR(pm_10_GR$y)
```
**extract**

**Extract variables or observations**

**Description**

Creates subsets of a fts object.

**Usage**

```r
eextract(data, direction = c("time", "x"), timeorder, xorder)
```

**Arguments**

- **data**: An object of fts.
- **direction**: In time direction or x variable direction?
- **timeorder**: Indexes of time order.
- **xorder**: Indexes of x variable order.

**Value**

When `xorder` is specified, it returns a fts object with same argument as data but with a subset of x variables.

When `timeorder` is specified, it returns a fts object with same argument as data but with a subset of time variables.

**Author(s)**

Han Lin Shang

**Examples**

```r
# ElNino is an object of class sliced functional time series.
# This function truncates the data series rowwise or columnwise.
eextract(data = ElNino_ERSST_region_land2, direction = "time",
timeorder = 1980:2006)  # Last 27 curves
eextract(data = ElNino_ERSST_region_land2, direction = "x",
xorder = 1:8)  # First 8 x variables
```
Description

Compute functional autocorrelation function at various lags

Usage

facf(fun_data, lag_value_range = seq(0, 20, by = 1))

Arguments

fun_data A data matrix of dimension (n by p), where n denotes sample size; and p denotes dimensionality
lag_value_range Lag value

Details

The autocovariance at lag $i$ is estimated by the function $\hat{\gamma}_i(t, s)$, a functional analog of the autocorrelation is defined as

$$\hat{\rho}_i = \frac{\|\hat{\gamma}_i\|}{\int \hat{\gamma}_0(t, t) dt}.$$

Value

A vector of functional autocorrelation function at various lags

Author(s)

Han Lin Shang

References


Examples

facf_value = facf(fun_data = t(ElNino_ERSST_region_1and2$y))
farforecast

Functional data forecasting through functional principal component autoregression

Description

The coefficients from the fitted object are forecasted using a multivariate time-series forecasting method. The forecast coefficients are then multiplied by the functional principal components to obtain a forecast curve.

Usage

```r
farforecast(object, h = 10, var_type = "const", Dmax_value, Pmax_value, level = 80, PI = FALSE)
```

Arguments

- **object**: An object of `fds`.
- **h**: Forecast horizon.
- **var_type**: Type of multivariate time series forecasting method; see `VAR` for details.
- **Dmax_value**: Maximum number of components considered.
- **Pmax_value**: Maximum order of VAR model considered.
- **level**: Nominal coverage probability of prediction error bands.
- **PI**: When `PI = TRUE`, a prediction interval will be given along with the point forecast.

Details

1. Decompose the smooth curves via a functional principal component analysis (FPCA).
2. Fit a multivariate time-series model to the principal component score matrix.
3. Forecast the principal component scores using the fitted multivariate time-series models. The order of VAR is selected optimally via an information criterion.
4. Multiply the forecast principal component scores by estimated principal components to obtain forecasts of \( f_{n+h}(x) \).
5. Prediction intervals are constructed by taking quantiles of the one-step-ahead forecast errors.

Value

- **point_fore**: Point forecast
- **order_select**: Selected VAR order and number of components
- **PI_lb**: Lower bound of a prediction interval
- **PI_ub**: Upper bound of a prediction interval
Author(s)

Han Lin Shang

References


See Also

*forecast.ftsm, forecastfplsr*

Examples

```r
sqrt_pm10 = sqrt(pm_10_GR$y)
x = seq(0,23.5, by=.5)
multi_forecast_sqrt_pm10 = farforecast(object = fts(x,sqrt_pm10), h = 10, Dmax_value = 21, Pmax_value = 3)
plot(multi_forecast_sqrt_pm10$point_fore, ylim = c(5.2,8.5))
```

---

**fbootstrap**

*Bootstrap independent and identically distributed functional data*

Description

Computes bootstrap or smoothed bootstrap samples based on independent and identically distributed functional data.

Usage

```r
fbootstrap(data, estad = func.mean, alpha = 0.05, nb = 200, suav = 0, media.dist = FALSE, graph = FALSE, ...)
```

Arguments

- `data` An object of class fds or fts.
- `estad` Estimate function of interest. Default is to estimate the mean function. Other options are `func.mode` or `func.var`.
- `alpha` Significance level used in the smooth bootstrapping.
- `nb` Number of bootstrap samples.
- `suav` Smoothing parameter.
- `media.dist` Estimate mean function.
- `graph` Graphical output.
- `...` Other arguments.
Value

A list containing the following components is returned.

- `estimate`: Estimate function.
- `max.dist`: Max distance of bootstrap samples.
- `rep.dist`: Distances of bootstrap samples.
- `resamples`: Bootstrap samples.
- `center`: Functional mean.

Author(s)

Han Lin Shang

References


See Also

- `pcscorebootstrapdata`

Examples

```r
# Bootstrapping the distribution of a summary statistics of functional data.
fbootstrap(data = ElNino_ERSST_region_land2)
```
**Description**

The coefficients from the fitted object are forecasted using either an ARIMA model (method = "arima"), an AR model (method = "ar"), an exponential smoothing method (method = "ets"), a linear exponential smoothing method allowing missing values (method = "ets.na"), or a random walk with drift model (method = "rwdrift"). The forecast coefficients are then multiplied by the principal components to obtain a forecast curve.

**Usage**

```r
## S3 method for class 'ftsm'
forecast(object, h = 10, method = c("ets", "arima", "ar", "ets.na", "rwdrift", "rw", "struct", "arfima"), level = 80, jumpchoice = c("fit", "actual"), pimethod = c("parametric", "nonparametric"), B = 100, usedata = nrow(object$coeff), adjust = TRUE, model = NULL, damped = NULL, stationary = FALSE, ...)
```

**Arguments**

- `object` Output from `ftsm`.
- `h` Forecast horizon.
- `method` Univariate time series forecasting methods. Current possibilities are “ets”, “arima”, “ets.na”, “rwdrift” and “rw”.
- `level` Coverage probability of prediction intervals.
- `jumpchoice` Jump-off point for forecasts. Possibilities are “actual” and “fit”. If “actual”, the forecasts are bias-adjusted by the difference between the fit and the last year of observed data. Otherwise, no adjustment is used. See Booth et al. (2006) for the detail on jump-off point.
- `pimethod` Indicates if parametric method is used to construct prediction intervals.
- `B` Number of bootstrap samples.
- `usedata` Number of time periods to use in forecasts. Default is to use all.
- `adjust` If `adjust = TRUE`, adjusts the variance so that the one-step forecast variance matches the empirical one-step forecast variance.
- `model` If the `ets` method is used, `model` allows a model specification to be passed to `ets()`.
- `damped` If the `ets` method is used, `damped` allows the damping specification to be passed to `ets()`.
- `stationary` If `stationary = TRUE`, method is set to `method = "ar"` and only stationary AR models are used.
- `...` Other arguments passed to forecast routine.
Details

1. Obtain a smooth curve \( f_t(x) \) for each \( t \) using a nonparametric smoothing technique.
2. Decompose the smooth curves via a functional principal component analysis.
3. Fit a univariate time series model to each of the principal component scores.
4. Forecast the principal component scores using the fitted time series models.
5. Multiply the forecast principal component scores by fixed principal components to obtain forecasts of \( f_{n+h}(x) \).
6. The estimated variances of the error terms (smoothing error and model residual error) are used to compute prediction intervals for the forecasts.

Value

List with the following components:

- mean: An object of class \( \text{fts} \) containing point forecasts.
- lower: An object of class \( \text{fts} \) containing lower bound for prediction intervals.
- upper: An object of class \( \text{fts} \) containing upper bound for prediction intervals.
- fitted: An object of class \( \text{fts} \) of one-step-ahead forecasts for historical data.
- error: An object of class \( \text{fts} \) of one-step-ahead errors for historical data.
- coeff: List of objects of type \( \text{forecast} \) containing the coefficients and their forecasts.
- coeff.error: One-step-ahead forecast errors for each of the coefficients.
- var: List containing the various components of variance: model, error, mean, total and coeff.
- model: Fitted \( \text{ftsm} \) model.
- bootsamp: An array of dimension \( c(p, B, h) \) containing the bootstrapped point forecasts. \( p \) is the number of variables. \( B \) is the number of bootstrap samples. \( h \) is the forecast horizon.

Author(s)

Rob J Hyndman

References


See Also

`ftsm, forecastfplsr, plot.ftsf, plot.fm, residuals.fm, summary.fm`

Examples

```r
# ElNino is an object of class sliced functional time series.
# Via functional principal component decomposition, the dynamic was captured
# by a few principal components and principal component scores.
# By using an exponential smoothing method,
# the principal component scores are forecasted.
# The forecasted curves are constructed by forecasted principal components
# times fixed principal components plus the mean function.
forecast(object = ftsm(ElNino_ERSST_region_1and2), h = 10, method = "ets")
forecast(object = ftsm(ElNino_ERSST_region_1and2, weight = TRUE))
```

---

**forecast.hdfpca**

Forecasting via a high-dimensional functional principal component regression

**Description**

Forecast high-dimensional functional principal component model.

**Usage**

```r
### S3 method for class 'hdfpca'
forecast(object, h = 3, level = 80, B = 50, ...)
```
Arguments

object  An object of class 'hdfpca'
h       Forecast horizon
level   Prediction interval level, the default is 80 percent
B       Number of bootstrap replications
...     Other arguments passed to forecast routine.

Details

The low-dimensional factors are forecasted with autoregressive integrated moving average (ARIMA) models separately. The forecast functions are then calculated using the forecast factors. Bootstrap prediction intervals are constructed by resampling from the forecast residuals of the ARIMA models.

Value

forecast A list containing the h-step-ahead forecast functions for each population
upper    Upper confidence bound for each population
lower    Lower confidence bound for each population

Author(s)

Y. Gao and H. L. Shang

References


See Also

hdfpca, hd_data

Examples

```r
hd_model = hdfpca(hd_data, order = 2, r = 2)
hd_model_fore = forecast.hdfpca(object = hd_model, h = 1)
```
**forecastfplsr**

**Forecast functional time series**

---

**Description**

The decentralized response is forecasted by multiplying the estimated regression coefficient with the new decentralized predictor.

**Usage**

```r
forecastfplsr(object, components, h)
```

**Arguments**

- `object`: An object of class `fts`.
- `components`: Number of optimal components.
- `h`: Forecast horizon.

**Value**

A `fts` class object, containing forecasts of responses.

**Author(s)**

Han Lin Shang

**References**


**See Also**

`forecast.ftsm`, `ftsm`, `plot.fm`, `plot.ftsf`, `residuals.fm`, `summary.fm`

**Examples**

```r
# A set of functions are decomposed by functional partial least squares decomposition.
# By forecasting univariate partial least squares scores, the forecasted curves are
# obtained by multiplying the forecasted scores by fixed functional partial least
# squares function plus fixed mean function.
forecastfplsr(object = E1NinoERSST_region_land2, components = 2, h = 5)
```
**fplsr**

*Functional partial least squares regression*

**Description**

Fits a functional partial least squares (PLSR) model using nonlinear partial least squares (NIPALS) algorithm or simple partial least squares (SIMPLS) algorithm.

**Usage**

```r
fplsr(data, order = 6, type = c("simpls", "nipals"), unit.weights = TRUE, weight = FALSE, beta = 0.1, interval = FALSE, method = c("delta", "boota"), alpha = 0.05, B = 100, adjust = FALSE, backh = 10)
```

**Arguments**

- **data**: An object of class `fts`.
- **order**: Number of principal components to fit.
- **type**: When `type = "nipals"`, uses the NIPALS algorithm; when `type = "simpls"`, uses the SIMPLS algorithm.
- **unit.weights**: Constrains predictor loading weights to have unit norm.
- **weight**: When `weight = TRUE`, a set of geometrically decaying weights is applied to the decentralized data.
- **beta**: When `weight = TRUE`, the speed of geometric decay is governed by a weight parameter.
- **interval**: When `interval = TRUE`, produces distributional forecasts.
- **method**: Method used for computing prediction intervals.
- **alpha**: $1-\alpha$ gives the nominal coverage probability.
- **B**: Number of replications.
- **adjust**: When `adjust = TRUE`, an adjustment is performed.
- **backh**: When `adjust = TRUE`, an adjustment is performed by evaluating the difference between predicted and actual values in a testing set. `backh` specifies the testing set.

**Details**

*Point forecasts:*

The NIPALS function implements the orthogonal scores algorithm, as described in Martens and Naes (1989). This is one of the two classical PLSR algorithms, the other is the simple partial least squares regression in DeJong (1993). The difference between these two approaches is that the NIPALS deflates the original predictors and responses, while the SIMPLS deflates the covariance.
matrix of original predictors and responses. Thus, SIMPLS is more computationally efficient than NIPALS.

In a functional data set, the functional PLSR can be performed by setting the functional responses to be 1 lag ahead of the functional predictors. This idea has been adopted from the Autoregressive Hilbertian processes of order 1 (ARH(1)) of Bosq (2000).

**Distributional forecasts:**

**Parametric method:**

Influenced by the works of Denham (1997) and Phatak et al. (1993), one way of constructing prediction intervals in the PLSR is via a local linearization method (also known as the Delta method). It can be easily understood as the first two terms in a Taylor series expansion. The variance of coefficient estimators can be approximated, from which an analytic-formula based prediction intervals are constructed.

**Nonparametric method:**

After discretizing and decentralizing functional data $f_t(x)$ and $g_s(y)$, a PLSR model with $K$ latent components is built. Then, the fit residuals $o_s(y_i)$ between $g_s(y_i)$ and $\hat{g}_s(y_i)$ are calculated as

$$o_s(y_i) = g_s(y_i) - \hat{g}_s(y_i), \quad i = 1, ..., p.$$

The next step is to generate $B$ bootstrap samples $\hat{o}_s^b(y_i)$ by randomly sampling with replacement from $[o_1(y_i), ..., o_n(y_i)]$. Adding bootstrapped residuals to the original response variables in order to generate new bootstrap responses,

$$\hat{g}_s^b(y_i) = g_s(y_i) + \hat{o}_s^b(y_i).$$

Then, the PLSR models are constructed using the centered and discretized predictors and bootstrapped responses to obtain the bootstrapped regression coefficients and point forecasts, from which the empirical prediction intervals and kernel density plots are constructed.

**Value**

A list containing the following components is returned.

- **B** $(p \times m)$ matrix containing the regression coefficients. $p$ is the number of variables in the predictors and $m$ is the number of variables in the responses.
- **P** $(p \times \text{order})$ matrix containing the predictor loadings.
- **Q** $(m \times \text{order})$ matrix containing the response loadings.
- **T** $(\text{ncol(data}$y$)-1) \times \text{order}$ matrix containing the predictor scores.
- **R** $(p \times \text{order})$ matrix containing the weights used to construct the latent components of predictors.
- **Yscores** $(\text{ncol(data}$y$)-1) \times \text{order}$ matrix containing the response scores.
- **projection** $(p \times \text{order})$ projection matrix used to convert predictors to predictor scores.
- **meanX** An object of class fts containing the column means of predictors.
- **meanY** An object of class fts containing the column means of responses.
Ypred  An object of class fts containing the 1-step-ahead predicted values of the responses.
fitted  An object of class fts containing the fitted values.
residuals  An object of class fts containing the regression residuals.
Xvar  A vector with the amount of predictor variance explained by each number of component.
Xtotvar  Total variance in predictors.
weight  When weight = TRUE, a set of geometrically decaying weights is given. When weight = FALSE, weights are all equal 1.
x1  Time period of a fts object, which can be obtained from colnames(data$y).
y1  Variables of a fts object, which can be obtained from data$x.
ypred  Returns the original functional predictors.
y  Returns the original functional responses.
bootsamp  Bootstrapped point forecasts.
lb  Lower bound of prediction intervals.
ub  Upper bound of prediction intervals.
lbadj  Adjusted lower bound of prediction intervals.
ubadj  Adjusted upper bound of prediction intervals.
lbadjfactor  Adjusted lower bound factor, which lies generally between 0.9 and 1.1.
ubadjfactor  Adjusted upper bound factor, which lies generally between 0.9 and 1.1.

Author(s)
Han Lin Shang

References


See Also

`ftsm, forecast.ftsm, plot.fm, summary.fm, residuals.fm, plot.fmres`

Examples

```r
# When weight = FALSE, all observations are assigned equally.
# When weight = TRUE, all observations are assigned geometrically decaying weights.
fplsr(data = ElNino_ERSST_region_1and2, order = 6, type = "nipals")
fplsr(data = ElNino_ERSST_region_1and2, order = 6)
fplsr(data = ElNino_ERSST_region_1and2, weight = TRUE)
fplsr(data = ElNino_ERSST_region_1and2, unit.weights = FALSE)
fplsr(data = ElNino_ERSST_region_1and2, unit.weights = FALSE, weight = TRUE)

# The prediction intervals are calculated numerically.
fplsr(data = ElNino_ERSST_region_1and2, interval = TRUE, method = "delta")

# The prediction intervals are calculated by bootstrap method.
fplsr(data = ElNino_ERSST_region_1and2, interval = TRUE, method = "boota")
```

**ftsm**

*Fit functional time series model*

**Description**

Fits a principal component model to a `fts` object. The function uses optimal orthonormal principal components obtained from a principal components decomposition.
Usage

```r
ftsm(y, order = 6, ngrid = max(500, ncol(y$x)), method = c("classical", "M", "rapca"), mean = TRUE, level = FALSE, lambda = 3, weight = FALSE, beta = 0.1, ...)
```

Arguments

- **y**: An object of class `fts`.
- **order**: Number of principal components to fit.
- **ngrid**: Number of grid points to use in calculations. Set to maximum of 500 and `ncol(y$x)`.
- **method**: Method to use for principal components decomposition. Possibilities are “M”, “rapca” and “classical”.
- **mean**: If `mean = TRUE`, it will estimate mean term in the model before computing basis terms. If `mean = FALSE`, the mean term is assumed to be zero.
- **level**: If `mean = TRUE`, it will include an additional (intercept) term that depends on `t` but not on `x`.
- **lambda**: Tuning parameter for robustness when `method = "M"`.
- **weight**: When `weight = TRUE`, a set of geometrically decaying weights is applied to the decentralized data.
- **beta**: When `weight = TRUE`, the speed of geometric decay is governed by a weight parameter.
- **...**: Additional arguments controlling the fitting procedure.

Details

If `method = "classical"`, then standard functional principal component decomposition is used, as described by Ramsay and Dalzell (1991).

If `method = "rapca"`, then the robust principal component algorithm of Hubert, Rousseeuw and Verboven (2002) is used.

If `method = "M"`, then the hybrid algorithm of Hyndman and Ullah (2005) is used.

Value

Object of class “ftsm” with the following components:

- **x1**: Time period of a `fts` object, which can be obtained from `colnames(y$x)`.
- **y1**: Variables of a `fts` object, which can be obtained from `y$x`.
- **y**: Original functional time series or sliced functional time series.
- **basis**: Matrix of principal components evaluated at value of `y$x` (one column for each principal component). The first column is the fitted mean or median.
- **basis2**: Matrix of principal components excluded from the selected model.
- **coeff**: Matrix of coefficients (one column for each coefficient series). The first column is all ones.
Matrix of coefficients associated with the principal components excluded from
the selected model.

An object of class fts containing the fitted values.

An object of class fts containing the regression residuals (difference between
observed and fitted).

Proportion of variation explained by each principal component.

Weight associated with each time period.

Measure of variation for each time period.

Measure of standard error associated with the mean.

Author(s)

Rob J Hyndman

References


M. Hubert and P. J. Rousseeuw and S. Verboven (2002) "A fast robust method for principal com-
ponents with applications to chemometrics", Chemometrics and Intelligent Laboratory Systems,
60(1-2), 101-111.

B. Erbas and R. J. Hyndman and D. M. Gertig (2007) "Forecasting age-specific breast cancer mor-
tality using functional data model", Statistics in Medicine, 26(2), 458-470.

R. J. Hyndman and M. S. Ullah (2007) "Robust forecasting of mortality and fertility rates: A func-
tional data approach", Computational Statistics and Data Analysis, 51(10), 4942-4956.


R. J. Hyndman and H. L. Shang (2009) "Forecasting functional time series (with discussion)",

See Also

ftsmweightselect, forecast.ftsm, plot.fm, plot.ftsf, residuals.fm, summary.fm

Examples

# ElNino is an object of class sliced functional time series, constructed
# from a univariate time series.
# By default, all observations are assigned with equal weighting.
ftsm(y = ElNino_ERSST_region_1and2, order = 6, method = 'classical', weight = FALSE)
# When weight = TRUE, geometrically decaying weights are used.
ftsm(y = ElNino_ERSST_region_1and2, order = 6, method = 'classical', weight = TRUE)
Description

The coefficients from the fitted object are forecasted using either an ARIMA model (method = "arima"), an AR model (method = "ar"), an exponential smoothing method (method = "ets"), a linear exponential smoothing method allowing missing values (method = "ets.na"), or a random walk with drift model (method = "rwdrift"). The forecast coefficients are then multiplied by the principal components to obtain a forecast curve.

Usage

ftsmiterativeforecasts(object, components, iteration = 20)

Arguments

object An object of class fts.
components Number of principal components.
iteration Number of iterative one-step-ahead forecasts.

Details

1. Obtain a smooth curve $f_t(x)$ for each $t$ using a nonparametric smoothing technique.
2. Decompose the smooth curves via a functional principal component analysis.
3. Fit a univariate time series model to each of the principal component scores.
4. Forecast the principal component scores using the fitted time series models.
5. Multiply the forecast principal component scores by fixed principal components to obtain forecasts of $f_{n+h}(x)$.
6. The estimated variances of the error terms (smoothing error and model residual error) are used to compute prediction intervals for the forecasts.

Value

List with the following components:

mean An object of class fts containing point forecasts.
lower An object of class fts containing lower bound for prediction intervals.
upper An object of class fts containing upper bound for prediction intervals.
fitted An object of class fts of one-step-ahead forecasts for historical data.
error An object of class fts of one-step-ahead errors for historical data.
coeff List of objects of type forecast containing the coefficients and their forecasts.
ftsmweightselect

- **coeff.error**: One-step-ahead forecast errors for each of the coefficients.
- **var**: List containing the various components of variance: model, error, mean, total and coeff.
- **model**: Fitted `ftsm` model.
- **bootsamp**: An array of \( \text{dim} = c(p, B, h) \) containing the bootstrapped point forecasts. \( p \) is the number of variables. \( B \) is the number of bootstrap samples. \( h \) is the forecast horizon.

**Author(s)**

Han Lin Shang

**References**


**See Also**

`ftsm, plot.ftsf, plot.fm, residuals.fm, summary.fm`

**Examples**

```r
# Iterative one-step-ahead forecasts via functional principal component analysis.
ftsmiterativeforecasts(object = Australiasmoothfertility, components = 2, iteration = 5)
```

**Description**

The geometrically decaying weights are used to estimate the mean curve and functional principal components, where more weights are assigned to the more recent data than the data from the distant past.

**Usage**

```r
ftsmweightselect(data, ncomp = 6, ntestyear, errorcriterion = c("mae", "mse", "mape"))
```
Arguments

- **data**: An object of class `fts`.
- **ncomp**: Number of components.
- **ntestyear**: Number of holdout observations used to assess the forecast accuracy.
- **errorcriterion**: Error measure.

Details

The data set is split into a fitting period and forecasting period. Using the data in the fitting period, we compute the one-step-ahead forecasts and calculate the forecast error. Then, we increase the fitting period by one, and carry out the same forecasting procedure until the fitting period covers entire data set. The forecast accuracy is determined by the averaged forecast error across the years in the forecasting period. By using an optimization algorithm, we select the optimal weight parameter that would result in the minimum forecast error.

Value

Optimal weight parameter.

Note

Can be computational intensive, as it takes about half-minute to compute. For example, `ftsmweightselect(ElNinosmooth, ntestyear = 1)`.

Author(s)

Han Lin Shang

References


See Also

`ftsm`, `forecast.ftsm`

**hdfpca**

*High-dimensional functional principal component analysis*

Description

Fit a high dimensional functional principal component analysis model to a multiple-population of functional time series data.
Usage

hdfpca(y, order, q = sqrt(dim(y)[1])[2]), r)

Arguments

y A list, where each item is a population of functional time series. Each item is
a data matrix of dimension p by n, where p is the number of discrete points in
each function and n is the sample size
order The number of principal component scores to retain in the first step dimension
reduction
q The tuning parameter used in the first step dimension reduction, by default it is
equal to the square root of the sample size
r The number of factors to retain in the second step dimension reduction

Details

In the first step, dynamic functional principal component analysis is performed on each population
and then in the second step, factor models are fitted to the resulting principal component scores.
The high-dimensional functional time series are thus reduced to low-dimensional factors.

Value

y The input data
p The number of discrete points in each function
fitted A list containing the fitted functions for each population
m The number of populations
model Model 1 includes the first step dynamic functional principal component analysis
models, model 2 includes the second step high-dimensional principal compo-
nent analysis models
order Input order
r Input r

Author(s)

Y. Gao and H. L. Shang

References


See Also

forecast.hdfpca, hd_data

Examples

hd_model = hdfpca(hd_data, order = 2, r = 2)
**hd_data**

*Simulated high-dimensional functional time series*

**Description**

We generate $N$ populations of functional time series. For each $i \in \{1, \ldots, N\}$, the $i$th function at time $t \in \{1, \ldots, T\}$ is given by

$$X_t^{(i)}(u) = \sum_{p=1}^{2} \beta_{p,t}^{(i)} \gamma_p(u) + \theta_t^{(i)}(u),$$

where $\theta_t^{(i)}(u) = \sum_{p=3}^{\infty} \beta_{p,t}^{(i)} \gamma_p(u)$.

**Usage**

`data("hd_data")`

**Details**

The coefficients $\beta_{p,t}^{(i)}$ for all $N$ populations are combined and generated, for all $p \in N$, by

$$\beta_{p,t} = A_p f_{p,t},$$

where $\beta_{p,t} = \{\beta_{p,t}^1, \ldots, \beta_{p,t}^N\}$. Here, $A_p$ is an $N \times N$ matrix, and $f_{p,t}$ is an $N \times 1$ vector. Furthermore, we assume that the $\beta_{p,t}^{(i)}$s have mean 0 and variance 0 when $p > 3$, so we only construct the coefficients $\beta_{p,t}$ for $p \in \{1, 2, 3\}$.

The first set of coefficients $\beta_{1,t}$ for $N$ populations are generated with $\beta_{1,t} = A_1 f_{1,t}$. Each element in the matrix $A_1$ is generated by $a_{ij} = N^{-1/4} \times b_{ij}$, where $b_{ij} \sim N(2,4)$.

The factors $f_{1,t}$ are generated using an autoregressive model of order 1, i.e., AR(1). Define the $i$th element in vector $f_{1,t}$ as $f_{1,t}^{(i)}$. Then, $f_{1,t}^{(i)}$ is generated by $f_{1,t}^{(i)} = 0.5 \times f_{1,t-1}^{(i)} + \omega_t$, where $\omega_t$ are independent $N(0,1)$ random variables. We generate $f_{1,t}^{(i)}$ for all $i \in \{2, \ldots, N\}$ by $f_{1,t}^{(i)} = (1/N) \times g_t^{(i)}$, where $g_t^{(2)}, \ldots, g_t^{(N)}$ are also AR(1) and follow $g_t^{(i)} = 0.2 \times g_{t-1}^{(i)} + \omega_t$. It is then ensured that most of the variance of $\beta_{1,t}$ can be explained by one factor. The second coefficient $\beta_{2,t}$ are constructed the same way as $\beta_{1,t}$.

We also generate the third functional principal component scores $\beta_{3,t}$ but with small values. Moreover, $A_3$ is generated by $a_{ij} = N^{-1/4} \times b_{ij}$, where $b_{ij} \sim N(0,0.04)$. The factors $bmf_{3,t}$ are generated as $f_{1,t}$.

The three basis functions are constructed by $\gamma_1^{(i)}(u) = \sin(2\pi u + \pi i/2)$, $\gamma_2^{(i)}(u) = \cos(2\pi u + \pi i/2)$ and $\gamma_3^{(i)}(u) = \sin(4\pi u + \pi i/2)$, where $u \in [0,1]$. Finally, the functional time series for the $i$th population is constructed by

$$X_t^{(i)}(u) = \beta_{1,t} \gamma_1^{(i)}(u) + \beta_{2,t} \gamma_2^{(i)}(u) + \beta_{3,t} \gamma_3^{(i)}(u),$$

where $(\cdot)_i$ denotes the $i$th element of the vector.
References


See Also

hdfpca, forecast.hdfpca

Examples

data(hd_data)

---

is.fts Test for functional time series

Description

Tests whether an object is of class fts.

Usage

is.fts(x)

Arguments

x Arbitrary R object.

Author(s)

Rob J Hyndman

Examples

# check if ElNino is the class of the functional time series.
is.fts(x = ElNino_ERSST_region_1and2)
isfe.fts

Integrated Squared Forecast Error for models of various orders

Description

Computes integrated squared forecast error (ISFE) values for functional time series models of various orders.

Usage


isfe.fts(data, max.order = N - 3, N = 10, h = 5:10, method =
c("classical", "M", "rapca"), mean = TRUE, level = FALSE,
method = c("arima", "ar", "ets", "ets.na", "struct", "rwdrift",
"rw", "arfima"), lambda = 3, ...)

Arguments

data An object of class fts.
max.order Maximum number of principal components to fit.
N Minimum number of functional observations to be used in fitting a model.
h Forecast horizons over which to average.
method Method to use for principal components decomposition. Possibilities are “M”, “rapca” and “classical”.
mean Indicates if mean term should be included.
level Indicates if level term should be included.
fmethod Method used for forecasting. Current possibilities are “ets”, “arima”, “ets.na”, “struct”, “rwdrift” and “rw”.
lambda Tuning parameter for robustness when method = “M”.
... Additional arguments controlling the fitting procedure.

Value

Numeric matrix with (max.order+1) rows and length(h) columns containing ISFE values for models of orders 0:(max.order).

Note

This function can be very time consuming for data with large dimensionality or large sample size. By setting max.order small, computational speed can be dramatically increased.

Author(s)

Rob J Hyndman
long_run_covariance_estimation

Estimating long-run covariance function for a functional time series

Description

Bandwidth estimation in the long-run covariance function for a functional time series, using different types of kernel function

Usage

long_run_covariance_estimation(dat, C0 = 3, H = 3)

Arguments

dat A matrix of p by n, where p denotes the number of grid points and n denotes sample size
C0 A tuning parameter used in the adaptive bandwidth selection algorithm of Rice
H A tuning parameter used in the adaptive bandwidth selection algorithm of Rice

Value

An estimated covariance function of size (p by p)

Author(s)

Han Lin Shang

References


MAF_multivariate

See Also

fts

Examples

dum = long_run_covariance_estimation(dat = ElNino_OISST_region_1and2$y[,1:5])

MAF_multivariate

Maximum autocorrelation factors

Description

Dimension reduction via maximum autocorrelation factors

Usage

MAF_multivariate(data, threshold)

Arguments

data: A p by n data matrix, where p denotes the number of variables and n denotes the sample size
threshold: A threshold level for retaining the optimal number of factors

Value

MAF: Maximum autocorrelation factor scores
MAF_LOADING: Maximum autocorrelation factors
Z: Standardized original data
recon: Reconstruction via maximum autocorrelation factors
recon_err: Reconstruction errors between the standardized original data and reconstruction via maximum autocorrelation factors
ncomp_threshold: Number of maximum autocorrelation factors selected by explaining autocorrelation at and above a given level of threshold
ncomp_eigen_ratio: Number of maximum autocorrelation factors selected by eigenvalue ratio tests

Author(s)

Han Lin Shang

References

mean.fts

See Also

fts

Examples

MAF_multivariate(data = pm_10_GR_sqrt$y, threshold = 0.85)

---

**Description**

Computes mean of functional time series at each variable.

**Usage**

```r
## S3 method for class 'fts'
mean(x, method = c("coordinate", "FM", "mode", "RP", "RPD", "radius"),
       na.rm = TRUE, alpha, beta, weight, ...)
```

**Arguments**

- `x`: An object of class `fts`.
- `method`: Method for computing the mean function.
- `na.rm`: A logical value indicating whether NA values should be stripped before the computation proceeds.
- `alpha`: Tuning parameter when `method="radius"`.
- `beta`: Trimming percentage, by default it is 0.25, when `method="radius"`.
- `weight`: Hard thresholding or soft thresholding.
- `...`: Other arguments.

**Details**

If `method = "coordinate"`, it computes the coordinate-wise functional mean.
If `method = "FM"`, it computes the mean of trimmed functional data ordered by the functional depth of Fraiman and Muniz (2001).
If `method = "mode"`, it computes the mean of trimmed functional data ordered by \( h \)-modal functional depth.
If `method = "RP"`, it computes the mean of trimmed functional data ordered by random projection depth.
If `method = "RPD"`, it computes the mean of trimmed functional data ordered by random projection derivative depth.
If `method = "radius"`, it computes the mean of trimmed functional data ordered by the notion of alpha-radius.
Value

A list containing \(x = \) variables and \(y = \) mean rates.

Author(s)

Rob J Hyndman, Han Lin Shang

References


See Also

*median.fts, var.fts, sd.fts, quantile.fts*

Examples

```r
# Calculate the mean function by the different depth measures.
mean(x = ElNino.ERSST_region.land2, method = "coordinate")
mean(x = ElNino.ERSST_region.land2, method = "FM")
mean(x = ElNino.ERSST_region.land2, method = "mode")
mean(x = ElNino.ERSST_region.land2, method = "RP")
mean(x = ElNino.ERSST_region.land2, method = "RPD")
mean(x = ElNino.ERSST_region.land2, method = "radius", alpha = 0.5, beta = 0.25, weight = "hard")
mean(x = ElNino.ERSST_region.land2, method = "radius", alpha = 0.5, beta = 0.25, weight = "soft")
```
Description

Computes median of functional time series at each variable.

Usage

```r
## S3 method for class 'fts'
median(x, na.rm, method = c("hossjercroux", "coordinate", "FM", "mode", 
  "RP", "RPD", "radius"), alpha, beta, weight, ...)
```

Arguments

- **x**: An object of class `fts`.
- **na.rm**: Remove any missing value.
- **method**: Method for computing median.
- **alpha**: Tuning parameter when `method="radius"`.
- **beta**: Trimming percentage, by default it is 0.25, when `method="radius"`.
- **weight**: Hard thresholding or soft thresholding.
- **...**: Other arguments.

Details

- If `method = "coordinate"`, it computes a coordinate-wise median.
- If `method = "hossjercroux"`, it computes the L1-median using the Hossjer-Croux algorithm.
- If `method = "FM"`, it computes the median of trimmed functional data ordered by the functional depth of Fraiman and Muniz (2001).
- If `method = "mode"`, it computes the median of trimmed functional data ordered by $h$-modal functional depth.
- If `method = "RP"`, it computes the median of trimmed functional data ordered by random projection depth.
- If `method = "RPD"`, it computes the median of trimmed functional data ordered by random projection derivative depth.
- If `method = "radius"`, it computes the mean of trimmed functional data ordered by the notion of alpha-radius.

Value

A list containing `x = variables` and `y = median rates`.

Author(s)

Rob J Hyndman, Han Lin Shang
References


See Also

`mean.fts, var.fts, sd.fts, quantile.fts`

Examples

```r
# Calculate the median function by the different depth measures.
median(x = ElNino_ERSST_region_1and2, method = "hossjercroux")
median(x = ElNino_ERSST_region_1and2, method = "coordinate")
median(x = ElNino_ERSST_region_1and2, method = "FM")
median(x = ElNino_ERSST_region_1and2, method = "mode")
median(x = ElNino_ERSST_region_1and2, method = "RP")
median(x = ElNino_ERSST_region_1and2, method = "RPD")
median(x = ElNino_ERSST_region_1and2, method = "radius",
       alpha = 0.5, beta = 0.25, weight = "hard")
median(x = ElNino_ERSST_region_1and2, method = "radius",
       alpha = 0.5, beta = 0.25, weight = "soft")
```

MFDM

Multilevel functional data method

Description

Fit a multilevel functional principal component model. The function uses two-step functional principal component decompositions.
Usage

MFDM(mort_female, mort_male, mort_ave, percent_1 = 0.95, percent_2 = 0.95, fh, 
level = 80, alpha = 0.2, MCMCiter = 100, fmethod = c("auto_arima", "ets"), 
BC = c(FALSE, TRUE), lambda)

Arguments

mort_female  Female mortality (p by n matrix), where p denotes the dimension and n denotes 
the sample size.

mort_male     Male mortality (p by n matrix).

mort_ave      Total mortality (p by n matrix).

percent_1    Cumulative percentage used for determining the number of common functional 
principal components.

percent_2    Cumulative percentage used for determining the number of sex-specific func-
tional principal components.

fh            Forecast horizon.

level         Nominal coverage probability of a prediction interval.

alpha         1 - Nominal coverage probability.

MCMCiter      Number of MCMC iterations.

fmethod       Univariate time-series forecasting method.

BC            If Box-Cox transformation is performed.

lambda        If BC = TRUE, specify a Box-Cox transformation parameter.

Details

The basic idea of multilevel functional data method is to decompose functions from different sub-
populations into an aggregated average, a sex-specific deviation from the aggregated average, a 
common trend, a sex-specific trend and measurement error. The common and sex-specific trends 
are modelled by projecting them onto the eigenvectors of covariance operators of the aggregated 
and sex-specific centred stochastic process, respectively.

Value

first_percent Percentage of total variation explained by the first common functional principal 
component.

female_percent Percentage of total variation explained by the first female functional principal 
component in the residual.

male_percent  Percentage of total variation explained by the first male functional principal 
component in the residual.

mort_female_fore Forecast female mortality in the original scale.

mort_male_fore Forecast male mortality in the original scale.
Note

It can be quite time consuming, especially when MCMCiter is large.

Author(s)

Han Lin Shang

References


See Also

`ftsm`, `forecast.ftsm`, `fplsr`, `forecastfplsr`

MFPCA

*Multilevel functional principal component analysis for clustering*

Description

A multilevel functional principal component analysis for performing clustering analysis

Usage

`MFPCA(y, M = NULL, J = NULL, N = NULL)`

Arguments

- **y**: A data matrix containing functional responses. Each row contains measurements from a function at a set of grid points, and each column contains measurements of all functions at a particular grid point
- **M**: Number of countries
- **J**: Number of functional responses in each country
- **N**: Number of grid points per function
Value

K1  Number of components at level 1
K2  Number of components at level 2
K3  Number of components at level 3
lambda1 A vector containing all level 1 eigenvalues in non-increasing order
lambda2 A vector containing all level 2 eigenvalues in non-increasing order
lambda3 A vector containing all level 3 eigenvalues in non-increasing order
phi1  A matrix containing all level 1 eigenfunctions. Each row contains an eigenfunction evaluated at the same set of grid points as the input data. The eigenfunctions are in the same order as the corresponding eigenvalues
phi2  A matrix containing all level 2 eigenfunctions. Each row contains an eigenfunction evaluated at the same set of grid points as the input data. The eigenfunctions are in the same order as the corresponding eigenvalues
phi3  A matrix containing all level 3 eigenfunctions. Each row contains an eigenfunction evaluated at the same set of grid points as the input data. The eigenfunctions are in the same order as the corresponding eigenvalues
scores1 A matrix containing estimated level 1 principal component scores. Each row corresponds to the level 1 scores for a particular subject in a cluster. The number of rows is the same as that of the input matrix y. Each column contains the scores for a level 1 component for all subjects
scores2 A matrix containing estimated level 2 principal component scores. Each row corresponds to the level 2 scores for a particular subject in a cluster. The number of rows is the same as that of the input matrix y. Each column contains the scores for a level 2 component for all subjects.
scores3 A matrix containing estimated level 3 principal component scores. Each row corresponds to the level 3 scores for a particular subject in a cluster. The number of rows is the same as that of the input matrix y. Each column contains the scores for a level 3 component for all subjects.
mu A vector containing the overall mean function
eta A matrix containing the deviation from overall mean function to country-specific mean function. The number of rows is the number of countries
Rj  Common trend
Uij Country-specific mean function

Author(s)

Chen Tang, Yanrong Yang and Han Lin Shang

See Also

mftsc
**Description**

Clustering the multiple functional time series. The function uses the functional panel data model to cluster different time series into subgroups.

**Usage**

```r
mftsc(X, alpha)
```

**Arguments**

- **X**: A list of sets of smoothed functional time series to be clustered, for each object, it is a \( p \times q \) matrix, where \( p \) is the sample size and \( q \) is the number of grid points of the function.
- **alpha**: A value input for adjusted rand index to measure similarity of the memberships with last iteration, can be any value big than 0.9.

**Details**

As an initial step, conventional k-means clustering is performed on the dynamic FPC scores, then an iterative membership updating process is applied by fitting the MFPCA model.

**Value**

- **iteration**: the number of iterations until convergence.
- **membership**: a list of all the membership matrices at each iteration.
- **member.final**: the final membership.

**Author(s)**

Chen Tang, Yanrong Yang and Han Lin Shang

**See Also**

MFPCA

**Examples**

```r
## Not run:
data(sim_ex_cluster)
cluster_result<mftsc(X=sim_ex_cluster, alpha=0.99)
cluster_result$member.final

## End(Not run)
```
**Description**
Computes bootstrap or smoothed bootstrap samples based on either independent and identically distributed functional data or functional time series.

**Usage**
```r
pcscorebootstrapdata(dat, bootrep, statistic, bootmethod = c("st", "sm", "mvn", "stiefel", "meboot"), smo)
```

**Arguments**
- `dat`: An object of class `matrix`.
- `bootrep`: Number of bootstrap samples.
- `statistic`: Summary statistics.
- `bootmethod`: Bootstrap method. When `bootmethod = "st"`, the sampling with replacement is implemented. To avoid the repeated bootstrap samples, the smoothed bootstrap method can be implemented by adding multivariate Gaussian random noise. When `bootmethod = "mvn"`, the bootstrapped principal component scores are drawn from a multivariate Gaussian distribution with the mean and covariance matrices of the original principal component scores. When `bootmethod = "stiefel"`, the bootstrapped principal component scores are drawn from a Stiefel manifold with the mean and covariance matrices of the original principal component scores. When `bootmethod = "meboot"`, the bootstrapped principal component scores are drawn from a maximum entropy algorithm of Vinod (2004).
- `smo`: Smoothing parameter.

**Details**
We will presume that each curve is observed on a grid of $T$ points with $0 \leq t_1 < t_2 \ldots < t_T \leq \tau$. Thus, the raw data set $(X_1, X_2, \ldots, X_n)$ of $n$ observations will consist of an $n \times T$ data matrix. By applying the singular value decomposition, $X_1, X_2, \ldots, X_n$ can be decomposed into $X = ULR^\top$, where the crossproduct of $U$ and $R$ is identity matrix.

Holding the mean and $L$ and $R$ fixed at their realized values, there are four re-sampling methods that differ mainly by the ways of re-sampling $U$.

(a) Obtain the re-sampled singular column matrix by randomly sampling with replacement from the original principal component scores.

(b) To avoid the appearance of repeated values in bootstrapped principal component scores, we adapt a smooth bootstrap procedure by adding a white noise component to the bootstrap.
(c) Because principal component scores follow a standard multivariate normal distribution asymptotically, we can randomly draw principal component scores from a multivariate normal distribution with mean vector and covariance matrix of original principal component scores.

(d) Because the crossproduct of U is identity matrix, U is considered as a point on the Stiefel manifold, that is the space of n orthogonal vectors, thus we can randomly draw principal component scores from the Stiefel manifold.

Value

bootdata Bootstrap samples. If the original data matrix is p by n, then the bootstrapped data are p by n by bootrep.

meanfunction Bootstrap summary statistics. If the original data matrix is p by n, then the bootstrapped summary statistics is p by bootrep.

Author(s)

Han Lin Shang

References


See Also

fbootstrap

Examples

# Bootstrapping the distribution of a summary statistics of functional data.
boot1 = pcscorebootstrapdata(ElNino_ERSST_region_1and2$y, 400, "mean", bootmethod = "st")
boot2 = pcscorebootstrapdata(ElNino_ERSST_region_1and2$y, 400, "mean", bootmethod = "sm", smo = 0.05)
boot3 = pcscorebootstrapdata(ElNino_ERSST_region_1and2$y, 400, "mean", bootmethod = "mnv")
boot4 = pcscorebootstrapdata(ElNino_ERSST_region_1and2$y, 400, "mean", bootmethod = "stiefel")
boot5 = pcscorebootstrapdata(ElNino_ERSST_region_1and2$y, 400, "mean", bootmethod = "meboot")
Description

When \texttt{class(x)[1] = ftsm}, plot showing the principal components in the top row of plots and the coefficients in the bottom row of plots.

When \texttt{class(x)[1] = fm}, plot showing the predictor scores in the top row of plots and the response loadings in the bottom row of plots.

Usage

```r
## S3 method for class 'fm'
plot(x, order, xlab1 = x$y$xname, ylab1 = "Principal component",
     xlab2 = "Time", ylab2 = "Coefficient", mean.lab = "Mean",
     level.lab = "Level", main.title = "Main effects", interaction.title
     = "Interaction", basiscol = 1, coeffcol = 1, outlier.col = 2,
     outlier.pch = 19, outlier.cex = 0.5, ...)
```

Arguments

- **x**: Output from \texttt{ftsm} or \texttt{fplsr}.
- **order**: Number of principal components to plot. Default is all principal components in a model.
- **xlab1**: x-axis label for principal components.
- **xlab2**: x-axis label for coefficient time series.
- **ylab1**: y-axis label for principal components.
- **ylab2**: y-axis label for coefficient time series.
- **mean.lab**: Label for mean component.
- **level.lab**: Label for level component.
- **main.title**: Title for main effects.
- **interaction.title**: Title for interaction terms.
- **basiscol**: Colors for principal components if \texttt{plot.type = "components"}.
- **coeffcol**: Colors for time series coefficients if \texttt{plot.type = "components"}.
- **outlier.col**: Colors for outlying years.
- **outlier.pch**: Plotting character for outlying years.
- **outlier.cex**: Size of plotting character for outlying years.
- ...: Plotting parameters.

Value

Function produces a plot.
Author(s)
Rob J Hyndman

References

See Also
ftsm, forecast.ftsm, residuals.fm, summary.fm, plot.fmres, plot.ftsf

Examples
plot(x = ftsm(y = ElNino_ERSST_region_1and2))

plot.fmres

Plot residuals from a fitted functional model.

Description
Functions to produce a plot of residuals from a fitted functional model.

Usage
## S3 method for class 'fmres'
plot(x, type = c("image", "fts", "contour", "filled.contour", "persp"), xlab = "Year", ylab = "Age", zlab = "Residual", ...)

Arguments
x Generated by residuals(fit), where fit is the output from ftsm or fplsr.
type Type of plot to use. Possibilities are image, fts, contour, filled.contour and persp.
xlab Label for x-axis.
ylab Label for y-axis.
zlab Label for z-axis.
... Plotting parameters.

Value
Produces a plot.
Author(s)
Rob J Hyndman

See Also
fts, forecast.ftsm, plot.fm, plot.fmres, residuals.fm, summary.fm

Examples
# colorspace package was used to provide a more coherent color option.
plot(residuals(fts(y = EINino_ERSST_region_land2)), type = "filled.contour", xlab = "Month", ylab = "Residual sea surface temperature")

Description
Plot fitted model components for a functional time series model.

Usage
## S3 method for class 'ftsf'
plot(x, plot.type = c("function", "components", "variance"), components, xlab1 = fit$y$xname, ylab1 = "Basis function", xlab2 = "Time", ylab2 = "Coefficient", mean.lab = "Mean", level.lab = "Level", main.title = "Main effects", interaction.title = "Interaction", vcol = 1:3, shadecols = 7, fcol = 4, basiscol = 1, coeffcol = 1, outlier.col = 2, outlier.pch = 19, outlier.cex = 0.5,...)

Arguments
x Output from forecast.ftsm.
plot.type Type of plot.
components Number of principal components.
xlab1 x-axis label for principal components.
xlab2 x-axis label for coefficient time series.
ylab1 y-axis label for principal components.
ylab2 y-axis label for coefficient time series.
mean.lab Label for mean component.
level.lab Label for level component.
main.title Title for main effects.
interaction.title Title for interaction terms.
vcol  Colors to use if plot.type = "variance".
shadecols  Color for shading of prediction intervals when plot.type = "components".
fcol  Color of point forecasts when plot.type = "components".
basiccol  Colors for principal components if plot.type = "components".
coeffcol  Colors for time series coefficients if plot.type = "components".
outlier.col  Colors for outlying years.
outlierpch  Plotting character for outlying years.
outlier.cex  Size of plotting character for outlying years.

Details

When plot.type = "function", it produces a plot of the forecast functions;
When plot.type = "components", it produces a plot of the principal components and coefficients
with forecasts and prediction intervals for each coefficient;
When plot.type = "variance", it produces a plot of the variance components.

Value

Function produces a plot.

Author(s)

Rob J Hyndman

References


See Also

ftsm, plot.fm, plot.fmres, residuals.fm, summary.fm

Examples

plot(x = forecast(object = ftsm(y = ElNino_ERSST_region_land2)))
Plot fitted model components for a functional time series model

Description

Plot showing the basis functions in the top row of plots and the coefficients in the bottom row of plots.

Usage

```r
## S3 method for class 'ftsm'
plot(x, components, components.start = 0, xlab1 = x$y$name, ylab1 = "Basis function", xlab2 = "Time", ylab2 = "Coefficient", mean.lab = "Mean", level.lab = "Level", main.title = "Main effects", interaction.title = "Interaction", basiscol = 1, coeffcol = 1, outlier.col = 2, outlier.pch = 19, outlier.cex = 0.5, ...)
```

Arguments

- `x`: Output from `ftsm`.
- `components`: Number of principal components to plot.
- `components.start`: Plotting specified component.
- `xlab1`: x-axis label for basis functions.
- `xlab2`: x-axis label for coefficient time series.
- `ylab1`: y-axis label for basis functions.
- `ylab2`: y-axis label for coefficient time series.
- `mean.lab`: Label for mean component.
- `level.lab`: Label for level component.
- `main.title`: Title for main effects.
- `interaction.title`: Title for interaction terms.
- `basiscol`: Colors for basis functions if plot.type="components".
- `coeffcol`: Colors for time series coefficients if plot.type="components".
- `outlier.col`: Colour for outlying years.
- `outlier.pch`: Plotting character for outlying years.
- `outlier.cex`: Size of plotting character for outlying years.
- `...`: Plotting parameters.

Value

None. Function produces a plot.


**Author(s)**

Rob J Hyndman

**References**


**See Also**

`forecast.ftsm`, `ftsm`, `plot.fm`, `plot.ftsf`, `residuals.fm`, `summary.fm`

**Examples**

```r
# plot different principal components.
plot.ftsm(ftsm(y = ELNino_ERSST_region_1and2, order = 2), components = 2)
```

---

**plotfplsr**

*Plot fitted model components for a functional time series model*

**Description**

Plot showing the basis functions of the predictors in the top row, followed by the basis functions of the responses in the second row, then the coefficients in the bottom row of plots.

**Usage**

```r
plotfplsr(x, xlab1 = x$ypred$xname, ylab1 = "Basis function", xlab2 = "Time", ylab2 = "Coefficient", mean.lab = "Mean", interaction.title = "Interaction")
```

**Arguments**

- `x`: Output from `fplsr`.
- `xlab1`: x-axis label for basis functions.
- `ylab1`: y-axis label for basis functions.
- `xlab2`: x-axis label for coefficient time series.
- `ylab2`: y-axis label for coefficient time series.
- `mean.lab`: Label for mean component.
- `interaction.title`: Title for interaction terms.

**Value**

None. Function produces a plot.
pm_10_GR

Author(s)

Han Lin Shang

References


See Also

`forecast.ftsm`, `ftsm`, `plot.fm`, `plot.ftsf`, `residuals.fm`, `summary.fm`

Examples

```r
# Fit the data by the functional partial least squares.
ausfplsr = fplsr(data = ElNino_ERSST_region_1and2, order = 2)
plotfplsr(x = ausfplsr)
```

pm_10_GR  Particulate Matter Concentrations (pm10)

Description

This data set consists of half-hourly measurement of the concentrations (measured in ug/m³) of particular matter with an aerodynamic diameter of less than 10μm, abbreviated PM10, in ambient air taken in Graz-Mitte, Austria from October 1, 2010 until March 31, 2011. To stabilise the variance, a square-root transformation can be applied to the data.

Usage

data(pm_10_GR)

Details

As epidemiological and toxicological studies have pointed to negative health effects, European Union (EU) regulation sets pollution standards for the level of the concentration. Policy makers have to ensure compliance with these EU rules and need reliable statistical tools to determine, and justify the public, appropriate measures such as partial traffic regulation (see Stadlober, Hormann and Pfeiler, 2008).

Source

Thanks Professor Siegfried. Hormann for providing this data set. The original data source is [https://www.umwelt.steiermark.at/cms/](https://www.umwelt.steiermark.at/cms/)
References


Examples

```r
plot(pm_10_GR)
```

---

<table>
<thead>
<tr>
<th>quantile</th>
<th>Quantile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic functions for quantile.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

```r
quantile(x, ...)
```

Arguments

- `x` Numeric vector whose sample quantiles are wanted, or an object of a class for which a method has been defined.
- `...` Arguments passed to specific methods.

Value

Refer to specific methods. For numeric vectors, see the `quantile` functions in the stats package.

Author(s)

Han Lin Shang

See Also

`quantile.fts`
quantile.fts

Quantile functions for functional time series

Description

Computes quantiles of functional time series at each variable.

Usage

## S3 method for class 'fts'
quantile(x, probs, ...)

Arguments

x
An object of class fts.
probs
Quantile percentages.
...
Other arguments.

Value

Return quantiles for each variable.

Author(s)

Han Lin Shang

See Also

mean.fts, median.fts, var.fts, sd.fts

Examples

quantile(x = ElNino_ERSST_region_land2)

residuals.fm

Compute residuals from a functional model

Description

After fitting a functional model, it is useful to inspect the residuals. This function extracts the relevant information from the fit object and puts it in a form suitable for plotting with image, persp, contour, filled.contour, etc.

Usage

## S3 method for class 'fm'
residuals(object, ...)

Arguments

- `object`: Output from `ftsm` or `fplsr`.
- `...`: Other arguments.

Value

Produces an object of class “fmres” containing the residuals from the model.

Author(s)

Rob J Hyndman

References


See Also

- `ftsm`, `forecast.ftsm`, `summary.fm`, `plot.fm`, `plot.fmres`

Examples

```r
plot(residuals(object = ftsm(y = ElNino_ERSST_region_1and2)),
     xlab = "Year", ylab = "Month")
```

---

**sd**

*Standard deviation*

Description

Generic functions for standard deviation.

Usage

```r
sd(...)
```
**sd.fts**

**Arguments**

... Arguments passed to specific methods.

**Details**

The `sd` functions in the `stats` package are replaced by `sd.default`.

**Value**

Refer to specific methods. For numeric vectors, see the `sd` functions in the `stats` package.

**Author(s)**

Han Lin Shang

**See Also**

`sd.fts`

---

### `sd.fts`

*Standard deviation functions for functional time series*

**Description**

Computes standard deviation of functional time series at each variable.

**Usage**

```r
## S3 method for class 'fts'
sd(x, method = c("coordinate", "FM", "mode", "RP", "RPD", "radius"),
   trim = 0.25, alpha, weight,...)
```

**Arguments**

- **x**: An object of class fts.
- **method**: Method for computing median.
- **trim**: Percentage of trimming.
- **alpha**: Tuning parameter when `method="radius"`.
- **weight**: Hard thresholding or soft thresholding.
- **...**: Other arguments.
Details

If method = "coordinate", it computes coordinate-wise standard deviation functions.

If method = "FM", it computes the standard deviation functions of trimmed functional data ordered by the functional depth of Fraiman and Muniz (2001).

If method = "mode", it computes the standard deviation functions of trimmed functional data ordered by $h$-modal functional depth.

If method = "RP", it computes the standard deviation functions of trimmed functional data ordered by random projection depth.

If method = "RPD", it computes the standard deviation functions of trimmed functional data ordered by random projection with derivative depth.

If method = "radius", it computes the standard deviation function of trimmed functional data ordered by the notion of alpha-radius.

Value

A list containing $x =$ variables and $y =$ standard deviation rates.

Author(s)

Han Lin Shang

References


See Also

mean.fts, median.fts, var.fts, quantile.fts
Examples

# Fraiman-Muniz depth was arguably the oldest functional depth.
sd(x = ElNino_ERSST_region_1and2, method = "FM")
sd(x = ElNino_ERSST_region_1and2, method = "coordinate")
sd(x = ElNino_ERSST_region_1and2, method = "mode")
sd(x = ElNino_ERSST_region_1and2, method = "RP")
sd(x = ElNino_ERSST_region_1and2, method = "RPD")
sd(x = ElNino_ERSST_region_1and2, method = "radius",
alpha = 0.5, weight = "hard")
sd(x = ElNino_ERSST_region_1and2, method = "radius",
alpha = 0.5, weight = "soft")

Description

We generate 2 groups of \( m \) functional time series. For each \( i \) in \( \{1, \ldots, m\} \) in a given cluster \( c, c \) in \( \{1,2\} \), the \( t \) th function, \( t \) in \( \{1, \ldots, T\} \), is given by

\[
Y_{it}^{(c)}(x) = \mu^{(c)}(x) + \sum_{k=1}^{2} \xi_{ik}^{(c)} \rho_{k}^{(c)}(x) + \sum_{l=1}^{2} \zeta_{itl}^{(c)} \psi_{l}^{(c)}(x) + \upsilon_{it}^{(c)}(x)
\]

Usage

data("sim_ex_cluster")

Details

The mean functions for each of these two clusters are set to be \( \mu^{(1)}(x) = 2(x - 0.25)^2 \) and \( \mu^{(2)}(x) = 2(x - 0.4)^2 + 0.1 \).

While the variates \( \xi_{ik}^{(c)} = (\xi_{1k}^{(c)}, \xi_{2k}^{(c)}, \ldots, \xi_{Tk}^{(c)})^T \) for both clusters, are generated from autoregressive of order 1 with parameter 0.7, while the variates \( \zeta_{it1}^{(c)} \) and \( \zeta_{it2}^{(c)} \) for both clusters, are generated from independent and identically distributed \( \mathcal{N}(0, 0.5) \) and \( \mathcal{N}(0, 0.25) \), respectively.

The basis functions for the common-time trend for the first cluster, \( \rho_{k}^{(1)}(x) \), for \( k \) in \( \{1,2\} \) are \( \text{sqrt}(2) \ast \sin(\pi \ast (0 : 200/200)) \) and \( \text{sqrt}(2) \ast \cos(\pi \ast (0 : 200/200)) \) respectively; and the basis functions for the common-time trend for the second cluster, \( \rho_{k}^{(2)}(x) \), for \( k \) in \( \{1,2\} \) are \( \text{sqrt}(2) \ast \sin(2\pi \ast (0 : 200/200)) \) and \( \text{sqrt}(2) \ast \cos(2\pi \ast (0 : 200/200)) \) respectively.

The basis functions for the residual for the first cluster, \( \psi_{l}^{(1)}(x) \), for \( l \) in \( \{1,2\} \) are \( \text{sqrt}(2) \ast \sin(3\pi \ast (0 : 200/200)) \) and \( \text{sqrt}(2) \ast \cos(3\pi \ast (0 : 200/200)) \) respectively; and the basis functions for the residual for the second cluster, \( \psi_{l}^{(2)}(x) \), for \( l \) in \( \{1,2\} \) are \( \text{sqrt}(2) \ast \sin(4\pi \ast (0 : 200/200)) \) and \( \text{sqrt}(2) \ast \cos(4\pi \ast (0 : 200/200)) \) respectively.

The measurement error \( \upsilon_{it} \) for each continuum \( x \) is generated from independent and identically distributed \( \mathcal{N}(0, 0.2^2) \)
Examples

data(sim_ex_cluster)

stop_time_detect  
Detection of the optimal stopping time in a curve time series

Description

Detecting the optimal stopping time for the glue curing of wood panels in an automatic process environment.

Usage

stop_time_detect(data, forecasting_method = c("ets", "arima", "rw"))

Arguments

data  
An object of class fts

forecasting_method  
A univariate time series forecasting method

Value

break_points_strucchange  
Breakpoints detected by the regression approach

break_points_ecp  
Breakpoints detected by the distance-based approach

err_forward  
Forward integrated squared forecast errors

err_backward  
Backward integrated squared forecast errors (ISFEs)

ncomp_select_forward  
Number of components selected by the eigenvalue ratio tests based on the forward ISFEs

ncomp_select_backward  
Number of components selected by the eigenvalue ratio tests based on the backward ISFEs

Author(s)

Han Lin Shang

References

**stop_time_sim_data**

Simulated functional time series from a functional autoregression of order one

Description

For detecting the optimal stopping time, we simulate a curve time series that follows a functional autoregression of order 1, with a breakpoint in the middle point of the entire sample.

Usage

```r
stop_time_sim_data(sample_size, omega, seed_number)
```

Arguments

- `sample_size`: Number of curves
- `omega`: Noise level
- `seed_number`: Random seed number

Value

An object of class `fts`

Author(s)

Han Lin Shang

See Also

`stop_time_detect`

Examples

```r
stop_time_sim_data(sample_size = 401, omega = 0.1, seed_number = 123)
```

---

**summary.fm**

Summary for functional time series model

Description

Summarizes a basis function model fitted to a functional time series. It returns various measures of goodness-of-fit.

Usage

```r
## S3 method for class 'fm'
summary(object, ...)
```
Arguments

object: Output from `ftsm` or `fplsr`.

... Other arguments.

Value

None.

Author(s)

Rob J Hyndman

See Also

`ftsm`, `forecast.ftsm`, `residuals.fm`, `plot.fm`, `plot.fmres`

Examples

```r
summary(object = ftsm(y = ElNino_ERSST_region_1and2))
```

---

**Test stationarity of functional time series**

Description

A hypothesis test for stationarity of functional time series.

Usage

```r
T_stationary(sample, L = 49, J = 500, MC_rep = 1000, cumulative_var = .90, Ker1 = FALSE, Ker2 = TRUE, h = ncol(sample)^.5, pivotal = FALSE)
```

Arguments

- **sample**: A matrix of discretised curves of dimension (p by n), where p represents the dimensionality and n represents sample size.
- **L**: Number of Fourier basis functions.
- **J**: Truncation level used to approximate the distribution of the squared integrals of Brownian bridges that appear in the limit distribution.
- **MC_rep**: Number of replications.
- **cumulative_var**: Amount of variance explained.
- **Ker1**: Flat top kernel in (4.1) of Horvath et al. (2014).
- **Ker2**: Flat top kernel in (7) of Politis (2003).
- **h**: Kernel bandwidth.
- **pivotal**: If `pivotal = TRUE`, a pivotal statistic is used.
Details

As in traditional (scalar and vector) time series analysis, many inferential procedures for functional time series assume stationarity. Stationarity is required for functional dynamic regression models, for bootstrap and resampling methods for functional time series and for the functional analysis of volatility.

Value

p-value When p-value is less than any level of significance, we reject the null hypothesis and conclude that the tested functional time series is not stationary.

Author(s)

Greg. Rice and Han Lin Shang

References


See Also

farforecast

Examples

result = T_stationary(sample = pm_10_GR_sqrt$y)
result_pivotal = T_stationary(sample = pm_10_GR_sqrt$y, J = 100, MC_rep = 5000, h = 20, pivotal = TRUE)

<table>
<thead>
<tr>
<th>var</th>
<th>Variance</th>
</tr>
</thead>
</table>

Description

Generic functions for variance.

Usage

var(…)

Arguments

… Arguments passed to specific methods.
Details

The \texttt{cor} functions in the \texttt{stats} package are replaced by \texttt{var.default}.

Value

Refer to specific methods. For numeric vectors, see the \texttt{cor} functions in the \texttt{stats} package.

Author(s)

Rob J Hyndman and Han Lin Shang

See Also

\texttt{var.fts}

\begin{tabular}{l}
\texttt{var.fts} \\
\textit{Variance functions for functional time series}
\end{tabular}

Description

Computes variance functions of functional time series at each variable.

Usage

\begin{verbatim}
## S3 method for class 'fts'
var(x, method = c("coordinate", "FM", "mode", "RP", "RPD", "radius"),
    trim = 0.25, alpha, weight, ...)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} An object of class \texttt{fts}.
\item \texttt{method} Method for computing median.
\item \texttt{trim} Percentage of trimming.
\item \texttt{alpha} Tuning parameter when \texttt{method="radius"}.
\item \texttt{weight} Hard thresholding or soft thresholding.
\item ... Other arguments.
\end{itemize}

Details

If \texttt{method = "coordinate"}, it computes coordinate-wise variance.

If \texttt{method = "FM"}, it computes the variance of trimmed functional data ordered by the functional depth of Fraiman and Muniz (2001).

If \texttt{method = "mode"}, it computes the variance of trimmed functional data ordered by $h$-modal functional depth.
If method = "RP", it computes the variance of trimmed functional data ordered by random projection depth.
If method = "RPD", it computes the variance of trimmed functional data ordered by random projection derivative depth.
If method = "radius", it computes the standard deviation function of trimmed functional data ordered by the notion of alpha-radius.

Value
A list containing \( x \) = variables and \( y \) = variance rates.

Author(s)
Han Lin Shang

References

See Also
mean.fts, median.fts, sd.fts, quantile.fts

Examples
```r
# Fraiman-Muniz depth was arguably the oldest functional depth.
var(x = ElNino_ERSST_region_land2, method = "FM")
var(x = ElNino_ERSST_region_land2, method = "coordinate")
var(x = ElNino_ERSST_region_land2, method = "mode")
var(x = ElNino_ERSST_region_land2, method = "RP")
```
var(x = ElNino_ERSST_region_land2, method = "RPD")
var(x = ElNino_ERSST_region_land2, method = "radius",
    alpha = 0.5, weight = "hard")
var(x = ElNino_ERSST_region_land2, method = "radius",
    alpha = 0.5, weight = "soft")
Index

* datasets
  hd_data, 36
  pm_10_GR, 57
  sim_ex_cluster, 63
  stop_time_sim_data, 65

* hplot
  plot.fm, 51
  plot.fmres, 52
  plot.ftsf, 53

* methods
  dmfpca, 7
  ER_GR, 15
  error, 13
  facf, 17
  ftsmweightselect, 33
  MAF_multivariate, 40
  mean.fts, 41
  median.fts, 43
  MFPCA, 46
  quantile.fts, 59
  sd.fts, 61
  stop_time_detect, 64
  var.fts, 68

* method
  long_run_covariance_estimation, 39

* models
  centre, 5
  dynamic_FLR, 8
  dynupdate, 10
  extract, 16
  farforecast, 18
  forecast.ftsm, 21
  forecast.hdfpca, 23
  forecastfplsr, 25
  fplsr, 26
  ftsm, 29
  ftsmIterativeForecasts, 32
  hdfpca, 34
  isfe.fts, 38
  MFDM, 44
  pcsscorebootstrapdata, 49
  plot.ftsm, 55
  plotfplsr, 56
  quantile, 58
  residuals.fm, 59
  sd, 60
  summary.fm, 65
  T_stationary, 66
  var, 67

* multivariate
  fbootstrap, 19

* package
  ftsa-package, 3

* ts
  diff.fts, 6
  is.fts, 37

  centre, 5
  contour, 52
  cor, 68

  diff.fts, 6
  dmfpca, 7
  dynamic_FLR, 8
  dynupdate, 10, 10

  ER_GR, 15
  error, 13
  ets, 21
  extract, 16

  facf, 17
  farforecast, 18, 67
  fbootstrap, 19, 50
  fds, 18
  filled.contour, 52
  forecast.ftsm, 12, 19, 21, 25, 29, 31, 34, 39, 46, 52, 53, 56, 57, 60, 66
  forecast.hdfpca, 23, 35, 37

71
INDEX

forecastfplsr, 19, 23, 25, 46
fplsr, 26, 46, 51, 52, 56, 60, 66
fts, 40
ftsa(ftsa-package), 3
ftsa-package, 3
ftsm, 12, 15, 21–23, 25, 29, 33, 34, 39, 41, 46, 51–57, 60, 66
ftsmiterativeforecasts, 32
ftsmweightselect, 31, 33

hd_data, 24, 35, 36
hdfpca, 24, 34, 37

image, 52
is.fts, 37
isfe.fts, 38

long_run_covariance_estimation, 39

MAF_multivariate, 40
mean.fts, 6, 41, 44, 59, 62, 69
median.fts, 6, 42, 43, 59, 62, 69
MFDM, 44
MFPCA, 46, 48
mftsc, 8, 47, 48

pcscorebootstrapdata, 6, 20, 49
persp, 52
plot.fm, 12, 23, 25, 29, 31, 33, 39, 51, 53, 54, 56, 57, 60, 66
plot.fmres, 29, 39, 52, 53, 54, 60, 66
plot.ftsf, 23, 25, 31, 33, 52, 53, 56, 57
plot.ftsm, 55
plotfplsr, 56
pm_10_GR, 57
pm_10_GR_sqrt (pm_10_GR), 57
quantile, 58, 58
quantile.fts, 42, 44, 58, 59, 62, 69
residuals.fm, 12, 23, 25, 29, 31, 33, 39, 52–54, 56, 57, 59, 66
sd, 60, 61
sd.fts, 6, 42, 44, 59, 61, 69
sim_ex_cluster, 63
stop_time_detect, 64, 65
stop_time_sim_data, 65
summary.fm, 12, 23, 25, 29, 31, 33, 39, 52–54, 56, 57, 60, 65

T_stationary, 66
VAR, 18
var, 67
var.fts, 6, 42, 44, 59, 62, 68, 68