Package 'fChange'

March 27, 2025

Title Functional Change Point Detection and Analysis

Version 2.0.0

Description Analyze functional data and its change points. Includes functionality to store and process data, summarize and validate assumptions, characterize and perform inference of change points, and provide visualizations. Data is stored as discretely collected observations without requiring the selection of basis functions. For more details see chapter 8 of Horvath and Rice (2024) <doi:10.1007/978-3-031-51609-2>. Additional papers are forthcoming. Focused works are also included in the documentation of corresponding functions.

Encoding UTF-8

Imports dplyr, fastmatrix, fda, ftsa, ggplot2, ggpubr, graphics, grDevices, MASS, methods, plot3D, plotly, rainbow, RColorBrewer, Rcpp, RcppArmadillo, Rfast, sandwich, scales, stats, tensorA, tidyr, vars

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.3

Suggests CompQuadForm, fda.usc, forecast, funData, jmuOutlier, knitr, lattice, rmarkdown, testthat (>= 3.0.0)

Depends R (>= 2.10)

LazyData true

URL `https://jrvanderdoes.github.io/fChange/`

Config/testthat/edition 3

License GPL (>= 3)

NeedsCompilation yes

Author Jeremy VanderDoes [aut, cre, cph] (<`https://orcid.org/0009-0001-9885-3073`>),
Gregory Rice [aut],
Martin Wendler [aut]

Maintainer Jeremy VanderDoes <`jeremy.vanderdoes@gmail.com`>

Repository CRAN

Date/Publication 2025-03-27 18:00:09 UTC

Contents

`acf` *ACF/PACF Functions*

Description

This function computes the ACF/PACF of data. This can be applied on traditional scalar time series or functional time series defined in `dfts()`.

Usage

```
acf(x, lag.max = NULL, ...)

## Default S3 method:
acf(x, lag.max = NULL, ...)

pacf(x, lag.max = NULL, ...)

## Default S3 method:
pacf(x, lag.max = NULL, ...)

## S3 method for class 'dfts'
acf(
  x,
  lag.max = NULL,
  alpha = 0.05,
  method = c("Welch", "MC", "Imhof"),
  WWN = TRUE,
  figure = TRUE,
  ...
)

## S3 method for class 'dfts'
pacf(x, lag.max = NULL, n_pcs = NULL, alpha = 0.95, figure = TRUE, ...)
```

Arguments

`x`	Object for computation of (partial) autocorrelation function (see `acf()` or `pacf`).
`lag.max`	Number of lagged covariance estimators for the time series that will be used to estimate the (partial) autocorrelation function.
`...`	Further arguments passed to the `.plot_FACF` function.
`alpha`	A value between 0 and 1 that indicates significant level for the confidence interval for the i.i.d. bounds of the (partial) autocorrelation function. By default `alpha = 0.05`.
`method`	Character specifying the method to be used when estimating the distribution under the hypothesis of functional white noise. Accepted values are: • "Welch": Welch approximation.

- "MC": Monte-Carlo estimation.
- "Imhof": Estimation using Imhof's method.

By default, `method = "Welch"`.

`WWN`	Logical. If `TRUE`, WWN bounds are also computed
`figure`	Logical. If `TRUE`, plots the estimated function with the specified bounds.
`n_pcs`	Number of principal components that will be used to fit the ARH(p) models.

Value

List with ACF or PACF values and plots

- `acfs/pacfs`: Autocorrelation values for each lag of the functional time series.
- `SWN_bound`: The upper prediction bound for the i.i.d. distribution under strong white noise assumption.
- `WWN_bound`: The upper prediction bound for the i.i.d. distribution under weak white noise assumption.
- `plot`: Plot of autocorrelation values for each lag of the functional time series.

References

Mestre G., Portela J., Rice G., Munoz San Roque A., Alonso E. (2021). *Functional time series model identification and diagnosis by means of auto- and partial autocorrelation analysis.* Computational Statistics & Data Analysis, 155, 107108.

Mestre, G., Portela, J., Munoz San Roque, A., Alonso, E. (2020). *Forecasting hourly supply curves in the Italian Day-Ahead electricity market with a double-seasonal SARMAHX model.* International Journal of Electrical Power & Energy Systems, 121, 106083.

Kokoszka, P., Rice, G., Shang, H.L. (2017). *Inference for the autocovariance of a functional time series under conditional heteroscedasticity* Journal of Multivariate Analysis, 162, 32–50.

See Also

`stats::acf()`

Examples

```
acf(1:10)
x <- generate_brownian_bridge(100, seq(0,1,length.out=20))
acf(x,20)

x <- generate_brownian_bridge(100, seq(0,1,length.out=20))
pacf(x,lag.max = 10, n_pcs = 2)
```

adaptive_bandwidth *Adaptive_bandwidth*

Description

Computes the data-driven bandwidth using a method based on the spectral density operator which adapts to the functional data.

Usage

```
adaptive_bandwidth(
  X,
  kernel = bartlett_kernel,
  name = NULL,
  order = NULL,
  weighting = NULL
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`kernel`	Kernel function. No additional parameters are needed for `bartlett_kernel()`, `parzen_kernel()`, `tukey_hanning_kernel()`, and `quadratic_spectral_kernel()`.
`name`, `order`, `weighting`	Additional parameters if non-standard kernels, e.g. those not in fChange, are used. See references for the definitions. Name is extracted from the kernel name to select `order`/`weighting` when not given, if the function aligns with the recommended functions, see `kernel` parameter.

Value

Scalar value of the data-adapted bandwidth.

References

Rice, G., & Shang, H. L. (2017). A Plug-in Bandwidth Selection Procedure for Long-Run Covariance Estimation with Stationary Functional Time Series. Journal of Time Series Analysis, 38(4), 591–609.

See Also

`bartlett_kernel()`, `truncated_kernel()`, `parzen_kernel()`, `tukey_hanning_kernel()`, `quadratic_spectral_kerne`
`daniell_kernel()`, `flat_top_kernel()`

Examples

```
adaptive_bandwidth(generate_brownian_motion(100))
adaptive_bandwidth(electricity, parzen_kernel)
```

autocorrelation *Estimate the autocorrelation function of the series*

Description

Obtain the empirical autocorrelation function for the given `lags` of a functional time series, `X`. Given a functional time series, the sample autocovariance functions $\hat{C}_h(u, v)$ are given by:

$$\hat{C}_h(u,v) = \frac{1}{N}\sum_{i=1}^{N-|h|}(X_i(u) - \overline{X}_N(u))(X_{i+|h|}(v) - \overline{X}_N(v))$$

where $\overline{X}_N(u) = \frac{1}{N}\sum_{i=1}^{N} X_i(t)$ denotes the sample mean function and h is the lag parameter. The autocorrelation functions are defined over the range $(0, 1)$ by normalizing these functions using the factor $\int \hat{C}_0(u, u)du$.

Usage

```
autocorrelation(X, lags)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`lags`	Numeric(s) for the lags to estimate the lagged operator.

Value

Return a list or data.frame with the lagged autocorrelation function(s) estimated from the data. Each function is given by a $(r$ x $r)$ matrix, where r is the number of points observed in each curve.

See Also

`autocovariance()`

Examples

```
N <- 100
v <- seq(from = 0, to = 1, length.out = 10)
bbridge <- generate_brownian_bridge(N = N, v = v)
lagged_autocor <- autocorrelation(X = bbridge, lags = 0:1)
```

`autocovariance` *Estimate the autocovariance function of the series*

Description

Obtain the empirical autocovariance function for the given `lags` of a functional time series, `X`. Given a functional time series, the sample autocovariance functions $\hat{C}_h(u, v)$ are given by:

$$\hat{C}_h(u, v) = \frac{1}{N} \sum_{i=1}^{N-|h|} (Y_i(u) - \overline{X}_N(u))(Y_{i+|h|}(v) - \overline{X}_N(v))$$

where $\overline{X}_N(u) = \frac{1}{N} \sum_{i=1}^{N} X_i(t)$ denotes the sample mean function and h is the lag parameter.

Usage

```
autocovariance(X, lags = 0:1, center = TRUE)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`lags`	Numeric(s) for the lags to estimate the lagged operator.
`center`	Boolean if the data should be centered. Default is true.

Value

Return a list or data.frame with the lagged autocovariance function(s) estimated from the data. Each function is given by a $(r$ x $r)$ matrix, where r is the number of points observed in each curve.

See Also

`autocorrelation()`, `var()`

Examples

```
v <- seq(0,1,length.out=20)
lagged_autocov <- autocovariance(
  X = generate_brownian_bridge(100,v=v),
  lags = 1)
```

average *Average Functions for dfts Objects*

Description

Compute the pointwise "average" values for dfts objects such as mean and median.

Usage

```
## S3 method for class 'dfts'
mean(x, na.rm = TRUE, ...)

## S3 method for class 'dfts'
median(x, na.rm = TRUE, ...)
```

Arguments

`x`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`na.rm`	Boolean if NA values should be removed. Defaults to TRUE.
`...`	Additional parameters to pass to base R's `min` or `max` functions. They are only used in the `type='fparam'` case.

Value

Numeric vector

Examples

```
results <- mean(electricity)
results <- median(electricity)
```

cancer *Breast Cancer*

Description

Percentage of cause-specific deaths out of total deaths for female breast cancer in the United States from 1950 to 2021.

Usage

```
cancer
```

Format

`cancer`:
A data.frame with columns being the year and rows the age groups (5 years).

`center` *Generic Centering of Data*

Description

Center data by removing the mean or median. Defining changes allow for regional centering.

Usage

```
center(object, changes = NULL, type = "mean", ...)

## Default S3 method:
center(object, changes = NULL, type = "mean", ...)

## S3 method for class 'data.frame'
center(object, changes = NULL, type = "mean", ...)

## S3 method for class 'matrix'
center(object, changes = NULL, type = "mean", ...)

## S3 method for class 'dfts'
center(object, changes = NULL, type = "mean", ...)
```

Arguments

`object`	Object for computation of centering.
`changes`	Change points for centering individual sections.
`type`	String of `mean` or `median` for method of centering.
`...`	Parameters that may be fed into other versions of centering.

Value

Centered data of the same format as the given data.

See Also

`center.default()`, `center.data.frame()`, `center.matrix()`, `center.dfts()`

Examples

```
center(1:10)
```

confidence_interval *Change Point Confidence Intervals*

Description

Compute confidence intervals for the data based on some changes. The current version is tuned to mean changes.

Usage

```
confidence_interval(
  X,
  changes,
  K = bartlett_kernel,
  h = 2 * ncol(X)^(1/5),
  weighting = 0.5,
  M = 5000,
  alpha = 0.1,
  method = "distribution"
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`changes`	Numeric vector for detected change points.
`K`	Function for the Kernel. Default is bartlett_kernel.
`h`	Numeric for bandwidth in computation of long run variance. The default is $2N^{1/5}$.
`weighting`	Weighting for the interval computation, value in [0,1]. Default is 0.5.
`M`	Numeric for the number of Brownian motion simulations in computation of the confidence interval. Default is 1000.
`alpha`	Numeric for the significance level, in [0,1]. Default is 0.1.
`method`	String to indicate the method for computing the percentiles used in the confidence intervals. The options are 'distribution' and 'simulation'. Default is 'distribution'.

Value

Data.frame with the first column for the changes, second for the lower bounds of confidence intervals, and the third for the upper bounds of confidence intervals.

References

Horvath, L., & Rice, G. (2024). Change Point Analysis for Time Series (First edition.). Springer.

Aue, A., Rice, G., & Sonmez, O. (2018). Detecting and dating structural breaks in functional data without dimension reduction. Journal of the Royal Statistical Society. Series B, Statistical Methodology, 80(3), 509-529.

Examples

```
X <- cbind(generate_brownian_motion(100,v=seq(0,1,0.05))$data,
           generate_brownian_motion(100,v=seq(0,1,0.05))$data+1000)
confidence_interval(X,changes = 100)
confidence_interval(X,changes=100,method = 'simulation')

X <- cbind(generate_brownian_motion(100,v=seq(0,1,0.05))$data,
           generate_brownian_motion(100,v=seq(0,1,0.05))$data+0.5)
confidence_interval(X,100,alpha = 0.1)
confidence_interval(X,changes=100,alpha = 0.1,method = 'simulation')

X <- generate_brownian_motion(200,v=seq(0,1,0.05))
confidence_interval(X,100)
confidence_interval(X,100,method = 'simulation')

X <- cbind(generate_brownian_motion(200,v=seq(0,1,0.05))$data,
           generate_brownian_motion(100,v=seq(0,1,0.05))$data+0.1,
           generate_brownian_motion(150,v=seq(0,1,0.05))$data-0.05)
confidence_interval(X,c(200,300))

confidence_interval(X = electricity, changes = c(64, 120),alpha = 0.1)
```

`dfts` *dfts Objects*

Description

The discrete functional time series (dfts) object is the main object in fChange. It stores functional data for use in functions throughout the package. Common functions have been extended to dfts. Details of the storage is best left to individual parameters descriptions and exploring examples.

Usage

```
dfts(X, name = NULL, labels = NULL, fparam = NULL, inc.warnings = TRUE)

as.dfts(X)

is.dfts(X)
```

Arguments

`X`	Data to convert into dfts object. Options include: data.frame, matrix, array, fda::fd, fda.usc::fdata, rainbow::fts (used in ftsa), rainbow::fds (used in ftsa), funData::funData, and dfts. For a matrix, each column is a unique observation, at the rows are the observed intra-observation (i.e. resolution) points.
`name`	String for the name of the object. Defaults to the name of the input variable.
`labels`	Labels for the observations. Defaults to the column names or names inside of the object X.
`fparam`	Vector of numerics indicating the points of evaluation for each observation. Defaults to even spacing on [0,1], or those included in the object. These may be unevenly spaced.
`inc.warnings`	Boolean on if warnings should be given. Defaults to TRUE.

Value

dfts / as.dfts: dfts object

is.dfts: Boolean indicating if `x` is a dfts object or not

Examples

```
bm <- dfts(generate_brownian_motion(100, c(0,0.1,0.25,0.5,1)))

result <- dfts(electricity)
result <- as.dfts(electricity)
result <- is.dfts(electricity)
```

---

dfts_group *Group Generic Functions*

Description

Group generic methods defined for things like Math, Ops, and so forth.

Usage

```
## S3 method for class 'dfts'
Math(x, ...)

## S3 method for class 'dfts'
Ops(e1, e2)

## S3 method for class 'dfts'
cumsum(x)
```

Arguments

`x, e1, e2`	A dfts object. See `dfts()`.
`...`	Further arguments passed to the methods.

Value

A dfts object with the applied operation

dfts object with data as cumsum

Examples

```
result <- sqrt( electricity )
result <- electricity + electricity
result1 <- electricity * electricity
cumsum(electricity)
```

`diff.dfts` *Difference dfts*

Description

Difference the functional data at some lag and iteration. For the ℓ'th difference at lag m, the differenced series is defined as $Y_i(t) = (1 - B^m)^\ell X_i(t)$ where B is the backshift operator.

Usage

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

Arguments

`x`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`lag`	An integer indicating which lag to use.
`differences`	An integer indicating the order of the difference.
`...`	Further arguments to be passed to methods.

Value

A dfts object with the differenced values.

Examples

```
result <- diff(electricity, lag=1)
result1 <- diff(electricity, differences=2)
```

`dim.dfts` *Dimension of dfts Object*

Description

Retrieve the dimension of a dfts object.

Usage

```
## S3 method for class 'dfts'
dim(x, ...)
```

Arguments

`x`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`...`	Additional parameters to pass to base R's `min` or `max` functions. They are only used in the `type='fparam'` case.

Value

Numerics indicating the dimension of the dfts object.

Examples

```
results <- dim(electricity)
```

`electricity` *Spanish Spot Electricity Data*

Description

The hourly electricity spot prices from Spain in 2014.

Usage

```
electricity
```

Format

`electricity`**:**
A dfts object.

Source

<www.omie.es>

`fchange` *Change Point Detection*

Description

Change point detection for dfts objects. Various change point methods are given, where single or multiple changes can be detected. Multiple change extensions currently include binary segmentation and elbow plots.

Usage

```
fchange(
  X,
 method = c("characteristic", "mean", "robustmean", "eigenjoint", "eigensingle",
    "trace", "covariance", "projmean", "projdistribution"),
  statistic = c("Tn", "Mn"),
  critical = c("simulation", "resample", "welch"),
  type = c("single", "segmentation", "elbow"),
  resample_blocks = "separate",
  replace = TRUE,
  max_changes = min(ncol(X), 20),
  changes = NULL,
  blocksize = 2 * ncol(X)^(1/5),
  eigen_number = 3,
  h = 2 * ncol(X)^(1/5),
  M = 1000,
  J = 50,
  W = space_measuring_functions(X = X, M = 20, space = "BM"),
  K = bartlett_kernel,
  alpha = 0.05,
  cov.res = 30,
  weighting = 1/4,
  TVE = 0.95,
  trim_function = function(X) {
     0
 },
  errors = "L2",
  recommendation_change_points = 2,
  recommendation_improvement = 0.15,
  silent.binary = FALSE
)
```

Arguments

X	A dfts object or data which can be automatically converted to that format. See `dfts()`.

`method`	Method to compute change point. Options include: 'characteristic', 'mean', 'robustmean', 'eigenjoint', 'eigensingle', 'trace', 'covariance', 'projmean', and 'projdistribution'.
`statistic`	String for the test statistic type: integrated, `Tn`, or supremum, `Mn`.
`critical`	String for method of computing threshold. Options are 'simulation', 'resample', and 'welch'. Not all ways to compute the critical thresholds are implemented for every method.
`type`	String for the type of change point detection, single change ('single'), binary segmentation ('segmentation'), or elbow plots ('elbow').
`resample_blocks`	String indicating the type of resample test to use. Using `separate` gives blocks which are separate while `overlapping` creates overlapping or sliding windows. When `blocksize=1` then these will be identical.
`replace`	Boolean for using a permutation or bootstrapped statistic when `critical='resample'`.
`max_changes`	Integer as the max number of changes to search when using type is `elbow`.
`changes`	Vector of change points to be given to the eigen test if the data should be centered on these values first.
`blocksize`	Integer for the width of the blocks when using a resampling test. Can use `adaptive_bandwidth()` if additional guidance is desired.
`eigen_number`	Which eigenvalue or the number of eigenvalues which should be checked in the eigenvalue tests.
`h`	Number of lags used when computing long run covariance estimates. Used in mean, characteristic, and eigenvalue tests.
`M`	Number of simulations or permutations for critical values
`J`	Resolution (J) in the characteristic method. The number of vectors is defined by `W`.
`W`	Space measuring functions used in characteristic method to explore the functional space.
`K`	Kernel function for use in characteristic, mean, eigen, covariance and projmean.
`alpha`	Significance in [0,1] for Welch approximation.
`cov.res`	Resolution to use when computing covariance kernel changes.
`weighting`	Weights used in covariance kernel method and pcadistribution.
`TVE`	Total variance explained for projmean and projdistribution.
`trim_function`	Trimming to be used in multiple change methods.
`errors`	Type of errors used in elbow plot. Options are L2 and Trace.
`recommendation_change_points`	Number of lags forward to examine in deciding automated elbow plot recommendation.
`recommendation_improvement`	Significant drop to look for in deciding automated elbow plot recommendation.
`silent.binary`	Boolean if output should be printed when running binary segmentation.

Value

When type is single, returns a list:

1. pvalue: p-value for detection of a change point.
2. location: location of the most likely change.

When type is elbow:

1. information: data.frame with the information on each change and the decrease in variability.
2. plots: list of plots showing the variability decrease or improvement
3. suggestion: list with plot and algorithmic change suggestion. The suggested changes are also returned.

When type is segmentation a data.frame with the locations and p-values is returned.

References

Aue, A., Rice, G., & Sonmez, O. (2018). Detecting and dating structural breaks in functional data without dimension reduction. Journal of the Royal Statistical Society. Series B, Statistical Methodology, 80(3), 509-529.

Wegner, L., Wendler, M. Robust change-point detection for functional time series based on U-statistics and dependent wild bootstrap. Stat Papers (2024).

Aue, A., Rice, G., & Sonmez, O. (2020). Structural break analysis for spectrum and trace of covariance operators. Environmetrics (London, Ont.), 31(1)

Horvath, L., Rice, G., & Zhao, Y. (2022). Change point analysis of covariance functions: A weighted cumulative sum approach. Journal of Multivariate Analysis, 189, 104877-.

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

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

Huskova, M., & Meintanis, S.G. (2006). Change Point Analysis based on Empirical Characteristic Functions. Metrika, 63, 145-168.

Examples

```
res <- fchange(electricity$data[,1:20],method='characteristic',critical = 'welch')
```

generate_brownian_bridge

Generate a Brownian Bridge Process

Description

Generate a functional time series from an iid Brownian Bridge process. If $W(t)$ is a Wiener process, the Brownian Bridge is defined as $W(t) - tW(1)$. Each functional observation is discretized on the points indicated in v.

Usage

```
generate_brownian_bridge(N, v = 30, sd = 1)
```

Arguments

N	Numeric. The number of observations for the generated data.
v	Numeric (vector). Discretization points of the curves.This can be the evaluated points or the number of evenly spaced points on [0,1]. By default it is evenly spaced on [0,1] with 30 points.
sd	Numeric. Standard deviation of the Brownian Motion process. The default is 1.

Value

Functional time series (dfts) object.

Examples

```
bbridge <- generate_brownian_bridge(N=100, v=c(0,0.2,0.6,1,1.3), sd=2)
bbridge <- generate_brownian_bridge(N=100, v=10, sd=1)
```

generate_brownian_motion

Generate a Brownian Motion Process

Description

Generate a functional time series according to an iid Brownian Motion process. Each observation is discretized on the points indicated in v.

Usage

```
generate_brownian_motion(N, v = 30, sd = 1)
```

Arguments

`N`	Numeric. The number of observations for the generated data.
`v`	Numeric (vector). Discretization points of the curves.This can be the evaluated points or the number of evenly spaced points on [0,1]. By default it is evenly spaced on [0,1] with 30 points.
`sd`	Numeric. Standard deviation of the Brownian Motion process. The default is `1`.

Value

Functional time series (dfts) object.

Examples

```
bmotion <- generate_brownian_motion(N=100,
  v=c(0,0.25,0.4,0.7, 1, 1.5), sd = 1)
bmotion1 <- generate_brownian_motion(N=100,
  v=10, sd = 2)
```

`generate_data` *Generate Functional Data*

Description

A general wrapper function to allow generation of functional data according to several approaches: `bbridge`, `bmotion`, `kl`, `ou`, and `far1`.

Usage

```
generate_data(fparam, data_details, burnin = 100)
```

Arguments

`fparam`	fparam of data (or resolution that will be equally spaced on [0,1]).
`data_details`	List of named lists indicating parameters for each data group. Each process can use different parameters, given below.

- **bmotion**: Brownian motion contains:
  - **N**: Numeric indicating the number of observations.
  - **sd**: Numeric for the standard deviation of the observations.
- **bbridge**: Brownian bridge contains:
  - **N**: Numeric indicating the number of observations.
  - **sd**: Numeric for the standard deviation of the observations.
- **kl**: Karhunen-Loeve expansion contains:
  - **N**: Numeric indicating the number of observations.

  - **distribution**: Distribution of the errors. Options include: `binomial`, `exponential`, `laplace`, `normal`, `t` (add dof argument), `gamma` (add shape argument), and `cauchy`.
  - **eigenvalues**: Numerics for the eigenvalues of the given distribution (value for each in basis).
  - **mean**: Numeric for the mean of the group.
  - **dependence**: Strength of dependence between observation.
  - **basis**: fda `basisfd` object.
  - **dof**: (Optional) Numeric for the degrees of freedom if using a `t` distribution.
  - **shape**: (Optional) Numeric for the shape if using a `gamma` distribution.
- **ou**: Ornstein-Uhlenbeck process requires:
  - **N**: Numeric indicating the number of observations.
  - **dependence**: Strength of dependence between observation.
- **far1**: Functional autoregressive process of order 1 contains:
  - **N**: Numeric indicating the number of observations.
  - **dependence**: Strength of dependence between observation.
  - **sd**: Numeric for the standard deviation of the observations.
  - **vary**: Boolean if the starting value each observation should be 0 or vary. It does this by dropping first value. If fparam is given as a number, it can adjust so that the length is the same. If fparam is a vector, the fparam will be one smaller.

`burnin` Numeric for amount of burnin for data. Only used for the first groups. Subsequent groups begin at the end of the last group.

Value

A dfts object for the generated data.

See Also

`generate_brownian_bridge()`, `generate_brownian_motion()`, `generate_far1()`, `generate_karhunen_loeve()`, `generate_ornstein_uhlenbeck()`

Examples

```
result <- generate_data(
  fparam=15,
  data_details =list('bmotion'=list('N'=100, 'sd'=1),
                     'bbridge'=list('N'=100, 'sd'=1),
                     'bbridge'=list('N'=100, 'sd'=1),
                     'kl'=list('N'=100,
                               'distribution'='Normal',
                               'eigenvalues'=1/1:4,
                               'mean'=0, 'dependence'=0,
                               'basis'=fda::create.bspline.basis(),
                               'sd'=1),
                     'ou'=list('N'=100, 'dependence'=0 ) ,
                     'far1'=list('N'=100, 'dependence'=0,
                                 'sd'=1,'vary'=FALSE) )
)
```

generate_far1 *Generate FAR(1) Data*

Description

Function to generate data according to FAR(1) process.

Usage

```
generate_far1(N, resolution, sd = 1, dependence = 1/2, drop_first = FALSE)
```

Arguments

N	Numeric for the number of observations.
resolution	Numeric for resolution of data or a vector specifying the observation points.
sd	Numeric for standard deviation with Brownian motion.
dependence	Numeric which indicates the dependence on the previous curve.
drop_first	Booolean if first values should be dropped so the data varies at the first rather than starting at 0 (given that is the observed first point). Note this will affect the resolution observed.

Value

dfts object of the data.

Examples

```
res <- generate_far1(20,10)
```

generate_karhunen_loeve *Generate functional data*

Description

generate_karhunen_loeve generates functional data via an autoregressive Karhunen-Loeve expansion. The approach easily accomondate change points in the mean, distribution, eigenvalues, eigenfunctions, and so forth. In a sense, the function creates m ‘groups’ of discretely observed functions with similar properties.

Usage

```
generate_karhunen_loeve(
  Ns,
  eigenvalues,
  basis,
  means,
  distribution,
  fparam,
  dependence = 0,
  burnin = 100,
  silent = TRUE,
  dof = NULL,
  shape = NULL,
  prev_eps = NULL
)
```

Arguments

`Ns`	Vector of Numerics. Each value in Ns is the number of observations for a given group, for m groups.
`eigenvalues`	Vector of eigenvalues, length 1 or m.
`basis`	A list of bases (eigenfunctions), length m.
`means`	A vector of means, length 1 or Ns.
`distribution`	A vector of distributions, length 1 or m.
`fparam`	A vector of points indicating the points to evaluate the functions on.
`dependence`	Numeric [0,1] indicating strength of VAR(1) process.
`burnin`	A numeric value indicating the number of burnin trials.
`silent`	A Boolean that toggles displaying the running status.
`dof`	Numeric for shape with gamma distribution (rate is set to 1).
`shape`	Numeric for degrees of freedom with t-distribution.
`prev_eps`	Previous epsilon for dependence across groups. This is only needed if a separate code was run but the new data should be appended. In general only used in internal functions.

Value

List with (1) dfts data and (2) the errors of the last iteration.

Examples

```
dat1 <- generate_karhunen_loeve(
  Ns=100, eigenvalues=c(1/(1:3)), basis=fda::create.bspline.basis(nbasis=3,norder=3),
  means=0, distribution='Normal',
  fparam=seq(0,1,0.1), dependence=0, burnin=100, silent=TRUE, dof=NULL, shape=NULL,
  prev_eps=NULL)
dat2 <- generate_karhunen_loeve(
  Ns=50, eigenvalues=c(1/(1:4)), basis=fda::create.bspline.basis(nbasis=4),
  means=5, distribution='exponential',
  fparam=seq(0,1,0.1), dependence=0, burnin=100, silent=TRUE, dof=NULL, shape=NULL,
  prev_eps=dat1$prev_eps)

dat <- dfts(cbind(dat1$data$data, dat2$data$data),fparam = dat1$data$fparam)
```

generate_ornstein_uhlenbeck *Generate Data via Ornstein-Uhlenbeck Process*

Description

Generate autoregressive process with errors according the Ornstein-Uhlenbeck process.

Usage

```
generate_ornstein_uhlenbeck(N, v, rho = 0)
```

Arguments

N	Numeric for the number of observations.
v	Numeric for resolution of data or a vector specifying the observation points.
rho	Numeric which indicates the dependence on the previous curve.

Value

A dfts object for the generated data.

Examples

```
generate_ornstein_uhlenbeck(N=100,v=20)
```

impute *Functional Imputatio*

Description

Several basic imputation methods for missing values in functional data formatted as dfts objects.

Usage

```
impute(
  X,
  method = c("zero", "mean_obs", "median_obs", "mean_data", "median_data", "linear",
    "functional"),
  obs_share_data = FALSE
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`method`	String to indicate method of imputation. • zero: Fill missing values with 0. • mean_obs: Fill missing values with the mean of each observation. • median_obs: Fill missing values with the median of each observation. • mean_data: Fill missing values with the mean of the data at that particular fparam value. • median_data: Fill missing values with the median of the data at that particular fparam value. • linear: Fill missing values with linear interpolation. • functional: Fill missing values with functional interpolation. This is done by fitting the data to basis with the package 'fda'.
`obs_share_data`	Boolean in linear interpolation that indicates if data should be shared across observations. For example, if the end of observation i related to the start of observation i+1. Default is FALSE, which suggests independence. If true, the distance between the end and start of observations is taken to be the mean average distance of points in fparam.

Value

A dfts object of the data with missing values interpolated.

Examples

```
temp <- data.frame(c(NA,NA,3:9,NA),
                   c(NA,stats::rnorm(2),NA, stats::rnorm(6)),
                   stats::rnorm(10),
                   c(stats::rnorm(4),rep(NA,3), stats::rnorm(3)),
                   rep(NA,10),
                   c(stats::rnorm(1), rep(NA,9)),
                   c(stats::rnorm(9),NA),
                   stats::rnorm(10),
                   stats::rnorm(10),
                   c(NA,NA,3:9, NA))
impute(temp, method='mean_obs')
impute(temp, method='linear', obs_share_data=TRUE)
```

`kernels` *Kernel Functions*

Description

There are an assortment of (vectorized) kernel functions located in the package.

Truncated Kernel: Kernel where $1, |x| \leq 1$ and 0 otherwise. If $x = 0/0$ then the value 1 is given.

Bartlett Kernel: Kernel where $max(0, 1 - |x|), h \neq 0$. If $x = 0/0$ then the value 1 is given.

Parzen Kernel: Kernel where $1 - 6 * x^2 + 6 * |x|^3, |x| <= 0.5$, $2 * (1 - |x|)^3, 0.5 < |x| < 1$, and $0, |x| > 1$. If $x = 0/0$ then the value 1 is given.

Tukey-Hanning Kernel: Kernel where $(1 + cos(\pi x))/2, |x| <= 1$ and $0, |x| > 1$. If $x = 0/0$ then the value 1 is given.

Quadratic Spectral Kernel: Kernel where $\frac{25}{12\pi^2 x^2}\left(\frac{sin(6\pi x/5)}{6\pi x/5} - cos(6\pi x/5)\right)$. If $x = 0/0$ then the value 1 is given.

Daniell Kernel: Kernel where $sin(pi * x)/(pi * x) * (1 + cos(pi * x)), abs(x) <= 1$. If $x = 0/0$ then the value 1 is given.

Flat-Top Kernel: Kernel where $min(1, max(1.1 - |x|, 0)), |x| \leq 1$. If $x = 0/0$ then the value 1 is given.

Usage

```
truncated_kernel(x)

bartlett_kernel(x)

parzen_kernel(x)

tukey_hanning_kernel(x)

quadratic_spectral_kernel(x)

daniell_kernel(x)

flat_top_kernel(x)
```

Arguments

x	Numeric value(s) at which to evaluate kernel. It often indicates current lag divided by window.

Value

Values from given lag(s) in the kernel.

References

Horvath, L., & Rice, G. (2024). Change point analysis for time series (1st ed. 2024.). Springer Nature Switzerland.

L. Horvath, P. Kokoszka, G. Rice (2014) "Testing stationarity of functional time series", Journal of Econometrics, 179(1), 66-82.

Politis, D. N. (2003). Adaptive bandwidth choice. Journal of Nonparametric Statistics, 15(4-5), 517-533.

Politis, D. N. (2011). Higher-order accurate, positive semidefinite estimation of large-sample covariance and spectral density matrices. Econometric Theory, 27(4), 703-744.

Examples

```
truncated_kernel(-20:20/15)
bartlett_kernel(-20:20/15)
parzen_kernel(-20:20/15)
tukey_hanning_kernel(-20:20/15)
quadratic_spectral_kernel(-20:20/15)
daniell_kernel(-20:20/15)
flat_top_kernel(-20:20/15)
```

`kpss_test` *Functional KPSS Test*

Description

Compute the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) statistic for functional data.

Usage

```
kpss_test(
  X,
  method = c("simulation", "resample"),
  resample_blocks = "separate",
  M = 1000,
  blocksize = 2 * ncol(X)^(1/5),
  TVE = 1,
  replace = TRUE
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`method`	String for the method in computing thresholds: Monte Carlo simulation (`simulation`) or resampling (`resample`).
`resample_blocks`	String indicating the type of resample test to use. Using `separate` gives blocks which are separate while `overlapping` creates overlapping or sliding windows. When `blocksize=1` then these will be identical.
`M`	Number of simulations to estimate theoretical distribution.
`blocksize`	Numeric for the block size when using a resample test.

`TVE`	Numeric for `pca()` to select the number of principle components.
`replace`	Boolean to indicate if blocks should be selected with replacement when using a resample test.

Value

List with the following elements:

1. pvalue: p-value from the test.
2. statistic: test statistic computed on the data.
3. simulations: Theoretical values for the null distribution.

References

Chen, Y., & Pun, C. S. (2019). A bootstrap-based KPSS test for functional time series. Journal of Multivariate Analysis, 174, 104535.

Kokoszka, P., & Young, G. (2016). KPSS test for functional time series. Statistics, 50(5), 957-973.

Examples

```
kpss_test(generate_brownian_motion(100,v=seq(0,1,length.out=20)))
kpss_test(generate_brownian_motion(100,v=seq(0,1,length.out=20)),
             method="resample")
kpss_test(generate_brownian_motion(100,v=seq(0,1,length.out=20)),
             method='resample', resample_blocks='overlapping')
```

`lag.dfts` *Lag dfts objects*

Description

Compute a lagged version of a functional time series, shifting the time base back by a given number of observations.

Usage

```
## S3 method for class 'dfts'
lag(x, k = 1, ...)
```

Arguments

`x`	A dfts object. See `dfts()`.
`k`	Integer for the number of lags (in units of observations).
`...`	Unused additional parameters.

Value

A dfts object.

See Also

`stats::lag()`, `diff.dfts()`

Examples

```
result <- lag(electricity)
```

`long_run_covariance` *Estimate Long-run Covariance Kernel*

Description

Estimate the long-run covariance kernel for functional data. That is, solve $C_\epsilon(t,t') = \sum_{l=-\inf}^{\inf} \text{Cov}(\epsilon_0(t), \epsilon_l(t'))$ with sequence $(\epsilon_i : i \in \mathbb{Z})$ defined as the centered data (can center based on changes if given).

Usage

```
long_run_covariance(
  X,
  h = 2 * ncol(X)^(1/5),
  K = bartlett_kernel,
  changes = NULL
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`h`	The window parameter parameter for the estimation of the long run covariance kernel. The default value is `h=2*ncol(X)^(1/5)`. Note there exists an internal check such that $h = min(h, ncol(X) - 1)$ when alternative options are given.
`K`	Function indicating the kernel to use if $h > 0$.
`changes`	Vector of numeric change point locations. Can be NULL.

Value

Symmetric data.frame of numerics with dim of ncol(data) x ncol(data).

Examples

```
result <- long_run_covariance(electricity,2)
```

minmax — *Max / Min for dfts Objects*

Description

Get the observation(s) or pointwise values with the min / max values. When using `type='Obs'`, the selected observation is the one with the minimum or maximum mean. When using `type='fparam'`, the values are given pointwise.

Usage

```
## S3 method for class 'dfts'
max(x, type = c("Obs", "fparam"), na.rm = TRUE, ...)

## S3 method for class 'dfts'
min(x, type = c("Obs", "fparam"), na.rm = TRUE, ...)
```

Arguments

`x`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`type`	String indicating if finding for observation ('Obs'), or for pointwise values ('fparam').
`na.rm`	Boolean if NA values should be removed. Defaults to TRUE.
`...`	Additional parameters to pass to base R's `min` or `max` functions. They are only used in the `type='fparam'` case.

Value

A dfts object.

Examples

```
results <- max(electricity)
results <- min(electricity)
```

pca — *Generic Function for Principal Component Analysis*

Description

This is a generic function to call PCA on various objects. The default method uses `stats::prcomp()`.

Usage

```
pca(object, TVE = 1, ...)

## Default S3 method:
pca(object, ...)

## S3 method for class 'dfts'
pca(object, TVE = 1, ...)
```

Arguments

`object`	Object for computation of principle components analysis.
`TVE`	Numeric in [0,1] for the total variance explained, this determines the number of components and can be used for dimension reduction.
`...`	Additional parameters to extensions based on data. Often this is additional information for `prcomp`.

Value

Principal component data.

Examples

```
pca(1:10)
pca(electricity)
```

`pca_examination` *Principal Component Exploration*

Description

Computes the principal component projection and re-combined data with a myriad of figures to understand the projection process.

Usage

```
pca_examination(X, TVE = 0.95)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`TVE`	Numeric in [0,1] for the total variance explained, this determines the number of components and can be used for dimension reduction.

Value

List with the following elements:

- figures: List with figures on all components, summaries, reconstructed values, and residuals.
- reconstruction: Reconstructed value from PCs.
- residuals: Difference between true and reconstruction.

See Also

`pca.dfts()`

Examples

```
results <- pca_examination(electricity, TVE=0.9)
```

---

plot.dfts *Plot dfts objects*

Description

Provides various visualizations for functional data.

Usage

```
## S3 method for class 'dfts'
plot(
  x,
  changes = NULL,
 type = c("spaghetti", "fast", "rainbow", "banded", "acf", "pacf", "summary", "qq",
    "distribution", "change", "interval", "surface"),
  plot_title = x$name,
  val_axis_title = NULL,
  res_axis_title = NULL,
  FD_axis_title = NULL,
  eye = NULL,
  aspectratio = NULL,
  showticklabels = TRUE,
  lag.max = 20,
  d.max = 2,
  alpha = 0.05,
  TVE = 0.95,
  distribution = c("norm"),
  method = c("Welch", "MC", "Imhof"),
  legend = TRUE,
  highlight_changes = TRUE,
  intervals = confidence_interval(x, changes),
  int.gradual = TRUE,
  ...
)
```

Arguments

`x`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`changes`	Vector of numeric change point locations. Can be NULL.
`type`	Choice of plotting method. Options include: 'spaghetti', 'fast', 'rainbow','banded','acf', 'pacf', 'summary', 'qq', 'distribution', 'change', 'interval', and'surface'.
`plot_title`	Title to include on the return plot.
`val_axis_title, res_axis_title, FD_axis_title`	Title for the axis giving the values (val), the resolution of the fparam (res), and the functional observations (FD).
`eye, aspectratio`	Angle (eye) and ratio (aspectratio) to view 3d plots.
`showticklabels`	Boolean if the tick marks should be shown.
`lag.max`	Max number of lags to consider for ACF/PACF and summary plots.
`d.max`	Max number of dimensions for qq/distribution and summary plots.
`alpha`	Significance level to be used in various plots. Value in [0,1].
`TVE`	Total variance explained used in qq/distribution plots. Value in [0,1].
`distribution`	String of the distribution to compare against in. distribution plot. The string can be anything such that there is a rdistribution and ddistribution function available. For example "exp", "gamma". Additional parameters can be passed using ... .
`method`	Method for computing ACF/PACF. Options include 'Welch', 'MC', and 'Imhof'.
`legend`	Boolean if legend should be given in qq/distribution plots.
`highlight_changes`	Boolean if changes should be highlighted in black.
`intervals`	Information on confidence intervals of changes for change plot. See `confidence_interval()`.
`int.gradual`	Boolean if confidence interval be solid gray (FALSE) or gradual colors (TRUE) when `type='change'` plot.
`...`	Details for plotting in acf/pacf, summary, or distribution function.

Value

Plot of varying types.

References

John Fox, & Sanford Weisberg (2019). An R Companion to Applied Regression. Sage.

Examples

```
plt <- plot(electricity)
plt <- plot(dfts(var(electricity)), type='surface')
```

`portmanteau_tests` *Functional Hypothesis Tests for Functional Data*

Description

Computes a variety of portmanteau hypothesis tests for functional data in the form of dfts objects.

Usage

```
portmanteau_tests(
  X,
  test = c("variety", "single", "multi", "spectral", "independence", "imhof"),
  lag = 5,
  M = 1000,
  method = c("iid", "bootstrap"),
  kernel = bartlett_kernel,
  block_size = NULL,
  bandwidth = NULL,
  components = 3,
  resample_blocks = "separate",
  replace = FALSE,
  alpha = 0.05
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`test`	A String specifying the hypothesis test. Currently available tests are: 'variety', 'single-lag', 'multi-lag', 'spectral', 'independence', and 'imhof'.
`lag`	A positive integer to specify the lag, or maximum lag, of interest. Only used for the "single-lag", "multi-lag", "independence", and "imhof" tests.
`M`	Numeric to specify the number of Monte-Carlo or resampled simulations to use for the limiting distributions.
`method`	String indicating the method for the `single` test, options include: • **iid**: The hypothesis test will use a strong-white noise assumption (instead of a weak-white noise assumption). • **resample**: The hypothesis test is evaluated by approximating the limiting distribution of the test statistic via a (block) resampling process. Additional methods are forthcoming.
`kernel`	Kernel function for spectral test or estimation of covariance.
`block_size`	Numeric to specify block size for resampling tests.
`bandwidth`	Numeric for bandwidth of covariance estimation. If left null, with be defined by $N^{1/(2*\mathrm{KO}+1)}$ where KO is the order of the selected kernel.

`components`	Number of functional principal components to use in the independence test.
`resample_blocks`	String indicating the type of resample test to use. Using `separate` gives blocks which are separate while `overlapping` creates overlapping or sliding windows. When `blocksize=1` then these will be identical.
`replace`	Boolean for using a permutation or bootstrapped statistic when `method='resample'`.
`alpha`	Numeric value for significance in [0,1].

Details

The "single"-lag portmanteau test assesses the significance of empirical lagged autocovariance operators at a single lag `lag`. It tests the null hypothesis that the lag-h autocovariance operator is equal to 0. The test is designed for stationary functional time-series, and is valid under conditional heteroscedasticity conditions.

The "multi"-lag portmanteau test assesses the cumulative significance of empirical lagged autocovariance operators, up to a user-selected maximum lag `lag`. It tests the null hypothesis that the first lag-h autocovariance operators, $h = 1, \ldots, lag$, is equal to 0. The test is designed for stationary functional time-series, and is valid under conditional heteroscedasticity conditions.

The "spectral" portmanteau test measures the proximity of a functional time series to a white noise. Comparison is made to the constant spectral density operator of an uncorrelated series. The test is not for general white noise series, and may not hold under functional conditionally heteroscedastic assumptions.

The "independence" portmanteau test measures independence and identical distribution based lagged cross-variances from dimension reduction using functional principal components analysis. The test is not for general white noise series, and may not hold under functional conditionally heteroscedastic assumptions.

The "imhof" portmanteau test is an analogue of the "single-lag" test. While the "single-lag" test computes the limiting distribution of the test statistic via a Welch-Satterthwaite approximation, the "imhof" test directly computes the coefficients of the quadratic form in normal variables. Hence, the test is computationally expensive.

Value

List with results dependent on the test. In general, returns the pvalue, statistic, and simulations/quantile.

References

Kim, M., Kokoszka P., & Rice G. (2023) White noise testing for functional time series. Statist. Surv., 17, 119-168, DOI: 10.1214/23-SS143

Characiejus, V., & Rice, G. (2020). A general white noise test based on kernel lag-window estimates of the spectral density operator. Econometrics and Statistics, 13, 175–196.

Kokoszka P., & Rice G., & Shang H.L. (2017). Inference for the autocovariance of a functional time series under conditional heteroscedasticity. Journal of Multivariate Analysis, 162, 32-50.

Zhang X. (2016). White noise testing and model diagnostic checking for functional time series. Journal of Econometrics, 194, 76-95.

Gabrys R., & Kokoszka P. (2007). Portmanteau Test of Independence for Functional Observations. Journal of the American Statistical Association, 102:480, 1338-1348, DOI: 10.1198/016214507000001111.

Chen W.W. & Deo R.S. (2004). Power transformations to induce normality and their applications. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 66, 117-130.

Examples

```
b <- generate_brownian_motion(250)
res0 <- portmanteau_tests(b, test = 'single', lag = 10)
res1 <- portmanteau_tests(b, test = 'multi', lag = 10, alpha = 0.01)
res2 <- portmanteau_tests(b, test = 'spectral', kernel = bartlett_kernel,
                          bandwidth = NULL, alpha = 0.05)
res3 <- portmanteau_tests(b, test = 'spectral', alpha = 0.1,
                          kernel = parzen_kernel,
                          bandwidth = adaptive_bandwidth(b, kernel=parzen_kernel))
res4 <- portmanteau_tests(b, test = 'independence', components = 3, lag = 3)
res5 <- portmanteau_tests(b, test = 'single', lag = 1, M = 250)
```

`print.dfts` *Print dfts objects*

Description

Basic formatting to print a dfts object.

Usage

```
## S3 method for class 'dfts'
print(x, ...)
```

Arguments

`x`	A dfts object. See `dfts()`.
`...`	unused parameter.

Value

No return value, called to format printing of dfts object to console.

Examples

```
electricity
```

`projection_model` *Projection-based functional data model*

Description

Model and forecast functional data using a Hyndman and Ullah projection-based model.

Usage

```
projection_model(
  X,
  TVE = 0.95,
  model = c("ets", "arima"),
  n.ahead = 0,
  alpha = 0.05,
  check.cp = TRUE,
  frequency = 1,
  ...
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`TVE`	Numeric in [0,1] for the total variance explained to select number of PCA components to use to model the data.
`model`	String to indicate method to model components, either "ets" or "arima".
`n.ahead`	Number of observations to forecast.
`alpha`	Significance in [0,1] for intervals when forecasting.
`check.cp`	Boolean which indicates if the errors should be checked for change points to change forecasts and plots.
`frequency`	Numeric for seasonal frequency when component is made a ts object for the models.
`...`	Additional information to pass into pca, change (if `check.cp=TRUE`), and plot.

Value

List with the following elements:

- fit: dfts object for fit.
- forecast_plot: plot of the data with any forecasted values.
- residuals: dfts object for residuals from the fit.
- changes: vector of any changes when using `detect.cp`.
- component_models: modeled PCs from the data.
- component_true: true data constucted via the PCs.
- parameters: list with fit parameters like pcs, TVE, model, and n.ahead.

References

Hyndman, R. J., & Shahid Ullah, M. (2007). Robust forecasting of mortality and fertility rates: A functional data approach. Computational Statistics & Data Analysis, 51(10), 4942-4956. https://doi.org/10.1016/j.csda.2006.

Examples

```
result <- projection_model(dfts(electricity$data[,50:100]),
 n.ahead=1, TVE=0.1, check.cp=FALSE)
```

---

qqplot *QQ Plot Generic Function*

Description

A generic function which by produces a qq-plot of some data. By default, it uses `stats::qqplot()`.

qqplot.dfts: Creates normal QQ plots on the principal components of functional data.

Usage

```
qqplot(x, ...)

## Default S3 method:
qqplot(x, ...)

## S3 method for class 'dfts'
qqplot(
  x,
  TVE = 0.95,
  d.max = NULL,
  alpha = 0.05,
  changes = NULL,
  legend = FALSE,
  ...
)
```

Arguments

`x`	A dfts object. See `dfts()`.
`...`	Additional parameters based on the data.
`TVE`	Numeric in [0,1] giving the total variance explained for selecting the number of principal components.
`d.max`	Max number of principal components. No max when NULL.
`alpha`	Significance level, alpha in [0,1].
`changes`	Vector of change points.
`legend`	Boolean indicating if legend should be shown on plot.

Value

qqplot.default: returns results from `stats::qqplot()`.

qqplot.dfts: ggplot2 for QQ plot.

See Also

`stats::qqplot()`

Examples

```
result <- qqplot(electricity, d.max=3)
```

`quantile.dfts` *Quantile dfts*

Description

Obtain the pointwise quantile information of functional data.

Usage

```
## S3 method for class 'dfts'
quantile(x, probs = seq(0, 1, 0.25), ...)
```

Arguments

`x`	A dfts object. See `dfts()`.
`probs`	Numerics in [0,1] indicating the probabilities of interest.
`...`	Additional parameters to pass into `stats::quantile()` function.

Value

Matrix with columns for each requested quantile computed pointwise.

See Also

`stats::quantile()`

Examples

```
result <- quantile(electricity)
result1 <- quantile(electricity,probs = 0.95)
```

`rates` *US Yield Curves*

Description

Yield curves in the US for 1 - 360 month maturity from 1990 to 2023 (removing days without information, i.e. weekends and holidays).

Usage

```
rates
```

Format

`rates`:
A dfts object.

`sdvar` *Generic Function for Variance and Standard Deviation Computation*

Description

Generic function to compute the variance and standard deviations. The default uses `stats::var()` and `stats::sd()`.

Usage

```
sd(object, ...)

## Default S3 method:
sd(object, ...)

## S3 method for class 'dfts'
sd(object, type = "pointwise", ...)

var(object, ...)

## Default S3 method:
var(object, ...)

## S3 method for class 'dfts'
var(object, type = c("operator", "pointwise"), ...)
```

Arguments

`object`	Object for computation of standard deviation or variance of the given data set.
`...`	Additional parameters for the particular extensions.
`type`	String to specify if an operator ('op') or pointwise ('pw') calculation is desired on the functional data.

Value

Numeric(s) to explain the standard deviation / variance

See Also

`stats::sd()`, `stats::var()`

Examples

```
sd(1:10)
var(1:10)
sd(electricity,type='pointwise')
var(electricity,type='pointwise')
var(electricity,type='operator')
```

---

space_measuring_functions
Compute Spacing Measuring Functions

Description

This function is used to compute discretized functions, i.e. vectors, to explore functional spaces.

Usage

```
space_measuring_functions(X, M = 20, space = "BM")
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`M`	Integer for the number of functions to generate.
`space`	String for the space of interest. Options are Brownian motion ('BM'), principal components ('PC'), and vectors in iid standard, random normals ('RN'). Additional options are forthcoming

Value

Data.frame with columns of discretized functions describing the space. Columns are independent functions.

See Also

`fchange()`

Examples

```
space_measuring_functions(M=10, space="BM", X=electricity)
space_measuring_functions(M=10, space="PC", X=electricity)
```

`SPYUS500` *S&P 500 Index Data*

Description

Intraday prices for the S&P500 index (SPY) for 2019 to 2023 with holidays and weekends removed. Minutely resolution and daily observations.

Usage

```
SPYUS500
```

Format

`SPY500`**:**
A dfts object.

`stationarity_test` *Functional Stationarity Test*

Description

Stationarity test for functional time series with different methods on determining the critical values of the test statistic. The Monte Carlo method was constructed in Horvath et al. (2014), while the resample-based methods have not been validated in the literature (use the provided option at your discretion).

Usage

```
stationarity_test(
  X,
  statistic = "Tn",
  critical = c("simulation", "resample"),
  perm_method = "separate",
  M = 1000,
  blocksize = 2 * ncol(X)^(1/5),
  TVE = 1,
  replace = TRUE
)
```

Arguments

`X`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`statistic`	String for test statistic. Options are integrated (`Tn`) and supremum (`Mn`). The default is `Tn`.
`critical`	String for method of determining the critical values. Options are `simulation` and `resample`. Default is `simulation`.
`perm_method`	String for method of resampling. Options are `separate` for block resampling and `overlapping` for sliding window. Default is `separate`.
`M`	Numeric for number of simulation to use in determining the null distribution. Default is 1000.
`blocksize`	Numeric for blocksize in resample test. Default is $2N^{1/5}$.
`TVE`	Numeric for total variance explained when using PCA for eigenvalues. Default is 1.
`replace`	Boolean if replacement should be used for resample test. Thus, this defines if a bootstrapped or permuted test is used. Default is TRUE.

Value

List with the following elements:

1. pvalue: p-value for the stationarity test.
2. statistic: test statistic from the test.
3. simulations: simulations which define the null distribution.

References

Horvath, L., Kokoszka, P., & Rice, G. (2014). Testing stationarity of functional time series. Journal of Econometrics, 179(1), 66-82.

Examples

```
res <- stationarity_test(
  generate_brownian_motion(100,v=seq(0,1,length.out=20)),
  critical='resample', statistic='Mn')
res2 <- stationarity_test(electricity)
```

stat_3D *Draw 3D Geoms for ggplot2*

Description

This function adds 3D geoms such as points and paths to a ggplot2 plot.

Usage

```
stat_3D(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
`data`	Default dataset to use for plot. If not already a data.frame, will be converted to one by `fortify()`. If not specified, must be supplied in each layer added to the plot.
`geom`	The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following: • A `Geom` ggproto subclass, for example `GeomPoint`. • A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`. • For more information and other ways to specify the geom, see the layer geom documentation.
`position`	Set position information. Find more details in `ggplot2` function.
`na.rm`	Boolean if na data should be removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`...`	Arguments passed on to layer. Often the aesthetics like `color = "red"` or `size = 3`. Two important ones are `theta` (azimuthal rotation) and `phi` (colatitude rotation) to define angle in degrees of viewing data.

Value

No direct return value, called to be used with `ggplot2::ggplot()` in designing the plot.

References

Acker D (2024). gg3D: 3D perspective plots for ggplot2. R package version 0.0.0.9000.

Examples

```
dat <- electricity

data_lines <- cbind(data.frame('Time'=dat$fparam), dat$data) %>%
tidyr::pivot_longer(cols = 1+1:ncol(dat$data))

colors_plot <- RColorBrewer::brewer.pal(11, "Spectral")
colors_plot <- grDevices::colorRampPalette(colors_plot)(ncol(dat$data))
data_lines$color <- rep(colors_plot, nrow(dat$data) )
data_lines$name <- as.numeric(data_lines$name)

result <- ggplot2::ggplot(data_lines,
  ggplot2::aes(y=Time, x=name, z=value, color=color)) +
  ggplot2::theme_void() +
  stat_3D(theta=0, phi=15, geom='path') +
  ggplot2::scale_color_manual(
     breaks = data_lines$color,
     values = data_lines$color
  ) +
  ggplot2::guides(color='none')
```

`summary.dfts` *Summary for dfts Object*

Description

General summary function to view data overview. Several plots and test statistics are returned to give a general view of the data. More details can be found with more specialized functions.

Usage

```
## S3 method for class 'dfts'
summary(object, changes = NULL, lag.max = 20, d.max = 2, demean = FALSE, ...)
```

Arguments

`object`	A dfts object or data which can be automatically converted to that format. See `dfts()`.
`changes`	Vector of change locations, if there are any. Default is NULL.
`lag.max`	Max lags to consider for ACF. Default is 20.
`d.max`	Max number of dimensions for QQ-plot.
`demean`	Boolean if data should be demeaned based on changes and create plots based on these residuals.
`...`	Data to pass into underlying functions like the KPSS, portmanteau, and stationary tests. In general it is recommended to not use this and instead apply the specialized functions directly.

Value

List with the elements:

1. summary_data: summary results for the data.
2. summary_plot: summary plot for the data.

See Also

`base::summary()`

Examples

```
res <- summary(electricity[,1:20], lag.max=2)
```

`temperature` *Australian Temperature Data*

Description

A list of daily minimum temperature data (1859 - 2012) from reporting station in Sydney, Australia. Any missing data was previously, linearly, interpolated and leap days were removed for consistency.

Usage

```
temperature
```

Format

`temperature`**:**
A dfts object.

Source

<www.bom.gov.au>

[.dfts *Extract or Replace parts of dfts object*

Description

Extract or replace subsets of dfts objects.

Usage

```
## S3 method for class 'dfts'
x[i, j, ...]

## S3 replacement method for class 'dfts'
x[i, j] <- value
```

Arguments

x	A dfts object. See dfts().
i, j	Numerics for elements to extract.
...	Additional parameters from generic function for extensions.
value	A suitable replacement value for selection.

Value

A dfts object.

Examples

```
electricity[1:3]
electricity[1:3,]
electricity[1:2,1:4]
electricity[,1:4]
tmp <- dfts(matrix(1:9,3,3))
tmp$data
tmp[1,1] <- 10
tmp$data
```

Index