Package: ftsatoo (via r-universe)

May 15, 2026

Type Package

Title Functional Time Series Analysis

Version 0.2.0

Date 2025-08-30

Depends R (>= 3.5.0), forecast, rainbow, sde

Suggests fds, R2jags, meboot, fdadensity, ahead

Imports colorspace, MASS, pcaPP, fda, pdfCluster, ecp, strucchange, e1071, psych, fGarch, KernSmooth, vars, boot, fdapace, LaplacesDemon, evgam, ROOPSD, glue, methods

LazyLoad yes

LazyData yes

LazyDataCompression xz

ByteCompile TRUE

Maintainer T. Moudiki <thierry.moudiki@gmail.com>

Description Functions for visualizing, modeling, (generic) forecasting and hypothesis testing of functional time series.

License GPL-3

NeedsCompilation no

RoxygenNote 7.3.2

Config/pak/sysreqs cmake make libicu-dev libuv1-dev

Repository https://techtonique.r-universe.dev

Date/Publication 2025-08-31 16:47:12 UTC

RemoteUrl https://github.com/thierrymoudiki/ftsatoo

RemoteRef HEAD

RemoteSha ca12ba77444b2fcc1ce8dcbcf444a38476da76a7

Contents

`centre` *Centre Functional Time Series*

Description

Centers a functional time series using various methods.

Usage

```
centre(x, type)
```

Arguments

`x`	A functional time series object.
`type`	Type of centering: "mean", "var", "median", or "trimmed".

Details

The available centering types are:

- "mean": Center by mean function
- "var": Center by variance function
- "median": Center by median function
- "trimmed": Center by trimmed mean function

Value

A centered functional time series.

See Also

`mean.fts`, `median.fts`

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Center by mean
centered_data <- centre(pm_10_GR, type = "mean")

## End(Not run)
```

---

dens2lqd *Function for converting densities to log quantile density functions*

Description

Function for converting densities to log quantile density functions

Usage

```
dens2lqd(
  dens,
  dSup,
  lqdSup = seq(0, 1, length.out = length(dSup)),
  t0 = dSup[1],
  verbose = TRUE
)
```

Arguments

dens	density values on dSup - must be strictly positive (otherwise will truncate) and integrate to 1
dSup	support (grid) for Density domain
lqdSup	support of length M for lqd domain - must begin at 0 and end at 1; (default: seq(0, 1, length(dSup)))
t0	value in dSup for which the cdf value c is retained, i.e. c = F(t0) (default: dSup[1])
verbose	if FALSE, repress some messages (default: TRUE)

Value

list with 'lqdSup' a grid on [0,1], 'lqd' the log quantile density on lqdSup, and 'c' the value of the cdf at t0

References

Functional Data Analysis for Density Functions by Transformation to a Hilbert space, Alexander Petersen and Hans-Georg Mueller, 2016

Examples

```
x <- seq(0,2,length.out =512)
y <- rep(0.5,length.out =512)
y.lqd <- dens2lqd( dens=y, dSup = x) # should equate # log(2)
```

`diff.fts` *Difference Functional Time Series*

Description

Computes differences of a functional time series.

Usage

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

Arguments

`x`	A functional time series object of class `fts` or `sfts`.
`lag`	Lag for differencing (default: 1).
`differences`	Order of differencing (default: 1).
`...`	Additional arguments passed to `diff()`.

Value

A functional time series object with differenced values.

See Also

`ftsm`, `is.fts`

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Compute first differences
diff_data <- diff(pm_10_GR, lag = 1, differences = 1)

# Plot differenced data
plot(diff_data)

## End(Not run)
```

dynupdate *Dynamic Update for Functional Time Series Forecasting*

Description

Performs dynamic updating of functional time series forecasts using various methods.

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", "M", "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	A functional time series object of class fts.
newdata	New observations to incorporate in the forecast.
holdoutdata	Holdout data for validation.
method	Update method: "ts", "block", "ols", "pls", or "ridge".
fmethod	Forecasting method: "arima", "ar", "ets", "ets.na", "rwdrift", or "rw".
pcdmethod	Functional PCA method: "classical", "M", or "rapca".
ngrid	Number of grid points for smoothing (default: max(1000, ncol(data$y))).
order	Number of principal components (default: 6).
robust_lambda	Robustness parameter (default: 2.33).
lambda	Ridge regression parameter (default: 0.01).
value	Logical. If TRUE, return forecast values (default: FALSE).
interval	Logical. If TRUE, compute prediction intervals (default: FALSE).
level	Confidence level for prediction intervals (default: 80).
pimethod	Method for prediction intervals: "parametric" or "nonparametric".
B	Number of bootstrap replications (default: 1000).

Value

If value=TRUE, returns forecast values. Otherwise, returns a list with error measures:

errormse	Mean squared error
errormae	Mean absolute error
errormape	Mean absolute percentage error

See Also

ftsm, forecast.ftsm

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Perform dynamic update
result <- dynupdate(pm_10_GR, newdata = pm_10_GR$y[,1:10],
                    holdoutdata = pm_10_GR$y[,11:12], method = "pls")

## End(Not run)
```

---

facf *Functional Autocorrelation Function*

Description

Computes the functional autocorrelation function for a functional time series.

Usage

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

Arguments

fun_data	A matrix or data frame containing functional time series data.
lag_value_range	Vector of lag values to compute autocorrelation for (default: seq(0, 20, by = 1)).

Details

The functional autocorrelation function measures the correlation between functional observations at different time lags. For lag 0, the value is set to NA.

Value

A vector of autocorrelation values corresponding to the lag values.

See Also

ftsm, forecast.ftsm

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Compute functional autocorrelation
acf_values <- facf(pm_10_GR$y)

# Plot autocorrelation function
plot(seq(0, 20, by = 1), acf_values, type = "l",
     xlab = "Lag", ylab = "Autocorrelation")

## End(Not run)
```

---

fbootstrap *Functional Bootstrap*

Description

Performs bootstrap resampling for functional data.

Usage

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

Arguments

data	A functional time series object of class fts.
estad	Function to compute the statistic of interest (default: func.mean).
alpha	Significance level for confidence intervals (default: 0.05).
nb	Number of bootstrap replications (default: 200).
suav	Smoothing parameter (default: 0.0).

media.dist	Logical. If TRUE, use mean distance (default: FALSE).
graph	Logical. If TRUE, produce graphical output (default: FALSE).
...	Additional arguments passed to the statistic function.

Value

A list containing:

estimate	Original estimate
max.dist	Maximum distance for confidence interval
rep.dist	Bootstrap distances
resamples	Bootstrap resamples
center	Center of bootstrap distribution

See Also

func.mean, ftsm

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Perform functional bootstrap
boot_result <- fbootstrap(pm_10_GR, estad = func.mean, nb = 100)

# Plot results
plot(boot_result$estimate, type = "l", col = "red", lwd = 2)

## End(Not run)
```

---

forecast.ftsm *Forecast a functional time series model*

Description

Produces forecasts from a functional time series model using various univariate time series forecasting methods applied to the coefficients.

Forecasts a functional time series model using various univariate time series methods.

Usage

```
## S3 method for class 'ftsm'
forecast(
  object,
  h = 10,
  method = c("ets", "arima", "ar", "ets.na", "rwdrift", "rw", "struct", "arfima",
    "ridge2f"),
  level = 95,
  jumpchoice = c("fit", "actual"),
  pimethod = c("parametric", "nonparametric"),
  B = 100,
  usedata = nrow(object$coeff),
  adjust = TRUE,
  model = NULL,
  damped = NULL,
  stationary = FALSE,
  FUN = NULL,
  FUN_method = c("block-bootstrap", "surrogate", "kde", "bootstrap", "fitdistr"),
  FUN_nsim = B,
  FUN_block_size = 5,
  FUN_seed = 123L,
  ...
)

## S3 method for class 'ftsm'
forecast(
  object,
  h = 10,
  method = c("ets", "arima", "ar", "ets.na", "rwdrift", "rw", "struct", "arfima",
    "ridge2f"),
  level = 95,
  jumpchoice = c("fit", "actual"),
  pimethod = c("parametric", "nonparametric"),
  B = 100,
  usedata = nrow(object$coeff),
  adjust = TRUE,
  model = NULL,
  damped = NULL,
  stationary = FALSE,
  FUN = NULL,
  FUN_method = c("block-bootstrap", "surrogate", "kde", "bootstrap", "fitdistr"),
  FUN_nsim = B,
  FUN_block_size = 5,
  FUN_seed = 123L,
  ...
)
```

Arguments

object	An object of class ftsm returned by ftsm().
h	Forecast horizon (number of periods to forecast).
method	Forecasting method: "ets" (default), "arima", "ar", "ets.na", "rwdrift", "rw", "struct", or "arfima".
level	Confidence level for prediction intervals (default: 80).
jumpchoice	Method for handling the jump-off point: "fit" (default) or "actual".
pimethod	Method for prediction intervals: "parametric" (default) or "nonparametric".
B	Number of bootstrap replications for nonparametric prediction intervals (default: 100).
usedata	Number of observations to use for fitting (default: all available).
adjust	Logical. If TRUE, adjust forecasts for bias (default: TRUE).
model	ETS model specification (optional).
damped	Logical. If TRUE, use damped trend (optional).
stationary	Logical. If TRUE, force stationarity (default: FALSE).
FUN	forecasting function (if provided, method is ignored)
FUN_method	Method to be used for conformalization (simulation of calibrated residuals) when FUN is not NULL.
FUN_nsim	Number of simulations when FUN is not NULL.
FUN_block_size	Block size for block-bootstrap when FUN is not NULL.
FUN_seed	Seed for reproducibility when FUN is not NULL.
...	Additional arguments passed to the forecasting method.

Details

This function forecasts a functional time series by applying univariate time series forecasting methods to the coefficients obtained from a functional principal component analysis. The forecasts are then transformed back to functional space.

For the method = "ets" option, the model parameter can be specified as:

- A single character string: Applied to all coefficients except the first
- A vector of length nb-1: Applied to coefficients 2 through nb
- A vector of length nb: Applied to all coefficients

The first coefficient (mean function) always uses "ANN" model unless specified otherwise.

Value

An object of class ftsf containing:

mean	Point forecasts as a functional time series object
lower	Lower prediction bounds (parametric method)
upper	Upper prediction bounds (parametric method)

`fitted`	One-step fitted values
`error`	One-step forecast errors
`coeff`	Forecasts and prediction intervals for each coefficient
`coeff.error`	Errors in coefficient estimation
`var`	Variance components
`model`	The original `ftsm` object
`bootsamp`	Bootstrap samples (nonparametric method)

An object of class `ftsf` containing:

`method`	Forecasting method used
`x`	Time points
`y`	Original functional time series
`fitted`	Fitted values
`residuals`	Residuals
`mean`	Point forecasts
`lower`	Lower prediction intervals
`upper`	Upper prediction intervals
`level`	Confidence level
`xname`	Name of x variable
`yname`	Name of y variable

References

Hyndman, R.J., & Shang, H.L. (2009). Forecasting functional time series. Journal of the Korean Statistical Society, 38(3), 199-221.

See Also

`ftsm`, `plot.ftsf`

`ftsm`, `plot.ftsf`

Examples

```
## Not run:
# Fit functional time series model
fmodel <- ftsm(y = fts(data))

# Forecast using ETS method
forecast_ets <- forecast(fmodel, h = 10, method = "ets")

# Forecast using ARIMA method
forecast_arima <- forecast(fmodel, h = 10, method = "arima")

# Forecast with custom function
custom_forecast <- forecast(fmodel, h = 10, FUN = forecast::thetaf)
```

```
# Plot forecasts
plot(forecast_ets)

## End(Not run)

## Not run:
# Fit functional time series model
fit <- ftsm(pm_10_GR, order = 3)
# Forecast 12 periods ahead
fc <- forecast(fit, h = 12, method = "ets")
# Plot forecasts
plot(fc)

fit <- ftsm(pm_10_GR, order = 3, mean=FALSE); fc <- forecast(fit, h = 12, method = "ridge2f")
# Forecast 12 periods ahead
# Plot forecasts
plot(fc)


## End(Not run)
```

`fplsr` *Functional Partial Least Squares Regression*

Description

Performs functional partial least squares regression for functional time series.

Usage

```
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	A functional time series object of class fts.
order	Number of components to include in the model (default: 6).
type	Type of PLS algorithm: "simpls" (default) or "nipals".
unit.weights	Logical. If TRUE, use unit weights (default: TRUE).
weight	Logical. If TRUE, use weighted PLS (default: FALSE).
beta	Weight parameter for exponential weighting (default: 0.1).
interval	Logical. If TRUE, compute prediction intervals (default: FALSE).
method	Method for prediction intervals: "delta" (default) or "boota".
alpha	Significance level for prediction intervals (default: 0.05).
B	Number of bootstrap replications (default: 100).
adjust	Logical. If TRUE, adjust for bias (default: FALSE).
backh	Number of steps back for validation (default: 10).

Value

An object of class fm containing:

x1	Time points
y1	Grid points
ypred	Predicted functional time series
y	Original functional time series
Ypred	Predicted values
B	Regression coefficients
P	X loadings
Q	Y loadings
T	X scores
R	Weights
fitted	Fitted values
residuals	Residuals
meanX	Mean of X
meanY	Mean of Y
call	Function call

References

Hyndman, R.J., & Shang, H.L. (2009). Forecasting functional time series. Journal of the Korean Statistical Society, 38(3), 199-221.

See Also

forecastfplsr, plotfplsr

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Fit functional PLS regression
fit <- fplsr(pm_10_GR, order = 3)

# Plot results
plot(fit)

## End(Not run)
```

`ftsm` *Functional Time Series Model*

Description

Fits a functional time series model using functional principal component analysis.

Usage

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

Arguments

`y`	A functional time series object of class `fts`.
`order`	Number of principal components to include in the model (default: 6).
`ngrid`	Number of grid points for smoothing (default: max(500, ncol(y$y))).
`method`	Method for functional principal component analysis: "classical" (default), "M", or "rapca".
`mean`	Logical. If TRUE, include mean function in the model (default: TRUE).
`level`	Logical. If TRUE, include level component in the model (default: FALSE).
`lambda`	Smoothing parameter for penalized splines (default: 3).

weight	Logical. If TRUE, use weighted functional principal component analysis (default: FALSE).
beta	Weight parameter for exponential weighting (default: 0.1).
...	Additional arguments passed to the functional PCA function.

Value

An object of class ftsm containing:

x1	Time points
y1	Grid points
y	Original functional time series
basis	Basis functions (eigenfunctions)
coeff	Coefficients (scores)
fitted	Fitted values
residuals	Residuals
varprop	Proportion of variance explained by each component
eigen_values	Eigenvalues
wt	Weights
v	Eigenvalues
basis2	Second set of basis functions
coeff2	Second set of coefficients
mean.se	Standard error of the mean function
call	Function call

References

Hyndman, R.J., & Shang, H.L. (2009). Forecasting functional time series. Journal of the Korean Statistical Society, 38(3), 199-221.

See Also

forecast.ftsm, plot.ftsm, summary.fm

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Fit functional time series model
fit <- ftsm(pm_10_GR, order = 3)

# Plot the model
plot(fit)
```

```
# Forecast
forecast(fit, h = 12)

## End(Not run)
```

`is.fts` *Check if Object is a Functional Time Series*

Description

Tests whether an object is a functional time series.

Usage

```
is.fts(x)
```

Arguments

`x`	An object to be tested.

Value

Logical. TRUE if the object is a functional time series, FALSE otherwise.

See Also

`ftsm`

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Check if it's a functional time series
is.fts(pm_10_GR)

## End(Not run)
```

mean.fts *Mean Function for Functional Time Series*

Description

Computes the mean function of a functional time series using various methods.

Usage

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

Arguments

x	A functional time series object of class fts, fds, or sfts.
method	Method for computing the mean: "coordinate" (default), "FM", "mode", "RP", "RPD", or "radius".
na.rm	Logical. If TRUE, remove missing values (default: TRUE).
alpha	Parameter for radius depth method.
beta	Parameter for radius depth method.
weight	Weight parameter for radius depth method.
...	Additional arguments passed to depth functions.

Details

The available methods are:

- "coordinate": Coordinate-wise mean
- "FM": Fraiman-Muniz depth-based mean
- "mode": Mode depth-based mean
- "RP": Random projection depth-based mean
- "RPD": Random projection depth-based mean
- "radius": Radius depth-based mean

Value

A list containing:

x	Grid points
y	Mean function values

References

Fraiman, R., & Muniz, G. (2001). Trimmed means for functional data. Test, 10(2), 419-440.

See Also

median.fts, var.fts, sd.fts

Examples

```
## Not run:
# Load example data
data(pm_10_GR)

# Compute mean function
mean_func <- mean(pm_10_GR, method = "coordinate")

# Plot mean function
plot(mean_func$x, mean_func$y, type = "l",
     xlab = "Time", ylab = "Mean function")

## End(Not run)
```

---

plot.ftsm *Plot Functional Time Series Model*

Description

Creates diagnostic plots for a functional time series model.

Usage

```
plot.ftsm(
  x,
  components,
  components.start = 0,
  xlab1 = x$y$xname,
  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	An object of class ftsm returned by ftsm().
components	Number of components to plot (default: all components).
components.start	Starting component number (default: 0).
xlab1	Label for x-axis of basis function plots.
ylab1	Label for y-axis of basis function plots.
xlab2	Label for x-axis of coefficient plots.
ylab2	Label for y-axis of coefficient plots.
mean.lab	Label for mean function plot.
level.lab	Label for level component plot.
main.title	Main title for the plot.
interaction.title	Title for interaction plots.
basiscol	Color for basis function lines.
coeffcol	Color for coefficient lines.
outlier.col	Color for outlier points.
outlier.pch	Plotting character for outlier points.
outlier.cex	Size of outlier points.
...	Additional arguments passed to plotting functions.

Value

A plot showing basis functions and their corresponding coefficients.

See Also

ftsm, forecast.ftsm

Examples

```
## Not run:
# Fit functional time series model
fit <- ftsm(pm_10_GR, order = 3)

# Plot the model
plot(fit)

# Plot only first 2 components
plot(fit, components = 2)

## End(Not run)
```

Index