

Package: ahead (via r-universe)

October 17, 2026

Type Package

Title Time Series Forecasting with uncertainty quantification

Version 0.39.0

Date 2026-10-17

Author T. Moudiki

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

Description Univariate and multivariate time series forecasting with uncertainty quantification.

License BSD_3_clause + file LICENSE

Imports Rcpp (>= 1.0.6), foreach, tseries

Depends R (>= 3.5.0)

Suggests caret, cclust, dfoptim, doSNOW, doParallel, knitr, rmarkdown, testthat, fpp2, glmnet, e1071, gam, quantreg, randomForest, spatial, vars, roxygen2, ForecastComb, ranger, mboost, misc, Mcomp, fGarch, VineCopula, forecast (>= 8.0), ggplot2 (>= 3.0.0), randtoolbox (>= 1.17), simulatetimeseries

LinkingTo Rcpp

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Additional_repositories <https://r-packages.techtonique.net>

Remotes thierrymoudiki/ForecastComb, thierrymoudiki/misc

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-10-17 13:24:54 UTC

RemoteUrl <https://github.com/Techtonique/ahead>

RemoteRef HEAD

RemoteSha d9ec67d8adc5a6230212b1829a95a76e104d378c

Contents

agnosticgarchf	3
armagarchf	4
basicf	6
comb_GLMNET	7
comb_OLS	9
comb_Ridge	10
computeattention	12
condvolf	13
conformalize	15
contextridge2f	16
create_2level_hts	17
createtrendseason	18
ctxthetaf	18
direct_sampling	20
dynrmf	21
dynrmf_sensi	23
dynrmf_shap	24
eatf	25
estimate_theta_slope	27
example_2level_forecast	29
fit_func	29
fitdistr_ahead	30
fitforecast	32
generate_synthetic_ts	33
genericforecast	34
geterror	35
getreturns	36
getsimulations	36
glmthetaf	37
loessf	38
loocvridge2f	40
meboot	42
ml_forecast	45
mlarchf	46
mlf	48
plot.foreccomb_res	49
plot.mtsforecast	50
plot.synthetic_ts	51
plot_dynrmf_sensitivity	52
plot_dynrmf_shap_waterfall	53
plot_hts_forecast	53
plot_simulations	54
predict.foreccomb_res	55
predict_func	56
print.summary.synthetic_ts	57
removenas	57

rfitdistr	58
rgaussiandens	59
ridge	59
ridge2f	61
rmultivariate	64
rsurrogate	65
sequential_conformal_hts	66
simple_forecast	66
simulator	67
splitts	68
stackridge2f	69
summary.foreccomb_res	70
summary.synthetic_ts	71
topdown_forecast	71
varf	72
Index	74

agnosticgarchf	<i>ANY MODEL+GARCH(1, 1) forecasting</i>
----------------	--

Description

ANY MODEL+GARCH(1, 1) forecasting

Usage

```
agnosticgarchf(  
  y,  
  h = 5,  
  level = 95,  
  FUN = forecast::auto.arima,  
  seed = 123,  
  ...  
)
```

Arguments

y	a univariate time series
h	number of periods for forecasting
level	confidence level for prediction intervals
FUN	forecasting function for the main model; default ahead::dynrmf
seed	reproducibility seed
...	Additional arguments passed to the forecasting function and fGarch::garchFit

Value

An object of class "forecast"; a list containing the following elements:

model	A list containing information about the fitted model
method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
x	The original time series
sims	Simulations of ANYMODEL+GARCH(1, 1)

Author(s)

T. Moudiki

Examples

```
y <- datasets::EuStockMarkets[ , "DAX"]

# require(forecast)
# z <- ahead::agnosticgarchf(y=y, h=20)
# plot(z)
```

armagarchf

ARMA(1, 1)-GARCH(1, 1) forecasting (with simulation)

Description

ARMA(1, 1)-GARCH(1, 1) forecasting (with simulation)

Usage

```
armagarchf(
  y,
  h = 5,
  level = 95,
  B = NULL,
  cl = 1L,
  dist = c("student", "gaussian"),
  seed = 123,
  ...
)
```

Arguments

<code>y</code>	a univariate time series
<code>h</code>	number of periods for forecasting
<code>level</code>	confidence level for prediction intervals
<code>B</code>	number of simulations for <code>arma.sim</code>
<code>cl</code>	an integer; the number of clusters for parallel execution
<code>dist</code>	distribution of innovations ("student" or "gaussian")
<code>seed</code>	reproducibility seed
<code>...</code>	Additional arguments passed to <code>fGarch::garchFit</code>

Value

An object of class "forecast"; a list containing the following elements:

<code>model</code>	A list containing information about the fitted model
<code>method</code>	The name of the forecasting method as a character string
<code>mean</code>	Point forecasts for the time series
<code>lower</code>	Lower bound for prediction interval
<code>upper</code>	Upper bound for prediction interval
<code>x</code>	The original time series
<code>sims</code>	Simulations of ARMA(1, 1)-GARCH(1, 1)

Author(s)

T. Moudiki

Examples

```
y <- datasets::EuStockMarkets[ , "DAX"]

# require(forecast)
# z <- ahead::armagarchf(y=y, h=200)
# plot(z)
```

basicf	<i>Basic forecasting (mean, median, random walk)</i>
--------	--

Description

Basic forecasting functions for multivariate time series

Usage

```
basicf(
  y,
  h = 5,
  level = 95,
  method = c("mean", "median", "rw"),
  type_pi = c("gaussian", "bootstrap", "blockbootstrap", "movingblockbootstrap"),
  block_length = NULL,
  seed = 1,
  B = 100,
  show_progress = TRUE
)
```

Arguments

y	A multivariate time series of class <code>ts</code> or a matrix
h	Forecasting horizon
level	Confidence level for prediction intervals
method	forecasting method, either "mean", "median", or random walk ("rw")
type_pi	type of prediction interval currently, "gaussian", "bootstrap", "blockbootstrap" or "movingblockbootstrap"
block_length	length of block for (circular) "blockbootstrap" or "movingblockbootstrap"
seed	reproducibility seed for <code>type_pi == 'bootstrap'</code>
B	Number of bootstrap replications for <code>type_pi == 'bootstrap'</code>
show_progress	A boolean; show progress bar for bootstrapping? Default is TRUE.

Value

An object of class "mtsforecast"; a list containing the following elements:

method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
sims	Model simulations for bootstrapping (basic, or block)
x	The original time series
residuals	Residuals from the fitted model
coefficients	Regression coefficients for <code>type_pi == 'gaussian'</code> for now

Examples

```
## Not run:

require(fpp2)

res <- ahead::basicf(fpp2::insurance, h=10)
par(mfrow=c(1, 2))
plot(res, "TV.advert")
plot(res, "Quotes")

res <- ahead::basicf(fpp2::insurance, method="rw", h=10)
par(mfrow=c(1, 2))
plot(res, "TV.advert")
plot(res, "Quotes")

# block bootstrap
res3 <- ahead::basicf(fpp2::insurance, h=10, type_pi = "bootstrap", B=10)
res5 <- ahead::basicf(fpp2::insurance, h=10, type_pi = "blockbootstrap", B=10,
                      block_length = 4)

print(res3$sims[[2]])
print(res5$sims[[2]])

par(mfrow=c(2, 2))
plot(res3, "Quotes")
plot(res3, "TV.advert")
plot(res5, "Quotes")
plot(res5, "TV.advert")

# moving block bootstrap
res6 <- ahead::basicf(fpp2::insurance, h=10,
                      type_pi = "movingblockbootstrap", B=10,
                      block_length = 4, method = "rw")

par(mfrow=c(1, 2))
plot(res6, "Quotes")
plot(res6, "TV.advert")

## End(Not run)
```

Description

Computes forecast combination weights using GLMNET Regression (OLS) regression.

Usage

```
comb_GLMNET(x, custom_error = NULL)
```

Arguments

x	An object of class 'foreccomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.
custom_error	Custom error function

Details

The function integrates the GLMNET Regression forecast combination implementation of the *ForecastCombinations* package into ForecastComb.

The results are stored in an object of class 'ForecastComb::foreccomb_res', for which separate plot and summary functions are provided.

Value

Returns an object of class ForecastComb::foreccomb_res with the following components:

Method	Returns the best-fit forecast combination method.
Models	Returns the individual input models that were used for the forecast combinations.
Weights	Returns the combination weights obtained by applying the combination method to the training set.
Intercept	Returns the intercept of the linear regression.
Fitted	Returns the fitted values of the combination method for the training set.
Accuracy_Train	Returns range of summary measures of the forecast accuracy for the training set.
Forecasts_Test	Returns forecasts produced by the combination method for the test set. Only returned if input included a forecast matrix for the test set.
Accuracy_Test	Returns range of summary measures of the forecast accuracy for the test set. Only returned if input included a forecast matrix and a vector of actual values for the test set.
Input_Data	Returns the data forwarded to the method.

Examples

```
## Not run:

library(ForecastComb)

data(electricity)

print(head(electricity))

forecasting_methods <- colnames(electricity)[1:5]
```

```

train_obs <- electricity[1:84, "Actual"]
train_pred <- electricity[1:84, forecasting_methods]
test_obs <- electricity[85:123, "Actual"]
test_pred <- electricity[85:123, forecasting_methods]
data <- ForecastComb::foreccomb(train_obs, train_pred, test_obs, test_pred)

obj <- ahead::comb_GLMNET(data))

## End(Not run)

```

comb_OLS

*Ordinary Least Squares Forecast Combination***Description**

Computes forecast combination weights using ordinary least squares (OLS) regression.

Usage

```
comb_OLS(x, custom_error = NULL)
```

Arguments

x	An object of class 'foreccomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.
custom_error	Custom error function

Details

The function integrates the ordinary least squares (OLS) forecast combination implementation of the *ForecastCombinations* package into ForecastComb.

The OLS combination method (Granger and Ramanathan (1984)) uses ordinary least squares to estimate the weights, $\mathbf{w}^{OLS} = (w_1, \dots, w_N)'$, as well as an intercept, b , for the combination of the forecasts.

Suppose that there are N not perfectly collinear predictors $\mathbf{f}_t = (f_{1t}, \dots, f_{Nt})'$, then the forecast combination for one data point can be represented as:

$$y_t = b + \sum_{i=1}^N w_i f_{it}$$

An appealing feature of the method is its bias correction through the intercept – even if one or more of the individual predictors are biased, the resulting combined forecast is unbiased. A disadvantage of the method is that it places no restriction on the combination weights (i.e., they do not add up to 1 and can be negative), which can make interpretation hard. Another issue, documented in Nowotarski et al. (2014), is the method's unstable behavior when predictors are highly correlated

(which is the norm in forecast combination): Minor fluctuations in the sample can cause major shifts of the coefficient vector ('bouncing betas') – often causing poor out-of-sample performance. This issue is addressed by the `ForecastComb::comb_LAD` method that is more robust to outliers.

The results are stored in an object of class 'ForecastComb::foreccomb_res', for which separate plot and summary functions are provided.

Value

Returns an object of class `ForecastComb::foreccomb_res` with the following components:

Method	Returns the best-fit forecast combination method.
Models	Returns the individual input models that were used for the forecast combinations.
Weights	Returns the combination weights obtained by applying the combination method to the training set.
Intercept	Returns the intercept of the linear regression.
Fitted	Returns the fitted values of the combination method for the training set.
Accuracy_Train	Returns range of summary measures of the forecast accuracy for the training set.
Forecasts_Test	Returns forecasts produced by the combination method for the test set. Only returned if input included a forecast matrix for the test set.
Accuracy_Test	Returns range of summary measures of the forecast accuracy for the test set. Only returned if input included a forecast matrix and a vector of actual values for the test set.
Input_Data	Returns the data forwarded to the method.

Examples

```
obs <- rnorm(100)
preds <- matrix(rnorm(1000, 1), 100, 10)
train_o<-obs[1:80]
train_p<-preds[1:80,]
test_o<-obs[81:100]
test_p<-preds[81:100,]

data<-ForecastComb::foreccomb(train_o, train_p, test_o, test_p)
ahead::comb_OLS(data)
```

comb_Ridge

Ridge Regression Forecast Combination

Description

Computes forecast combination weights using Ridge Regression (OLS) regression.

Usage

```
comb_Ridge(x, custom_error = NULL)
```

Arguments

x	An object of class 'foreccomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.
custom_error	Custom error function

Details

The function integrates the Ridge Regression forecast combination implementation of the *Forecast-Combinations* package into ForecastComb.

The results are stored in an object of class 'ForecastComb::foreccomb_res', for which separate plot and summary functions are provided.

Value

Returns an object of class `ForecastComb::foreccomb_res` with the following components:

Method	Returns the best-fit forecast combination method.
Models	Returns the individual input models that were used for the forecast combinations.
Weights	Returns the combination weights obtained by applying the combination method to the training set.
Intercept	Returns the intercept of the linear regression.
Fitted	Returns the fitted values of the combination method for the training set.
Accuracy_Train	Returns range of summary measures of the forecast accuracy for the training set.
Forecasts_Test	Returns forecasts produced by the combination method for the test set. Only returned if input included a forecast matrix for the test set.
Accuracy_Test	Returns range of summary measures of the forecast accuracy for the test set. Only returned if input included a forecast matrix and a vector of actual values for the test set.
Input_Data	Returns the data forwarded to the method.

Examples

```
obs <- rnorm(100)
preds <- matrix(rnorm(1000, 1), 100, 10)
train_o<-obs[1:80]
train_p<-preds[1:80,]
test_o<-obs[81:100]
test_p<-preds[81:100,]

data<-ForecastComb::foreccomb(train_o, train_p, test_o, test_p)
ahead::comb_Ridge(data)
```

computeattention	<i>Compute global attention weights and context vectors for time series</i>
------------------	---

Description

Compute global attention weights and context vectors for time series

Usage

```
computeattention(
  series,
  attention_type = c("dot_product", "scaled_dot_product", "cosine", "exponential",
    "gaussian", "linear", "value_based", "hybrid", "parametric"),
  window_size = 3,
  decay_factor = 5,
  temperature = 1,
  sigma = 1,
  sensitivity = 1,
  alpha = 0.5,
  beta = 0.5
)
```

Arguments

series	Numeric vector containing the time series of length n
attention_type	String specifying the type of attention mechanism to use. Options are: "cosine", "exponential", "dot_product", "scaled_dot_product", "gaussian", "linear", "value_based", "hybrid", "parametric". Default is "cosine".
window_size	Integer parameter for window size (applicable for "cosine" attention).
decay_factor	Double for decay factor (applicable for "exponential" attention).
temperature	Double for temperature (applicable for "scaled_dot_product" attention).
sigma	Double for sigma (applicable for "gaussian" attention).
sensitivity	Double for sensitivity (applicable for "value_based" or "hybrid" attention).
alpha	Double for alpha (applicable for "parametric" attention).
beta	Double for beta (applicable for "parametric" attention).

Value

List containing:

attention_weights	$n \times n$ matrix where entry (i,j) represents the attention weight of time j on time i . Only entries $j \leq i$ are non-zero (causal attention).
context_vectors	Vector of length n where each entry i is the weighted sum of all values up to time i , using the attention weights.

Examples

```
# For a series of length 5 using "cosine" attention
series <- c(1, 2, 3, 4, 5)
result <- computeattention(series, attention_type = "cosine", window_size = 3)

# attention_weights will be 5x5 matrix
# context_vectors will be length 5
dim(result$attention_weights) # [1] 5 5
length(result$context_vectors) # [1] 5
```

condvolf	<i>Model-agnostic statistical probabilistic forecasting with conditional volatility</i>
----------	---

Description

Combines a mean model and a volatility model to produce probabilistic forecasts. Uncertainty is propagated from two sources: simulated innovation draws and simulated volatility paths derived from the variance model's residuals via bootstrap.

Usage

```
condvolf(
  y,
  h = 10L,
  mean_model = forecast::auto.arima,
  sigma_model = forecast::auto.arima,
  innovation = c("empirical", "gaussian", "student"),
  level = 95,
  B = 1000L,
  bootstrap_vol = TRUE,
  ...
)
```

Arguments

y	A numeric vector or time series of class ts.
h	Forecasting horizon.
mean_model	Function to fit the mean model (default: forecast::auto.arima). Options include forecast::auto.arima, forecast::ets, forecast::thetaf, or any custom function that returns an object with forecast method.
sigma_model	Function to fit the variance model on squared residuals (default: forecast::auto.arima).
innovation	Distribution for standardized innovations. One of "gaussian" (fastest, often sufficient), "student" (heavy-tailed), or "empirical" (non-parametric, most flexible).

level	Confidence level for prediction intervals (default: 95).
B	Number of simulated paths (default: 2000).
bootstrap_vol	Whether to bootstrap volatility residuals (default: TRUE). If FALSE, uses parametric normal approximation for volatility.
...	Additional arguments passed to mean_model and sigma_model.

Value

A forecast object with components:

- x Original time series.
- mean Point forecast (mean of simulations).
- lower Lower prediction interval bound.
- upper Upper prediction interval bound.
- level Confidence level.
- sims Matrix of simulated forecast paths (h x B).
- model List containing mean and sigma model fits.
- residuals Residuals from the mean model.
- standardized_residuals Scaled standardized residuals used for innovation fitting.
- method Character string describing the method used.

Examples

```
library(forecast)

# Basic usage with Google stock data
y <- fpp2::goog200

# Gaussian innovations (fastest, often sufficient)
fc1 <- condvolf(y, h = 20, innovation = "gaussian")
plot(fc1)

# Student-t innovations (heavy-tailed)
fc2 <- condvolf(y, h = 20, innovation = "student")
plot(fc2)

# Empirical innovations (non-parametric)
fc3 <- condvolf(y, h = 20, innovation = "empirical")
plot(fc3)

# Using different mean and volatility models
fc4 <- condvolf(y, h = 20,
               mean_model = forecast::thetaf,
               sigma_model = forecast::ets,
               innovation = "gaussian")

# Compare prediction intervals
par(mfrow = c(2, 2))
```

```

plot(fc1, main = "Gaussian innovations")
plot(fc2, main = "Student-t innovations")
plot(fc3, main = "Empirical innovations")
plot(fc4, main = "Different mean and volatility models")
par(mfrow = c(1, 1))

```

conformalize

*Conformalize a forecasting function***Description**

This function allows to conformalize any forecasting function.

Usage

```

conformalize(
  FUN,
  y,
  h,
  level = 95,
  method = c("block-bootstrap", "surrogate", "kde", "bootstrap", "fitdistr", "meboot"),
  nsim = 250L,
  block_size = 5,
  seed = 123L,
  B = NULL,
  ...
)

```

Arguments

<code>FUN</code>	A forecasting function.
<code>y</code>	A time series (ts object or vector).
<code>h</code>	Forecasting horizon.
<code>level</code>	Confidence level.
<code>method</code>	Method to be used for conformalization (simulation of calibrated residuals).
<code>nsim</code>	Number of simulations.
<code>block_size</code>	Block size for block-bootstrap.
<code>seed</code>	Seed for reproducibility.
<code>B</code>	Alias for <code>nsim</code>
<code>...</code>	Additional arguments to be passed to the forecasting function.

Value

An object of class `forecast`.

Examples

```
## Not run:
y <- fdeaths
h <- 25L
obj <- conformalize(FUN=forecast::ets, y, h); plot(obj)
obj <- conformalize(FUN=HoltWinters, y=y, h=h, seasonal = "mult"); plot(obj)

## End(Not run)
```

contextridge2f

Ridge Regression Forecasting with Attention-based Context Vectors

Description

This function performs ridge regression forecasting using attention-based context vectors as external regressors. Context vectors are computed from the training data using various attention mechanisms and then used to enhance the forecast model.

Usage

```
contextridge2f(
  y,
  h = 5L,
  attention_type = "exponential",
  window_size = 3,
  decay_factor = 5,
  temperature = 1,
  sigma = 1,
  sensitivity = 1,
  alpha = 0.5,
  beta = 0.5,
  ...
)
```

Arguments

y	A multivariate time series object.
h	Integer. Forecast horizon. Defaults to 5.
attention_type	String specifying the type of attention mechanism. Options are: "cosine", "exponential", "dot_product", "scaled_dot_product", "gaussian", "linear", "value_based", "hybrid", "parametric". Default is "exponential".
window_size	Integer parameter for window size (applicable for "cosine" attention). Defaults to 3.
decay_factor	Double for decay factor (applicable for "exponential" and "hybrid" attention). Defaults to 5.0.

temperature	Double for temperature (applicable for "scaled_dot_product" attention). Defaults to 1.0.
sigma	Double for sigma (applicable for "gaussian" attention). Defaults to 1.0.
sensitivity	Double for sensitivity (applicable for "value_based" or "hybrid" attention). Defaults to 1.0.
alpha	Double for alpha (applicable for "parametric" attention). Defaults to 0.5.
beta	Double for beta (applicable for "parametric" attention). Defaults to 0.5.
...	Additional arguments passed to <code>ahead::ridge2f</code> (e.g., lags, lambda_1, lambda_2, nb_hidden, etc.).

Details

This approach allows the model to leverage temporal dependencies captured by attention mechanisms, potentially improving forecast accuracy by incorporating weighted historical information.

Value

An object returned by `ahead::ridge2f` for the forecast, typically a list including mean and prediction intervals.

See Also

[ahead::computeattention](#)

Examples

```
plot(contextridge2f(AirPassengers, h = 15, lags = 15, attention_type = "exponential"))

plot(contextridge2f(fdeaths, h = 20, lags = 15, attention_type = "exponential"))
```

create_2level_hts	Create simple 2-level hierarchical time series
-------------------	--

Description

Create simple 2-level hierarchical time series

Usage

```
create_2level_hts(bts)
```

Arguments

bts Bottom-level time series matrix (T x m)

Value

List with total and bottom series

createtrendseason	<i>Create trend and seasonality features for univariate time series</i>
-------------------	---

Description

Create trend and seasonality features for univariate time series

Usage

```
createtrendseason(y)
```

Arguments

`y` a univariate time series

Value

a vector or matrix of features

Examples

```
y <- ts(rnorm(100), start = 1, frequency = 12)
createtrendseason(y)

createtrendseason(USAccDeaths)
```

ctxthetaf	<i>Context-Aware Theta method forecast</i>
-----------	--

Description

Returns forecasts and prediction intervals for a context-aware theta method forecast.

Usage

```
ctxthetaf(
  y,
  h = if (frequency(y) > 1) 2 * frequency(y) else 10,
  level = 95,
  theta = 0.5,
  fit_func = lm,
  predict_func = predict,
  type_pi = c("gaussian", "block-bootstrap", "surrogate", "kde", "bootstrap", "fitdistr",
    "meboot"),
  nsim = 100L,
```

```

    block_size = 5,
    seed = 123L,
    B = NULL,
    x = y,
    ...
)

```

Arguments

y	Time series (ts object or numeric vector)
h	Forecast horizon (default: 2*frequency for seasonal, 10 for non-seasonal)
level	Confidence level(s) for prediction intervals (default: 95)
theta	Theta coefficient for drift weighting (default: 0.5, giving theta line = 2) • theta = 0: No drift (pure SES) • theta = 0.5: Classical Theta method (theta line = 2) • theta = 1: Full drift • theta > 1: Amplified drift
fit_func	Model fitting function (default: lm)
predict_func	Prediction function (default: predict)
type_pi	Default is "gaussian", otherwise, uses sequential split conformal prediction as in Moudiki, T. (2025)
nsim	Number of simulations when type_pi is not "gaussian"
block_size	Size of block when type_pi is "block-bootstrap"
seed	Reproducibility seed
B	Alias for nsim
x	Deprecated, use y instead
...	Additional arguments passed to fit_func

Details

The classical theta method of Assimakopoulos and Nikolopoulos (2000) is equivalent to simple exponential smoothing with drift, using $\theta = 2$. This implementation extends the method by:

1. Allowing flexible specification of theta parameter (default 0.5, which gives $\theta = 2$)
2. Using machine learning models to capture non-linear drift patterns
3. Computing time-varying slopes via numerical differentiation

Value

An object of class forecast

Author(s)

T. Moudiki (based on RJH)

References

- Hyndman, R.J., and Billah, B. (2003) Unmasking the Theta method. *International J. Forecasting*, **19**, 287-290.
- Moudiki, T. (2025). Conformal Predictive Simulations for Univariate Time Series. *Proceedings of Machine Learning Research*, **266**, 1-2.

See Also

`forecast::thetaf()`, `forecast::ses()`

Examples

```
# Classical theta-like behavior (theta = 0.5)
fit1 <- ctxthetaf(AirPassengers)
plot(fit1)

# No drift (theta = 0)
fit2 <- ctxthetaf(AirPassengers, theta = 0)
plot(fit2)

# Amplified drift (theta = 1)
fit3 <- ctxthetaf(AirPassengers, theta = 1)
plot(fit3)

# With Random Forest
fit4 <- ctxthetaf(AirPassengers, fit_func = randomForest::randomForest)
plot(fit4)
```

direct_sampling

Direct sampling

Description

Direct sampling

Usage

```
direct_sampling(
  data = NULL,
  n = 100L,
  method = c("kde", "surrogate", "bootstrap"),
  kde = NULL,
  seed = NULL,
  ...
)
```

Arguments

data	A numeric vector or matrix.
n	The number of samples to draw.
method	The method to use for sampling.
kde	The kernel density estimate to use for sampling.
seed	The seed to use for sampling.
...	Additional arguments to pass to the density function.

 dynrmf

Dynamic regression model

Description

Adapted from forecast::nnetar, with alternative fitting functions (see examples)

Usage

```

dynrmf(
  y,
  h = 5,
  level = 95,
  fit_func = ahead::ridge,
  predict_func = predict,
  fit_params = NULL,
  type_pi = c("gaussian", "E", "A", "T"),
  xreg_fit = NULL,
  xreg_predict = NULL,
  ...
)
```

Arguments

y	A numeric vector or time series of class ts
h	Forecasting horizon
level	Confidence level for prediction intervals
fit_func	Fitting function (Statistical/ML model). Default is Ridge regression.
predict_func	Prediction function (Statistical/ML model)
fit_params	a list of additional parameters for the fitting function fit_func (see examples)
type_pi	Type of prediction interval (currently "gaussian", ETS: "E", Arima: "A" or Theta: "T")
xreg_fit	Optionally, a vector or matrix of external regressors, which must have the same number of rows as y. Must be numeric.
xreg_predict	Future values of external regressor variables.
...	additional parameters

Value

a list; an object of class forecast.

The function summary is used to obtain and print a summary of the results.

The generic accessor functions fitted.values and residuals extract useful features.

Author(s)

T. Moudiki

References

Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice. OTexts.

Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E, Yasmeeen F (2021). forecast: Forecasting functions for time series and linear models. R package version 8.14, <URL: <https://pkg.robjhyndman.com/forecast/>>.

Examples

```
# Example 0: with Ridge regression
```

```
par(mfrow=c(3, 2))
plot(dynrmf(USAccDeaths, h=20, level=95))
plot(dynrmf(AirPassengers, h=20, level=95))
plot(dynrmf(lynx, h=20, level=95))
plot(dynrmf(WWWusage, h=20, level=95))
plot(dynrmf(Nile, h=20, level=95))
plot(dynrmf(fdeaths, h=20, level=95))
```

```
# Example 1: with Random Forest
```

```
## Not run:
```

```
require(randomForest)
```

```
par(mfrow=c(3, 2))
plot(dynrmf(USAccDeaths, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
plot(dynrmf(AirPassengers, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
plot(dynrmf(lynx, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
plot(dynrmf(WWWusage, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
plot(dynrmf(Nile, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
plot(dynrmf(fdeaths, h=20, level=95, fit_func = randomForest::randomForest,
  fit_params = list(ntree = 50), predict_func = predict))
```

```
## End(Not run)

# Example 2: with SVM

## Not run:

require(e1071)

par(mfrow=c(2, 2))
plot(dynrmf(fdeaths, h=20, level=95, fit_func = e1071::svm,
  fit_params = list(kernel = "linear"), predict_func = predict))
plot(dynrmf(fdeaths, h=20, level=95, fit_func = e1071::svm,
  fit_params = list(kernel = "polynomial"), predict_func = predict))
plot(dynrmf(fdeaths, h=20, level=95, fit_func = e1071::svm,
  fit_params = list(kernel = "radial"), predict_func = predict))
plot(dynrmf(fdeaths, h=20, level=95, fit_func = e1071::svm,
  fit_params = list(kernel = "sigmoid"), predict_func = predict))

## End(Not run)
```

dynrmf_sensi

Compute First-Order Sensitivity Effects for Dynamic Regression Forecasts

Description

This function computes the sensitivity of a forecasted time series (y) to small perturbations in an external regressor ($xreg$) using the `dynrmf` function.

Usage

```
dynrmf_sensi(y, xreg, h = NULL, level = 99, zero = 1e-04, ...)
```

Arguments

<code>y</code>	A univariate time series object to forecast.
<code>xreg</code>	A time series of external regressors (same length as <code>y</code>).
<code>h</code>	Forecast horizon (length of test set). If <code>NULL</code> , uses length of <code>xreg</code> .
<code>level</code>	Confidence level for <code>dynrmf</code> (default 99).
<code>zero</code>	Small positive constant to avoid division by zero (default <code>1e-4</code>).
<code>...</code>	Additional parameters to be passed to <code>ahead::dynrmf</code>

Value

A list containing:

effects_mean Sensitivity effects on the mean forecast.

forecast_central Forecast without xreg.

forecast_with_xreg Forecast with xreg.

forecast_plus Forecast with positive perturbation on xreg.

forecast_minus Forecast with negative perturbation on xreg.

h Forecast horizon used.

xreg_test Test set of xreg used in forecasting.

hh Perturbation applied to xreg.

split List with training and testing sets of y.

Examples

```
sensitivity_results_auto <- ahead::dynrmf_sensi(
  y = fpp2::uschange[, "Consumption"],
  xreg = fpp2::uschange[, "Income"],
  h = 10
)

plot1 <- ahead::plot_dynrmf_sensitivity(sensitivity_results_auto,
                                     title = "Sensitivity of Consumption to Income",
                                     y_label = "Effect (DeltaConsumption / DeltaIncome)")

print(plot1)
```

dynrmf_shap

Compute exact Shapley values for an ahead::dynrmf model

Description

Compute exact Shapley values for an ahead::dynrmf model

Usage

```
dynrmf_shap(
  y,
  xreg_fit,
  xreg_predict,
  fit_func = ahead::ridge,
  verbose = TRUE,
  tol = 1e-10
)
```

Arguments

<code>y</code>	Training time series (passed to <code>ahead::dynrmf</code> as <code>y</code>)
<code>xreg_fit</code>	Regressor matrix used during training
<code>xreg_predict</code>	Regressor matrix for the forecast horizon ($h \times p$); h and feature names are derived from this matrix.
<code>fit_func</code>	Fitting function for <code>ahead::dynrmf</code> (default: <code>ahead::ridge</code>)
<code>verbose</code>	Print progress per horizon and efficiency check (default: <code>TRUE</code>)
<code>tol</code>	Tolerance for the efficiency-axiom check (default: <code>1e-10</code>)

Value

A list of class "dynrmf_shap" with: `$S` : $h \times p$ matrix of Shapley values (per variable, per horizon)
`$X` : test regressor matrix `$baseline` : scalar baseline (mean of empty-coalition forecasts) `$baselines` : per-horizon baseline forecasts `$predictions` : per-horizon full-coalition forecasts `$feature_names`
`$h`

eatf

*Combined ets-arma-theta forecasts***Description**

Combined ets, arima, and theta (eat) forecasting (uses `forecast::ets`, `forecast::auto.arima`, `forecast::thetaf`)

Usage

```
eatf(
  y,
  h = 5,
  level = 95,
  method = c("EAT", "E", "A", "T"),
  weights = rep(1/3, 3),
  type_pi = c("gaussian", "E", "A", "T"),
  ...
)
```

Arguments

<code>y</code>	a univariate time series
<code>h</code>	number of periods for forecasting
<code>level</code>	confidence level for prediction intervals
<code>method</code>	forecasting method: "E" for <code>forecast::ets</code> ; "A" for <code>forecast::auto.arima</code> ; "T" for <code>forecast::thetaf</code> ; or "EAT" for the combination of the three (default, with weights)

weights	weights for each method, in method EAT. Must add up to 1.
type_pi	type of prediction interval: currently ETS: "E", Auto.Arima: "A" or Theta: "T"
...	additional parameters to be passed to <code>forecast::ets</code> , <code>forecast::auto.arima</code> , <code>forecast::thetaf</code> and <code>forecast::forecast</code>

Details

ensemble forecasts obtained from `forecast::ets`, `forecast::auto.arima` and `forecast::theta` (with weights)

Value

An object of class "forecast"; a list containing the following elements:

model	A list containing information about the fitted model
method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
x	The original time series
residuals	Residuals from the fitted model

Author(s)

T. Moudiki

References

Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E, Yasmeeen F (2021). `forecast`: Forecasting functions for time series and linear models. R package version 8.14, <URL: <https://pkg.robjhyndman.com/forecast/>>.

Hyndman RJ, Khandakar Y (2008). 'Automatic time series forecasting: the forecast package for R.' *Journal of Statistical Software*, 26 (3), 1-22. <URL: <https://www.jstatsoft.org/article/view/v027i03>>.

Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. *International Journal of Forecasting* 16, 521-530.

Hyndman, R.J., and Billah, B. (2003) Unmasking the Theta method. *International J. Forecasting*, 19, 287-290.

Examples

```
require(forecast)

## Not run:

#print(ahead::eatf(WWWusage, method = "EAT",
#weights = c(0.5, 0, 0.5)))
```

```

#print(ahead::eatf(WWWusage, method = "EAT"))
print(ahead::eatf(WWWusage, method = "E"))
#print(ahead::eatf(WWWusage, method = "A"))
print(ahead::eatf(WWWusage, method = "T"))

obj <- ahead::eatf(WWWusage, method = "EAT",
weights = c(0, 0.5, 0.5), h=10,
type_pi = "T")
plot(obj)

obj <- ahead::eatf(WWWusage, method = "EAT",
weights = c(0, 0.5, 0.5), h=10, type_pi="A")
plot(obj)

par(mfrow=c(3, 2))
plot(ahead::eatf(USAccDeaths, h=10, level=95))
plot(ahead::eatf(AirPassengers, h=10, level=95, type_pi = "T"))
#plot(ahead::eatf(lynx, h=10, level=95, type_pi = "A"))
plot(ahead::eatf(WWWusage, h=10, level=95, type_pi = "E"))
plot(ahead::eatf(Nile, h=10, level=95))
plot(ahead::eatf(fdeaths, h=10, level=95))

## End(Not run)

```

estimate_theta_slope	<i>Estimate local slope of a fitted model's response surface with respect to time</i>
----------------------	---

Description

Fits a user-supplied model to a univariate time series (using time index and an auxiliary random covariate as predictors), then numerically differentiates the fitted model's predictions with respect to time using a central finite-difference (perturbation) scheme. Slopes are computed for every historical time point as well as for h future time points, making this useful for characterizing local trend/momentum in the series both in-sample and over a forecast horizon.

Usage

```
estimate_theta_slope(fit_func, predict_func, y, h = 0, ...)
```

Arguments

fit_func	A function used to fit the model. It is tried successively with a formula interface ($y \sim .$, $data = \dots$), an x/y interface, and a positional (x, y) interface, so it should support at least one of these calling conventions (e.g. <code>lm</code> , <code>randomForest</code> , <code>glmnet</code> , etc.).
----------	---

<code>predict_func</code>	A function used to generate predictions from the object returned by <code>fit_func</code> . It is called as <code>predict_func(object, newdata)</code> and, if that fails, as <code>predict_func(object, as.matrix(newdata))</code> . Must return a numeric vector of predictions.
<code>y</code>	A numeric vector containing the univariate time series of historical observations.
<code>h</code>	Integer; forecasting horizon, i.e. the number of future time points (beyond the length of <code>y</code>) for which slopes should also be computed. Defaults to 0 (in-sample slopes only).
<code>...</code>	Additional arguments passed on to <code>fit_func</code> .

Details

The function builds a data frame with columns `y` (the response, padded with NA for the `h` forecast periods), `t` (an integer time index running from 0 to $n + h - 1$), and `z` (an i.i.d. standard normal covariate used only to satisfy models that require more than one predictor). The model is fit on the first `n` rows (the historical data) via `fit_func`, trying in order:

- the formula interface `fit_func(y ~ ., data = df_train, ...)`;
- the matrix/vector interface `fit_func(x = cbind(t, z), y = y, ...)`;
- the positional interface `fit_func(cbind(t, z), y, ...)`.

If none of these succeed, the function stops with an error.

For every time point (historical and future), an adaptive step size $h_eps = \max(\text{zero}^{(1/3)} * |t|, \text{zero})$ (with $\text{zero} = 1e-4$) is used to perturb `t` up and down while holding `z` fixed. The fitted model is used to predict the response at $t + h_eps$ and $t - h_eps$, and the central-difference slope $(fx_plus - fx_minus) / (2 * h_eps)$ is returned for each time point.

A fixed random seed (123) is set internally so that the random covariate `z` — and therefore the resulting slopes — are reproducible across calls.

Value

A numeric vector of length $\text{length}(y) + h$ containing the estimated local slope (numerical derivative of the fitted model's predictions with respect to time) at each historical and future time point, in chronological order.

Examples

```
## Not run:
set.seed(1)
y <- cumsum(rnorm(50))
slopes <- estimate_theta_slope(
  fit_func = lm,
  predict_func = predict,
  y = y,
  h = 10
)
plot(slopes, type = "l")

## End(Not run)
```

example_2level_forecast

Example usage

Description

Example usage

Usage

```
example_2level_forecast()
```

fit_func

Fit univariate time series using caret ML model (for use with dynrmf)

Description

Fit univariate time series using caret ML model (for use with dynrmf)

Usage

```
fit_func(
  x,
  y,
  method = "ranger",
  initial_window = 10L,
  horizon = 10L,
  fixed_window = FALSE,
  tune_length = 5,
  summary_function = NULL,
  verbose = TRUE
)
```

Arguments

x	A matrix of predictors
y	A vector of responses
method	The caret method to use for fitting the model
initial_window	The initial window size
horizon	The forecast horizon
fixed_window	Whether to use a fixed window size
tune_length	Length of the tuning grid
summary_function	A summary function
verbose	Whether to print the model summary

Value

A model object

fitdistr_ahead

Fit a Univariate Distribution to Data via Maximum Likelihood

Description

A lightweight, `stats::fitdistr`-inspired routine that fits the parameters of a one-dimensional distribution to a numeric vector by maximum likelihood, using `stats::nlminb` as the optimizer. Common distributions can be specified by name (in which case sensible defaults and closed-form/starting-value shortcuts are used where available), or an arbitrary density function can be supplied directly.

Usage

```
fitdistr_ahead(x, densfun, start = NULL, seed = 123, ...)
```

Arguments

<code>x</code>	A non-empty numeric vector of observations to fit. Must not contain NA, NaN, or infinite values.
<code>densfun</code>	Either a character string naming one of the supported distributions (see Details), or a density function taking <code>x</code> as its first argument, followed by the distribution's parameters, and optionally a <code>log</code> argument.
<code>start</code>	Optional named list of starting values for the distribution's parameters, passed on to the density function by name. Required (and must be a named list) unless <code>densfun</code> is "normal" or "t", for which defaults are supplied automatically. Ignored (and an error is raised if supplied) for "normal".
<code>seed</code>	Integer; random seed. Currently accepted for interface compatibility but not used internally. Defaults to 123.
<code>...</code>	Additional arguments passed on to <code>densfun</code> (e.g. fixed parameters not being estimated, such as lower/upper bounds for some distributions, which are excluded when checking for user-supplied starting parameters).

Details

If `densfun` is a function, it is called internally as `densfun(x, <named parameters from start/parm>, ..., log = )`, with matching done by name — so no reordering of `densfun`'s formals is required. The negative log-likelihood is minimized with `stats::nlminb`, using `densfun`'s own `log = TRUE` density when it exposes a `log` argument (for numerical stability), and falling back to summing `log(density)` otherwise.

If `densfun` is a character string, it is matched (case-insensitively) against a fixed set of supported distributions: "beta", "cauchy", "chi-squared", "exponential", "f", "gamma", "geometric", "log-normal" / "lognormal", "logistic", "negative binomial", "normal", "poisson", "t", and "weibull". An unrecognized name raises an error.

Two distributions receive special handling:

- **normal**: the mean and (bias-corrected) standard deviation are computed in closed form (no optimization is performed), and supplying `start` for this distribution is not supported and raises an error.
- **t**: if `start` is not supplied, it defaults to `list(m = median(x), s = IQR(x) / 2, df = 10)`, with any of `m`, `s`, `df` passed via `...` removed from the default starting list. The Student-t density is evaluated via an internal location-scale helper (`mydt`) built on `stats::dt`.

For all other distributions, `start` must be supplied as a named list of initial parameter values matching `densfun`'s parameters.

Note: the `seed` argument is accepted but is not currently used inside the function body (the optimization is deterministic given `start`).

Value

For `densfun = "normal"`, an object of class `"fitdistr"`, a list with components:

estimate Named numeric vector of estimated mean and standard deviation.

sd Named numeric vector of standard errors of the estimates.

vcov Variance-covariance matrix of the estimates.

n Number of observations.

loglik Log-likelihood at the estimate.

For all other distributions, a plain list with components:

estimate Numeric vector of estimated parameters, as returned by `stats::nlminb`.

convergence Convergence code returned by `stats::nlminb` (0 indicates success).

objective Value of the (negative log-likelihood) objective function at the solution.

Examples

```
## Not run:
set.seed(1)
x <- rnorm(100, mean = 5, sd = 2)

# Named distribution, closed-form normal fit
fitdistr_ahead(x, "normal")

# Named distribution with default starting values (t distribution)
fitdistr_ahead(x, "t")

# Custom density function with explicit starting values
fitdistr_ahead(x, stats::dgamma, start = list(shape = 1, rate = 1))

## End(Not run)
```

fitforecast

*Fit and forecast for benchmarking purposes***Description**

Fit and forecast for benchmarking purposes

Usage

```
fitforecast(
  y,
  h = NULL,
  pct_train = 0.9,
  pct_calibration = 0.5,
  method = c("thetaf", "arima", "ets", "te", "tbats", "tslm", "dynrmf", "ridge2f",
    "naive", "snaive", "arimagarch"),
  level = 95,
  B = 1000L,
  seed = 17223L,
  graph = TRUE,
  conformalize = FALSE,
  type_calibration = c("splitconformal", "cv1", "loocv"),
  gap = 3L,
  agg = c("mean", "median"),
  vol = c("constant", "garch"),
  type_sim = c("kde", "surrogate", "bootstrap", "meboot"),
  ...
)
```

Arguments

y	A univariate time series of class ts
h	Forecasting horizon (default is NULL, in that case, pct_train and pct_calibration are used)
pct_train	Percentage of data in the training set, when h is NULL
pct_calibration	Percentage of data in the calibration set for conformal prediction
method	For now "thetaf" (default), "arima", "ets", "tbats", "tslm", "dynrmf" (from ahead), "ridge2f" (from ahead), "naive", "snaive"
level	Confidence level for prediction intervals in %, default is 95
B	Number of bootstrap replications or number of simulations (yes, 'B' is unfortunate)
seed	Reproducibility seed
graph	Plot or not?

conformalize	Calibrate or not?
type_calibration	"splitconformal" (default conformal method), "cv1" (do not use), "loocv" (do not use)
gap	length of training set for loocv conformal (do not use)
agg	"mean" or "median" (aggregation method)
vol	"constant" or "garch" (type of volatility modeling for calibrated residuals)
type_sim	"kde", "surrogate", "bootstrap" (type of simulation for calibrated residuals)
...	additional parameters

Value

an object of class 'forecast' with additional information

Examples

```
par(mfrow=c(2, 2))
obj1 <- ahead::fitforecast(AirPassengers)
obj2 <- ahead::fitforecast(AirPassengers, conformalize = TRUE)
plot(AirPassengers)
plot(obj1)
obj2$plot()
obj2$plot("simulations")
```

generate_synthetic_ts *Generate synthetic time series via model-based residual bootstrap*

Description

Generate synthetic time series via model-based residual bootstrap

Usage

```
generate_synthetic_ts(
  y,
  model_type = "auto",
  model_func = NULL,
  formula = NULL,
  order = NULL,
  seasonal = list(order = c(0, 0, 0), period = NA),
  length_out = NULL,
  n_sim = 1000,
  seed = NULL,
  ...
)
```

Arguments

y	Time series data (ts or mts object)
model_type	One of "auto", "auto.arima", "tslm", "ets", "arima", "custom"
model_func	Custom model function (required if model_type = "custom")
formula	Formula for tslm (default: trend + season)
order	ARIMA order (p,d,q) for model_type = "arima"
seasonal	List with order and period for seasonal ARIMA
length_out	Length of synthetic series to generate (default = length of y)
n_sim	Number of synthetic series to generate
seed	Random seed for reproducibility (optional)
...	Extra arguments passed to the modelling function

Value

A list with synthetic series and model information

genericforecast	<i>Generic Forecasting Function (Unified interface)</i>
-----------------	---

Description

This function allows to call any function "of class forecast" in a unified way.

Usage

```
genericforecast(FUN, y, h, level = 95, ...)
```

Arguments

FUN	A forecasting function.
y	A time series (ts object or vector).
h	Forecasting horizon.
level	The confidence level.
...	Additional arguments to be passed to the forecasting function.

Value

An object of class forecast.

Examples

```
y <- fdeaths
h <- 25L
plot(ahead::genericforecast(FUN=forecast::thetaf, y, h))
plot(ahead::genericforecast(FUN=ahead::dynrmf, y, h))
# plot(ahead::genericforecast(FUN=forecast::tbats, y=y, h=h, use.box.cox = TRUE, use.trend=FALSE))
# plot(ahead::genericforecast(FUN=forecast::auto.arima, y, h))
```

`geterror`*Get error metrics*

Description

This function calculates various error metrics for a given prediction object.

Usage

```
geterror(obj, actual, level = 95)
```

Arguments

<code>obj</code>	A prediction object containing mean predictions and intervals.
<code>actual</code>	A numeric vector of actual values.
<code>level</code>	The confidence level for the prediction intervals.

Value

A numeric vector containing the error metrics.

Examples

```
obj <- list(mean = 10, lower = 8, upper = 12)
actual <- 11
geterror(obj, actual)
```

getreturns	<i>Calculate returns or log-returns for multivariate time series</i>
------------	--

Description

Calculate returns or log-returns for multivariate time series

Usage

```
getreturns(x, type = c("basic", "log"))
```

Arguments

x	Multivariate time series
type	Type of return: basic return ("basic") or log-return ("log")

Value

The returns

Examples

```
returns <- getreturns(EuStockMarkets)
log_returns <- getreturns(EuStockMarkets,
                          type = "log")

par(mfrow=c(1, 3))
matplot(EuStockMarkets, type = 'l', main = "Closing Prices of \n European stocks (1991-1998)",
        xlab = "time")
matplot(returns, type = 'l', main = "Returns", xlab = "time")
matplot(log_returns, type = 'l', main = "Log-returns", xlab = "time")
```

getsimulations	<i>Obtain simulations (when relevant) from a selected time series</i>
----------------	---

Description

Obtain simulations (when relevant) from a selected time series

Usage

```
getsimulations(obj, selected_series, transpose = FALSE)
```

Arguments

obj	result from ridge2f (multivariate time series forecast with simulations)
selected_series	name of the time series selected
transpose	return a transposed time series

Examples

```
require(fpp2)

obj <- ahead::ridge2f(fpp2::insurance, h = 7,
                     type_pi = "bootstrap", B = 5)
print(getsimulations(obj, selected_series = "TV.advert"))
print(getsimulations(obj, selected_series = "Quotes"))
print(getsimulations(obj, selected_series = "TV.advert", transpose = TRUE))
print(getsimulations(obj, selected_series = "Quotes", transpose = TRUE))
```

glmthetaf

*Generalized Linear Model Theta Forecast and not only***Description**

This function implements the Theta method using a Generalized Linear Model (GLM)

Usage

```
glmthetaf(
  y,
  h = ifelse(frequency(y) > 1, 2 * frequency(y), 10),
  level = 95L,
  fit_func = stats::glm,
  predict_func = predict,
  fan = FALSE,
  x = y,
  type_pi = c("gaussian", "conformal-split", "conformal-surrogate",
             "conformal-bootstrap", "conformal-block-bootstrap", "conformal-kde",
             "conformal-fitdistr"),
  attention = TRUE,
  attention_type = c("dot_product", "scaled_dot_product", "cosine", "exponential",
                   "gaussian", "linear", "value_based", "hybrid", "parametric"),
  attention_method = c("heuristic", "historical"),
  scale_ctxt = 1,
  historical_lookback = 50L,
  max_adjustment = 0.5,
  B = 250L,
  nsim = B,
```

```
    ...  
  )
```

Arguments

y	The time series data
h	The number of periods to forecast
level	The confidence level for the forecast intervals
fit_func	The function to use for fitting the GLM (and other models)
predict_func	The function to use for predict the other models (not GLMs)
fan	Logical flag for fan plot
x	The time series data
type_pi	Type of prediction intervals
attention	Logical flag for using attention mechanism
attention_type	The type of attention mechanism to use
attention_method	The method for computing attention-based adjustments: "heuristic" (original) or "historical" (new evidence-based)
scale_ctxt	Scaling coefficient for context vector
historical_lookback	Number of periods to consider for historical matching
max_adjustment	Maximum allowed adjustment percentage (0-1)
B	Number of bootstrap replications or number of simulations (alias: nsim)
nsim	Alias for B
...	Additional arguments to pass to the fit_func

Value

A forecast object

loessf	<i>Loess forecasting</i>
--------	--------------------------

Description

Loess forecasting

Usage

```
loessf(
  y,
  h = 5,
  level = 95,
  span = 0.75,
  degree = 2,
  type_pi = c("bootstrap", "blockbootstrap", "movingblockbootstrap"),
  b = NULL,
  B = 250,
  type_aggregation = c("mean", "median"),
  seed = 123
)
```

Arguments

y	A numeric vector or time series of class <code>ts</code>
h	Forecasting horizon
level	Confidence level for prediction intervals
span	the parameter which controls the degree of smoothing
degree	the degree of the polynomials to be used, normally 1 or 2. (Degree 0 is also allowed, but see <code>stats::loess</code>)
type_pi	Type of prediction interval currently (independent) "bootstrap", (circular) "block-bootstrap", or "movingblockbootstrap"
b	block length for circular block bootstrap
B	number of bootstrap replications
type_aggregation	Type of aggregation, ONLY for bootstrapping; either "mean" or "median"
seed	reproducibility seed

Value

An object of class "forecast"; a list containing the following elements:

model	A list containing information about the fitted model
method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
x	The original time series
residuals	Residuals from the fitted model
sims	Model simulations for bootstrapping

Examples

```

par(mfrow = c(3, 1))

plot(loessf(Nile, h=20, level=95, B=10))

plot(loessf(Nile, h=20, level=95, B=10,
  type_pi = "blockbootstrap"))

plot(loessf(Nile, h=20, level=95, B=10,
  type_pi = "movingblockbootstrap"))

```

loocvridge2f

LOOCV for Ridge2 model

Description

LOOCV for Random Vector functional link network model with 2 regularization parameters

Usage

```

loocvridge2f(
  y,
  xreg = NULL,
  h = 5,
  level = 95,
  lags = 1,
  nb_hidden = 5,
  nodes_sim = c("sobol", "halton", "unif"),
  activ = c("relu", "sigmoid", "tanh", "leakyrelu", "elu", "linear"),
  a = 0.01,
  lambda_1 = 0.1,
  lambda_2 = 0.1,
  dropout = 0,
  type_forecast = c("recursive", "direct"),
  type_pi = c("gaussian", "bootstrap", "blockbootstrap", "movingblockbootstrap",
    "rvinecopula", "splitconformal"),
  block_length = NULL,
  margins = c("gaussian", "empirical", "student"),
  seed = 1,
  B = 100L,
  type_aggregation = c("mean", "median"),
  centers = NULL,
  type_clustering = c("kmeans", "hclust"),
  ym = NULL,
  cl = 1L,
  show_progress = TRUE,

```

```
    ...
)
```

Arguments

<code>y</code>	A multivariate time series of class <code>ts</code> (preferred) or a <code>matrix</code>
<code>xreg</code>	External regressors. A <code>data.frame</code> (preferred) or a <code>matrix</code>
<code>h</code>	Forecasting horizon
<code>level</code>	Confidence level for prediction intervals
<code>lags</code>	Number of lags
<code>nb_hidden</code>	Number of nodes in hidden layer
<code>nodes_sim</code>	Type of simulation for nodes in the hidden layer
<code>activ</code>	Activation function
<code>a</code>	Hyperparameter for activation function "leakyrelu", "elu"
<code>lambda_1</code>	Regularization parameter for original predictors
<code>lambda_2</code>	Regularization parameter for transformed predictors
<code>dropout</code>	dropout regularization parameter (dropping nodes in hidden layer)
<code>type_forecast</code>	Recursive or direct forecast
<code>type_pi</code>	Type of prediction interval currently "gaussian", "bootstrap", "blockbootstrap", "movingblockbootstrap", "splitconformal" (very experimental right now), "rvinecopula" (with Gaussian margins for now, Student-t coming soon)
<code>block_length</code>	Length of block for circular or moving block bootstrap
<code>margins</code>	Distribution of margins: "gaussian", "empirical", "student" (postponed or never) for <code>type_pi == "rvinecopula"</code>
<code>seed</code>	Reproducibility seed for random stuff
<code>B</code>	Number of bootstrap replications or number of simulations (yes, 'B' is unfortunate)
<code>type_aggregation</code>	Type of aggregation, ONLY for bootstrapping; either "mean" or "median"
<code>centers</code>	Number of clusters for <code>type_clustering</code>
<code>type_clustering</code>	"kmeans" (K-Means clustering) or "hclust" (Hierarchical clustering)
<code>ym</code>	Univariate time series (<code>stats::ts</code>) of yield to maturities with frequency = frequency(y) and start = tsp(y)[2] + 1 / frequency(y). Default is NULL.
<code>cl</code>	An integer; the number of clusters for parallel execution, for bootstrap
<code>show_progress</code>	A boolean; show progress bar for bootstrapping? Default is TRUE.
<code>...</code>	Additional parameters to be passed to <code>kmeans</code> or <code>hclust</code>

Value

An object of class "mtsforecast"; a list containing the following elements:

method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
sims	Model simulations for bootstrapping (basic, or block)
x	The original time series
residuals	Residuals from the fitted model
coefficients	Regression coefficients for type_pi == 'gaussian' for now

Author(s)

T. Moudiki

References

Moudiki, T., Planchet, F., & Cousin, A. (2018). Multiple time series forecasting using quasi-randomized functional link neural networks. *Risks*, 6(1), 22.

Examples

```
require(fpp2)

print(ahead::loocvridge2f(fpp2::insurance))

#foo <- function(xx) ahead::loocvridge2f(fpp2::insurance, lambda_1=10^xx[1], lambda_2=10^xx[2])
#(opt <- stats::nlminb(objective=foo, lower=c(-10,-10), upper=c(10,10), start=c(0, 0)))
#print(ahead::loocvridge2f(fpp2::insurance, lambda_1=10^opt$par[1], lambda_2=10^opt$par[2]))
```

meboot

Maximum Entropy Bootstrap for Time Series using Rcpp

Description

Generates bootstrap replicates of a time series using the maximum entropy bootstrap algorithm with Rcpp implementation for improved performance. This method is particularly useful for non-stationary time series and preserves the dependence structure of the original data.

Usage

```
meboot(
  x,
  reps = 999,
  trim = list(trim = 0.1, xmin = NULL, xmax = NULL),
  reachbnd = TRUE,
  expand.sd = TRUE,
  force.clt = TRUE,
  scl.adjustment = FALSE,
  sym = FALSE,
  colsubj,
  coldata,
  coltimes,
  ...
)
```

Arguments

x	A numeric vector or time series object to be bootstrapped.
reps	Number of bootstrap replicates to generate (default: 999).
trim	Controls tail behavior. Can be a single numeric value specifying the trim proportion (e.g., 0.10 for 10% trimming), or a list with components: trim Trim proportion for tail calculation (default: 0.10) xmin Lower bound for generated values (optional) xmax Upper bound for generated values (optional)
reachbnd	Logical indicating whether to allow generated values to reach the boundaries xmin and xmax (default: TRUE).
expand.sd	Logical indicating whether to expand the standard deviation of the ensemble (default: TRUE).
force.clt	Logical indicating whether to force the central limit theorem compliance by centering each replicate (default: TRUE).
scl.adjustment	Logical indicating whether to adjust the scale of the ensemble to match the original data's variance (default: FALSE).
sym	Logical indicating whether to force symmetry in the maximum entropy density (default: FALSE).
colsubj	Deprecated parameter from original meboot (included for compatibility).
coldata	Deprecated parameter from original meboot (included for compatibility).
coltimes	Deprecated parameter from original meboot (included for compatibility).
...	Additional arguments passed to expansion functions.

Details

The maximum entropy bootstrap algorithm generates replicates that:

- Preserve the dependence structure of the original time series

- Can handle non-stationary time series
- Satisfy the ergodic theorem and central limit theorem
- Maintain the mean and autocorrelation structure

The Rcpp implementation provides significant performance improvements over the original R implementation, especially for large datasets and many replications.

Value

A list with the following components:

x	Original time series data
ensemble	Matrix of bootstrap replicates (n x reps)
xx	Sorted original data
z	Intermediate points between sorted values
dv	Absolute differences between consecutive observations
dvtrim	Trimmed mean of differences
xmin	Lower bound used for generation
xmax	Upper bound used for generation
desintxb	Interval means satisfying mean-preserving constraint
ordxx	Ordering index of original data
kappa	Scale adjustment factor (if scl.adjustment = TRUE)

References

Vinod, H. D., & Lopez-de-Lacalle, J. (2009). Maximum entropy bootstrap for time series: The meboot R package. *Journal of Statistical Software*, 29(5), 1-19.

Examples

```
# Basic usage with a time series
set.seed(123)
x <- ts(rnorm(100), start = c(2000, 1), frequency = 12)
result <- meboot(x, reps = 1000)

# Plot first few replicates
matplot(result$ensemble[, 1:5], type = "l", lty = 1)
lines(result$x, col = "black", lwd = 2)

# With custom bounds
result_bounded <- meboot(x, reps = 100,
                        trim = list(trim = 0.1, xmin = -3, xmax = 3))

# With scale adjustment
result_scaled <- meboot(x, reps = 100, scl.adjustment = TRUE)
```

ml_forecast

*Forecasting using Machine Learning models***Description**

Forecasting using Machine Learning models

Usage

```
ml_forecast(
  y,
  h,
  level = 95,
  lags = 1,
  fit_func = ahead::ridge,
  predict_func = predict,
  xreg = NULL,
  coeffs = NULL,
  ...
)
```

Arguments

y	A numeric vector or time series of class <code>ts</code>
h	Forecasting horizon
level	Confidence level for prediction intervals
lags	Number of lags of the input time series considered in the regression
fit_func	Fitting function (Statistical/ML model). Default is Ridge regression.
predict_func	Prediction function (Statistical/ML model)
xreg	External regressor variable
coeffs	Coefficients of the fitted model. If provided, a linear combination with the coefficients is used to compute the prediction.
...	additional parameters passed to the fitting function <code>fit_func</code>

Value

An object of class 'forecast'

Examples

```
## Not run:
plot(ahead::ml_forecast(AirPassengers, h=20L))

plot(ahead::ml_forecast(AirPassengers, h=25L, lags=20L, fit_func=glmnet::cv.glmnet))
```

```

res <- ahead::ml_forecast(USAccDeaths, h=15L, lags=15L)
plot(res)

res <- ahead::ml_forecast(USAccDeaths, fit_func = glmnet::cv.glmnet, h=15L, lags=15L)
plot(res)

res <- ahead::ml_forecast(USAccDeaths, fit_func = e1071::svm, h=15L, lags=15L)
plot(res)

res <- ahead::ml_forecast(mdeaths, h=15L, lags=15L)
plot(res)

res <- ahead::ml_forecast(fdeaths, fit_func = glmnet::cv.glmnet, h=15L, lags=25L)
plot(res)

res <- ahead::ml_forecast(fdeaths, fit_func = randomForest::randomForest, h=15L, lags=15L)
plot(res)

## End(Not run)

```

mlarchf

Conformalized Forecasting using Machine Learning (and statistical) models with ARCH effects

Description

Conformalized Forecasting using Machine Learning (and statistical) models with ARCH effects

Usage

```

mlarchf(
  y,
  h = 10L,
  mean_model = forecast::auto.arima,
  model_residuals = forecast::meanf,
  fit_func = ahead::ridge,
  predict_func = predict,
  conformal = FALSE,
  lags_vol = 10L,
  type_pi = c("surrogate", "bootstrap", "kde"),
  type_sim_conformalize = c("surrogate", "block-bootstrap", "bootstrap", "kde",
    "fitdistr"),
  ml_method = NULL,
  level = 95,
  B = 250L,
  ml = TRUE,
  stat_model = NULL,
  n_clusters = 0,

```

```

    clustering_dist = "euclidean",
    clustering_method = "kmeans",
    ...
)

```

Arguments

<code>y</code>	A numeric vector or time series of class <code>ts</code>
<code>h</code>	Forecasting horizon
<code>mean_model</code>	Function to fit the mean model (default: <code>forecast::auto.arima</code>)
<code>model_residuals</code>	Function to model the residuals (default: <code>forecast::thetaf</code>)
<code>fit_func</code>	Fitting function for the variance model (default: <code>ahead::ridge</code>)
<code>predict_func</code>	Prediction function for the variance model (default: <code>predict</code>)
<code>conformal</code>	A boolean; either use conformal prediction or not
<code>lags_vol</code>	Number of lags for squared residuals regression
<code>type_pi</code>	Type of prediction interval ("kde", "surrogate", or "bootstrap") for volatility modeling
<code>type_sim_conformalize</code>	Type of simulation for conformalization of standardized residuals ("block-bootstrap", "surrogate", "kde", "bootstrap", or "fitdistr")
<code>ml_method</code>	caret package Machine learning method to use, if <code>fit_func</code> and <code>predict_func</code> aren't provided. If NULL, uses <code>fit_func</code> and <code>predict_func</code> . See <code>unique(caret::modelLookup())\$model</code>
<code>level</code>	Confidence level for prediction intervals
<code>B</code>	Number of bootstrap replications or simulations
<code>ml</code>	If TRUE, <code>fit_func</code> and <code>predict_func</code> are used, otherwise a statistical model in <code>stat_model</code>
<code>stat_model</code>	A statistical model, e.g <code>forecast::thetaf</code> or <code>forecast::auto.arima</code>
<code>n_clusters</code>	Number of clusters for residuals (default is 0) for <code>ml == TRUE</code> and <code>ml_method</code> not NULL
<code>clustering_dist</code>	Only if <code>n_clusters > 0</code> . If "euclidean", then mean square error, if "manhattan", the mean absolute error is used.
<code>clustering_method</code>	Only if <code>n_clusters > 0</code> . If "kmeans", then we have the kmeans clustering method, if "hardcl" we have the On-line Update (Hard Competitive learning) method, and if "neuralgas", we have the Neural Gas (Soft Competitive learning) method. Abbreviations of the method names are accepted.
<code>...</code>	Additional parameters to be passed to <code>stat_model</code>

Value

A forecast object containing predictions and prediction intervals

Examples

```

y <- fpp2::goog200

par(mfrow=c(2, 2))

(obj_ridge <- ahead::mlarchf(y, h=20L, B=500L, conformal=TRUE))
plot(obj_ridge)
print(mean(obj_ridge$upper - obj_ridge$lower))

(obj_ridge <- ahead::mlarchf(y, h=20L, B=500L, conformal=FALSE))
plot(obj_ridge)
print(mean(obj_ridge$upper - obj_ridge$lower))

(obj_ridge <- ahead::mlarchf(y, h=20L, B=500L, conformal=TRUE,
stat_model=forecast::thetaf, ml = FALSE))
plot(obj_ridge)
print(mean(obj_ridge$upper - obj_ridge$lower))

(obj_ridge <- ahead::mlarchf(y, h=20L, B=500L, conformal=FALSE,
stat_model=forecast::thetaf, ml = FALSE))
plot(obj_ridge)
print(mean(obj_ridge$upper - obj_ridge$lower))

par(mfrow=c(1, 1))

```

mlf

*Conformalized Forecasting using Machine Learning models***Description**

Conformalized Forecasting using Machine Learning models

Usage

```

mlf(
  y,
  h = 5,
  level = 95,
  lags = 15L,
  fit_func = ahead::ridge,
  predict_func = predict,
  stack = FALSE,
  stacking_models = NULL,
  coeffs = NULL,
  type_pi = c("surrogate", "bootstrap", "kde"),
  B = 250L,
  agg = c("mean", "median"),

```

```

    seed = 123,
    show_progress = TRUE,
    ...
)

```

Arguments

y	A numeric vector or time series of class <code>ts</code>
h	Forecasting horizon
level	Confidence level for prediction intervals
lags	Number of lags of the input time series considered in the regression
fit_func	Fitting function (Statistical/ML model). Default is Ridge regression.
predict_func	Prediction function (Statistical/ML model)
stack	Boolean, use stacking regression or not
stacking_models	A list of <code>fit_funcs</code> and <code>predict_funcs</code> for an ensemble of stacked models (you should set <code>stack=TRUE</code>)
coeffs	Coefficients of the fitted model. If provided, a linear combination with the coefficients is used to compute the prediction.
type_pi	Type of prediction interval
B	Number of bootstrap replications or number of simulations
agg	"mean" or "median" (aggregation method)
seed	reproducibility seed
show_progress	show progress bar for stacking, if <code>stacking_models</code> is not NULL
...	additional parameters passed to the fitting function <code>fit_func</code>

Value

An object of class 'forecast'

plot.forecomb_res	<i>Plot results from forecast combination model</i>
-------------------	---

Description

Produces plots for the results of a forecast combination method. Either an actual vs. fitted plot (which = 1) or a barplot of the combination weights (which = 2).

Usage

```

## S3 method for class 'forecomb_res'
plot(x, which = 1, ...)

```

Arguments

x	An object of class 'foreccomb_res'.
which	Type of plot: 1 = actual vs. fitted, 2 = combination weights.
...	Other arguments passing to plot.default .

Value

A plot for the foreccomb_res class.

Author(s)

adapted from Christoph E. Weiss and Gernot R. Roetzer (ForecastComb)

plot.mtsforecast	<i>Plot multivariate time series forecast or residuals</i>
------------------	--

Description

Plot multivariate time series forecast or residuals

Usage

```
## S3 method for class 'mtsforecast'
plot(x, selected_series, type = c("pi", "dist", "sims"), level = 95, ...)
```

Arguments

x	result from basicf, ridge2f or varf (multivariate time series forecast)
selected_series	name of the time series selected for plotting
type	"pi": basic prediction intervals; "dist": a distribution of predictions; "sims": the simulations
level	confidence levels for prediction intervals
...	additional parameters to be passed to plot or matplot

Examples

```
require(fpp2)

fit_obj_VAR <- ahead::varf(fpp2::insurance, lags = 2,
h = 10, level = 95)

fit_obj_ridge2 <- ahead::ridge2f(fpp2::insurance, lags = 2,
h = 10, level = 95)

par(mfrow=c(2, 2))
```

```

plot(fit_obj_VAR, "Quotes")
plot(fit_obj_VAR, "TV.advert")
plot(fit_obj_ridge2, "Quotes")
plot(fit_obj_ridge2, "TV.advert")

obj <- ahead::ridge2f(fpp2::insurance, h = 10, type_pi = "blockbootstrap",
  block_length=5, B = 10)
par(mfrow=c(1, 2))
plot(obj, selected_series = "Quotes", type = "sims",
  main = "Predictive simulation for Quotes")
plot(obj, selected_series = "TV.advert", type = "sims",
  main = "Predictive simulation for TV.advert")

par(mfrow=c(1, 2))
plot(obj, selected_series = "Quotes", type = "dist",
  main = "Predictive simulation for Quotes")
plot(obj, selected_series = "TV.advert", type = "dist",
  main = "Predictive simulation for TV.advert")

```

plot.synthetic_ts

Plot method for synthetic_ts objects

Description

Plot method for synthetic_ts objects

Usage

```

## S3 method for class 'synthetic_ts'
plot(x, type = "series", which_sim = 1, series_index = 1, ...)

```

Arguments

x	synthetic_ts object
type	Type of plot: "series", "residuals", "acf", "density", "qq"
which_sim	Which simulation to plot (for type = "series")
series_index	Which series to plot (for multivariate data)
...	Additional arguments passed to plot functions

`plot_dynrmf_sensitivity`*Plot First-Order Sensitivity Effects*

Description

This function plots the computed sensitivity effects over the forecast horizon.

Usage

```
plot_dynrmf_sensitivity(  
  sensitivity_results,  
  title = "First-Order Sensitivity Effect",  
  y_label = "Effect (DeltaY / DeltaX)",  
  x_label = "Forecast Horizon"  
)
```

Arguments

<code>sensitivity_results</code>	Output list from <code>dynrmf_sensi</code> .
<code>title</code>	Character. Plot title.
<code>y_label</code>	Character. Label for y-axis.
<code>x_label</code>	Character. Label for x-axis (default "Forecast Horizon").

Value

ggplot2 object displaying the sensitivity effects.

Examples

```
## Not run:  
plot1 <- plot_dynrmf_sensitivity(res,  
                                title = "Sensitivity of Consumption to Income",  
                                y_label = "Effect (DeltaConsumption / DeltaIncome)")  
print(plot1)  
  
## End(Not run)
```

plot_dynrmf_shap_waterfall

Waterfall plot for a dynrmf_shap object

Description

Aggregates SHAP values across horizons and draws a horizontal waterfall.

Usage

```
plot_dynrmf_shap_waterfall(
  shap,
  title = "SHAP Waterfall",
  agg_fun = sum,
  colors = list(pos = "#4575b4", neg = "#d73027", base = "#878787")
)
```

Arguments

shap	A "dynrmf_shap" object from dynrmf_shap()
title	Plot title string
agg_fun	How to aggregate across horizons: sum (default) or mean
colors	Named list with elements "pos", "neg", "base"

Value

A ggplot2 object

plot_hts_forecast

Plot hierarchical time series forecast

Description

Plots a historical series (either the total series or a chosen bottom-level series of a hierarchical time series) alongside its point forecast, mean forecast (from simulations), and prediction interval, as produced by a hierarchical forecasting routine. A vertical line marks the train/test split point.

Usage

```
plot_hts_forecast(result, hts, series_type = "total", series_idx = 1)
```

Arguments

result	a list containing forecast results, with elements point_forecasts (list with total and bottom components), mean_forecasts (list with total and bottom components), prediction_intervals (list with total, a matrix with columns "lower"/"upper", and bottom, a list with lower and upper matrices, one column per bottom series), and split_info (list with split_point and alpha)
hts	a hierarchical time series object containing total (the historical total series) and bts (a matrix of historical bottom-level series, one column per series)
series_type	character string; either "total" to plot the total series, or any other value to plot a bottom-level series (default "total")
series_idx	integer; index of the bottom-level series to plot when series_type != "total" (default 1)

Value

No return value; called for the side effect of producing a plot

Author(s)

T. Moudiki

Examples

```
# plot_hts_forecast(result, hts, series_type = "total")
# plot_hts_forecast(result, hts, series_type = "bottom", series_idx = 2)
```

plot_simulations	<i>Plot simulation paths for a hierarchical time series forecast</i>
------------------	--

Description

Plots a subset of simulated forecast paths (either for the total series or a chosen bottom-level series) over the forecast horizon, along with the mean simulated path and a prediction interval derived from the empirical quantiles of the simulations.

Usage

```
plot_simulations(
  result,
  series_type = "total",
  series_idx = 1,
  n_paths = 50,
  main = NULL
)
```

Arguments

result	a list containing forecast results, with elements simulations (list with total, an $h \times n_{sim}$ matrix, and bottom, an $h \times n_{series} \times n_{sim}$ array) and split_info (list with alpha, the significance level used for the prediction interval)
series_type	character string; either "total" to plot simulations of the total series, or any other value to plot simulations of a bottom-level series (default "total")
series_idx	integer; index of the bottom-level series to plot when series_type != "total" (default 1)
n_paths	integer; maximum number of individual simulation paths to display (default 50)
main	character string; plot title. If NULL (the default), a title is generated automatically based on series_type and series_idx

Value

No return value; called for the side effect of producing a plot

Author(s)

T. Moudiki

Examples

```
# plot_simulations(result, series_type = "total", n_paths = 50)
# plot_simulations(result, series_type = "bottom", series_idx = 2)
```

predict.forecomb_res *Prediction function for Forecast Combinations*

Description

prediction method for class 'forecomb_res'. Uses the previously created forecast combination result to predict the combination for a newly provided prediction dataset.

Usage

```
## S3 method for class 'forecomb_res'
predict(object, newpreds, newobs = NULL, simplify = TRUE, byrow = FALSE, ...)
```

Arguments

object	An object of class 'forecomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.
--------	---

newpreds	A matrix or multivariate time series; contains individual model forecasts if a test set is used (optional). Does not require specification of newobs – in the case in which a forecaster only wants to train the forecast combination method with a training set and apply it to future individual model forecasts, only newpreds is required, not newobs.
newobs	A vector or univariate time series; contains ‘actual values’ if a test set is used (optional).
simplify	logical. The default (TRUE) returns the predictions separately. If set to (FALSE) the predictions are incorporated into the foreccomb_res object, that is, the object is equal to the one that would have been obtained, if the new prediction set would have been provided when the forecast combination method was trained originally.
byrow	logical. The default (FALSE) assumes that each column of the forecast matrices (prediction_matrix and – if specified – newpreds) contains forecasts from one forecast model; if each row of the matrices contains forecasts from one forecast model, set to TRUE.
...	potential further arguments (require by generic)

Author(s)

Adapted from Christoph E. Weiss and Gernot R. Roetzer (ForecastComb)

See Also

[foreccomb](#),

predict_func	<i>Predict univariate time series using caret ML model(for use with dynrmf)</i>
--------------	---

Description

Predict univariate time series using caret ML model(for use with dynrmf)

Usage

```
predict_func(obj, newx)
```

Arguments

obj	Fitted object
newx	Test set data

```
print.summary.synthetic_ts
```

Print method for summary.synthetic_ts objects

Description

Print method for summary.synthetic_ts objects

Usage

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

Arguments

x	summary.synthetic_ts object
...	Additional arguments

removenas	<i>Remove Missing Values from a Time Series via Linear Interpolation</i>
-----------	--

Description

Replaces NA values in a ts object with linearly interpolated values, using the non-missing observations as anchor points. The result preserves the original time series' start time and frequency.

Usage

```
removenas(y)
```

Arguments

y	A univariate time series object of class ts. An error is raised if y is not a ts object.
---	--

Details

If y contains no missing values, it is returned unchanged. Otherwise, stats::approx is used to linearly interpolate the missing entries based on the time index (stats::time(y)) and the non-missing values, evaluated at every time point in y (so any NA at the very start or end of the series may itself remain NA, per the default behavior of approx, since there are no bracketing points to interpolate from). The interpolated values are then reassembled into a new ts object with the same start and frequency as the input.

Value

A `ts` object of the same length, start, and frequency as `y`, with internal NA values replaced by linearly interpolated values. If `y` has no missing values, `y` is returned as-is.

Examples

```
## Not run:
y <- ts(c(1, 2, NA, 4, 5, NA, 7), start = c(2020, 1), frequency = 12)
removenas(y)

## End(Not run)
```

rfitdistr

Random generation from a fitted distribution

Description

Fits a parametric distribution to a (standardized) input vector using `misc::fit_param_dist`, and generates random draws from the fitted distribution. Useful for simulating residuals or innovations whose distribution is estimated from data rather than assumed.

Usage

```
rfitdistr(x, n = length(x), p = 1)
```

Arguments

<code>x</code>	a numeric vector used to fit the distribution (the vector is centered and scaled internally before fitting)
<code>n</code>	number of observations to simulate per column/path (default <code>length(x)</code>)
<code>p</code>	number of columns/paths to simulate; if <code>p <= 1</code> , a vector of length <code>n</code> is returned, otherwise a matrix of dimension <code>n x p</code> is returned (default 1)

Value

If `p <= 1`, a numeric vector of length `n` containing simulated values. If `p > 1`, a numeric matrix with `n` rows and `p` columns containing simulated values.

Author(s)

T. Moudiki

Examples

```
x <- rnorm(100)
res <- rfitdistr(x, n = 10, p = 5)
```

rgaussiandens

Simulate Gaussian Kernel Density

Description

Generates samples from a Gaussian kernel density estimate of a numeric vector.

Usage

```
rgaussiandens(x, n = length(x), p = 1, seed = 32124)
```

Arguments

x	Numeric vector to estimate density from.
n	Integer. Number of samples per replicate (default: length of x).
p	Integer. Number of replicates (default: 1).
seed	Integer. Random seed for reproducibility (default: 123).

Value

A vector or matrix of samples from the estimated density.

Examples

```
x <- rnorm(10)
rgaussiandens(x, n = 10, p = 3)
```

ridge

Fit Ridge Regression Across a Grid of Penalty Values

Description

Fits ridge regression (adapted from `MASS::lm.ridge`) of y on x for one or several ridge penalty (`lambda`) values simultaneously, using an SVD-based solution path. Predictors are centered and scaled internally (columns with zero or undefined standard deviation are left unscaled), and both generalized cross-validation (GCV) and BIC are computed for each `lambda` to help with penalty selection.

Usage

```
ridge(x, y, lambda = 10^seq(-10, 10, length.out = 100))
```

Arguments

<code>x</code>	A numeric matrix (or vector/data frame coercible to a matrix) of predictors, with observations in rows and variables in columns.
<code>y</code>	A numeric response vector, of the same length as <code>nrow(x)</code> .
<code>lambda</code>	A numeric vector of ridge penalty values to fit. Defaults to a grid of 100 values log-spaced between $1e-10$ and $1e10$.

Details

The predictor matrix `x` is coerced to a matrix (as a single column if it has only one column or one row) and centered/scaled by column means and standard deviations; columns with zero or NA standard deviation are assigned a scale of 1 to avoid division by zero. The response `y` is mean-centered. Ridge coefficients for the full `lambda` grid are then computed in one pass via the singular value decomposition of the scaled, centered design matrix (`La.svd`), following the same approach as `MASS::lm.ridge`.

For each value of `lambda`, the function also computes:

- fitted values (on the original scale of `y`);
- residuals;
- the generalized cross-validation (GCV) score, used to select `best_lam` as the `lambda` minimizing GCV;
- the Bayesian information criterion (BIC).

When `length(lambda) > 1`, `coef` and `fitted_values` are returned as matrices with one column per `lambda` (column names set to the corresponding `lambda` values); when a single `lambda` is supplied, they are returned as vectors.

Value

An object of class `"ridge"`, a list with components:

coef Ridge regression coefficients; a vector if a single `lambda` was supplied, otherwise a matrix with one column per `lambda`.

ym Mean of the response `y`, used for centering.

xm Column means of `x`, used for centering.

xsd Column standard deviations of `x` used for scaling (zero/NA values replaced by 1).

lambda The `lambda` grid used.

best_lam The value of `lambda` minimizing the GCV score.

fitted_values Fitted values on the original scale of `y`; a vector or a matrix with one column per `lambda`.

residuals Residuals (centered `y` minus centered fitted values); a vector or matrix matching `fitted_values`.

GCV Generalized cross-validation score for each `lambda`.

BIC Bayesian information criterion for each `lambda`.

x The (unscaled) predictor matrix used for fitting.

y The original response vector.

Examples

```
## Not run:
set.seed(1)
x <- matrix(rnorm(100 * 5), ncol = 5)
y <- x %*% c(1, 0, -1, 0, 2) + rnorm(100)
fit <- ridge(x, y)
fit$best_lam
plot(fit$lambda, fit$GCV, log = "x", type = "l")

## End(Not run)
```

ridge2f

Ridge2 model

Description

Random Vector functional link network model with 2 regularization parameters

Usage

```
ridge2f(
  y,
  h = 5,
  level = 95,
  xreg = NULL,
  lags = 1,
  nb_hidden = 5,
  nodes_sim = c("sobol", "halton", "unif"),
  activ = c("relu", "sigmoid", "tanh", "leakyrelu", "elu", "linear"),
  a = 0.01,
  lambda_1 = 0.1,
  lambda_2 = 0.1,
  dropout = 0,
  type_forecast = c("recursive", "direct"),
  type_pi = c("gaussian", "bootstrap", "blockbootstrap", "movingblockbootstrap",
    "rvinecopula", "conformal-split", "conformal-bootstrap", "conformal-block-bootstrap"),
  block_length = NULL,
  margins = c("gaussian", "empirical"),
  seed = 1,
  B = 100L,
  type_aggregation = c("mean", "median"),
  centers = NULL,
  type_clustering = c("kmeans", "hclust"),
  ym = NULL,
  cl = 1L,
  show_progress = TRUE,
```

```
    ...
)
```

Arguments

<code>y</code>	A univariate or multivariate time series of class <code>ts</code> (preferred) or a <code>matrix</code>
<code>h</code>	Forecasting horizon
<code>level</code>	Confidence level for prediction intervals
<code>xreg</code>	External regressors. A <code>data.frame</code> (preferred) or a <code>matrix</code>
<code>lags</code>	Number of lags
<code>nb_hidden</code>	Number of nodes in hidden layer
<code>nodes_sim</code>	Type of simulation for nodes in the hidden layer
<code>activ</code>	Activation function
<code>a</code>	Hyperparameter for activation function "leakyrelu", "elu"
<code>lambda_1</code>	Regularization parameter for original predictors
<code>lambda_2</code>	Regularization parameter for transformed predictors
<code>dropout</code>	dropout regularization parameter (dropping nodes in hidden layer)
<code>type_forecast</code>	Recursive or direct forecast
<code>type_pi</code>	Type of prediction interval currently "gaussian", "bootstrap", "blockbootstrap", "movingblockbootstrap", "rvinecopula" (with Gaussian and empirical margins for now) "conformal-split", "conformal-bootstrap", "conformal-block-bootstrap"
<code>block_length</code>	Length of block for circular or moving block bootstrap
<code>margins</code>	Distribution of margins: "gaussian", "empirical" for <code>type_pi == "rvinecopula"</code>
<code>seed</code>	Reproducibility seed for random stuff
<code>B</code>	Number of bootstrap replications or number of simulations (yes, 'B' is unfortunate)
<code>type_aggregation</code>	Type of aggregation, ONLY for bootstrapping; either "mean" or "median"
<code>centers</code>	Number of clusters for <code>type_clustering</code>
<code>type_clustering</code>	"kmeans" (K-Means clustering) or "hclust" (Hierarchical clustering)
<code>ym</code>	Univariate time series (<code>stats::ts</code>) of yield to maturities with frequency = frequency(y) and start = tsp(y)[2] + 1 / frequency(y). Default is NULL.
<code>cl</code>	An integer; the number of clusters for parallel execution, for bootstrap
<code>show_progress</code>	A boolean; show progress bar for bootstrapping? Default is TRUE.
<code>...</code>	Additional parameters to be passed to <code>kmeans</code> or <code>hclust</code>

Value

An object of class "mtsforecast"; a list containing the following elements:

method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
sims	Model simulations for bootstrapping (basic, or block)
x	The original time series
residuals	Residuals from the fitted model
coefficients	Regression coefficients for type_pi == 'gaussian' for now

Author(s)

T. Moudiki

References

Moudiki, T., Planchet, F., & Cousin, A. (2018). Multiple time series forecasting using quasi-randomized functional link neural networks. *Risks*, 6(1), 22.

Examples

```
require(fpp2)

print(ahead::ridge2f(fpp2::insurance)$mean)

res <- ahead::ridge2f(fpp2::insurance, h=10, lags=2)
par(mfrow=c(1, 2))
plot(res, "Quotes")
plot(res, "TV.advert")

# include a trend (just for the example)
xreg <- as.numeric(time(fpp2::insurance))
res2 <- ahead::ridge2f(fpp2::insurance, xreg=xreg,
h=10, lags=2)
par(mfrow=c(1, 2))
plot(res2, "Quotes")
plot(res2, "TV.advert")

# block bootstrap
xreg <- as.numeric(time(fpp2::insurance))
res3 <- ahead::ridge2f(fpp2::insurance, xreg=xreg,
                      h=10, lags=1L, type_pi = "bootstrap", B=10)
res5 <- ahead::ridge2f(fpp2::insurance, xreg=xreg,
                      h=10, lags=1L, type_pi = "blockbootstrap", B=10,
                      block_length = 4)
```

```

print(res3$sims[[2]])
print(res5$sims[[2]])

par(mfrow=c(2, 2))
plot(res3, "Quotes")
plot(res3, "TV.advert")
plot(res5, "Quotes")
plot(res5, "TV.advert")

# moving block bootstrap
xreg <- as.numeric(time(fpp2::insurance))
res6 <- ahead::ridge2f(fpp2::insurance, xreg=xreg,
                      h=10, lags=1L,
                      type_pi = "movingblockbootstrap", B=10,
                      block_length = 4)

print(res6$sims[[2]])

par(mfrow=c(1, 2))
plot(res6, "Quotes")
plot(res6, "TV.advert")

```

rmultivariate

Simulate multivariate data

Description

Simulate multivariate data

Usage

```

rmultivariate(
  data,
  method = c("bootstrap", "block-bootstrap"),
  n = 100L,
  block_size = 5
)

```

Arguments

data	A numeric vector or matrix.
method	The method to use for sampling.
n	The number of samples to draw.
block_size	The size of the blocks to use for the block bootstrap.

rsurrogate	<i>Random generation of surrogate series</i>
------------	--

Description

Generates surrogate time series from an input series `x` using the phase-randomized Fourier transform (FFT) method implemented in `tseries::surrogate`. Surrogates preserve the linear (spectral) properties of the original series while randomizing its phase structure, and can be used as an alternative to parametric simulation for generating plausible future/alternative paths.

Usage

```
rsurrogate(x, n = length(x), p = 1, seed = 123)
```

Arguments

<code>x</code>	a numeric vector (the original time series) from which surrogates are generated
<code>n</code>	number of observations to return per surrogate path; must not exceed <code>length(x)</code> (default <code>length(x)</code>)
<code>p</code>	number of surrogate paths to generate (default 1)
<code>seed</code>	reproducibility seed (default 123)

Value

A numeric vector or matrix of length/dimension `n` (rows) by `p` (columns) containing the simulated surrogate series

Author(s)

T. Moudiki

Examples

```
x <- rnorm(100)
res <- rsurrogate(x, n = 10, p = 5)
```

```
sequential_conformal_hts
```

Sequential split conformal prediction for hierarchical forecasting Returns simulations at both total and bottom levels

Description

Sequential split conformal prediction for hierarchical forecasting Returns simulations at both total and bottom levels

Usage

```
sequential_conformal_hts(
  hts,
  h = 12,
  split_ratio = 0.6,
  n_sim = 1000,
  alpha = 0.1
)
```

Arguments

hts	2-level HTS object
h	Forecast horizon
split_ratio	Training/calibration split ratio
n_sim	Number of simulation samples
alpha	Significance level for prediction intervals

Value

Forecast with prediction intervals and simulations

```
simple_forecast
```

Simple ETS forecast function

Description

Simple ETS forecast function

Usage

```
simple_forecast(y, h)
```

Arguments

y	Time series vector
h	Forecast horizon

Value

Forecast vector

simulator	<i>simulate from a forecasting function</i>
-----------	---

Description

This function allows to obtain predictive simulations from any forecasting function.

Usage

```
simulator(
  FUN,
  y,
  h,
  level = 95,
  method = c("block-bootstrap", "surrogate", "kde", "bootstrap", "fitdistr", "meboot"),
  nsim = 100L,
  block_size = 5,
  seed = 123L,
  B = NULL,
  ...
)
```

Arguments

FUN	A forecasting function.
y	A time series (ts object or vector).
h	Forecasting horizon.
level	Confidence level.
method	Method to be used for the simulation of in-sample residuals.
nsim	Number of simulations.
block_size	Block size for block-bootstrap.
seed	Seed for reproducibility.
B	Alias for nsim
...	Additional arguments to be passed to the forecasting function.

Value

An object of class forecast, with a slot sims

Examples

```
y <- fdeaths
h <- 25L
obj <- simulator(FUN=forecast::thetaf, y, h); plot(obj)
# obj <- simulator(FUN=forecast::auto.arima, y, h); plot(obj)
obj <- simulator(FUN=forecast::ets, y, h); plot(obj)
```

splitts

Partition a time series object

Description

Partition a time series object

Usage

```
splitts(y, split_prob = 0.5, return_indices = FALSE)
```

Arguments

y A time series object

split_prob A float; Splitting ratio

return_indices A boolean; if TRUE, returns series' indices, otherwise, time series objects

Examples

```
ahead::splitts(ts(1:10))
```

stackridge2f

*Stacked Doubly-Constrained RVFL for Multivariate Forecasting***Description**

This function performs a two-stage stacked forecast using ridge regression. In the first stage, it generates base forecasts from the training portion of the time series. In the second stage, these forecasts are used as external regressors to produce the final forecast on the testing set.

Usage

```
stackridge2f(y, h = 5L, split_fraction = 0.5, ...)
```

Arguments

<code>y</code>	A multivariate time series object.
<code>h</code>	Integer. Forecast horizon for the second-stage (final) forecast. Defaults to 5.
<code>split_fraction</code>	Numeric between 0 and 1. Fraction of the time series used for training. The rest is used for testing and generating stacking features. Defaults to 0.5.
<code>...</code>	Additional arguments passed to <code>ahead::ridge2f</code> in both stages (e.g., <code>lags</code> , <code>lambda_1</code> , <code>lambda_2</code> , <code>nb_hidden</code> , etc.).

Details

The function works as follows:

1. Split the time series into training and testing sets according to `split_fraction`.
2. Generate base forecasts from the training set using `ahead::ridge2f`
3. Use these base forecasts as external regressors (`xreg`) to predict the testing set with a second `ahead::ridge2f` model.

This approach allows stacking of forecasts to potentially improve accuracy by leveraging the predictions of multiple first-stage models.

For multivariate base learners (e.g., `tslm` with multiple dependent variables), the function automatically extracts forecasts from each series and combines them into the feature matrix.

Value

An object returned by `ahead::ridge2f` for the stacked forecast, typically a list including mean and prediction intervals.

Examples

```
## Not run:

# Univariate example with default ridge2f base learner
result1 <- stackridge2f(fpp2::insurance, h = 10,
                        split_fraction = 0.5)

## End(Not run)
```

summary.foreccomb_res *Summary of Forecast Combination*

Description

summary method for class 'foreccomb_res'. Includes information about combination method, combination weights assigned to the individual forecast models, as well as an accuracy evaluation of the combined forecast.

Usage

```
## S3 method for class 'foreccomb_res'
summary(object, ...)

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

Arguments

object	An object of class 'foreccomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.
...	potential further arguments (require by generic)
x	An object of class 'foreccomb'. Contains training set (actual values + matrix of model forecasts) and optionally a test set.

Author(s)

Christoph E. Weiss and Gernot R. Roetzer

summary.synthetic_ts	<i>Summary method for synthetic_ts objects</i>
----------------------	--

Description

Summary method for synthetic_ts objects

Usage

```
## S3 method for class 'synthetic_ts'  
summary(object, ...)
```

Arguments

object	synthetic_ts object
...	Additional arguments

topdown_forecast	<i>Top-down forecast using historical proportions</i>
------------------	---

Description

Top-down forecast using historical proportions

Usage

```
topdown_forecast(hts, h)
```

Arguments

hts	2-level HTS object
h	Forecast horizon

Value

List with total and bottom forecasts

varf

Vector Autoregressive model (adapted from vars::VAR)

Description

Vector Autoregressive model adapted from vars::VAR (only for benchmarking)

Usage

```
varf(
  y,
  h = 5,
  level = 95,
  lags = 1,
  type_VAR = c("const", "trend", "both", "none"),
  ...
)
```

Arguments

y	A multivariate time series of class ts
h	Forecasting horizon
level	Confidence level for prediction intervals
lags	Number of lags
type_VAR	Type of deterministic regressors to include.
...	Additional parameters to be passed to vars::VAR.

Value

An object of class "mtsforecast"; a list containing the following elements:

method	The name of the forecasting method as a character string
mean	Point forecasts for the time series
lower	Lower bound for prediction interval
upper	Upper bound for prediction interval
x	The original time series
residuals	Residuals from the fitted model

Author(s)

T. Moudiki

References

Bernhard Pfaff (2008). VAR, SVAR and SVEC Models: Implementation Within R Package vars. Journal of Statistical Software 27(4). URL <http://www.jstatsoft.org/v27/i04/>.

Pfaff, B. (2008) Analysis of Integrated and Cointegrated Time Series with R. Second Edition. Springer, New York. ISBN 0-387-27960-1

Examples

```
require(fpp2)

print(varf(fpp2::insurance, lags=2, h=10))
```

Index

- * **bootstrap**
 - meboot, [42](#)
- * **models**
 - comb_GLMNET, [7](#)
 - comb_OLS, [9](#)
 - comb_Ridge, [10](#)
- * **ts**
 - ctxthetaf, [18](#)
 - meboot, [42](#)
- agnosticgarchf, [3](#)
- ahead::computeattention, [17](#)
- ahead::dynrmf, [23](#)
- ahead::ridge2f, [17](#)
- armagarchf, [4](#)
- basicf, [6](#)
- comb_GLMNET, [7](#)
- comb_OLS, [9](#)
- comb_Ridge, [10](#)
- computeattention, [12](#)
- condvolf, [13](#)
- conformalize, [15](#)
- contextridge2f, [16](#)
- create_2level_hts, [17](#)
- createtrendseason, [18](#)
- ctxthetaf, [18](#)
- direct_sampling, [20](#)
- dynrmf, [21](#)
- dynrmf_sensi, [23](#)
- dynrmf_shap, [24](#)
- eatf, [25](#)
- estimate_theta_slope, [27](#)
- example_2level_forecast, [29](#)
- fit_func, [29](#)
- fitdistr_ahead, [30](#)
- fitforecast, [32](#)
- forecast::ses(), [20](#)
- forecast::thetaf(), [20](#)
- ForecastComb::comb_LAD, [10](#)
- foreccomb, [56](#)
- generate_synthetic_ts, [33](#)
- genericforecast, [34](#)
- geterror, [35](#)
- getreturns, [36](#)
- getsimulations, [36](#)
- glmthetaf, [37](#)
- hclust, [41, 62](#)
- kmeans, [41, 62](#)
- loessf, [38](#)
- loocvridge2f, [40](#)
- meboot, [42](#)
- ml_forecast, [45](#)
- mlarchf, [46](#)
- mlf, [48](#)
- plot.default, [50](#)
- plot.foreccomb_res, [49](#)
- plot.mtsforecast, [50](#)
- plot.synthetic_ts, [51](#)
- plot_dynrmf_sensitivity, [52](#)
- plot_dynrmf_shap_waterfall, [53](#)
- plot_hts_forecast, [53](#)
- plot_simulations, [54](#)
- predict.foreccomb_res, [55](#)
- predict_func, [56](#)
- print.foreccomb_res_summary
(summary.foreccomb_res), [70](#)
- print.summary.synthetic_ts, [57](#)
- removenas, [57](#)
- rfitdistr, [58](#)
- rgaussiandens, [59](#)

ridge, [59](#)
ridge2f, [61](#)
rmultivariate, [64](#)
rsurrogate, [65](#)

sequential_conformal_hts, [66](#)
simple_forecast, [66](#)
simulator, [67](#)
splitts, [68](#)
stackridge2f, [69](#)
summary.foreccomb_res, [70](#)
summary.synthetic_ts, [71](#)

topdown_forecast, [71](#)

varf, [72](#)