Package ‘tsaux’

March 31, 2025

Type Package

Title Time Series Forecasting Auxiliary Functions

Version 1.0.0

Maintainer Alexios Galanos <alexios@4dscape.com>

Description
A suite of auxiliary functions that enhance time series estimation and forecasting, including a robust anomaly detection routine based on Chen and Liu (1993) <doi:10.2307/2290724> (imported and wrapped from the 'tsoutliers' package), utilities for managing calendar and time conversions, performance metrics to assess both point forecasts and distributional predictions, advanced simulation by allowing the generation of time series components—such as trend, seasonal, ARMA, irregular, and anomalies—in a modular fashion based on the innovations form of the state space model and a number of transformation methods including Box-Cox, Logit, 'Softplus-Logit' and Sigmoid.

Depends R (>= 4.1.0), tsmethods

Imports methods, zoo, xts, lubridate, car, Rdpack, scoringRules,
stlplus, tsoutliers, forecast, data.table

RdMacros Rdpack

License GPL-2

Encoding UTF-8

BugReports https://github.com/tsmodels/tsaux/issues

URL https://github.com/tsmodels/tsaux

RoxygenNote 7.3.2

Suggests knitr, kableExtra, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Alexios Galanos [aut, cre, cph]
(<https://orcid.org/0009-0000-9308-0457>)

Repository CRAN

Date/Publication 2025-03-31 17:30:01 UTC

Contents

`additive_outlier` *Anomaly Creation*

Description

Creates specific types of anomalies given a series.

Usage

```
additive_outlier(y, time = 1, parameter = 0.5, add = TRUE)

temporary_change(y, time = 1, parameter = 0.5, alpha = 0.7, add = TRUE)

level_shift(y, time = 1, parameter = 0.5, add = TRUE)
```

Arguments

`y`	a univariate xts object or numeric series.
`time`	the time index at which the anomaly takes place.
`parameter`	the coefficient on the anomaly (the percent of the value of y at the specified time index representing the anomaly).
`add`	whether to contaminate the series (add the anomaly to the series) else will return a matrix with the anomaly (without the effect of the parameter).
`alpha`	the AR(1) coefficient for the temporary change which determines how quickly the effect decays.

Details

These functions allow the generation of anomalies and may be chained together.

Value

Either the contaminated series else a matrix of the anomaly.

Author(s)

Alexios Galanos for this wrapper function.

add_anomaly *Anomaly Component*

Description

Anomaly Component

Usage

```
add_anomaly(x, ...)

## S3 method for class 'issm.component'
add_anomaly(x, time = NULL, delta = 0, ratio = 0.5, ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`time`	the numeric index of when the anomaly occurs. If NULL, a random time will be chosen.
`delta`	the autoregressive component determining the type of anomaly. A value of zero results in an additive outlier, a value of 1 in a level shift and anything in between a temporary change with a half life of -log(2)/log(delta).
`ratio`	the anomaly to series ratio at the time it occurs. For instance, a value of 1 means that the anomaly will jump by 100 percent compared to the data series.

Value

An object of class issm.component updated with the anomaly component.

add_arma *ARMA Component*

Description

ARMA Component

Usage

```
add_arma(x, ...)

## S3 method for class 'issm.component'
add_arma(x, order = c(0, 0), ar = 0, ma = 0, mu = 0, ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`order`	the ar and ma orders.
`ar`	a vector of ar coefficients.
`ma`	a vector of ma coefficients.
`mu`	the mean parameter (defaults to zero) of the ARMA process.

Value

An object of class issm.component updated with the ARMA component.

`add_custom` *Custom Component*

Description

Custom Component

Usage

```
add_custom(x, ...)

## S3 method for class 'issm.component'
add_custom(x, custom = NULL, ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`custom`	a matrix of custom components

Value

An object of class issm.component updated with the custom components.

`add_polynomial` *Polynomial Trend Component*

Description

Polynomial Trend Component

Usage

```
add_polynomial(x, ...)

## S3 method for class 'issm.component'
add_polynomial(
  x,
  order = 1,
  alpha = 0.1,
  beta = 0.01,
  phi = 1,
  l0 = 100,
  b0 = 1,
  ...
)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`order`	the order of the polynomial (min 1 and max 2).
`alpha`	the decay coefficient on the error of the level.
`beta`	the decay coefficient on the error of the slope.
`phi`	dampening parameter for the slope.
`l0`	initial level.
`b0`	initial slope.

Value

An object of class issm.component updated with the polynomial trend component.

`add_regressor` *Regressor Component*

Description

Regressor Component

Usage

```
add_regressor(x, ...)

## S3 method for class 'issm.component'
add_regressor(x, xreg = NULL, pars = NULL, ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`xreg`	a matrix of regressors.
`pars`	regressors coefficients.

Value

An object of class issm.component updated with the regressor components.

`add_seasonal` *Seasonal Trend Component*

Description

Seasonal Trend Component

Usage

```
add_seasonal(x, ...)

## S3 method for class 'issm.component'
add_seasonal(
  x,
  frequency = 12,
  gamma = 0.01,
  s0 = NULL,
  init_harmonics = frequency/2,
  normalized_seasonality = TRUE,
  init_scale = 1,
  ...
)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`frequency`	seasonal frequency.
`gamma`	the decay coefficient on the error of the seasonal component
`s0`	a vector of length frequency - 1 for the initial seasonal component.
`init_harmonics`	number of harmonics to initialize s0 when this is not provided.
`normalized_seasonality`	whether normalize the seasonal component based on the method of Roberts and McKenzie. This is applied only to a single seasonal frequency.
`init_scale`	the scaling multiplier for s0 (when this is not provided).

Value

An object of class issm.component updated with the seasonal component.

`add_transform` *Transform*

Description

Transform

Usage

```
add_transform(x, ...)

## S3 method for class 'issm.component'
add_transform(x, method = "box-cox", lambda = 1, lower = 0, upper = 1, ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`...`	additional parameters.
`method`	a valid transform.
`lambda`	the Box-Cox parameter.
`lower`	the lower bound for the transform.
`upper`	the upper bound for the transform.

Details

The inverse transform is applied to the simulated series. Valid methods are the “box-cox”, “logit”, “softplus-logit” and “sigmoid” transforms.

Value

An object of class issm.component updated with the transformation.

`auto_clean` *Automatic Cleaning of Outliers and Temporary Changes*

Description

A wrapper function for `tso` from the tsoutliers package. Takes as input a univariate xts object and returns a series decontaminated from outliers and temporary changes.

Usage

```
auto_clean(
  y,
  frequency = 1,
  lambda = NULL,
  types = c("AO", "TC"),
  stlm_opts = list(etsmodel = "AAN"),
  auto_arima_opts = list(max.p = 1, max.q = 1, d = 1, allowdrift = FALSE),
  method = c("sequential", "full"),
  ...
)
```

Arguments

`y`	a univariate xts object.
`frequency`	the frequency of the time series. If the frequency is 1 then seasonal estimation will be turned off. Will also accept multiple seasonal frequencies.
`lambda`	an optional Box Cox transformation parameter. The routines are then run on the transformed dataset.
`types`	the types of anomalies to search and decontaminate series from. Defaults to Additive outliers and temporary changes. Can be enhanced with trend breaks but not suggested for the purpose of forecasting.
`stlm_opts`	additional arguments to the stlm function.
`auto_arima_opts`	additional arguments to the auto.arima function in the tso routine.
`method`	whether to apply a sequential identification of anomalies using STL decomposition in order to only pass the stationary residuals to the tso function, else to pass the series directly to the tso package.
`...`	any additional arguments passed to the tso functions (refer to the documentation of the tsoutliers package).

Details

Calls the `auto_regressors` function to obtain the matrix of regressors and coefficients which are then used to decontaminate the series. If lambda is not NULL, the series is first transformed to perform the decontamination and then back transformed afterwards.

Value

A xts vector.

Author(s)

Alexios Galanos for this wrapper function.
Rob Hyndman for the forecast package.
Javier López-de-Lacalle for the tsoutliers package.

`auto_regressors` *Automatic Detection of Outliers, Trends Breaks and Temporary Changes*

Description

A wrapper function for function `tso` from the tsoutliers package. Takes as input a univariate xts object and returns a list with an xts object with any identified outliers, trend breaks and/or temporary changes to be used as regressors during estimation as well initial coefficients (see details).

Usage

```
auto_regressors(
  y,
  frequency = 1,
  lambda = NULL,
  forc_dates = NULL,
  sampling = NULL,
  h = 0,
  stlm_opts = list(etsmodel = "AAN"),
  auto_arima_opts = list(max.p = 1, max.q = 1, d = 1, allowdrift = FALSE),
  return_table = FALSE,
  method = c("sequential", "full"),
  ...
)
```

Arguments

`y`	a univariate xts object.
`frequency`	the frequency of the time series. If the frequency is 1 then seasonal estimation will be turned off. Will also accept multiple seasonal frequencies.

`lambda`	an optional Box Cox transformation parameter. The routines are then run on the transformed dataset.
`forc_dates`	an optional vector of Date to be used for indexing the series when h is not NULL. If this is not provided then the sampling frequency of the series will be estimated in order to generate this.
`sampling`	the sampling frequency the series. If h>0 and forc_dates is not provided, then this is required in order to generate future time indices (valid values are days, months, hours, mins, secs etc).
`h`	an optional value for the forecast horizon (if planning to also use for prediction).
`stlm_opts`	additional arguments to the stlm function.
`auto_arima_opts`	additional arguments to the auto.arima function in the tso routine.
`return_table`	whether to return a data.table instead with the anomalies detected rather than an xts matrix with the pre-processed and ready to use anomalies.
`method`	whether to apply a sequential identification of anomalies using STL decomposition in order to only pass the stationary residuals to the tso function, else to pass the series directly to the tso package.
`...`	any additional arguments passed to the tso functions (refer to the documentation of the tsoutliers package).

Details

For generating future values of the identified outliers, the filter function is used with additive outliers having a filter value of 0, trend changes a value of 1, and temporary changes have value between 0 and 1. For the sequential method, the routine first interpolates any missing values, followed by an optional Box Cox transformation, and then elimination (and identification) of any outliers during the first pass. The cleaned series is then run through an stl filter (if any frequency is greater than 1) in order to deseasonalize the data (with multiple seasonality supported), after which the deseasonalized series is passed to the tso function where any additive outliers (AO), temporary shifts (TC) or level shift (LS) are identified. Additive outliers from this stage are added to any identified outliers from the initial stage. For each regressor, initial parameter values are returned together with the regressor matrix which should be passed to the estimation routine. This is critically important since in the absence of good parameter scaling, initial values are key to good convergence. Care should be taken with regards to any automatic Box Cox parameter estimation. In the presence of large outliers or level shifts, this is likely to be badly estimated which is why we do not allow automatic calculation of this, but instead place the burden on the user to decide what is a reasonable value (if any). If a Box Cox transformation is used in the estimation routine, then it is important to use the same lambda parameter in this function in order to get sensible results. Again, avoid automatic Box Cox calculations throughout when you suspect significant contamination of the series by outliers and breaks. For the full method, the series is directly passed to the tso function of the tsoutliers package. Finally, it should be noted that this function is still experimental, and may change in the future.

Value

A list with an xts outlier matrix (if any where identified) as well as a vector of initial parameter for use in the initialization of the optimizer.

Author(s)

Alexios Galanos for this wrapper function.
Rob Hyndman for the forecast package.
Javier Lopez-de-Lacalle for the tsoutliers package.

Examples

```
library(xts)
set.seed(200)
y = cumprod(c(100,(1+rnorm(100,0.01, 0.04))))
y = xts(y, as.Date(1:101, origin = as.Date("2000-01-01")))
yclean = y
outlier1 = rep(0, 101)
outlier1[20] = 0.35
outlier2 = rep(0, 101)
outlier2[40] = 0.25
outlier2 = as.numeric(filter(outlier2, filter = 0.6, method = "recursive"))
y = y + y*xts(outlier1, index(y))
y = y + y*xts(outlier2, index(y))
# may need some tweaking of the tso options.
x = auto_regressors(y, frequency = 1, sampling = "days", h = 20,
check.rank = TRUE, discard.cval = 4)
head(x$xreg)
tail(x$xreg)
min(which(x$xreg[,1]==1))
min(which(x$xreg[,2]==1))
#plot(as.numeric(y), type = "l", ylab = "")
#lines(as.numeric(yclean) + (x$xreg %*% x$init)[1:101], col = 2)
```

---

box_cox *Box-Cox transform specification*

Description

Creates a specification for the Box Cox transformation.

Usage

```
box_cox(lambda = NA, lower = 0, upper = 1.5, multivariate = FALSE, ...)
```

Arguments

`lambda`	the power parameters. If NA then it will automatically calculate the optimal parameter using the method of Guerrero (for univariate case) else for the multivariate case, the method of Velilla (1993) which is implemented in the `car` package of John Fox. This targets a transformation to multivariate normality. If any of the inputs has a frequency other than 1, then an stl decomposition is first applied and the seasonal component removed prior to the estimation in order

	to avoid confounding the estimation by seasonality. It is also possible to pass a vector equal to the number of columns of the dataset (with numeric values mixed with NAs which will calculate the univariate optimal lambda).
`lower`	optional parameter lower bound for cases when it is calculated.
`upper`	optional parameter upper bound for cases when it is calculated.
`multivariate`	flag for the multivariate case. If lambda is a single parameter, then that is applied to all series (including NA which results in the multivariate transformation described above).
`...`	not currently used.

Details

The function returns a list of 2 functions called "transform" and "inverse" which can be called with a data object and a frequency to calculate the transformed values. The auto_lambda function uses the method of Guerrero(1993).

Value

A list with the transform and inverse functions.

Note

The returned transform function will take additional argument "frequency" which determines whether a series is seasonal or not. When estimating lambda (when setting this to NA), a series with frequency > 1 will first be de-seasonalized using an STL decomposition.

Author(s)

Alexios Galanos for the BoxCox function.
John Fox for the powerTransform function used in the multivariate case.

References

Box GE, Cox DR (1964). "An analysis of transformations." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **26**(2), 211–243.
Velilla S (1993). "A note on the multivariate Box–Cox transformation to normality." *Statistics & Probability Letters*, **17**(4), 259–263.
Guerrero VM (1993). "Time-series analysis supported by power transformations." *Journal of forecasting*, **12**(1), 37–48.

Examples

```
y = cumprod(c(1, 1 + rnorm(100,0.01, 0.005)))
B = box_cox(lambda = NA)
yt = B$transform(y, frequency = 1)
lambda = attr(yt,"lambda")
ye = B$inverse(yt, lambda)
```

`calendar_eom` *End of Month Date*

Description

Returns the last day of the month from a Date within the month.

Usage

```
calendar_eom(date, ...)
```

Arguments

`date`	a Date vector
`...`	not used

Details

Given a Date (such as 2019-01-02), will return the last Date within that year month.

Value

Date object

Author(s)

Alexios Galanos

`calendar_eoq` *End of Quarter Date*

Description

Returns the last day of the quarter from a Date.

Usage

```
calendar_eoq(date, ...)
```

Arguments

`date`	a Date vector
`...`	not used

Details

Given a date (such as 2019-01-02), will return the last date within that year quarter.

Value

Date object

Author(s)

Alexios Galanos

`calendar_eow` *End of Week Date*

Description

Returns the last day of the week from a Date given a choice of week days.

Usage

```
calendar_eow(date, day = 7, ...)
```

Arguments

`date`	a Date vector
`day`	a value between 1 (Monday) and 7 (Sunday).
`...`	not used

Details

Given a Date (such as 2019-01-02) and a day of 7, will return the Date for the Sunday at or immediately after that. The week starting day is Monday (1). A simple use case is when one wants to aggregate daily data to a regular weekly sequence.

Value

Date object

Author(s)

Alexios Galanos

`calendar_eoy` *End of Year Date*

Description

Returns the last day of the year from a Date.

Usage

```
calendar_eoy(date, ...)
```

Arguments

`date`	a Date vector
`...`	not used

Details

Given a date (such as 2019-01-02), will return the last date within that year.

Value

Date object

Author(s)

Alexios Galanos

`check_xreg` *Checks on regressor matrix.*

Description

Used internally by other packages, these functions provides some commonly used validation checks on regressor matrices in both in and out of sample.

Usage

```
check_xreg(xreg, valid_index)

check_newxreg(newdata, xreg_names = NULL, h = 1, forc_dates = NULL)
```

Arguments

`xreg`	an xts matrix of named regressors.
`valid_index`	a vector of dates against which the xreg matrix index is compared for validity.
`newdata`	an xts matrix of out of named sample regressors.
`xreg_names`	names of regressors used in sample.
`h`	the forecast horizon
`forc_dates`	an optional vector of forecast dates. This is used if newdata is not an xts matrix in which case it formats the data into such using the forc_dates vector.

Value

Returns the xts input matrix if checks are passed else raises an error.

`fourier_series` *Fourier terms for modeling seasonality*

Description

Returns a matrix containing terms from a Fourier series, up to order K

Usage

```
fourier_series(dates, period = NULL, K = NULL)
```

Arguments

`dates`	a Date vector representing the length of the series for which the fourier terms are required.
`period`	frequency of the underlying series, if NULL will try to infer it from the difference in the Date vector.
`K`	maximum order of the Fourier terms.

Value

A matrix of size N (length of dates) by 2*K.

`future_dates` *Generate Regular Interval Future Dates*

Description

Generates regular interval future dates for use in forecast routine.

Usage

```
future_dates(start, frequency, n = 1)
```

Arguments

`start`	a Date string for the start date.
`frequency`	frequency of the interval (daily, weekly, monthly or yearly).
`n`	number of future periods to generate dates for.

Value

A Date vector

Author(s)

Alexios Galanos

`initialize_simulator` *Simulator Initializer*

Description

Simulator Initializer

Usage

```
initialize_simulator(x, index = NULL, sampling = NULL, model = "issm", ...)
```

Arguments

`x`	a vector of zero mean errors to use in the model.
`index`	an optional Date or POSIXct vector of same length as x. Used for indexing the simulated values.
`sampling`	an optional string denoting the sampling frequency for the simulator. If no index is present, will automatically generate one based on the sampling frequency given with start date 2000-01-01. Valid sampling frequencies are days, weeks, months, years, secs, mins, hours and subintervals of those as documented in the `seq.POSIXt` function.

`model`	the type of model to initialize a class for.
`...`	additional parameters to the function (not currently used).

Value

A object whose class depends on the type of model used.

`lines.issm.component` *Add Connected Line Segments to a Simulation Object*

Description

Add Connected Line Segments to a Simulation Object

Usage

```
## S3 method for class 'issm.component'
lines(x, y = NULL, type = "l", ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`y`	not used.
`type`	character indicating the type of plotting.
`...`	additional parameters passed to the lines function.

Details

Overlays the simulated series from the object (x), and is meant to be used when plotting different simulations from the same series for comparison.

Value

a line plot.

logit — *The logit transformation*

Description

The logit transformation as an alternative to the Box Cox for bounded outcomes.

Usage

```
logit(lower = 0, upper = 1, ...)
```

Arguments

`lower`	lower bound of the variable.
`upper`	upper bound of the variable.
`...`	not currently used.

Value

A list with the transform and inverse functions.

Author(s)

Alexios Galanos

mape — *Forecast Performance Metrics*

Description

Functions to calculate a number of performance metrics.

Usage

```
mape(actual, predicted)

bias(actual, predicted)

mslre(actual, predicted)

mase(actual, predicted, original_series = NULL, frequency = 1)

mis(actual, lower, upper, alpha)

wape(actual, predicted, weights)
```

```
wslre(actual, predicted, weights)

wse(actual, predicted, weights)

pinball(actual, distribution, alpha = 0.1)

crps(actual, distribution)

rmape(actual, predicted)

smape(actual, predicted)

msis(actual, lower, upper, original_series, frequency = 1, alpha)
```

Arguments

`actual`	the actual values corresponding to the forecast period.
`predicted`	the predicted values corresponding to the forecast period.
`original_series`	the actual values corresponding to the training period.
`frequency`	the seasonal frequency of the series used in the model.
`lower`	the lower distributional forecast for the quantile corresponding to the coverage ratio alpha (i.e. alpha/2).
`upper`	the upper distributional forecast for the quantile corresponding to the coverage ratio alpha (i.e. 1 - alpha/2).
`alpha`	the distributional coverage.
`weights`	a vector of weights for generating weighted metrics. If the actual and predicted inputs are univariate, this should be equal to the length of the actual series and calculates a time-weighted average; otherwise, the weights should be of length equal to the number of series in a multivariate case, in which case a cross-sectional average is calculated.
`distribution`	the forecast distribution (returned in the distribution slot of the prediction object). This is used in the continuous ranked probability score (crps) of Gneiting et al. (2005), and calculated using the function from the ‘scoringRules‘ package.

Details

The following performance metrics are implemented:

Mean Absolute Percentage Error (MAPE) Measures the average percentage deviation of predictions from actual values.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - P_t}{A_t}\right|$$

where A_t is the actual value and P_t is the predicted value.

Rescaled Mean Absolute Percentage Error (RMAPE) A transformation of MAPE using a Box-Cox transformation for scale invariance (Swanson et al.).

Symmetric Mean Absolute Percentage Error (SMAPE) An alternative to MAPE that symmetrizes the denominator.

$$SMAPE = \frac{2}{n}\sum_{t=1}^{n}\frac{|A_t - P_t|}{|A_t| + |P_t|}$$

Mean Absolute Scaled Error (MASE) Compares the absolute error to the mean absolute error of a naive seasonal forecast.

$$MASE = \frac{\frac{1}{n}\sum_{t=1}^{n}|P_t - A_t|}{\frac{1}{N-s}\sum_{t=s+1}^{N}|A_t - A_{t-s}|}$$

where s is the seasonal period.

Mean Squared Logarithmic Relative Error (MSLRE) Measures squared log relative errors to penalize large deviations.

$$MSLRE = \frac{1}{n}\sum_{t=1}^{n}(\log(1 + A_t) - \log(1 + P_t))^2$$

Mean Interval Score (MIS) Evaluates the accuracy of prediction intervals.

$$MIS = \frac{1}{n}\sum_{t=1}^{n}(U_t - L_t) + \frac{2}{\alpha}[(L_t - A_t)I(A_t < L_t) + (A_t - U_t)I(A_t > U_t)]$$

where L_t and U_t are the lower and upper bounds of the interval.

Mean Scaled Interval Score (MSIS) A scaled version of MIS, dividing by the mean absolute seasonal error.

$$MSIS = \frac{1}{h}\sum_{t=1}^{h}\frac{(U_t - L_t) + \frac{2}{\alpha}[(L_t - A_t)I(A_t < L_t) + (A_t - U_t)I(A_t > U_t)]}{\frac{1}{N-s}\sum_{t=s+1}^{N}|A_t - A_{t-s}|}$$

Bias Measures systematic overestimation or underestimation.

$$Bias = \frac{1}{n}\sum_{t=1}^{n}(P_t - A_t)$$

Weighted Absolute Percentage Error (WAPE) A weighted version of MAPE.

$$WAPE = \sum_{t=1}^{n}\mathbf{w}\frac{|P_t - A_t|}{A_t}$$

where $\mathbf{w}$ is the weight vector.

Weighted Squared Logarithmic Relative Error (WSLRE) A weighted version of squared log relative errors.

$$WSLRE = \sum_{t=1}^{n}\mathbf{w}(\log(P_t/A_t))^2$$

Weighted Squared Error (WSE) A weighted version of squared errors.

$$WSE = \sum_{t=1}^{n} \mathbf{w} \left(\frac{P_t}{A_t} \right)^2$$

Pinball Loss A scoring rule used for quantile forecasts.

$$\text{Pinball} = \frac{1}{n} \sum_{t=1}^{n} \left[\tau (A_t - Q_t^\tau) I(A_t \geq Q_t^\tau) + (1 - \tau)(Q_t^\tau - A_t) I(A_t < Q_t^\tau) \right]$$

where

$$Q_t^\tau$$

is the predicted quantile at level

$$\tau$$

.

Continuous Ranked Probability Score (CRPS) A measure of probabilistic forecast accuracy.

$$CRPS = \frac{1}{n} \sum_{t=1}^{n} \int_{-\infty}^{\infty} (F_t(y) - I(y \geq A_t))^2 dy$$

where $F_t(y)$ is the cumulative forecast distribution.

Value

A numeric value.

Note

The RMAPE is the rescaled measure for MAPE based on the paper by Swanson et al.

Author(s)

Alexios Galanos

References

Tofallis C (2015). "A better measure of relative prediction accuracy for model selection and model estimation." *Journal of the Operational Research Society*, **66**(8), 1352–1362.
Hyndman RJ, Koehler AB (2006). "Another look at measures of forecast accuracy." *International journal of forecasting*, **22**(4), 679–688.
Gneiting T, Raftery AE, Westveld III AH, Goldman T (2005). "Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation." *Monthly weather review*, **133**(5), 1098–1118.
Gneiting T, Raftery AE (2007). "Strictly proper scoring rules, prediction, and estimation." *Journal of the American statistical Association*, **102**(477), 359–378.
Swanson DA, Tayman J, Bryan TM (2011). "MAPE-R: a rescaled measure of accuracy for cross-sectional subnational population forecasts." *Journal of Population Research*, **28**, 225–243.

`mixture_modelspec` *Ensemble Setup*

Description

Ensemble Setup

Usage

```
mixture_modelspec(...)
```

Arguments

`...`	either a list of valid simulation objects or individual objects passed to the function

Details

The function performs certain checks on the inputs to ensure they conform to the simulation models in the package and are of the same length.

Value

A object of class tssim.mixture ready for ensembling,

`plot.issm.component` *Plot Simulation Object*

Description

Plot Simulation Object

Usage

```
## S3 method for class 'issm.component'
plot(x, y = c("simulated", "components"), ...)
```

Arguments

`x`	an object of class issm.component or other supported class.
`y`	the type of output to plot.
`...`	additional parameters passed to the `plot.zoo` function.

Value

a plot of the simulated series or multiple plots of the simulation components.

`process_time` *POSIXct Processing*

Description

Ceiling, Floor and Other operations on a POSIXct object

Usage

```
process_time(x, second_precision = 3600, method = ceiling, ...)
```

Arguments

`x`	a POSIXct vector
`second_precision`	the precision in seconds on which the processing operates on
`method`	the method for processing
`...`	not used

Value

POSIXct object

Author(s)

Alexios Galanos

Examples

```
# end of hour
process_time(as.POSIXct('2022-08-03 03:00:01', tz = 'UTC'), 3600, method = ceiling)
# start of hour
process_time(as.POSIXct('2022-08-03 03:00:01', tz = 'UTC'), 3600, method = floor)
# end of minute
process_time(as.POSIXct('2022-08-03 03:00:01', tz = 'UTC'), 60, method = ceiling)
```

`sampling_frequency` *Infers the sampling frequency of a time series*

Description

Given either a vector of time indices or an xts object will infer the sampling frequency.

Usage

```
sampling_frequency(x)
```

Arguments

`x`	either an xts object (or one which has an index attribute) else a vector of class Date or POSIX based time index

Value

the sampling period (character).

Examples

```
w <- sampling_frequency(seq(as.Date("2010-01-01"), as.Date("2011-01-01"), by="weeks"))
m <- sampling_frequency(seq(as.POSIXct("2010-01-01 12:00:00"),
as.POSIXct("2010-01-02 12:00:00"), by="15 mins"))
```

`sampling_sequence` *Sampling frequency sequence*

Description

Given a sampling period, the function will return the proportion of units of that period in secs, mins, hours, days, weeks, months and years, but will return NA for periods of higher frequency i.e. for a period of days it will return NA for secs, mins and hours. The function serves as a helper for seasonal periodicity calculations.

Usage

```
sampling_sequence(period)
```

Arguments

`period`	the period returned by a call to the function `sampling_frequency`.

Value

A named numeric vector.

Author(s)

Alexios Galanos

Examples

```
w <- sampling_sequence(sampling_frequency(seq(as.Date("2010-01-01"),
as.Date("2011-01-01"), by="weeks")))
m <- sampling_sequence(sampling_frequency(seq(as.POSIXct("2010-01-01 12:00:00"),
as.POSIXct("2010-01-02 12:00:00"), by="15 mins")))
```

`seasonality_test` *Simple Seasonality Test*

Description

Checks for the presence of seasonality based on the QS test of Gomez and Maravall (1996).

Usage

```
seasonality_test(x, frequency = NULL)
```

Arguments

`x`	an (xts) vector (usually of a stationary series).
`frequency`	overrides any frequency automatically identified in the index of x.

Details

Given the identified frequency of the xts vector (using the `sampling_frequency`), the function checks for seasonality at that frequency. The frequency can be overridden by directly supplying a frequency argument, in which case y does not need to be a xts vector.

Value

Logical.

Author(s)

Alexios Galanos

References

Gómez V, Maravall A (1995). *Programs TRAMO and SEATS*. European University Institute, Florence.

`seasonal_dummies` *Seasonal Dummies*

Description

Creates a matrix of seasonal dummies.

Usage

```
seasonal_dummies(y = NULL, n = nrow(y), seasons = 12)
```

Arguments

`y`	optional data series.
`n`	if y is missing, then the length of the series is required.
`seasons`	number of seasons in a cycle.

Details

Generates seasons-1 dummy variables.

Value

Either a matrix (if y is missing or y is not an xts vector) or an xts matrix (when y is an xts vector).

Author(s)

Alexios Galanos

Examples

```
head(seasonal_dummies(n=100, seasons=12))
```

`sigmoid` *The sigmoid transformation*

Description

The sigmoid function is a smooth, S-shaped function that maps any real-valued input into a bounded interval, typically (0,1) . It is widely used in probability modeling, logistic regression, and neural networks as an activation function.

Usage

```
sigmoid(lower = 0, upper = 1, ...)
```

Arguments

`lower`	lower bound of the variable.
`upper`	upper bound of the variable.
`...`	not currently used.

Value

A list with the transform and inverse functions.

Author(s)

Alexios Galanos

Examples

```
y = cumprod(c(1, 1 + rnorm(100,0.01, 0.005)))
B = sigmoid()
yt = B$transform(y)
ye = B$inverse(yt)
```

softlogit *The softplus logit transformation*

Description

The softplus logit transformation is an alternative to the logit transform for bounded outcomes with positive output.

Usage

```
softlogit(lower = 0, upper = 1, ...)
```

Arguments

`lower`	lower bound of the variable.
`upper`	upper bound of the variable.
`...`	not currently used.

Value

A list with the transform and inverse functions.

Author(s)

Alexios Galanos

Examples

```
y = cumprod(c(1, 1 + rnorm(100,0.01, 0.005)))
B = softlogit(lower = 0,  upper = 15)
yt = B$transform(y)
ye = B$inverse(yt)
```

`time_splits` *Generate Train/Test Splits*

Description

Generates train/test splits given a vector of dates and other options

Usage

```
time_splits(
  x,
  start = x[1],
  test_length = 1,
  by = test_length,
  window_size = NULL,
  calendar_end = NULL,
  complete_index = TRUE,
  ...
)
```

Arguments

`x`	a vector of timestamps (POSIXct) or dates (Date) in the dataset
`start`	starting date (first estimation/train date)
`test_length`	type of calendar period to split on
`by`	every how many periods to split on
`window_size`	the size of the training set (for moving window). If NULL will use an expanding window.
`calendar_end`	an optional function to use for the period ending split, such as `calendar_eow`, applied to x. This should be greater in frequency than the underlying frequency of x (i.e. do not use calendar_eow on monthly indices). This overwrites the use of window_size.
`complete_index`	whether to return the full indices for train and test else just the start and end indices.
`...`	any additional parameters passed to the calendar_end function. For example, the “day” argument when using the calendar_eow function.

Value

A list with each slot having the training dates and test dates

Note

For months, quarters and years this will split into the end date of these. For splitting into mins or hours, x must also have this resolution else will throw an error. Additionally, the strict requirement of regularly spaced time is required (no gaps).

Author(s)

Alexios Galanos

---

tsdecompose.issm.component *State Decomposition*

Description

State Decomposition

Usage

```
## S3 method for class 'issm.component'
tsdecompose(object, ...)
```

Arguments

`object`	an object of class issm.component or other supported class.
`...`	additional parameters.

Details

Creates a simplified decomposition of the states and aligns their time indices so that the sum up to the simulated component per period.

Value

A matrix of the simplified state decomposition.

---

tsensemble.tssim.mixture *Ensembling of Simulations*

Description

Ensembling of Simulations

Usage

```
## S3 method for class 'tssim.mixture'
tsensemble(object, weights = NULL, difference = TRUE, ...)
```

Arguments

`object`	an object of class tssim.mixture.
`weights`	the weighting (or probability) matrix for aggregating the simulations (see details).
`difference`	whether to take the rates of changes first before aggregating and reconverting to levels.
`...`	additional parameters.

Details

When mixing dynamics for the same series, and when series are not stationary, differences should be used. In that case the rate of change transformation is applied to each simulated series and then weighted by the weights matrix. Since the weights matrix will have one more row than is required (the first row), this can be used to choose how the initial level is generated. For instance, if we want to use the level of the first simulated series, then the first row would have a 1 on the first column and zeros in the rest. For aggregating series, difference should be set to FALSE since we are looking at summation of data (under the assumption of flow variables). In this case, the p matrix is usually static by column (i.e. the same weights).

Value

A vector of the simulated series.

`tslinear` *Linear Time Series Filter*

Description

Estimates a simple linear time series model with trend, seasonal and regressors.

Usage

```
tslinear(y, trend = FALSE, seasonal = FALSE, xreg = NULL, frequency = 1, ...)
```

Arguments

`y`	a vector.
`trend`	whether to include a linear trend.
`seasonal`	whether to include seasonal dummies.
`xreg`	an optional matrix of regressors.
`frequency`	the frequency of the series (required if seasonal is TRUE).
`...`	not currently used.

Value

An object of class “tslinear” which also inherits “lm”.

Author(s)

Alexios Galanos

`tstransform` *General transformation function*

Description

Includes the Box Cox, logit, softplus-logit and sigmoid transforms. Returns a list of functions for the transform and its inverse.

Usage

```
tstransform(method = "box-cox", lambda = NULL, lower = 0, upper = 1, ...)
```

Arguments

`method`	valid methods are currently “box-cox”, “logit”, “softplus-logit” and “sigmoid”.
`lambda`	parameter in the Box Cox transformation.
`lower`	lower bound for the transformations.
`upper`	upper bound for the transformations.
`...`	additional arguments taken by the transformations.

Value

A list with the transform and inverse functions.

Author(s)

Alexios Galanos

Index