Package 'feasts'

February 5, 2021

Title Feature Extraction and Statistics for Time Series

Version 0.1.7

Description Provides a collection of features, decomposition methods, statistical summaries and graphics functions for the analysing tidy time series data. The package name 'feasts' is an acronym comprising of its key features: Feature Extraction And Statistics for Time Series.

Depends R (>= 3.5.0), fabletools (>= 0.2.0)

Imports rlang (>= 0.2.0), tibble (>= 1.4.1), tsibble (>= 0.9.0), ggplot2 (>= 3.0.0), dplyr (>= 1.0.0), tidyr (>= 0.8.3), scales (>= 1.1.0), vctrs, lubridate, grid, slider, utils, lifecycle

Suggests tsibbledata, pillar (>= 1.0.1), knitr, rmarkdown, testthat, covr, seasonal, urca, fracdiff, fable

ByteCompile true

VignetteBuilder knitr

License GPL-3

URL http://feasts.tidyverts.org/, https://github.com/tidyverts/feasts/

BugReports https://github.com/tidyverts/feasts/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Language en-GB

RdMacros lifecycle

NeedsCompilation no

Author Mitchell O'Hara-Wild [aut, cre],
Rob Hyndman [aut],
Earo Wang [aut],
Di Cook [ctb],
Thiyanga Talagala [ctb] (Correlation features),
Leanne Chhay [ctb] (Guerrero's method)

Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Repository CRAN

Date/Publication 2021-02-05 14:10:02 UTC

R topics documented:

feasts-package 2
ACF 3
autoplot.tbl_cf 5
classical_decomposition 5
coef_hurst 7
feat_acf 7
feat_intermittent 8
feat_pacf 9
feat_spectral 9
feat_stl 11
generate.stl_decomposition 11
gg_arma 12
gg_lag 13
gg_season 14
gg_subseries 15
gg_tsdisplay 16
gg_tsresiduals 18
guerrero 19
ljung_box 20
longest_flat_spot 21
n_crossing_points 21
scale_cf_lag 22
shift_level_max 22
stat_arch_lm 23
STL 24
unitroot_kpss 25
unitroot_ndiffs 26
var_tiled_var 27

Index **28**

`feasts-package` *feasts: Feature Extraction and Statistics for Time Series*

Description

Provides a collection of features, decomposition methods, statistical summaries and graphics functions for the analysing tidy time series data. The package name 'feasts' is an acronym comprising of its key features: Feature Extraction And Statistics for Time Series.

Author(s)

Maintainer: Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Authors:

- Rob Hyndman
- Earo Wang

Other contributors:

- Di Cook [contributor]
- Thiyanga Talagala (Correlation features) [contributor]
- Leanne Chhay (Guerrero's method) [contributor]

See Also

Useful links:

- http://feasts.tidyverts.org/
- https://github.com/tidyverts/feasts/
- Report bugs at https://github.com/tidyverts/feasts/issues

ACF *(Partial) Autocorrelation and Cross-Correlation Function Estimation*

Description

The function ACF computes an estimate of the autocorrelation function of a (possibly multivariate) tsibble. Function PACF computes an estimate of the partial autocorrelation function of a (possibly multivariate) tsibble. Function CCF computes the cross-correlation or cross-covariance of two columns from a tsibble.

Usage

```
ACF(
  .data,
  ...,
  lag_max = NULL,
  demean = TRUE,
  type = c("correlation", "covariance", "partial")
)

PACF(.data, ..., lag_max = NULL)

CCF(.data, ..., lag_max = NULL, type = c("correlation", "covariance"))
```

Arguments

.data	A tsibble
...	The column(s) from the tsibble used to compute the ACF, PACF or CCF.
lag_max	maximum lag at which to calculate the acf. Default is 10*log10(N/m) where N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series.
demean	logical. Should the covariances be about the sample means?
type	character string giving the type of acf to be computed. Allowed values are "correlation" (the default), "covariance" or "partial". Will be partially matched.

Details

The functions improve the `stats::acf()`, `stats::pacf()` and `stats::ccf()` functions. The main differences are that `ACF` does not plot the exact correlation at lag 0 when `type=="correlation"` and the horizontal axes show lags in time units rather than seasonal units.

The resulting tables from these functions can also be plotted using `autoplot.tbl_cf()`.

Value

The `ACF`, `PACF` and `CCF` functions return objects of class "tbl_cf", which is a tsibble containing the correlations computed.

Author(s)

Mitchell O'Hara-Wild and Rob J Hyndman

References

Hyndman, R.J. (2015). Discussion of "High-dimensional autocovariance matrices and optimal linear prediction". *Electronic Journal of Statistics*, 9, 792-796.

McMurry, T. L., & Politis, D. N. (2010). Banded and tapered estimates for autocovariance matrices and the linear process bootstrap. *Journal of Time Series Analysis*, 31(6), 471-482.

See Also

`stats::acf()`, `stats::pacf()`, `stats::ccf()`

Examples

```
library(tsibble)
library(tsibbledata)
library(dplyr)

vic_elec %>% ACF(Temperature)

vic_elec %>% ACF(Temperature) %>% autoplot()
```

```
vic_elec %>% PACF(Temperature)

vic_elec %>% PACF(Temperature) %>% autoplot()

global_economy %>%
  filter(Country == "Australia") %>%
  CCF(GDP, Population)

global_economy %>%
  filter(Country == "Australia") %>%
  CCF(GDP, Population) %>%
  autoplot()
```

autoplot.tbl_cf — *Auto- and Cross- Covariance and -Correlation plots*

Description

Produces an appropriate plot for the result of ACF(), PACF(), or CCF().

Usage

```
## S3 method for class 'tbl_cf'
autoplot(object, level = 95, ...)
```

Arguments

object	A tbl_cf object (the result ACF(), PACF(), or CCF()).
level	The level of confidence for the blue dashed lines.
...	Unused.

Value

A ggplot object showing the correlations.

classical_decomposition — *Classical Seasonal Decomposition by Moving Averages*

Description

Decompose a time series into seasonal, trend and irregular components using moving averages. Deals with additive or multiplicative seasonal component.

Usage

```
classical_decomposition(formula, type = c("additive", "multiplicative"), ...)
```

Arguments

`formula`	Decomposition specification (see "Specials" section).
`type`	The type of seasonal component. Can be abbreviated.
`...`	Other arguments passed to \link[stats]{decompose}.

Details

The additive model used is:

$$Y_t = T_t + S_t + e_t$$

The multiplicative model used is:

$$Y_t = T_t\,S_t\,e_t$$

The function first determines the trend component using a moving average (if `filter` is `NULL`, a symmetric window with equal weights is used), and removes it from the time series. Then, the seasonal figure is computed by averaging, for each time unit, over all periods. The seasonal figure is then centered. Finally, the error component is determined by removing trend and seasonal figure (recycled as needed) from the original time series.

This only works well if `x` covers an integer number of complete periods.

Value

A `fabletools::dable()` containing the decomposed trend, seasonality and remainder from the classical decomposition.

Specials

season: The `season` special is used to specify seasonal attributes of the decomposition.

```
season(period = NULL)
```

`period` The periodic nature of the seasonality. This can be either a number indicating the number of observations in each sea

Examples

```
as_tsibble(USAccDeaths) %>%
  model(classical_decomposition(value)) %>%
  components()

as_tsibble(USAccDeaths) %>%
  model(classical_decomposition(value ~ season(12), type = "mult")) %>%
  components()
```

`coef_hurst` *Hurst coefficient*

Description

Computes the Hurst coefficient indicating the level of fractional differencing of a time series.

Usage

```
coef_hurst(x)
```

Arguments

`x`	a vector. If missing values are present, the largest contiguous portion of the vector is used.

Value

A numeric value.

Author(s)

Rob J Hyndman

`feat_acf` *Autocorrelation-based features*

Description

Computes various measures based on autocorrelation coefficients of the original series, first-differenced series and second-differenced series

Usage

```
feat_acf(x, .period = 1, lag_max = NULL, ...)
```

Arguments

`x`	a univariate time series
`.period`	The seasonal period (optional)
`lag_max`	maximum lag at which to calculate the acf. The default is `max(.period,10L)` for `feat_acf`, and `max(.period,5L)` for `feat_pacf`
`...`	Further arguments passed to `stats::acf()` or `stats::pacf()`

Value

A vector of 6 values: first autocorrelation coefficient and sum of squared of first ten autocorrelation coefficients of original series, first-differenced series, and twice-differenced series. For seasonal data, the autocorrelation coefficient at the first seasonal lag is also returned.

Author(s)

Thiyanga Talagala

`feat_intermittent` *Intermittency features*

Description

Computes various measures that can indicate the presence and structures of intermittent data.

Usage

```
feat_intermittent(x)
```

Arguments

`x`	A vector to extract features from.

Value

A vector of named features:

- zero_run_mean: The average interval between non-zero observations
- nonzero_squared_cv: The squared coefficient of variation of non-zero observations
- zero_start_prop: The proportion of data which starts with zero
- zero_end_prop: The proportion of data which ends with zero

References

Kostenko, A. V., & Hyndman, R. J. (2006). A note on the categorization of demand patterns. *Journal of the Operational Research Society*, 57(10), 1256-1257.

`feat_pacf` *Partial autocorrelation-based features*

Description

Computes various measures based on partial autocorrelation coefficients of the original series, first-differenced series and second-differenced series.

Usage

```
feat_pacf(x, .period = 1, lag_max = NULL, ...)
```

Arguments

`x`	a univariate time series
`.period`	The seasonal period (optional)
`lag_max`	maximum lag at which to calculate the acf. The default is `max(.period,10L)` for `feat_acf`, and `max(.period,5L)` for `feat_pacf`
`...`	Further arguments passed to `stats::acf()` or `stats::pacf()`

Value

A vector of 3 values: Sum of squared of first 5 partial autocorrelation coefficients of the original series, first differenced series and twice-differenced series. For seasonal data, the partial autocorrelation coefficient at the first seasonal lag is also returned.

Author(s)

Thiyanga Talagala

`feat_spectral` *Spectral features of a time series*

Description

Computes spectral entropy from a univariate normalized spectral density, estimated using an AR model.

Usage

```
feat_spectral(x, .period = 1, ...)
```

Arguments

x	a univariate time series
.period	The seasonal period.
...	Further arguments for `stats::spec.ar()`

Details

The *spectral entropy* equals the Shannon entropy of the spectral density $f_x(\lambda)$ of a stationary process x_t:

$$H_s(x_t) = -\int_{-\pi}^{\pi} f_x(\lambda) \log f_x(\lambda) d\lambda,$$

where the density is normalized such that $\int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1$. An estimate of $f(\lambda)$ can be obtained using `spec.ar` with the burg method.

Value

A non-negative real value for the spectral entropy $H_s(x_t)$.

Author(s)

Rob J Hyndman

References

Jerry D. Gibson and Jaewoo Jung (2006). "The Interpretation of Spectral Entropy Based Upon Rate Distortion Functions". IEEE International Symposium on Information Theory, pp. 277-281.

Goerg, G. M. (2013). "Forecastable Component Analysis". Journal of Machine Learning Research (JMLR) W&CP 28 (2): 64-72, 2013. Available at https://jmlr.org/proceedings/papers/v28/goerg13.html.

See Also

`spec.ar`

Examples

```
feat_spectral(rnorm(1000))
feat_spectral(lynx)
feat_spectral(sin(1:20))
```

`feat_stl` *STL features*

Description

Computes a variety of measures extracted from an STL decomposition of the time series. This includes details about the strength of trend and seasonality.

Usage

```
feat_stl(x, .period, s.window = 13, ...)
```

Arguments

`x`	A vector to extract features from.
`.period`	The period of the seasonality.
`s.window`	The seasonal window of the data (passed to `stats::stl()`)
`...`	Further arguments passed to `stats::stl()`

Value

A vector of numeric features from a STL decomposition.

See Also

Forecasting Principle and Practices: Measuring strength of trend and seasonality

`generate.stl_decomposition` *Generate block bootstrapped series from an STL decomposition*

Description

Produces new data with the same structure by resampling the residuals using a block bootstrap procedure. This method can only generate within sample, and any generated data out of the trained sample will produce NA simulations.

Usage

```
## S3 method for class 'stl_decomposition'
generate(x, new_data, specials = NULL, ...)
```

Arguments

`x`	A fitted model.
`new_data`	A `tsibble` containing future information used to forecast.
`specials`	(passed by `fabletools::forecast.mdl_df()`).
`...`	Additional arguments for forecast model methods.

References

Bergmeir, C., R. J. Hyndman, and J. M. Benitez (2016). Bagging Exponential Smoothing Methods using STL Decomposition and Box-Cox Transformation. International Journal of Forecasting 32, 303-312.

Examples

```
as_tsibble(USAccDeaths) %>%
  model(STL(log(value))) %>%
  generate(as_tsibble(USAccDeaths), times = 3)
```

`gg_arma` *Plot characteristic ARMA roots*

Description

Produces a plot of the inverse AR and MA roots of an ARIMA model. Inverse roots outside the unit circle are shown in red.

Usage

```
gg_arma(data)
```

Arguments

`data`	A mable containing models with AR and/or MA roots.

Details

Only models which compute ARMA roots can be visualised with this function. That is to say, the `glance()` of the model contains `ar_roots` and `ma_roots`.

Value

A ggplot object the characteristic roots from ARMA components.

Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)
library(tsibble)
library(dplyr)

tsibbledata::aus_retail %>%
  filter(
    State == "Victoria",
    Industry == "Cafes, restaurants and catering services"
  ) %>%
  model(ARIMA(Turnover ~ pdq(0,1,1) + PDQ(0,1,1))) %>%
  gg_arma()
}
```

---

gg_lag *Lag plots*

Description

A lag plot shows the time series against lags of itself. It is often coloured the seasonal period to identify how each season correlates with others.

Usage

```
gg_lag(
  data,
  y = NULL,
  period = NULL,
  lags = 1:9,
  geom = c("path", "point"),
  arrow = FALSE,
  ...
)
```

Arguments

data	A tidy time series object (tsibble)
y	The variable to plot (a bare expression). If NULL, it will automatically selected from the data.
period	The seasonal period to display.
lags	A vector of lags to display as facets.
geom	The geometry used to display the data.
arrow	Arrow specification to show the direction in the lag path. If TRUE, an appropriate default arrow will be used. Alternatively, a user controllable arrow created with grid::arrow() can be used.
...	Additional arguments passed to the geom.

Value

A ggplot object showing a lag plot of a time series.

Examples

```
library(tsibble)
library(dplyr)
tsibbledata::aus_retail %>%
  filter(
    State == "Victoria",
    Industry == "Cafes, restaurants and catering services"
  ) %>%
  gg_lag(Turnover)
```

---

gg_season *Seasonal plot*

Description

Produces a time series seasonal plot. A seasonal plot is similar to a regular time series plot, except the x-axis shows data from within each season. This plot type allows the underlying seasonal pattern to be seen more clearly, and is especially useful in identifying years in which the pattern changes.

Usage

```
gg_season(
  data,
  y = NULL,
  period = NULL,
  facet_period = NULL,
  max_col = 15,
  pal = (scales::hue_pal())(9),
  polar = FALSE,
  labels = c("none", "left", "right", "both"),
  ...
)
```

Arguments

data	A tidy time series object (tsibble)
y	The variable to plot (a bare expression). If NULL, it will automatically selected from the data.
period	The seasonal period to display.
facet_period	A secondary seasonal period to facet by (typically smaller than period).

max_col	The maximum number of colours to display on the plot. If the number of seasonal periods in the data is larger than max_col, the plot will not include a colour. Use max_col = 0 to never colour the lines, or Inf to always colour the lines. If labels are used, then max_col will be ignored.
pal	A colour palette to be used.
polar	If TRUE, the season plot will be shown on polar coordinates.
labels	Position of the labels for seasonal period identifier.
...	Additional arguments passed to geom_line()

Value

A ggplot object showing a seasonal plot of a time series.

References

Hyndman and Athanasopoulos (2019) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. https://OTexts.com/fpp3/

Examples

```
library(tsibble)
library(dplyr)
tsibbledata::aus_retail %>%
  filter(
    State == "Victoria",
    Industry == "Cafes, restaurants and catering services"
  ) %>%
  gg_season(Turnover)
```

---

gg_subseries *Seasonal subseries plots*

Description

A seasonal subseries plot facets the time series by each season in the seasonal period. These facets form smaller time series plots consisting of data only from that season. If you had several years of monthly data, the resulting plot would show a separate time series plot for each month. The first subseries plot would consist of only data from January. This case is given as an example below.

Usage

```
gg_subseries(data, y = NULL, period = NULL, ...)
```

Arguments

data	A tidy time series object (tsibble)
y	The variable to plot (a bare expression). If NULL, it will automatically selected from the data.
period	The seasonal period to display.
...	Additional arguments passed to geom_line()

Details

The horizontal lines are used to represent the mean of each facet, allowing easy identification of seasonal differences between seasons. This plot is particularly useful in identifying changes in the seasonal pattern over time.

similar to a seasonal plot (gg_season()), and

Value

A ggplot object showing a seasonal subseries plot of a time series.

References

Hyndman and Athanasopoulos (2019) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. https://OTexts.com/fpp3/

Examples

```
library(tsibble)
library(dplyr)
tsibbledata::aus_retail %>%
  filter(
    State == "Victoria",
    Industry == "Cafes, restaurants and catering services"
  ) %>%
  gg_subseries(Turnover)
```

---

gg_tsdisplay *Ensemble of time series displays*

Description

Plots a time series along with its ACF along with an customisable third graphic of either a PACF, histogram, lagged scatterplot or spectral density.

Usage

```
gg_tsdisplay(
  data,
  y = NULL,
  plot_type = c("auto", "partial", "season", "histogram", "scatter", "spectrum"),
  lag_max = NULL
)
```

Arguments

data	A tidy time series object (tsibble)
y	The variable to plot (a bare expression). If NULL, it will automatically selected from the data.
plot_type	type of plot to include in lower right corner. By default ("auto") a season plot will be shown for seasonal data, a spectrum plot will be shown for non-seasonal data without missing values, and a PACF will be shown otherwise.
lag_max	maximum lag at which to calculate the acf. Default is 10*log10(N/m) where N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series.

Value

A list of ggplot objects showing useful plots of a time series.

Author(s)

Rob J Hyndman & Mitchell O'Hara-Wild

References

Hyndman and Athanasopoulos (2019) *Forecasting: principles and practice*, 3rd edition, OTexts: Melbourne, Australia. https://OTexts.com/fpp3/

See Also

plot.ts, ACF, spec.ar

Examples

```
library(tsibble)
library(dplyr)
tsibbledata::aus_retail %>%
  filter(
    State == "Victoria",
    Industry == "Cafes, restaurants and catering services"
  ) %>%
  gg_tsdisplay(Turnover)
```

gg_tsresiduals *Ensemble of time series residual diagnostic plots*

Description

Plots the residuals using a time series plot, ACF and histogram.

Usage

```
gg_tsresiduals(data, ...)
```

Arguments

data	A mable containing one model with residuals.
...	Additional arguments passed to gg_tsdisplay().

Value

A list of ggplot objects showing a useful plots of a time series model's residuals.

References

Hyndman and Athanasopoulos (2019) *Forecasting: principles and practice*, 3rd edition, OTexts: Melbourne, Australia. https://OTexts.com/fpp3/

See Also

gg_tsdisplay()

Examples

```
if (requireNamespace("fable", quietly = TRUE)) {
library(fable)

tsibbledata::aus_production %>%
  model(ETS(Beer)) %>%
  gg_tsresiduals()
}
```

guerrero *Guerrero's method for Box Cox lambda selection*

Description

Applies Guerrero's (1993) method to select the lambda which minimises the coefficient of variation for subseries of x.

Usage

```
guerrero(x, lower = -0.9, upper = 2, .period = 2L)
```

Arguments

`x`	A numeric vector. The data used to identify the transformation parameter lambda.
`lower`	The lower bound for lambda.
`upper`	The upper bound for lambda.
`.period`	The length of each subseries (usually the length of seasonal period). Subseries length must be at least 2.

Details

Note that this function will give slightly different results to `forecast::BoxCox.lambda(y)` if your data does not start at the start of the seasonal period. This function will make use of all of your data, whereas the forecast package will not use data that doesn't complete a seasonal period.

Value

A Box Cox transformation parameter (lambda) chosen by Guerrero's method.

References

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. JRSS B 26 211–246.

Guerrero, V.M. (1993) Time-series analysis supported by power transformations. Journal of Forecasting, 12, 37–48.

ljung_box *Portmanteau tests*

Description

Compute the Box–Pierce or Ljung–Box test statistic for examining the null hypothesis of independence in a given time series. These are sometimes known as 'portmanteau' tests.

Usage

```
ljung_box(x, lag = 1, dof = 0, ...)

box_pierce(x, lag = 1, dof = 0, ...)

portmanteau_tests
```

Arguments

x	A numeric vector
lag	The number of lag autocorrelation coefficients to use in calculating the statistic
dof	Degrees of freedom of the fitted model (useful if x is a series of residuals).
...	Unused.

Format

An object of class `list` of length 2.

Value

A vector of numeric features for the test's statistic and p-value.

See Also

`stats::Box.test()`

Examples

```
ljung_box(rnorm(100))

box_pierce(rnorm(100))
```

`longest_flat_spot` *Longest flat spot length*

Description

"Flat spots" are computed by dividing the sample space of a time series into ten equal-sized intervals, and computing the maximum run length within any single interval.

Usage

```
longest_flat_spot(x)
```

Arguments

x	a vector

Value

A numeric value.

Author(s)

Earo Wang and Rob J Hyndman

`n_crossing_points` *Number of crossing points*

Description

Computes the number of times a time series crosses the median.

Usage

```
n_crossing_points(x)
```

Arguments

x	a univariate time series

Value

A numeric value.

Author(s)

Earo Wang and Rob J Hyndman

scale_cf_lag *lagged datetime scales This set of scales defines new scales for lagged time structures.*

Description

lagged datetime scales This set of scales defines new scales for lagged time structures.

Usage

```
scale_x_cf_lag(...)
```

Arguments

`...`	Further arguments to be passed on to scale_x_continuous()

Value

A ggproto object inheriting from `Scale`

shift_level_max *Sliding window features*

Description

Computes feature of a time series based on sliding (overlapping) windows. `shift_level_max` finds the largest mean shift between two consecutive windows. `shift_var_max` finds the largest var shift between two consecutive windows. `shift_kl_max` finds the largest shift in Kulback-Leibler divergence between two consecutive windows.

Usage

```
shift_level_max(x, .size = NULL, .period = 1)

shift_var_max(x, .size = NULL, .period = 1)

shift_kl_max(x, .size = NULL, .period = 1)
```

Arguments

`x`	a univariate time series
`.size`	size of sliding window, if NULL `.size` will be automatically chosen using `.period`
`.period`	The seasonal period (optional)

Details

Computes the largest level shift and largest variance shift in sliding mean calculations

Value

A vector of 2 values: the size of the shift, and the time index of the shift.

Author(s)

Earo Wang, Rob J Hyndman and Mitchell O'Hara-Wild

---

stat_arch_lm *ARCH LM Statistic*

Description

Computes a statistic based on the Lagrange Multiplier (LM) test of Engle (1982) for autoregressive conditional heteroscedasticity (ARCH). The statistic returned is the R^2 value of an autoregressive model of order `lags` applied to x^2.

Usage

```
stat_arch_lm(x, lags = 12, demean = TRUE)
```

Arguments

`x`	a univariate time series
`lags`	Number of lags to use in the test
`demean`	Should data have mean removed before test applied?

Value

A numeric value.

Author(s)

Yanfei Kang

STL *Multiple seasonal decomposition by Loess*

Description

Decompose a time series into seasonal, trend and remainder components. Seasonal components are estimated iteratively using STL. Multiple seasonal periods are allowed. The trend component is computed for the last iteration of STL. Non-seasonal time series are decomposed into trend and remainder only. In this case, supsmu is used to estimate the trend. Optionally, the time series may be Box-Cox transformed before decomposition. Unlike stl, mstl is completely automated.

Usage

```
STL(formula, iterations = 2, ...)
```

Arguments

formula	Decomposition specification (see "Specials" section).
iterations	Number of iterations to use to refine the seasonal component.
...	Other arguments passed to stats::stl().

Value

A fabletools::dable() containing the decomposed trend, seasonality and remainder from the STL decomposition.

Specials

trend: The trend special is used to specify the trend extraction parameters.

```
trend(window, degree, jump)
```

window	The span (in lags) of the loess window, which should be odd. If NULL, the default, nextodd(ceiling((1.5*period) / (
degree	The degree of locally-fitted polynomial. Should be zero or one.
jump	Integers at least one to increase speed of the respective smoother. Linear interpolation happens between every jumpt

season: The season special is used to specify the season extraction parameters.

```
season(period = NULL, window = 13, degree, jump)
```

period	The periodic nature of the seasonality. This can be either a number indicating the number of observations in each se
window	The span (in lags) of the loess window, which should be odd. If the window is set to "periodic" or Inf, the seasona
degree	The degree of locally-fitted polynomial. Should be zero or one.
jump	Integers at least one to increase speed of the respective smoother. Linear interpolation happens between every jumpt

lowpass: The lowpass special is used to specify the low-pass filter parameters.

```
lowpass(window, degree, jump)
```

`window`	The span (in lags) of the loess window of the low-pass filter used for each subseries. Defaults to the smallest odd int
`degree`	The degree of locally-fitted polynomial. Must be zero or one.
`jump`	Integers at least one to increase speed of the respective smoother. Linear interpolation happens between every `jumpt`

References

R. B. Cleveland, W. S. Cleveland, J.E. McRae, and I. Terpenning (1990) STL: A Seasonal-Trend Decomposition Procedure Based on Loess. Journal of Official Statistics, 6, 3–73.

See Also

`stl`, `supsmu`

Examples

```
as_tsibble(USAccDeaths) %>%
  model(STL(value ~ trend(window = 10))) %>%
  components()
```

`unitroot_kpss` *Unit root tests*

Description

Performs a test for the existence of a unit root in the vector.

Usage

```
unitroot_kpss(x, type = c("mu", "tau"), lags = c("short", "long", "nil"), ...)

unitroot_pp(
  x,
  type = c("Z-tau", "Z-alpha"),
  model = c("constant", "trend"),
  lags = c("short", "long"),
  ...
)
```

Arguments

`x`	A vector to be tested for the unit root.
`type`	Type of deterministic part.
`lags`	Maximum number of lags used for error term correction.
`...`	Arguments passed to unit root test function.
`model`	Determines the deterministic part in the test regression.

Details

unitroot_kpss computes the statistic for the Kwiatkowski et al. unit root test with linear trend and lag 1.

unitroot_pp computes the statistic for the Z-tau version of Phillips & Perron unit root test with constant trend and lag 1.

Value

A vector of numeric features for the test's statistic and p-value.

See Also

urca::ur.kpss()

urca::ur.pp()

---

unitroot_ndiffs *Number of differences required for a stationary series*

Description

Use a unit root function to determine the minimum number of differences necessary to obtain a stationary time series.

Usage

```
unitroot_ndiffs(
  x,
  alpha = 0.05,
  unitroot_fn = ~unitroot_kpss(.)["kpss_pvalue"],
  differences = 0:2,
  ...
)

unitroot_nsdiffs(
  x,
  alpha = 0.05,
  unitroot_fn = ~feat_stl(., .period)[2] < 0.64,
  differences = 0:2,
  .period = 1,
  ...
)
```

Arguments

`x`	A vector to be tested for the unit root.
`alpha`	The level of the test.
`unitroot_fn`	A function (or lambda) that provides a p-value for a unit root test.
`differences`	The possible differences to consider.
`...`	Additional arguments passed to the `unitroot_fn` function
`.period`	The period of the seasonality.

Details

Note that the default 'unit root function' for `unitroot_nsdiffs()` is based on the seasonal strength of an STL decomposition. This is not a test for the presence of a seasonal unit root, but generally works reasonably well in identifying the presence of seasonality and the need for a seasonal difference.

Value

A numeric corresponding to the minimum required differences for stationarity.

---

var_tiled_var *Time series features based on tiled windows*

Description

Computes feature of a time series based on tiled (non-overlapping) windows. Means or variances are produced for all tiled windows. Then stability is the variance of the means, while lumpiness is the variance of the variances.

Usage

```
var_tiled_var(x, .size = NULL, .period = 1)

var_tiled_mean(x, .size = NULL, .period = 1)
```

Arguments

`x`	a univariate time series
`.size`	size of sliding window, if NULL `.size` will be automatically chosen using `.period`
`.period`	The seasonal period (optional)

Value

A numeric vector of length 2 containing a measure of lumpiness and a measure of stability.

Author(s)

Earo Wang and Rob J Hyndman

Index

∗ **datasets**
    ljung_box, 20
∗ **package**
    feasts-package, 2

ACF, 3, *17*
ACF(), *5*
autoplot.tbl_cf, 5
autoplot.tbl_cf(), *4*

box_pierce *(*ljung_box*)*, 20

CCF *(*ACF*)*, 3
CCF(), *5*
classical_decomposition, 5
coef_hurst, 7

fabletools::dable(), *6*, *24*
fabletools::forecast.mdl_df(), *12*
feasts *(*feasts-package*)*, 2
feasts-package, 2
feat_acf, 7
feat_intermittent, 8
feat_pacf, 9
feat_spectral, 9
feat_stl, 11

generate.stl_decomposition, 11
gg_arma, 12
gg_lag, 13
gg_season, 14
gg_season(), *16*
gg_subseries, 15
gg_tsdisplay, 16
gg_tsdisplay(), *18*
gg_tsresiduals, 18
grid::arrow(), *13*
guerrero, 19

ljung_box, 20
longest_flat_spot, 21

n_crossing_points, 21
n_flat_spots *(*longest_flat_spot*)*, 21

PACF *(*ACF*)*, 3
PACF(), *5*
plot.ts, *17*
portmanteau_tests *(*ljung_box*)*, 20

scale_cf_lag, 22
scale_x_cf_lag *(*scale_cf_lag*)*, 22
shift_kl_max *(*shift_level_max*)*, 22
shift_level_max, 22
shift_var_max *(*shift_level_max*)*, 22
spec.ar, *10*, *17*
stat_arch_lm, 23
stats::acf(), *4*, *7*, *9*
stats::Box.test(), *20*
stats::ccf(), *4*
stats::pacf(), *4*, *7*, *9*
stats::spec.ar(), *10*
stats::stl(), *11*, *24*
STL, 24
stl, *24*, *25*
supsmu, *24*, *25*

unitroot_kpss, 25
unitroot_ndiffs, 26
unitroot_nsdiffs *(*unitroot_ndiffs*)*, 26
unitroot_pp *(*unitroot_kpss*)*, 25
urca::ur.kpss(), *26*
urca::ur.pp(), *26*

var_tiled_mean *(*var_tiled_var*)*, 27
var_tiled_var, 27