Package ‘deseats’

March 16, 2026

Type Package

Title Data-Driven Locally Weighted Regression for Trend and Seasonality in TS

Version 1.1.2

Date 2026-03-10

Maintainer Dominik Schulz <dominik.schulz@uni-paderborn.de>

Description Various methods for the identification of trend and seasonal components in time series (TS) are provided. Among them is a data-driven locally weighted regression approach with automatically selected bandwidth for equidistant short-memory time series. The approach is a combination / extension of the algorithms by Feng (2013) <doi:10.1080/02664763.2012.740626> and Feng, Y., Gries, T., and Fritz, M. (2020) <doi:10.1080/10485252.2020.1759598> and a brief description of this new method is provided in the package documentation. Furthermore, the package allows its users to apply the base model of the Berlin procedure, version 4.1, as described in Speth (2004) <https://www.destatis.de/DE/Methoden/Saisonbereinigung/BV41-methodenbericht-Heft3_2004.pdf?__blob=publicationFile>. Permission to include this procedure was kindly provided by the Federal Statistical Office of Germany.

Imports Rcpp (>= 1.0.6), ggplot2, stats, graphics, animation, utils, shiny, tools, zoo, future, furrr, future.apply, progressr, purrr, rlang, tidyr

License GPL-3

Encoding UTF-8

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 2.10), methods

Suggests badger, knitr, rmarkdown, smoots, testthat (>= 3.0.0)

RoxygenNote 7.3.3

LazyData true

Collate 'AttachMessage.R' 'RcppExports.R' 'generics.R' 'helper_functions.R' 'subplot_functions.R' 'accuracy_measures.R' 'class-smoothing_options.R' 'class-decomp.R' 'class-deseats.R' 'class-s_semiarma.R' 'class-bv41.R' 'class-hfilter.R' 'class-lmdecomp.R' 'class-madecomp.R' 'class-llindecomp.R' 'data_documentation.R' 'fitting_functions.R' 'runDecomposition.R' 'class-deseats_fc.R' 'forecasting_functions.R' 'bwidth_bootstrap.R' 'gain_function.R' 'linear_filters.R' 'ts_conversion.R' 'hA_calc.R' 'arima_no_warn.R' 'deseats-package.R' 'seasonplot.R'

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Yuanhua Feng [aut] (Paderborn University, Germany), Dominik Schulz [aut, cre] (Paderborn University, Germany)

Repository CRAN

Date/Publication 2026-03-16 11:30:02 UTC

Contents

`deseats-package` *Deseasonalize Time Series*

Description

A library of decomposition methods for equidistant time series with trend and seasonality.

Details

`deseats` is an R package for the decomposition of equidistant time series with trend and seasonality. First and foremost, an own algorithm for bandwidth selection in locally weighted regression of such time series (with short-range dependence) is implemented that is based on both the algorithms by Feng (2013) and Feng et al. (2020). For comparison, a simplified version of the BV4.1 (Berlin Procedure 4.1, Speth, 2004), is implemented as well that allows to implement the BV4.1 base model (trend component + seasonality component + irregular component) without any of the additional BV4.1 components (such as the calendar component). Permission to include the BV4.1 base model procedure was kindly provided by the Federal Statistical Office of Germany.

Main Functions

The main functions of the package are:

`deseats`**:** locally weighted regression with automatically selected bandwidth for decomposition,

`BV4.1`**:** BV4.1 base model for decomposition,

`lm_decomp`**:** ordinary least squares for decomposition,

`llin_decomp`**:** local linear regression for decomposition,

`ma_decomp`**:** moving averages for decomposition,

`hamilton_filter`**:** the time series filter by Hamilton.

Datasets

The package includes a few datasets. Follow the corresponding links to the documentation of the datasets to find additional information including the sources.

`CIVLABOR`**:** civilian labor force level in the USA.

`CONSUMPTION`**:** real final consumption expenditure for Australia.

`COVID`**:** new COVID-19 cases in Germany.

`DEATHS`**:** recorded number of deaths in Germany.

`ENERGY`**:** production and distribution of electricity, gas, steam and air conditioning in Germany.

`EXPENDITURES`**:** consumption expenditures in the USA.

`GDP`**:** GDP of the USA.

`HOUSES`**:** new one family houses sold in the USA.

`LIVEBIRTHS`**:** recorded number of livebirths in Germany.

`NOLABORFORCE`**:** number of persons in the USA not belonging to the labor force.

`RAINFALL`**:** average amount of rain in Germany.

`RETAIL`**:** Retail sale volume in Germany.

`SAVINGS`**:** savings of private households in Germany.

`SUNSHINE`**:** average hours of sunshine in Germany.

`TEMPERATURE`**:** average temperature in Germany.

License

The package is distributed under the General Public License v3 ([GPL-3](https://tldrlegal.com/license/gnu-general-public-license-v3-(gpl-3))).

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator
- Yuanhua Feng (Department of Economics, Paderborn University), Author

References

- Feng, Y. (2013). An iterative plug-in algorithm for decomposing seasonal time series using the Berlin Method. Journal of Applied Statistics, 40(2): 266-281. DOI: 10.1080/02664763.2012.740626.
- Feng, Y., Gries. T, and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32(2): 510-533. DOI: 10.1080/10485252.2020.1759598.
- Speth, H.-T. (2004). Komponentenzerlegung und Saisonbereinigung ökonomischer Zeitreihen mit dem Verfahren BV4.1. Methodenberichte 3. Statistisches Bundesamt. URL: https://www.destatis.de/DE/Methoden/ methodenbericht-Heft3_2004.pdf?__blob=publicationFile.

`animate` *Automatic Creation of Animations*

Description

A generic that is the basis for methods that allow the user to create specific animations on-the-fly.

Usage

```
animate(object, ...)
```

Arguments

`object`	the input object.
`...`	currently of no use; included for future compatibility.

Details

A generic that can be extended by methods to automatically create animations based on certain objects.

Value

This generic does not return anything and is just the basis for more sophisticated methods.

animate,deseats-method

Animate Locally Weighted Regression Results

Description

The results of locally weighted regression results acquired through decomposition of seasonal time series via the function deseats can be animated automatically.

Usage

```
## S4 method for signature 'deseats'
animate(
  object,
  col.obs = "grey74",
  col.fit = "red",
  col.weights = "#00D40E",
  col.window = "deepskyblue4",
  col.spot = "orange",
  save = NULL,
  xlab = "Time",
  ylab1 = "Estimated trend + seasonality",
  ylab2 = "Active kernel weights",
  main = NULL,
  ...
)
```

Arguments

`object`	an object of class `"deseats"`.
`col.obs`	the color to use for the observations.
`col.fit`	the color to use for the fitted values.
`col.weights`	the color to use for the active kernel weights.
`col.window`	the color to use for the window defined through the bandwidth
`col.spot`	the color to highlight the estimation time point.
`save`	whether to save the animation or not; for `NULL`, it will not be saved; to save it in the current working directory, use either `save = "PDF"` or `save = "HTML"`, which also specifies the file format.
`xlab`	the label for the x-axis.
`ylab1`	the label for the y-axis on the left-hand side.
`ylab2`	the label for the second y-axis on the right-hand side.
`main`	the plot title.
`...`	currently without use; implemented for possible future compatibility.

Details

deseats estimation results are automatically animated through this method. It shows the observed series together with fitted values (trend + seasonality), the smoothing window, the fitted values from the local regression, and the active kernel weights.

Value

The function returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
### Creating the animation might take a while
Xt <- log(EXPENDITURES)
smoothing_options <- set_options(order_poly = 3)
est <- deseats(Xt, smoothing_options = smoothing_options)
animate(est)
```

`arma_to_ar` *AR Representation of an ARMA Model*

Description

Calculate the coefficients of the infinite-order AR-representation of a given ARMA model.

Usage

```
arma_to_ar(ar = numeric(0), ma = numeric(0), max_i = 1000)
```

Arguments

`ar`	a numeric vector with the AR parameters of the ARMA model; should be ordered from a_1 to a_p (see Details).
`ma`	a numeric vector with the MA parameters of the ARMA model; should be ordered from b_1 to b_q (see Details).
`max_i`	a single numeric value that indicates how many coefficients should be returned; returned will be `max_i + 1` coefficients (the coefficient for index 0 is also returned as the first value).

Details

Consider an ARMA model

$$X_t = a_1 X_{t-1} + ... + a_p X_{t-p} + b_1 \epsilon_{t-1} + ... + b_q \epsilon_{t-q} + \epsilon_t,$$

where $a_1, ..., a_p$ and $b_1, ..., b_q$ are its real-valued coefficients. The function `arma_to_ar()` uses these coefficients as input to calculate the coefficients of the truncated infinite-order AR-representation of the model defined through these coefficients. Note that the stationarity and invertibility of the model defined through the provided coefficients is not being checked.

NOTE:

This function implements C++ code by means of the Rcpp and RcppArmadillo packages for better performance.

Value

A numeric vector is returned.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
ar <- c(1.2, -0.4)
ma <- c(0.5)
arma_to_ar(ar = ar, ma = ma, max_i = 100)
```

`arma_to_ma` *MA Representation of an ARMA Model*

Description

Calculate the coefficients of the infinite-order MA-representation of a given ARMA model.

Usage

```
arma_to_ma(ar = numeric(0), ma = numeric(0), max_i = 1000)
```

Arguments

`ar`	a numeric vector with the AR parameters of the ARMA model; should be ordered from a_1 to a_p (see Details).
`ma`	a numeric vector with the MA parameters of the ARMA model; should be ordered from b_1 to b_q (see Details).
`max_i`	a single numeric value that indicates how many coefficients should be returned; returned will be `max_i + 1` coefficients (the coefficient for index 0 is also returned as the first value).

Details

Consider an ARMA model

$$X_t = a_1 X_{t-1} + ... + a_p X_{t-p} + b_1 \epsilon_{t-1} + ... + b_q \epsilon_{t-q} + \epsilon_t,$$

where $a_1, ..., a_p$ and $b_1, ..., b_q$ are its real-valued coefficients. The function `arma_to_ar()` uses these coefficients as input to calculate the coefficients of the truncated infinite-order MA-representation of the model defined through these coefficients. Note that the stationarity and invertibility of the model defined through the provided coefficients is not being checked.

NOTE:

This function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

A numeric vector is returned.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
ar <- c(1.2, -0.4)
ma <- c(0.5)
arma_to_ma(ar = ar, ma = ma, max_i = 100)
```

`autoplot,decomp-method` *Plot Method for Decomposition Results in the Style of ggplot2*

Description

This is method for producing various plots of the decomposition results returned by this package.

Usage

```
## S4 method for signature 'decomp'
autoplot(object, which = NULL, ...)
```

Arguments

`object`	an object returned by the function `deseats`.
`which`	various plots can be selected either via a keyword or a number; enter `"facets"` or 1 to show a facet plot of the estimated time series components; enter `"observations"` or 2 to show the input time series; enter `"fitted"` or 3 to show the observations alongside the estimated trend with seasonality; enter `"detailed_fit"` or 4 to show the observations together with the fitted values and the trend; enter `"trend_season"` or 5 to show the observations together with the trend and with the seasonality (the latter shown around the series mean); enter `"residuals"` or 6 to plot the both detrended and seasonally adjusted series; use 7 or `"deseasonalized"` to show the seasonally adjusted series; enter 8 or `"detrended"` to plot the detrended series; the default is `which = NULL` which then lets you select a plot interactively in the R console.
`...`	no purpose and only implemented for compatibility.

Details

Create predefined standard plots of the decomposition objects returned by the `deseats` package, e.g. returned by the function `deseats`. Plots are created in the ggplot2 plot style. The type of plot can be chosen either interactively from the console, or the argument `which` can be used to directly select the kind of plot to create (see also the description of the argument `which`) within the function call.

If plot type 5 (`which = 5`) is selected, the estimated seasonality will be displayed around the mean of the observations by default. Setting the additional argument `s_around` to some other value, will lead to the seasonality being displayed around that constant value.

Value

A ggplot2-graphic object is returned, i.e. an object of classes `"gg"` and `"ggplot"`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
Xt <- log(EXPENDITURES)
est <- deseats(Xt)
autoplot(est, which = 3)
```

autoplot,deseats_fc-method
`ggplot2` *Plot Method for Class* `"deseats_fc"`

Description

Create `ggplot2` R plots for forecasting objects of class `"deseats_fc"`.

Usage

```
## S4 method for signature 'deseats_fc'
autoplot(object, ...)
```

Arguments

`object`	an object of class `"deseats_fc"`, for example generated by a call to `predict,s_semiarma-method`.
`...`	currently without use; implemented for compatibility.

Details

This is a plot method to visualize the forecasting results for a Seasonal Semi-ARMA model. Common plot arguments can be implemented to change the appearance.

Value

This method returns a `ggplot2` plot object, i.e. an object of classes `"gg"` and `"ggplot"`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- s_semiarma(log(EXPENDITURES))
fc <- predict(est, n.ahead = 4)
fc_e <- expo(fc)
autoplot(fc_e)
```

autoplot,hfilter-method
ggplot2 *Plot Method for the Results of a Hamilton Filter*

Description

Visualize the results of an applied Hamilton filter in the style of ggplot2.

Usage

```
## S4 method for signature 'hfilter'
autoplot(object, which = NULL, ...)
```

Arguments

object	an object of class "hfilter", as returned by the function hamilton_filter.
which	either a string or a number can be entered to select a plot type from the function call; options are (1) a facet plot of the components, (2) the observed time series, (3) the observations together with the fitted values, and (4) the residuals; for which = NULL, the plot type can be selected interactively in the console.
...	currently without use, implemented for compatibility.

Value

This method returns a ggplot2 plot object, i.e. an object of classes "gg" and "ggplot".

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- hamilton_filter(log(EXPENDITURES))
autoplot(est, which = 3, col = c(1, 6))
autoplot(est, which = 4)
```

BV4.1 *Trend and Seasonality Estimation Using the Berlin Procedure 4.1*

Description

Economic data can be decomposed into a trend, a seasonal and a remainder component using the Berlin procedure 4.1 (German: Berliner Verfahren 4.1), as used by the National Statistical Office of Germany. Currently with this version of the R package, only the trend and seasonal components can be estimated following BV4.1. All further component estimations, for example the estimation of the calendar component, of the official procedure BV4.1 are not yet implemented. The function supports quarterly and monthly data.

Usage

```
BV4.1(yt, type = NULL)
```

Arguments

`yt`	a time series object of class `ts` or an object that can be converted into such an object with `as.ts`.
`type`	a single character value that indicates, whether the data was quarterly (`"quarterly"`) or monthly (`"monthly"`) observed; the default is `"monthly"`; if a time series object is passed to `yt`, the value for this argument will be automatically selected according to the frequency in `yt`.

Details

The BV4.1 base model is as follows:

trend and seasonality are estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + s(x_t) + \epsilon_t,$$

where y_t is the observed time series with $t = 1, ...n$, $x_t = t/n$ is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth trend function, $s(x_t)$ is a (slowly changing) seasonal component with seasonal period p_s and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ that are furthermore assumed to be independent but identically distributed (i.i.d.).

It is assumed that m and s can be approximated locally by a polynomial of small order and by a trigonometric polynomial, respectively. Through locally weighted regression, m and s can therefore be estimated suitably.

The advantage of the Berlin Procedure 4.1 (BV4.1) is that it makes use of fixed filters based on locally weighted regression (both with a weighted mixture of local linear and local cubic components for the trend) at all observation time points. Thus, BV4.1 results in fixed weighting matrices both for the trend estimation step and for the seasonality estimation step that can be immediately applied to all economic time series. Those matrices are saved internally in the package and when applying `BV4.1`, only weighted sums of the observations (with already obtained weights) have to be obtained at all time points. Thus, this procedure is quite fast.

Permission to include the BV4.1 base model procedure was kindly provided by the Federal Statistical Office of Germany.

Value

An S4 object with the following elements is returned.

`decomp` An object of class `"mts"` that consists of the decomposed time series data.

`frequency` the frequency of the time series.

`ts_name` the object name of the initially provided time series object.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

References

- Speth, H.-T. (2004). Komponentenzerlegung und Saisonbereinigung ökonomischer Zeitreihen mit dem Verfahren BV4.1. Methodenberichte 3. Statistisches Bundesamt. URL: https://www.destatis.de/DE/Methoden/
methodenbericht-Heft3_2004.pdf?__blob=publicationFile.

Examples

```
Xt <- log(EXPENDITURES)
est <- BV4.1(Xt)
est
```

`bwidth,deseats-method` *Retrieve the Used Bandwidth from an Estimation Object*

Description

If either `deseats` or `s_semiarma` are used to fit a model to time series data, this method retrieves the applied bandwidth from the output object.

Usage

```
## S4 method for signature 'deseats'
bwidth(object)

## S4 method for signature 's_semiarma'
bwidth(object)
```

Arguments

`object`	an object either of class `"deseats"` or `"s_semiarma"`.

Details

When applying deseats or `link{s_semiarma}`, one approach is to let the functions automatically choose a bandwidth for locally weighted regression. Using this method, the applied bandwidth can be retrieved.

Value

A numeric vector of length one that represents the bandwidth used in the smoothing procedure is returned.

Examples

```
Xt <- log(EXPENDITURES)
smoothing_options <- set_options(order_poly = 3)
est <- deseats(Xt, smoothing_options = smoothing_options)
bwidth(est)
```

`bwidth_confint` *Bootstrapping Confidence Intervals for Locally Weighted Regression Bandwidths*

Description

A stationary block bootstrap is applied to resample from a time series that was decomposed into a trend, a seasonal component and a remainder by means of data-driven local polynomial regression with automatically selected bandwidth. Bandwidth re-estimation from each bootstrapped sample results in confidence bounds for the bandwidth.

Usage

```
bwidth_confint(
  nonpar_model,
  blocklen = NULL,
  npaths = 1000,
  parallel = TRUE,
  num_cores = future::availableCores() - 1,
  ...
)
```

Arguments

`nonpar_model`	the object with the nonparametric trend and seasonality estimation results returned by for example the function deseats.

`blocklen`	a numerical vector of length one that indicates the average block length to be drawn from the detrended series; the default is `NULL`, which means `8` for quarterly and `24` for monthly data; selecting a suitable expected blocklength and checking the sensitivity of the blocklength are left for the user.
`npaths`	a numeric vector of length one that indicates the number of bootstrap paths; the default is `npaths = 1000`.
`parallel`	a logical vector of length one that indicates whether or not to employ parallel programming for the resampling and the subsequently data-driven bandwidth estimations from the bootstrapped samples; the default is `patrallel = TRUE`.
`num_cores`	a numeric vector of length one that indicates the number of CPU cores to use for parallel programming, if `parallel = TRUE`; the default is `num_cores = future::availableCores() - 1`.
`...`	further arguments to pass to `deseats`.

Details

Confidence bounds for the bandwidth in local polynomial regression for identifying the trend in a trend-stationary short-memory time series are obtained via a block bootstrap, which ensures that no specific model assumptions are required for the detrended series.

This function makes use of the `future` parallel programming framework to ensure exactly the same results regardless of whether sequential or parallel programming, and then also regardless of the number of workers, is employed.

Value

A list with the following elements is returned.

`conf` A vector with named elements that gives the original bandwidth estimate as well as the bootstrapped bounds of the 95 and 99 percent confidence intervals of the bandwidth.

`bwidth_estimates` a vector with all the obtained bandwidths for the bootstrapped series.

`se_bwidth` the sample standard deviation of `bwidth_estimates`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
xt <- log(EXPENDITURES)
est <- deseats(xt, set_options(order_poly = 3))
conf <- bwidth_confint(est, npaths = 200, num_cores = 2)
conf
```

CIVLABOR *Monthly Civilian Labor Force Level in the USA*

Description

A `ts` object that contains the monthly observed civilian labor force level in the USA (in millions of persons) from January 1948 to December 2019. The object contains 864 observations.

Usage

```
CIVLABOR
```

Format

An object of class `ts` of length 864.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2022-09-01) and then transformed into a time series object using R.

`https://fred.stlouisfed.org/series/LNU01000000`

CONSUMPTION *Quarterly Real Final Consumption Expenditure for Australia*

Description

A `ts` object that contains the quarterly real final consumption expenditure (in thousands of domestic currency) from the third quarter of 1959 to the last quarter in 2019. The object contains 242 observations.

Usage

```
CONSUMPTION
```

Format

An object of class `ts` with 242 rows and 1 columns.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-09-26) and then transformed into a time series object using R.

`https://fred.stlouisfed.org/series/NCRNSAXDCAUQ`

`COVID` *Daily Confirmed New COVID-19 Cases in Germany*

Description

A `ts` object that contains the daily confirmed new COVID-19 cases in Germany (in thousands of cases) from June 2021 to November 2021. The object contains 183 observations. The time series object is created as a time series object with frequency 7, i.e. the time unit is in calendar weeks of the year 2021 and not in years.

Usage

```
COVID
```

Format

An object of class `ts` of length 183.

Source

The data was obtained from the COVID-19 Data Hub (accessed: 2023-09-25) and then transformed into a time series object using R.

https://covid19datahub.io/

`create.gain` *Create Gain Function from a Linear Time Series Filter*

Description

This function takes a coefficient series of a linear time series filter as an input and then returns the corresponding gain function as an R function.

Usage

```
create.gain(filter.coefs = c(1), zero.at = ceiling(length(filter.coefs)/2))
```

Arguments

`filter.coefs`	a numeric vector with the filter coefficients ordered by coefficient index; see details for more info.
`zero.at`	a numeric vector of length one that indicates the position of the coefficient for the present observation in `filter.coefs`; by default, the position is in the middle or just below the midpoint.

Details

This is a functional. The function returns a function that represents the gain function for the input filter `filter.coefs`. The returned function only has the argument `lambda`, which is the frequency for which the value of the gain function should be obtained.

Let (y_t) be the input series and (c_j) the linear filter; then the element c_j is the weight assigned to y_{t-j}. The corresponding index j is important for the order of the argument `filter.coefs`.

Note: This function is deprecated. Switch to `create_gain()`.

Value

The function returns a "gain function" function that has the numeric argument `lambda` only that represents frequencies to calculate the values of the gain function for.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
# Moving average with smoothing over three values
a <- 1 / 3
gain_ma <- create.gain(rep(a, 3))
lambda <- seq(0, 0.5, 0.001)
GF <- gain_ma(lambda)
plot(lambda, GF, type = "l")



# First differences filter
b <- c(1, -1)
gain_diff <- create.gain(b)
lambda <- seq(0, 0.5, 0.001)
GF2 <- gain_diff(lambda)
plot(lambda, GF2, type = "l")


# For a fully data-driven local linear trend +
# trigonometric polynomial seasonality
# (Note: we get various filters for different observation time points)

xt <- EXPENDITURES
est <- deseats(log(xt), set_options(order_poly = 3))
ws <- est@weights[, , "Combined"]
l <- (length(ws[, 1]) - 1) / 2

lambda <- seq(0, 0.5, 0.001)
mat <- matrix(0, ncol = length(lambda), nrow = l + 1)
colF <- colorRampPalette(c("deepskyblue4", "deepskyblue"))
cols <- colF(l)
```

```
for (j in 1:(l + 1)) {

  gainF <- create.gain(ws[j, ], zero.at = j)
  mat[j, ] <- gainF(lambda)

}

matplot(lambda, t(mat), type = paste0(rep("l", l + 1), collapse = ""),
        lty = rep(1, l + 1), col = cols)
title(
  main = paste0(
    "Gain functions for the applied data-driven locally weighted ",
    "regression\napproach at boundary points and the first interior ",
    "point"
  )
)

# Same example as before but not for the trend but for the detrending filters
# (Note: we get various filters for different observation time points)

ll <- l * 2 + 1
mat2 <- mat

for (j in 1:(l + 1)) {

  zero.vec <- rep(0, ll)
  zero.vec[[j]] <- 1
  gainF <- create.gain(zero.vec - ws[j, ], zero.at = j)
  mat2[j, ] <- gainF(lambda)

}

matplot(lambda, t(mat2), type = paste0(rep("l", l + 1), collapse = ""),
        lty = rep(1, l + 1), col = cols)
title(
  main = paste0(
    "Gain functions for the applied data-driven detrending filter\n",
    "at boundary points and the first interior ",
    "point"
  )
)
```

create_gain *Create Gain Function from a Linear Time Series Filter*

Description

This function takes a coefficient series of a linear time series filter as an input and then returns the corresponding gain function as an R function.

Usage

```
create_gain(filter.coefs = c(1), zero.at = ceiling(length(filter.coefs)/2))
```

Arguments

`filter.coefs`	a numeric vector with the filter coefficients ordered by coefficient index; see details for more info.
`zero.at`	a numeric vector of length one that indicates the position of the coefficient for the present observation in `filter.coefs`; by default, the position is in the middle or just below the midpoint.

Details

This is a functional. The function returns a function that represents the gain function for the input filter `filter.coefs`. The returned function only has the argument `lambda`, which is the frequency for which the value of the gain function should be obtained.

Let (y_t) be the input series and (c_j) the linear filter; then the element c_j is the weight assigned to y_{t-j}. The corresponding index j is important for the order of the argument `filter.coefs`.

Value

The function returns a "gain function" function that has the numeric argument `lambda` only that represents frequencies to calculate the values of the gain function for.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
# Moving average with smoothing over three values
a <- 1 / 3
gain_ma <- create_gain(rep(a, 3))
lambda <- seq(0, 0.5, 0.001)
GF <- gain_ma(lambda)
plot(lambda, GF, type = "l")



# First differences filter
b <- c(1, -1)
gain_diff <- create_gain(b)
lambda <- seq(0, 0.5, 0.001)
GF2 <- gain_diff(lambda)
```

```
plot(lambda, GF2, type = "l")


# For a fully data-driven local linear trend +
# trigonometric polynomial seasonality
# (Note: we get various filters for different observation time points)

xt <- EXPENDITURES
est <- deseats(log(xt), set_options(order_poly = 3))
ws <- est@weights[, , "Combined"]
l <- (length(ws[, 1]) - 1) / 2

lambda <- seq(0, 0.5, 0.001)
mat <- matrix(0, ncol = length(lambda), nrow = l + 1)
colF <- colorRampPalette(c("deepskyblue4", "deepskyblue"))
cols <- colF(l)

for (j in 1:(l + 1)) {

  gainF <- create_gain(ws[j, ], zero.at = j)
  mat[j, ] <- gainF(lambda)

}

matplot(lambda, t(mat), type = paste0(rep("l", l + 1), collapse = ""),
        lty = rep(1, l + 1), col = cols)
title(
  main = paste0(
    "Gain functions for the applied data-driven locally weighted ",
    "regression\napproach at boundary points and the first interior ",
    "point"
  )
)

# Same example as before but not for the trend but for the detrending filters
# (Note: we get various filters for different observation time points)

ll <- l * 2 + 1
mat2 <- mat

for (j in 1:(l + 1)) {

  zero.vec <- rep(0, ll)
  zero.vec[[j]] <- 1
  gainF <- create_gain(zero.vec - ws[j, ], zero.at = j)
  mat2[j, ] <- gainF(lambda)

}

matplot(lambda, t(mat2), type = paste0(rep("l", l + 1), collapse = ""),
        lty = rep(1, l + 1), col = cols)
title(
  main = paste0(
    "Gain functions for the applied data-driven detrending filter\n",
    "at boundary points and the first interior ",
    "point"
  )
)
```

DEATHS — *Monthly Deaths in Germany*

Description

A `ts` object that contains the monthly observed deaths in Germany (in thousands of cases) from January 1990 to April 2022. The object contains 388 observations.

Usage

```
DEATHS
```

Format

An object of class `ts` of length 388.

Source

The data was obtained from the databank "Genesis" of the National Statistical Office of Germany (accessed: 2022-07-21) and then transformed into a time series object using R.

https://www-genesis.destatis.de/genesis/online?operation=abruftabelleBearbeiten&levelindex=2&levelid=1658407055248&auswahloperation=abruftabelleAuspraegungAuswaehlen&auswahlverzeichnis=ordnungsstruktur&auswahlziel=werteabruf&code=12613-0005&auswahltext=&werteabruf=Werteabruf#abreadcrumb

deseats — *Locally Weighted Regression for Trend and Seasonality in Equidistant Time Series under Short Memory*

Description

Simultaneously estimate the trend and the seasonality via locally weighted regression in an equidistant time series under short memory. The default setting uses an iterative plug-in algorithm for identifying the asymptotically globally optimal bandwidth for smoothing.

Usage

```
deseats(
  y,
  smoothing_options = set_options(),
  bwidth_start = NULL,
  inflation_rate = c("optimal", "naive"),
  correction_factor = FALSE,
  autocor = TRUE,
  drop = NULL,
  error_model = c("free", "ARMA"),
  nar_lim = c(0, 3),
  nma_lim = c(0, 3),
  arma_mean = FALSE,
  inf_criterion = c("bic", "aic")
)
```

Arguments

`y`	a numerical vector or a time series object of class `ts` or that can be transformed with `as.ts` to an object of class `ts`; for these observations, trend and seasonality will be obtained.
`smoothing_options`	an S4 object of class `smoothing_options`, which is returned by the function `set_options`; it includes details about the options to consider in the locally weighted regression such as the order of polynomial and the bandwidth for smoothing among others.
`bwidth_start`	a single numeric value that is only relevant if the slot `bwidth` in `smoothing_options` is set to `NA`; as the bandwidth will then be selected automatically, `bwidth_start` sets the initial bandwidth for the algorithm; the default, `bwidth_start = NULL`, corresponds to `bwidth_start = 0.1` for a local linear trend and to `bwidth_start = 0.2` for a local cubic trend.
`inflation_rate`	a character vector of length one that indicates, which inflation rate to use in the bandwidth selection; for a local linear trend, we have `inflation_rate = "optimal"` as the default, for a local cubic trend it is `inflation_rate = "naive"`, which correspond to inflation rates of 5/7 and 9/13, respectively.
`correction_factor`	A logical vector of length one; theoretically, a larger bandwidth to estimate the sum of autocovariances from residuals of pilot trend and seasonality estimates is advisable than for estimating trend and seasonality; for `correction_factor = TRUE`, this is implemented; for `error_model = "ARMA"`, `correction_factor = FALSE` is enforced; the default is `correction_factor = FALSE`, because it was found that setting this argument to `TRUE` often overestimates the bandwidth.
`autocor`	a logical vector of length one; indicates whether to consider autocorrelated errors (`TRUE`) or independent but identically distributed errors (`FALSE`); the default is `autocor = TRUE`.
`drop`	a numeric vector of length one that indicates the proportion of the observations to not include at each boundary in the bandwidth estimation process, if a band-

	width is selected automatically; the default is `drop = NULL`, which corresponds to `drop = 0.05` for a local linear trend and to `drop = 0.1` for a local cubic trend.
`error_model`	a character vector of length one that indicates whether for `autocor = TRUE` the sum of autocovariances of the errors is obtained purely nonparametrically (`"free"`) or whether an autoregressive moving-average (ARMA) model is assumed `"ARMA"`; the default is `error_model = "free"`.
`nar_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum AR order to check via the BIC in each iteration of the algorithm via a two-element vector.
`nma_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum MA order to check via the BIC in each iteration of the algorithm via a two-element vector.
`arma_mean`	only valid for `error_model = "ARMA"`; decide whether to include an estimate of the mean in the ARMA fitting for the detrended series.
`inf_criterion`	selects the information criterion upon which the ARMA orders for the residuals are chosen; defaults to `"bic"` for the Bayesian information criterion; `"aic"` uses the Akaike information criterion.

Details

Trend and seasonality are estimated based on the additive nonparametric regression model for an equidistant time series

$$y_t = m(x_t) + s(x_t) + \epsilon_t,$$

where y_t is the observed time series with $t = 1, ...n$, $x_t = t/n$ is the rescaled time on the interval $[0, 1]$, $m(x_t)$ is a smooth and deterministic trend function, $s(x_t)$ is a (slowly changing) seasonal component with seasonal period p_s and ϵ_t are stationary errors with $E(\epsilon_t) = 0$ and short-range dependence (see for example also Feng, 2013, for a similar model, where the stochastic term is however i.i.d.).

It is assumed that m and s can be approximated locally by a polynomial of small order and by a trigonometric polynomial, respectively. Through locally weighted regression, m and s can therefore be estimated given a suitable bandwidth.

The iterative-plug-in (IPI) algorithm, which numerically minimizes the asymptotic mean squared error (AMISE) to select a bandwidth is an extension of Feng (2013) to the case with short-range dependence in the errors. To achieve this goal, the error variance in the AMISE in Feng (2013) is replaced by the sum of autocovariances of the error process and this quantity is being estimated using a slightly adjusted version of the Bühlmann (1996) algorithm. This procedure is similar to the method described in Feng, Gries and Fritz (2020), where data-driven local polynomial regression with an automatically selected bandwidth is used to estimate the trend according to a model without seasonality and where the same adjusted Bühlmann (1996) algorithm is considered to estimate the sum of autocovariances in the error process.

Define $I[m^{(k)}] = \int_{c_b}^{d_b} [m^{(k)}(x)]^2 dx$, $\beta_{(k)} = \int_{-1}^{1} u^k K(u) du$ and $R(K) = \int_{-1}^{1} K^2(u) du$, where p is the order of the (local) polynomial considered for m, $k = p + 1$ is the order of the asymptotically equivalent kernel K for estimating m, $0 \leq c_b < d_b \leq 1$, and c_f is the variance factor, i.e. the sum of autocovariances divided by 2π.

Furthermore, we define

$$C_1 = \frac{I[m^{(k)}]\beta_{(k)}^2}{(k!)^2}$$

and

$$C_2 = \frac{2\pi c_f(d_b - c_b)[R(K) + (p_s - 1)R(W)]}{nh}$$

with h being the bandwidth, n being the number of observations and W being the weighting function considered in the weighted least squares approach, for example a second-order kernel function with support on $[-1, 1]$. The AMISE is then

$$AMISE(h) = h^{2k}C_1 + C_2.$$

The function calculates suitable estimates for c_f, the variance factor, and $I[m^{(k)}]$ over different iterations. In each iteration, a bandwidth is obtained in accordance with the AMISE that once more serves as an input for the following iteration. The process repeats until either convergence or the 40th iteration is reached. For further details on the asymptotic theory or the algorithm, please consult Feng, Gries and Fritz (2020) or Feng et al. (2019).

To apply the function, only few arguments are needed: a data input `y`, an object with smoothing options `smoothing_options` returned by `set_options` and a starting value for the relative bandwidth `bwidth_start`. Aside from `y`, each argument has a default setting. By default, a local linear trend is considered. In some cases, a local cubic trend may, however, be more suitable. For more specific information on the input arguments consult the section *Arguments*.

When applying the function, an optimal bandwidth is obtained based on the IPI algorithm proposed by Feng, Gries and Fritz (2020). In a second step, the nonparametric trend of the series and the seasonality are calculated with respect to the chosen bandwidth.

Note that with this function $m(x_t)$ and $s(x_t)$ can be estimated without a parametric model assumption for the error series. Thus, after estimating and removing the trend and the seasonality, any suitable parametric model, e.g. an ARMA(p, q) model for `errors = "autocor"`, can be fitted to the residuals (see `arima`).

Usually, a local cubic trend (`smoothing_options = set_options(order_poly = 3)`) gives more suitable results. Moreover, if the resulting bandwidth is too large, adjustments to the arguments `inflation_rate` and `drop` should be tried first in that order before any other changes to the input arguments.

The default print method for this function delivers key numbers such as the bandwidth considered for smoothing.

NOTE:

This function implements C++ code by means of the `Rcpp` and `RcppArmadillo` packages for better performance.

Value

The function returns and S4 object with the following elements (access them via `@`):

`boundary_method` the applied boundary method.

`bwidth` the globally applied bandwidth in the smoothing process; if not if no input is given in the function call, this is the automatically selected optimal bandwidth.

`decomp` An object of class `"mts"` that consists of the decomposed time series data.

`frequency` the frequency of the time series.

`kernel_fun` the second-order kernel function considered for weighting.

`order_poly` the order of polynomial considered locally for the trend.

`order_poly` the order of polynomial considered locally for the trend.

`sum_autocov` the estimated sum of autocovariances.

`ts_name` the object name of the initially provided time series object.

`weights_wfun` a matrix that gives the weights of the weighting function K at each estimation time point; ; if n is the length of the given time series and b is the applied (relative) bandwidth, then the first row of the weighting system gives the weighting function weights when estimating at $t = 1$, the second row gives the weights when estimating at $t = 2$ and so on for all left-hand side boundary points until the middle row, which contains the weights used at all interior points; the rows following the middle row contain the weights for right-hand side boundary points (the rows are ordered chronologically)

`weights` an array with many slices that represent the weighting systems for various filters; each slice is a matrix, which gives the weighting system to estimate a component, for example trend + seasonality, as a weighted average from the given time series; if n is the length of the given time series and b is the applied (relative) bandwidth, then the first row of the weighting system gives the weights to obtain estimates at $t = 1$, the second row gives the weights to obtain estimates at $t = 2$ and so on for all left-hand side boundary points until the middle row, which contains the weights used at all interior points; the rows following the middle row contain the weights for right-hand side boundary points (the rows are ordered chronologically); the slice names are `"Trend"`, `"Season"` and `"Combined"`, where `"Combined"` are the weights to estimate trend + seasonality combined.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University),
  Author and Package Creator
- Yuanhua Feng (Department of Economics, Paderborn University),
  Author

References

- Bühlmann, P. (1996). Locally Adaptive Lag-Window Spectral Estimation. Journal of Time Series Analysis, 17(3): 247-270. DOI: 10.1111/j.1467-9892.1996.tb00275.x.
- Feng, Y. (2013). An iterative plug-in algorithm for decomposing seasonal time series using the Berlin Method. Journal of Applied Statistics, 40(2): 266-281. DOI: 10.1080/02664763.2012.740626.
- Feng, Y., Gries. T, and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32(2): 510-533. DOI: 10.1080/10485252.2020.1759598.

Examples

```
Xt <- log(EXPENDITURES)
smoothing_options <- set_options(order_poly = 3)
```

```
est <- deseats(Xt, smoothing_options = smoothing_options)
est
plot(est, which = 1)
```

ENERGY — *Monthly Total Production and Distribution of Electricity, Gas, Steam, and Air Conditioning for Germany*

Description

A `ts` object that contains the monthly production and distribution of electricity gas steam, and air conditioning for Germany (Index 2015 = 100) from January 1991 to June 2023. The object contains 390 observations.

Usage

```
ENERGY
```

Format

An object of class `ts` with 390 rows and 1 columns.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-10-11) and then transformed into a time series object using R.

https://fred.stlouisfed.org/series/DEUPREND401IXOBM

EXPENDITURES — *Quarterly Personal Consumption Expenditures in the USA*

Description

A `ts` object that contains the quarterly observed personal consumption expenditures in the USA in trillions of dollars from the first quarter of 1947 to the last quarter of 2019. The object contains 292 observations.

Usage

```
EXPENDITURES
```

Format

An object of class `ts` of length 292.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2022-07-15) and then transformed into a time series object using R.

https://fred.stlouisfed.org/series/NA000349Q

`expo` *Automatic Creation of Animations*

Description

A generic that is the basis for methods that allow the user to exponentiate certain results quickly.

Usage

```
expo(object, ...)
```

Arguments

`object`	the input object.
`...`	currently of no use; included for future compatibility.

Details

A generic that can be extended by methods to exponentiate the numeric information in certain objects.

Value

This generic does not return anything and is just the basis for more sophisticated methods.

`expo,deseats_fc-method` *Exponentiate* `deseats` *Forecasts*

Description

Exponentiate, i.e. as act of retransformation, (point and interval) forecasts obtained via the `deseats` package.

Usage

```
## S4 method for signature 'deseats_fc'
expo(object, ...)
```

Arguments

`object`	an object of class `"deseats_fc"`.
`...`	currently without purpose; included for compatibility only.

Details

This function simply exponentiates point forecasts and bounds of interval forecasts within an object of class `"deseats_fc"`. This does not have the option to account for potential biases in point forecasts. For this purpose, use the predict method directly with `expo = TRUE` and `adjust.bias = TRUE`.

Value

An object of class `"deseats_fc"` is returned.

Examples

```
est <- s_semiarma(log(EXPENDITURES), set_options(order_poly = 3))
fc <- predict(est, n.ahead = 8)
fc2 <- expo(fc)
fc2
```

`fitted,hfilter-method` *Fitted Components of the Hamilton Filter*

Description

Obtain either fitted values or residuals from a fitted Hamilton filter.

Usage

```
## S4 method for signature 'hfilter'
fitted(object, ...)

## S4 method for signature 'hfilter'
residuals(object, ...)
```

Arguments

`object`	an object of class `"hfilter"`, i.e. generated by a call to `hamilton_filter`.
`...`	currently without further use; implemented for compatibility.

Details

Obtain the fitted and the residual values from the result of a fitted Hamilton filter. The name of the method indicates, what is returned.

Value

A time series object of class "ts" is returned.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- hamilton_filter(log(EXPENDITURES))
residuals(est)
fitted(est)
```

`gain` *Gain Function Generic*

Description

Gain Function Generic

Usage

```
gain(object, lambda = seq(0, 0.5, 1e-04), ...)
```

Arguments

`object`	an input object; for the linear filters considered in this object, gain function values should be obtained.
`lambda`	a vector of frequencies (from 0 to 0.5) for which to obtain the gain function.
`...`	currently without use; for possible future compatibility.

Details

A standard generic function. The purpose is to build various methods to instantaneously obtain gain function values for linear filters of different decomposition objects.

A generic that can be extended by methods to automatically obtain gain function values of linear filters described in certain objects.

Value

This generic does not return anything and is just the basis for more sophisticated methods.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

gain,deseats-method *Obtain gain function values for DeSeaTS Trend and Detrend Filters*

Description

Obtain gain function values for DeSeaTS Trend and Detrend Filters

Usage

```
## S4 method for signature 'deseats'
gain(object, lambda = seq(0, 0.5, 1e-04), ...)
```

Arguments

`object`	an object of class `"deseats"`.
`lambda`	a numeric vector with the frequencies at which to get the gain function values.
`...`	no current purpose for this ellipsis.

Details

The various filters obtained via `deseats` (represented by the returned weighting systems) have a representation in the frequency domain. Using this method, those gain function values can be easily obtained.

Value

A list is returned. Each element represents gain function values at the specified frequencies `lambda` for the filter defined through the element name.

`gain_trend` gain function values for the trend filter.

`gain_detrend` gain function values for the detrending filter.

`gain_season` gain function values for the seasonality filter.

`gain_deseason` gain function values for the seasonal adjustment filter.

`gain_comb` gain function values for the trend + seasonality filter.

`gain_decomb` gain function values for the detrending + seasonal adjustment filter.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
xt <- log(EXPENDITURES)
est <- deseats(xt)

lambda <- seq(0, 0.5, 0.01)
gain_values <- gain(est, lambda = lambda)
m <- length(gain_values$gain_trend[, 1])
k <- (m - 1) / 2
colF <- colorRampPalette(c("deepskyblue4", "deepskyblue"))
cols <- colF(m)

matplot(lambda, t(gain_values$gain_decomb[1:(k + 1), ]),
 type = paste0(rep("l", k + 1), collapse = ""),
 col = cols, lty = rep(1, k + 1))
title("Gain functions of the combined detrend and deseasonalization filters")

matplot(lambda, t(gain_values$gain_trend[1:(k + 1), ]),
 type = paste0(rep("l", k + 1), collapse = ""),
 col = cols, lty = rep(1, k + 1))
title("Gain functions of the trend filters")

matplot(lambda, t(gain_values$gain_deseason[1:(k + 1), ]),
 type = paste0(rep("l", k + 1), collapse = ""),
 col = cols, lty = rep(1, k + 1))
title("Gain functions of the seasonal adjustment filters")
```

GDP *Quarterly US GDP*

Description

A `ts` object that contains the quarterly US GDP (in billions of USD) from the first quarter of 1947 to the last quarter in 2019. The object contains 292 observations.

Usage

```
GDP
```

Format

An object of class `ts` with 292 rows and 1 columns.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-09-25) and then transformed into a time series object using R.

https://fred.stlouisfed.org/series/NA000334Q

`hamilton_filter` *Time Series Filtering Using the Hamilton Filter*

Description

A stationary remainder is obtained from a univariate time series using the filter proposed by Hamilton. The filter is capable of estimating the trend together with the seasonality in a series.

Usage

```
hamilton_filter(yt, h = NULL, p = NULL)
```

Arguments

`yt`	a time series object of class `ts` or an object that can be transformed to that class using `as.ts`.
`h`	the backwards time skip for the first regressor; the default is the seasonal period in `yt` multiplied by 2.
`p`	the number of regressors; the default is the seasonal period in `yt`.

Details

Implement the filter by Hamilton (2018) to decompose a time series.

Value

A list with the following elements is returned.

decomp an object of class `"mts"` that consists of the decomposed time series data.

ts_name the object name of the initially provided time series object.

frequency the frequency of the time series.

regression_output an object of class `"lm"`, i.e. basic regression output.

References

Hamilton, J. D. (2018). Why You Should Never Use the Hodrick-Prescott Filter. The Review of Economics and Statistics, 100(5): 831–843. DOI: 10.1162/rest_a_00706.

Examples

```
est <- hamilton_filter(log(EXPENDITURES))
est
```

hA_calc — *Calculation of Theoretically Optimal Bandwidth and Its Components*

Description

Allows to calculate the theoretically optimal bandwidth for estimating the trend and the seasonality in an equidistant time series with short-range dependence using locally weighted regression, if the trend function and the exact ARMA dependence structure of the errors are known.

Usage

```
hA_calc(
  m,
  arma = list(ar = NULL, ma = NULL, sd_e = 1),
  p = c(1, 3),
  mu = c(0, 1, 2, 3),
  frequ = c(4, 12),
  n = 300,
  cb = 0.1
)
```

Arguments

`m`	an expression that defines the trend function in terms of `x`, where `x` is the rescaled time on the interval $[0, 1]$.
`arma`	a list with the elements `ar`, `ma` and `sd_e`; `ar` is a numeric vector with the AR-coefficients, `ma` is a numeric vector with the MA-coefficients and `sd_e` is the innovation standard deviation.
`p`	the order of polynomial to use locally for the trend estimation.
`mu`	the smoothness parameter of the second-order kernel function used in the weighting process.
`frequ`	the frequency of the theoretical time series (4 for quarterly and 12 for monthly time series).
`n`	the number of observations.
`cb`	the part of observations to drop at each boundary.

Details

For simulation studies of the function `deseats` one may be interested in obtaining the theoretically optimal bandwidth for local regression first for a given theoretical process (from which realizations will be drawn in the simulation). This function assists in obtaining this theoretical bandwidth.

Value

This function returns a list with various elements. See the documentation of `deseats` to understand, what each quantity signifies.

`b` This is the theoretical quantity $\beta_{(k)}$.

`hA` The theoretically asymptotically optimal global bandwidth for locally weighted regression applied to the theoretical time series under consideration.

`Imk` This is the theoretical quantity $I[m^{(k)}]$.

`RK` This is the theoretical quantity $R(K)$.

`RW` This is the theoretical quantity $R(W)$.

`sum_autocov` This is the theoretical quantity $2\pi c_f$.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
arma <- list(ar = 0.8, sd_e = 0.01)
m_f <- expression(13.1 + 3.1 * x + (dnorm(x / 0.15 - 0.5 / 0.15) / 0.15) / 4)
n <- 500
p <- 1
mu <- 1
frequ <- 4
cb <- 0.05

hA_calc(
 m = m_f,
 arma = arma,
 p = p,
 mu = mu,
 frequ = frequ,
 n = n,
 cb = cb
)

t <- 1:n
xt <- t / n
mxt <- 13.1 + 3.1 * xt + dnorm(xt, mean = 0.5, sd = 0.15) / 4

S2 <- rep(c(0, 1, 0, 0), length.out = n)
S3 <- rep(c(0, 0, 1, 0), length.out = n)
S4 <- rep(c(0, 0, 0, 1), length.out = n)
sxt <- -0.5 + 0.25 * S2 + 0.5 * S3 + 1.25 * S4

set.seed(123)
et <- arima.sim(model = list(ar = 0.8), sd = 0.01, n = n)
yt <- ts(mxt + sxt + et, frequency = frequ)
plot(yt)
```

```
est <- deseats(yt)
est@bwidth
est@sum_autocov
```

`HOUSES` *Monthly New One Family Houses Sold in the USA*

Description

A `ts` object that contains the monthly new one family houses sold in the USA (in thousands of units) from January 1985 to December 2005. The object contains 252 observations.

Usage

```
HOUSES
```

Format

An object of class `ts` with 252 rows and 1 columns.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-09-30) and then transformed into a time series object using R.

`https://fred.stlouisfed.org/series/HSN1FNSA`

`LIVEBIRTHS` *Monthly Live Births in Germany*

Description

A `ts` object that contains the monthly observed live births in Germany (in thousands of cases) from January 1995 to March 2023. The object contains 339 observations.

Usage

```
LIVEBIRTHS
```

Format

An object of class `ts` of length 339.

Source

The data was obtained from the databank "Genesis" of the National Statistical Office of Germany (accessed: 2023-06-15) and then transformed into a time series object using R.

https://www-genesis.destatis.de/genesis/online?operation=previous&levelindex=4&levelid=1686843345946&levelid=1686843308009&step=3#abreadcrumb

---

`llin_decomp` *Decomposition of Time Series Using Local Linear Regression*

Description

Trend and seasonality are modelled in a two-step approach, where first the trend is being estimated using local linear regression and then the seasonality is being estimated using various local linear regressions as well. In both cases a manually selected bandwidth is required.

Usage

```
llin_decomp(
  yt,
  bwidth_trend = 4,
  bwidth_season = 5,
  kernel_par = 1,
  boundary_method = c("extend", "shorten"),
  season = NULL
)
```

Arguments

`yt`	a time series object of class `"ts"` or an object that can be transformed to that class using `as.ts`.
`bwidth_trend`	half of the absolute bandwidth (in years); represents the amount of data to use around the estimation time point to consider for trend smoothing.
`bwidth_season`	half of the absolute bandwidth (in years); represents the amount of data (only from the same quarter, month, etc.) to use around the estimation time point for the seasonality estimation.
`kernel_par`	the smoothness parameter for the second-order kernel function used in the weighting process; for `kernel_par = 0` a uniform kernel is used, for `kernel_par = 1` an epanechnikov kernel, and so on.
`boundary_method`	a single character value; it indicates, what bandwidth method to use at boundary points; for `"extend"`, the default, the smoothing window around boundary points will be extended towards the center of the data; for `"shorten"`, the window width will keep decreasing at boundary points when approaching the very first and the very last observation.
`season`	the seasonal period in `yt`; by default, the seasonal period is obtained automatically from `yt`.

Details

Apply local linear regression to estimate trend and seasonality in a given time series y_t. Assume that y_t follows an additive component model with trend and seasonality components. First, a local linear regression with a first (absolute) bandwidth is conducted to estimate the trend from the series. If the seasonal period is s, then afterwards s local linear regressions (for each individual seasonal subseries of the detrended series) are conducted with a second (absolute) bandwidth to obtain seasonality estimates.

Value

An S4 object with the following elements is returned.

decomp an object of class `"mts"` that consists of the decomposed time series data.

ts_name the object name of the initially provided time series object.

frequency the frequency of the time series.

bwidth_trend the same as the input argument `bwidth_trend`.

bwidth_season the same as the input argument `bwidth_season`.

boundary_method the same as the input argument `boundary_method`.

kernel_par the same as the input argument `kernel_par`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- llin_decomp(log(EXPENDITURES), bwidth_trend = 4, bwidth_season = 28)
est
```

`lm_decomp` *Decomposition of Time Series Using Linear Regression*

Description

Trend and seasonality are simultaneously modelled by considering a polynomial for the trend and a polynomial in the seasonality (via dummy variables and their interactions with time) for the different time units (e.g. months).

Usage

```
lm_decomp(yt, order_poly = 1, order_poly_s = 1, season = NULL)
```

Arguments

yt	a time series object of class "ts" or an object that can be transformed to that class using as.ts.
order_poly	the order of the polynomial considered for the trend; the default is order_poly = 1.
order_poly_s	the order of the polynomial considered for the seasonality; the default is order_poly_s = 1.
season	the seasonal period in yt; by default, the seasonal period is obtained automatically from yt.

Details

Apply ordinary least squares to estimate trend and seasonality simultaneously in a given time series. This a global approach in contrast to for example deseats, which is a local estimation method.

Value

An S4 object with the following elements is returned.

decomp an object of class "mts" that consists of the decomposed time series data.

ts_name the object name of the initially provided time series object.

frequency the frequency of the time series.

regression_output an object of class "lm", i.e. basic regression output; the time variable t used in the regression is encoded as seq_along(yt); the dummy variable S2 encodes the first observation time point (and the yearly corresponding time points) as -1 and the second observation time point (and the yearly corresponding time points) as 1, the dummy variable S3 does the same but has instead for the third observation time point (and the yearly corresponding time points) a 1, and so on.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- lm_decomp(log(EXPENDITURES), order_poly = 3, order_poly_s = 2)
est
```

ma_decomp — *Decomposition of Time Series Using Moving Averages*

Description

Trend and seasonality are modelled in a two-step approach, where first the trend is being estimated using moving averages and then trend + seasonality are being estimated using moving averages. The difference is then the estimated seasonality.

Usage

```
ma_decomp(yt, k_trend = 4, k_season = 5, season = NULL)
```

Arguments

`yt`	a time series object of class `"ts"` or an object that can be transformed to that class using `as.ts`.
`k_trend`	the complete absolute bandwidth (in years); represents the data of how many years to use around the estimation time point to consider for trend smoothing.
`k_season`	the complete absolute bandwidth (in years); represents the data of how many years (only from the same quarter, month, etc.) to use around the estimation time point for trend + seasonality smoothing.
`season`	the seasonal period in `yt`; by default, the seasonal period is obtained automatically from `yt`.

Details

Apply moving averages to estimate trend and seasonality in a given time series. This approach results in missings `NA` at boundary points.

Value

An S4 object with the following elements is returned.

decomp an object of class `"mts"` that consists of the decomposed time series data.

ts_name the object name of the initially provided time series object.

frequency the frequency of the time series.

k_trend the same as the input argument `k_trend`.

k_season the same as the input argument `k_season`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- ma_decomp(log(EXPENDITURES), k_trend = 6, k_season = 7)
est
```

---

measures *Forecasting Accuracy Measure Calculation*

Description

Given point forecasts and observations, calculate various forecasting accuracy measures.

Usage

```
measures(preds, obs)
```

Arguments

preds	the point predictions for the test data period.
obs	the observation series (training data and test data)

Details

Given one-step-ahead rolling forecasts as well as the whole series of given observations (training together with test data), different forecasting accuracy measures (MAE, RMSE, Pearson's correlation, MASE, RMSSE) are being calculated.

Value

A named vector with the obtained criteria values is returned.

Examples

```
xt <- EXPENDITURES
xt_in <- window(xt, end = c(2017, 4))
yt <- log(xt_in)
est <- s_semiarma(yt, set_options(order_poly = 3), inflation_rate = "optimal")
fc_results <- predict(est, n.ahead = 8, expo = TRUE)
point_fc <- fc_results@pred
measures(point_fc, xt)
```

`NOLABORFORCE` *Monthly Number of US Persons Not in the Labor Force*

Description

A `ts` object that contains the monthly observed number of persons in the USA that do not belong to the labor force (in millions of persons) from January 1990 to December 2019. The object contains 360 observations.

Usage

```
NOLABORFORCE
```

Format

An object of class `ts` of length 360.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-06-15) and then transformed into a time series object using R.

`https://fred.stlouisfed.org/series/LNU05000000`

`order_poly` *Smoothing Option Generics*

Description

Various generics that can be used write methods to either retrieve or set smoothing options.

Usage

```
order_poly(object)

order_poly(object) <- value

season(object, ...)

season(object) <- value

kernel_fun(object)

kernel_fun(object) <- value

bwidth(object)
bwidth(object) <- value

boundary_method(object)

boundary_method(object) <- value
```

Arguments

`object`	an object from which to either retrieve options or in which to set an option.
`value`	the value to set the corresponding option to.
`...`	without use; implemented for possible future compatibility only.

Details

`order_poly`, `season`, `kernel_fun`, `bwidth`, and `boundary_method` can be used to retrieve the order of polynomial, the seasonal frequency, the kernel function setting, the bandwidth and the boundary method from an object with smoothing options. The corresponding generics beginning with `<-` are useful to set such options instead.

The generics themselves are without a direct purpose.

Generics that can be extended by methods to set or obtain settings (e.g. smoothing options) in certain objects.

Value

These generics do not return anything and are just the basis for more sophisticated methods.

`order_poly,smoothing_options-method`
Retrieve or Set Smoothing Options

Description

Retrieve smoothing options from or set them in an object of class `"smoothing_options"`.

Usage

```
## S4 method for signature 'smoothing_options'
order_poly(object)

## S4 replacement method for signature 'smoothing_options'
order_poly(object) <- value

## S4 method for signature 'smoothing_options'
season(object)
```

```
## S4 replacement method for signature 'smoothing_options'
season(object) <- value

## S4 method for signature 'smoothing_options'
kernel_fun(object)

## S4 replacement method for signature 'smoothing_options'
kernel_fun(object) <- value

## S4 method for signature 'smoothing_options'
bwidth(object)

## S4 replacement method for signature 'smoothing_options'
bwidth(object) <- value

## S4 method for signature 'smoothing_options'
boundary_method(object)

## S4 replacement method for signature 'smoothing_options'
boundary_method(object) <- value
```

Arguments

`object`	an object of class `"smoothing_options"`.
`value`	the value to set the corresponding option to.

Details

Various methods are provided to either retrieve smoothing option settings from an object of class `"smoothing_options"` or to adjust them within such an object.

Value

The methods without `<-` return the corresponding settings in the supplied object. The methods with `<-` set the corresponding option in the provided object.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
opts <- set_options()
opts
order_poly(opts)
order_poly(opts) <- 3
opts
```

plot,decomp-method *Plot Method for Decomposition Results in the Style of Base R Plots*

Description

This is method for producing various plots of the decomposition results returned by this package.

Usage

```
## S4 method for signature 'decomp'
plot(x, which = NULL, ...)
```

Arguments

`x`	an object returned by the function `deseats`.
`which`	various plots can be selected either via a keyword or a number; enter `"facets"` or `1` to show a facet plot of the estimated time series components; enter `"observations"` or `2` to show the input time series; enter `"fitted"` or `3` to show the observations alongside the estimated trend with seasonality; enter `"detailed_fit"` or `4` to show the observations together with the fitted values and the trend; enter `"trend_season"` or `5` to show the observations together with the trend and with the seasonality (the latter shown around the series mean); enter `"residuals"` or `6` to plot the both detrended and seasonally adjusted series; use `7` or `"deseasonalized"` to show the seasonally adjusted series; enter `8` or `"detrended"` to plot the detrended series; the default is `which = NULL` which then lets you select a plot interactively in the R console.
`...`	further arguments to pass to `plot.ts` or `matplot` (depending on whether only one time series or multiple time series are shown in the plot).

Details

Create predefined standard plots of the decomposition objects returned by the `deseats` package, e.g. returned by the function `deseats`. Plots are created in the base R plot style. The type of plot can be chosen either interactively from the console, or the argument `which` can be used to directly select the kind of plot to create (see also the description of the argument `which`) within the function call.

If plot type 5 (`which = 5`) is selected, the estimated seasonality will be displayed around the mean of the observations by default. Setting the additional argument `s_around` to some other value, will lead to the seasonality being displayed around that constant value.

Value

A graphic is created in the plots windows, the function itself, however, returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
Xt <- log(EXPENDITURES)
est <- deseats(Xt)
plot(est, which = 3)
```

---

plot,deseats_fc-method *Plot Method for Class* "deseats_fc"

Description

Create basic R plots for forecasting objects of class "deseats_fc".

Usage

```
## S4 method for signature 'deseats_fc'
plot(x, y = NULL, ...)
```

Arguments

x	an object of class "deseats_fc", for example generated by a call to predict,s_semiarma-method.
y	currently without use; for compatibility only.
...	further arguments of plot.ts to adjust for example the axis limits via xlim and ylim.

Details

This is a plot method to visualize the forecasting results for a Seasonal Semi-ARMA model. Common plot arguments can be implemented to change the appearance.

Value

This method returns NULL.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- s_semiarma(log(EXPENDITURES))
fc <- predict(est, n.ahead = 4)
fc_e <- expo(fc)
plot(fc_e)
```

`plot,hfilter-method` *Plot Method for the Results of a Hamilton Filter*

Description

Visualize the results of an applied Hamilton filter.

Usage

```
## S4 method for signature 'hfilter'
plot(x, which = NULL, ...)
```

Arguments

`x`	an object of class `"hfilter"`, as returned by the function `hamilton_filter`.
`which`	either a string or a number can be entered to select a plot type from the function call; options are (1) a facet plot of the components, (2) the observed time series, (3) the observations together with the fitted values, and (4) the residuals; for `which = NULL`, the plot type can be selected interactively in the console.
`...`	further arguments to pass to `plot.ts` or `matplot` (depending on whether only one time series or multiple time series are shown in the plot).

Value

This function returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
est <- hamilton_filter(log(EXPENDITURES))
plot(est, which = 3, col = c(1, 6))
plot(est, which = 4)
```

predict,s_semiarma-method

Point and Interval Forecasts for Seasonal Semi-ARMA Models

Description

Obtain point and interval forecasts based on fitted Seasonal Semi-ARMA models.

Usage

```
## S4 method for signature 's_semiarma'
predict(
  object,
  n.ahead = 1,
  intervals = TRUE,
  alpha = c(0.95, 0.99),
  method = c("norm", "boot"),
  bootMethod = c("simple", "advanced"),
  npaths = 5000,
  quant.type = 8,
  expo = FALSE,
  adjust.bias = TRUE,
  ...
)
```

Arguments

`object`	an object of class `"s_semiarma"`.
`n.ahead`	a numeric vector of length one that represents the forecasting horizon; assume that `object` is based on observations at time points $t = 1, 2, \ldots, n$; forecasts are then obtained for time points $t = n + 1, n + 2, \ldots, n +$ `n.ahead`; the default is `n.ahead = 1`.
`intervals`	a logical vector of length one that indicates whether or not forecasting intervals should be obtained; the default is `intervals = TRUE`.
`alpha`	a numeric vector of variable length that indicates the confidence levels at which to obtain forecasting intervals; the default is `alpha = c(0.95, 0.99)`, i.e. confidence levels of 95 and 99 percent.
`method`	a character vector that indicates the method used to obtain forecasting intervals; available are theoretical intervals based on the assumption of normal innovations (`"norm"`) and intervals through a bootstrap (`"boot"`); the default is `method = "norm"`.
`bootMethod`	a character vector that allows the user to select a bootstrap procedure for the forecasting intervals when `method = "boot"` is selected; the default `bootMethod = "simple"` simulates future observations by resampling the obtained residuals; the second approach `bootMethod = "advanced"` also considers the variation in the ARMA coefficient estimates by simulating and reestimating complete

	ARMA paths upon which forecasts are obtained (see also the B-ARMARoots algorithm in Lu and Wang, 2020); the second approach is often time-consuming.
`npaths`	the number of paths to simulate, if the forecasting intervals are obtained via a bootstrap.
`quant.type`	the method to obtain sample quantiles from the simulated forecasting errors; see also the argument `type` of the function `quantile`.
`expo`	a logical vector of length one; indicates whether the forecasting results should be exponentiated at the end; the default is `expo = FALSE`.
`adjust.bias`	a logical vector of length one; indicates whether or not the point forecasts should have a bias adjustment when simultaneously `expo = TRUE`; the default is `TRUE`; for `TRUE`, the point forecasts on the exponentiated scale will be mean forecasts.
`...`	only for comparability with the standard `predict` method.

Details

Assume a Seasonal Semi-ARMA model was fitted using `s_semiarma`. Pass the resulting object to this method, in order to obtain point and interval forecasts.

If `expo = TRUE` and `adjust.bias = TRUE`, the point forecasts will be exponentiated and adjusted for bias, so that the resulting forecasts can be seen as mean forecasts on the exponentiated scale. Forecasting intervals do not need such an adjustment. For `expo = FALSE`, no adjustment will be done. Let `y_{n+k}` be the forecast on the log-scale (with `n` as the number of observations and `k` as the forecast horizon). Under `method = "norm"`, we use $\exp y_{n+k} \exp \sigma_h^2/2$ as the bias-adjusted point forecast, where $\sigma_h^2/2$ is the estimated variance of the prediction error in accordance with the infinite-order moving-average representation of the fitted ARMA model part. For `method = "boot"`, the sample mean of future simulated paths on the exponentiated scale is obtained.

Value

A list with the following elements is returned.

`pred` the obtained point forecasts.

`interv` the obtained forecasting intervals.

`obs` the observation series.

`ts_name` the name of the observation series object.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

Examples

```
xt <- log(EXPENDITURES)
est <- s_semiarma(xt)
predict(est, n.ahead = 10)
```

`RAINFALL` *Monthly Average Rainfall in Germany*

Description

A `ts` object that contains the monthly observed average rainfall in Germany (in mm) from January 1881 to August 2023. The object contains 1712 observations.

Usage

```
RAINFALL
```

Format

An object of class `ts` of length 1712.

Source

The data was obtained from the databank of the German Weather Service (DWD) (accessed: 2023-09-25) and then transformed into a time series object using R.

`https://www.dwd.de/DE/leistungen/zeitreihen/zeitreihen.html#buehneTop`

`read_ts` *Read in a Dataset Directly as an Object of Class `"ts"` or `"mts"`*

Description

Allows the user to read in a data file directly as a `"ts"` or `"mts"` object, where a time point column in the data file is immediately used to set starting points and frequency of the time series automatically correctly. Works for equidistant observation time points, e.g. quarterly or monthly observations.

Usage

```
read_ts(
  file,
  time_column = 1,
  sep = ",",
  dec = ".",
  header = TRUE,
  time_format = NULL
)
```

Arguments

`file`	a data file name given as a string (including the file ending); the file should have at least two columns: one time column and at least one or multiple columns for time series observations; alternatively, a data frame can be passed to this argument.
`time_column`	a number that indicates which column in the dataset is the variable with the time points; by default, the first column is assumed to contain the information on time points.
`sep`	the separation symbol between the dataset columns.
`dec`	the decimal symbol in the dataset.
`header`	`TRUE` or `FALSE`; does the dataset have a row with headers at the beginning?
`time_format`	with the default `NULL`, standard date formats will be tried to transform the time points column into a date object; if the formatting of the time column is unusual, the formatting can be specified here as a string.

Details

The data file is internally read into R as a `"zoo"` object and then transformed into a `"ts"` object using `zoo_to_ts`. This happens without the user noticing. The result is an immediate transformation of the input data into an object of class `"ts"` or `"mts"` for the user.

Value

An object of class `"ts"` or `"mts"` is returned.

Examples

```
### Create an example data file
a <- 1:12
b <- 21:32
tp <- seq(from = as.Date("2020-01-01"), to = as.Date("2020-12-01"), by = "month")
df <- data.frame(
 Time = tp,
 a = a,
 b = b
)

file <- file.path(tempdir(), "ExampleFile.csv")

write.table(df, file = file, quote = FALSE, sep = ",",
 row.names = FALSE, col.names = TRUE)

### Use the function to read in the data
xt <- read_ts(file)
xt
```

`RETAIL` *Monthly Total Volume of Retail Trade in Germany*

Description

A `ts` object that contains the monthly total volume of retail trade in Germany (Index 2015 = 100) from January 1991 to December 2019. The object contains 348 observations.

Usage

```
RETAIL
```

Format

An object of class `ts` with 348 rows and 1 columns.

Source

The data was obtained from the databank of the Federal Reserve Bank of St. Louis (accessed: 2023-10-11) and then transformed into a time series object using R.

`https://fred.stlouisfed.org/series/DEUSLRTTO01IXOBM`

`runDecomposition` *Shiny App for Decomposing Seasonal Time Series (Deprecated)*

Description

A shiny app is started that allows its user to run some of the decomposition approaches of this package interactively.

Usage

```
runDecomposition()
```

Details

Time series data can be uploaded to the app and then decomposed using some of the approaches implemented in this package. The decomposition is immediately visualized within the app, so that the user can assess the suitability of the decomposition graphically. The decomposed data can then be saved in CSV format.

Note: This function is deprecated. Switch to `run_decomposition()`.

Value

This function returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

---

`run_decomposition` *Shiny App for Decomposing Seasonal Time Series*

Description

A shiny app is started that allows its user to run some of the decomposition approaches of this package interactively.

Usage

```
run_decomposition()
```

Details

Time series data can be uploaded to the app and then decomposed using some of the approaches implemented in this package. The decomposition is immediately visualized within the app, so that the user can assess the suitability of the decomposition graphically. The decomposed data can then be saved in CSV format.

Value

This function returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

---

`SAVINGS` *Quarterly Savings of Private Households in Germany*

Description

A `ts` object that contains the quarterly observed savings of private households in Germany (in billions of EUR) from January 1991 to December 2019. The object contains 116 observations.

Usage

```
SAVINGS
```

Format

An object of class `ts` of length 116.

Source

The data was obtained from the databank "Genesis" of the National Statistical Office of Germany (accessed: 2023-06-15) and then transformed into a time series object using R.

https://www-genesis.destatis.de/genesis/online?operation=abruftabelleBearbeiten&levelindex=1&levelid=1686857137066&auswahloperation=abruftabelleAuspraegungAuswaehlen&auswahlverzeichnis=ordnungsstruktur&auswahlziel=werteabruf&code=81000-0010&auswahltext=&werteabruf=Werteabruf#abreadcrumb

---

`seasonplot` *Creation of Seasonal Plots*

Description

Simplified seasonal plot creation of time series in order to identify seasonal patterns more easily.

Usage

```
seasonplot(
  x,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  labels = TRUE,
  colorpalette = NULL,
  rm_xticks = FALSE
)
```

Arguments

`x`	a time series object of class `"ts"`.
`xlab`	a label for the plot's x-axis.
`ylab`	a label for the plot's y-axis.
`main`	a title for the plot.
`labels`	whether or not to show tick labels.
`colorpalette`	a color palette (either a numeric or character vector) that gives an alternative color palette to use for the subseries.
`rm_xticks`	whether to remove x-axis ticks and tick labels; the default is `FALSE`.

Details

The function creates a seasonal plot of the provided time series object. The series is split into different subseries, each reflecting one season, and then these subseries are plotted together. This helps identifying possible seasonal patterns in the data.

Value

The function does not return anything, however a plot is created in the plot window.

Examples

```
seasonplot(TEMPERATURE, xlab = "Degrees Celsius",
 main = "Seasonal plot of German temperature data")
```

`seasonplot_gg` *Creation of Seasonal Plots in the Style of ggplot2*

Description

Simplified seasonal plot creation of time series objects in order to identify seasonal patterns.

Usage

```
seasonplot_gg(x)
```

Arguments

`x`	a time series object of class `"ts"`.

Details

The function returns an plot object in the style of `ggplot2`. The plot can therefore be adjusted easily using common `ggplot2` syntax.

Value

The function returns a `ggplot2` plot object.

Examples

```
suppressWarnings(print(seasonplot_gg(TEMPERATURE) +
 ggplot2::ylab("Degrees Celsius") +
 ggplot2::ggtitle("Seasonal plot of German temperature data")))
```

select_bwidth — *Optimal Bandwidth Estimation for Locally Weighted Regression in Equidistant Time Series under Short Memory*

Description

Optimal Bandwidth Estimation for Locally Weighted Regression in Equidistant Time Series under Short Memory

Usage

```
select_bwidth(
  y,
  smoothing_options = set_options(),
  bwidth_start = NULL,
  inflation_rate = c("optimal", "naive"),
  correction_factor = FALSE,
  autocor = TRUE,
  drop = NULL,
  error_model = c("free", "ARMA"),
  nar_lim = c(0, 3),
  nma_lim = c(0, 3),
  arma_mean = FALSE,
  inf_criterion = c("bic", "aic")
)
```

Arguments

`y`	a numerical vector or a time series object of class `ts` or that can be transformed with `as.ts` to an object of class `ts`; for these observations, trend and seasonality will be obtained.
`smoothing_options`	an S4 object of class `smoothing_options`, which is returned by the function `set_options`; it includes details about the options to consider in the locally weighted regression such as the order of polynomial and the bandwidth for smoothing among others.
`bwidth_start`	a single numeric value that is only relevant if the slot `bwidth` in `smoothing_options` is set to `NA`; as the bandwidth will then be selected automatically, `bwidth_start` sets the initial bandwidth for the algorithm; the default, `bwidth_start = NULL`, corresponds to `bwidth_start = 0.1` for a local linear trend and to `bwidth_start = 0.2` for a local cubic trend.
`inflation_rate`	a character vector of length one that indicates, which inflation rate to use in the bandwidth selection; for a local linear trend, we have `inflation_rate = "optimal"` as the default, for a local cubic trend it is `inflation_rate = "naive"`, which correspond to inflation rates of 5/7 and 9/13, respectively.

`correction_factor`	A logical vector of length one; theoretically, a larger bandwidth to estimate the sum of autocovariances from residuals of pilot trend and seasonality estimates is advisable than for estimating trend and seasonality; for `correction_factor = TRUE`, this is implemented; the default is `correction_factor = FALSE`, because it was found that setting this argument to `TRUE` often overestimates the bandwidth.
`autocor`	a logical vector of length one; indicates whether to consider autocorrelated errors (`TRUE`) or independent but identically distributed errors (`FALSE`); the default is `autocor = TRUE`.
`drop`	a numeric vector of length one that indicates the proportion of the observations to not include at each boundary in the bandwidth estimation process, if a bandwidth is selected automatically; the default is `drop = NULL`, which corresponds to `drop = 0.05` for a local linear trend and to `drop = 0.1` for a local cubic trend.
`error_model`	a character vector of length one that indicates whether for `autocor = TRUE` the sum of autocovariances of the errors is obtained purely nonparametrically (`"free"`) or whether an autoregressive moving-average (ARMA) model is assumed `"ARMA"`; the default is `error_model = "free"`.
`nar_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum AR order to check via the BIC in each iteration of the algorithm via a two-element vector.
`nma_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum MA order to check via the BIC in each iteration of the algorithm via a two-element vector.
`arma_mean`	only valid for `error_model = "ARMA"`; decide whether to include an estimate of the mean in the ARMA fitting for the detrended series.
`inf_criterion`	selects the information criterion upon which the ARMA orders for the residuals are chosen; defaults to `"bic"` for the Bayesian information criterion; `"aic"` uses the Akaike information criterion.

Details

See further details in the documentation of the function `deseats`, where this function is applied internally by default to select an optimal bandwidth.

Value

The function returns a list with different components:

`bopt` the obtained optimal bandwidth.

`bwidths` the obtained bandwidth for each iteration of the IPI-algorithm.

`Imk` the final estimate of $I[m^{(k)}]$.

`sum_autocov` the final estimate of the sum of autocovariances.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator
- Yuanhua Feng (Department of Economics, Paderborn University), Author

References

- Feng, Y. (2013). An iterative plug-in algorithm for decomposing seasonal time series using the Berlin Method. Journal of Applied Statistics, 40(2): 266-281. DOI: 10.1080/02664763.2012.740626.
- Feng, Y., Gries. T, and Fritz, M. (2020). Data-driven local polynomial for the trend and its derivatives in economic time series. Journal of Nonparametric Statistics, 32(2): 510-533. DOI: 10.1080/10485252.2020.1759598.

Examples

```
Xt <- log(EXPENDITURES)
select_bwidth(Xt)
```

`set_options` *Specification of Smoothing Options*

Description

Set the smoothing specifications for locally weighted regression for identifying the trend and the seasonality in an equidistant time series.

Usage

```
set_options(
  order_poly = 3,
  season = NA_real_,
  kernel_fun = "epanechnikov",
  bwidth = NA_real_,
  boundary_method = "shorten"
)
```

Arguments

`order_poly`	the order of the local polynomials used for estimating the smooth nonparametric trend; the default is `3`.
`season`	the frequency of observations per time unit, for example per year; set to `12` for monthly data and to `4` for quarterly data and so on; the default is `NA_real_`, which leads to an automated frequency selection for time series objects in smoothing functions; if the argument is set to `NA_real_` and the observations used for smoothing are not formatted as time series objects, the frequency `1` will be used.

`kernel_fun`	the weighting function to consider; supported are four second-order kernel functions with compact support on $[-1, 1]$; enter `"uniform"` for the uniform kernel, `"epanechnikov"` for the Epanechnikov kernel, `"bisquare"` for the bisquare kernel or `"triweight"` for the triweight kernel; the default is `"epanechnikov"`.
`bwidth`	a numeric value that indicates the relative bandwidth to consider in the smoothing process; the default is `NA`, which then triggers a data-driven selection of an globally optimal bandwidth when the output of this function is passed to a smoothing function.
`boundary_method`	a single character value; it indicates, what bandwidth method to use at boundary points; for `"extend"`, the smoothing window around boundary points will be extended towards the center of the data; for `"shorten"`, the default, the window width will keep decreasing at boundary points when approaching the very first and the very last observation.

Details

An object of class `"smoothing_options"` is created that contains all required information to conduct a locally weighted regression for decomposing a seasonal time series. The information include the order of the trend polynomials, the frequency of the observed series, the second-order kernel function to use in the weighting process, the (relative) bandwidth to employ, and the boundary method for the bandwidth.

Value

The function returns an S4 object with the following elements (access via `@`):

order_poly identical to the input argument with that name; please see the description of that input argument.

season identical to the input argument with that name; please see the description of that input argument.

kernel_fun identical to the input argument with that name; please see the description of that input argument.

bwidth identical to the input argument with that name; please see the description of that input argument.

boundary_method identical to the input argument with that name; please see the description of that input argument.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

`show,deseats-method` *Printing of* `deseats` *Function Results*

Description

Print results of the function `deseats`.

Usage

```
## S4 method for signature 'deseats'
show(object)
```

Arguments

`object`	an object of class `"deseats"`.

Details

After trend and seasonality in a time series have been estimated using the function `deseats`, basic estimation results can be easily printed to the console.

Value

The function returns `NULL`.

`show,smoothing_options-method` *Show Method for Smoothing Options*

Description

Show smoothing settings in an object of class `"smoothing_options"`.

Usage

```
## S4 method for signature 'smoothing_options'
show(object)
```

Arguments

`object`	an object of class `"smoothing_options"`.

Details

This is a special printing method for objects object of class `"smoothing_options"`. Calling this method shows a nice looking overview of the saved smoothing settings.

Value

This method returns `NULL`.

Examples

```
opts <- set_options()
opts
```

`show,s_semiarma-method`

Show Method for Objects of Class `"s_semiarma"`

Description

Print results of the function `s_semiarma` to the console.

Usage

```
## S4 method for signature 's_semiarma'
show(object)
```

Arguments

`object`	an object of class `"s_semiarma"`.

Details

Use this method to create a nice looking overview of the contents of objects of class `"s_semiarma"`.

Value

This method returns `NULL`.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator

SUNSHINE *Monthly Hours of Sunshine in Germany*

Description

A `ts` object that contains the monthly observed hours of sunshine in Germany from January 1951 to August 2023. The object contains 872 observations.

Usage

```
SUNSHINE
```

Format

An object of class `ts` of length 872.

Source

The data was obtained from the databank of the German Weather Service (DWD) (accessed: 2023-09-25) and then transformed into a time series object using R.

https://www.dwd.de/DE/leistungen/zeitreihen/zeitreihen.html#buehneTop

s_semiarma *Fitting of a Seasonal Semiparametric ARMA Model*

Description

Fit a seasonal semiparametric autoregressive moving-average (S-Semi-ARMA) model to a univariate time series. The estimation is in two steps: firstly, the series is detrended and seasonally adjusted using the function `deseats`. Then an ARMA model is fitted to the residuals using `arima`.

Usage

```
s_semiarma(
  yt,
  smoothing_options = set_options(),
  arma_options = list(ar_order = NULL, ma_order = NULL),
  bwidth_start = 0.2,
  inflation_rate = c("optimal", "naive"),
  correction_factor = FALSE,
  drop = NULL,
  error_model = c("free", "ARMA"),
  nar_lim = c(0, 3),
  nma_lim = c(0, 3),
  arma_mean = FALSE,
  inf_criterion = c("bic", "aic")
)
```

Arguments

`yt`	a numerical vector or a time series object of class `ts` or that can be transformed with `as.ts` to an object of class `ts`; for these observations, trend and seasonality will be obtained.
`smoothing_options`	an S4 object of class `smoothing_options`, which is returned by the function `set_options`; it includes details about the options to consider in the locally weighted regression, such as the order of polynomial and the bandwidth for smoothing among others, for the nonparametric part of the model; the nonparametric model is fitted using `deseats`.
`arma_options`	a list with the two elements `ar_order` and `ma_order` that indicates the AR and MA orders to consider for the parametric part of the model.
`bwidth_start`	a single numeric value that is only relevant if the slot `bwidth` in `smoothing_options` is set to `NA`; as the bandwidth will then be selected automatically, `bwidth_start` sets the initial bandwidth for the algorithm.
`inflation_rate`	a character vector of length one that indicates, which inflation rate to use in the bandwidth selection; for a local linear trend, we have `inflation_rate = "optimal"` as the default, for a local cubic trend it is `inflation_rate = "naive"`, which correspond to inflation rates of 5/7 and 9/13, respectively.
`correction_factor`	A logical vector of length one; theoretically, a larger bandwidth to estimate the sum of autocovariances from residuals of pilot trend and seasonality estimates is advisable than for estimating trend and seasonality; for `correction_factor = TRUE`, this is implemented; for `error_model = "ARMA"`, `correction_factor = FALSE` is enforced; the default is `correction_factor = FALSE`, because it was found that setting this argument to `TRUE` often overestimates the bandwidth.
`drop`	a numeric vector of length one that indicates the proportion of the observations to not include at each boundary in the bandwidth estimation process, if a bandwidth is selected automatically; the default is `drop = 0.1`.
`error_model`	a character vector of length one that indicates whether for `autocor = TRUE` the sum of autocovariances of the errors is obtained purely nonparametrically (`"free"`) or whether an autoregressive moving-average (ARMA) model is assumed `"ARMA"`; the default is `error_model = "free"`.
`nar_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum AR order to check via the BIC in each iteration of the algorithm via a two-element vector.
`nma_lim`	only valid for `error_model = "ARMA"`; set the minimum and maximum MA order to check via the BIC in each iteration of the algorithm via a two-element vector.
`arma_mean`	only valid for `error_model = "ARMA"`; decide whether to include an estimate of the mean in the ARMA fitting for the detrended series.
`inf_criterion`	selects the information criterion upon which the ARMA orders for the residuals are chosen; defaults to `"bic"` for the Bayesian information criterion; `"aic"` uses the Akaike information criterion.

Details

For information on the nonparametric regression step, see `deseats`. After the trend and the seasonality have been removed from the data, an autoregressive moving-average (ARMA) model is fitted to the residuals either with orders selected by the Bayesian information criterion (BIC) or with manually selected orders. The ARMA model is fitted using `arima`.

All function arguments except for `arma_options` are identical to those in `deseats`. If all elements in `arma_options` are set to `NULL`, the ARMA model orders are selected for p, q from `nar_lim[[1]]` and `nma_lim[[1]]` up until `nar_lim[[2]]` and `nma_lim[[2]]` according to the BIC.

Value

The function returns and S4 object with the following elements (access them via `@`):

`decomp` an object of class `"mts"` that includes the observed time series and its estimated components.

`nonpar_model` an object of class `"deseats"`; this is the result of applying `deseats`.

`par_model` an object of class `"Arima"`; the result of applying `arima` to the residuals of the nonparametric estimation step.

Author(s)

- Dominik Schulz (Research Assistant) (Department of Economics, Paderborn University), Author and Package Creator
- Yuanhua Feng (Department of Economics, Paderborn University), Author

Examples

```
Xt <- log(EXPENDITURES)
est <- s_semiarma(Xt)
est
```

`TEMPERATURE` *Monthly Average Temperature in Germany*

Description

A `ts` object that contains the monthly observed average temperature in Germany (in degrees Celsius) from January 1881 to August 2023. The object contains 1712 observations.

Usage

```
TEMPERATURE
```

Format

An object of class `ts` of length 1712.

Source

The data was obtained from the databank of the German Weather Service (DWD) (accessed: 2023-09-25) and then transformed into a time series object using R.

https://www.dwd.de/DE/leistungen/zeitreihen/zeitreihen.html#buehneTop

---

`trend` *Obtain Estimated Components of a Time Series*

Description

Obtain estimated components, such as the estimated trend, the seasonally adjusted series, and so on from an estimation object.

Usage

```
trend(object, ...)

deseasonalize(object, ...)

detrend(object, ...)
```

Arguments

`object`	the estimation object.
`...`	currently without use; included for future compatibility.

Details

Generics that can be extended by methods to obtain (estimated) time series components from certain estimation objects.

Value

These generics do not return anything and are just the basis for more sophisticated methods.

`trend,decomp-method` *Obtain Individual Components of a Decomposed Time Series*

Description

The provided methods work for decomposition objects created within this package. They allow the user to retrieve individual components among the estimated ones.

Usage

```
## S4 method for signature 'decomp'
trend(object, ...)

## S4 method for signature 'decomp'
season(object, ...)

## S4 method for signature 'decomp'
fitted(object, ...)

## S4 method for signature 'decomp'
residuals(object, ...)

## S4 method for signature 'decomp'
deseasonalize(object, ...)

## S4 method for signature 'decomp'
detrend(object, ...)
```

Arguments

`object`	the estimation / decomposition object, for example of class `"deseats"`.
`...`	without further use; implemented for compatibility only.

Details

Apply these functions directly to an estimation object, i.e. the result of a decomposition of a seasonal time series, in order to retrieve individual estimated components.

Value

These methods return time series objects of class `"ts"` that represent the corresponding estimated component in the time series originally used in the estimation process.

Examples

```
Xt <- log(EXPENDITURES)
smoothing_options <- set_options(order_poly = 3)
est <- deseats(Xt, smoothing_options = smoothing_options)
trend_e <- trend(est)           # Trend estimates
season_e <- season(est)         # Seasonality estimates
trend_season_e <- fitted(est)   # Trend + seasonality estimates
resid_e <- residuals(est)       # Residuals (observ. - trend - seasonality)
ts_adj <- deseasonalize(est)    # Seasonally adjusted series
ts_notrend <- detrend(est)      # Detrended series
```

---

zoo_to_ts — *Time Series Object Conversion from* "zoo" *to* "ts"

Description

Allows for the conversion of time series objects of class "zoo" to time series objects of class "ts". This is only suitable, if the time series is observed in regular time intervals (monthly, quarterly, yearly, etc.). The correct observation time points are then kept.

Usage

```
zoo_to_ts(xt)
```

Arguments

xt	a time series object of class "zoo" with equidistant observation time points (monthly, quarterly, yearly, etc.).

Details

An equidistant time series object of class "zoo" is transformed to class "ts". This is particularly useful, since most functions of this package work with objects of class "ts" only.

Value

An object of class "ts" is returned.

Examples

```
# Create example zoo-object
tp <- seq(from = as.Date("2020-01-01"), to = as.Date("2020-10-01"), by = "month")
xt <- zoo::zoo(1:10, order.by = tp)
xt

# Transform into ts-object
yt <- zoo_to_ts(xt)
```

```
yt
```

Index