Package 'trendseries'

July 14, 2026

Type Package

Title Extract Trends from Time Series

Version 1.4.0

Description Provides a unified interface to extract trends, cycles, and seasonal components from monthly and quarterly time series using established econometric filters and smoothing methods, with frequency-aware defaults for common economic frequencies.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

URL https://github.com/viniciusoike/trendseries, https://viniciusoike.github.io/trendseries/

BugReports https://github.com/viniciusoike/trendseries/issues

Imports cli, dlm, hpfilter, lubridate, mFilter, RcppRoll, rlang, stats, tibble, tsbox

Depends R (>= 4.1.0)

RoxygenNote 7.3.3

Suggests dplyr, ggplot2 (>= 4.0.0), knitr, rmarkdown, scales, seasonal, testthat (>= 3.0.0), tidyr, xts

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Vinicius Oike [aut, cre, cph] (ORCID: <https://orcid.org/0009-0005-8015-9189>)

Maintainer Vinicius Oike <viniciusoike@gmail.com>

Repository CRAN

Date/Publication 2026-07-14 02:20:02 UTC

Contents

augment_trends . . . 2
coffee_arabica . . . 5
coffee_robusta . . . 6
converters . . . 7
decompose_series . . . 7
deseason_series . . . 11
detrend_series . . . 14
df_to_ts . . . 17
electric . . . 18
electricity . . . 18
extract_trends . . . 19
gdp_construction . . . 23
ibcbr . . . 24
metadata_series . . . 24
oil_derivatives . . . 25
retail_autofuel . . . 25
retail_volume . . . 26
transit_london_avgs . . . 27
transit_london_monthly . . . 27
ts_to_df . . . 28
vehicles . . . 29

Index **30**

`augment_trends` *Add trend columns to data frame*

Description

Pipe-friendly function that adds trend columns to a tibble or data.frame. Designed for exploratory analysis of monthly and quarterly economic time series. Supports multiple trend extraction methods and handles grouped data.

Usage

```
augment_trends(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  group_vars = NULL,
  methods = "stl",
  frequency = NULL,
  suffix = NULL,
  window = NULL,
  smoothing = NULL,
```

```
  band = NULL,
  align = NULL,
  params = list(),
  .quiet = FALSE
)
```

Arguments

data	A data.frame, tibble, or data.table containing the time series data.
date_col	Name of the date column. Defaults to "date". Must be of class Date.
value_col	Name of the value column(s). Defaults to "value". Must be numeric. A character vector of length > 1 is accepted; trends are extracted for each column and named trend_{method}_{col} (e.g. trend_stl_consumption).
group_cols	Optional grouping variables for multiple time series. Can be a character vector of column names.
group_vars	Deprecated. Use group_cols instead.
methods	Character vector of trend methods. Options: "hp", "bk", "cf", "ma", "stl", "loess", "spline", "poly", "bn", "ucm", "hamilton", "spencer", "ewma", "wma", "triangular", "kernel", "kalman", "median", "gaussian". Default is "stl".
frequency	The frequency of the series. Supports 4 (quarterly) or 12 (monthly). Will be auto-detected if not specified.
suffix	Optional suffix for trend column names. If NULL, uses method names.
window	Unified window/period parameter for moving average methods (ma, wma, triangular, stl, ewma, median, gaussian). Must be positive. If NULL, uses frequency-appropriate defaults. For EWMA, the window is converted to the smoothing factor via alpha = 2 / (window + 1). Cannot be used simultaneously with smoothing for EWMA method. For ma, median, and henderson methods, a numeric vector is accepted (e.g., c(9, 13, 23)), which adds one column per window value named trend_henderson_9, trend_henderson_13, etc. Other methods ignore extra values (with a warning).
smoothing	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: use large values (1600+) or small values (0-1) that get converted. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with window for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: controls the ratio of measurement to process noise (higher = more smoothing). For others: typically 0-1 range.
band	Unified band parameter for bandpass filters (bk, cf). Both values must be positive. Provide as c(low, high) where low/high are periods in quarters, e.g., c(6, 32).
align	Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: "center" (default, uses surrounding values), "right" (causal, uses past values only), "left" (anti-causal, uses future values only). Note: triangular only supports "center" and "right". If NULL, uses "center" as default.

params	Optional list of method-specific parameters for fine control.
.quiet	If TRUE, suppress informational messages.

Details

This function is designed for monthly (frequency = 12) and quarterly (frequency = 4) economic data. It uses economic-appropriate defaults for all trend extraction methods.

For grouped data, the function applies trend extraction to each group separately, maintaining the original data structure while adding trend columns.

Value

A tibble with original data plus trend columns named `trend_{method}` or `trend_{method}_{suffix}` if suffix is provided.

Examples

```
# Simple STL decomposition on quarterly GDP construction data
gdp_construction |> augment_trends(value_col = "index")

# Multiple smoothing methods with unified parameter
gdp_construction |>
  augment_trends(
    value_col = "index",
    methods = c("hp", "loess", "ewma"),
    smoothing = 0.3
  )

# Moving averages with unified window on monthly data
vehicles |>
  tail(60) |>
  augment_trends(
    value_col = "production",
    methods = c("ma", "wma", "triangular"),
    window = 8
  )

# Economic indicators with different methods
ibcbr |>
  tail(48) |>
  augment_trends(
    value_col = "index",
    methods = c("median", "kalman", "kernel"),
    window = 9,
    smoothing = 0.15
  )

# Moving average with right alignment (causal filter)
vehicles |>
  tail(60) |>
  augment_trends(
```

```
    value_col = "production",
    methods = "ma",
    window = 12,
    align = "right"
  )

# Advanced: fine-tune specific methods
electric |>
  tail(72) |>
  augment_trends(
    value_col = "consumption",
    methods = "median",
    window = 7
  )

# Multiple MA windows in a single call (adds trend_ma_3, trend_ma_6, trend_ma_12)
vehicles |>
  tail(60) |>
  augment_trends(
    value_col = "production",
    methods = "ma",
    window = c(3, 6, 12)
  )
```

`coffee_arabica` *Daily Arabica Coffee Price*

Description

Daily Arabica coffee price data from CEPEA/ESALQ (USP) with inflation adjustment.

Prices are provided in Brazilian reais, with USD values calculated using the daily USD/BRL exchange rate at 16:30. All reported prices include taxes and freight costs.

The data tracks prices for standard 60kg sacks of type 6 Arabica coffee negotiated in São Paulo (SP), representing production from five major regions: Cerrado, Sul de Minas Gerais, Mogiana (SP), Garça (SP), and Northeast Paraná. Regional prices are weighted by production volume to create the final value.

Usage

```
coffee_arabica
```

Format

A tibble with daily observations:

date Date column

spot_rs Spot price in Brazilian Reais per 60-kg bag

spot_us Spot price in US Dollars per 60-kg bag

usd_2022 US Dollar price adjusted for inflation (base year 2022)

trend_ma 22-day moving average of inflation-adjusted prices

Source

Center for Advanced Studies in Applied Economics (CEPEA - ESALQ/USP).

See Also

coffee_robusta

`coffee_robusta` *Daily Robusta Coffee Price*

Description

Daily Robusta coffee price data from CEPEA/ESALQ (USP) with inflation adjustment.

Prices are provided in Brazilian reais, with USD values calculated using the daily USD/BRL exchange rate at 16:30. All reported prices include taxes and freight costs.

The data tracks prices for standard 60kg sacks of type 6 Arabica coffee negotiated in São Paulo (SP), representing production from two major regions: Colatina (ES) and São Gabriel da Palha (ES). The final value is an arithmetic average of the regional prices.

Usage

```
coffee_robusta
```

Format

A tibble with daily observations:

date Date column

spot_rs Spot price in Brazilian Reais per 60-kg bag

spot_us Spot price in US Dollars per 60-kg bag

usd_2022 US Dollar price adjusted for inflation (base year 2022)

trend_ma 22-day moving average of inflation-adjusted prices

Source

Center for Advanced Studies in Applied Economics (CEPEA - ESALQ/USP).

See Also

coffee_arabica

converters *Data Format Conversion Utilities*

Description

Functions for converting between different time series formats, frequency detection, and data frame manipulation for the trendseries package. These functions handle the interface between tibble/data.frame workflows and time series objects.

decompose_series *Decompose time series into trend, seasonal, and remainder components*

Description

Pipe-friendly function that decomposes a time series into its trend, seasonal, and remainder components, adding them as columns to the input data frame. Supports STL decomposition and regression-based decomposition.

Usage

```
decompose_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "stl",
  trend = "linear",
  transform = "none",
  frequency = NULL,
  seasadj = FALSE,
  params = list(),
  .quiet = FALSE
)
```

Arguments

data	A data.frame, tibble, or data.table containing the time series data.
date_col	Name of the date column. Defaults to "date". Must be of class Date.
value_col	Name of the value column. Defaults to "value". Must be numeric.
group_cols	Optional grouping variables for multiple time series. A character vector of column names. When provided, decomposition is applied independently to each group.

`methods` Decomposition method(s). One or more of `"stl"`, `"regression"`, `"classic"`, `"bsm"`, or `"seats"`. Default is `"stl"`. When several methods are supplied (e.g. `c("stl", "classic")`), each one contributes its own `trend_*`, `seasonal_*`, and `remainder_*` columns so decompositions can be compared side by side.

- `"stl"`: Seasonal-Trend decomposition via Loess (`stats::stl()`).
- `"regression"`: joint OLS trend + seasonal-dummy model.
- `"classic"`: classical decomposition via moving averages (`stats::decompose()`).
- `"bsm"`: Basic Structural (state-space) Model via the Kalman smoother (`stats::StructTS()`).
- `"seats"`: X-13ARIMA-SEATS decomposition (requires the `seasonal` package; see Details).

`trend` For `methods = "regression"` only: the polynomial form of the trend component. One of `"linear"`, `"quadratic"`, or `"cubic"`. Ignored by the other methods. Default is `"linear"`.

`transform` Transformation applied to the series before decomposition. One of `"none"` (default, additive decomposition) or `"log"`. With `"log"`, the series is log-transformed, decomposed additively, and the components are exponentiated back, yielding a *multiplicative* decomposition.

`frequency` The frequency of the series. Supports 4 (quarterly) or 12 (monthly). Will be auto-detected if not specified. All methods require `frequency > 1`.

`seasadj` If `TRUE`, also add a `seasadj_{method}` column holding the seasonally adjusted series (the series with the seasonal component removed: `trend + remainder` for additive decompositions, `trend * remainder` for multiplicative ones). Default `FALSE`.

`params` Optional list of method-specific parameters for fine control. Sensible defaults are provided for all parameters; this argument is only needed for non-standard use cases.

For **STL** (`methods = "stl"`):

- `s.window` or `stl_s_window`: seasonal smoothing window. Either `"periodic"` (default, assumes constant seasonal pattern) or a positive odd integer (larger values allow more slowly evolving seasonality).
- `t.window` or `stl_t_window`: trend smoothing window (odd integer, or `NULL` to let `stats::stl()` choose automatically — recommended default).
- `robust` or `stl_robust`: logical. If `TRUE`, uses robust fitting to reduce the influence of outliers. Default `FALSE`.

For **regression** (`methods = "regression"`):

- `poly_raw`: logical. If `FALSE` (default), uses orthogonal polynomials (numerically stable, recommended). If `TRUE`, uses raw polynomials (more interpretable coefficients, less stable for degree >= 2).

classic, **bsm**, and **seats** take no `params`. For multiplicative seasonality with any method, use `transform = "log"`.

`.quiet` If `TRUE`, suppress informational messages.

Details

All methods require seasonal data (`frequency > 1`). For non-seasonal (annual) series, use `augment_trends()` to extract a trend component only.

STL Decomposition:

Uses `stats::stl()` (Seasonal-Trend decomposition via Loess). The seasonal component is estimated with a loess smoother, the trend with an adaptive moving average, and the remainder is the residual. Default settings (`s.window = "periodic"`, `robust = FALSE`) are appropriate for most economic series with stable seasonal patterns.

Regression Decomposition:

Fits a joint OLS model:

$$y_t = f(t) + s(t) + \epsilon_t$$

where $f(t)$ is a polynomial in time and $s(t)$ is captured by period dummy variables (month or quarter indicators). The components are isolated via `stats::predict(type = "terms")`:

- **Trend**: constant + polynomial terms (captures the long-run level and direction).
- **Seasonal**: period dummy terms, centred to mean zero over the sample.
- **Remainder**: residuals from the full model.

By default, orthogonal polynomials (`poly_raw = FALSE`) are used for numerical stability. For `trend = "cubic"`, this is especially recommended.

Classical Decomposition:

Uses `stats::decompose()`. The trend is a centred moving average of order equal to the frequency; the seasonal component is the average detrended value for each period; the remainder is the residual. Simple and fast, but shouldn't be used in practice.

Basic Structural Model (BSM):

Uses `stats::StructTS(type = "BSM")`, a state-space model with stochastic level, slope, and seasonal components estimated by maximum likelihood and extracted with the Kalman smoother (`stats::tsSmooth()`). Unlike the moving-average methods it produces trend and seasonal estimates for every observation, including the endpoints, and lets both components evolve over time. Fitting relies on numerical optimisation and can occasionally fail to converge on short or irregular series.

X-13ARIMA-SEATS (SEATS):

Uses the `seasonal` package, which wraps the U.S. Census Bureau's X-13ARIMA-SEATS program. `seas()` is run with its automatic defaults (model selection, log/level transformation, outlier detection, and calendar adjustment), and the SEATS trend-cycle (`s12`) and seasonally adjusted series (`s11`) are mapped to an additive trend/seasonal/remainder so the exact identity holds regardless of the internal transformation. Because X-13 picks its own log/level transformation, `seats` is best used with the default `transform = "none"`; an outer log transform is redundant.

Multiplicative Seasonality:

When the seasonal amplitude grows with the level of the series (a multiplicative pattern, common in economic data), set `transform = "log"`. The series is log-transformed, decomposed additively, and the components are exponentiated back. This works uniformly for every method and requires strictly positive values.

Value

A tibble with the original columns plus, for each requested method, three new columns (and a fourth when `seasadj = TRUE`):

- `trend_{method}`: the estimated trend component.
- `seasonal_{method}`: the estimated seasonal component.
- `remainder_{method}`: what remains after removing trend and seasonal.
- `seasadj_{method}`: the seasonally adjusted series (only if `seasadj = TRUE`).

With `transform = "none"` the additive identity `value = trend + seasonal + remainder` holds exactly for every method. With `transform = "log"` the *product* identity `value = trend * seasonal * remainder` holds instead. For `"classic"` the trend (and hence remainder) is `NA` for the first and last `frequency / 2` observations (the centred moving average has no boundary support).

Output rows are ordered by date within each group; the original row order is not preserved.

Examples

```
# STL decomposition (default settings work well for most economic series)
gdp_construction |>
  decompose_series(value_col = "index")

# STL with robust fitting (useful when the series has outliers)
gdp_construction |>
  decompose_series(
    value_col = "index",
    params = list(robust = TRUE)
  )

# STL with evolving seasonality (s.window controls how fast it can change)
gdp_construction |>
  decompose_series(
    value_col = "index",
    params = list(s.window = 13)
  )

# Regression with cubic trend
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "regression",
    trend = "cubic"
  )

# Classical decomposition via moving averages (boundary trend is NA)
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "classic"
  )

# Basic Structural Model (state-space, components for every observation)
```

```
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods = "bsm"
  )

# X-13ARIMA-SEATS (requires the 'seasonal' package)
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    decompose_series(
      value_col = "index",
      methods = "seats"
    )
}

# Multiplicative decomposition via log transform (works for any method)
oil_derivatives |>
  decompose_series(
    value_col = "production",
    transform = "log"
  )

# Several methods at once for side-by-side comparison
gdp_construction |>
  decompose_series(
    value_col = "index",
    methods   = c("stl", "classic")
  )

# Also return the seasonally adjusted series
gdp_construction |>
  decompose_series(
    value_col = "index",
    seasadj   = TRUE
  )

# Grouped decomposition: one decomposition per electricity sector
electricity |>
  decompose_series(
    group_cols = "name_series"
  )
```

`deseason_series` *Seasonally adjust (deseason) a time series*

Description

Pipe-friendly convenience wrapper around `decompose_series()` focused on a single task: removing the seasonal component from a time series. It adds a `seasadj_{method}` column holding the

seasonally adjusted (deseasoned) series and, optionally, the underlying trend, seasonal, and remainder components.

Usage

```
deseason_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "stl",
  transform = "none",
  frequency = NULL,
  components = FALSE,
  params = list(),
  .quiet = FALSE
)
```

Arguments

`data`	A `data.frame`, `tibble`, or `data.table` containing the time series data.
`date_col`	Name of the date column. Defaults to `"date"`. Must be of class `Date`.
`value_col`	Name of the value column. Defaults to `"value"`. Must be numeric.
`group_cols`	Optional grouping variables for multiple time series. A character vector of column names. When provided, decomposition is applied independently to each group.
`methods`	Seasonal-adjustment method(s). One or more of `"stl"` (default) or `"seats"`. When both are supplied, each contributes its own `seasadj_{method}` column (and component columns when `components = TRUE`) so the adjustments can be compared side by side. • `"stl"`: Seasonal-Trend decomposition via Loess (`stats::stl()`). • `"seats"`: X-13ARIMA-SEATS decomposition (requires the `seasonal` package; see `decompose_series()` for details).
`transform`	Transformation applied to the series before decomposition. One of `"none"` (default, additive decomposition) or `"log"`. With `"log"`, the series is log-transformed, decomposed additively, and the components are exponentiated back, yielding a *multiplicative* decomposition.
`frequency`	The frequency of the series. Supports 4 (quarterly) or 12 (monthly). Will be auto-detected if not specified. All methods require `frequency > 1`.
`components`	If `FALSE` (default), only the seasonally adjusted `seasadj_{method}` column is added. If `TRUE`, the `trend_{method}`, `seasonal_{method}`, and `remainder_{method}` columns are also added (the full `decompose_series()` output).
`params`	Optional list of method-specific parameters for fine control. Sensible defaults are provided for all parameters; this argument is only needed for non-standard use cases. For **STL** (`methods = "stl"`):

- `s.window` or `stl_s_window`: seasonal smoothing window. Either `"periodic"` (default, assumes constant seasonal pattern) or a positive odd integer (larger values allow more slowly evolving seasonality).
- `t.window` or `stl_t_window`: trend smoothing window (odd integer, or `NULL` to let `stats::stl()` choose automatically — recommended default).
- `robust` or `stl_robust`: logical. If `TRUE`, uses robust fitting to reduce the influence of outliers. Default `FALSE`.

For **regression** (`methods = "regression"`):

- `poly_raw`: logical. If `FALSE` (default), uses orthogonal polynomials (numerically stable, recommended). If `TRUE`, uses raw polynomials (more interpretable coefficients, less stable for degree >= 2).

classic, **bsm**, and **seats** take no `params`. For multiplicative seasonality with any method, use `transform = "log"`.

`.quiet` If `TRUE`, suppress informational messages.

Details

`deseason_series()` is a thin wrapper: it calls `decompose_series()` with `seasadj = TRUE` and then keeps only the seasonally adjusted column unless `components = TRUE`. All seasonal-adjustment behaviour, validation, grouping, and the `transform = "log"` (multiplicative) path are inherited unchanged from `decompose_series()`. See its documentation for method internals and the meaning of the `params` argument.

For a full trend/seasonal/remainder decomposition, or for the `regression`, `classic`, and `bsm` methods, use `decompose_series()` directly.

Value

A tibble with the original columns plus, for each requested method, a `seasadj_{method}` column holding the seasonally adjusted series. When `components = TRUE`, the `trend_{method}`, `seasonal_{method}`, and `remainder_{method}` columns are added as well.

The seasonally adjusted series is the series with the seasonal component removed: `trend + remainder` for additive decompositions, `trend * remainder` when `transform = "log"`. Output rows are ordered by date within each group; the original row order is not preserved.

See Also

`decompose_series()` for the underlying decomposition and the full set of methods; `augment_trends()` to extract a trend component only.

Examples

```
# Seasonally adjust a quarterly series (STL, the default)
gdp_construction |>
  deseason_series(value_col = "index")

# Also keep the trend, seasonal, and remainder components
gdp_construction |>
  deseason_series(value_col = "index", components = TRUE)
```

```
# Multiplicative adjustment via log transform (seasonal swings grow with level)
gdp_construction |>
  deseason_series(value_col = "index", transform = "log")

# X-13ARIMA-SEATS adjustment (requires the 'seasonal' package)
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    deseason_series(value_col = "index", methods = "seats")
}

# Compare STL and SEATS adjustments side by side
if (requireNamespace("seasonal", quietly = TRUE)) {
  gdp_construction |>
    deseason_series(value_col = "index", methods = c("stl", "seats"))
}

# Grouped seasonal adjustment: one adjustment per electricity sector
electricity |>
  deseason_series(group_cols = "name_series")
```

`detrend_series` *Detrend a time series*

Description

Pipe-friendly convenience wrapper around `augment_trends()` focused on a single task: removing the trend from a time series. It adds a `detrend_{method}` column holding the detrended series (the deviation from trend, often called the *cycle* in economics) and, optionally, the underlying trend itself.

For econometric filters such as `"hp"` (the default), `"bk"`, `"cf"`, and `"hamilton"`, the detrended series is the business-cycle component those filters were designed to isolate (e.g. the output gap).

Usage

```
detrend_series(
  data,
  date_col = "date",
  value_col = "value",
  group_cols = NULL,
  methods = "hp",
  transform = "none",
  frequency = NULL,
  components = FALSE,
  window = NULL,
  smoothing = NULL,
  band = NULL,
```

```
  align = NULL,
  params = list(),
  .quiet = FALSE
)
```

Arguments

data	A data.frame, tibble, or data.table containing the time series data.
date_col	Name of the date column. Defaults to "date". Must be of class Date.
value_col	Name of the value column(s). Defaults to "value". Must be numeric. A character vector of length > 1 is accepted; each column is detrended separately and the results are named detrend_{method}_{col} (e.g. detrend_hp_consumption).
group_cols	Optional grouping variables for multiple time series. Can be a character vector of column names.
methods	Character vector of trend methods used for detrending. Any method supported by augment_trends() is accepted. Default is "hp" (Hodrick-Prescott filter with frequency-appropriate smoothing). When several methods are supplied, each one contributes its own detrend_{method} column so the detrended series can be compared side by side.
transform	Transformation applied before detrending. One of: • "none" (default): the trend is fitted to the raw series and detrend = value - trend, in the units of the series. • "log": the trend is fitted to the log series and detrend = log(value) - log(trend), the *log deviation from trend*. Multiplied by 100, this is approximately the percentage deviation from trend (the convention for output gaps). Requires strictly positive values. The trend_{method} columns (when components = TRUE) are reported back in the units of the series.
frequency	The frequency of the series. Supports 4 (quarterly) or 12 (monthly). Will be auto-detected if not specified.
components	If FALSE (default), only the detrended detrend_{method} column is added. If TRUE, the fitted trend_{method} column is also kept.
window	Unified window/period parameter for moving average methods; see augment_trends(). For "ma", "median", and "henderson", a numeric vector is accepted (e.g. c(6, 12)), which adds one detrended column per window value (detrend_ma_6, detrend_ma_12, ...).
smoothing	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: use large values (1600+) or small values (0-1) that get converted. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with window for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: controls the ratio of measurement to process noise (higher = more smoothing). For others: typically 0-1 range.
band	Unified band parameter for bandpass filters (bk, cf). Both values must be positive. Provide as c(low, high) where low/high are periods in quarters, e.g., c(6, 32).

align	Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: "center" (default, uses surrounding values), "right" (causal, uses past values only), "left" (anti-causal, uses future values only). Note: triangular only supports "center" and "right". If NULL, uses "center" as default.
params	Optional list of method-specific parameters for fine control.
.quiet	If TRUE, suppress informational messages.

Details

detrend_series() is a thin wrapper: it calls augment_trends() with the requested methods and subtracts each fitted trend from the series (on the log scale when transform = "log"). All trend-fitting behaviour, validation, grouping, and the unified parameters (window, smoothing, band, align, params) are inherited unchanged from augment_trends(). See its documentation for method internals and parameter details.

Detrending does **not** remove seasonality: the detrended series of a raw seasonal series still contains the seasonal swings, and seasonality can leak into the cycle estimated by filters such as HP. For seasonal data, seasonally adjust first and detrend the adjusted series (see Examples), or use decompose_series() for a full trend/seasonal/remainder split.

Value

A tibble with the original columns plus, for each requested method, a detrend_{method} column holding the detrended series. When components = TRUE, the trend_{method} column is kept as well.

Each detrended column mirrors the name of the trend column it derives from: window vectors yield detrend_ma_6, detrend_ma_12, and a trend column renamed to avoid a naming conflict yields a matching detrended name.

With transform = "none" the additive identity value = trend + detrend holds exactly. With transform = "log" the identity value = trend * exp(detrend) holds instead. Methods with boundary effects (e.g. "bk", "hamilton") produce NA trend values at the affected observations, and the detrended series is NA there too.

Output rows are ordered by date within each group; the original row order is not preserved.

See Also

augment_trends() for the underlying trend extraction and the full set of methods; deseason_series() to remove seasonality; decompose_series() for a full decomposition.

Examples

```
# HP-filter detrending (the default): adds a detrend_hp column
gdp_construction |>
  detrend_series(value_col = "index")

# Log deviation from trend (x 100 ~ percentage gap, the output-gap convention)
gdp_construction |>
  detrend_series(value_col = "index", transform = "log")
```

```
# Keep the fitted trend alongside the detrended series
gdp_construction |>
  detrend_series(value_col = "index", components = TRUE)

# Compare detrending methods side by side
gdp_construction |>
  detrend_series(value_col = "index", methods = c("hp", "stl", "loess"))

# Seasonal data: deseason first, then detrend the adjusted series
gdp_construction |>
  deseason_series(value_col = "index") |>
  detrend_series(value_col = "seasadj_stl")

# Grouped detrending: one trend per electricity sector
electricity |>
  detrend_series(group_cols = "name_series")
```

`df_to_ts` *Convert a data.frame into a time series (ts)*

Description

Converts a series, stored in a data.frame or tibble, into a ts object.

Usage

```
df_to_ts(x, date_col = "date", value_col = "value", frequency = 12)
```

Arguments

`x`	A `data.frame`, `tibble` or `data.table`.
`date_col`	Name of the date column. Defaults to `'date'`. Must be of class `Date`.
`value_col`	Name of the value column. Defaults to `'value'`. Must be `numeric`.
`frequency`	The frequency of the series. Can be a shortened string (e.g. "M" for monthly) or a number (e.g. 12).

Value

A `ts` object

Examples

```
ibc <- df_to_ts(ibcbr, value_col = "index", frequency = "M")
class(ibc)
plot(ibc)
```

electric *Electric Consumption Residential*

Description

Monthly residential electric consumption in Brazil (GWh).

Usage

```
electric
```

Format

A tibble with monthly observations:

date Date column

consumption Electric consumption in GWh

Source

Brazilian Central Bank SGS (code 1403)

electricity *Electricity Consumption by Sector*

Description

Monthly electricity consumption in Brazil by sector (GWh), in long format. Combines residential, commercial, and industrial sub-series from the Brazilian National Electric System Operator (ONS) as reported by the Brazilian Central Bank SGS.

Usage

```
electricity
```

Format

A tibble with monthly observations:

date Date column

name_series Sector identifier: `"electric_residential"`, `"electric_commercial"`, or `"electric_industrial"`

value Electricity consumption in GWh

Source

ONS via Brazilian Central Bank SGS (codes 1403 — residential, 1402 — commercial, 1404 — industrial).

See Also

electric for the residential-only wide-format series.

`extract_trends` *Extract trends from time series objects*

Description

Extract trend components from time series objects using various econometric methods. Designed for monthly and quarterly economic data analysis. Returns trend components as time series objects or a list of time series.

Usage

```
extract_trends(
  ts_data,
  methods = "stl",
  window = NULL,
  smoothing = NULL,
  band = NULL,
  align = NULL,
  params = list(),
  .quiet = FALSE
)
```

Arguments

`ts_data`	A time series object (`ts`, `xts`, or `zoo`) or any object convertible via tsbox.
`methods`	Character vector of trend methods. Options: `"hp"`, `"bk"`, `"cf"`, `"ma"`, `"stl"`, `"loess"`, `"spline"`, `"poly"`, `"bn"`, `"ucm"`, `"hamilton"`, `"spencer"`, `"ewma"`, `"wma"`, `"triangular"`, `"kernel"`, `"kalman"`, `"median"`, `"gaussian"`. Default is `"stl"`.
`window`	Unified window/period parameter for moving average methods (ma, wma, triangular, stl, ewma, median, gaussian). Must be positive. If NULL, uses frequency-appropriate defaults. For EWMA, the window is converted to the smoothing factor via `alpha = 2 / (window + 1)`. Cannot be used simultaneously with `smoothing` for EWMA method. For `ma`, `median`, and `henderson` methods, a numeric vector is accepted (e.g., `c(9, 13, 23)`), which runs the method once per window value and returns a named list with keys like `henderson_9`, `henderson_13`, `henderson_23`. Other methods ignore extra values (with a warning).
`smoothing`	Unified smoothing parameter for smoothing methods (hp, loess, spline, ewma, kernel, kalman). For hp: use large values (1600+) or small values (0-1) that get converted. For EWMA: specifies the alpha parameter (0-1) for traditional exponential smoothing. Cannot be used simultaneously with `window` for EWMA method. For kernel: multiplier of optimal bandwidth (1.0 = optimal, <1 = less smooth, >1 = more smooth). For kalman: controls the ratio of measurement to process noise (higher = more smoothing). For others: typically 0-1 range.

band — Unified band parameter for bandpass filters (bk, cf). Both values must be positive. For bk/cf: Provide as `c(low, high)` where low/high are periods in quarters, e.g., `c(6, 32)`.

align — Unified alignment parameter for moving average methods (ma, wma, triangular, gaussian). Valid values: `"center"` (default, uses surrounding values), `"right"` (causal, uses past values only), `"left"` (anti-causal, uses future values only). Note: triangular only supports `"center"` and `"right"`. If NULL, uses `"center"` as default.

params — Optional list of method-specific parameters for fine control:

- **HP Filter**: `hp_onesided` (logical, default FALSE) - Use one-sided (real-time) filter instead of two-sided
- **STL**: `stl_s_window` or `s.window` (numeric/"periodic", default "periodic") - Seasonal window, `stl_t_window` or `t.window` (numeric/NULL, default NULL) - Trend window, `stl_robust` or `robust` (logical, default FALSE) - Use robust fitting. Note: Both dot notation (`s.window`) and underscore notation (`stl_s_window`) are accepted.
- **Spline**: `spline_cv` (logical/NULL) - Cross-validation method: NULL (none), TRUE (leave-one-out), FALSE (GCV)
- **Polynomial**: `poly_degree` (integer, default 1), `poly_raw` (logical, default FALSE for orthogonal polynomials)
- **UCM**: `ucm_type` (character, default "level") - Model type: "level", "trend", or "BSM"
- **Others**: `bn_ar_order`, `hamilton_h`, `hamilton_p`, `kernel_type`, `kalman_measurement_noise`, `kalman_process_noise`, `median_endrule`, `gaussian_sigma`, `wma_weights`.
- **Note**: Alignment parameters (`ma_align`, `wma_align`, `triangular_align`, `gaussian_align`) can still be passed via `params` but it's recommended to use the unified `align` parameter instead.

.quiet — If `TRUE`, suppress informational messages.

Details

This function focuses on monthly (frequency = 12) and quarterly (frequency = 4) economic data. It uses established econometric methods with appropriate defaults:

- **HP Filter**: lambda=1600 (quarterly), lambda=14400 (monthly). Supports both two-sided and one-sided (real-time) variants
- **Baxter-King**: Bandpass filter for business cycles (6-32 quarters default)
- **Christiano-Fitzgerald**: Asymmetric bandpass filter
- **Moving Average**: Centered, frequency-appropriate windows
- **STL**: Seasonal-trend decomposition
- **Loess**: Local polynomial regression
- **Spline**: Smoothing splines
- **Polynomial**: Linear/polynomial trends
- **Beveridge-Nelson**: Permanent/transitory decomposition

- **UCM**: Unobserved Components Model (local level)
- **Hamilton**: Regression-based alternative to HP filter
- **Advanced MA**: EWMA with various implementations
- **Kernel Smoother**: Non-parametric regression with various kernel functions
- **Kalman Smoother**: Adaptive filtering for noisy time series
- **Median Filter**: Robust filtering using running medians to remove outliers
- **Gaussian Filter**: Weighted average with Gaussian (normal) density weights

Parameter Usage Notes:

- **HP Filter**: Use `hp_onesided=TRUE` for real-time analysis or when future data should not influence current estimates. One-sided filter is appropriate for nowcasting, policy analysis, and avoiding look-ahead bias. Default two-sided filter is optimal for historical analysis.
- **EWMA**: Use either `window` (converted to `alpha = 2 / (window + 1)`) OR `smoothing` (alpha parameter), not both
- **Kalman**: Use `smoothing` parameter or `params` list for fine control of noise parameters
- **Spline**: Use `spline_cv` to control cross-validation (NULL=none, TRUE=LOO-CV, FALSE=GCV)
- **Polynomial**: Use `poly_raw=FALSE` for orthogonal polynomials (more stable for degree > 2) or `poly_raw=TRUE` for raw polynomials. Warning issued for degree > 3 (overfitting risk).
- **UCM**: Choose model type - "level" (simplest), "trend" (time-varying slope), or "BSM" (with seasonal component, requires seasonal data)

Value

If single method, returns a `ts` object. If multiple methods, returns a named list of `ts` objects.

Examples

```
# Single method
hp_trend <- extract_trends(AirPassengers, methods = "hp")

# Multiple methods with unified smoothing
smooth_trends <- extract_trends(
  AirPassengers,
  methods = c("hp", "loess", "ewma"),
  smoothing = 0.3
)

# EWMA with window (alpha derived from window size)
ewma_window <- extract_trends(AirPassengers, methods = "ewma", window = 12)

# EWMA with alpha (traditional formula)
ewma_alpha <- extract_trends(AirPassengers, methods = "ewma", smoothing = 0.2)

# Moving averages with unified window
ma_trends <- extract_trends(
  AirPassengers,
  methods = c("ma", "wma", "triangular"),
```

```
  window = 8
)

# Bandpass filters with unified band
bp_trends <- extract_trends(
  AirPassengers,
  methods = c("bk", "cf"),
  band = c(6, 32)
)

# Moving average with right alignment (causal filter)
ma_causal <- extract_trends(
  AirPassengers,
  methods = "ma",
  window = 12,
  align = "right"
)

# Signal processing methods with specific parameters
finance_trends <- extract_trends(
  AirPassengers,
  methods = c("kalman", "gaussian"),
  window = 9,  # For Gaussian filter
  params = list(kalman_measurement_noise = 0.1)  # Kalman-specific parameter
)

# Spline with cross-validation options
spline_trends <- extract_trends(
  AirPassengers,
  methods = "spline",
  params = list(spline_cv = FALSE)  # Use GCV instead of default
)

# Polynomial with orthogonal vs raw polynomials
poly_trends <- extract_trends(
  AirPassengers,
  methods = "poly",
  params = list(poly_degree = 2, poly_raw = FALSE)  # Orthogonal (default)
)

# UCM with different model types
ucm_trends <- extract_trends(
  AirPassengers,
  methods = "ucm",
  params = list(ucm_type = "BSM")  # Basic Structural Model with seasonality
)

# HP Filter: One-sided (real-time) vs Two-sided (historical)
hp_realtime <- extract_trends(
  AirPassengers,
  methods = "hp",
  params = list(hp_onesided = TRUE)  # For nowcasting and real-time analysis
)
```

```
# STL with custom parameters via params (both notations work)
stl_custom1 <- extract_trends(
  AirPassengers,
  methods = "stl",
  params = list(s.window = 21, robust = TRUE)  # Dot notation
)

stl_custom2 <- extract_trends(
  AirPassengers,
  methods = "stl",
  params = list(stl_s_window = 21, stl_robust = TRUE)  # Underscore notation
)

# Advanced: fine-tune specific methods
custom_trends <- extract_trends(
  AirPassengers,
  methods = c("median", "kalman"),
  window = 7,
  params = list(median_endrule = "constant")
)
```

---

gdp_construction *GDP Construction Index*

Description

Quarterly Brazilian GDP construction sector index (Base: average 1995 = 100).

Usage

```
gdp_construction
```

Format

A tibble with quarterly observations:

date Date column

index Construction index value

Source

DEPEC/GEATI/COACE. Fetched from Brazilian Central Bank SGS (code 22087)

ibcbr *Central Bank Economic Activity Index*

Description

Monthly Central Bank Economic Activity Index (IBC-Br). The IBC-Br was built based on proxies for the evolution of agriculture, industry and service-sector products. The proxies are aggregated with weights derived from the tables of supply and use of the Brazilian National Accounts.

Usage

```
ibcbr
```

Format

A tibble with monthly observations:

date Date column
index Index (2003 = 100)

Source

BACEN. Fetched from Brazilian Central Bank SGS (code 24363).

metadata_series *Series Metadata*

Description

Metadata for all economic series included in the package.

Usage

```
metadata_series
```

Format

A tibble with metadata:

series_name Short series identifier
description Full series description
frequency Data frequency (D = daily, M = monthly, Q = quarterly)
source Data source
date_col Name of the date column in the dataset
value_col Name of the main value column(s) in the dataset
group_cols Grouping column(s) for long-format datasets, or NA
date_min First observation date
date_max Last observation date

Source

Various (BCB-SGS, ONS, CEPEA/ESALQ)

---

oil_derivatives *Oil Derivatives Production*

Description

Monthly production of petroleum derivatives in Brazil (thousand barrels/day).

Usage

```
oil_derivatives
```

Format

A tibble with monthly observations:

date Date column

production Oil derivatives production

Source

ANP. Fetched from Brazilian Central Bank SGS (code 1391).

---

retail_autofuel *UK Retail Sales - Automotive Fuel*

Description

Chained volume of retail sales for automotive fuel in the UK, non-seasonally adjusted. Index numbers of sales per week (100 = 2023). The monthly period consists of 4 weeks except for March, June, September and December which are 5 weeks. January 2025 is also a 5 week period. The series considers the "All Businesses" specification and covers Great Britain from 1998 to 2025.

Usage

```
retail_autofuel
```

Format

A tibble with monthly observations:

date Date column

value Retail sales index (chained volume)

name Series name

frequency Frequency ("M")

source Data source ("ONS")

Source

UK Office for National Statistics (ONS). (Table 3M).

See Also

retail_volume

`retail_volume` *UK Retail Index*

Description

Chained volume of retail sales, non-seasonally adjusted, selected sub-series. Index numbers of sales per week (100 = 2023). The monthly period consists of 4 weeks except for March, June, September and December which are 5 weeks. January 2025 is also a 5 week period. The included series consider the "All Businesses" specification and cover Great Britain from 1998 to 2025.

Usage

```
retail_volume
```

Format

A tibble with monthly observations:

date Date column

name_series Series name derived from the unofficial SIC

value Retail sales index (chained volume)

Source

UK Office for National Statistics (ONS). (Table 3M).

See Also

retail_autofuel

transit_london_avgs *London Transit - Average Daily Journeys*

Description

Average daily journeys on London's bus and tube networks, split by business day and non-business day. Aggregated from daily Transport for London (TfL) network demand data using the UK calendar.

Usage

```
transit_london_avgs
```

Format

A tibble with monthly observations:

date_month First day of each month (Date)

transit_mode Transit mode: `"bus"` or `"tube"`

is_business_day 1 for business days, 0 for weekends/holidays

avg_daily_journeys Average daily journeys for the given day type

Source

Transport for London (TfL) network demand data.

See Also

transit_london_monthly

transit_london_monthly *London Transit - Monthly Journey Totals*

Description

Monthly total journeys on London's bus and tube (underground) networks. Aggregated from daily Transport for London (TfL) network demand data.

Usage

```
transit_london_monthly
```

Format

A tibble with monthly observations:

date_month First day of each month (Date)

transit_mode Transit mode: "bus" or "tube"

journey_monthly Total journeys in the month

Source

Transport for London (TfL) network demand data.

See Also

transit_london_avgs

---

ts_to_df *Convert time series to tibble*

Description

Convert time series to tibble

Usage

```
ts_to_df(x, date_col = NULL, value_col = NULL)
```

Arguments

x	A time series as a ts object
date_col	Optional name for the date column
value_col	Optional name for the value column

Value

a tibble

Examples

```
# example code
ts_to_df(AirPassengers)

# Using a custom name for the value column
ts_to_df(AirPassengers, value_col = "passengers")
```

`vehicles` *Vehicle Production*

Description

Monthly vehicle production in Brazil (units).

Usage

```
vehicles
```

Format

A tibble with monthly observations:

date Date column

production Vehicle production in absolute units

Source

Anfavea. Fetched from Brazilian Central Bank SGS (code 1378).

Index

∗ **datasets**
  coffee_arabica, 5
  coffee_robusta, 6
  electric, 18
  electricity, 18
  gdp_construction, 23
  ibcbr, 24
  metadata_series, 24
  oil_derivatives, 25
  retail_autofuel, 25
  retail_volume, 26
  transit_london_avgs, 27
  transit_london_monthly, 27
  vehicles, 29

augment_trends, 2
augment_trends(), *9*, *13–16*

coffee_arabica, 5, *6*
coffee_robusta, *6*, 6
converters, 7

decompose_series, 7
decompose_series(), *11–13*, *16*
deseason_series, 11
deseason_series(), *16*
detrend_series, 14
df_to_ts, 17

electric, 18, *19*
electricity, 18
extract_trends, 19

gdp_construction, 23

ibcbr, 24

metadata_series, 24

oil_derivatives, 25

retail_autofuel, 25, *26*
retail_volume, *26*, 26

transit_london_avgs, 27, *28*
transit_london_monthly, *27*, 27
ts_to_df, 28

vehicles, 29