

Package: rjd3highfreq (via r-universe)

September 21, 2026

Type Package

Title Seasonal Adjustment of High Frequency Data with 'JDemetra+ 3.x'

Version 3.0.0

Description R interface to 'JDemetra+ 3.x'

(<https://github.com/jdemetra>), the time series analysis software. It provides functions for seasonal adjustment of higher-frequency data, including those with multiple or non-integer periodicities. It includes pre-adjustment via the extended airline model and several seasonal decomposition methods: extended airline model based decomposition, extended X-11, and augmented STL.

Depends R (>= 4.1.0)

Imports rJava (>= 1.0-6), rjd3toolkit (>= 3.7.1), rjd3xjars (>= 0.1.0)

SystemRequirements Java (>= 21)

License file LICENSE

URL <https://github.com/rjdverse/rjd3highfreq>,
<https://rjdverse.github.io/rjd3highfreq/>,
<https://rjdverse.github.io/rjd3x11plus>

LazyData TRUE

Suggests knitr, rmarkdown, testthat

BugReports <https://github.com/rjdverse/rjd3highfreq/issues>

VignetteBuilder knitr

Encoding UTF-8

Collate 'auxiliaries.R' 'ic_r.R' 'jd3_extendedairline.R'
'jd3_fractionalairline.R' 'jd3_stl.R' 'jd3_x11.R' 'plot.R'
'print.R' 'summary.R' 'x11correction.R' 'zzz.R'

Config/roxygen2/version 8.0.0

Config/pak/sysreqs make default-jdk libprotobuf-dev protobuf-compiler libprotoc-dev

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

Date/Publication 2026-09-16 05:15:35 UTC
RemoteUrl <https://github.com/rjdverse/rjd3highfreq>
RemoteRef HEAD
RemoteSha 178cb1bfc066d5f3487f6555dec9bd5581b73803

Contents

extreme_values_correction	2
find_outliers	3
fractional_airline_decomposition	4
fractional_airline_estimation	6
henderson	8
ic_ratio	9
istl	10
loess	12
loglevel	13
mstl	14
multi_airline_decomposition	16
stlplus	18
x11plus	20
Index	24

extreme_values_correction
<i>'X11' Extreme Values Corrector</i>

Description

'X11' Extreme Values Corrector

Usage

```
extreme_values_correction(  
  series,  
  period,  
  corrected_s,  
  lsigma = 1.5,  
  usigma = 2.5,  
  multiplicative = FALSE,  
  start = 0,  
  clean_extremities = TRUE  
)
```

Arguments

series	the analysed time series.
period	the period of the input time series if 'series' is not a "ts" object.
corrected_s	other time series if the series being corrected is different from series.
lsigma	the lower sigma boundaries for the detection of extreme values.
usigma	the upper sigma boundaries for the detection of extreme values.
multiplicative	boolean indicating if the decomposition is multiplicative or additive.
start	position of the first "complete" considered period.
clean_extremities	boolean indicating if the extremities should be cleaned.

Details

The 'X11' Extreme Values Corrector is used to compute the tables b4, b4g, b9, b9g, b17, b20, c17 and c20.

#' The returned correction contains the following columns:

obs_weight Extreme value corrections weight of each observation

correction_factors Extreme value correction factors

Examples

```
extreme_values_correction(rjd3toolkit::ABS$X0.2.09.10.M, 12)
```

find_outliers

Outlier Detection for Extended Airline Models

Description

This internal function performs automatic outlier detection in high-frequency time series using Extended Airline models.

Usage

```
find_outliers(
  input,
  types = c("A0"),
  start = 0,
  end = 0,
  critical_value = 0,
  max_outliers = 30,
  max_round = 30,
  precision = 1e-05,
  deps = 1e-04
)
```

Arguments

input	time series
types	Character vector specifying the types of outliers to detect. Default: c("AO") (additive outliers only). Common options include: • "AO" - Additive Outlier • "LS" - Level Shift • "WO" - Switch Outlier
start	Integer specifying the starting position (1-based R indexing) for outlier detection. Default: 0 (detection from the beginning of the series).
end	Integer specifying the ending position (1-based R indexing) for outlier detection. Default: 0 (detection until the end of the series).
critical_value	Numeric value for the critical value threshold. Uses the maximum value among the specified value and a global max-t threshold based on extreme-value theory, roughly of order $\sqrt{2 * \log(n)}$. Default: 0 (global max-t threshold is used).
max_outliers	Integer specifying the maximum number of outliers to detect. Default: 30.
max_round	Integer specifying the maximum number of detection iterations. Default: 30.
precision	Numeric value specifying the tolerance for convergence of optimization algorithms. Default: 1e-5.
deps	Numeric scalar. Step in the computation of the numerical derivatives, used in the optimisation routine. Default: 1e-4.

Value

A numeric matrix with dimensions [number of outliers detected $\times$ 2].

References

Outlier critical value using the Ljung algorithm as given in Ljung, G. M. (1993). On outlier detection in time series. *Journal of Royal Statistical Society B* 55, 559-567. Solution proposed by Brian Monsell (LBS), January 2022

Examples

```
series <- rjd3toolkit::Exports$Malta
find_outliers(series, critical_value = 3.5, types = c("LS", "WO"))
```

fractional_airline_decomposition

Perform an Arima Model Based (AMB) decomposition

Description

Performs an Arima Model Based (AMB) decomposition using a (fractional) airline model, suitable for high-frequency time series. The method decomposes the input series into trend, seasonal and irregular components, with optional signal–noise decomposition.

Usage

```
fractional_airline_decomposition(
  series,
  period,
  sn = FALSE,
  stde = FALSE,
  nbcasts = 0,
  nfcasts = 0,
  eps = 1e-09,
  deps = 1e-04,
  log = FALSE,
  series_time = NULL
)
```

Arguments

series	input time series.
period	period of the seasonal component, any positive real number.
sn	decomposition into signal and noise (2 components only). The signal is the seasonally adjusted series and the noise the seasonal component. Default: FALSE.
stde	Boolean: TRUE: compute standard deviations of the components. In some cases (e.g. memory limits), it is currently not possible to compute them. Default: FALSE.
nbcasts	number of backcasts. Default: 0.
nfcasts	number of forecasts. Default: 0.
eps	precision of the optimisation routine. Default: 1e-9.
deps	step in the computation of the numerical derivatives, used in the optimisation routine. Default: 1e-4
log	logical indicating whether the series is on the log scale. Default: FALSE.
series_time	vector of times at which ‘series’ is indexed. Optional.

Details

If ‘sn = TRUE’, the decomposition is restricted to two components only (signal and noise). When ‘stde = TRUE’, the computation of standard deviations may fail for long series or high-frequency data due to memory constraints.

Value

An object containing the AMB decomposition results, including the estimated components and, if requested, their standard deviations.

Examples

```
series <- rnorm(70)+100

### Example with a daily time series with a day-of-the-week effect
weekday <- rjd3highfreq::fractional_airline_decomposition(
  series,
  period = 7,
  log = TRUE,
  series_time = seq.Date(from=as.Date("2025-01-01"),
                        by = "days",
                        length.out = length(series))
)

### Example with a weekly time series
series <- rnorm(200)+100
weekly <- rjd3highfreq::fractional_airline_decomposition(
  series,
  period = 52.18,
  log = TRUE,
  series_time = seq.Date(from=as.Date("2025-01-01"),
                        by = "days",
                        length.out = length(series))
)
```

fractional_airline_estimation

Linearize a time series using a fractional airline model

Description

This function estimates a (fractional) airline RegARIMA model and returns the linearized series together with regression effects, outlier components, estimation results and likelihood diagnostics. It is typically used as a preprocessing step prior to AMB or UCM-based decompositions.

Usage

```
fractional_airline_estimation(
  series,
  period,
  xreg = NULL,
  ndiff = 2,
  ar = FALSE,
  mean = FALSE,
  outliers = NULL,
  critical_value = 6,
  precision = 1e-12,
  deps = 1e-04,
  approximate_hessian = FALSE,
```

```

    nfcasts = 0,
    log = FALSE,
    series_time = NULL
  )

```

Arguments

<code>series</code>	input time series.
<code>period</code>	numeric vector of seasonal periods. Each value must be a positive real number (e.g. 7 for weekly, 365.2425 for annual seasonality).
<code>xreg</code>	optional matrix of user-defined regression variables (e.g. calendar regressors built using <code>rjd3toolkit</code>).
<code>ndiff</code>	integer specifying the number of regular differences. Default is 2.
<code>ar</code>	logical. If TRUE, an autoregressive component is included in the model. Default is FALSE.
<code>mean</code>	logical. If TRUE, a mean component is included in the model. Default is FALSE.
<code>outliers</code>	character vector specifying the types of outliers to detect. Possible values include "AO", "LS" and "WO". Default is NULL (no automatic outlier detection).
<code>critical_value</code>	numeric. Critical value for automatic outlier detection. Larger values imply more conservative detection. Default is 6.
<code>precision</code>	numeric. Precision of the likelihood optimization. Default is 1e-12.
<code>deps</code>	step in the computation of the numerical derivatives, used in the optimisation routine. Default: 1e-4
<code>approximate_hessian</code>	logical. If TRUE, compute an approximate Hessian matrix based on the optimization procedure. Default is FALSE.
<code>nfcasts</code>	number of forecasts. Default is 0.
<code>log</code>	logical. If TRUE, the model is estimated on the log-scale. Default is FALSE.
<code>series_time</code>	optional vector of time indices associated with <code>series</code> .

Details

Automatic outlier detection can be enabled by specifying the outlier types and a critical value for the detection threshold.

Value

An object of class "hf_estimation" containing:

- the original and linearized series,
- estimated regression effects and outlier components,
- model parameters and covariance matrices,
- likelihood and diagnostic information.

Examples

```
# Simulated examples with and without regressor
set.seed(125)
reg <- data.frame(
  reg1 = round(runif(1000, min = 0, max = 1))*2-1,
  reg2 = round(runif(1000, min = 0, max = 1))*2-1
)

y = 100 + 2*reg$reg1 -2*reg$reg2 + rnorm(nrow(reg))

# input data
data <- list(
  series = y,
  date = seq.Date(from = as.Date("2020-01-01"),
    by = "day",
    length.out = 1000)
)

# Linearize the series using weekly and annual periodicities
est <- fractional_airline_estimation(
  data$series,
  period = c(7, 30.4),
  log = FALSE,
  xreg = reg,
  series_time = data$date
)

est

est2 <- fractional_airline_estimation(
  data$series,
  period = c(7, 30.4),
  log = FALSE,
  series_time = data$date
)

est2
```

henderson

*Apply Henderson linear filter***Description**

Apply Henderson linear filter

Usage

henderson(series, length, musgrave = TRUE, ic = 4.5)

Arguments

series	Input time series.
length	Length of the symmetric Henderson filter. Must be odd.
musgrave	Boolean indicating if Musgrave asymmetric filters should be used. Default is TRUE. If FALSE, no asymmetric filters will be used.
ic	I/C ratio between the volatility of the tentative irregular and trend-cycle estimates (needed for calculating the Musgrave asymmetric filters).

Value

A numeric array corresponding to the estimated trend.

Examples

```
q <- x11plus(rjd3toolkit::ABS$X0.2.09.10.M, 12)
henderson(q$decomposition$sa, 13)
```

ic_ratio	<i>Compute IC-Ratio</i>
----------	-------------------------

Description

Compute IC-Ratio

Usage

```
ic_ratio(
  series,
  frequency = NULL,
  trend_length = NULL,
  musgrave = FALSE,
  multiplicative = FALSE,
  seasonally_adjust = TRUE,
  sma = "S3X5",
  ...
)
```

Arguments

series	input time series
frequency	frequency of the seasonal component. Can be NULL if series is ts
trend_length	length of Henderson filter used for trend adjustment. If NULL it is nextodd(frequency)
musgrave	Boolean indicating if Musgrave asymmetric filters should be used in trend adjustment

`multiplicative` boolean indicating if the decomposition is multiplicative or additive
`seasonally_adjust` boolean. Should series be seasonally adjusted?
`sma` seasonal moving average used in `x11plus()`
`...` additional arguments for trend adjustment. See `?henderson`

Details

In the traditional calculation of the IC-ratio, no asymmetric filters used for the preliminary trend estimation (see Ladiray and Quenneville 2001, Table B7).

References

Ladiray, D., Quenneville, B. (2001). The Various Tables. In: Seasonal Adjustment with the X-11 Method. Lecture Notes in Statistics, vol 158. Springer, New York, NY. https://doi.org/10.1007/978-1-4613-0175-2_5

Examples

```
series <- rjd3toolkit::Retail$AllOtherGenMerchandiseStores
ic_ratio(series)
```

istl

Iterative multiple seasonal decomposition using ISTL

Description

Performs an iterative `stlplus`-type decomposition for time series with multiple seasonal periods. This function provides an R interface to the JD+ Java implementation. It decomposes a time series into a trend component, several seasonal components, and an irregular component using an iterative STL-based procedure.

Usage

```
istl(
  series,
  period,
  multiplicative = TRUE,
  swindow = NULL,
  twindow = NULL,
  robust = NULL,
  ninnerloop = 1,
  nouterloop = 15,
  nojump = FALSE,
  weight_threshold = 0.001,
  weight_function = c("biweight", "uniform", "triangular", "epanechnikov", "tricube",
    "triweight")
)
```

Arguments

<code>series</code>	Numeric vector. Input time series to be decomposed.
<code>period</code>	Numeric vector. Seasonal periods to be modelled. For example, <code>c(7, 365)</code> for daily data with weekly and yearly seasonal patterns. Values are passed to the underlying Java implementation as integers.
<code>multiplicative</code>	Logical. If TRUE, a multiplicative decomposition is used. If FALSE, an additive decomposition is used.
<code>swindow</code>	Optional integer vector. Lengths of the seasonal smoothing windows, one for each seasonal period. If NULL, the seasonal windows are selected automatically by the underlying Java implementation.
<code>twindow</code>	Optional integer vector. Lengths of the trend smoothing windows. If NULL, the trend windows are selected automatically by the underlying Java implementation.
<code>robust</code>	Boolean. Analogue to robust parameter in <code>stats::stl</code> (see details)
<code>ninnerloop</code>	Integer. Number of inner iterations of the ISTL algorithm.
<code>nouterloop</code>	Integer. Number of outer iterations used to compute robust weights. Set to 0 to disable robust fitting.
<code>nojump</code>	Logical. If TRUE, disables jump-based acceleration in the smoothing computations. If FALSE, the underlying implementation may use jumps to speed up the decomposition.
<code>weight_threshold</code>	Numeric scalar in $[0, 0.3]$. Threshold used in the computation of robust weights.
<code>weight_function</code>	Character string specifying the weighting function used by the LOESS smoothers. One of "biweight", "uniform", "triangular", "epanechnikov", "tricube" or "triweight".

Details

The returned decomposition contains the following columns:

<code>series</code>	The original input series.
<code>sa</code>	The seasonally adjusted series (trend+irregular for additive, trend*irregular for multiplicative decomposition).
<code>t</code>	The trend component.
<code>s<period></code>	One seasonal component for each value supplied in period. For example, if <code>period = c(7, 365)</code> , the output contains columns <code>s7</code> and <code>s365</code> .
<code>i</code>	The irregular component.
<code>fit</code>	The fitted values from the decomposition (trend+seasonal for additive, trend*seasonal for multiplicative decomposition).
<code>weights</code>	The final robust weights.

If `multiplicative = TRUE`, the decomposition is interpreted as a multiplicative decomposition. If `multiplicative = FALSE`, it is interpreted as an additive decomposition.

If `robust = TRUE`, the parameters `nouterloop` and `ninnerloop` are overwritten, so that 15 iterations of the outer loop and one run of the inner loop are completed. If `robust = FALSE`, `nouterloop` is set to 0, and `ninnerloop` is set to 2.

Value

An object of class "hf_decomposition", consisting of a list with two elements:

`decomposition` A data.frame containing the original series, the seasonally adjusted series, the trend, one seasonal component for each period, the irregular component, fitted values and robust weights.

`parameters` A list containing the main parameters used for the decomposition.

See Also

[stlplus](#), [mstl](#), [plot.hf_decomposition](#)

Examples

```
q <- istl(
  series = rjd3toolkit::ABS$X0.2.09.10.M,
  period = c(12, 19)
)

plot(q)
```

loess	<i>Fit a LOESS smoother</i>
-------	-----------------------------

Description

Applies a LOESS smoother to a numeric time series.

Usage

```
loess(series, window, degree = 1, jump = 0)
```

Arguments

- | | |
|---------------------|--|
| <code>series</code> | Numeric vector. Input time series to be smoothed. |
| <code>window</code> | Integer. Length of the LOESS smoothing window. Larger values produce a smoother result. |
| <code>degree</code> | Integer. Degree of the local polynomial. Supported values are 0 for locally constant smoothing and 1 for locally linear smoothing. The default is 1. |
| <code>jump</code> | Integer. Number of jumps used to speed up the computation. If <code>jump = 0</code> , the smoother is evaluated at every observation. Values greater than zero evaluate the smoother at fewer points and interpolate between them. The default is 0. |

Details

This function provides a simple R interface to the JD+ Java implementation of the LOESS smoother used internally by the stlplus decomposition routines. It smoothes the input series using a local polynomial of degree 0 or 1.

This function is not the same as [loess](#) from the **stats** package. It does not use a formula interface and is intended for smoothing a single numeric series with the JD+ stlplusloess implementation. Also, it can handle time series with missing values.

Value

A numeric vector containing the smoothed series.

See Also

[stlplus](#), [mstl](#), [istl](#), [loess](#)

Examples

```
q <- stlplus(
  series = rjd3toolkit::ABS$X0.2.09.10.M,
  period = 12
)

trend <- q$decomposition[, "t"]

smoothed_trend <- loess(
  series = trend,
  window = 121
)

ts.plot(
  cbind(trend, smoothed_trend),
  col = c("black", "red")
)
```

loglevel

Log-Level test for Extended Airline Models

Description

This function performs a statistical test to determine whether data should be transformed to logarithmic scale or kept in level (original scale)

Usage

```
loglevel(input, precision = 1e-05, deps = 1e-04)
```

Arguments

input	time series
precision	Numeric value specifying the tolerance for convergence of optimization algorithms. Default: 1e-5.
deps	Numeric scalar. Step in the computation of the numerical derivatives, used in the optimisation routine. Default: 1e-4.

Value

An object containing the log-level test results. First the AICc of the model in levels and then the AICc of the model with logs

Examples

```
series <- rjd3toolkit::Retail$BookStores
loglevel(series)
```

mstl

Simultaneous, multiple seasonal decomposition using MSTL

Description

Performs an stlplus-type decomposition for time series with multiple seasonal periods. This function provides an R interface to the JD+ Java implementation.. It simultaneously decomposes a time series into a trend component, several seasonal components, and an irregular component.

Usage

```
mstl(
  series,
  period,
  multiplicative = TRUE,
  swindow = NULL,
  twindow = 0,
  robust = NULL,
  ninnerloop = 1,
  nouterloop = 15,
  nojump = FALSE,
  weight_threshold = 0.001,
  weight_function = c("biweight", "uniform", "triangular", "epanechnikov", "tricube",
    "triweight")
)
```

Arguments

<code>series</code>	Numeric vector. Input time series to be decomposed.
<code>period</code>	Numeric vector. Seasonal periods to be modelled. For example, <code>c(7, 365)</code> for daily data with weekly and yearly seasonal patterns. Values are passed to the underlying Java implementation as integers.
<code>multiplicative</code>	Logical. If TRUE, a multiplicative decomposition is used. If FALSE, an additive decomposition is used.
<code>swindow</code>	Optional integer vector. Lengths of the seasonal smoothing windows, one for each seasonal period. If NULL, the seasonal windows are selected automatically by the underlying Java implementation.
<code>twindow</code>	Integer. Length of the trend smoothing window. If set to 0, the value is selected automatically by the underlying Java implementation.
<code>robust</code>	Boolean. Analogue to robust parameter in <code>stats::stl</code> (see details)
<code>ninnerloop</code>	Integer. Number of inner iterations of the MSTL algorithm.
<code>nouterloop</code>	Integer. Number of outer iterations used to compute robust weights. Set to 0 to disable robust fitting.
<code>nojump</code>	Logical. If TRUE, disables jump-based acceleration in the smoothing computations. If FALSE, the underlying implementation may use jumps to speed up the decomposition.
<code>weight_threshold</code>	Numeric scalar in $[0, 0.3]$. Threshold used in the computation of robust weights.
<code>weight_function</code>	Character string specifying the weighting function used by the LOESS smoothers. One of "biweight", "uniform", "triangular", "epanechnikov", "tricube" or "triweight".

Details

The returned decomposition contains the following columns:

<code>series</code>	The original input series.
<code>sa</code>	The seasonally adjusted series (trend+irregular for additive, trend*irregular for multiplicative decomposition).
<code>t</code>	The trend component.
<code>s<period></code>	One seasonal component for each value supplied in period. For example, if <code>period = c(7, 365)</code> , the output contains columns <code>s7</code> and <code>s365</code> .
<code>i</code>	The irregular component.
<code>fit</code>	The fitted values from the decomposition (trend+seasonal(s) for additive, trend*seasonal(s) for multiplicative decomposition).
<code>weights</code>	The final robust weights.

If `multiplicative = TRUE`, the decomposition is interpreted as a multiplicative decomposition. If `multiplicative = FALSE`, it is interpreted as an additive decomposition.

If `robust = TRUE`, the parameters `nouterloop` and `ninnerloop` are overwritten, so that 15 iterations of the outer loop and one run of the inner loop are completed. If `robust = FALSE`, `nouterloop` is set to 0, and `ninnerloop` is set to 2.

Value

An object of class "hf_decomposition", consisting of a list with two elements:

`decomposition` A data.frame containing the original series, the seasonally adjusted series, the trend, one seasonal component for each period, the irregular component, fitted values and robust weights.

`parameters` A list containing the main parameters used for the decomposition.

See Also

[stlplus](#), [plot.hf_decomposition](#)

Examples

```
q <- mstl(
  series = rjd3toolkit::ABS$X0.2.09.10.M,
  period = c(12, 19)
)

plot(q)
```

`multi_airline_decomposition`

Perform an Arima Model Based (AMB) decomposition with multiple periodicities

Description

This function performs an AMB decomposition based on (fractional) airline models allowing for multiple seasonal periodicities at once. It is intended for high-frequency time series where more than one seasonal cycle may be present (e.g. weekly and annual effects).

Usage

```
multi_airline_decomposition(
  series,
  period,
  ndiff = 2,
  ar = FALSE,
  stde = FALSE,
  nbcasts = 0,
  nfcasts = 0,
  eps = 1e-09,
  deps = 1e-04,
  log = FALSE,
  series_time = NULL
)
```

stlplus

Seasonal decomposition using STL+

Description

Performs an STL-like seasonal decomposition. It can handle missing values and does allow a multiplicative decomposition.

Usage

```
stlplus(
  series,
  period,
  multiplicative = TRUE,
  swindow = 7,
  twindow = 0,
  lwindow = 0,
  sdegree = 0,
  tdegree = 1,
  ldegree = 1,
  sjump = 0,
  tjump = 0,
  ljump = 0,
  robust = NULL,
  ninnerloop = 1,
  nouterloop = 15,
  weight_threshold = 0.001,
  weight_function = c("biweight", "uniform", "triangular", "epanechnikov", "tricube",
    "triweight"),
  legacy = FALSE
)
```

Arguments

series	Numeric vector. Input time series to be decomposed.
period	Numeric scalar. Seasonal period of the series. For example, use 12 for monthly data with yearly seasonality, 4 for quarterly data, or 7 for daily data with weekly seasonality. In the current implementation, this value is passed to Java as an integer.
multiplicative	Logical. If TRUE, a multiplicative decomposition is used. If FALSE, an additive decomposition is used.
swindow	Integer. Length of the seasonal smoothing window.
twindow	Integer. Length of the trend smoothing window. If set to 0, the value is selected automatically by the underlying Java implementation.

lwindow	Integer. Length of the low-pass filter used to remove the trend from the seasonal component. If set to 0, the value is selected automatically by the underlying Java implementation.
sdegree	Integer. Degree of the local polynomial used for seasonal smoothing. Usually 0 or 1.
tdegree	Integer. Degree of the local polynomial used for trend smoothing. Usually 0 or 1.
ldegree	Integer. Degree of the local polynomial used for low-pass smoothing. Usually 0 or 1.
sjump	Integer. Number of jumps used in the computation of the seasonal component. Values greater than zero speed up the computation by evaluating the smoother at fewer points and interpolating between them.
tjump	Integer. Number of jumps used in the computation of the trend component.
ljump	Integer. Number of jumps used in the computation of the low-pass component.
robust	Boolean. Analogue to robust parameter in stats::stl (see details)
ninnerloop	Integer. Number of inner iterations of the STL algorithm.
nouterloop	Integer. Number of outer iterations used to compute robust weights. Set to 0 to disable robust fitting.
weight_threshold	Numeric scalar in $[0, 0.3]$. Threshold used in the computation of robust weights.
weight_function	Character string specifying the weighting function used by the LOESS smoothers. One of "biweight", "uniform", "triangular", "epanechnikov", "tricube" or "triweight".
legacy	Logical. If TRUE, uses the legacy MAD computation of the underlying implementation. This option is mainly provided for backward compatibility.

Details

This function provides an R interface to the JD+ Java implementation of STL decomposition. It decomposes a time series into trend, seasonal and irregular components and returns the result as an object of class "hf_decomposition".

The returned decomposition contains the following columns:

series The original input series.

sa The seasonally adjusted series (trend+irregular for additive, trend*irregular for multiplicative decomposition).

t The trend component.

s The seasonal component.

i The irregular component.

fit The fitted values from the decomposition (trend+seasonal(s) for additive, trend*seasonal(s) for multiplicative decomposition).

weights The final robust weights.

If `multiplicative = TRUE`, the decomposition is interpreted as a multiplicative decomposition. If `multiplicative = FALSE`, it is interpreted as an additive decomposition.

If `robust = TRUE`, the parameters `nouterloop` and `ninnerloop` are overwritten, so that 15 iterations of the outer loop and one run of the inner loop are completed. If `robust = FALSE`, `nouterloop` is set to 0, and `ninnerloop` is set to 2.

Value

An object of class "hf_decomposition", consisting of a list with two elements:

`decomposition` A data.frame containing the original series, the seasonally adjusted series, the trend, seasonal and irregular components, fitted values and robust weights (see details).

`parameters` A list containing the main parameters used for the decomposition.

See Also

[plot.hf_decomposition](#)

Examples

```
decomp <- stlplus(
  series = rjd3toolkit::ABS$X0.2.09.10.M,
  period = 12
)

plot(decomp)
```

x11plus

Modified X-11 seasonal adjustment

Description

This function implements a modified version of the classical X-11 seasonal adjustment approach. The modified X-11 method is applicable to time series with arbitrary seasonal periodicity.

Usage

```
x11plus(
  series,
  period,
  multiplicative = TRUE,
  trend_horizon = 6,
  trend_degree = 3,
  trend_kernel = c("Henderson", "BiWeight", "TriWeight", "TriCube", "Uniform",
    "Triangular", "Epanechnikov", "Trapezoidal"),
  trend_asymmetric = c("CutAndNormalize", "Direct", "MMSRE"),
  trend_coefs,
```

```

seas_s0 = c("S3X3", "S3X1", "S3X5", "S3X9", "S3X15"),
seas_s1 = c("S3X5", "S3X3", "S3X1", "S3X9", "S3X15"),
extreme_lsig = 1.5,
extreme_usig = 2.5,
user_defined = NULL
)

```

Arguments

series	Input time series.
period	Seasonal periodicity of series. Must be a positive real number larger than or equal to 2.
multiplicative	Decomposition mode for series (Boolean). If TRUE (default), then series is decomposed into multiplicative trend-cyclical, seasonal, and irregular components; if FALSE, then it is decomposed into additive unobserved components.
trend_horizon	Bandwidth of the symmetric local polynomial regression filter. Default is 6, giving a 13-term filter. See details.
trend_degree	Polynomial degree that should be preserved by the symmetric local polynomial regression filter. Default is 3, giving preservation of cubic polynomials. See details.
trend_kernel	A kernel defining the weights in the objective function. See details.
trend_asymmetric	Approach for deriving asymmetric local polynomial regression filters. See details.
trend_coefs	A filter bank containing the weights of the symmetric trend-cycle filter and its asymmetric variants. Can be an object of class "list", "matrix", "lp_filter" or "rkhs_filter". See details.
seas_s0	$3 \times k$ seasonal filter for preliminary seasonal estimation (B5, C5, D5). Default is 3×3 .
seas_s1	$3 \times k$ seasonal filter for refined and final seasonal estimation (B10, C10, D10). Default is 3×5 .
extreme_lsig	Lower σ -limit for extreme-value correction in the seasonal-irregular component. Must be non-negative, default is 1.5.
extreme_usig	Upper σ -limit for extreme-value correction in the seasonal-irregular component. Must be greater than extreme_lsig, default is 2.5.
user_defined	A vector containing additional output tables. Default is NULL.

Details

The main novelty of the modified X-11 method is the implementation of advanced options for refined and final trend-cycle estimation (B7, C7, D7, and D12), generalising the use of classical Henderson filters. The following logic applies, see also Webel (2026) and Webel and Smyk (2024) for technical details:

- If `trend_coefs` is unspecified (default), then the local polynomial regression filters suggested in Proietti and Luati (2008) are applied. The underlying regression model, from which the symmetric trend-cycle filter arises, is further specified through the `trend_horizon`, `trend_degree`, and `trend_kernel` parameters:
 - $2 * \text{trend_horizon} + 1$ is the number of observations to be considered in the local trend approximation and, hence, the length of the resulting symmetric trend-cycle filter.
 - `trend_degree` is the order of the polynomial in the local trend approximation.
 - `trend_kernel` is a kernel function that defines the sequence of non-negative weights in the objective function, which is then minimised with respect to the regression parameters.

In addition, `trend_asymmetric` defines the method to be used for deriving the requisite asymmetric local polynomial regression filters. Three methods are currently available: the cut-and-normalise approach of Gasser and Müller (1979) (`CutAndNormalize`), the direct asymmetric filters suggested in Proietti and Luati (2008) (`Direct`), and the minimum mean squared revision error approach developed in Grun-Rehomme et al. (2018) (`MMSRE`).

- If `trend_coefs` is specified, then all other parameters related to trend-cycle estimation are ignored, and the provided trend-cycle filter bank is applied. Note that the **rjd3filters** package can be used to create filters, e.g. by setting `trend_coefs = rjd3filters::rkhs_filter()`.

Two final warnings regarding seasonal estimation.

1. Extremes in the seasonal-irregular component are currently corrected only in iterations B and C, where the moving irregular standard deviation is always calculated from 5-year moving windows. That is, no final replacement values (D9) are currently computed.
2. Users must specify seasonal filters `seas_s0` and `seas_s1` that match the length of series, as there is currently no automatic resetting of ill-specified seasonal filters. That is, any specification of seasonal filters that are too long for the given series will simply produce an error message.

Value

An object of class "hf_decomposition". It contains the specified parameters and a matrix that stores the input series and the final estimates of the seasonally adjusted series (D11), the trend-cyclical component (D12), the seasonal component (D10), and the irregular component (D13).

References

- Dagum, E. B. and S. Bianconcini (2008). The Henderson Smoother in Reproducing Kernel Hilbert Space. *Journal of Business and Economic Statistics* 26 (4), 536–545. <https://doi.org/10.1198/073500107000000322>
- Gasser, T. and H.-G. Müller (1979). Kernel Estimation of Regression Functions. In T. Gasser and M. Rosenblatt (Eds), *Smoothing Techniques for Curve Estimation*, 23–68. Heidelberg: Springer. <https://doi.org/10.1007/BFb0098489>
- Grun-Rehomme, M., F. Guggemos and D. Ladiray (2018). Asymmetric Moving Averages Minimizing Phase Shift. In G. L. Mazzi, D. Ladiray and D. A. Riester (Eds), *Handbook on Seasonal Adjustment*, 391–413. Luxembourg: Publications Office of the European Union.
- Proietti, T. and A. Luati (2008). Real Time Estimation in Local Polynomial Regression, with Application to Trend-Cycle Analysis. *Annals of Applied Statistics* 2 (4), 1523–1553. <https://doi.org/10.1214/08-AOAS195>

Webel, K. (2026). Some Thoughts on Modified X-11 Seasonal Adjustments for Time Series with Complex Seasonality. Deutsche Bundesbank Discussion Paper XX/2026. Forthcoming

Webel, K. and A. Smyk (2024). Seasonal Adjustment of Infra-Monthly Time Series with JDemetera+. Journal of Official Statistics 40 (4), 783–828. <https://doi.org/10.1177/0282423X241277602>

Examples

```
x11decomp <- x11plus(  
  series = rjd3toolkit::ABS$X0.2.09.10.M,  
  period = 12,  
  trend_horizon = 99  
)
```

Index

extreme_values_correction, [2](#)

find_outliers, [3](#)

fractional_airline_decomposition, [4](#)

fractional_airline_estimation, [6](#)

henderson, [8](#)

ic_ratio, [9](#)

istl, [10](#), [13](#)

loess, [12](#), [13](#)

loglevel, [13](#)

mstl, [12](#), [13](#), [14](#)

multi_airline_decomposition, [16](#)

plot.hf_decomposition, [12](#), [16](#), [20](#)

stlplus, [12](#), [13](#), [16](#), [18](#)

x11plus, [20](#)