Package: rjd3filters (via r-universe)

September 10, 2024

Type Package

Title Trend-Cycle Extraction with Linear Filters based on JDemetra+ v3.x

Version 2.1.1

Description This package provides functions to build and apply symmetric and asymmetric moving averages (= linear filters) for trend-cycle extraction. In particular, it implements several modern approaches for real-time estimates from the viewpoint of revisions and time delay in detecting turning points. It includes the local polynomial approach of Proietti and Luati (2008), the Reproducing Kernel Hilbert Space (RKHS) of Dagum and Bianconcini (2008) and the Fidelity-Smoothness-Timeliness approach of Grun-Rehomme, Guggemos, and Ladiray (2018). It is based on Java libraries developped in 'JDemetra+' (<https://github.com/jdemetra>), time series analysis software.

Depends R (>= 4.1.0)

Imports rJava (>= 1.0-6), methods, MASS, graphics, stats, rjd3toolkit (>= 3.2.2)

Remotes github::rjdverse/rjd3toolkit@*release

SystemRequirements Java (>= 17)

License EUPL

LazyData TRUE

URL https://github.com/rjdverse/rjd3filters,
https://rjdverse.github.io/rjd3filters/

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Encoding UTF-8

Repository https://rjdverse.r-universe.dev

RemoteUrl https://github.com/rjdverse/rjd3filters

RemoteRef v2.1.1

RemoteSha bcce59ddc574e3ef5884810f8b79c991d57c77c2

Contents

`confint_filter` *Confidence intervals*

Description

Confidence intervals

Usage

```
confint_filter(x, coef, coef_var = coef, level = 0.95, ...)
```

Arguments

`x`	input time series.
`coef`	moving-average (`moving_average()`) or finite filter (`finite_filters()`) used to filter the series.
`coef_var`	moving-average (`moving_average()`) or finite filter (`finite_filters()`) used compute the variance (throw `var_estimator()`). By default equal to `coef`.
`level`	confidence level.
`...`	other arguments passed to the function `moving_average()` to convert `coef` to a `"moving_average"` object.

Details

Let $(\theta_i)_{-p\leq i\leq q}$ be a moving average of length $p+q+1$ used to filter a time series $(y_i)_{1\leq i\leq n}$. Let denote $\hat{\mu}_t$ the filtered series computed at time t as:

$$\hat{\mu}_t = \sum_{i=-p}^{q} \theta_i y_{t+i}.$$

If $\hat{\mu}_t$ is unbiased, a approximate confidence for the true mean is:

$$\left[\hat{\mu}_t - z_{1-\alpha/2}\hat{\sigma}\sqrt{\sum_{i=-p}^{q} \theta_i^2}; \hat{\mu}_t + z_{1-\alpha/2}\hat{\sigma}\sqrt{\sum_{i=-p}^{q} \theta_i^2}\right],$$

where $z_{1-\alpha/2}$ is the quantile $1-\alpha/2$ of the standard normal distribution.

The estimate of the variance $\hat{\sigma}$ is obtained using `var_estimator()` with the parameter `coef_var`. The assumption that $\hat{\mu}_t$ is unbiased is rarely exactly true, so variance estimates and confidence intervals are usually computed at small bandwidths where bias is small.

When `coef` (or `coef_var`) is a finite filter, the last points of the confidence interval are computed using the corresponding asymmetric filters

References

Loader, Clive. 1999. Local regression and likelihood. New York: Springer-Verlag.

Examples

```
x <- retailsa$DrinkingPlaces
coef <- lp_filter(6)
confint <- confint_filter(x, coef)
plot(confint, plot.type = "single",
     col = c("red", "black", "black"),
     lty = c(1, 2, 2))
```

deprecated-rjd3filters *Deprecated function*

Description

Deprecated function

Usage

```
cross_validation(x, coef, ...)
```

Arguments

x	input time series.
coef	vector of coefficients or a moving-average (`moving_average()`).
...	other arguments passed to the function `moving_average()` to convert `coef` to a `"moving_average"` object.

dfa_filter *Direct Filter Approach*

Description

Direct Filter Approach

Usage

```
dfa_filter(
  horizon = 6,
  degree = 0,
  density = c("uniform", "rw"),
  targetfilter = lp_filter(horizon = horizon)[, 1],
  passband = 2 * pi/12,
  accuracy.weight = 1/3,
  smoothness.weight = 1/3,
  timeliness.weight = 1/3
)
```

Arguments

`horizon`	horizon (bandwidth) of the symmetric filter.
`degree`	degree of polynomial.
`density`	hypothesis on the spectral density: `"uniform"` (= white woise, the default) or `"rw"` (= random walk).
`targetfilter`	the weights of the symmetric target filters (by default the Henderson filter).
`passband`	passband threshold.
`accuracy.weight`, `smoothness.weight`, `timeliness.weight`	the weight used for the optimisation. The weight associated to the residual is derived so that the sum of the four weights are equal to 1.

Details

Moving average computed by a minimisation of a weighted mean of three criteria under polynomials constraints. The criteria come from the decomposition of the mean squared error between th trend-cycle

Let $\boldsymbol{\theta} = (\theta_{-p}, \dots, \theta_f)'$ be a moving average where p and f are two integers defined by the parameter `lags` and `leads`. The three criteria are:

Examples

```
dfa_filter(horizon = 6, degree = 0)
dfa_filter(horizon = 6, degree = 2)
```

`diagnostics-fit` *Diagnostics and goodness of fit of filtered series*

Description

Set of functions to compute diagnostics and goodness of fit of filtered series: cross validation (`cv()`) and cross validate estimate (`cve()`), leave-one-out cross validation estimate (`loocve`), CP statistic (`cp()`) and Rice's T statistics (`rt()`).

Usage

```
cve(x, coef, ...)

cv(x, coef, ...)

loocve(x, coef, ...)

rt(x, coef, ...)

cp(x, coef, var, ...)
```

Arguments

`x`	input time series.
`coef`	vector of coefficients or a moving-average (`moving_average()`).
`...`	other arguments passed to the function `moving_average()` to convert `coef` to a `"moving_average"` object.
`var`	variance used to compute the CP statistic (`cp()`).

Details

Let $(\theta_i)_{-p\leq i\leq q}$ be a moving average of length $p+q+1$ used to filter a time series $(y_i)_{1\leq i\leq n}$. Let denote $\hat{\mu}_t$ the filtered series computed at time t as:

$$\hat{\mu}_t = \sum_{i=-p}^{q} \theta_i y_{t+i}.$$

The cross validation estimate (`cve()`) is defined as the time series $Y_t - \hat{\mu}_{-t}$ where $\hat{\mu}_{-t}$ is the leave-one-out cross validation estimate (`loocve()`) defined as the filtered series computed deleting the observation t and remaining all the other points. The cross validation statistics (`cv()`) is defined as:

$$CV = \frac{1}{n-(p+q)} \sum_{t=p+1}^{n-q} \left(y_t - \hat{\mu}_{-t}\right)^2.$$

In the case of filtering with a moving average, we can show that:

$$\hat{\mu}_{-t} = \frac{\hat{\mu}_t - \theta_0 y_t}{1-\theta_0}$$

and

$$CV = \frac{1}{n-(p+q)} \sum_{t=p+1}^{n-q} \left(\frac{y_t - \hat{\mu}_t}{1-\theta_0}\right)^2.$$

In the case of filtering with a moving average, the CP estimate of risk (introduced by Mallows (1973); `cp()`) can be defined as:

$$CP = \frac{1}{\sigma^2} \sum_{t=p+1}^{n-q} \left(y_t - \hat{\mu}_t\right)^2 - (n-(p+q))(1-2\theta_0).$$

The CP method requires an estimate of σ^2 (`var` parameter). The usual use of CP is to compare several different fits (for example different bandwidths): one should use the same estimate of $\hat{\sigma}^2$ for all fits (using for example `var_estimator()`). The recommendation of Cleveland and Devlin (1988) is to compute $\hat{\sigma}^2$ from a fit at the smallest bandwidth under consideration, at which one should be willing to assume that bias is negligible.

The Rice's T statistic (`rt()`) is defined as:

$$\frac{1}{n-(p+q)} \sum_{t=p+1}^{n-q} \frac{\left(y_t - \hat{\mu}_t\right)^2}{1-2\theta_0}$$

References

Loader, Clive. 1999. Local regression and likelihood. New York: Springer-Verlag.

Mallows, C. L. (1973). Some comments on Cp. Technometrics 15, 661– 675.

Cleveland, W. S. and S. J. Devlin (1988). Locally weighted regression: An approach to regression analysis by local fitting. Journal of the American Statistical Association 83, 596–610.

`diagnostic_matrix` *Compute quality criteria for asymmetric filters*

Description

Function du compute a diagnostic matrix of quality criteria for asymmetric filters

Usage

```
diagnostic_matrix(x, lags, passband = pi/6, sweights, ...)
```

Arguments

`x`	Weights of the asymmetric filter (from -lags to m).
`lags`	Lags of the filter (should be positive).
`passband`	passband threshold.
`sweights`	Weights of the symmetric filter (from 0 to lags or -lags to lags). If missing, the criteria from the functions `mse` are not computed.
`...`	optional arguments to `mse`.

Details

For a moving average of coefficients $\theta = (\theta_i)_{-p\leq i\leq q}$ `diagnostic_matrix` returns a `list` with the following ten criteria:

- `b_c` Constant bias (if $b_c = 0$, θ preserve constant trends)

$$\sum_{i=-p}^{q} \theta_i - 1$$

- `b_l` Linear bias (if $b_c = b_l = 0$, θ preserve constant trends)

$$\sum_{i=-p}^{q} i\theta_i$$

- `b_q` Quadratic bias (if $b_c = b_l = b_q = 0$, θ preserve quadratic trends)

$$\sum_{i=-p}^{q} i^2\theta_i$$

- `F_g` Fidelity criterium of Grun-Rehomme et al (2018)

- `S_g` Smoothness criterium of Grun-Rehomme et al (2018)
- `T_g` Timeliness criterium of Grun-Rehomme et al (2018)
- `A_w` Accuracy criterium of Wildi and McElroy (2019)
- `S_w` Smoothness criterium of Wildi and McElroy (2019)
- `T_w` Timeliness criterium of Wildi and McElroy (2019)
- `R_w` Residual criterium of Wildi and McElroy (2019)

References

Grun-Rehomme, Michel, Fabien Guggemos, and Dominique Ladiray (2018). "Asymmetric Moving Averages Minimizing Phase Shift". In: Handbook on Seasonal Adjustment.

Wildi, Marc and McElroy, Tucker (2019). "The trilemma between accuracy, timeliness and smoothness in real-time signal extraction". In: International Journal of Forecasting 35.3, pp. 1072–1084.

`filter` *Linear Filtering on a Time Series*

Description

Applies linear filtering to a univariate time series or to each series separately of a multivariate time series using either a moving average (symmetric or asymmetric) or a combination of symmetric moving average at the center and asymmetric moving averages at the bounds.

Usage

```
filter(x, coefs, remove_missing = TRUE)
```

Arguments

`x`	a univariate or multivariate time series.
`coefs`	a `matrix` or a `list` that contains all the coefficients of the asymmetric and symmetric filters. (from the symmetric filter to the shortest). See details.
`remove_missing`	if `TRUE` (default) leading and trailing NA are removed before filtering.

Details

The functions `filter` extends `filter` allowing to apply every kind of moving averages (symmetric and asymmetric filters) or to apply aset multiple moving averages to deal with the boundaries.

Let x_t be the input time series to filter.

- If coef is an object `moving_average()`, of length q, the result y is equal at time t to:

  $$y[t] = x[t - lags] * coef[1] + x[t - lags + 1] * coef[1] + ... + x[t - lags + q] * coef[q]$$

  . It extends the function `filter` that would add `NA` at the end of the time series.
- If coef is a `matrix`, `list` or `finite_filters()` object, at the center, the symmetric moving average is used (first column/element of `coefs`). At the boundaries, the last moving average of `coefs` is used to compute the filtered time series $y[n]$ (no future point known), the second to last to compute the filtered time series $y[n-1]$ (one future point known)...

Examples

```
x <- retailsa$DrinkingPlaces

lags <- 6
leads <- 2
fst_coef <- fst_filter(lags = lags, leads = leads, smoothness.weight = 0.3, timeliness.weight = 0.3)
lpp_coef <- lp_filter(horizon = lags, kernel = "Henderson", endpoints = "LC")

fst_ma <- filter(x, fst_coef)
lpp_ma <- filter(x, lpp_coef[,"q=2"])

plot(ts.union(x, fst_ma, lpp_ma), plot.type = "single", col = c("black","red","blue"))

trend <- filter(x, lpp_coef)
# This is equivalent to:
trend <- localpolynomials(x, horizon = 6)
```

`filters_operations` *Operations on Filters*

Description

Manipulation of `moving_average()` or `finite_filters()` objects

Usage

```
## S3 method for class 'moving_average'
sum(..., na.rm = FALSE)

## S4 method for signature 'moving_average,numeric'
x[i]

## S4 method for signature 'moving_average,logical'
x[i]

## S4 replacement method for signature 'moving_average,ANY,missing,numeric'
x[i] <- value
```

```
## S3 method for class 'moving_average'
cbind(..., zero_as_na = FALSE)

## S3 method for class 'moving_average'
rbind(...)

## S4 method for signature 'moving_average,moving_average'
e1 + e2

## S4 method for signature 'moving_average,numeric'
e1 + e2

## S4 method for signature 'numeric,moving_average'
e1 + e2

## S4 method for signature 'moving_average,missing'
e1 + e2

## S4 method for signature 'moving_average,missing'
e1 - e2

## S4 method for signature 'moving_average,moving_average'
e1 - e2

## S4 method for signature 'moving_average,numeric'
e1 - e2

## S4 method for signature 'numeric,moving_average'
e1 - e2

## S4 method for signature 'moving_average,moving_average'
e1 * e2

## S4 method for signature 'moving_average,numeric'
e1 * e2

## S4 method for signature 'numeric,moving_average'
e1 * e2

## S4 method for signature 'ANY,moving_average'
e1 * e2

## S4 method for signature 'moving_average,ANY'
e1 * e2

## S4 method for signature 'moving_average,numeric'
e1 / e2
```

```
## S4 method for signature 'moving_average,numeric'
e1 ^ e2

## S4 method for signature 'finite_filters,moving_average'
e1 * e2

## S4 method for signature 'moving_average,finite_filters'
e1 * e2

## S4 method for signature 'finite_filters,numeric'
e1 * e2

## S4 method for signature 'ANY,finite_filters'
e1 * e2

## S4 method for signature 'finite_filters,ANY'
e1 * e2

## S4 method for signature 'numeric,finite_filters'
e1 + e2

## S4 method for signature 'finite_filters,moving_average'
e1 + e2

## S4 method for signature 'moving_average,finite_filters'
e1 + e2

## S4 method for signature 'finite_filters,missing'
e1 + e2

## S4 method for signature 'finite_filters,missing'
e1 - e2

## S4 method for signature 'finite_filters,moving_average'
e1 - e2

## S4 method for signature 'moving_average,finite_filters'
e1 - e2

## S4 method for signature 'finite_filters,numeric'
e1 - e2

## S4 method for signature 'numeric,finite_filters'
e1 - e2

## S4 method for signature 'finite_filters,numeric'
e1 / e2
```

```
## S4 method for signature 'finite_filters,numeric'
e1 ^ e2

## S4 method for signature 'finite_filters,finite_filters'
e1 * e2

## S4 method for signature 'finite_filters,finite_filters'
e1 + e2

## S4 method for signature 'finite_filters,finite_filters'
e1 - e2

## S4 method for signature 'finite_filters,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'finite_filters,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

..., drop, na.rm	other parameters.
x, e1, e2	object
i, j, value	indices specifying elements to extract or replace and the new value
zero_as_na	boolean indicating if, when merging several moving averages (cbind) if trealing and leading zeros added to have a matrix form should be replaced by NA.

---

finite_filters *Manipulating Finite Filters*

Description

Manipulating Finite Filters

Usage

```
finite_filters(
  sfilter,
  rfilters = NULL,
  lfilters = NULL,
  first_to_last = FALSE
)

is.finite_filters(x)

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

Arguments

sfilter	the symmetric filter (moving_average() object) or a matrix or list with all the coefficients.
rfilters	the right filters (used on the last points).
lfilters	the left filters (used on the first points).
first_to_last	boolean indicating if the first element of rfilters is the first asymmetric filter (when only one observation is missing) or the last one (real-time estimates).
x	object to test the class.
object	finite_filters object.

Examples

```
ff_lp <- lp_filter()
ff_simple_ma <- finite_filters(moving_average(c(1, 1, 1), lags = -1)/3,
               rfilters = list(moving_average(c(1, 1), lags = -1)/2))
ff_lp
ff_simple_ma
ff_lp * ff_simple_ma
```

---

fst — *FST criteria*

Description

Compute the Fidelity, Smoothness and Timeliness (FST) criteria

Usage

```
fst(weights, lags, passband = pi/6, ...)
```

Arguments

weights	either a "moving_average" or a numeric vector containing weights.
lags	Lags of the moving average (when weights is not a "moving_average").
passband	Passband threshold for timeliness criterion.
...	other unused arguments.

Value

The values of the 3 criteria, the gain and phase of the associated filter.

References

Grun-Rehomme, Michel, Fabien Guggemos, and Dominique Ladiray (2018). “Asymmetric Moving Averages Minimizing Phase Shift”. In: Handbook on Seasonal Adjustment, https://ec.europa.eu/eurostat/web/products-manuals-and-guidelines/-/ks-gq-18-001.

Examples

```
filter <- lp_filter(horizon = 6, kernel = "Henderson", endpoints = "LC")
fst(filter[, "q=0"])
# To compute the statistics on all filters:
fst(filter)
```

`fst_filter` *Estimation of a filter using the Fidelity-Smoothness-Timeliness criteria*

Description

Estimation of a filter using the Fidelity-Smoothness-Timeliness criteria

Usage

```
fst_filter(
  lags = 6,
  leads = 0,
  pdegree = 2,
  smoothness.weight = 1,
  smoothness.degree = 3,
  timeliness.weight = 0,
  timeliness.passband = pi/6,
  timeliness.antiphase = TRUE
)
```

Arguments

`lags` Lags of the filter (should be positive).

`leads` Leads of the filter (should be positive or 0).

`pdegree` Local polynomials preservation: max degree.

`smoothness.weight` Weight for the smoothness criterion (in $[0, 1]$).

`smoothness.degree` Degree of the smoothness criterion (3 for Henderson).

`timeliness.weight` Weight for the Timeliness criterion (in $[0, 1[$). `sweight+tweight` should be in $[0, 1]$.

`timeliness.passband` Passband for the timeliness criterion (in radians). The phase effect is computed in $[0, passband]$.

`timeliness.antiphase` boolean indicating if the timeliness should be computed analytically (`TRUE`) or numerically (`FALSE`).

Details

Moving average computed by a minimisation of a weighted mean of three criteria under polynomials constraints. Let $\boldsymbol{\theta} = (\theta_{-p}, \dots, \theta_f)'$ be a moving average where p and f are two integers defined by the parameter `lags` and `leads`. The three criteria are:

- *Fidelity*, F_g: it's the variance reduction ratio.

$$F_g(\boldsymbol{\theta}) = \sum_{k=-p}^{+f} \theta_k^2$$

- *Smoothness*, S_g: it measures the flexibility of the coefficient curve of a filter and the smoothness of the trend.

$$S_g(\boldsymbol{\theta}) = \sum_j (\nabla^q \theta_j)^2$$

  The integer q is defined by parameter `smoothness.degree`. By default, the Henderson criteria is used (`smoothness.degree = 3`).

- *Timeliness*, T_g :

$$T_g(\boldsymbol{\theta}) = \int_0^{\omega_2} f(\rho_{\boldsymbol{\theta}}(\omega), \varphi_{\boldsymbol{\theta}}(\omega)) d\omega$$

  with $\rho_{\boldsymbol{\theta}}$ and $\varphi_{\boldsymbol{\theta}}$ the gain and phase shift functions of $\boldsymbol{\theta}$, and f a penalty function defined as $f\colon (\rho, \varphi) \mapsto \rho^2 \sin(\varphi)^2$ to have an analytically solvable criterium. ω_2 is defined by the parameter `timeliness.passband` and is it by default equal to $2\pi/12$: for monthly time series, we focus on the timeliness associated to cycles of 12 months or more.

The moving average is then computed solving the problem:

$$\begin{cases} \min\limits_{\theta} & J(\theta) = (1 - \beta - \gamma) F_g(\theta) + \beta S_g(\theta) + \gamma T_g(\theta) \\ s.t. & C\theta = a \end{cases}$$

Where $C\theta = a$ represents linear constraints to have a moving average that preserve polynomials of degree q (`pdegree`):

$$C = \begin{pmatrix} 1 & \cdots & 1 \\ -h & \cdots & h \\ \vdots & \cdots & \vdots \\ (-h)^d & \cdots & h^d \end{pmatrix}, \quad a = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

References

Grun-Rehomme, Michel, Fabien Guggemos, and Dominique Ladiray (2018). "Asymmetric Moving Averages Minimizing Phase Shift". In: Handbook on Seasonal Adjustment, https://ec.europa.eu/eurostat/web/products-manuals-and-guidelines/-/ks-gq-18-001.

Examples

```
filter <- fst_filter(lags = 6, leads = 0)
filter
```

get_kernel — *Get the coefficients of a kernel*

Description

Function to get the coefficient associated to a kernel. Those coefficients are then used to compute the different filters.

Usage

```
get_kernel(
  kernel = c("Henderson", "Uniform", "Triangular", "Epanechnikov", "Parabolic",
    "BiWeight", "TriWeight", "Tricube", "Trapezoidal", "Gaussian"),
  horizon,
  sd_gauss = 0.25
)
```

Arguments

`kernel`	kernel uses.
`horizon`	horizon (bandwidth) of the symmetric filter.
`sd_gauss`	standard deviation for gaussian kernel. By default 0.25.

Value

`tskernel` object (see kernel).

Examples

```
get_kernel("Henderson", horizon = 3)
```

get_moving_average — *Get Moving Averages from ARIMA model*

Description

Get Moving Averages from ARIMA model

Usage

```
get_moving_average(x, ...)
```

Arguments

`x`	the object.
`...`	unused parameters

Examples

```
fit <- stats::arima(log10(AirPassengers), c(0, 1, 1),
seasonal = list(order = c(0, 1, 1), period = 12))
get_moving_average(fit)
```

---

get_properties_function *Get properties of filters*

Description

Get properties of filters

Usage

```
get_properties_function(
  x,
  component = c("Symmetric Gain", "Symmetric Phase", "Symmetric transfer",
    "Asymmetric Gain", "Asymmetric Phase", "Asymmetric transfer"),
  ...
)
```

Arguments

x	a "moving_average" or "finite_filters" object.
component	the component to extract.
...	unused other arguments.

Examples

```
filter <- lp_filter(3, kernel = "Henderson")
sgain <- get_properties_function(filter, "Symmetric Gain")
plot(sgain, xlim= c(0, pi/12))
```

---

implicit_forecast *Retrieve implicit forecasts corresponding to the asymmetric filters*

Description

Function to retrieve the implicit forecasts corresponding to the asymmetric filters

Usage

```
implicit_forecast(x, coefs)
```

Arguments

`x`	a univariate or multivariate time series.
`coefs`	a `matrix` or a `list` that contains all the coefficients of the asymmetric and symmetric filters. (from the symmetric filter to the shortest). See details.

Details

Let h be the bandwidth of the symmetric filter, $v_{-h}, \dots, v_h$ the coefficients of the symmetric filter and $w^q_{-h}, \dots, w^q_h$ the coefficients of the asymmetric filter used to estimate the trend when q future values are known (with the convention $w^q_{q+1} = \dots = w^q_h = 0$). Let denote $y_{-h}, \dots, y_0$ the las h available values of the input times series. Let also note $y_{-h}, \dots, y_0$ the observed series studied and $y^*_1, \dots y^*_h$the implicit forecast induced by $w^0, \dots w^{h-1}$. This means that:

$$\forall q, \quad \sum_{i=-h}^{0} v_i y_i + \sum_{i=1}^{h} v_i y^*_i = \sum_{i=-h}^{0} w^q_i y_i + \sum_{i=1}^{h} w^q_i y^*_i$$

which is equivalent to

$$\forall q, \sum_{i=1}^{h} (v_i - w^q_i) y^*_i = \sum_{i=-h}^{0} (w^q_i - v_i) y_i.$$

Note that this is solved numerically: the solution isn't exact.

Examples

```
x <- retailsa$AllOtherGenMerchandiseStores
ql <- lp_filter(horizon = 6, kernel = "Henderson", endpoints = "QL")
lc <- lp_filter(horizon = 6, kernel = "Henderson", endpoints = "LC")
f_ql <- implicit_forecast(x, ql)
f_lc <- implicit_forecast(x, lc)

plot(window(x, start = 2007),
     xlim = c(2007,2012))
lines(ts(c(tail(x,1), f_ql), frequency = frequency(x), start = end(x)),
      col = "red", lty = 2)
lines(ts(c(tail(x,1), f_lc), frequency = frequency(x), start = end(x)),
      col = "blue", lty = 2)
```

`impute_last_obs` *Impute Incomplete Finite Filters*

Description

Impute Incomplete Finite Filters

Usage

```
impute_last_obs(x, n, nperiod = 1, backward = TRUE, forward = TRUE)
```

Arguments

`x`	a `finite_filters()` object.
`n`	integer specifying the number of imputed periods. By default all the missing moving averages are imputed.
`nperiod`	integer specifying how to imput missing date. `nperiod = 1` means imputation using last filtered data (1 period backward), `nperiod = 12` with monthly data means imputation using last year filtered data, etc.
`backward, forward`	boolean indicating if the imputation should be done backward (on left filters), forward (on right filters).

Details

When combining finite filters and a moving average, the first and/or the last points cannot be computed.

For example, using the M2X12 moving average, that is to say the symmetric moving average with coefficients

$$\theta = \frac{1}{24}B^6 + \frac{1}{12}B^5 + \cdots + \frac{1}{12}B^{-5} + \frac{1}{24}B^{-6},$$

the first and last 6 points cannot be computed.

`impute_last_obs()` allows to impute the first/last points using the `nperiod` previous filtered data. With `nperiod = 1`, the last filtered data is used for the imputation, with `nperiod = 12` and monthly data, the last year filtered data is used for the imputation, etc.

Examples

```
y <- window(retailsa$AllOtherGenMerchandiseStores, start = 2008)
M3 <- moving_average(rep(1/3, 3), lags = -1)
M3X3 <- M3 * M3
M2X12 <- (simple_ma(12, -6) + simple_ma(12, -5)) / 2
composite_ma <- M3X3 * M2X12
# The last 6 points cannot be computed
composite_ma
composite_ma * y
# they can be computed using the last filtered data
# e.g. to impute the first 3 missing months with last period:
impute_last_obs(composite_ma, n = 3, nperiod = 1) * y
# or using the filtered data of the same month in previous year
impute_last_obs(composite_ma, n = 6, nperiod = 12) * y
```

`localpolynomials` *Apply Local Polynomials Filters*

Description

Apply Local Polynomials Filters

Usage

```
localpolynomials(
  x,
  horizon = 6,
  degree = 3,
 kernel = c("Henderson", "Uniform", "Biweight", "Trapezoidal", "Triweight", "Tricube",
    "Gaussian", "Triangular", "Parabolic"),
  endpoints = c("LC", "QL", "CQ", "CC", "DAF"),
  ic = 4.5,
  tweight = 0,
  passband = pi/12
)
```

Arguments

`x`	input time-series.
`horizon`	horizon (bandwidth) of the symmetric filter.
`degree`	degree of polynomial.
`kernel`	kernel uses.
`endpoints`	methode for endpoints.
`ic`	ic ratio.
`tweight`	timeliness weight.
`passband`	passband threshold.

Value

the target signal

References

Proietti, Tommaso and Alessandra Luati (2008). “Real time estimation in local polynomial regression, with application to trend-cycle analysis”.

See Also

`lp_filter()`.

Examples

```
x <- retailsa$AllOtherGenMerchandiseStores
trend <- localpolynomials(x, horizon = 6)
plot(x)
lines(trend, col = "red")
```

`lp_filter` *Local Polynomials Filters*

Description

Local Polynomials Filters

Usage

```
lp_filter(
  horizon = 6,
  degree = 3,
 kernel = c("Henderson", "Uniform", "Biweight", "Trapezoidal", "Triweight", "Tricube",
    "Gaussian", "Triangular", "Parabolic"),
  endpoints = c("LC", "QL", "CQ", "CC", "DAF", "CN"),
  ic = 4.5,
  tweight = 0,
  passband = pi/12
)
```

Arguments

`horizon`	horizon (bandwidth) of the symmetric filter.
`degree`	degree of polynomial.
`kernel`	kernel uses.
`endpoints`	methode for endpoints.
`ic`	ic ratio.
`tweight`	timeliness weight.
`passband`	passband threshold.

Details

- "LC": Linear-Constant filter
- "QL": Quadratic-Linear filter
- "CQ": Cubic-Quadratic filter
- "CC": Constant-Constant filter
- "DAF": Direct Asymmetric filter
- "CN": Cut and Normalized Filter

Value

a `finite_filters()` object.

References

Proietti, Tommaso and Alessandra Luati (2008). "Real time estimation in local polynomial regression, with application to trend-cycle analysis".

See Also

localpolynomials().

Examples

```
henderson_f <- lp_filter(horizon = 6, kernel = "Henderson")
plot_coef(henderson_f)
```

---

moving_average *Manipulation of moving averages*

Description

Manipulation of moving averages

Usage

```
moving_average(
  x,
  lags = -length(x),
  trailing_zero = FALSE,
  leading_zero = FALSE
)

is.moving_average(x)

is_symmetric(x)

upper_bound(x)

lower_bound(x)

mirror(x)

## S3 method for class 'moving_average'
rev(x)

## S3 method for class 'moving_average'
length(x)

to_seasonal(x, s)
```

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

Arguments

`x`	vector of coefficients.
`lags`	integer indicating the number of lags of the moving average.
`trailing_zero`, `leading_zero`	boolean indicating wheter to remove leading/trailing zero and NA.
`s`	seasonal period for the `to_seasonal()` function.
`object`	`moving_average` object.

Details

A moving average is defined by a set of coefficient $\boldsymbol{\theta} = (\theta_{-p}, \ldots, \theta_f)'$ such all time series X_t are transformed as:

$$M_{\boldsymbol{\theta}}(X_t) = \sum_{k=-p}^{+f} \theta_k X_{t+k} = \left(\sum_{k=-p}^{+f} \theta_k B^{-k}\right) X_t$$

The integer p is defined by the parameter `lags`.

The function `to_seasonal()` transforms the moving average $\boldsymbol{\theta}$ to:

$$M_{\boldsymbol{\theta}'}(X_t) = \sum_{k=-p}^{+f} \theta_k X_{t+ks} = \left(\sum_{k=-p}^{+f} \theta_k B^{-ks}\right) X_t$$

Examples

```
y <- retailsa$AllOtherGenMerchandiseStores
e1 <- moving_average(rep(1,12), lags = -6)
e1 <- e1/sum(e1)
e2 <- moving_average(rep(1/12, 12), lags = -5)
M2X12 <- (e1 + e2)/2
coef(M2X12)
M3 <- moving_average(rep(1/3, 3), lags = -1)
M3X3 <- M3 * M3
# M3X3 moving average applied to each month
M3X3
M3X3_seasonal <- to_seasonal(M3X3, 12)
# M3X3_seasonal moving average applied to the global series
M3X3_seasonal

def.par <- par(no.readonly = TRUE)
par(mai = c(0.5, 0.8, 0.3, 0))
layout(matrix(c(1,2), nrow = 1))
plot_gain(M3X3, main = "M3X3 applied to each month")
plot_gain(M3X3_seasonal, main = "M3X3 applied to the global series")
par(def.par)

# To apply the moving average
```

```
t <- y * M2X12
# Or use the filter() function:
t <- filter(y, M2X12)
si <- y - t
s <- si * M3X3_seasonal
# or equivalently:
s_mm <- M3X3_seasonal * (1 - M2X12)
s <- y * s_mm
plot(s)
```

---

mse *Accuracy/smoothness/timeliness criteria through spectral decomposition*

Description

Accuracy/smoothness/timeliness criteria through spectral decomposition

Usage

```
mse(aweights, sweights, density = c("uniform", "rw"), passband = pi/6, ...)
```

Arguments

aweights	moving_average object or weights of the asymmetric filter (from -n to m).
sweights	moving_average object or weights of the symmetric filter (from 0 to n or -n to n).
density	hypothesis on the spectral density: "uniform" (= white woise, the default) or "rw" (= random walk).
passband	passband threshold.
...	other unused arguments.

Value

The criteria

References

Wildi, Marc and McElroy, Tucker (2019). "The trilemma between accuracy, timeliness and smoothness in real-time signal extraction". In: International Journal of Forecasting 35.3, pp. 1072–1084.

Examples

```
filter <- lp_filter(horizon = 6, kernel = "Henderson", endpoints = "LC")
sweights <- filter[, "q=6"]
aweights <- filter[, "q=0"]
mse(aweights, sweights)
# Or to compute directly the criteria on all asymmetric filters:
mse(filter)
```

plot_filters *Plots filters properties*

Description

Functions to plot the coefficients, the gain and the phase functions.

Usage

```
plot_coef(x, nxlab = 7, add = FALSE, ...)

## Default S3 method:
plot_coef(
  x,
  nxlab = 7,
  add = FALSE,
  zero_as_na = TRUE,
  q = 0,
  legend = FALSE,
  legend.pos = "topright",
  ...
)

## S3 method for class 'moving_average'
plot_coef(x, nxlab = 7, add = FALSE, ...)

## S3 method for class 'finite_filters'
plot_coef(
  x,
  nxlab = 7,
  add = FALSE,
  zero_as_na = TRUE,
  q = 0,
  legend = length(q) > 1,
  legend.pos = "topright",
  ...
)

plot_gain(x, nxlab = 7, add = FALSE, xlim = c(0, pi), ...)

## S3 method for class 'moving_average'
plot_gain(x, nxlab = 7, add = FALSE, xlim = c(0, pi), ...)

## S3 method for class 'finite_filters'
plot_gain(
  x,
  nxlab = 7,
```

```
  add = FALSE,
  xlim = c(0, pi),
  q = 0,
  legend = length(q) > 1,
  legend.pos = "topright",
  n = 101,
  ...
)

plot_phase(x, nxlab = 7, add = FALSE, xlim = c(0, pi), normalized = FALSE, ...)

## S3 method for class 'moving_average'
plot_phase(x, nxlab = 7, add = FALSE, xlim = c(0, pi), normalized = FALSE, ...)

## S3 method for class 'finite_filters'
plot_phase(
  x,
  nxlab = 7,
  add = FALSE,
  xlim = c(0, pi),
  normalized = FALSE,
  q = 0,
  legend = length(q) > 1,
  legend.pos = "topright",
  n = 101,
  ...
)
```

Arguments

`x`	coefficients, gain or phase.
`nxlab`	number of xlab.
`add`	boolean indicating if the new plot is added to the previous one.
`...`	other arguments to `matplot`.
`zero_as_na`	boolean indicating if the trailing zero of the coefficients should be plotted (`FALSE`) or removed (`TRUE`).
`q`	q.
`legend`	boolean indicating if the legend is printed.
`legend.pos`	position of the legend.
`xlim`	vector containing x limits.
`n`	number of points used to plot the functions.
`normalized`	boolean indicatif if the phase function is normalized by the frequency.

Examples

```
filter <- lp_filter(6, endpoints = "DAF", kernel = "Henderson")
```

```
plot_coef(filter, q = c(0,3), legend = TRUE)
plot_gain(filter, q = c(0,3), legend = TRUE)
plot_phase(filter, q = c(0,3), legend = TRUE)
```

`retailsa` *Seasonally Adjusted Retail Sales*

Description

A dataset containing monthly seasonally adjusted retailed sales

Usage

```
retailsa
```

Format

A `list` of `ts` objects from january 1992 to december 2010.

`rkhs_filter` *Reproducing Kernel Hilbert Space (RKHS) Filters*

Description

Estimation of a filter using Reproducing Kernel Hilbert Space (RKHS)

Usage

```
rkhs_filter(
  horizon = 6,
  degree = 2,
  kernel = c("BiWeight", "Henderson", "Epanechnikov", "Triangular", "Uniform",
    "TriWeight"),
 asymmetricCriterion = c("Timeliness", "FrequencyResponse", "Accuracy", "Smoothness",
    "Undefined"),
  density = c("uniform", "rw"),
  passband = 2 * pi/12,
  optimalbw = TRUE,
  optimal.minBandwidth = horizon,
  optimal.maxBandwidth = 3 * horizon,
  bandwidth = horizon + 1
)
```

Arguments

- `horizon` horizon (bandwidth) of the symmetric filter.
- `degree` degree of polynomial.
- `kernel` kernel uses.
- `asymmetricCriterion` the criteria used to compute the optimal bandwidth. If "Undefined", $m + 1$ is used.
- `density` hypothesis on the spectral density: "uniform" (= white woise, the default) or "rw" (= random walk).
- `passband` passband threshold.
- `optimalbw` boolean indicating if the bandwith should be choosen by optimisation (between `optimal.minBandwidth` and `optimal.minBandwidth` using the criteria `asymmetricCriterion`). If `optimalbw` = `FALSE` then the bandwith specified in `bandwidth` will be used.
- `optimal.minBandwidth`, `optimal.maxBandwidth` the range used for the optimal bandwith selection.
- `bandwidth` the bandwidth to use if `optimalbw` = `FALSE`.

Value

a `finite_filters()` object.

References

Dagum, Estela Bee and Silvia Bianconcini (2008). "The Henderson Smoother in Reproducing Kernel Hilbert Space". In: Journal of Business & Economic Statistics 26, pp. 536–545. URL: https://ideas.repec.org/a/bes/jnlbes/v26y2008p536-545.html.

Examples

```
rkhs <- rkhs_filter(horizon = 6, asymmetricCriterion = "Timeliness")
plot_coef(rkhs)
```

---

rkhs_kernel *Get RKHS kernel function*

Description

Get RKHS kernel function

Usage

```
rkhs_kernel(
  kernel = c("Biweight", "Henderson", "Epanechnikov", "Triangular", "Uniform",
    "Triweight"),
  degree = 2,
  horizon = 6
)
```

Arguments

`kernel`	kernel uses.
`degree`	degree of polynomial.
`horizon`	horizon (bandwidth) of the symmetric filter.

`rkhs_optimal_bw` *Optimal Bandwith of Reproducing Kernel Hilbert Space (RKHS) Filters*

Description

Function to export the optimal bandwidths used in Reproducing Kernel Hilbert Space (RKHS) filters

Usage

```
rkhs_optimal_bw(
  horizon = 6,
  degree = 2,
  kernel = c("Biweight", "Henderson", "Epanechnikov", "Triangular", "Uniform",
    "Triweight"),
 asymmetricCriterion = c("Timeliness", "FrequencyResponse", "Accuracy", "Smoothness"),
  density = c("uniform", "rw"),
  passband = 2 * pi/12,
  optimal.minBandwidth = horizon,
  optimal.maxBandwidth = 3 * horizon
)
```

Arguments

`horizon` horizon (bandwidth) of the symmetric filter.

`degree` degree of polynomial.

`kernel` kernel uses.

`asymmetricCriterion`
the criteria used to compute the optimal bandwidth. If `"Undefined"`, $m + 1$ is used.

`density` hypothesis on the spectral density: `"uniform"` (= white woise, the default) or `"rw"` (= random walk).

`passband` passband threshold.

`optimal.minBandwidth`, `optimal.maxBandwidth`
the range used for the optimal bandwith selection.

Examples

```
rkhs_optimal_bw(asymmetricCriterion = "Timeliness")
rkhs_optimal_bw(asymmetricCriterion = "Timeliness", optimal.minBandwidth = 6.2)
```

`rkhs_optimization_fun` *Optimization Function of Reproducing Kernel Hilbert Space (RKHS) Filters*

Description

Export function used to compute the optimal bandwidth of Reproducing Kernel Hilbert Space (RKHS) filters

Usage

```
rkhs_optimization_fun(
  horizon = 6,
  leads = 0,
  degree = 2,
  kernel = c("Biweight", "Henderson", "Epanechnikov", "Triangular", "Uniform",
    "Triweight"),
 asymmetricCriterion = c("Timeliness", "FrequencyResponse", "Accuracy", "Smoothness"),
  density = c("uniform", "rw"),
  passband = 2 * pi/12
)
```

Arguments

`horizon`	horizon (bandwidth) of the symmetric filter.
`leads`	Leads of the filter (should be positive or 0).
`degree`	degree of polynomial.
`kernel`	kernel uses.
`asymmetricCriterion`	the criteria used to compute the optimal bandwidth. If `"Undefined"`, $m+1$ is used.
`density`	hypothesis on the spectral density: `"uniform"` (= white woise, the default) or `"rw"` (= random walk).
`passband`	passband threshold.

Examples

```
plot(rkhs_optimization_fun(horizon = 6, leads = 0,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X0 filter")
plot(rkhs_optimization_fun(horizon = 6, leads = 1,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X1 filter")
plot(rkhs_optimization_fun(horizon = 6, leads = 2,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X2 filter")
```

```
plot(rkhs_optimization_fun(horizon = 6, leads = 3,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X3 filter")
plot(rkhs_optimization_fun(horizon = 6, leads = 4,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X4 filter")
plot(rkhs_optimization_fun(horizon = 6, leads = 5,degree = 3, asymmetricCriterion = "Timeliness"),
     5.5, 6*3, ylab = "Timeliness",
     main = "6X5 filter")
```

---

simple_ma *Simple Moving Average*

Description

A simple moving average is a moving average whose coefficients are all equal and whose sum is 1

Usage

```
simple_ma(order, lags = -trunc((order - 1)/2))
```

Arguments

`order`	number of terms of the moving_average
`lags`	integer indicating the number of lags of the moving average.

Examples

```
# The M2X12 moving average is computed as
(simple_ma(12, -6) + simple_ma(12, -5)) / 2
# The M3X3 moving average is computed as
simple_ma(3, -1) ^ 2
# The M3X5 moving average is computed as
simple_ma(3, -1) * simple_ma(5, -2)
```

---

var_estimator *Variance Estimator*

Description

Variance Estimator

Usage

```
var_estimator(x, coef, ...)
```

Arguments

`x`	input time series.
`coef`	vector of coefficients or a moving-average (`moving_average()`).
`...`	other arguments passed to the function `moving_average()` to convert `coef` to a `"moving_average"` object.

Details

Let $(\theta_i)_{-p\leq i\leq q}$ be a moving average of length $p+q+1$ used to filter a time series $(y_i)_{1\leq i\leq n}$. It is equivalent to a local regression and the associated error variance σ^2 can be estimated using the normalized residual sum of squares, which can be simplified as:

$$\hat{\sigma}^2 = \frac{1}{n-(p+q)}\sum_{t=p+1}^{n-q}\frac{(y_t-\hat{\mu}_t)^2}{1-2w_0^2+\sum_{i=-p}^{q}w_i^2}$$

References

Loader, Clive. 1999. Local regression and likelihood. New York: Springer-Verlag.

Index