Package ‘robfilter’

June 18, 2009

Version 2.5

Date 2009-06-16

Title Robust Time Series Filters

Author Roland Fried <fried@statistik.tu-dortmund.de> and Karen Schettlinger <schettlinger@statistik.tu-dortmund.de>.

Maintainer Roland Fried <fried@statistik.tu-dortmund.de>

Depends R (>= 2.5.0), robustbase, MASS

Description A set of functions to filter time series based on concepts from robust statistics.

License GPL (>= 2)

URL http://www.statistik.tu-dortmund.de/de/content/einrichtungen/lehrstuehle/personen/fried.html

LazyData yes

Repository CRAN

Date/Publication 2009-06-18 12:23:15

R topics documented:

`adore.filter` *A Robust Adaptive Online Repeated Median Filter for Univariate Time Series*

Description

Procedure for robust online extraction of low frequency components (the *signal*) from a univariate time series by a moving window technique with adaptive window width selection (ADaptive Online REpeated median FILTER).

Usage

```
adore.filter(y, p.test = 15, minNonNAs = 5,
                min.width = 11, max.width = 100, width.search="geometric",
                rtr=2, extrapolate=TRUE, calc.qn = FALSE)
```

Arguments

`y` a numeric vector or (univariate) time series object.

`p.test` defines the number of most recent Repeated Median residuals within each window used to test the goodness of fit of the online signal level.
It can be either a value in (0.25, 0.3, 0.5), meaning that `floor(p.test*width)` residuals are considered for the goodness of fit test, where `width` is the currently used window width, or it can also be a positive integer `>= 5` specifying a fixed number of most recent residuals (default).
If the number of residuals considered for the test exceeds `width/2`, the procedure sets it to `floor(width/2)`, if it is smaller than five, the number is set to five.

`minNonNAs` a positive integer `>= 5` defining the minimum number of non-missing observations within one window which is required for a ‘sensible’ estimation.

`min.width` a positive integer `>= 5` specifying the minimal window width.

`max.width` a positive integer `>= min.width` specifying the maximal window width.

`width.search` a character string defining the search algorithm used for finding an adequate window width at each point in time.

`"linear"` The linear search always results in the largest window width possible and hence yields the smoothest online signal. However, if sudden changes (like level shifts) appear in the signal it requires a lot of computation time and thus, an increased variability of the extracted signal may be observed.

`"binary"` The binary search is recommended if it can be expected that the window width needs to be reduced drastically from a large to a very small value at certain times (for example at level shifts or trend changes). However, it may not always result in the largest possible window width.

"geometric" (default) The geometric search is as fast as the binary search but it puts more weight on large window widths. It offers a good compromise between the linear and the binary search (computation time vs. smooth output signal).

`rtr` a value in 0, 1, 2 specifying whether a 'restrict to range' rule should be applied.

- **rtr=0** The estimated signal level consists of the last fitted value of a Repeated Median regression fit within a time window of adequate width.
- **rtr=1** The level estimation is restricted to the range of the observations within each time window.
- **rtr=2 (default)** The level estimation is restricted to the range of the most recent observations (specified by `p.test`) i.e., to the range of the observations which are used to evaluate the goodness of fit.

`extrapolate` a logical indicating whether the level estimations should be extrapolated to the beginning of the time series. The extrapolation consists of all fitted values within the first time window.

`calc.qn` a logical indicating whether the Qn scale (Rousseeuw, Croux, 1993) should also be calculated along with the signal level as an estimate of the standard deviation in each window. Here, the `Qn` command from the `robustbase` library is applied with the built-in finite sample correction.

Details

The `adore.filter` works by applying Repeated Median (RM) regression (Siegel, 1982) to a moving time window with a length varying between `min.width` and `max.width`.

For each point in time, the window width is adapted to the current data situation by a goodness of fit test for the most recent signal level estimation. The test uses the absolute value of the sum of the RM residuals in the subset specified by `p.test`. The critical value for the test decision corresponds to a slightly modified 0.95-quantile of the distribution of the test statistic and is stored in the data set `critvals`.

A more detailed description of the filter can be found in Schettlinger, Fried, Gather (2008).

Value

`adore.filter` returns an object of class `adore.filter`. An object of class `adore.filter` is a list containing the following components:

`level` a numeric vector containing the signal level extracted by the RM filter with adaptive window width.

`slope` a numeric vector containing the corresponding slope within each time window.

`width` a numeric vector containing the corresponding window width used for the level and slope estimations.

`level.list` a list which contains with as many elements as the length of the input time series. If at time `t`, the window width was not reduced, the entry `level.list[[t]]` simply corresponds to `level[t]`. However, if more than one iteration took place, `level.list[[t]]` is a vector which contains all level estimations which were evaluated until the final estimate `mu[t]` passed the goodness of fit test and was stored.

slope.list a list containing the slope estimations corresponding to the values in `level.list`.

width.list a list containing the window widths used for the estimations in `level.list` and `slope.list`.

sigma a numeric vector containing the corresponding scale within each time window estimated by the robust Qn estimator (only calculated if `calc.qn = TRUE`, else `sigma` does not exist).

In addition, the original input time series is returned as list member `y`, and the settings used for the analysis are returned as the list members `min.width`, `max.width`, `width.search`, `p.test`, `minNonNAs`, `rtr`, `extrapolate`, and `calc.qn`.

Application of the function `plot` to an object of class `aoRM` returns a plot showing the original time series with the filtered output.

Author(s)

Karen Schettlinger

References

Schettlinger, K., Fried, R., Gather, U. (2008) Real Time Signal Processing by Adaptive Repeated Median Filters, *International Journal of Adaptive Control and Signal Processing*, submitted.

Siegel, A.F. (1982) Robust Regression Using Repeated Medians, *Biometrika* **69** (1), 242-244.

Rousseeuw, P. J., Croux, C. (1993) Alternatives to the Median Absolute Deviation, *Journal of the American Statistical Association* **88**, 1273-1283.

See Also

`robreg.filter`, `wrm.filter`, `madore.filter`.

Examples

```
# # # # # # # # # #
# Short and noise-free time series
series <- c(rep(0,30),rep(10,30),seq(10,5,length=20),seq(5,15,length=20))

# Adaptive online signal extraction without & with 'restrict to range' rule
t.without.rtr <- adore.filter(series, rtr=0)
plot(t.without.rtr)
t.with.rtr1 <- adore.filter(series, rtr=1)
lines(t.with.rtr1$level, col="blue")
t.with.rtr2 <- adore.filter(series)
lines(t.with.rtr2$level, col="green3",lty=2)
legend("top",c("Signal with rtr=1","Signal with rtr=2"),col=c("blue","green3"),lty=c(1,2),bt

# # # # # # # # # #
# Short and noise-free time series + 1 outlier
ol.series <- series
```

```
ol.series[63] <- 3

# Adaptive online signal extraction without & with 'restrict to range' rule
t.without.rtr <- adore.filter(ol.series, rtr=0)
plot(t.without.rtr)
t.with.rtr1 <- adore.filter(ol.series, rtr=1)
lines(t.with.rtr1$level, col="blue")
t.with.rtr2 <- adore.filter(ol.series)
lines(t.with.rtr2$level, col="green3",lty=2)
legend("top",c("Signal with rtr=1","Signal with rtr=2"),col=c("blue","green3"),lty=c(1,2),bt

# # # # # # # # # #
# Noisy time series with level shifts, trend changes and shifts in the scale of the error te
true.signal  <- c(rep(0,150),rep(10,150),seq(10,5,length=100),seq(5,15,length=100))
series2      <- true.signal + c(rnorm(250,sd=1), rnorm(200,sd=3), rnorm(50,sd=1))

# Adaptive online signal extraction with additional Qn scale estimation
s2 <- adore.filter(series2, calc.qn=TRUE)
par(mfrow=c(3,1))
plot(s2)
plot(s2$sigma,type="l",main="Corresponding Qn Scale Estimation",ylab="sigma",xlab="time")
lines(c(rep(1,250),rep(3,200),rep(1,150)),col="grey")
legend("topleft",c("True scale","Qn"),lty=c(1,1),col=c("grey","black"),bty="n")
plot(s2$width,type="l",main="Corresponding Window Width",ylab="width",xlab="time")
```

`const` *Correction factors to achieve unbiasedness of the Qn scale estimator*

Description

This matrix contains correction factors for the univariate Qn scale estimator (Rousseeuw, Croux, 1993) to achieve unbiasedness under Gaussian noise. The `madore.filter` estimates the local error covariance matrix by the orthogonalized Gnanadesikan-Kettenring estimator (Gnanadesikan, Kettenring, 1972, Maronna, Zamar, 2002) which is based on the Qn scale estimator.

Usage

```
const
```

Format

A (96x2)-matrix containing the correction factors for the univariate Qn scale estimator for the samples sizes $n = 10, 11, ..., 100, 200, 300, 400, 500, 1000$.

Source

The correction factors have been obtained by simulations.

References

Gnanadesikan, R., Kettenring, J.R. (1972) Robust Estimates, Residuals, and Outlier Detection with Multiresponse Data, *Biometrics* **28**, 81-124.

Maronna, R.A., Zamar, R.H. (2002) Robust Estimates of Location and Dispersion for High-Dimensional Datasets, *Technometrics* **44**, 307-317.

Rousseeuw, P.J., Croux, C. (1993) Alternatives to the Median Absolute Deviation, *Journal of the American Statistical Association* **88**, 1273-1283.

`critvals` *Critical Values for the RM Goodness of Fit Test*

Description

This matrix contains critical values for the goodness of fit test for the last fitted value of a Repeated Median regression fit to a sample of size `n`. The critical values are based on the 0.95-quantiles of the distribution of a test statistic corresponding to the absolute value of the sum of a subset of residual signs. The critical value for a test based on the last `nI` out of `n` observations corresponds to `critvals[n,nI]`.

Usage

```
critvals
```

Format

A (600x61)-matrix containing 30550 observations.

Source

Simulation.

References

Schettlinger, K., Fried, R., Gather, U. (2008) Real Time Signal Processing by Adaptive Repeated Median Filters, *International Journal of Adaptive Control and Signal Processing*, submitted.

Siegel, A.F. (1982) Robust Regression Using Repeated Medians, *Biometrika* **69** (1), 242-244.

dw.filter *Robust Double Window Filtering Methods for Univariate Time Series*

Description

Procedures for robust (online) extraction of low frequency components (the *signal*) from a univariate time series based on a moving window technique using two nested time windows in each step.

Usage

```
dw.filter(y, outer.width, inner.width, method = "all",
             scale = "MAD", d = 2,
             minNonNAs = 5, online = FALSE, extrapolate = TRUE)
```

Arguments

`y` a numeric vector or (univariate) time series object.

`outer.width` a positive integer specifying the window width of the outer window used for determining the final estimate.
If `online=FALSE` (see below) this needs to be an odd integer.

`inner.width` a positive integer (not larger than `outer.width`) specifying the window width of the inner window used for determining the initial estimate and trimming features.
If `online=FALSE` (see below) this needs to be an odd integer.

`method` a (vector of) character string(s) containing the method(s) to be used for the estimation of the signal level.
It is possible to specify any combination of `"MED"`, `"RM"`, `"MTM"`, `"TRM"`, `"MRM"`, `"DWRM"`, `"DWMTM"`, `"DWTRM"`, `"DWMRM"` and `"all"` (for all of the above). Default is `method="all"`. For a detailed description see the section 'Methods' below.

`scale` a character string specifying the method to be used for robust estimation of the local variability (within one time window). Possible values are:

- **`"MAD"`** Median absolute deviation about the median (default)
- **`"QN"`** Rousseeuw's and Croux' (1993) Q_n scale estimator
- **`"SN"`** Rousseeuw's and Croux' (1993) S_n scale estimator

`d` a positive integer defining factor the current scale estimate is multiplied with for determining the trimming boundaries for outlier detection.
Observations deviating more than $d \cdot \hat{\sigma}_t$ from the current level approximation $\hat{\mu}_t$ are replaced by $\hat{\mu}_t$ where $\hat{\sigma}_t$ denotes the current scale estimate. Default is `d = 2` meaning a 2σ rule for outlier detection.

`minNonNAs` a positive integer defining the minimum number of non-missing observations within each window which is required for a 'sensible' estimation. Default: if windows contain less than `minNonNAs = 5` observations `NA`s are returned.

`online` a logical indicating whether the current level and scale estimates are evaluated at the most recent time within each (inner and outer) window (`TRUE`) or centred within the windows (`FALSE`). Setting `online=FALSE` requires odd `inner.width` and `outer.width`. Default is `online=FALSE`.

`extrapolate` a logical indicating whether the level estimations should be extrapolated to the edges of the time series.
If `online=FALSE` the extrapolation consists of the fitted values within the first half of the first window and the last half of the last window; if `online=TRUE` the extrapolation consists of the all fitted values within the first time window.

Details

`dw.filter` is suitable for extracting low frequency components (the *signal*) from a time series which may be contaminated with outliers and can contain level shifts. For this, moving window techniques are applied.

A short inner window of length `inner.width` is used in each step for calculating an initial level estimate (by using either the median or a robust regression fit) and a robust estimate of the local standard deviation. Observations deviating strongly from this initial fit are trimmed from an outer time window of length `outer.width`, and the signal level is estimated from the remaining observations (by using either a location or regression estimator). Values specified in `method` determine which combination of estimation methods should be applied to the inner and outer window (see section 'Methods' below).

The applied `method` should be chosen based on an a-priori guess of the underlying signal and the data quality: Location based method (`MED` / `MTM`) are recommended in case of a locally (piecewise) constant signal, regression based approaches (`RM` / `DWRM` / `TRM` / `MRM`) in case of locally linear, monotone trends.

Since no big differences have been reported between `TRM` and `MRM`, the quicker and somewhat more efficient `TRM` option might be preferred. `DWRM` is the quickest of all regression based methods and performs better than the ordinary `RM` at shifts, but it is the least robust and least efficient method.

If location based methods are used, the `inner.width` should be chosen at least twice the length of expected patches of subsequent outliers in the time series; if regression based methods are used, the `inner.width` should be at least three times that length, otherwise outlier patches can influence the estimations strongly. To increase the efficiency of the final estimates, `outer.width` can then be chosen rather large - provided that it is smaller than the time between subsequent level shifts.

For robust scale estimation, `MAD` is the classical choice; `SN` is a somewhat more efficient and almost equally robust alternative, while `QN` is much more efficient if the window widths are not too small, and it performs very well at the occurrence of level shifts.

The factor `d`, specifying the trimming boundaries as a multiple of the estimated scale, can be chosen similarly to classical rules for detecting unusual observations in a Gaussian sample. Choosing `d=3` instead of `d=2` increases efficiency, but decreases robustness; `d=2.5` might be seen as a compromise.

Value

`dw.filter` returns an object of class `dw.filter`. An object of class `dw.filter` is a list containing the following components:

level	a data frame containing the corresponding signal level extracted by the filter(s) specified in method.
slope	a data frame containing the corresponding slope within each time window.
sigma	a data frame containing inner.loc.sigma, inner.reg.sigma, outer.loc.sigma and outer.reg.sigma, the scale estimated from the observations (loc) or the residuals from the Repeated Median regression (reg) within the inner window of length inner.width or the outer window of length outer.width, respectively. MTM uses outer.loc.sigma for trimming outliers, MRM and TRM use outer.reg.sigma for trimming outliers, DWMTM uses inner.loc.sigma for trimming outliers, DWMRM and DWTRM use inner.reg.sigma for trimming outliers; MED, RM and RM require no scale estimation. The function only returns values for inner.loc.sigma, inner.reg.sigma, outer.loc.sigma or outer.reg.sigma if any specified method requires their estimation; otherwise NAs are returned.

In addition, the original input time series is returned as list member y, and the settings used for the analysis are returned as the list members outer.width, inner.width, method, scale, d, minNonNAs, online and extrapolate.

Application of the function plot to an object of class dw.filter returns a plot showing the original time series with the filtered output.

Methods

The following methods are available as method for signal extraction, whereby the prefix DW denotes the fact that different window widths are used in the first and second step of the calculations within one window (i.e. inner.width<outer.width) while for the methods MED, RM, MTM, TRM and MRM the first and second step take place in a window of fixed length outer.width.

MED ordinary running median filter.
The simple median is applied to the observations within a moving time window of length outer.width.

RM ordinary repeated median filter.
Repeated median regression is applied to the observations within a moving time window of length outer.width.

MTM, DWMTM modified trimmed mean filters.
In a first step the median is applied to (MTM): the whole window with outer.width or (DWMTM): the inner window with inner.width; in a second step the mean is applied to the (trimmed) observations in the whole window (with outer.width).

TRM, DWTRM trimmed repeated median filters.
In a first step repeated median regression is applied to (TRM): the whole window with outer.width or (DWTRM): the inner window with inner.width; in a second step least squares regression is applied to the (trimmed) observations in the whole window (with outer.width).

MRM, DWMRM modified repeated median filters.
In a first step repeated median regression is applied to (MRM): the whole window with outer.width or (DWMRM): the inner window with inner.width; in a second step another repeated median regression is applied to the (trimmed) observations in the whole window (with outer.width).

DWRM double window repeated median filter.
In a first step repeated median regression is applied to the inner window with `inner.width` to determine the trend (slope); in a second step the median is applied to the trend corrected observations in the whole window with `outer.width` (without trimming).

Note

Missing values are treated by omitting them and thus by reducing the corresponding window width. `MED`, `RM`, `MTM`, `TRM` and `MRM` require at least `minNonNAs` non-missing observations in each outer window; `DWRM`, `DWMTM`, `DWTRM` and `DWMRM` require at least `minNonNAs` non-missing observations in each inner window. Otherwise `NA`s are returned for `level`, `slope` and `sigma`.

Author(s)

Roland Fried and Karen Schettlinger

References

Bernholt, T., Fried, R., Gather, U., Wegener, I. (2006) Modified Repeated Median Filters, *Statistics and Computing* **16**, 177-192.
(earlier version: http://www.sfb475.uni-dortmund.de/berichte/tr46-04.ps)

Schettlinger, K., Fried, R., Gather, U. (2006) Robust Filters for Intensive Care Monitoring: Beyond the Running Median, *Biomedizinische Technik* **51**(2), 49-56.

See Also

`robreg.filter`, `robust.filter`, `hybrid.filter`, `wrm.filter`.

Examples

```
# Generate random time series:
y <- cumsum(runif(500)) - .5*(1:500)
# Add jumps:
y[200:500] <- y[200:500] + 5
y[400:500] <- y[400:500] - 7
# Add noise:
n <- sample(1:500, 30)
y[n] <- y[n] + rnorm(30)

# Filtering with all methods:
y.dw <- dw.filter(y, outer.width=31, inner.width=11, method="all")
# Plot:
plot(y.dw)

# Filtering with trimmed RM and double window TRM only:
y2.dw <- dw.filter(y, outer.width=31, inner.width=11, method=c("TRM","DWTRM"))
plot(y2.dw)
```

hybrid.filter *Robust Hybrid Filtering Methods for Univariate Time Series*

Description

Procedures for robust extraction of low frequency components (the *signal*) from a univariate time series based on a moving window technique using the median of several one-sided half-window estimates (subfilters) in each step.

Usage

```
hybrid.filter(y, width, method = "all", minNonNAs=3, extrapolate = TRUE)
```

Arguments

`y`	a numeric vector or (univariate) time series object.
`width`	an odd positive integer (≥ 3) defining the window width used for fitting.
`method`	a (vector of) character string(s) containing the method(s) to be used for the estimation of the signal level. It is possible to specify any combination of `"MED"`, `"RM"`, `"MEAN"`, `FMH`, `"PFMH"`, `"CFMH"`, `"MH"`, `"PRMH"`, `"CRMH"`, `"MMH"`, `"PRMMH"`, `"CRMMH"`, and `"all"` (for all of the above). Default is `method="all"`. For a detailed description see the section 'Methods' below.
`minNonNAs`	a positive integer defining the minimum number of non-missing observations within each window (half) which is required for a 'sensible' estimation. Default: if a window (half) contains less than `minNonNAs = 3` observations an `NA` is returned (for that subfilter).
`extrapolate`	a logical indicating whether the level estimations should be extrapolated to the edges of the time series. The extrapolation extends the first estimated value to the first time in the first window and the last estimated value to the last time in the last time window. Default is `extrapolate=TRUE`.

Details

`hybrid.filter` is suitable for extracting low frequency components (the *signal*) from a time series which may be contaminated with outliers and can contain level shifts or local extremes. For this, moving window techniques are applied.

Within each time window several subfilters are applied to half-windows (left and right of the centre); the final signal level in the centre of the time window is then estimated by the median of the subfilter outputs.

For the subfilters, both, location-based and regression-based method are available, the former applying means or medians and the idea of a locally constant signal value, the latter using ordinary least squares (LS) regression or Siegel's (1982) repeated median (RM) and the idea of an underlying locally linear trend.

The methods should be chosen based on an a-priori guess of the underlying signal and the data quality. Location based methods (`MED`, `MEAN`, `FMH`, `MH`, `MMH`) are recommended in case of a locally (piecewise) constant signal. Regression based and predictive approaches (`RM`, `PFMH`, `PRMH`, `PRMMH`) in case of locally linear monotone trends. The combined filters (`CFMH`, `CRMH`, `CRMMH`) can be seen as a compromise, but are computationally somewhat more expensive and may be inferior to the predictive filters during steep trends.

The approaches based on the median and RM are robust alternatives to the (in Gaussian samples) more efficient mean and least squares methods. The hybrid filters preserve shifts and local extremes much better than `MED`, `MEAN` or `RM` for the price of decreased robustness and / or Gaussian efficiency.

Value

`hybrid.filter` returns an object of class `hybrid.filter`. An object of class `hybrid.filter` is a list containing the following components:

`level` a data frame containing the signal level extracted by the filter(s) specified in `method`.

`slope` a data frame (possibly) containing `RM`, `RM.left`, `RM.right`, `LS.left` and `LS.right`: the slope estimated by Repeated Median regression in the whole window (for `method="RM"`) or in the left and right window half (for any `method` in `"PRMH"`, `"CRMH"`, `"PRMMH"` and `"CRMMH"`) or the least squares slope estimated from the left and right window half (for any `method` in `"PRFMH"` or `"CFMH"`).
Only those slopes are returned which are required by the filters specified in `method`. If only location-based filters are applied (i.e. `"MED"`, `"MEAN"`, `"FMH"`, `"MH"` and /or `"MMH"`) `NULL` is returned for the `slope`.

In addition, the original input time series is returned as list member `y`, and the settings used for the analysis are returned as the list members `width`, `method` and `extrapolate`.

Application of the function `plot` to an object of class `hybrid.filter` returns a plot showing the original time series with the filtered output.

Methods

The following methods are available as `method` for signal extraction.

Filters applying only *one* location or regression estimate to the whole window of length `width` and taking the location (in the centre of the time window) as final signal level estimate:

MED ordinary running median filter.

MEAN ordinary moving average filter.

RM ordinary repeated median filter.
Applies repeated median regression to each time window.

Filters applying several subfilters within one window, taking the median of the values listed below as the final signal level estimate:

FMH FIR median hybrid filter.
Uses half-window averages and the central observation.

PFMH predictive FMH filter.
Uses half-window least squares regression and the central observation.

CFMH combined FMH filter.
Uses half-window averages, half-window least squares regression, and the central observation.

MH median hybrid filter.
Uses half-window medians and the central observation.

PRMH predictive repeated median hybrid filter.
Uses half-window repeated median regression and the central observation.

CRMH combined repeated median hybrid filter.
Uses half-window medians, half-window repeated median regression, and the central observation.

MMH median/median hybrid filter.
Uses half-window medians and the median of all observations in the window.

PRMMH predictive repeated median/median filter.
Uses half-window repeated median regression and the median of all observations in the window.

CRMMH combined repeated median/median filter.
Uses half-window medians, half-window repeated median regression, and the median of all observations in the window.

Note

Missing values are treated by omitting them and thus by reducing the corresponding window width. The `hybrid.filter` function only offers filters for signal extraction delayed by (`width`+1)/2 time units, in contrast to other filters available from the `robfilter` package which also offer online time series analysis without time delay.

Author(s)

Roland Fried and Karen Schettlinger

References

Fried, R., Bernholt, T., Gather, U. (2006) Repeated Median and Hybrid Filters, *Computational Statistics & Data Analysis* **50**, 2313-2338.
(earlier version: http://www.sfb475.uni-dortmund.de/berichte/tr10-04.ps)

Schettlinger, K., Fried, R., Gather, U. (2006) Robust Filters for Intensive Care Monitoring: Beyond the Running Median, *Biomedizinische Technik* **51**(2), 49-56.

See Also

`robreg.filter`, `robust.filter`, `dw.filter`, `wrm.filter`.

Examples

```
# Generate random time series:
y <- cumsum(runif(500)) - .5*(1:500)
# Add jumps:
y[200:500] <- y[200:500] + 5
y[400:500] <- y[400:500] - 7
# Add noise:
n <- sample(1:500, 30)
y[n] <- y[n] + rnorm(30)
# Filtering with all methods:
y.hy <- hybrid.filter(y, width=31)
# Plot:
plot(y.hy)

# Filtering with running median and PRMH only:
y2.hy <- hybrid.filter(y, width=31, method=c("MED","PRMH"))
plot(y2.hy)
```

`madore.filter` *A Robust Adaptive Online Filter for Multivariate Time Series*

Description

Procedure for robust online extraction of a *signal* from a multivariate time series by a moving window technique with adaptive window width selection (*m*ultivariate *ad*aptive *o*nline *re*peated median filter). The window width adaption is based on the univariate `adore.filter`.

Usage

```
madore.filter(Y, byrow=FALSE,
                    min.width=20, max.width=200, start.width=min.width,
                    test.sample.size=min.width/2, width.search="geometric",
                    rtr.size=10, extr.delay=0,
                    NA.sample.size=10, minNonNAs=5)
```

Arguments

`Y`	a numeric matrix or (multivariate) time series object.
`byrow`	logical. If `FALSE` (the default), the filtering is done by columns, otherwise the filtering is done by rows.
`min.width`	a positive integer $\geq$ 10 specifying the minimal width of the moving time window.
`max.width`	a positive integer $\geq$ `min.width` specifying the maximal width of the moving time window. If `min.width` = `max.width`, the window width is fixed.
`start.width`	a positive integer $\geq$ `min.width`, $\leq$ `max.width` specifying the width of the first time window and thus the time point of the first signal estimation.

`test.sample.size`
: a positive integer ≥ 5, $\leq$ `min.width` defining a test window which indicates the rightmost/recent `test.sample.size` time points within the current time window. The *Repeated Median* (RM) regression residuals within the test window are used for a goodness of fit test (see `adore.filter`) which serves for determining an adequate window width at the given time point. For more details about the test, see Schettlinger, Fried, Gather (2008).

`width.search`
: a character string defining the search algorithm used for finding an adequate window width at each point in time.

"linear" The linear search always results in the largest window width possible and hence yields the smoothest online signal. However, if sudden changes (like level shifts) appear in the signal it requires a lot of computation time and thus, an increased variability of the extracted signal may be observed.

"binary" The binary search is recommended if it can be expected that the window width needs to be reduced drastically from a large to a very small value at certain times (for example at level shifts or trend changes). However, it may not always result in the largest possible window width.

"geometric" (default) The geometric search is as fast as the binary search but it puts more weight on large window widths. It offers a good compromise between the linear and the binary search (computation time vs. smooth output signal).

`rtr.size`
: a non-negative integer specifying the size of a subset of the most recent observations within each window. The signal estimation is restricted to the range of the observations within this subset.

`extr.delay`
: a non-negative integer $\leq$ `min.width`/2. The signal at time t is estimated with a delay of `extr.delay` time points.

`NA.sample.size`
: a positive integer ≥ 10, $\leq$ `min.width` specifying the size of a subset of the most recent observations within each window. See `minNonNAs`.

`minNonNAs`
: a positive integer ≥ 5, $\leq$ `NA.sample.size`. If a variable does not offer at least `minNonNAs` non-missing observations within the subset specified by `NA.sample.size`, the signal is not estimated for this variable at this time point t.

Details

The `madore.filter` is based on *Repeated Median* regression (Siegel, 1982) in moving time windows and serves for separating signals from noise and outliers in multivariate time series. At each time point t the test procedure of the *adaptive online Repeated Median* filter (Schettlinger, Fried, Gather, 2008) is used to determine an appropriate window width $n(t) \in [\texttt{min.width}, \texttt{max.width}]$. Then the signal vector at time t is estimated within the time window $(t - n(t) + 1, \ldots, t)$ by a slight modification of the multivariate *Trimmed Repeated Median-Least Squares* regression (Lanius, Gather, 2004). A more detailed description of the `madore.filter` can be found in Borowski, Schettlinger, Gather (2009).

Value

`madore.filter` returns an object of class `madore.filter`. An object of class `madore.filter` is a list containing the following components:

- `signals` a matrix containing the estimated signal vectors at each time point t.
- `widths` a matrix containing the individual window widths of each variable at each time point t.
- `overall.width` a vector containing the overall window widths at each time point t.

In addition, the original input data is returned as list member `Y`, and the settings used for the analysis are returned as the list members `byrow`, `min.width`, `max.width`, `start.width`, `test.sample.size`, `width.search`, `rtr.size`, `extr.delay`, `NA.sample.size`, and `minNonNAs`. Application of the function `plot` to an object of class `madore.filter` returns a plot showing the original multivariate time series with the filtered output.

Author(s)

Matthias Borowski

References

Borowski, M., Schettlinger, K., Gather, U. (2009) Multivariate Real Time Signal Extraction by a Robust Adaptive Regression Filter, *Communications in Statistics - Simulation and Computation* **38**, 426-440.

Lanius, V., Gather, U. (2007) Robust Online Signal Extraction from Multivariate Time Series, *Technical Report 38/07, SFB 475, Universität Dortmund, Germany.*

Schettlinger, K., Fried, R., Gather, U. (2009) Real Time Signal Processing by Adaptive Repeated Median Filters, *International Journal of Adaptive Control and Signal Processing, Special Issue on 'Signal Processing and Diagnosis: Biomedical Applications'*, to appear.

Siegel, A.F. (1982) Robust Regression Using Repeated Medians, *Biometrika* **69** (1), 242-244.

See Also

`robreg.filter`, `adore.filter`.

Examples

```
# load multivariate time series sample
data(multi.ts)

# extract signals from 'multi.ts' by madore.filter
# this may take some time, depending on your system
extr <- madore.filter(multi.ts, min.width=30, max.width=100, extr.delay=5)
plot(extr)
```

multi.ts *Generated Multivariate Time Series*

Description

This data matrix contains a 4-variate time series of length 500. It consists of two Blocks and two Doppler signals each overlaid by highly correlated bivariate Gaussian noise.

Usage

```
multi.ts
```

Format

A (500x4)-matrix containing a 4-variate time series of length 500.

Source

Data generated by means of the packages `wmtsa` and `MASS`.

robreg.filter *Robust Regression Filters for Univariate Time Series*

Description

Procedures for robust (online) extraction of low frequency components (the *signal*) from a univariate time series by applying robust regression techniques to moving time windows.

Usage

```
robreg.filter(y, width, method = "all", h = floor(width/2)+1,
                 minNonNAs = 5, online = FALSE, extrapolate = TRUE)

lts.filter(y, width, h = floor(width/2)+1,
              online = FALSE, extrapolate = TRUE)

med.filter(y, width, minNonNAs = 5, online = FALSE, extrapolate = TRUE)
rm.filter (y, width, minNonNAs = 5, online = FALSE, extrapolate = TRUE)

dr.filter (y, width, online = FALSE, extrapolate = TRUE)
lms.filter(y, width, online = FALSE, extrapolate = TRUE)
lqd.filter(y, width, online = FALSE, extrapolate = TRUE)
```

Arguments

`y` a numeric vector or (univariate) time series object.

`width` a positive integer defining the window width used for fitting.
If `online=FALSE` (see below) this needs to be an odd integer.

`method` a (vector of) character string(s) containing the method(s) to be used for robust approximation of the signal within one time window. It is possible to specify any combination of the values:

- **"DR"** Deepest Regression
- **"LMS"** Least Median of Squares regression
- **"LQD"** Least Quartile Difference regression
- **"LTS"** Least Trimmed Squares regression
- **"MED"** Median
- **"RM"** Repeated Median regression
- **"all"** all of the above (default)

Using `dr.filter`, `lms.filter`, `lqd.filter`, `lts.filter`, `med.filter` or `rm.filter` forces `"DR"`, `"LMS"`, `"LQD"`, `"LTS"`, `"MED"` or `"RM"` respectively.
Currently, only `method="MED"` and `method="RM"` (`med.filter` / `rm.filter`) can handle missing values in the input time series. For the other regression filters missing values have to be replaced before the analysis.

`h` a positive integer defining the trimming quantile for LTS regression.

`minNonNAs` a positive integer defining the minimum number of non-missing observations within one window which is required for a 'sensible' estimation. Currently, this option only has an effect for the two methods `"MED"` and /or `"RM"` (see `method`).

`online` a logical indicating whether the current level estimate is evaluated at the most recent time within each time window (`TRUE`) or centred within each window (`FALSE`). Setting `online=FALSE` requires the `width` to be odd. Default is `online=FALSE`.

`extrapolate` a logical indicating whether the level estimations should be extrapolated to the edges of the time series.
If `online=FALSE` the extrapolation consists of the fitted values within the first half of the first window and the last half of the last window; if `online=TRUE` the extrapolation consists of the fitted values within the first time window.

Details

`robreg.filter` is suitable for extracting low frequency components (the *signal*) from a time series which may be contaminated with outliers and can contain level shifts. For this, robust regression methods are applied to a moving window, and the signal level is estimated by the fitted value either at the end of each time window for online signal extraction without time delay (`online=TRUE`) or in the centre of each time window (`online=FALSE`).

Value

`robreg.filter` returns an object of class `robreg.filter`. An object of class `robreg.filter` is a list containing the following components:

`level`	a data frame containing the signal level extracted by the filter(s) specified in `method`.
`slope`	a data frame containing the corresponding slope within each time window.

In addition, the original input time series is returned as list member `y`, and the settings used for the analysis are returned as the list members `width`, `method`, `h`, `minNonNAs`, `online` and `extrapolate`.

Application of the function `plot` to an object of class `robreg.filter` returns a plot showing the original time series with the filtered output.

Note

Missing values are treated by omitting them and thus by reducing the corresponding window width. The estimated signal level is only returned as `NA` if the window the estimation is based on contains less than `minNonNAs` non-missing values.

Author(s)

C++ code: Thorsten Bernholt and Robin Nunkesser
Port to R: Roland Fried and Karen Schettlinger

References

Davies, P.L., Fried, R., Gather, U. (2004) Robust Signal Extraction for On-Line Monitoring Data, *Journal of Statistical Planning and Inference* **122**, 65-78.
(earlier version: `http://www.sfb475.uni-dortmund.de/berichte/tr02-02.ps`)

Gather, U., Schettlinger, K., Fried, R. (2006) Online Signal Extraction by Robust Linear Regression, *Computational Statistics* **21**(1), 33-51.
(earlier version: `http://www.sfb475.uni-dortmund.de/berichte/tr53-04.ps`)

Schettlinger, K., Fried, R., Gather, U. (2006) Robust Filters for Intensive Care Monitoring: Beyond the Running Median, *Biomedizinische Technik* **51**(2), 49-56.

See Also

`wrm.filter`, `robust.filter`, `dw.filter`, `hybrid.filter`.

Examples

```
# Generate random time series:
y <- cumsum(runif(500)) - .5*(1:500)
# Add jumps:
y[200:500] <- y[200:500] + 5
y[400:500] <- y[400:500] - 7
# Add noise:
n <- sample(1:500, 30)
```

```
y[n] <- y[n] + rnorm(30)

# Filtering with all methods:
y.rr <- robreg.filter(y, width=31, method=c("RM", "LMS", "LTS", "DR", "LQD"))
# Plot:
plot(y.rr)

# Delayed filtering with RM and LMS filter:
y2.rr <- robreg.filter(y,width=31,method=c("RM","LMS"))
plot(y2.rr)

# Online filtering with RM filter:
y3.rr <- rm.filter(y,width=41,online=TRUE)
plot(y3.rr)
```

`robust.filter` *Robust Filtering Methods for Univariate Time Series*

Description

Procedure for robust (online) extraction of low frequency components (the *signal*) from a univariate time series with optional rules for outlier replacement and shift detection.

Usage

```
robust.filter(y, width, trend = "RM", scale = "QN", outlier = "T",
                           shiftd = 2, wshift = floor(width/2), lbound = 0.1, p = 0.9,
                           adapt = 0, max.width = width,
                           online = FALSE, extrapolate = TRUE)
```

Arguments

`y` a numeric vector or (univariate) time series object.

`width` a positive integer defining the window width used for fitting. If `online=FALSE` (default) this needs to be an odd number.

`trend` a character string defining the method to be used for robust approximation of the signal within one time window. Possible values are:

- **"MED":** Median
- **"RM":** Repeated Median regression (default)
- **"LTS":** Least Trimmed Squares regression
- **"LMS":** Least Median of Squares regression

`scale` a character string defining the method to be used for robust estimation of the local variability (within one time window). Possible values are:

- **"MAD":** Median absolute deviation about the median

"QN": Rousseeuw's and Croux' (1993) Q_n scale estimator (default)

"SN": Rousseeuw's and Croux' (1993) S_n scale estimator

"LSH": Length of the shortest half

`outlier` a single character defining the rule to be used for outlier detection and outlier treatment. Observations deviating more than $d \cdot \hat{\sigma}_t$ from the current level approximation $\hat{\mu}_t$ are replaced by $\hat{\mu}_t \pm k\hat{\sigma}_t$ where $\hat{\sigma}_t$ denotes the current scale estimate. Possible values are:

"T": Replace ('trim') large outliers detected by a 3σ-rule ($d = 3$) by the current level estimate ($k = 0$). (default)

"L": Shrink large outliers ($d = 3$) strongly towards the current level estimate ($k = 1$).

"M": Shrink large and moderatly sized outliers ($d = 2$) strongly towards the current level estimate ($k = 1$).

"W": Shrink large and moderatly sized outliers ($d = 2$) towards the current level estimate ($k = 2$).

`W` is the most efficient, `T` the most robust method (which should ideally be combined with a suitable value of `lbound`).

`shiftd` a positive numeric value defining the factor the current scale estimate is multiplied with for shift detection. Default is `shiftd`=2 corresponding to a 2σ rule for shift detection.

`wshift` a positive integer specifying the number of the most recent observations used for shift detection (regulates therefore also the delay of shift detection). Only used in the `online` mode; should be less than half the (minimal) window width then. In the offline mode (`online=FALSE`, default), shift detection is based on the right half of the time window, i.e. `wshift=floor(width/2)` (default).

`lbound` a positive real value specifying an optional lower bound for the scale to prevent the scale estimate from reaching zero (implosion).

`p` a fraction $\in [2/3, 1]$ of observations for additional rules in case of only two or three different values within one window.
If 100 percent of the observations within one window take on only two different values, the current level is estimated by the mean of these values regardless of the `trend` specification. In case of three differing values the median is taken as the current level estimate.

`adapt` a numeric value defining the fraction which regulates the adaption of the moving window width. `adapt` can be either 0 or a value $\in [0.6, 1]$. `adapt = 0` means that a fixed window width is used. Otherwise, the window width is reduced whenever more than a fraction of `adapt` $\in [0.6, 1]$ of the residuals in a certain part of the current time window are all positive or all negative.

`max.width` a positive integer (`>= width`) specifying the maximal width of the time window.
`width` specifies the minimal (and also the initial) width.

`online` a logical indicating whether the current level and scale estimates are evaluated at the most recent time within each window (`TRUE`) or centered within the window (`FALSE`). `online=FALSE` (default) requires an odd `width` for the window and means a time delay of (`width`+1)/2 time units.

extrapolate a logical indicating whether the level estimations should be extrapolated to the edges of the time series.
If online=FALSE the extrapolation consists of the fitted values within the first half of the first window and the last half of the last window; if online=TRUE the extrapolation consists of all fitted values within the first time window.

Details

robust.filter works by applying the methods specified by trend and scale to a moving time window of length width.

Before moving the time window, it is checked whether the next (incoming) observation is considered an 'outlier' by applying the rule specified by outlier. Therefore, the trend in the current time window is extrapolated to the next point in time and the residual of the incoming observation is standardised by the current scale estimate.

After moving the time window, it can be tested whether a level shift has occurred within the window: If more than half of the residuals in the right part of the window are larger than shiftd$\cdot\sigma_t$, a shift is detected and appropriate actions are taken. In the online mode, the number of the rightmost residuals can be chosen by wshift to regulate the resistance of the detection rule against outliers, its power and the time delay of detection.

A more detailed description of the filter can be found in Fried (2004). The adaption of the window width is described by Gather and Fried (2004). For more explanations on shift detection, see Fried and Gather (2007).

Value

robust.filter returns an object of class robust.filter. An object of class robust.filter is a list containing the following components:

level a numeric vector containing the signal level extracted by the (regression) filter specified by trend, scale and outlier.

slope a numeric vector containing the corresponding slope within each time window.

sigma a numeric vector containing the corresponding scale within each time window.

ol an outlier indicator. 0: no outlier, +1: positive outlier, -1: negative outlier

level.shift a level shift indicator. 0: no level shift, t: positive level shift detected at processing time t, -t: negative level shift detected at processing time t (the position in the vector gives an estimate of the point in time before which the shift has occurred).

In addition, the original input time series is returned as list member y, and the settings used for the analysis are returned as the list members width, trend, scale, outlier, shiftd, wshift, lbound, p, adapt, max.width, online and extrapolate.

Application of the function plot to an object of class robust.filter returns a plot showing the original time series with the filtered output.

Note

Missing values have to be replaced or removed from the time series before applying robust.filter.

Author(s)

Roland Fried and Karen Schettlinger

References

Fried, R. (2004), Robust Filtering of Time Series with Trends, *Journal of Nonparametric Statistics* **16**, 313-328.
(earlier version: http://www.sfb475.uni-dortmund.de/berichte/tr30-03.ps)

Fried, R., Gather, U. (2007), On Rank Tests for Shift Detection in Time Series, *Computational Statistics and Data Analysis, Special Issue on Machine Learning and Robust Data Mining* **52**, 221-233.
(earlier version: http://www.sfb475.uni-dortmund.de/berichte/tr48-06.pdf)

Gather, U., Fried, R. (2004), Methods and Algorithms for Robust Filtering, *COMPSTAT 2004: Proceedings in Computational Statistics*, J. Antoch (eds.), Physika-Verlag, Heidelberg, 159-170.

Schettlinger, K., Fried, R., Gather, U. (2006) Robust Filters for Intensive Care Monitoring: Beyond the Running Median, *Biomedizinische Technik* **51**(2), 49-56.

See Also

robreg.filter, hybrid.filter, dw.filter, wrm.filter.

Examples

```
# Generate random time series:
y <- cumsum(runif(500)) - .5*(1:500)
# Add jumps:
y[200:500] <- y[200:500] + 5
y[400:500] <- y[400:500] - 7
# Add noise:
n <- sample(1:500, 30)
y[n] <- y[n] + rnorm(30)

# Delayed Filtering of the time series with window width 23:
y.rf <- robust.filter(y, width=23)
# Plot:
plot(y.rf)

# Delayed Filtering with different settings and fixed window width 31:
y.rf2 <- robust.filter(y, width=31, trend="LMS", scale="QN", outlier="W")
plot(y.rf2)

# Online Filtering with fixed window width 24:
y.rf3 <- robust.filter(y, width=24, online=TRUE)
plot(y.rf3)

# Delayed Filtering with adaptive window width (minimal width 11, maximal width 51):
y.rf4 <- robust.filter(y, width=11, adapt=0.7, max.width=51)
plot(y.rf4)
```

`wrm.filter` *Weighted Repeated Median Filters for Univariate Time Series*

Description

Filtering procedure based on a weighted version of Siegel's (1982) repeated median (RM) and a moving time window for robust extraction of low frequency components (the signal) in the presence of outliers and shifts. One of several weight functions can be chosen to weight the observations in each time window.

Usage

```
wrm.filter(y, width, weight = 1, del = floor(width/2), extrapolate = TRUE)
```

Arguments

`y`	a numeric vector or (univariate) time series object.
`width`	a positive integer defining the window width used for fitting. If `del = floor(width/2)` (default) this needs to be an odd number.
`weight`	Indicates the weight function used. **`weight=0:`** equal weighting **`weight=1:`** triangular weights (default) **`weight=2:`** Epanechnikov weights
`del`	a positve integer (smaller than width) specifying the delay of the signal extraction. `del=0` means online signal extraction without delay. Default is `del=floor(width/2)`
`extrapolate`	a logical indicating whether the level estimations should be extrapolated to the edges of the time series. If `del = floor(width/2)` (default) the extrapolation consists of the fitted values within the first half of the first window and the last half of the last window; if `del=0` the extrapolation consists of the all fitted values within the first time window.

Details

For online signal extraction without time delay, weighted repeated median filtering with triangular weights is recommendable in the presence of isolated outliers and abrupt level shifts since it reacts more quickly to shifts than unweighted repeated median filtering and provides higher efficiencies. The window width should be chosen based on a guess of the minimal time period in which the signal can be approximated by a straight line without abrupt shifts. Better results can be obtained by increasing the delay, but often minimization of the time delay itself is one of the objectives so that one prefers del=0. The procedure replaces missing values by simple extrapolations if these are not within the first time window used for initialization.

For "offline" situations, it is intuitive to set `del` roughly equal to `width/2`. If the focus is rather on smoothing than on signal extraction, the Epanechnikov kernel should be used rather than the triangular kernel. In this case one can also use directly function `wrm.smooth`.

Value

`wrm.filter` returns an object of class `wrm.filter`. An object of class `wrm.filter` is a list containing the following components:

`y`	the original input time series.
`level`	the corresponding signal level extracted by the filter.
`slope`	the corresponding slope within each time window.
`del`	the parameter specifying the delay of the signal extraction.
`width`	width of the time window.
`weight`	name of the weight function used for the fit.

The function `plot` returns a plot showing the original time series with the filtered output.

Author(s)

Roland Fried and Jochen Einbeck

References

These filtering procedures are described and investigated in
Fried, R., Einbeck, J., Gather, U. (2007), Weighted Repeated Median Smoothing and Filtering, *Journal of the American Statistical Association* **102**, 1300-1308.
Preliminary version available as technical report from `www.sfb475.uni-dortmund.de/berichte/tr33-05.pdf`

See Also

`dw.filter`, `hybrid.filter`, `wrm.smooth`

Examples

```
data(Nile)
nile <- as.numeric(Nile)
obj <- wrm.filter(nile, width=11)
plot(obj)
```

`wrm.smooth` *Weighted Repeated Median Smoothing*

Description

A robust smoothing tool using a kernel weighted version of Siegel's (1982) repeated median. It can be seen as an alternative to local linear L1 regression.

Usage

```
wrm.smooth(x, y, h, xgrid, weight = 2)
```

Arguments

`x`	Vector of predictors.
`y`	Vector of responses, needs to have the same length as `x`.
`h`	Bandwidth, measured in the same units as the explanatory (independent) variable `x`: (x[0]-`h`,x[0]+`h`) is the range of x-values to be included in the local smoothing at x[0]. Needs to be a positive number.
`xgrid`	Grid on which fitted values are to be evaluated. The default is here to take the input values `x` for a sample size of at most 100, and `seq(min(x),max(x), l=100)` otherwise.
`weight`	Indicates the weight function used. **`weight=1`** triangular weights **`weight=2`** Epanechnikov weights (default) **`weight=3`** Gaussian weights **`weight=4`** Biweight **`weight=5`** Uniform weights

Details

Weighted repeated median (WRM) smoothing was suggested in a signal extraction framework by Fried, Einbeck & Gather (2007). It combines the advantages of weighted and repeated medians, i.e. the WRM smoother is robust to outliers and adapts to linear trends (through the slope parameter of the repeated median, which is calculated by applying two consecutive weighted medians onto the pairwise slopes). The theory and simulations provided by Fried, Einbeck & Gather focus on online signal extraction from time series. Warning: The case of a kernel weighted repeated median smoother for arbitraty non-equidistant design (as implemented here) is not fully investigated yet.

The procedure copes with missing values by omitting them.

Value

wrm.smooth returns an object of class wrm.smooth. An object of class wrm.smooth is a list containing the following components:

y	the original input time series.
level	the corresponding signal level extraceted by the weighted Repeated Median filter.
slope	the corresponding WRM slope within each time window.
h	bandwidth.
xgrid	vector with grid values.
weight	name of the weight function used for the fit.

The function plot returns a plot showing the original data with the smoothed output.

Author(s)

Jochen Einbeck and Roland Fried

References

Fried, R., Einbeck, J., Gather, U. (2007), Weighted Repeated Median Smoothing and Filtering, *Journal of the American Statistical Association* **102**, 1300-1308.
Preliminary version available as technical report from www.sfb475.uni-dortmund.de/berichte/tr33-05.pdf

Siegel, A.F. (1982). Robust regression using repeated medians. *Biometrika* **68**, 242-244.

See Also

wrm.filter

Examples

```
data(faithful) # Old Faithful Geyser data
faith.WRM <- wrm.smooth(faithful$w, faithful$e,h=4)
plot(faith.WRM)
```

Index