Package ‘highfrequency’

August 17, 2022

Version 0.9.5

Date 2022-08-17

Title Tools for Highfrequency Data Analysis

Description Provide functionality to manage, clean and match highfrequency trades and quotes data, calculate various liquidity measures, estimate and forecast volatility, detect price jumps and investigate microstructure noise and intraday periodicity. A detailed vignette can be found in the paper ``Analyzing intraday financial data in R: The highfrequency package'' by Boudt, Kleen, and Sjoerup (2022, <doi:10.2139/ssrn.3917548>).

License GPL (>= 2)

Encoding UTF-8

LazyData true

URL https://github.com/jonathancornelissen/highfrequency

BugReports https://github.com/jonathancornelissen/highfrequency/issues

Depends R (>= 3.5.0)

Imports xts, zoo, Rcpp, graphics, methods, stats, utils, grDevices, robustbase, data.table (>= 1.12.0), RcppRoll, quantmod, sandwich, numDeriv, Rsolnp

LinkingTo Rcpp, RcppArmadillo

Suggests mvtnorm, covr, FKF, rugarch, testthat, knitr, rmarkdown

RoxygenNote 7.2.1

NeedsCompilation yes

Author Kris Boudt [aut, cre] (<https://orcid.org/0000-0002-1000-5142>),
Jonathan Cornelissen [aut],
Scott Payseur [aut],
Giang Nguyen [ctb],
Onno Kleen [aut] (<https://orcid.org/0000-0003-4731-4640>),
Emil Sjoerup [aut]

Maintainer Kris Boudt <kris.boudt@ugent.be>

Repository CRAN

Date/Publication 2022-08-17 16:10:02 UTC

R topics documented:

`highfrequency-package` *highfrequency: Tools for Highfrequency Data Analysis*

Description

The **highfrequency** package provides numerous tools for analyzing high-frequency financial data, including functionality to:

- Clean, handle, and manage high frequency trades and quotes data.
- Calculate liquidity measures
- Calculate (multivariate) realized measures of the distribution of high-frequency returns
- Estimate models for realized measures of volatility and the corresponding forecasts
- Detect jumps in prices
- Analyze market microstructure noise in asset prices
- Estimate spot volatility and drift as well as analyze intraday periodicity of spot volatility

Author(s)

Kris Boudt, Jonathan Cornelissen, Onno Kleen, Scott Payseur, Emil Sjoerup Maintainer: Kris Boudt <Kris.Boudt@ugent.be>

Contributors: Giang Nguyen

Thanks: We would like to thank Brian Peterson, Chris Blakely, Dirk Eddelbuettel, Maarten Schermer, and Eric Zivot

See Also

Useful links:

- `https://github.com/jonathancornelissen/highfrequency`
- Report bugs at `https://github.com/jonathancornelissen/highfrequency/issues`

aggregatePrice *Aggregate a time series but keep first and last observation*

Description

Function to aggregate high frequency data by last tick aggregation to an arbitrary periodicity based on wall clocks. Alternatively the aggregation can be done by number of ticks. In case we DON'T do tick-based aggregation, this function accepts arbitrary number of symbols over a arbitrary number of days. Although the function has the word Price in the name, the function is general and works on arbitrary time series, either `xts` or `data.table` objects the latter requires a `DT` column containing POSIXct time stamps.

Usage

```
aggregatePrice(
  pData,
  alignBy = "minutes",
  alignPeriod = 1,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  fill = FALSE,
  tz = NULL
)
```

Arguments

`pData`	`data.table` or `xts` object to be aggregated containing the intraday price series, possibly across multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: "secs", "seconds", "mins", "minutes","hours", and "ticks". To aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
`marketOpen`	the market opening time, by default: `marketOpen = "09:30:00"`.
`marketClose`	the market closing time, by default: `marketClose = "16:00:00"`.
`fill`	indicates whether rows without trades should be added with the most recent value, FALSE by default.
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`

Details

The time stamps of the new time series are the closing times and/or days of the intervals. The element of the returned series with e.g. time stamp 09:35:00 contains the last observation up to that point, including the value at 09:35:00 itself.

In case `alignBy = "ticks"`, the sampling is done such the sampling starts on the first tick, and the last tick is always included. For example, if 14 observations are made on one day, and these are 1, 2, 3, ... 14. Then, with `alignBy = "ticks"` and `alignPeriod = 3`, the output will be 1, 4, 7, 10, 13, 14.

Value

A `data.table` or `xts` object containing the aggregated time series.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Examples

```
# Aggregate price data to the 30 second frequency
aggregatePrice(sampleTData, alignBy = "secs", alignPeriod = 30)
# Aggregate price data to the 30 second frequency including zero return price changes
aggregatePrice(sampleTData, alignBy = "secs", alignPeriod = 30)

# Aggregate price data to half a second frequency including zero return price changes
aggregatePrice(sampleTData, alignBy = "milliseconds", alignPeriod = 500, fill = TRUE)
```

---

aggregateQuotes   *Aggregate a* `data.table` *or* `xts` *object containing quote data*

Description

Aggregate tick-by-tick quote data and return a `data.table` or `xts` object containing the aggregated quote data. See `sampleQData` for an example of the argument qData. This function accepts arbitrary number of symbols over an arbitrary number of days.

Usage

```
aggregateQuotes(
  qData,
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL
)
```

Arguments

- `qData` — `data.table` or `xts` object to be aggregated, containing the intraday quote data of a stock for one day.
- `alignBy` — character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: "secs", "seconds", "mins", "minutes","hours", and "ticks". To aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
- `alignPeriod` — positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
- `marketOpen` — the market opening time, by default: `marketOpen = "09:30:00"`.
- `marketClose` — the market closing time, by default: `marketClose = "16:00:00"`.
- `tz` — fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`

Details

The output "BID" and "OFR" columns are constructed using previous tick aggregation.

The variables "BIDSIZ" and "OFRSIZ" are aggregated by taking the sum of the respective inputs over each interval.

The timestamps of the new time series are the closing times of the intervals.

Please note: Returned objects always contain the first observation (i.e. opening quotes,...).

Value

A `data.table` or an `xts` object containing the aggregated quote data.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Examples

```
# Aggregate quote data to the 30 second frequency
qDataAggregated <- aggregateQuotes(sampleQData, alignBy = "seconds", alignPeriod = 30)
qDataAggregated # Show the aggregated data
```

aggregateTrades *Aggregate a* `data.table` *or* `xts` *object containing trades data´*

Description

Aggregate tick-by-tick trade data and return a time series as a `data.table` or `xts` object where first observation is always the opening price and subsequent observations are the closing prices over the interval. This function accepts arbitrary number of symbols over an arbitrary number of days.

Usage

```
aggregateTrades(
  tData,
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL
)
```

Arguments

`tData`	`data.table` or `xts` object to be aggregated, containing the intraday price series of a stock for possibly multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`. To aggregate based on a 5 minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5 minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`marketOpen`	the market opening time, by default: `marketOpen = "09:30:00"`.
`marketClose`	the market closing time, by default: `marketClose = "16:00:00"`.
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`

Details

The time stamps of the new time series are the closing times and/or days of the intervals.

The output `"PRICE"` column is constructed using previous tick aggregation.

The variable `"SIZE"` is aggregated by taking the sum over each interval.

The variable `"VWPRICE"` is the aggregated price weighted by volume.

The time stamps of the new time series are the closing times of the intervals.

In case of previous tick aggregation or `alignBy = "seconds"/"minutes"/"hours"`, the element of the returned series with e.g. time stamp 09:35:00 contains the last observation up to that point, including the value at 09:35:00 itself.

Value

A `data.table` or `xts` object containing the aggregated time series.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Examples

```
# Aggregate trade data to 5 minute frequency
tDataAggregated <- aggregateTrades(sampleTData, alignBy = "minutes", alignPeriod = 5)
tDataAggregated
```

---

aggregateTS — *Aggregate a time series*

Description

Aggregate a time series as `xts` or `data.table` object. It can handle irregularly spaced time series and returns a regularly spaced one. Use univariate time series as input for this function and check out `aggregateTrades` and `aggregateQuotes` to aggregate Trade or Quote data objects.

Usage

```
aggregateTS(
  ts,
  FUN = "previoustick",
  alignBy = "minutes",
  alignPeriod = 1,
  weights = NULL,
  dropna = FALSE,
  tz = NULL,
  ...
)
```

Arguments

`ts`	`xts` or `data.table` object to aggregate.
`FUN`	function to apply over each interval. By default, previous tick aggregation is done. Alternatively one can set e.g. FUN = "mean". In case weights are supplied, this argument is ignored and a weighted average is taken.

alignBy — character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: "secs", "seconds", "mins", "minutes", "hours", "days", "weeks", "ticks".

alignPeriod — positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.

weights — By default, no weighting scheme is used. When you assign an `xts` object with weights to this argument, a weighted mean is taken over each interval. Of course, the weights should have the same time stamps as the supplied time series.

dropna — boolean, which determines whether empty intervals should be dropped. By default, an NA is returned in case an interval is empty, except when the user opts for previous tick aggregation, by setting `FUN = "previoustick"` (default).

tz — character denoting which timezone the output should be in. Defaults to `NULL`

... — extra parameters passed on to `FUN`

Details

The time stamps of the new time series are the closing times and/or days of the intervals. For example, for a weekly aggregation the new time stamp is the last day in that particular week (namely Sunday).

In case of previous tick aggregation, for `alignBy` is either `"seconds"` `"minutes"`, or `"hours"`, the element of the returned series with e.g. timestamp 09:35:00 contains the last observation up to that point, including the value at 09:35:00 itself.

Please note: In case an interval is empty, by default an NA is returned.. In case e.g. previous tick aggregation it makes sense to fill these NAs by the function `na.locf` (last observation carried forward) from the **zoo** package.

In case `alignBy = "ticks"`, the sampling is done such the sampling starts on the first tick and the last tick is always included. For example, if 14 observations are made on one day, and these are 1, 2, 3, ... 14. Then, with `alignBy = "ticks"` and `alignPeriod = 3`, the output will be 1, 4, 7, 10, 13, 14.

Value

An `xts` object containing the aggregated time series.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

Examples

```
# Load sample price data
## Not run:
library(xts)
ts <- as.xts(sampleTData[, list(DT, PRICE, SIZE)])
```

```
# Previous tick aggregation to the 5-minute sampling frequency:
tsagg5min <- aggregateTS(ts, alignBy = "minutes", alignPeriod = 5)
head(tsagg5min)
# Previous tick aggregation to the 30-second sampling frequency:
tsagg30sec <- aggregateTS(ts, alignBy = "seconds", alignPeriod = 30)
tail(tsagg30sec)
tsagg3ticks <- aggregateTS(ts, alignBy = "ticks", alignPeriod = 3)

## End(Not run)
```

`AJjumpTest` *Ait-Sahalia and Jacod (2009) tests for the presence of jumps in the price series.*

Description

This test examines the presence of jumps in highfrequency price series. It is based on the theory of Ait-Sahalia and Jacod (2009). It consists in comparing the multi-power variation of equi-spaced returns computed at a fast time scale (h), $r_{t,i}$ ($i = 1, \ldots, N$) and those computed at the slower time scale (kh), $y_{t,i}$($i = 1, \ldots,$ N/k).

They found that the limit (for $N \to \infty$) of the realized power variation is invariant for different sampling scales and that their ratio is 1 in case of jumps and k$^{p/2} - 1$ if no jumps. Therefore the AJ test detects the presence of jump using the ratio of realized power variation sampled from two scales. The null hypothesis is no jumps.

The function returns three outcomes: 1.z-test value 2.critical value under confidence level of 95% and 3. p-value.

Assume there is N equispaced returns in period t. Let $r_{t,i}$ be a return (with $i = 1, \ldots, N$) in period t.

And there is N/k equispaced returns in period t. Let $y_{t,i}$ be a return (with $i = 1, \ldots,$ N/k) in period t.

Then the AJjumpTest is given by:

$$\text{AJjumpTest}_{t,N} = \frac{S_t(p,k,h) - k^{p/2-1}}{\sqrt{V_{t,N}}}$$

in which,

$$\text{S}_t(p,k,h) = \frac{PV_{t,M}(p,kh)}{PV_{t,M}(p,h)}$$

$$\text{PV}_{t,N}(p,kh) = \sum_{i=1}^{N/k} |y_{t,i}|^p$$

$$\mathrm{PV}_{t,N}(p,h) = \sum_{i=1}^{N} |r_{t,i}|^p$$

$$\mathrm{V}_{t,N} = \frac{N(p,k)A_{t,N(2p)}}{NA_{t,N(p)}}$$

$$\mathrm{N}(p,k) = \left(\frac{1}{\mu_p^2}\right)(k^{p-2}(1+k))\mu_{2p} + k^{p-2}(k-1)\mu_p^2 - 2k^{p/2-1}\mu_{k,p}$$

$$\mathrm{A}_{t,n(2p)} = \frac{(1/N)^{(1-p/2)}}{\mu_p} \sum_{i=1}^{N} |r_{t,i}|^p \text{ for } |r_j| < \alpha(1/N)^w$$

$$\mu_{k,p} = E(|U|^p|U + \sqrt{k-1}V|^p)$$

U, V: independent standard normal random variables; $h = 1/N$; p, k, α, w: parameters.

Usage

```
AJjumpTest(
  pData,
  p = 4,
  k = 2,
  alignBy = NULL,
  alignPeriod = NULL,
  alphaMultiplier = 4,
  alpha = 0.975,
  ...
)
```

Arguments

`pData`	either an `xts` or a `data.table` containing the prices of a single asset, possibly over multiple days.
`p`	can be chosen among 2 or 3 or 4. The author suggests 4. 4 by default.
`k`	can be chosen among 2 or 3 or 4. The author suggests 2. 2 by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"` To aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to "minutes".
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5 minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.

`alphaMultiplier`
: alpha multiplier

`alpha`
: numeric of length one with the significance level to use for the jump test(s). Defaults to 0.975.

`...`
: used internally

Details

The theoretical framework underlying jump test is that the logarithmic price process X_t belongs to the class of Brownian semimartingales, which can be written as:

$$\mathrm{X}_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u + Z_t$$

where a is the drift term, σ denotes the spot volatility process, W is a standard Brownian motion and Z is a jump process defined by:

$$\mathrm{Z}_t = \sum_{j=1}^{N_t} k_j$$

where k_j are nonzero random variables. The counting process can be either finite or infinite for finite or infinite activity jumps.

Using the convergence properties of power variation and its dependence on the time scale on which it is measured, Ait-Sahalia and Jacod (2009) define a new variable which converges to 1 in the presence of jumps in the underlying return series, or to another deterministic and known number in the absence of jumps (Theodosiou and Zikes, 2009).

Value

a list or `xts` in depending on whether input prices span more than one day.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Ait-Sahalia, Y. and Jacod, J. (2009). Testing for jumps in a discretely observed process. The Annals of Statistics, 37(1), 184-222.

Theodosiou, M., & Zikes, F. (2009). A comprehensive comparison of alternative tests for jumps in asset prices. Unpublished manuscript, Graduate School of Business, Imperial College London.

Examples

```
jt <- AJjumpTest(sampleTData[, list(DT, PRICE)], p = 2, k = 3,
                 alignBy = "seconds", alignPeriod = 5, makeReturns = TRUE)
```

---

autoSelectExchangeQuotes
Retain only data from the stock exchange with the highest volume

---

Description

Filters raw quote data and return only data that stems from the exchange with the highest value for the sum of "BIDSIZ" and "OFRSIZ", i.e. the highest quote volume.

Usage

```
autoSelectExchangeQuotes(qData, printExchange = TRUE)
```

Arguments

qData	a data.table or xts object with at least a column "EX", indicating the exchange symbol and columns "BIDSIZ" and "OFRSIZ", indicating the volume available at the bid and ask respectively.
printExchange	indicates whether the chosen exchange is printed on the console, default is TRUE. The possible exchanges are: • A: AMEX • N: NYSE • B: Boston • P: Arca • C: NSX • T/Q: NASDAQ • D: NASD ADF and TRF • X: Philadelphia • I: ISE • M: Chicago • W: CBOE • Z: BATS

Value

data.table or xts object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Examples

```
autoSelectExchangeQuotes(sampleQDataRaw)
```

`autoSelectExchangeTrades` *Retain only data from the stock exchange with the highest trading volume*

Description

Filters raw trade data and return only data that stems from the exchange with the highest value for the variable `"SIZE"`, i.e. the highest trade volume.

Usage

```
autoSelectExchangeTrades(tData, printExchange = TRUE)
```

Arguments

`tData`	an `xts` object with at least a column `"EX"` indicating the exchange symbol and `"SIZE"` indicating the trade volume.
`printExchange`	indicates whether the chosen exchange is printed on the console, default is `TRUE`. The possible exchanges are: • A: AMEX • N: NYSE • B: Boston • P: Arca • C: NSX • T/Q: NASDAQ • D: NASD ADF and TRF • X: Philadelphia • I: ISE • M: Chicago • W: CBOE • Z: BATS

Value

`data.table` or `xts` object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Examples

```
autoSelectExchangeTrades(sampleTDataRaw)
```

BNSjumpTest *Barndorff-Nielsen and Shephard (2006) tests for the presence of jumps in the price series.*

Description

This test examines the presence of jumps in highfrequency price series. It is based on theory of Barndorff-Nielsen and Shephard (2006). The null hypothesis is that there are no jumps.

Usage

```
BNSjumpTest(
  rData,
  IVestimator = "BV",
  IQestimator = "TP",
  type = "linear",
  logTransform = FALSE,
  max = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  alpha = 0.975
)
```

Arguments

rData	either an xts or a data.table containing the log-returns or prices of a single asset, possibly over multiple days-
IVestimator	can be chosen among jump robust integrated variance estimators: rBPCov, rMinRVar, rMedRVar, rOWCov and corrected threshold bipower variation (rThresholdCov). If rThresholdCov is chosen, an argument of startV, start point of auxiliary estimators in threshold estimation can be included. rBPCov by default.
IQestimator	can be chosen among jump robust integrated quarticity estimators: rTPQuar, rQPVar, rMinRQuar and rMedRQuar. rTPQuar by default.
type	a method of BNS testing: can be linear or ratio. Linear by default.
logTransform	boolean, should be TRUE when QVestimator and IVestimator are in logarithm form. FALSE by default.
max	boolean, should be TRUE when max adjustment in SE. FALSE by default.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours" To aggregate based on a 5 minute frequency, set alignPeriod = 5 and alignBy = "minutes".
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5 minute frequency, set alignPeriod = 5 and alignBy = "minutes".

makeReturns	boolean, should be TRUE when pData contains prices. FALSE by default.
alpha	numeric of length one with the significance level to use for the jump test(s). Defaults to 0.975.

Details

Assume there is N equispaced returns in period t. Assume the Realized variance (RV), IVestimator and IQestimator are based on N equi-spaced returns.

Let $r_{t,i}$ be a return (with $i = 1, \ldots, N$) in period t.

Then the BNSjumpTest is given by

$$\text{BNSjumpTest} = \frac{\texttt{RV} - \texttt{IVestimator}}{\sqrt{(\theta - 2)\frac{1}{N}\texttt{IQestimator}}}.$$

The options for IVestimator and IQestimator are listed above. θ depends on the chosen IVestimator (Huang and Tauchen, 2005).

The theoretical framework underlying the jump test is that the logarithmic price process X_t belongs to the class of Brownian semimartingales, which can be written as:

$$\mathrm{X}_t = \int_0^t a_u \, du + \int_0^t \sigma_u \, dW_u + Z_t$$

where a is the drift term, σ denotes the spot volatility process, W is a standard Brownian motion and Z is a jump process defined by:

$$\mathrm{Z}_t = \sum_{j=1}^{N_t} k_j$$

where k_j are nonzero random variables. The counting process can be either finite or infinite for finite or infinite activity jumps.

Since the realized volatility converges to the sum of integrated variance and jump variation, while the robust IVestimator converges to the integrated variance, it follows that the difference between RV and the IVestimator captures the jump part only, and this observation underlines the BNS test for jumps (Theodosiou and Zikes, 2009).

Value

a list or xts (depending on whether input prices span more than one day) with the following values:

- z-test value.
- critical value (with confidence level of 95%).
- p-value of the test.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., and Shephard, N. (2006). Econometrics of testing for jumps in financial economics using bipower variation. *Journal of Financial Econometrics*, 4, 1-30.

Corsi, F., Pirino, D., and Reno, R. (2010). Threshold bipower variation and the impact of jumps on volatility forecasting. *Journal of Econometrics*, 159, 276-288.

Huang, X., and Tauchen, G. (2005). The relative contribution of jumps to total price variance. *Journal of Financial Econometrics*, 3, 456-499.

Theodosiou, M., and Zikes, F. (2009). A comprehensive comparison of alternative tests for jumps in asset prices. Unpublished manuscript, Graduate School of Business, Imperial College London.

Examples

```
bns <- BNSjumpTest(sampleTData[, list(DT, PRICE)], IVestimator= "rMinRVar",
                   IQestimator = "rMedRQuar", type= "linear", makeReturns = TRUE)
bns
```

`businessTimeAggregation` *Business time aggregation*

Description

Time series aggregation based on 'business time' statistics. Instead of equidistant sampling based on time during a trading day, business time sampling creates measures and samples equidistantly using these instead. For example when sampling based on volume, business time aggregation will result in a time series that has an equal amount of volume between each observation (if possible).

Usage

```
businessTimeAggregation(
  pData,
  measure = "volume",
  obs = 390,
  bandwidth = 0.075,
  tz = NULL,
  ...
)
```

Arguments

`pData`	`xts` or `data.table` containing data to aggregate.
`measure`	character denoting which measure to use. Valid options are `"intensity"`, `"vol"`, and `"volume"`, denoting the trade intensity process of Oomen (2005), volatility, and volume, respectively. Default is `"volume"`.

obs	integer valued numeric of length 1 denoting how many observations is wanted after the aggregation procedure.
bandwidth	numeric of length one, denoting which bandwidth parameter to use in the trade intensity process estimation of Oomen (2005).
tz	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`.
...	extra arguments passed on to `spotVol` when measure is `"vol"`.

Value

A list containing `"pData"` which is the aggregated data and a list containing the intensity process, split up day by day.

Author(s)

Emil Sjoerup.

References

Dong, Y., and Tse, Y. K. (2017). Business time sampling scheme with applications to testing semi-martingale hypothesis and estimating integrated volatility. *Econometrics*, 5, 51.

Oomen, R. C. A. (2006). Properties of realized variance under alternative sampling schemes. *Journal of Business & Economic Statistics*, 24, 219-237

Examples

```
pData <- sampleTData[,list(DT, PRICE, SIZE)]
# Aggregate based on the trade intensity measure. Getting 390 observations.
agged <- businessTimeAggregation(pData, measure = "intensity", obs = 390, bandwidth = 0.075)
# Plot the trade intensity measure
plot.ts(agged$intensityProcess$`2018-01-02`)
rCov(agged$pData[, list(DT, PRICE)], makeReturns = TRUE)
rCov(pData[,list(DT, PRICE)], makeReturns = TRUE, alignBy = "minutes", alignPeriod = 1)

# Aggregate based on the volume measure. Getting 78 observations.
agged <- businessTimeAggregation(pData, measure = "volume", obs = 78)
rCov(agged$pData[,list(DT, PRICE)], makeReturns = TRUE)
rCov(pData[,list(DT, PRICE)], makeReturns = TRUE, alignBy = "minutes", alignPeriod = 5)
```

`driftBursts` *Inference on drift burst hypothesis*

Description

Calculates the test-statistic for the drift burst hypothesis

Let the efficient log-price be defined as:

$$dX_t = \mu_t dt + \sigma_t dW_t + dJ_t,$$

where μ_t, σ_t, and J_t are the spot drift, the spot volatility, and a jump process respectively. However, due to microstructure noise, the observed log-price is

$$Y_t = X_t + \varepsilon_t$$

In order robustify the results to the presence of market microstructure noise, the pre-averaged returns are used:

$$\Delta_i^n \overline{Y} = \sum_{j=1}^{k_n - 1} g_j^n \Delta_{i+j}^n Y,$$

where $g(\cdot)$ is a weighting function, $min(x, 1-x)$, and k_n is the pre-averaging horizon.

The test statistic for the Drift Burst Hypothesis can then be calculated as

$$\bar{T}_t^n = \sqrt{\frac{h_n}{K_2}} \frac{\hat{\bar{\mu}}_t^n}{\sqrt{\hat{\bar{\sigma}}_t^n}},$$

where

$$\hat{\bar{\mu}}_t^n = \frac{1}{h_n} \sum_{i=1}^{n-k_n+2} K\left(\frac{t_{i-1} - t}{h_n}\right) \Delta_{i-1}^n \overline{Y},$$

and

$\hat{\bar{\sigma}}_t^n = \frac{1}{h'_n} \left[\sum_{i=1}^{n-k_n+2} \left(K\left(\frac{t_{i-1}-t}{h'_n}\right) \Delta_{i-1}^n \overline{Y} \right)^2 \right.$
$\left. + 2 \sum_{L=1}^{L_n} \omega\left(\frac{L}{L_n}\right) \sum_{i=1}^{n-k_n-L+2} K\left(\frac{t_{i-1}-t}{h'_n}\right) K\left(\frac{t_{i+L-1}-t}{h'_n}\right) \Delta_{i-1}^n \overline{Y} \Delta_{i-1+L}^n \overline{Y} \right],$

where $\omega(\cdot)$ is a smooth kernel function, in this case the Parzen kernel. L_n is the lag length for adjusting for auto-correlation and $K(\cdot)$ is a kernel weighting function, which in this case is the left-sided exponential kernel.

Usage

```
driftBursts(
  pData,
  testTimes = seq(34260, 57600, 60),
  preAverage = 5,
```

```
  ACLag = -1L,
  meanBandwidth = 300L,
  varianceBandwidth = 900L,
  parallelize = FALSE,
  nCores = NA,
  warnings = TRUE
)
```

Arguments

pData	Either a `data.table` or an `xts` object. If pData is a data.table, columns DT and PRICE must be present, containing timestamps of the trades and the price of the trades (in levels) respectively. If pData is an `xts` object and the number of columns is greater than one, PRICE must be present.
testTimes	A `numeric` containing the times at which to calculate the tests. The standard of `seq(34260, 57600, 60)` denotes calculating the test-statistic once per minute, i.e. 390 times for a typical 6.5 hour trading day from 9:31:00 to 16:00:00. See details. Additionally, `testTimes` can be set to 'all' where the test statistic will be calculated on each tick more than 5 seconds after opening
preAverage	A positive `integer` denoting the length of pre-averaging window for the log-prices. Default is `5`
ACLag	A positive `integer` greater than 1 denoting how many lags are to be used for the HAC estimator of the variance - the default of `-1` denotes using an automatic lag selection algorithm for each iteration. Default is `-1L`
meanBandwidth	An `integer` denoting the bandwidth for the left-sided exponential kernel for the mean. Default is `300L`
varianceBandwidth	An `integer` denoting the bandwidth for the left-sided exponential kernel for the variance. Default is `900L`
parallelize	A `logical` to determine whether to parallelize the underlying C++ code (Using OpenMP). Default is `FALSE`. Note that the parallelized code is not interruptable, while the non-parallel code is interruptable and it's checked every 100 iterations.
nCores	An `integer` denoting the number of cores to use for calculating the code when parallelized. If this argument is not provided, sequential evaluation will be used even though `parallelize` is TRUE. Default is `NA`
warnings	A `logical` denoting whether warnings should be shown. Default is `TRUE`

Details

If the `testTimes` vector contains instructions to test before the first trade, or more than 15 minutes after the last trade, these entries will be deleted, as not doing so may cause crashes. The test statistic is unstable before `max(meanBandwidth , varianceBandwidth)` seconds has passed. The lags from the Newey-West algorithm is increased by `2 * (preAveage-1)` due to the pre-averaging we know at least this many lags should be corrected for. The maximum of 20 lags is also increased by this factor for the same reason.

Value

An object of class DBH and `list` containing the series of the drift burst hypothesis test-statistic as well as the estimated spot drift and variance series. The list also contains some information such as the variance and mean bandwidths along with the pre-averaging setting and the amount of observations. Additionally, the list will contain information on whether testing happened for all `testTimes` entries. Objects of class DBH has the methods `print.DBH`, `plot.DBH`, and `getCriticalValues.DBH` which prints, plots, and retrieves critical values for the test described in appendix B of Christensen, Oomen, and Reno (2020).

Author(s)

Emil Sjoerup

References

Christensen, K., Oomen, R., and Reno, R. (2020) The drift burst hypothesis. Journal of Econometrics. Forthcoming.

Examples

```
# Usage with data.table object
dat <- sampleTData[as.Date(DT) == "2018-01-02"]
# Testing every 60 seconds after 09:45:00
DBH1 <- driftBursts(dat, testTimes = seq(35100, 57600, 60), preAverage = 2, ACLag = -1L,
                    meanBandwidth = 300L, varianceBandwidth = 900L)
print(DBH1)

plot(DBH1, pData = dat)
# Usage with xts object (1 column)
library("xts")
dat <- xts(sampleTData[as.Date(DT) == "2018-01-03"]$PRICE,
           order.by = sampleTData[as.Date(DT) == "2018-01-03"]$DT)
# Testing every 60 seconds after 09:45:00
DBH2 <- driftBursts(dat, testTimes = seq(35100, 57600, 60), preAverage = 2, ACLag = -1L,
                    meanBandwidth = 300L, varianceBandwidth = 900L)
plot(DBH2, pData = dat)

## Not run:
# This block takes some time
dat <- xts(sampleTDataEurope$PRICE,
           order.by = sampleTDataEurope$DT)
# Testing every 60 seconds after 09:00:00
system.time({DBH4 <- driftBursts(dat, testTimes = seq(32400 + 900, 63000, 60), preAverage = 2,
             ACLag = -1L, meanBandwidth = 300L, varianceBandwidth = 900L)})

system.time({DBH4 <- driftBursts(dat, testTimes = seq(32400 + 900, 63000, 60), preAverage = 2,
                             ACLag = -1L, meanBandwidth = 300L, varianceBandwidth = 900L,
                                 parallelize = TRUE, nCores = 8)})
plot(DBH4, pData = dat)

# The print method for DBH objects takes an argument alpha that determines the confidence level
```

```
# of the test performed
print(DBH4, alpha = 0.99)
# Additionally, criticalValue can be passed directly
print(DBH4, criticalValue = 3)
max(abs(DBH4$tStat)) > getCriticalValues(DBH4, 0.99)$quantile

## End(Not run)
```

`exchangeHoursOnly` *Extract data from an* `xts` *object for the exchange hours only*

Description

Filter raw trade data such and return only data between market close and market open. By default, `marketOpen = "09:30:00"` and `marketClose = "16:00:00"` (see Brownlees and Gallo (2006) for more information on good choices for these arguments).

Usage

```
exchangeHoursOnly(
  data,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL
)
```

Arguments

`data`	a `data.table` or `xts` object containing the time series data. Multiple days of input are allowed.
`marketOpen`	character in the format of `"HH:MM:SS"`, specifying the opening time of the exchange(s).
`marketClose`	character in the format of `"HH:MM:SS"`, specifying the closing time of the exchange(s).
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`

Value

`xts` or `data.table` object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Brownlees, C. T. and Gallo, G. M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. Computational Statistics & Data Analysis, 51, pages 2232-2245.

Examples

```
exchangeHoursOnly(sampleTDataRaw)
```

---

gatherPrices *Make TAQ format*

Description

Convenience function to gather data from one `xts` or `data.table` with at least `"DT"`, and d columns containing price data to a `"DT"`, `"SYMBOL"`, and `"PRICE"` column. This function the opposite of `spreadPrices`.

Usage

```
gatherPrices(data)
```

Arguments

`data`	An `xts` or a `data.table` object with at least `"DT"` and d columns with price data with their names corresponding to the respective symbols.

Value

a `data.table` with columns `DT`, `SYMBOL`, and `PRICE`

Author(s)

Emil Sjoerup

See Also

`spreadPrices`

Examples

```
## Not run:
library(data.table)
data1 <- copy(sampleTData)[,  `:=`(PRICE = PRICE * runif(.N, min = 0.99, max = 1.01),
                                   DT = DT + runif(.N, 0.01, 0.02))]
data2 <- copy(sampleTData)[, SYMBOL := 'XYZ']
dat1 <- rbind(data1[, list(DT, SYMBOL, PRICE)], data2[, list(DT, SYMBOL, PRICE)])
setkeyv(dat1, c("DT", "SYMBOL"))
dat1
dat <- spreadPrices(dat1) # Easy to use for realized measures
```

```
dat
dat <- gatherPrices(dat)
dat
all.equal(dat1, dat) # We have changed to RM format and back.

## End(Not run)
```

`getAlphaVantageData` *Get high frequency data from Alpha Vantage*

Description

Function to retrieve high frequency data from Alpha Vantage - wrapper around quantmod's getSymbols.av function

Usage

```
getAlphaVantageData(
  symbols = NULL,
  interval = "5min",
  outputType = "xts",
  apiKey = NULL,
  doSleep = TRUE
)
```

Arguments

`symbols`	character vector with the symbols to import.
`interval`	the sampling interval of the data retrieved. Should be one of one of "1min", "5min", "15min", "30min", or "60min"
`outputType`	string either `"xts"` or `"DT"` to denote the type of output wanted. `"xts"` will yield an xts object, `"DT"` will yield a data.table object.
`apiKey`	string with the api key provided by Alpha Vantage.
`doSleep`	logical when the length of symbols > 5 the function will sleep for 12 seconds by default.

Details

The `doSleep` argument is set to true as default because Alpha Vantage has a limit of five calls per minute. The function does not try to extract when the last API call was made which means that if you made successive calls to get 3 symbols in rapid succession, the function may not retrieve all the data.

Value

An object of type `xts` or `data.table` in case the length of symbols is 1. If the length of symbols > 1 the `xts` and `data.table` objects will be put into a list.

Author(s)

Emil Sjoerup (wrapper only) Paul Teetor (for quantmod's getSymbols.av)

See Also

The getSymbols.av function in the quantmod package

Examples

```
## Not run:
# Get data for SPY at an interval of 1 minute in the standard xts format.
data <- getAlphaVantageData(symbols = "SPY", apiKey = "yourKey", interval = "1min")

# Get data for 3M and Goldman Sachs  at a 5 minute interval in the data.table format.
# The data.tables will be put in a list.
data <- getAlphaVantageData(symbols = c("MMM", "GS"), interval = "5min",
                  outputType = "DT", apiKey = 'yourKey')

# Get data for JPM and Citicorp at a 15 minute interval in the xts format.
# The xts objects will be put in a list.
data <- getAlphaVantageData(symbols = c("JPM", "C"), interval = "15min",
                  outputType = "xts", apiKey = "yourKey")

## End(Not run)
```

---

getCriticalValues   *Get critical value for the drift burst hypothesis t-statistic*

Description

Method for DBH objects to calculate the critical value for the presence of a burst of drift. The critical value is that of the test described in appendix B in Christensen Oomen Reno

Usage

```
getCriticalValues(x, alpha = 0.95)
```

Arguments

x	object of class DBH
alpha	numeric denoting the confidence level for the critical value. Possible values are c(0.9 0.95 0.99 0.995 0.999 0.9999)

Author(s)

Emil Sjoerup

References

Christensen, K., Oomen, R., and Reno, R. (2020) The drift burst hypothesis. Journal of Econometrics. Forthcoming.

`getLiquidityMeasures` *Compute Liquidity Measure*

Description

Function returns an `xts` or `data.table` object containing 23 liquidity measures. Please see details below.

Note that this assumes a regular time grid.

Usage

```
getLiquidityMeasures(tqData, win = 300)
```

Arguments

`tqData`	A `data.table` or `xts` object as in the **highfrequency** merged trades and quotes data.
`win`	A windows length for the forward-prices used for 'realized' spread

Details

NOTE: `xts` or `data.table` should only contain one day of observations Some markets have publish information about whether it was a buyer or a seller who initiated the trade. This information can be passed in a column `DIRECTION` this column must only have 1 or -1 as values.

The respective liquidity measures are defined as follows:

- effectiveSpread

$$\text{effective spread}_t = 2 * D_t * (\text{PRICE}_t - \frac{(\text{BID}_t + \text{OFR}_t)}{2}),$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)). Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- realizedSpread: realized spread

$$\text{realized spread}_t = 2 * D_t * (\text{PRICE}_t - \frac{(\text{BID}_{t+300} + \text{OFR}_{t+300})}{2}),$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)). Note that the time indication of BID and OFR refers to the registered time of the quote in seconds.

- valueTrade: trade value

$$\text{trade value}_t = \text{SIZE}_t * \text{PRICE}_t.$$

- signedValueTrade: signed trade value

$$\text{signed trade value}_t = D_t * (\text{SIZE}_t * \text{PRICE}_t),$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)).

- depthImbalanceDifference: depth imbalance (as a difference)

$$\text{depth imbalance (as difference)}_t = \frac{D_t * (\text{OFRSIZ}_t - \text{BIDSIZ}_t)}{(\text{OFRSIZ}_t + \text{BIDSIZ}_t)},$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)). Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- depthImbalanceRatio: depth imbalance (as ratio)

$$\text{depth imbalance (as ratio)}_t = (\frac{\text{OFRSIZ}_t}{\text{BIDSIZ}_t})^{D_t},$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)). Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- proportionalEffectiveSpread: proportional effective spread

$$\text{proportional effective spread}_t = \frac{\text{effective spread}_t}{(\text{OFR}_t + \text{BID}_t)/2}$$

  (Venkataraman, 2001).
  Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- proportionalRealizedSpread: proportional realized spread

$$\text{proportional realized spread}_t = \frac{\text{realized spread}_t}{(\text{OFR}_t + \text{BID}_t)/2}$$

  (Venkataraman, 2001).
  Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered

- priceImpact: price impact

$$\text{price impact}_t = \frac{\text{effective spread}_t - \text{realized spread}_t}{2}$$

  (see Boehmer (2005), Bessembinder (2003)).

- proportionalPriceImpact: proportional price impact

$$\text{proportional price impact}_t = \frac{\frac{(\text{effective spread}_t - \text{realized spread}_t)}{2}}{\frac{\text{OFR}_t + \text{BID}_t}{2}}$$

  (Venkataraman, 2001). Note that the input of this function consists of the matched trades and quotes, so this is where the time indication refers to (and thus not to the registered quote timestamp).

- halfTradedSpread: half traded spread

$$\text{half traded spread}_t = D_t * (\text{PRICE}_t - \frac{(\text{BID}_t + \text{OFR}_t)}{2}),$$

  where D_t is 1 (-1) if $trade_t$ was buy (sell) (see Boehmer (2005), Bessembinder (2003)). Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- proportionalHalfTradedSpread: proportional half traded spread

$$\text{proportional half traded spread}_t = \frac{\text{half traded spread}_t}{\frac{\text{OFR}_t + \text{BID}_t}{2}}.$$

  Note that the input of this function consists of the matched trades and quotes, so this is were the time indication refers to (and thus not to the registered quote timestamp).

- squaredLogReturn: squared log return on trade prices

$$\text{squared log return on Trade prices}_t = (\log(\text{PRICE}_t) - \log(\text{PRICE}_{t-1}))^2.$$

- absLogReturn: absolute log return on trade prices

$$\text{absolute log return on Trade prices}_t = |\log(\text{PRICE}_t) - \log(\text{PRICE}_{t-1})|.$$

- quotedSpread: quoted spread

$$\text{quoted spread}_t = \text{OFR}_t - \text{BID}_t$$

  Note that the input of this function consists of the matched trades and quotes, so this is where the time indication refers to (and thus not to the registered quote timestamp).

- proportionalQuotedSpread: proportional quoted spread

$$\text{proportional quoted spread}_t = \frac{\text{quoted spread}_t}{\frac{\text{OFR}_t + \text{BID}_t}{2}}$$

  (Venkataraman, 2001). Note that the input of this function consists of the matched trades and quotes, so this is where the time indication refers to (and thus not to the registered quote timestamp).

- logQuotedSpread: log quoted spread

$$\text{log quoted spread}_t = \log(\frac{\text{OFR}_t}{\text{BID}_t})$$

  (Hasbrouck and Seppi, 2001). Note that the input of this function consists of the matched trades and quotes, so this is where the time indication refers to (and thus not to the registered quote timestamp).

- logQuotedSize: log quoted size

$$\text{log quoted size}_t = \log(\text{OFRSIZ}_t) + \log(\text{BIDSIZ}_t)$$

  (Hasbrouck and Seppi, 2001). Note that the input of this function consists of the matched trades and quotes, so this is where the time indication refers to (and thus not to the registered quote timestamp).

- quotedSlope: quoted slope

$$\text{quoted slope}_t = \frac{\text{quoted spread}_t}{\text{log quoted size}_t}$$

(Hasbrouck and Seppi, 2001).

- logQSlope: log quoted slope

$$\text{log quoted slope}_t = \frac{\text{log quoted spread}_t}{\text{log quoted size}_t}.$$

- midQuoteSquaredReturn: midquote squared return

$$\text{midquote squared return}_t = (\log(\text{midquote}_t) - \log(\text{midquote}_{t-1}))^2,$$

where $\text{midquote}_t = \frac{\text{BID}_t + \text{OFR}_t}{2}$.

- midQuoteAbsReturn: midquote absolute return

$$\text{midquote absolute return}_t = |\log(\text{midquote}_t) - \log(\text{midquote}_{t-1})|,$$

where $\text{midquote}_t = \frac{\text{BID}_t + \text{OFR}_t}{2}$.

- signedTradeSize: signed trade size

$$\text{signed trade size}_t = D_t * \text{SIZE}_t,$$

where D_t is 1 (-1) if $trade_t$ was buy (sell).

Value

A modified (enlarged) `xts` or `data.table` with the new measures.

References

Bessembinder, H. (2003). Issues in assessing trade execution costs. *Journal of Financial Markets*, 223-257.

Boehmer, E. (2005). Dimensions of execution quality: Recent evidence for US equity markets. *Journal of Financial Economics*, 78, 553-582.

Hasbrouck, J. and Seppi, D. J. (2001). Common factors in prices, order flows and liquidity. *Journal of Financial Economics*, 59, 383-411.

Venkataraman, K. (2001). Automated versus floor trading: An analysis of execution costs on the Paris and New York exchanges. *The Journal of Finance*, 56, 1445-1485.

Examples

```
tqData <- matchTradesQuotes(sampleTData[as.Date(DT) == "2018-01-02"],
                            sampleQData[as.Date(DT) == "2018-01-02"])
res <- getLiquidityMeasures(tqData)
res
```

`getTradeDirection` *Get trade direction*

Description

Function returns a vector with the inferred trade direction which is determined using the Lee and Ready algorithm (Lee and Ready, 1991).

Usage

```
getTradeDirection(tqData)
```

Arguments

`tqData`	`data.table` or `xts` object, containing joined trades and quotes (e.g. using `matchTradesQuotes`)

Details

NOTE: By convention the first observation is always marked as a buy.

Value

A vector which has values 1 or (-1) if the inferred trade direction is buy or sell respectively.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup. Special thanks to Dirk Eddelbuettel.

References

Lee, C. M. C. and Ready, M. J. (1991). Inferring trade direction from intraday data. *Journal of Finance*, 46, 733-746.

Examples

```
# Generate matched trades and quote data set
tqData <- matchTradesQuotes(sampleTData[as.Date(DT) == "2018-01-02"],
                            sampleQData[as.Date(DT) == "2018-01-02"])
directions <- getTradeDirection(tqData)
head(directions)
```

HARmodel *Heterogeneous autoregressive (HAR) model for realized volatility model estimation*

Description

Function returns the estimates for the heterogeneous autoregressive model (HAR) for realized volatility discussed in Andersen et al. (2007) and Corsi (2009). This model is mainly used to forecast the next day's volatility based on the high-frequency returns of the past.

Usage

```
HARmodel(
  data,
  periods = c(1, 5, 22),
  periodsJ = c(1, 5, 22),
  periodsQ = c(1),
  leverage = NULL,
  RVest = c("rCov", "rBPCov", "rQuar"),
  type = "HAR",
  inputType = "RM",
  jumpTest = "ABDJumptest",
  alpha = 0.05,
  h = 1,
  transform = NULL,
  externalRegressor = NULL,
  periodsExternal = c(1),
  ...
)
```

Arguments

`data` an `xts` object containing either: intraday (log-)returns or realized measures already computed from such returns. In case more than one realized measure is needed, the object should have the as many columns as realized measures needed. The first column should always be the realized variance proxy. In case type is either `"HARQJ"` or `"CHARQ"` the order should be `"RV"`, `"BPV"`, `"RQ"`, or the relevant proxies.

`periods` a vector of integers indicating over how days the realized measures in the model should be aggregated. By default `periods = c(1,5,22)`, which corresponds to one day, one week and one month respectively. This default is in line with Andersen et al. (2007).

`periodsJ` a vector of integers indicating over what time periods the jump components in the model should be aggregated. By default `periodsJ = c(1,5,22)`, which corresponds to one day, one week and one month respectively.

periodsQ
: a vector of integers indicating over what time periods the realized quarticity in the model should be aggregated. By default `periodsQ = c(1,5,22)`, which corresponds to one day, one week and one month respectively.

leverage
: a vector of integers indicating over what periods the negative returns should be aggregated. See Corsi and Reno (2012) for more information. By default `leverage = NULL` and the model assumes the absence of a leverage effect. Set `leverage = c(1,5,22)` to mimic the analysis in Corsi and Reno (2012).

RVest
: a character vector with one, two, or three elements. The first element always refers to the name of the function to estimate the daily integrated variance (non-jump-robust). The second and third element depends on which type of model is estimated: If `type = "HARJ"`, `type = "HARCJ"`, `type = "HARQJ"` the second element refers to the name of the function to estimate the continuous component of daily volatility (jump robust). If type = "HARQ", the second element refers to the name of the function used to estimate the integrated quarticity. If `type = "HARQJ"` the third element always refers to the name of the function used to estimate integrated quarticity. By default `RVest = c("rCov","rBPCov","rQuar")`, i.e. using the realized volatility, realized bi-power variance, and realized quarticity.

type
: a string referring to the type of HAR model you would like to estimate. By default `type = "HAR"`, the most basic model. Other valid options for type are `"HARJ"`, `"HARCJ"`, `"HARQ"`, `"HARQJ"`, `"CHAR"`, or `"CHARQ"`.

inputType
: a string denoting if the input data consists of realized measures or high-frequency returns. Default "RM" is the only way to denote realized measures and everything else denotes returns.

jumpTest
: the function name of a function used to test whether the test statistic which determines whether the jump variability is significant that day. By default `jumpTest = "ABDJumptest"`, hence using the test statistic in Equation or Equation (18) of Andersen et al. (2007). It is also possible to provide pre-computed test statistics for jump tests by setting `jumpTest` to `"testStat"`. These test statistics should still be passed as the third column.

alpha
: a real indicating the confidence level used in testing for jumps. By default `alpha = 0.05`.

h
: an integer indicating the number over how many days the dependent variable should be aggregated. By default, `h = 1`, i.e. no aggregation takes place, you just model the daily realized volatility.

transform
: optionally a string referring to a function that transforms both the dependent and explanatory variables in the model. By default `transform = NULL`, so no transformation is done. Typical other choices in this context would be "log" or "sqrt".

externalRegressor
: an `xts` object of same number of rows as `data`, and one column. This is used as an external regressor. Default is `NULL`.

periodsExternal
: a vector of integers indicating over how days `externalRegressor` should be aggregated.

...
: extra arguments for jump test.

Details

The basic specification in Corsi (2009) is as follows. Let RV_t be the realized variances at day t and $RV_{t-k:t}$ the average realized variance in between $t-k$ and t, $k \geq 0$.

The dynamics of the model are given by

$$RV_{t+1} = \beta_0 + \beta_1 \, RV_t + \beta_2 \, RV_{t-4:t} + \beta_3 \, RV_{t-21:t} + \varepsilon_{t+1}$$

which is estimated by ordinary least squares under the assumption that at time t, the conditional mean of ε_{t+1} is equal to zero.

For other specifications, please refer to the cited papers.

The standard errors reporting in the `print` and `summary` methods are Newey-West standard errors calculated with the **sandwich** package.

Value

The function outputs an object of class `HARmodel` and `lm` (so `HARmodel` is a subclass of `lm`). Objects of class `HARmodel` has the following methods `plot.HARmodel`, `predict.HARmodel`, `print.HARmodel`, and `summary.HARmodel`.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Andersen, T. G., Bollerslev, T., and Diebold, F. (2007). Roughing it up: Including jump components in the measurement, modelling and forecasting of return volatility. *The Review of Economics and Statistics*, 89, 701-720.

Corsi, F. (2009). A simple approximate long memory model of realized volatility. *Journal of Financial Econometrics*, 7, 174-196.

Corsi, F. and Reno R. (2012). Discrete-time volatility forecasting with persistent leverage effect and the link with continuous-time volatility modeling. *Journal of Business & Economic Statistics*, 30, 368-380.

Bollerslev, T., Patton, A., and Quaedvlieg, R. (2016). Exploiting the errors: A simple approach for improved volatility forecasting, *Journal of Econometrics*, 192, 1-18.

Examples

```
# Example 1: HAR
# Forecasting daily Realized volatility for the S&P 500 using the basic HARmodel: HAR
library(xts)
RVSPY <- as.xts(SPYRM$RV5, order.by = SPYRM$DT)

x <- HARmodel(data = RVSPY , periods = c(1,5,22), RVest = c("rCov"),
              type = "HAR", h = 1, transform = NULL, inputType = "RM")
class(x)
x
summary(x)
```

```
plot(x)
predict(x)


# Example 2: HARQ
# Get the highfrequency returns
dat <- as.xts(sampleOneMinuteData[, makeReturns(STOCK), by = list(DATE = as.Date(DT))])
x <- HARmodel(dat, periods = c(1,5,10), periodsJ = c(1,5,10),
            periodsQ = c(1), RVest = c("rCov", "rQuar"),
              type="HARQ", inputType = "returns")
# Estimate the HAR model of type HARQ
class(x)
x
# plot(x)
# predict(x)

# Example 3: HARQJ with already computed realized measures
dat <- SPYRM[, list(DT, RV5, BPV5, RQ5)]
x <- HARmodel(as.xts(dat), periods = c(1,5,22), periodsJ = c(1),
              periodsQ = c(1), type = "HARQJ")
# Estimate the HAR model of type HARQJ
class(x)
x
# plot(x)
predict(x)

# Example 4: CHAR with already computed realized measures
dat <- SPYRM[, list(DT, RV5, BPV5)]

x <- HARmodel(as.xts(dat), periods = c(1, 5, 22), type = "CHAR")
# Estimate the HAR model of type CHAR
class(x)
x
# plot(x)
predict(x)

# Example 5: CHARQ with already computed realized measures
dat <- SPYRM[, list(DT, RV5, BPV5, RQ5)]

x <- HARmodel(as.xts(dat), periods = c(1,5,22), periodsQ = c(1), type = "CHARQ")
# Estimate the HAR model of type CHARQ
class(x)
x
# plot(x)
predict(x)

# Example 6: HARCJ with pre-computed test-statistics
## BNSJumptest manually calculated.
testStats <- sqrt(390) * (SPYRM$RV1 - SPYRM$BPV1)/sqrt((pi^2/4+pi-3 - 2) * SPYRM$medRQ1)
model <- HARmodel(cbind(as.xts(SPYRM[, list(DT, RV5, BPV5)]), testStats), type = "HARCJ")
```

HEAVYmodel *HEAVY model estimation*

Description

This function calculates the High frEquency bAsed VolatilitY (HEAVY) model proposed in Shephard and Sheppard (2010).

Usage

```
HEAVYmodel(data, startingValues = NULL)
```

Arguments

data	an xts object where the first column is a vector of returns and the second column is a vector of realized stock market variation
startingValues	a vector of alternative starting values: first three arguments for variance equation and last three arguments for measurement equation.

Details

Let r_t and RM_t be series of demeaned returns and realized measures of daily stock price variation. The HEAVY model is a two-component model. We assume $r_t = h_t^{1/2} Z_t$ where Z_t is an i.i.d. zero-mean and unit-variance innovation term. The dynamics of the HEAVY model are given by

$$h_t = \omega + \alpha RM_{t-1} + \beta h_{t-1}$$

and

$$\mu_t = \omega_R + \alpha_R RM_{t-1} + \beta_R \mu_{t-1}.$$

The two equations are estimated separately as mentioned in Shephard and Sheppard (2010). We report robust standard errors based on the matrix-product of inverted Hessians and the outer product of gradients.

Note that we always demean the returns in the data input as we don't include a constant in the mean equation.

Value

The function outputs an object of class HEAVYmodel, a list containing

- coefficients = estimated coefficients.
- se = robust standard errors based on inverted Hessian matrix.
- residuals = the residuals in the return equation.
- llh = the two-component log-likelihood values.
- varCondVariances = conditional variances in the variance equation.
- RMCondVariances = conditional variances in the RM equation.

- data = the input data.

The class HEAVYmodel has the following methods: plot.HEAVYmodel, predict.HEAVYmodel, print.HEAVYmodel, and summary.HEAVYmodel.

Author(s)

Onno Kleen and Emil Sjorup.

References

Shephard, N. and Sheppard, K. (2010). Realising the future: Forecasting with high frequency based volatility (HEAVY) models. Journal of Applied Econometrics 25, 197–231.

See Also

predict.HEAVYmodel

Examples

```
# Calculate returns in percentages
logReturns <- 100 * makeReturns(SPYRM$CLOSE)[-1]

# Combine both returns and realized measures into one xts
# Due to return calculation, the first observation is missing
dataSPY <- xts::xts(cbind(logReturns, SPYRM$BPV5[-1] * 10000), order.by = SPYRM$DT[-1])

# Fit the HEAVY model
fittedHEAVY <- HEAVYmodel(dataSPY)

# Examine the estimated coefficients and robust standard errors
fittedHEAVY

# Calculate iterative multi-step-ahead forecasts
predict(fittedHEAVY, stepsAhead = 12)
```

ICov *Estimators of the integrated covariance*

Description

This documentation page functions as a point of reference to quickly look up the estimators of the integrated covariance provided in the **highfrequency** package.

The implemented estimators are:

Realized covariance rCov

Realized bipower covariance rBPCov

Hayashi-Yoshida realized covariance rHYCov

Realized kernel covariance rKernelCov

Realized outlyingness-weighted covariance rOWCov

Realized threshold covariance rThresholdCov

Realized two-scale covariance rTSCov

Robust realized two-scale covariance rRTSCov

Subsampled realized covariance rAVGCov

Realized semi-covariance rSemiCov

Modulated Realized covariance rMRCov

Realized Cholesky covariance rCholCov

Beta-adjusted realized covariance rBACov

See Also

IVar for a list of implemented estimators of the integrated variance.

---

intradayJumpTest *Intraday jump tests*

Description

This function can be used to test for jumps in intraday price paths.

The tests are of the form $L(t) = (R(t) - mu(t))/sigma(t)$.

See spotVol and spotDrift for Estimators for $\sigma(t)$ and $\mu(t)$, respectively.

Usage

```
intradayJumpTest(
  pData,
  volEstimator = "RM",
  driftEstimator = "none",
  alpha = 0.95,
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL,
  n = NULL,
  ...
)
```

Arguments

pData	xts or data.table of the price data in levels. This data can (and should in some cases) be tick-level data. The data can span more than one day. Should only contain a sinlge SYMBOL
volEstimator	character denoting which volatility estimator to use for the tests. See spotVol. Default = "RM" denoting realized measures.
driftEstimator	character denoting which drift estimator to use for the tests. See spotDrift. Default = "none" denoting no drift estimation.
alpha	numeric of length one determining what confidence level to use when constructing the critical values.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set alignPeriod to 5 and alignBy to "minutes". alignPeriod = 5 and alignBy = "minutes".
marketOpen	the market opening time. This should be in the time zone specified by tz. By default, marketOpen = "09:30:00".
marketClose	the market closing time. This should be in the time zone specified by tz. By default, marketClose = "16:00:00".
tz	fallback time zone used in case we we are unable to identify the timezone of the data, by default: tz = NULL. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use tz if specified, and if it is not specified, we use "UTC"
n	number of observation to use in the calculation of the critical values of the test statistic. If this is left as NULL we fall back to the total number of observations in the sample.
...	extra arguments passed on to spotVol for the volatility estimation, and to spotDrift. The null hypothesis of the tests in this function is that there are no jumps in the price series

Author(s)

Emil Sjoerup

References

Christensen, K., Oomen, R. C. A., Podolskij, M. (2014): Fact or Friction: Jumps at ultra high frequency. *Journal of Financial Economics*, 144, 576-599

Examples

```
## Not run:
# We can easily make a Lee-Mykland jump test.
LMtest <- intradayJumpTest(pData = sampleTData[, list(DT, PRICE)],
                           volEstimator = "RM", driftEstimator = "none",
                           RM = "rBPCov", lookBackPeriod = 20,
```

```
                              alignBy = "minutes", alignPeriod = 5, marketOpen = "09:30:00",
                               marketClose = "16:00:00")
plot(LMtest)

# We can just as easily use the pre-averaged version from the "Fact or Friction" paper
FoFtest <- intradayJumpTest(pData = sampleTDatal[, list(DT, PRICE)],
                            volEstimator = "PARM", driftEstimator = "none",
                            RM = "rBPCov", lookBackPeriod = 20, theta = 1.2,
                            marketOpen = "09:30:00", marketClose = "16:00:00")
plot(FoFtest)


## End(Not run)
```

IVar *Estimators of the integrated variance*

Description

This documentation page functions as a point of reference to quickly look up the estimators of the integrated variance provided in the **highfrequency** package.

The implemented estimators are: Realized Variance `rRVar`

Median realized variance `rMedRVar`

Minimum realized variance `rMinRVar`

Realized quadpower variance `rQPVar`

Realized multipower variance `rMPVar`

Realized semivariance `rSVar`

Note that almost all estimators in the list in `ICov` also work yield estimates of the integrated variance on the diagonals.

See Also

`ICov` for a list of implemented estimators of the integrated covariance.

IVinference *Function returns the value, the standard error and the confidence band of the integrated variance (IV) estimator.*

Description

This function supplies information about standard error and confidence band of integrated variance (IV) estimators under Brownian semimartingales model such as: bipower variation, rMinRV, rMedRV. Depending on users' choices of estimator (integrated variance (IVestimator), integrated quarticity (IQestimator)) and confidence level, the function returns the result.(Barndorff (2002)) Function returns three outcomes: 1.value of IV estimator 2.standard error of IV estimator and 3.confidence band of IV estimator.

Assume there is N equispaced returns in period t.

Then the IVinference is given by:

$$\text{standard error} = \frac{1}{\sqrt{N}} * sd$$

$$\text{confidence band} = \hat{IV} \pm cv * se$$

in which,

$$\text{sd} = \sqrt{\theta \times \hat{IQ}}$$

cv : critical value.

se : standard error.

θ : depending on IQestimator, θ can take different value (Andersen et al. (2012)).

$\hat{IQ}$ integrated quarticity estimator.

Usage

```
IVinference(
  rData,
  IVestimator = "RV",
  IQestimator = "rQuar",
  confidence = 0.95,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  ...
)
```

Arguments

`rData`	`xts` object containing all returns in period t for one asset.
`IVestimator`	can be chosen among integrated variance estimators: RV, BV, rMinRV or rMedRV. RV by default.
`IQestimator`	can be chosen among integrated quarticity estimators: rQuar, realized tri-power quarticity (TPQ), quad-power quarticity (QPQ), rMinRQuar or rMedRQuar. TPQ by default.
`confidence`	confidence level set by users. 0.95 by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`

`alignPeriod` positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.

`makeReturns` boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

`...` additional arguments.

Details

The theoretical framework is the logarithmic price process X_t belongs to the class of Brownian semimartingales, which can be written as:

$$\mathrm{X}_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u$$

where a is the drift term, σ denotes the spot vivInferenceolatility process, W is a standard Brownian motion (assume that there are no jumps).

Value

list

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

Barndorff-Nielsen, O. E. (2002). Econometric analysis of realized volatility and its use in estimating stochastic volatility models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64, 253-280.

Examples

```
## Not run:
library("xts") # This function only accepts xts data currently
ivInf <- IVinference(as.xts(sampleTData[, list(DT, PRICE)]), IVestimator= "rMinRV",
                     IQestimator = "rMedRQ", confidence = 0.95, makeReturns = TRUE)
ivInf

## End(Not run)
```

JOjumpTest *Jiang and Oomen (2008) tests for the presence of jumps in the price series.*

Description

This test examines the jump in highfrequency data. It is based on theory of Jiang and Oomen (JO). They found that the difference of simple return and logarithmic return can capture one half of integrated variance if there is no jump in the underlying sample path. The null hypothesis is no jumps.

Function returns three outcomes: 1.z-test value 2.critical value under confidence level of 95% and 3.p-value.

Assume there is N equispaced returns in period t.

Let $r_{t,i}$ be a logarithmic return (with $i = 1, \ldots, N$) in period t.

Let $R_{t,i}$ be a simple return (with $i = 1, \ldots, N$) in period t.

Then the JOjumpTest is given by:

$$\text{JOjumpTest}_{t,N} = \frac{NBV_t}{\sqrt{\Omega_{SwV}}\left(1 - \frac{RV_t}{SwV_t}\right)}$$

in which, BV: bipower variance; RV: realized variance (defined by Andersen et al. (2012));

$$\text{SwV}_t = 2\sum_{i=1}^{N}(R_{t,i} - r_{t,i})$$

$$\Omega_{SwV} = \frac{\mu_6}{9}\frac{N^3\mu_{6/p}^{-p}}{N-p-1}\sum_{i=0}^{N-p}\prod_{k=1}^{p}|r_{t,i+k}|^{6/p}$$

$$\mu_p = \text{E}[|\text{U}|^p] = 2^{p/2}\frac{\Gamma(1/2(p+1))}{\Gamma(1/2)}$$

U: independent standard normal random variables

p: parameter (power).

Usage

```
JOjumpTest(
  pData,
  power = 4,
  alignBy = NULL,
  alignPeriod = NULL,
  alpha = 0.975,
  ...
)
```

Arguments

`pData`	a zoo/xts object containing all prices in period t for one asset.
`power`	can be chosen among 4 or 6. 4 by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to "minutes".
`alpha`	numeric of length one with the significance level to use for the jump test(s). Defaults to 0.975.
`...`	Used internally, do not set.

Details

The theoretical framework underlying jump test is that the logarithmic price process X_t belongs to the class of Brownian semimartingales, which can be written as:

$$\mathrm{X}_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u + Z_t$$

where a is the drift term, σ denotes the spot volatility process, W is a standard Brownian motion and Z is a jump process defined by:

$$\mathrm{Z}_t = \sum_{j=1}^{N_t} k_j$$

where k_j are nonzero random variables. The counting process can be either finite or infinite for finite or infinite activity jumps.

The the Jiang and Ooment test is that in the absence of jumps, the accumulated difference between the simple returns and log returns captures half of the integrated variance. (Theodosiou and Zikes, 2009). If this difference is too great, the null hypothesis of no jumps is rejected.

Value

list

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75- 93.

Jiang, J. G., and Oomen, R. C. A (2008). Testing for jumps when asset prices are observed with noise- a "swap variance" approach. *Journal of Econometrics*, 144, 352-370.

Theodosiou, M., Zikes, F. (2009). A comprehensive comparison of alternative tests for jumps in asset prices. Unpublished manuscript, Graduate School of Business, Imperial College London.

Examples

```
joDT <- JOjumpTest(sampleTData[, list(DT, PRICE)])
```

---

knChooseReMeDI *ReMeDI tuning parameter*

Description

Function to choose the tuning parameter, kn in ReMeDI estimation. The optimal parameter kn is the smallest value that where the criterion:

$$SqErr(k_n)_t^n = \left(\hat{R}_{t,0}^{n,k_n} - \hat{R}_{t,1}^{n,k_n} - \hat{R}_{t,2}^{n,k_n} + \hat{R}_{t,3}^{n,k_n} - \hat{R}_{t,l}^{n,k_n}\right)^2$$

is perceived to be zero. The tuning parameter `tol` can be set to choose the tolerance of the perception of 'close to zero', a higher tolerance will lead to a higher optimal value.

Usage

```
knChooseReMeDI(
  pData,
  knMax = 10,
  tol = 0.05,
  size = 3,
  lower = 2,
  upper = 5,
  plot = FALSE
)
```

Arguments

`pData`	`xts` or `data.table` containing the log-prices of the asset.
`knMax`	max value of `kn` to be considered.
`tol`	tolerance for the minimizing value. If `tol` is high, the algorithm will choose a lower optimal value.
`size`	size of the local window.
`lower`	lower boundary for the method if it fails to find an optimal value. If this is the case, the best kn between lower and upper is returned
`upper`	upper boundary for the method if it fails to find an optimal value. If this is the case, the best kn between lower and upper is returned
`plot`	logical whether to plot the errors.

Details

This is the algorithm B.2 in the appendix of the Li and Linton (2019) working paper.

Value

integer containing the optimal kn

Note

We Thank Merrick Li for contributing his Matlab code for this estimator.

Author(s)

Emil Sjoerup.

References

Li, M. and Linton, O. (2019). A ReMeDI for microstructure noise. Cambridge Working Papers in Economics 1908.

Examples

```
optimalKn <- knChooseReMeDI(sampleTData[as.Date(DT) == "2018-01-02",],
                            knMax = 10, tol = 0.05, size = 3,
                            lower = 2, upper = 5, plot = TRUE)
optimalKn
## Not run:
# We can also have a much larger search-space
optimalKn <- knChooseReMeDI(sampleTDataEurope,
                            knMax = 50, tol = 0.05,
                            size = 3, lower = 2, upper = 5, plot = TRUE)
optimalKn

## End(Not run)
```

---

leadLag *Lead-Lag estimation*

Description

Function that estimates whether one series leads (or lags) another.

Let X_t and Y_t be two observed price over the time interval $[0, 1]$.

For every integer $k \in \mathcal{Z}$, we form the shifted time series

$$Y_{(k+i)/n}, \quad i = 1, 2, \ldots$$

$H = \left(\underline{H}, \overline{H}\right]$ is an interval for $\vartheta \in \Theta$, define the shift interval $H_\vartheta = H + \vartheta = \left(\underline{H} + \vartheta, \overline{H} + \vartheta\right]$ then let

$$X\left(H\right)_t = \int_0^t 1_H\left(s\right) \mathrm{d}X_s$$

Which will be abbreviated:

$$X\left(H\right)=X\left(H\right)_{T+\delta}=\int_{0}^{T+\delta}1_{H}\left(s\right)\mathrm{d}X_{s}$$

Then the shifted HY contrast function is:

$$\tilde{\vartheta}\rightarrow U^{n}\left(\tilde{\vartheta}\right)=1_{\tilde{\vartheta}\geq 0}\sum_{I\in\mathcal{I},J\in\mathcal{J},\overline{I}\leq T}X\left(I\right)Y\left(J\right)1_{\left\{I\cap J_{-\tilde{\vartheta}}\neq\emptyset\right\}}+1_{\tilde{\vartheta}<0}\sum_{I\in\mathcal{I},J\in\mathcal{J},\overline{J}\leq T}X\left(I\right)Y\left(Y\right)1_{\left\{J\cap I_{\tilde{\vartheta}}\neq\emptyset\right\}}$$

This contrast function is then calculated for all the lags passed in the argument `lags`

Usage

```
leadLag(
  price1 = NULL,
  price2 = NULL,
  lags = NULL,
  resolution = "seconds",
  normalize = TRUE,
  parallelize = FALSE,
  nCores = NA
)
```

Arguments

`price1`	`xts` or `data.table` containing prices in levels, in case of data.table, use a column DT to denote the date-time in POSIXct format, and a column PRICE to denote the price
`price2`	`xts` or `data.table` containing prices in levels, in case of data.table, use a column DT to denote the date-time in POSIXct format, and a column PRICE to denote the price
`lags`	a numeric denoting which lags (in units of `resolution`) should be tested as leading or lagging
`resolution`	the resolution at which the lags is measured. The default is "seconds", use this argument to gain 1000 times resolution by setting it to either "ms", "milliseconds", or "milli-seconds".
`normalize`	logical denoting whether the contrasts should be normalized by the product of the L2 norms of both the prices. Default = TRUE. This does not change the value of the lead-lag-ratio.
`parallelize`	logical denoting whether to use a parallelized version of the C++ code (parallelized using OPENMP). Default = FALSE
`nCores`	integer valued numeric denoting how many cores to use for the lead-lag estimation procedure in case parallelize is TRUE. Default is NA, which does not parallelize the code.

Details

The lead-lag-ratio (LLR) can be used to see if one asset leads the other. If LLR < 1, then price1 MAY be leading price2 and vice versa if LLR > 1.

Value

A list with class `leadLag` which contains `contrasts`, `lead-lag-ratio`, and `lags`, denoting the estimated values for each lag calculated, the lead-lag-ratio, and the tested lags respectively.

References

Hoffmann, M., Rosenbaum, M., and Yoshida, N. (2013). Estimation of the lead-lag parameter from non-synchronous data. *Bernoulli*, 19, 1-37.

Examples

```
## Not run:
# Toy example to show the usage
# Spread prices
spread <- spreadPrices(sampleMultiTradeData[SYMBOL %in% c("ETF", "AAA")])
# Use lead-lag estimator
llEmpirical <- leadLag(spread[!is.na(AAA), list(DT, PRICE = AAA)],
                        spread[!is.na(ETF), list(DT, PRICE = ETF)], seq(-15,15))
plot(llEmpirical)

## End(Not run)
```

`listAvailableKernels` *Available kernels*

Description

Returns a vector of the available kernels.

Usage

```
listAvailableKernels()
```

Details

The available kernels are:

- Rectangular: $K(x) = 1$.
- Bartlett: $K(x) = 1 - x$.
- Second-order: $K(x) = 1 - 2x - x^2$.
- Epanechnikov: $K(x) = 1 - x^2$.
- Cubic: $K(x) = 1 - 3x^2 + 2x^3$.
- Fifth: $K(x) = 1 - 10x^3 + 15x^4 - 6x^5$.
- Sixth: $K(x) = 1 - 15x^4 + 24x^5 - 10x^6$
- Seventh: $K(x) = 1 - 21x^5 + 35x^6 - 15x^7$.

- Eighth: $K(x) = 1 - 28x^6 + 48x^7 - 21x^8$.
- Parzen: $K(x) = 1 - 6x^2 + 6x^3$ if $k \leq 0.5$ and $K(x) = 2(1-x)^3$ if $k > 0.5$.
- TukeyHanning: $K(x) = 1 + \sin(\pi/2 - \pi \cdot x))/2$.
- ModifiedTukeyHanning: $K(x) = (1 - \sin(\pi/2 - \pi\ (1-x)^2)/2$.

Value

a character vector.

Author(s)

Scott Payseur.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., and Shephard, N. (2008). Designing realized kernels to measure the ex post variation of equity prices in the presence of noise. *Econometrica*, 76, 1481-1536.

Examples

```
listAvailableKernels
```

`listCholCovEstimators` *Utility function listing the available estimators for the CholCov estimation*

Description

Utility function listing the available estimators for the CholCov estimation

Usage

```
listCholCovEstimators()
```

Value

This function returns a character vector containing the available estimators.

makeOHLCV *Make Open-High-Low-Close-Volume bars*

Description

This function makes OHLC-V bars at arbitrary intervals. If the SIZE column is not present in the input, no volume column is created.

Usage

```
makeOHLCV(pData, alignBy = "minutes", alignPeriod = 5, tz = NULL)
```

Arguments

pData	`data.table` or `xts` object to make the bars out of, containing the intraday price series of possibly multiple stocks for possibly multiple days.
alignBy	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`, and `"ticks"`. To aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
tz	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. With the non-disk functionality, we attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`.

Author(s)

Emil Sjoerup

Examples

```
## Not run:
minuteBars <- makeOHLCV(sampleTDataEurope, alignBy = "minutes", alignPeriod = 1)
# We can use the quantmod package's chartSeries function to plot the ohlcv data
quantmod::chartSeries(minuteBars)

minuteBars <- makeOHLCV(sampleTDataEurope[,], alignBy = "minutes", alignPeriod = 1)
# Again we plot the series with chartSeries
quantmod::chartSeries(minuteBars)

# We can also handle data across multiple days.
fiveMinuteBars <- makeOHLCV(sampleTData)
# Again we plot the series with chartSeries
quantmod::chartSeries(fiveMinuteBars)
```

```
# We can use arbitrary alignPeriod, here we choose pi
bars <- makeOHLCV(sampleTDataEurope, alignBy = "seconds", alignPeriod = pi)
# Again we plot the series with chartSeries
quantmod::chartSeries(bars)

## End(Not run)
```

`makePsd` *Returns the positive semidefinite projection of a symmetric matrix using the eigenvalue method*

Description

Function returns the positive semidefinite projection of a symmetric matrix using the eigenvalue method.

Usage

```
makePsd(S, method = "covariance")
```

Arguments

`S`	a non-PSD matrix.
`method`	character, indicating whether the negative eigenvalues of the correlation or covariance should be replaced by zero. Possible values are "covariance" and "correlation".

Details

We use the eigenvalue method to transform S into a positive semidefinite covariance matrix (see, e.g., Barndorff-Nielsen and Shephard, 2004, and Rousseeuw and Molenberghs, 1993). Let Γ be the orthogonal matrix consisting of the p eigenvectors of S. Denote $\lambda_1^+, \dots, \lambda_p^+$ its p eigenvalues, whereby the negative eigenvalues have been replaced by zeroes. Under this approach, the positive semi-definite projection of S is $S^+ = \Gamma' \mathrm{diag}(\lambda_1^+, \dots, \lambda_p^+)\Gamma$.

If method = "correlation", the eigenvalues of the correlation matrix corresponding to the matrix S are transformed, see Fan et al. (2010).

Value

A matrix containing the positive semi definite matrix.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E. and Shephard, N. (2004). Measuring the impact of jumps in multivariate price processes using bipower covariation. Discussion paper, Nuffield College, Oxford University.

Fan, J., Li, Y., and Yu, K. (2012). Vast volatility matrix estimation using high frequency data for portfolio selection. *Journal of the American Statistical Association*, 107, 412-428

Rousseeuw, P. and Molenberghs, G. (1993). Transformation of non positive semidefinite correlation matrices. *Communications in Statistics - Theory and Methods*, 22, 965-984.

`makeReturns` *Compute log returns*

Description

Convenience function to calculate log-returns, also used extensively internally. Accepts `xts` and `matrix`-like objects. If you use this with a `data.table` object, remember to not pass the `DT` column.

$$\text{log return}_t = (\log(\text{PRICE}_t) - \log(\text{PRICE}_{t-1})).$$

Usage

```
makeReturns(ts)
```

Arguments

`ts`	a possibly multivariate matrix-like object containing prices in levels. If `ts` is an `xts` object, we return an `xts` object. Other types will result in a `matrix`

Details

Note: the first (row of) observation(s) is set to zero.

Value

Depending on input, either a `matrix` or an `xts` object containing the log returns.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup

makeRMFormat — *DEPRECATED use* spreadPrices

Description

DEPRECATED use spreadPrices

Usage

```
makeRMFormat(data)
```

Arguments

data	DEPRECATED

matchTradesQuotes — *Match trade and quote data*

Description

Match the trades and quotes of the input data. All trades are retained and the latest bids and offers are retained, while 'old' quotes are discarded.

Usage

```
matchTradesQuotes(
  tData,
  qData,
  lagQuotes = 0,
  BFM = FALSE,
  backwardsWindow = 3600,
  forwardsWindow = 0.5,
  plot = FALSE,
  ...
)
```

Arguments

tData	`data.table` or xts-object containing the trade data possibly with multiple symbols and over multiple days possible
qData	`data.table` or xts-object containing the quote data possibly with multiple symbols and over multiple days possible
lagQuotes	numeric, number of seconds the quotes are registered faster than the trades (should be round and positive). Default is 0. For older datasets, i.e. before 2010, it may be a good idea to set this to e.g. 2. See Vergote (2005)

`BFM`	a logical determining whether to conduct 'Backwards - Forwards matching' of trades and quotes. The algorithm tries to match trades that fall outside the bid - ask and first tries to match a small window forwards and if this fails, it tries to match backwards in a bigger window. The small window is a tolerance for inaccuracies in the timestamps of bids and asks. The backwards window allow for matching of late reported trades. I.e. block trades.
`backwardsWindow`	a numeric denoting the length of the backwards window used when `BFM = TRUE`. Default is 3600, corresponding to one hour.
`forwardsWindow`	a numeric denoting the length of the forwards window used when `BFM = TRUE`. Default is 0.5, corresponding to one half second.
`plot`	a logical denoting whether to visualize the forwards, backwards, and unmatched trades in a plot.
`...`	used internally. Don't set this parameter

Value

Depending on the input data type, we return either a `data.table` or an `xts` object containing the matched trade and quote data. When using the BFM algorithm, a report of the matched and unmatched trades are also returned (This is omitted when we call this function from the `tradesCleanupUsingQuotes` function).

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Vergote, O. (2005). How to match trades and quotes for NYSE stocks? K.U.Leuven working paper.

Christensen, K., Oomen, R. C. A., Podolskij, M. (2014): Fact or Friction: Jumps at ultra high frequency. *Journal of Financial Economics*, 144, 576-599

Examples

```
# Multi-day input allowed
tqData <- matchTradesQuotes(sampleTData, sampleQData)
# Show output
tqData
```

`mergeQuotesSameTimestamp` *Merge multiple quote entries with the same time stamp*

Description

Merge quote entries that have the same time stamp to a single one and returns an `xts` or a `data.table` object with unique time stamps only.

Usage

```
mergeQuotesSameTimestamp(qData, selection = "median")
```

Arguments

qData	an `xts` object or `data.table` containing the time series data, with at least two columns named `BID` and `OFR` indicating the bid and ask price as well as two columns `BIDSIZ`, `OFRSIZ` indicating the number of round lots available at these prices. For `data.table` an additional column `DT` is necessary that stores the date/time information.
selection	indicates how the bid and ask price for a certain time stamp should be calculated in case of multiple observation for a certain time stamp. By default, `selection = "median"`, and the median price is taken. Alternatively: • `selection = "max.volume"`: use the (bid/ask) price of the entry with largest (bid/ask) volume. • `selection = "weighted.average"`: take the weighted average of all bid (ask) prices, weighted by "BIDSIZ" ("OFRSIZ").

Value

Depending on the input data type, we return either a `data.table` or an `xts` object containing the quote data which has been cleaned.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

---

mergeTradesSameTimestamp

Merge multiple transactions with the same time stamp

Description

Merge trade entries that have the same time stamp to a single one and returns an `xts` or a `data.table` object with unique time stamps only.

Usage

```
mergeTradesSameTimestamp(tData, selection = "median")
```

Arguments

tData	an `xts` object containing the time series data, with one column named `PRICE` indicating the transaction price and one column `SIZE` indicating the number of shares traded.

selection indicates how the price for a certain time stamp should be calculated in case of multiple observation for a certain time stamp. By default, `selection = "median"`, and the median price is taken. Alternatively:

- `selection = "max.volume"`: use the price of the transaction with largest volume.
- `selection = "weighted.average"`: take the weighted average of all prices.

Value

`data.table` or `xts` object depending on input.

Note

previously this function returned the mean of the size of the merged trades (pre version 0.7 and when not using max.volume as the criterion), now it returns the sum.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

---

noZeroPrices *Delete the observations where the price is zero*

Description

Function deletes the observations where the price is zero.

Usage

```
noZeroPrices(tData)
```

Arguments

tData	an `xts` or `data.table` object at least containing a column `PRICE`.

Value

an `xts` or `data.table` object depending on input.

Author(s)

Jonathan Cornelissen and Kris Boudt.

noZeroQuotes — *Delete the observations where the bid or ask is zero*

Description

Function deletes the observations where the bid or ask is zero.

Usage

```
noZeroQuotes(qData)
```

Arguments

`qData`	an `xts` or `data.table` object at least containing the columns `BID` and `OFR`.

Value

`xts` object or `data.table` depending on type of input.

Author(s)

Jonathan Cornelissen and Kris Boudt.

plot.DBH — *Plotting method for `DBH` objects*

Description

Plotting method for `DBH` objects

Usage

```
## S3 method for class 'DBH'
plot(x, ...)
```

Arguments

`x`	an object of class `DBH`
`...`	optional arguments, see details

Details

The plotting method has the following optional parameters:

- `pData` A `data.table` or an `xts` object, containing the prices and timestamps of the data used to calculate the test statistic. If specified, and `which = "tStat"`, the price will be shown on the right y-axis along with the test statistic
- `which` A string denoting which of four plots to make. `"tStat"` denotes plotting the test statistic. `"sigma"` denotes plotting the estimated volatility process. `"mu"` denotes plotting the estimated drift process. If `which = c("sigma", "mu")` or `which = c("mu", "sigma")`, both the drift and volatility processes are plotted. CaPiTAlizAtIOn doesn't matter

Author(s)

Emil Sjoerup

Examples

```
# Testing every 60 seconds after 09:15:00
DBH <- driftBursts(sampleTDataEurope, testTimes = seq(32400 + 900, 63000, 60), preAverage = 2,
                   ACLag = -1L, meanBandwidth = 300L, varianceBandwidth = 900L)
plot(DBH)
plot(DBH, pData = sampleTDataEurope)
plot(DBH, which = "sigma")
plot(DBH, which = "mu")
plot(DBH, which = c("sigma", "mu"))
```

---

plot.HARmodel *Plotting method for HARmodel objects*

Description

Plotting method for HARmodel objects

Usage

```
## S3 method for class 'HARmodel'
plot(x, ...)
```

Arguments

`x`	an object of class `HARmodel`
`...`	extra arguments, see details

Details

The plotting method has the following optional parameter:

- `legend.loc` A string denoting the location of the legend passed on to `addLegend` of the **xts** package

plot.HEAVYmodel — *Plotting method for HEAVYmodel objects*

Description

Plotting method for HEAVYmodel objects

Usage

```
## S3 method for class 'HEAVYmodel'
plot(x, ...)
```

Arguments

`x`	an object of class `HEAVYmodel`.
`...`	extra arguments, see details.

Details

The plotting method has the following optional parameter:

- `legend.loc` A string denoting the location of the legend passed on to `addLegend` of the **xts** package
- `type` A string denoting the type of lot to be made. If `type` is `"condVar"` the fitted values of the conditional variance of the returns is shown. If `type` is different from `"condVar"`, the fitted values of the realized measure is shown. Default is `"condVar"`

plotTQData — *Plot Trade and Quote data*

Description

Plot trade and quote data, trades are marked by crosses, and quotes are plotted as boxes denoting the bid-offer spread for all the quotes.

Usage

```
plotTQData(
  tData,
  qData = NULL,
  xLim = NULL,
  tradeCol = "black",
  quoteCol = "darkgray",
  format = "%H:%M:%S",
  axisCol = "black",
  ...
)
```

Arguments

tData	cleaned trades data
qData	cleaned quotes data
xLim	timestamps for the start and the end of the plots.
tradeCol	color in which to paint the trade crosses.
quoteCol	color in which to fill out the bid-offer spread.
format	format string to pass to `axis.POSIXct` when creating the timestamps on the x axis. If you are plotting a very short time interval, use `"%H:%M:%OS"` to get fractional seconds on the time axis.
axisCol	string to denote which color to use for the x axis
...	passed to `plot` and `points`.

Examples

```
## Not run:
cleanedQuotes = quotesCleanup(qDataRaw = sampleQDataRaw, report = FALSE, printExchange = FALSE)
cleanedTrades <- tradesCleanupUsingQuotes(
      tData = tradesCleanup(tDataRaw = sampleTDataRaw, report = FALSE, printExchange = FALSE),
      qData = quotesCleanup(qDataRaw = sampleQDataRaw, report = FALSE, printExchange = FALSE)
        )[as.Date(DT) == "2018-01-03"]
xLim <- range(as.POSIXct(c("2018-01-03 15:30:00", "2018-01-03 16:00:00"), tz = "EST"))
plotTQData(cleanedTrades, cleanedQuotes, xLim = xLim,
           main = "Raw trade and quote data from NYSE TAQ")

## End(Not run)
```

`predict.HARmodel` *Predict method for objects of type* `HARmodel`

Description

Predict method for objects of type `HARmodel`

Usage

```
## S3 method for class 'HARmodel'
predict(object, ...)
```

Arguments

object	an object of class `HARmodel`
...	extra arguments. See details

Details

The print method has the following optional parameters:

- `newdata` new data to use for forecasting
- `warnings` A logical denoting whether to display warnings, detault is `TRUE`
- `backtransform` A string. If the model is estimated with transformation this parameter can be set to transform the prediction back into variance The possible values are `"simple"` which means inverse of transformation, i.e. `exp` when log-transformation is applied. If using log transformation, the option `"parametric"` can also be used to transform back. The parametric method adds a correction that stems from using the log-transformation

---

`predict.HEAVYmodel` *Iterative multi-step-ahead forecasting for HEAVY models*

Description

Calculates forecasts for h_{T+k}, where T denotes the end of the estimation period for fitting the HEAVYmodel and $k = 1, \ldots,$ `stepsAhead`.

Usage

```
## S3 method for class 'HEAVYmodel'
predict(object, stepsAhead = 10, ...)
```

Arguments

`object`	an object of class HEAVYmodel.
`stepsAhead`	the number of days iterative forecasts are calculated for (default 10).
`...`	further arguments passed to or from other methods.

`print.DBH` *Printing method for* `DBH` *objects*

Description

Printing method for `DBH` objects

Usage

```
## S3 method for class 'DBH'
print(x, ...)
```

Arguments

`x`	an object of class `DBH`
`...`	optional arguments, see details

Details

The print method has the following optional parameters:

- `criticalValue` A numeric denoting a custom critical value of the test.
- `alpha` A numeric denoting the confidence level of the test. The alpha value is passed on to `getCriticalValues`. The default value is 0.95

Author(s)

Emil Sjoerup

Examples

```
## Not run:
DBH <- driftBursts(sampleTDataEurope, testTimes = seq(32400 + 900, 63000, 300), preAverage = 2,
                   ACLag = -1L, meanBandwidth = 300L, varianceBandwidth = 900L)
print(DBH)
print(DBH, criticalValue = 1) # This value doesn't make sense - don't actually use it!
print(DBH, alpha = 0.95) # 5% confidence level - this is the standard
print(DBH, alpha = 0.99) # 1% confidence level

## End(Not run)
```

`print.HARmodel` *Printing method for* `HARmodel` *objects*

Description

Printing method for `HARmodel` objects

Usage

```
## S3 method for class 'HARmodel'
print(x, ...)
```

Arguments

`x`	object of type `HARmodel`
`...`	extra options

Details

The printing method has the extra option `digits` which can be used to set the number of digits for printing pass `lag` to determine the maximum order of the Newey West estimator. Default is `22` quotesCleanup *Cleans quote data*

Description

This is a wrapper function for cleaning the quote data in the entire folder `dataSource`. The result is saved in the folder `dataDestination`.

In case you supply the argument `qDataRaw`, the on-disk functionality is ignored and the function returns the cleaned quotes as `xts` or `data.table` object (see examples).

The following cleaning functions are performed sequentially: `noZeroQuotes`, `exchangeHoursOnly`, `autoSelectExchangeQuotes` or `selectExchange`, `rmNegativeSpread`, `rmLargeSpread` `mergeQuotesSameTimestamp`, `rmOutliersQuotes`.

Usage

```
quotesCleanup(
  dataSource = NULL,
  dataDestination = NULL,
  exchanges = "auto",
  qDataRaw = NULL,
  report = TRUE,
  selection = "median",
  maxi = 50,
  window = 50,
  type = "standard",
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  rmoutliersmaxi = 10,
  printExchange = TRUE,
  saveAsXTS = FALSE,
  tz = NULL
)
```

Arguments

`dataSource` character indicating the folder in which the original data is stored.

`dataDestination` character indicating the folder in which the cleaned data is stored.

`exchanges` vector of stock exchange symbols for all data in dataSource, e.g. `exchanges = c("T","N")` retrieves all stock market data from both NYSE and NASDAQ. The possible exchange symbols are:

- A: AMEX
- N: NYSE
- B: Boston
- P: Arca

- C: NSX
- T/Q: NASDAQ
- D: NASD ADF and TRF
- X: Philadelphia
- I: ISE
- M: Chicago
- W: CBOE
- Z: BATS

. The default value is "auto" which automatically selects the exchange for the stocks and days independently using the autoSelectExchangeQuotes

qDataRaw — xts or data.table object containing raw quote data, possibly for multiple symbols over multiple days. This argument is NULL by default. Enabling it means the arguments dataSource and dataDestination will be ignored. (only advisable for small chunks of data)

report — boolean and TRUE by default. In case it is true and we don't use the on-disk functionality, the function returns (also) a vector indicating how many quotes were deleted by each cleaning step.

selection — argument to be passed on to the cleaning routine mergeQuotesSameTimestamp. The default is "median".

maxi — spreads which are greater than median spreads of the day times maxi are excluded.

window — argument to be passed on to the cleaning routine rmOutliersQuotes.

type — argument to be passed on to the cleaning routine rmOutliersQuotes.

marketOpen — passed to exchangeHoursOnly. A character in the format of "HH:MM:SS", specifying the starting hour, minute and second of an exchange.

marketClose — passed to exchangeHoursOnly. A character in the format of "HH:MM:SS", specifying the closing hour, minute and second of an exchange.

rmoutliersmaxi — argument to be passed on to the cleaning routine rmOutliersQuotes.

printExchange — Argument passed to autoSelectExchangeQuotes indicates whether the chosen exchange is printed on the console, default is TRUE. This is only used when exchanges is "auto"

saveAsXTS — indicates whether data should be saved in xts format instead of data.table when using on-disk functionality. FALSE by default, which means we save as data.table.

tz — fallback time zone used in case we we are unable to identify the timezone of the data, by default: tz = NULL. With the non-disk functionality, we attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use tz if specified, and if it is not specified, we use "UTC". In the on-disk functionality, if tz is not specified, the timezone used will be the system default.

Details

Using the on-disk functionality with .csv.zip files which is the standard from the WRDS database will write temporary files on your machine - we try to clean up after it, but cannot guarantee that there won't be files that slip through the crack if the permission settings on your machine does not match ours.

If the input `data.table` does not contain a `DT` column but it does contain `DATE` and `TIME_M` columns, we create the `DT` column by REFERENCE, altering the `data.table` that may be in the user's environment!

Value

The function converts every (compressed) csv (or rds) file in `dataSource` into multiple `xts` or `data.table` files.

In `dataDestination`, there will be one folder for each symbol containing .rds files with cleaned data stored either in `data.table` or `xts` format.

In case you supply the argument `qDataRaw`, the on-disk functionality is ignored and the function returns a list with the cleaned quotes as an `xts` or `data.table` object depending on input (see examples).

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., and Shephard, N. (2009). Realized kernels in practice: Trades and quotes. Econometrics Journal 12, C1-C32.

Brownlees, C.T. and Gallo, G.M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. Computational Statistics & Data Analysis, 51, pages 2232-2245.

Falkenberry, T.N. (2002). High frequency data filtering. Unpublished technical report.

Examples

```
# Consider you have raw quote data for 1 stock for 2 days
head(sampleQDataRaw)
dim(sampleQDataRaw)
qDataAfterCleaning <- quotesCleanup(qDataRaw = sampleQDataRaw, exchanges = "N")
qDataAfterCleaning$report
dim(qDataAfterCleaning$qData)

# In case you have more data it is advised to use the on-disk functionality
# via "dataSource" and "dataDestination" arguments
```

rankJumpTest *Rank jump test*

Description

Calculate the rank jump test of Li et al. (2019). The procedure tests for the rank of the jump matrix at simultaneous jump events in market returns as well as individual assets.

Usage

```
rankJumpTest(
  marketPrice,
  stockPrices,
  alpha = c(5, 3),
  coarseFreq = 10,
  localWindow = 30,
  rank = 1,
  BoxCox = 1,
  quantiles = c(0.9, 0.95, 0.99),
  nBoot = 1000,
  dontTestAtBoundaries = TRUE,
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL
)
```

Arguments

`marketPrice`	data.table or `xts`containing the market prices in levels
`stockPrices`	list containing the individual stock prices in either data.table or `xts`format. The format should be the the same as `marketPrice`
`alpha`	significance level (in standard deviations) to use for the jump detections. Default is `c(5,3)` for 5 and 3 in the market and stocks respectively.
`coarseFreq`	numeric denoting the coarse sampling frequency. Default is `10`
`localWindow`	numeric denoting the local window for the bootstrap algorithm. Default is `30`
`rank`	rank of the jump matrix under the null hypothesis. Default is `1`
`BoxCox`	numeric of exponents for the Box-Cox transformation, default is `1`
`quantiles`	numeric denoting which quantiles of the bootstrapped critical values to return and compare against. Default is `c(0.9, 0.95, 0.99)`
`nBoot`	numeric denoting how many replications to be used for the bootstrap algorithm. Default is `1000`

`dontTestAtBoundaries`
: logical determining whether to exclude data across different days. Default is `TRUE`

`alignBy`
: character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: "secs", "seconds", "mins", "minutes","hours", and "ticks". To aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.

`alignPeriod`
: positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5 minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.

`marketOpen`
: the market opening time, by default: `marketOpen = "09:30:00"`.

`marketClose`
: the market closing time, by default: `marketClose = "16:00:00"`.

`tz`
: fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`

Details

Let the jump times be defined as:

$$\mathcal{I}_n = \{i : |\Delta_i^n Z| > u_n\}$$

Then the estimated jump matrix is:

$$\hat{\boldsymbol{J}}_n = \left[\Delta_{i,k}^n \boldsymbol{X}\right]_{i \in \mathcal{I}_n}$$

Let $\hat{\lambda}_{n,1}^2 \geq \hat{\lambda}_{n,2}^2 \geq \cdots \geq \hat{\lambda}_{n,d}^2$ be the ordered eigenvalues of $\hat{\boldsymbol{J}}_n \hat{\boldsymbol{J}}_n'$, then test statistic is

$$\hat{S}_{n,t} = \sum_{j=r+1}^{d} \hat{\lambda}_{n,j}^2.$$

The critical values are computed by applying a bootstrapping method

The singular value decomposition of the jump matrix $\hat{\boldsymbol{J}}_n$ is:

$$\hat{\boldsymbol{J}} = \hat{\boldsymbol{U}}_n \hat{\boldsymbol{D}}_n \hat{\boldsymbol{V}}_n'$$

then $\hat{\boldsymbol{U}}_n = \left[\hat{\boldsymbol{U}}_{1n} : \hat{\boldsymbol{U}}_{2n}\right]$ and $\hat{\boldsymbol{V}}_n = \left[\hat{\boldsymbol{V}}_{1n} : \hat{\boldsymbol{V}}_{2n}\right]$

$\boldsymbol{\upsilon}_n = \left(\upsilon_{j,n}\right)_{1 \leq j \leq d}$ such that $\upsilon_{j,n} \asymp \Delta_n^{\varpi} for \varpi \in (0, 1/2)$ which is used to trim jumps. The bootstrapping method is calculated by the following algorithm

- Step 1.
  For each $i \in \mathcal{I}_n$, draw $\kappa_i^\star \sim$ Uniform $[0, 1]$ and draw with equal probability,

  $$\boldsymbol{\xi}_{n,i-}^\star \text{from} \left\{\min\left(\max\left(\Delta_{i-j}^n \boldsymbol{X}, -\boldsymbol{\upsilon}_n\right), \boldsymbol{\upsilon}_n\right) : 1 \leq j \leq k_n\right\},$$

  $$\boldsymbol{\xi}_{n,i+}^\star \text{from} \left\{\min\left(\max\left(\Delta_{i+j}^n \boldsymbol{X}, -\boldsymbol{\upsilon}_n\right), \boldsymbol{\upsilon}_n\right) : 1 \leq j \leq k_n\right\},$$

  and set $\boldsymbol{\zeta}_{n,i}^\star = \sqrt{\kappa_i^\star} \boldsymbol{\xi}_{n,i-}^\star + \sqrt{k - \kappa_i^\star} \boldsymbol{\xi}_{n,i+}^\star$ and $\boldsymbol{\zeta}_n^\star = \left[\boldsymbol{\zeta}_{n,i}^\star\right]_{i \in \mathcal{I}_n}$

- Step 2.

  Repeat 1 for a large number of iterations. Set $cv_{n,\alpha}$ as as the $1-\alpha$ quantile of $\left\|\hat{\boldsymbol{U}}_{2n}'\boldsymbol{\xi}_n^\star\hat{\boldsymbol{V}}_{2n}\right\|^2$ in the simulated sample.

Value

A list containing `criticalValues` which are the bootstrapped critical values, `testStatistic` the test statistic of the jump test, `dimensions` which are the dimensions of the jump matrix `marketJumpDetections` the jumps detected in the market prices, `stockJumpDetections` the co-jumps detected in the individual stock prices, and `jumpIndices` which are the indices of the detected jumps.

Author(s)

Emil Sjoerup, based on Matlab code provided by Li et al. (2019)

References

Li, j., Todorov, V., Tauchen, G., and Lin, H. (2019). Rank Tests at Jump Events. *Journal of Business & Economic Statistics*, 37, 312-321.

`rAVGCov` *Realized covariances via subsample averaging*

Description

Calculates realized variances via averaging across partially overlapping grids, first introduced by Zhang et al. (2005). This estimator is basically an average across different `rCov` estimates that start at different points in time, see details below.

Usage

```
rAVGCov(
  rData,
  cor = FALSE,
  alignBy = "minutes",
  alignPeriod = 5,
  k = 1,
  makeReturns = FALSE,
  ...
)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`cor`	boolean, in case it is `TRUE`, and the input data is multivariate, the correlation is returned instead of the covariance matrix. `FALSE` by default.

alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
k	numeric denoting which horizon to use for the subsambles. This can be a fraction as long as k is a divisor of alignPeriod default is 1.
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally, do not change.

Details

Suppose that in period t, there are N equispaced returns $r_{i,t}$ and let Δ be equal to alignPeriod. For $i \geq \Delta$, we define the subsampled Δ-period return as

$$\tilde{r}_{t,i} = \sum_{k=0}^{\Delta-1} r_{t,i-k}, .$$

Now define $N^*(j) = N/\Delta$ if $j = 0$ and $N^*(j) = N/\Delta - 1$ otherwise. The j-th component of the rAVGCov estimator is given by

$$RV_t^j = \sum_{i=1}^{N^*(j)} \tilde{r}^2_{t,j+i\cdot\Delta}.$$

Now taking the average across the different RV_t^j, $j = 0, \ldots, \Delta - 1$, defines the rAVGCov estimator. The multivariate version follows analogously.

Note that Liu et al. (2015) show that rAVGCov is not only theoretically but also empirically a more reliable estimator than rCov.

Value

in case the input is and contains data from one day, an N by N matrix is returned. If the data is a univariate xts object with multiple days, an xts is returned. If the data is multivariate and contains multiple days (xts or data.table), the function returns a list containing N by N matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Scott Payseur, Onno Kleen, and Emil Sjoerup.

References

Liu, L. Y., Patton, A. J., Sheppard, K. (2015). Does anything beat 5-minute RV? A comparison of realized measures across multiple asset classes. *Journal of Econometrics*, 187, 293-311.

Zhang, L., Mykland, P. A. , and Ait-Sahalia, Y. (2005). A tale of two time scales: Determining integrated volatility with noisy high-frequency data. *Journal of the American Statistical Association*, 100, 1394-1411.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
# Average subsampled realized variance/covariance aligned at one minute returns at
# 5 sub-grids (5 minutes).

# Univariate subsampled realized variance
rvAvgSub <- rAVGCov(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
                    makeReturns = TRUE)
rvAvgSub

# Multivariate subsampled realized variance
rvAvgCovSub <- rAVGCov(rData = sampleOneMinuteData[1:391], makeReturns = TRUE)
rvAvgCovSub
```

---

rBACov *rBACov*

Description

The Beta Adjusted Covariance (BAC) equals the pre-estimator plus a minimal adjustment matrix such that the covariance-implied stock-ETF beta equals a target beta.

The BAC estimator works by applying a minimum correction factor to a pre-estimated covariance matrix such that a target beta derived from the ETF is reached.

Let

$$\bar{\beta}$$

denote the implied beta derived from the pre-estimator, and

$$\beta_{\bullet}$$

denote the target beta, then the correction factor is calculated as:

$$L\left(\bar{\beta} - \beta_{\bullet}\right),$$

where

$$L = \left(I_{d^2} - \frac{1}{2}\mathcal{Q}\right)\bar{W}'\left(I_{d^2}\left(\sum_{k=1}^{d}\frac{\sum_{k=1}^{n_k}\left(w_{t_{m-1}^k}^k\right)^2}{n_k}\right) - \frac{\bar{W}\mathcal{Q}\bar{W}'}{2}\right)^{-1},$$

where d is the number of assets in the ETF, and n_k is the number of trades in the kth asset, and

$$\bar{W}^k = \left(0'_{(k-1)d}, \frac{1}{n_1}\sum_{m=1}^{n_1} w_{t_{m-1}^1}^1, \ldots, \frac{1}{n_d}\sum_{m=1}^{n_d} w_{t_{m-1}^d}^d, 0'_{(d-k)d}\right),$$

where $w_{t_{m-1}^k}^k$ is the weight of the kth asset in the ETF.

and

$$\mathcal{Q}^{(i-1)d+j}$$

is defined by the following two cases:

$\left(0'_{(i-1)d+j-1}, 1, 0'_{(d-i+1)d-j}\right) + \left(0'_{(j-1)d+i-1}, -1, 0'_{(d-j+1)d-i}\right) \quad$ if $i \neq j$;

$0'_{d^2} \quad$ otherwise.

$\bar{W}^k$ has dimensions $d \times d^2$ and $\mathcal{Q}^{(i-1)d+j}$ has dimensions $d^2 \times d^2$.

The Beta-Adjusted Covariance is then

$$\Sigma^{\text{BAC}} = \Sigma - L\left(\bar{\beta} - \beta_\bullet\right),$$

where Σ is the pre-estimated covariance matrix.

Usage

```
rBACov(
  pData,
  shares,
  outstanding,
  nonEquity,
  ETFNAME = "ETF",
  unrestricted = TRUE,
  targetBeta = c("HY", "VAB", "expert"),
  expertBeta = NULL,
  preEstimator = "rCov",
  noiseRobustEstimator = rTSCov,
  noiseCorrection = FALSE,
  returnL = FALSE,
  ...
)
```

Arguments

`pData`	a named list. Each list-item contains an `xts` or `data.table` object with the intraday price data of an ETF and it's component stocks. `xts` objects are turned into `data.tables`
`shares`	a `numeric` with length corresponding to the number of component stocks in the ETF. The entries are the stock holdings of the ETF in the corresponding stock. The order of these entries should correspond to the order the stocks are listed in the `list` passed in the `pData` argument.
`outstanding`	number of shares outstanding of the ETF
`nonEquity`	aggregated value of the additional components (like cash, money-market funds, bonds, etc.) of the ETF which are not included in the components in `pData`.
`ETFNAME`	a `character` denoting which entry in the `pData` list is the ETF. Default is `"ETF"`
`unrestricted`	a `logical` denoting whether to use the unrestricted estimator, which is an extension that also affects the diagonal. Default is `FALSE`

targetBeta	a character, one of c("HY", "VAB", "expert") (default) denoting which target beta to use, only the first entry will be used. A value "HY" means using the Hayashi-Yoshida estimator to estimate the empirical beta. A value of "VAB" denotes using the variance adjusted beta. A value of "expert" denotes using a user-supplied target beta, which can be supplied in the expertBeta argument.
expertBeta	a numeric containing the user supplied expert beta used when targetBeta is "expert". The expertBeta must be of length equal to the number of assets in the ETF. Default is NULL
preEstimator	a function which estimates the integrated covariance matrix. Default is rCov
noiseRobustEstimator	a function which estimates the integrated (co)variance and is robust to microstructure noise (only the diagonal will be estimated). This function is only used when noiseCorrection is TRUE. Default is rTSCov
noiseCorrection	a logical which denotes whether to use the extension of the estimator which corrects for microstructure noise by using the noiseRobustEstimator function. Default is FALSE
returnL	a logical which denotes whether to return the L matrix. Default is FALSE
...	extra arguments passed to preEstimator and noiseRobustEstimator.

Author(s)

Emil Sjoerup, (Kris Boudt and Kirill Dragun for the Python version)

References

Boudt, K., Dragun, K., Omauri, S., and Vanduffel, S. (2021) Beta-Adjusted Covariance Estimation (working paper).

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
## Not run:
# Since we don't have any data in this package that is of the required format we must simulate it.
library(xts)
library(highfrequency)
# The mvtnorm package is needed for this example
# Please install this package before running this example
library("mvtnorm")
# Set the seed for replication
set.seed(123)
iT <- 23400 # Number of observations
# Simulate returns
rets <- rmvnorm(iT * 3 + 1, mean = rep(0,4),
                sigma = matrix(c(0.1, -0.03 , 0.02, 0.04,
                                -0.03, 0.05, -0.03, 0.02,
```

```
                                 0.02, -0.03, 0.05, -0.03,
                                 0.04, 0.02, -0.03, 0.08), ncol = 4))
# We assume that the assets don't trade in a synchronous manner
w1 <- rets[sort(sample(1:nrow(rets), size = nrow(rets) * 0.5)), 1]
w2 <- rets[sort(sample(1:nrow(rets), size = nrow(rets) * 0.75)), 2]
w3 <- rets[sort(sample(1:nrow(rets), size = nrow(rets) * 0.65)), 3]
w4 <- rets[sort(sample(1:nrow(rets), size = nrow(rets) * 0.8)), 4]
w5 <- rnorm(nrow(rets) * 0.9, mean = 0, sd = 0.005)
timestamps1 <- seq(34200, 57600, length.out = length(w1))
timestamps2 <- seq(34200, 57600, length.out = length(w2))
timestamps3 <- seq(34200, 57600, length.out = length(w3))
timestamps4 <- seq(34200, 57600, length.out = length(w4))
timestamps4 <- seq(34200, 57600, length.out = length(w4))
timestamps5 <- seq(34200, 57600, length.out = length(w5))

w1 <- xts(w1 * c(0,sqrt(diff(timestamps1) / (max(timestamps1) - min(timestamps1)))),
          as.POSIXct(timestamps1, origin = "1970-01-01"), tzone = "UTC")
w2 <- xts(w2 * c(0,sqrt(diff(timestamps2) / (max(timestamps2) - min(timestamps2)))),
          as.POSIXct(timestamps2, origin = "1970-01-01"), tzone = "UTC")
w3 <- xts(w3 * c(0,sqrt(diff(timestamps3) / (max(timestamps3) - min(timestamps3)))),
          as.POSIXct(timestamps3, origin = "1970-01-01"), tzone = "UTC")
w4 <- xts(w4 * c(0,sqrt(diff(timestamps4) / (max(timestamps4) - min(timestamps4)))),
          as.POSIXct(timestamps4, origin = "1970-01-01"), tzone = "UTC")
w5 <- xts(w5 * c(0,sqrt(diff(timestamps5) / (max(timestamps5) - min(timestamps5)))),
          as.POSIXct(timestamps5, origin = "1970-01-01"), tzone = "UTC")

p1  <- exp(cumsum(w1))
p2  <- exp(cumsum(w2))
p3  <- exp(cumsum(w3))
p4  <- exp(cumsum(w4))

weights <- runif(4) * 1:4
weights <- weights / sum(weights)
p5 <- xts(rowSums(cbind(w1 * weights[1], w2 * weights[2], w3 * weights[3], w4 * weights[4]),
                   na.rm = TRUE),
                   index(cbind(p1, p2, p3, p4)))
p5 <- xts(cumsum(rowSums(cbind(p5, w5), na.rm = TRUE)), index(cbind(p5, w5)))

p5 <- exp(p5[sort(sample(1:length(p5), size = nrow(rets) * 0.9))])


BAC <- rBACov(pData = list(
                 "ETF" = p5, "STOCK 1" = p1, "STOCK 2" = p2, "STOCK 3" = p3, "STOCK 4" = p4
                   ), shares = 1:4, outstanding = 1, nonEquity = 0, ETFNAME = "ETF",
                   unrestricted = FALSE, preEstimator = "rCov", noiseCorrection = FALSE,
                   returnL = FALSE, K = 2, J = 1)

# Noise robust version of the estimator
noiseRobustBAC <- rBACov(pData = list(
                 "ETF" = p5, "STOCK 1" = p1, "STOCK 2" = p2, "STOCK 3" = p3, "STOCK 4" = p4
                   ), shares = 1:4, outstanding = 1, nonEquity = 0, ETFNAME = "ETF",
                   unrestricted = FALSE, preEstimator = "rCov", noiseCorrection = TRUE,
                   noiseRobustEstimator = rHYCov, returnL = FALSE, K = 2, J = 1)
```

```
# Use the Variance Adjusted Beta method
# Also use a different pre-estimator.
VABBAC <- rBACov(pData = list(
                 "ETF" = p5, "STOCK 1" = p1, "STOCK 2" = p2, "STOCK 3" = p3, "STOCK 4" = p4
                   ), shares = 1:4, outstanding = 1, nonEquity = 0, ETFNAME = "ETF",
                   unrestricted = FALSE, targetBeta = "VAB", preEstimator = "rHYov",
                noiseCorrection = FALSE, returnL = FALSE, Lin = FALSE, L = 0, K = 2, J = 1)


## End(Not run)
```

`rBeta` *Realized beta*

Description

Depending on users' choices of estimator (realized covariance (RCOVestimator) and realized variance (RVestimator)), the function returns the realized beta, defined as the ratio between both.

The realized beta is given by

$$\beta_{jm} = \frac{RCOVestimator_{jm}}{RVestimator_m}$$

in which

$RCOVestimator$: Realized covariance of asset j and market index m.

$RVestimator$: Realized variance of market index m.

Usage

```
rBeta(
  rData,
  rIndex,
  RCOVestimator = "rCov",
  RVestimator = "rRVar",
  makeReturns = FALSE,
  ...
)
```

Arguments

`rData`	a `xts` object containing all returns in period t for one asset.
`rIndex`	a `xts` object containing return in period t for an index.
`RCOVestimator`	can be chosen among realized covariance estimators: `"rCov"`, `"rAVGCov"`, `"rBPCov"`, `"rHYCov"`, `"rKernelCov"`, `"rOWCov"`, `"rRTSCov"`, `"rThresholdCov"` and `"rTSCov"` `"rCov"` by default.

RVestimator — can be chosen among realized variance estimators: "rRVar", "rMinRVar" and "rMedRVar". "rRVar" by default. In case of missing RVestimator, RCOVestimator function applying for rIndex will be used.

makeReturns — boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.

... — arguments passed to RCOVestimator and RVestimator

Details

Suppose there are N equispaced returns on day t for the asset j and the index m. Denote $r_{(j)i,t}$, $r_{(m)i,t}$ as the ith return on day t for asset j and index m (with $i = 1, \ldots, N$).

By default, the RCov is used and the realized beta coefficient is computed as:

$$\hat{\beta}_{(jm)t} = \frac{\sum_{i=1}^{N} r_{(j)i,t} r_{(m)i,t}}{\sum_{i=1}^{N} r_{(m)i,t}^2}.$$

Note: The function does not support to calculate betas across multiple days.

Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E. and Shephard, N. (2004). Econometric analysis of realized covariation: high frequency based covariance, regression, and correlation in financial economics. *Econometrica*, 72, 885-925.

Examples

```
## Not run:
library("xts")
a <- as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, MARKET)])
b <-  as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, STOCK)])
rBeta(a, b, RCOVestimator = "rBPCov", RVestimator = "rMinRVar", makeReturns = TRUE)

## End(Not run)
```

rBPCov *Realized bipower covariance*

Description

Calculate the Realized BiPower Covariance (rBPCov), defined in Barndorff-Nielsen and Shephard (2004).

Let $r_{t,i}$ be an intraday $Nx1$ return vector and $i = 1, ..., M$ the number of intraday returns.

The rBPCov is defined as the process whose value at time t is the N-dimensional square matrix with k, q-th element equal to

$$\text{rBPCov}[k,q]_t = \frac{\pi}{8}\left(\sum_{i=2}^{M}\left|r_{(k)t,i} + r_{(q)t,i}\right| \left|r_{(k)t,i-1} + r_{(q)t,i-1}\right| - \left|r_{(k)t,i} - r_{(q)t,i}\right| \left|r_{(k)t,i-1} - r_{(q)t,i-1}\right|\right),$$

where $r_{(k)t,i}$ is the k-th component of the return vector $r_{i,t}$.

Usage

```
rBPCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  makePsd = FALSE,
  ...
)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days
`cor`	boolean, in case it is `TRUE`, and the input data is multivariate, the correlation is returned instead of the covariance matrix. `FALSE` by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. E.g. to aggregate based on a 5-minute frequency, set `alignPeriod` to 5 and `alignBy` to `"minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.
`makePsd`	boolean, in case it is `TRUE`, the positive definite version of rBPCov is returned. `FALSE` by default.
`...`	used internally, do not change.

Value

in case the input is and contains data from one day, an N by N matrix is returned. If the data is a univariate `xts` object with multiple days, an `xts` is returned. If the data is multivariate and contains multiple days (`xts` or `data.table`), the function returns a list containing N by N matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., and Shephard, N. (2004). Measuring the impact of jumps in multivariate price processes using bipower covariation. Discussion paper, Nuffield College, Oxford University.

See Also

`ICov` for a list of implemented estimators of the integrated covariance.

Examples

```
# Realized Bipower Variance/Covariance for a price series aligned
# at 5 minutes.

# Univariate:
rbpv <- rBPCov(rData = sampleTData[, list(DT, PRICE)], alignBy ="minutes",
               alignPeriod = 5, makeReturns = TRUE)
# Multivariate:
rbpc <- rBPCov(rData = sampleOneMinuteData, makeReturns = TRUE, makePsd = TRUE)
rbpc
```

---

rCholCov — *CholCov estimator*

Description

Positive semi-definite covariance estimation using the CholCov algorithm. The algorithm estimates the integrated covariance matrix by sequentially adding series and using ‘refreshTime‘ to synchronize the observations. This is done in order of liquidity, which means that the algorithm uses more data points than most other estimation techniques.

Usage

```
rCholCov(
  pData,
  IVest = "rMRCov",
  COVest = "rMRCov",
  criterion = "squared duration",
  ...
)
```

Arguments

`pData`	a list. Each list-item i contains an `xts` object with the intraday price data (in levels) of stock i for day t. The order of the data does not matter as it will be sorted according to the criterion specified in the `criterion` argument
`IVest`	integrated variance estimator, default is `"rMRCov"`. For a list of implemented estimators, use `listCholCovEstimators()`.
`COVest`	covariance estimator, default is `"rMRCov"`. For a list of implemented estimators, use `listCholCovEstimators()`.
`criterion`	criterion to use for sorting the data according to liquidity. Possible values are `"squared duration"`, `"duration"`, `"count"`, defaults to `"squared duration"`.
`...`	additional arguments to pass to `IVest` and `COVest`. See details.

Details

Additional arguments for `IVest` and `COVest` should be passed in the ... argument. For the `rMRCov` estimator, which is the default, the `theta` and `delta` parameters can be set. These default to 1 and 0.1 respectively.

The CholCov estimation algorithm is useful for estimating covariances of d series that are sampled asynchronously and with different liquidities. The CholCov estimation algorithm is as follows:

- First sort the series in terms of decreasing liquidity according to a liquidity criterion, such that series 1 is the most liquid, and series d the least.
- Step 1:
  Apply refresh-time on $a = \{1\}$ to obtain the grid τ^a.
  Estimate $\hat{g}_{11}$ using an IV estimator on $f^{(1)}_{\tau^a_j} = \hat{u}^{(1)}_{\tau^a_j}$.
- Step 2:
  Apply refresh-time on $b = \{1, 2\}$ to obtain the grid τ^b.
  Estimate $\hat{h}^b_{21}$ as the realized beta between $f^{(1)}_{\tau^b_j}$ and $\hat{u}^{(2)}_{\tau^b_j}$. Set $\hat{h}_{21} = \hat{h}^b_{21}$.
  Estimate $\hat{g}_{22}$ using an IV estimator on $f^{(2)}_{\tau^b_j} = \hat{u}^{(2)}_{\tau^b_j} - \hat{h}_{21} f^{(1)}_{\tau^b_j}$.
- Step 3:
  Apply refresh-time on $c = \{1, 3\}$ to obtain the grid τ^c.
  Estimate $\hat{h}^c_{31}$ as the realized beta between $f^{(1)}_{\tau^c_j}$ and $\hat{u}^{(3)}_{\tau^c_j}$. Set $\hat{h}_{31} = \hat{h}^c_{31}$.
  Apply refresh-time on $d = \{1, 2, 3\}$ to obtain the grid τ^d.
  Re-estimate $\hat{h}^d_{21}$ at the new grid, such that the projections $f^{(1)}_{\tau^d_j}$ and $f^{(2)}_{\tau^d_j}$ are orthogonal.
  Estimate $\hat{h}^d_{32}$ as the realized beta between $f^{(2)}_{\tau^d_j}$ and $\hat{u}^{(3)}_{\tau^d_j}$. Set $\hat{h}_{32} = \hat{h}^d_{32}$.
  Estimate $\hat{g}_{33}$ using an IV estimator on $f^{(3)}_{\tau^d_j} = \hat{u}^{(3)}_{\tau^d_j} - \hat{h}_{32} f^{(2)}_{\tau^d_j} - \hat{h}_{31} f^{(1)}_{\tau^d_j}$.
- Step 4 to d:
  Continue in the same fashion by sampling over $1, ..., k, l$ to estimate h_{lk} using the smallest possible set.
  Re-estimate the h_{nm} with $m < n \leq k$ at every new grid to obtain orthogonal projections.
  Estimate the g_{kk} as the IV of projections based on the final estimates, $\hat{h}$.

Value

a list containing the covariance matrix "CholCov", and the Cholesky decomposition "L" and "G" such that $\mathtt{L} \times \mathtt{G} \times \mathtt{L}' = \mathtt{CholCov}$.

Author(s)

Emil Sjoerup

References

Boudt, K., Laurent, S., Lunde, A., Quaedvlieg, R., and Sauri, O. (2017). Positive semidefinite integrated covariance estimation, factorizations and asynchronicity. *Journal of Econometrics*, 196, 347-367.

See Also

ICov for a list of implemented estimators of the integrated covariance.

---

rCov *Realized covariance*

Description

Function returns the Realized Covariation (rCov). Let $r_{t,i}$ be an intraday $N \times M$ return vector and $i = 1, ..., M$ the number of intraday returns.

Then, the rCov is given by

$$\mathrm{rCov}_t = \sum_{i=1}^{M} r_{t,i} r'_{t,i}.$$

Usage

```
rCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  ...
)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
cor	boolean, in case it is TRUE, and the input data is multivariate, the correlation is returned instead of the covariance matrix. FALSE by default.

alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally, do not change.

Value

in case the input is and contains data from one day, an $N \times N$ matrix is returned. If the data is a univariate xts object with multiple days, an xts is returned. If the data is multivariate and contains multiple days (xts or data.table), the function returns a list containing N by N matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
# Realized Variance/Covariance for prices aligned at 5 minutes.

# Univariate:
rv = rCov(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
          alignPeriod = 5, makeReturns = TRUE)
rv

# Multivariate:
rc = rCov(rData = sampleOneMinuteData, makeReturns = TRUE)
rc
```

---

refreshTime *Synchronize (multiple) irregular timeseries by refresh time*

Description

This function implements the refresh time synchronization scheme proposed by Harris et al. (1995). It picks the so-called refresh times at which all assets have traded at least once since the last refresh time point. For example, the first refresh time corresponds to the first time at which all stocks have traded. The subsequent refresh time is defined as the first time when all stocks have traded again. This process is repeated until the end of one time series is reached.

Usage

```
refreshTime(pData, sort = FALSE, criterion = "squared duration")
```

Arguments

pData	a list. Each list-item contains an `xts` or a `data.table` object (with first column DT (datetime)) containing the original time series (one day only and typically a price series).
sort	logical determining whether to sort the index based on a criterion (will only sort descending; i.e., most liquid first). Default is `FALSE`.
criterion	character determining which criterion used. Currently supports `"squared duration"` and `"duration"`. Default is `"squared duration"`.

Value

An `xts` or `data.table` object containing the synchronized time series - depending on the input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Harris, F., T. McInish, Shoesmith, G., and Wood, R. (1995). Cointegration, error correction, and price discovery on informationally linked security markets. *Journal of Financial and Quantitative Analysis*, 30, 563-581.

Examples

```
# Suppose irregular timepoints:
start <- as.POSIXct("2010-01-01 09:30:00")
ta <- start + c(1,2,4,5,9)
tb <- start + c(1,3,6,7,8,9,10,11)

# Yielding the following timeseries:
a <- xts::as.xts(1:length(ta), order.by = ta)
b <- xts::as.xts(1:length(tb), order.by = tb)

# Calculate the synchronized timeseries:
refreshTime(list(a,b))
```

ReMeDI *ReMeDI*

Description

This function estimates the auto-covariance of market-microstructure noise

Let the observed price Y_t be given as $Y_t = X_t + \varepsilon_t$, where X_t is the efficient price and ε_t is the market microstructure noise

The estimator of the l'th lag of the market microstructure is defined as:

$$\hat{R}^n_{t,l} = \frac{1}{n_t} \sum_{i=2k_n}^{n_t-k_n-l} \left(Y^n_{i+l} - Y^n_{i+l+k_n}\right) \left(Y^n_i - Y^n_{i-2k_n}\right),$$

where k_n is a tuning parameter. In the function knChooseReMeDI, we provide a function to estimate the optimal k_n parameter.

Usage

```
ReMeDI(pData, kn = 1, lags = 1, makeCorrelation = FALSE)
```

Arguments

`pData`	`xts` or `data.table` containing the log-prices of the asset
`kn`	numeric of length 1 determining the tuning parameter kn this controls the lengths of the non-overlapping interval in the ReMeDI estimation
`lags`	numeric containing integer values indicating the lags for which to estimate the (co)variance
`makeCorrelation`	logical indicating whether to transform the autocovariances into autocorrelations. The estimate of variance is imprecise and thus, constructing the correlation like this may show correlations that fall outside $(-1, 1)$.

Note

We Thank Merrick Li for contributing his Matlab code for this estimator.

Author(s)

Emil Sjoerup.

References

Li, M. and Linton, O. (2021). A ReMeDI for microstructure noise. Econometrica, forthcoming

Examples

```
remed <- ReMeDI(sampleTData[as.Date(DT) == "2018-01-02", ], kn = 2, lags = 1:8)
# We can also use the algorithm for choosing the kn tuning parameter
optimalKn <- knChooseReMeDI(sampleTData[as.Date(DT) == "2018-01-02",],
                            knMax = 10, tol = 0.05, size = 3,
                            lower = 2, upper = 5, plot = TRUE)
optimalKn
remed <- ReMeDI(sampleTData[as.Date(DT) == "2018-01-02", ], kn = optimalKn, lags = 1:8)
```

ReMeDIAsymptoticVariance *Asymptotic variance of ReMeDI estimator*

Description

Estimates the asymptotic variance of the ReMeDI estimator.

Usage

```
ReMeDIAsymptoticVariance(pData, kn, lags, phi, i)
```

Arguments

pData	xts or data.table containing the log-prices of the asset
kn	numerical value determining the tuning parameter kn this controls the lengths of the non-overlapping interval in the ReMeDI estimation
lags	numeric containing integer values indicating the lags for which to estimate the (co)variance
phi	tuning parameter phi
i	tuning parameter i

Details

Some notation is needed for the estimator of the asymptotic covariance of the ReMeDI estimator. Let

$$\delta\left(n,i\right)=t_{i}^{n}-t_{t-1}^{n},i\geq1,$$

$$\hat{\delta}_{t}^{n}=\left(\frac{k_{n}\delta\left(n,i+1+k_{n}\right)-t_{i+2+2k_{n}}^{n}+t_{i+2+k_{n}}^{n}}{\left(t_{i+k_{n}}^{n}-t_{i}^{n}\right)\vee\phi_{n}}\right)^{2},$$

$$U\left(1\right)_{t}^{n}=\sum_{i=0}^{n_{t}-\omega(1)_{n}}\hat{\delta}_{i}^{n},$$

$$U\left(2,\boldsymbol{j}\right)_{t}^{n}=\sum_{i=0}^{n_{t}-\omega(2)_{n}}\hat{\delta}_{i}^{n}\Delta_{\boldsymbol{j}}\left(Y\right)_{i+\omega(2)_{2}^{n}}^{n},$$

$$U\left(3,\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \sum_{i=0}^{n_t-\omega(3)_n} \hat{\delta}_i^n \Delta_{\boldsymbol{j}}\left(Y\right)_{i+\omega(3)_2^n}^n \Delta_{\boldsymbol{j}'}\left(Y\right)_{i+\omega(3)_3^n}^n ,$$

$$U\left(4;\boldsymbol{j},\boldsymbol{j}'\right)_t^n = -\sum_{i=2^{q-1}k_n}^{n_t-\omega(4)_n} \Delta_{\boldsymbol{j}}\left(Y\right)\Delta_{\boldsymbol{j}'}\left(Y\right)_{i+\omega(3)_3^n}^n ,$$

$$U\left(5,k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \sum_{Q_q\in\mathcal{Q}_q}\sum_{i=2^{e(Q_q)}k_n}^{n_t-\omega(5)_n} \Delta_{\boldsymbol{j}_{Q_q\oplus\left(\boldsymbol{j}'_{Q_{q'}}(+k)\right)}}\left(Y\right)_i^n \prod_{\ell:l_\ell\in Q_q^c} \Delta_{\left(j_{l_\ell},j'_{l_\ell}+k\right)(Y)_{i+\omega(5)_{\ell+1}^n}}',$$

$$U\left(6,k;\boldsymbol{j},\boldsymbol{j}'\right) = \textstyle\sum_{j_l\in\boldsymbol{j},j'_{l'}\in\boldsymbol{j}'}\sum_{i=2k_n}^{n_t-\omega(6)n}\Delta_{\left(j_l,j'_{l'}+k\right)}\left(Y\right)_i^n\Delta_{\boldsymbol{j}_{-l}}\left(Y\right)_{i+\omega(6)_2^n}^n\Delta_{\boldsymbol{j}'_{-l'}}\left(Y\right)_{i+\omega(6)_3^n}^n$$
$$-\textstyle\sum_{j_l\in\boldsymbol{j}}\sum_{i=2^qk_n}^{n_t-\omega'(6)_n}\Delta_{\{j_l\}\oplus\boldsymbol{j}'(+k)}\left(Y\right)_i^n\Delta_{\boldsymbol{j}-l}\left(Y\right)_{i+\omega'(6)_2^n}^n$$
$$-\textstyle\sum_{j'_{l'}\in\boldsymbol{j}'}\sum_{i=2^qk_n}^{n_t-\omega''(6)n}\Delta_{\left\{j'_{l'}+k\right\}\oplus\boldsymbol{j}}\left(Y\right)_i^n\Delta_{\boldsymbol{j}'_{-l'}}\left(Y\right)_{i+\omega''(6)_2^n}^n{}',$$

$$U\left(7,k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n = ReMeDI\left(\boldsymbol{j}\oplus\boldsymbol{j}'\left(+k\right)\right)_t^n ,$$

$$U\left(k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \sum_{\ell=5}^{7} U\left(\ell,k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n ,$$

$$U\left(k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \sum_{\ell=5}^{7} U\left(\ell,k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n ,$$

Where the indices are given by:

$$\omega\left(1\right)_n = 2+2k_n,\ \omega\left(2\right)_2^n = 2+\left(3+2^{q-1}\right)k_n,\ \omega\left(2\right)_n = \omega\left(2\right)_2^n + j_1 + k_n,$$

$$\omega\left(3\right)_2^n = 2+\left(3+2^{q-1}\right)k_n,\ \omega\left(3\right)_3^n = 2+\left(5+2^{q-1}+2^{q'-1}\right)k_n + j_1,$$

$$\omega\left(3\right)_n = \omega\left(3\right)_3^n + j'_1 + k_n,\ \omega\left(4\right)_2^n = 2k_n + q'_n + j_1,\ \omega\left(4\right)_n = \omega\left(4\right)_2^n + j'_1 + k_n,$$

$$e\left(Q_q\right) = \left(2\left|Q_q\right| + q' - q - 1\right)\vee 1,\ \omega\left(5\right)_{\ell+1}^n = 4\ell k_n + \sum_{\ell'=1}^{\ell} j_{l_{\ell'}}\vee\left(j'_{l_\ell}+k\right)\text{for}\ell\geq 1,$$

$$\omega\left(5\right)_n = \omega\left(5\right)_{\left|Q_q^c\right|+1}^n + j_{l_{\left|Q_q^c\right|}}\vee\left(j_{l_{\left|Q_q^c\right|}}+k\right) + k_n,$$

$$\omega\left(6\right)_2^n = \left(2^{q-2}+2\right)k_n + j_\ell\vee\left(j'_{\ell'}+k\right),\ \omega\left(6\right)_3^n = \left(2^{q-2}+2^{q'-2}+2\right)k_n + j_1 + j_\ell\vee\left(j'_\ell+k\right),$$

$$\omega'(6)_2^n = \left(2^{q-2}+2\right)k_n + j_\ell \vee \left(j_1' + k\right),\ \omega''(6)_2^n = \left(2^{q'-2}+1\right)k_n + \left(j_{\ell'}' + k\right) \vee j_1,$$

$$\omega(6)_n = \omega(6)_3^n + j' + k_n,\ \omega'(6)_n = \omega'(6)_2^n + j_1 + k_n,\ \omega''(6)_n = \omega''(6)_2^n\, j_1' + k_n,$$

The asymptotic variance estimator is then given by

$$\hat{\sigma}\left(\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \frac{1}{n_t}\sum_{\ell=1}^{3}\hat{\sigma}_\ell\left(\boldsymbol{j},\boldsymbol{j}'\right)_t^n,$$

where

$$\hat{\sigma}_1\left(\boldsymbol{j},\boldsymbol{j}'\right)_t^n = U\left(0;\boldsymbol{j},\boldsymbol{j}'\right) + \sum_{k=1}^{i_n}\left(U\left(k;\boldsymbol{j},\boldsymbol{j}'\right)_t^n\right) + \left(2i_n+1\right)U\left(4;\boldsymbol{j},\boldsymbol{j}\right)_t^n,$$

$$\hat{\sigma}_2\left(\boldsymbol{j},\boldsymbol{j}'\right)_t^n = U\left(3;\boldsymbol{j},\boldsymbol{j}'\right),$$

$$\hat{\sigma}_3\left(\boldsymbol{j},\boldsymbol{j}'\right)_t^n = \frac{1}{n_t^2}\mathrm{ReMeDI}\left(Y,\boldsymbol{j}\right)_t^n \mathrm{ReMeDI}\left(Y,\boldsymbol{j}'\right)_t^n U\left(1\right)_t^n,$$

$$-\frac{1}{n_t}\left(\mathrm{ReMeDI}\left(Y,\boldsymbol{j}\right)_t^n U\left(2,\boldsymbol{j}'\right)_t^n + \mathrm{ReMeDI}\left(Y,\boldsymbol{j}'\right)_t^n U\left(2,\boldsymbol{j}\right)_t^n\right),$$

Value

a list with components `ReMeDI` and `asympVar` containing the ReMeDI estimation and it's asymptotic variance respectively

Note

We Thank Merrick Li for contributing his Matlab code for this estimator.

Examples

```
kn <- knChooseReMeDI(sampleTDataEurope[, list(DT, PRICE)])

remedi <- ReMeDI(sampleTDataEurope[, list(DT, PRICE)], kn = kn, lags = 0:15)

asympVar <- ReMeDIAsymptoticVariance(sampleTDataEurope[, list(DT, PRICE)],
                                     kn = kn, lags = 0:15, phi = 0.9, i = 2)
```

rHYCov *Hayashi-Yoshida covariance*

Description

Calculates the Hayashi-Yoshida Covariance estimator

Usage

```
rHYCov(
  rData,
  cor = FALSE,
  period = 1,
  alignBy = "seconds",
  alignPeriod = 1,
  makeReturns = FALSE,
  makePsd = TRUE,
  ...
)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
cor	boolean, in case it is TRUE, and the input data is multivariate, the correlation is returned instead of the covariance matrix. FALSE by default.
period	Sampling period
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
makePsd	boolean, in case it is TRUE, the positive definite version of rHYCov is returned. FALSE by default.
...	used internally. Do not set.

Author(s)

Scott Payseur and Emil Sjoerup.

References

Hayashi, T. and Yoshida, N. (2005). On covariance estimation of non-synchronously observed diffusion processes. *Bernoulli*, 11, 359-379.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
library("xts")
hy <- rHYCov(rData = as.xts(sampleOneMinuteData)["2001-08-05"],
             period = 5, alignBy = "minutes", alignPeriod = 5, makeReturns = TRUE)
```

---

rKernelCov *Realized kernel estimator*

Description

Realized covariance calculation using a kernel estimator. The different types of kernels available can be found using listAvailableKernels.

Usage

```
rKernelCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  kernelType = "rectangular",
  kernelParam = 1,
  kernelDOFadj = TRUE,
  ...
)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days
cor	boolean, in case it is TRUE, and the input data is multivariate, the correlation is returned instead of the covariance matrix. FALSE by default.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod to 5 and alignBy to "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
kernelType	Kernel name.

kernelParam	Kernel parameter.
kernelDOFadj	Kernel degree of freedom adjustment.
...	used internally, do not change.

Details

Let $r_{t,i}$ be N returns in period t, $i = 1, \ldots, N$. The returns or prices do not have to be equidistant. The kernel estimator for $H =$ kernelParam is given by

$$\gamma_0 + 2 \sum_{h=1}^{H} k\left(\frac{h-1}{H}\right) \gamma_h,$$

where $k(x)$ is the chosen kernel function and

$$\gamma_h = \sum_{i=h}^{N} r_{t,i} \times r_{t,i-h}$$

is the empirical autocovariance function. The multivariate version employs the cross-covariances instead.

Value

in case the input is and contains data from one day, an N by N matrix is returned. If the data is a univariate xts object with multiple days, an xts is returned. If the data is multivariate and contains multiple days (xts or data.table), the function returns a list containing N by N matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Scott Payseur, Onno Kleen, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., and Shephard, N. (2008). Designing realized kernels to measure the ex post variation of equity prices in the presence of noise. *Econometrica*, 76, 1481-1536.

Hansen, P. and Lunde, A. (2006). Realized variance and market microstructure noise. *Journal of Business and Economic Statistics*, 24, 127-218.

Zhou., B. (1996). High-frequency data and volatility in foreign-exchange rates. *Journal of Business & Economic Statistics*, 14, 45-52.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
# Univariate:
rvKernel <- rKernelCov(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
                       alignPeriod = 5, makeReturns = TRUE)
rvKernel

# Multivariate:
rcKernel <- rKernelCov(rData = sampleOneMinuteData, makeReturns = TRUE)
rcKernel
```

`rKurt` *Realized kurtosis of highfrequency return series.*

Description

Calculate the realized kurtosis as defined in Amaya et al. (2015).

Assume there are N equispaced returns in period t. Let $r_{t,i}$ be a return (with $i = 1, \ldots, N$) in period t. Then, `rKurt` is given by

$$\mbox{rKurt}_{t} = \frac{N \sum_{i=1}^{N}(r_{t,i})^4}{\left(\sum_{i=1}^{N} r^2_{t,i}\right)^2}.$$

Usage

```
rKurt(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Amaya, D., Christoffersen, P., Jacobs, K., and Vasquez, A. (2015). Does realized skewness and kurtosis predict the cross-section of equity returns? *Journal of Financial Economics*, 118, 135-167.

Examples

```
rk <- rKurt(sampleTData[, list(DT, PRICE)], alignBy = "minutes",
            alignPeriod = 5, makeReturns = TRUE)
rk
```

---

rMedRQ *DEPRECATED*

Description

DEPRECATED USE `rMedRQuar`

Usage

```
rMedRQ(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	DEPRECATED USE `rMedRQuar`
`alignBy`	DEPRECATED USE `rMedRQuar`
`alignPeriod`	DEPRECATED USE `rMedRQuar`
`makeReturns`	DEPRECATED USE `rMedRQuar`

---

rMedRQuar *An estimator of integrated quarticity from applying the median operator on blocks of three returns*

Description

Calculate the rMedRQ, defined in Andersen et al. (2012). Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$. Then, the rMedRQ is given by

$$\mathrm{rMedRQ}_t = \frac{3\pi N}{9\pi + 72 - 52\sqrt{3}} \left(\frac{N}{N-2}\right) \sum_{i=2}^{N-1} \mathrm{med}(|r_{t,i-1}|, |r_{t,i}|, |r_{t,i+1}|)^4.$$

Usage

```
rMedRQuar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod to 5 and alignBy to "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.

Value

- In case the input is an xts object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in xts format, an xts will be returned.
- If the input data is a data.table object, the function returns a data.table with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

Examples

```
rq <- rMedRQuar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
               alignPeriod = 5, makeReturns = TRUE)
rq
```

rMedRV *DEPRECATED*

Description

DEPRECATED USE rMedRVar

Usage

```
rMedRV(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

rData	DEPRECATED USE rMedRVar
alignBy	DEPRECATED USE rMedRVar
alignPeriod	DEPRECATED USE rMedRVar
makeReturns	DEPRECATED USE rMedRVar

rMedRVar *rMedRVar*

Description

Calculate the rMedRVar, defined in Andersen et al. (2012). Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t. Then, the rMedRVar is given by

$$\mathrm{rMedRVar}_t = \frac{\pi}{6 - 4\sqrt{3} + \pi} \left(\frac{M}{M-2} \right) \sum_{i=2}^{M-1} \mathrm{med}(|r_{t,i-1}|, |r_{t,i}|, |r_{t,i+1}|)^2$$

Usage

```
rMedRVar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE, ...)
```

Arguments

rData	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days
alignBy	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
makeReturns	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.
...	used internally, do not change.

Details

The rMedRVar belongs to the class of realized volatility measures in this package that use the series of high-frequency returns $r_{t,i}$ of a day t to produce an ex post estimate of the realized volatility of that day t. rMedRVar is designed to be robust to price jumps. The difference between RV and rMedRVar is an estimate of the realized jump variability. Disentangling the continuous and jump components in RV can lead to more precise volatility forecasts, as shown in Andersen et al. (2012)

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

See Also

`IVar` for a list of implemented estimators of the integrated variance.

Examples

```
medrv <- rMedRVar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
                  alignPeriod = 5, makeReturns = TRUE)
medrv
```

---

rMinRQ — *DEPRECATED*

Description

DEPRECATED USE `rMinRQuar`

Usage

```
rMinRQ(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	DEPRECATED USE `rMinRQuar`
`alignBy`	DEPRECATED USE `rMinRQuar`
`alignPeriod`	DEPRECATED USE `rMinRQuar`
`makeReturns`	DEPRECATED USE `rMinRQuar`

`rMinRQuar` *An estimator of integrated quarticity from applying the minimum operator on blocks of two returns*

Description

Calculate the rMinRQuar, defined in Andersen et al. (2012). Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$. Then, the rMinRQuar is given by

$$\mathrm{rMinRQuar}_t = \frac{\pi N}{3\pi - 8}\left(\frac{N}{N-1}\right)\sum_{i=1}^{N-1}\min(|r_{t,i}|, |r_{t,i+1}|)^4$$

Usage

```
rMinRQuar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

Examples

```
rq <- rMinRQuar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
            alignPeriod = 5, makeReturns = TRUE)
rq
```

---

rMinRV *DEPRECATED*

Description

DEPRECATED USE rMinRVar

Usage

```
rMinRV(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

rData	DEPRECATED USE rMinRVar
alignBy	DEPRECATED USE rMinRVar
alignPeriod	DEPRECATED USE rMinRVar
makeReturns	DEPRECATED USE rMinRVar

---

rMinRVar *rMinRVar*

Description

Calculate the rMinRVar, defined in Andersen et al. (2009). Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t. Then, the rMinRVar is given by

$$\text{rMinRVar}_t = \frac{\pi}{\pi - 2}\left(\frac{M}{M-1}\right)\sum_{i=1}^{M-1}\min(|r_{t,i}|, |r_{t,i+1}|)^2$$

Usage

```
rMinRVar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE, ...)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.
`...`	used internally, do not change.

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Jonathan Cornelissen, Kris Boudt, Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

See Also

`IVar` for a list of implemented estimators of the integrated variance.

Examples

```
minrv <- rMinRVar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
                  alignPeriod = 5, makeReturns = TRUE)
minrv
```

`rmLargeSpread` *Delete entries for which the spread is more than* `maxi` *times the median spread*

Description

Function deletes entries for which the spread is more than `"maxi"` times the median spread on that day.

Usage

```
rmLargeSpread(qData, maxi = 50, tz = NULL)
```

Arguments

`qData`	an `xts` or `data.table` object at least containing the columns `"BID"` and `"OFR"`.
`maxi`	an integer. By default `maxi = "50"`, which means that entries are deleted if the spread is more than 50 times the median spread on that day.
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. With the non-disk functionality, we attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`. In the on-disk functionality, if tz is not specified, the timezone used will be the system default.

Value

`xts` or `data.table` object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

`rmNegativeSpread` *Delete entries for which the spread is negative*

Description

Function deletes entries for which the spread is negative.

Usage

```
rmNegativeSpread(qData)
```

Arguments

`qData`	an `xts` object at least containing the columns "BID" and "OFR".

Value

`data.table` or `xts` object

Author(s)

Jonathan Cornelissen, Kris Boudt and Onno Kleen

Examples

```
rmNegativeSpread(sampleQDataRaw)
```

`rmOutliersQuotes` *Remove outliers in quotes*

Description

Delete entries for which the mid-quote is outlying with respect to surrounding entries.

Usage

```
rmOutliersQuotes(qData, maxi = 10, window = 50, type = "standard", tz = NULL)
```

Arguments

`qData`	a `data.table` or `xts` object at least containing the columns `"BID"` and `"OFR"`.
`maxi`	an integer, indicating the maximum number of median absolute deviations allowed.
`window`	an integer, indicating the time window for which the "outlyingness" is considered.
`type`	should be `"standard"` or `"advanced"` (see details).
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. With the non-disk functionality, we attempt to extract the timezone from the `DT` column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`.

Details

- If `type = "standard"`: Function deletes entries for which the mid-quote deviated by more than "maxi" median absolute deviations from a rolling centered median (excluding the observation under consideration) of window observations.
- If `type = "advanced"`: Function deletes entries for which the mid-quote deviates by more than "maxi" median absolute deviations from the value closest to the mid-quote of these three options:
  1. Rolling centered median (excluding the observation under consideration)
  2. Rolling median of the following window of observations
  3. Rolling median of the previous window of observations

  The advantage of this procedure compared to the "standard" proposed by Barndorff-Nielsen et al. (2010) is that it will not incorrectly remove large price jumps. Therefore this procedure has been set as the default for removing outliers.

  Note that the median absolute deviation is taken over the entire day. In case it is zero (which can happen if mid-quotes don't change much), the median absolute deviation is taken over a subsample without constant mid-quotes.

Value

`xts` object or `data.table` depending on type of input.

Author(s)

Jonathan Cornelissen and Kris Boudt.

References

Barndorff-Nielsen, O. E., P. R. Hansen, A. Lunde, and N. Shephard (2009). Realized kernels in practice: Trades and quotes. *Econometrics Journal*, 12, C1-C32.

Brownlees, C.T., and Gallo, G.M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, 51, 2232-2245.

`rMPV` *DEPRECATED*

Description

DEPRECATED USE `rMPVar`

Usage

```
rMPV(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	DEPRECATED
`alignBy`	DEPRECATED
`alignPeriod`	DEPRECATED
`makeReturns`	DEPRECATED

`rMPVar` *Realized multipower variation*

Description

Calculate the Realized Multipower Variation rMPVar, defined in Andersen et al. (2012).

Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$. Then, the rMPVar is given by

$$\text{rMPVar}_N(m,p) = d_{m,p} \frac{N^{p/2}}{N-m+1} \sum_{i=1}^{N-m+1} |r_{t,i}|^{p/m} \ldots |r_{t,i+m-1}|^{p/m}$$

in which

$d_{m,p} = \mu_{p/m}^{-m}$:

m: the window size of return blocks;

p: the power of the variation;

and $m > p/2$.

Usage

```
rMPVar(
  rData,
  m = 2,
  p = 2,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  ...
)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`m`	the window size of return blocks. 2 by default.
`p`	the power of the variation. 2 by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`

`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.
`...`	used internally, do not change.

Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

See Also

`IVar` for a list of implemented estimators of the integrated variance.

Examples

```
mpv <- rMPVar(sampleTData[, list(DT, PRICE)], m = 2, p = 3, alignBy = "minutes",
           alignPeriod = 5, makeReturns = TRUE)
mpv
```

`rMRC` *DEPRECATED rMRC*

Description

DEPRECATED USE `rMRCov`

Usage

```
rMRC(pData, pairwise = FALSE, makePsd = FALSE, theta = 0.8, ...)
```

Arguments

`pData`	DEPRECATED USE `rMRCov`
`pairwise`	DEPRECATED USE `rMRCov`
`makePsd`	DEPRECATED USE `rMRCov`
`theta`	DEPRECATED USE `rMRCov`
`...`	DEPRECATED USE `rMRCov`

rMRCov *Modulated realized covariance*

Description

Calculate univariate or multivariate pre-averaged estimator, as defined in Hautsch and Podolskij (2013).

Usage

```
rMRCov(
  pData,
  pairwise = FALSE,
  makePsd = FALSE,
  theta = 0.8,
  crossAssetNoiseCorrection = FALSE,
  ...
)
```

Arguments

pData	a list. Each list-item contains an `xts` or `data.table` object with the intraday price data of a stock.
pairwise	boolean, should be `TRUE` when refresh times are based on pairs of assets. `FALSE` by default.
makePsd	boolean, in case it is `TRUE`, the positive definite version of rMRCov is returned. `FALSE` by default.
theta	a `numeric` controlling the preaveraging horizon. Detaults to `0.8` as recommended by Hautsch and Podolskij (2013)
crossAssetNoiseCorrection	a `logical` denoting whether to apply the bias correction term on the off-diagonals (covariance) terms. We set this to `FALSE` by default as noise is typically seen as independent across assets.
...	used internally, do not change.

Details

In practice, market microstructure noise leads to a departure from the pure semimartingale model. We consider the process Y in period τ:

$$\mathrm{Y}_{\tau} = X_{\tau} + \epsilon_{\tau},$$

where the observed d dimensional log-prices are the sum of underlying Brownian semimartingale process X and a noise term ϵ_{τ}.

ϵ_{τ} is an *i.i.d.* process with X.

It is intuitive that under mean zero *i.i.d.* microstructure noise some form of smoothing of the observed log-price should tend to diminish the impact of the noise. Effectively, we are going to approximate a continuous function by an average of observations of Y in a neighborhood, the noise being averaged away.

Assume there is N equispaced returns in period τ of a list (after refreshing data). Let r_{τ_i} be a return (with $i = 1, \ldots, N$) of an asset in period τ. Assume there is d assets.

In order to define the univariate pre-averaging estimator, we first define the pre-averaged returns as

$$\bar{r}_{\tau_j}^{(k)} = \sum_{h=1}^{k_N - 1} g\left(\frac{h}{k_N}\right) r_{\tau_{j+h}}^{(k)}$$

where g is a non-zero real-valued function $g : [0, 1] \to R$ given by $g(x) = \min(x, 1 - x)$. k_N is a sequence of integers satisfying $\mathrm{k}_N = \lfloor \theta N^{1/2} \rfloor$. We use $\theta = 0.8$ as recommended in Hautsch and Podolskij (2013). The pre-averaged returns are simply a weighted average over the returns in a local window. This averaging diminishes the influence of the noise. The order of the window size k_n is chosen to lead to optimal convergence rates. The pre-averaging estimator is then simply the analogue of the realized variance but based on pre-averaged returns and an additional term to remove bias due to noise

$$\hat{C} = \frac{N^{-1/2}}{\theta \psi_2} \sum_{i=0}^{N-k_N+1} (\bar{r}_{\tau_i})^2 - \frac{\psi_1^{k_N} N^{-1}}{2\theta^2 \psi_2^{k_N}} \sum_{i=0}^{N} r_{\tau_i}^2$$

with

$$\psi_1^{k_N} = k_N \sum_{j=1}^{k_N} \left(g\left(\frac{j+1}{k_N}\right) - g\left(\frac{j}{k_N}\right) \right)^2,$$

$$\psi_2^{k_N} = \frac{1}{k_N} \sum_{j=1}^{k_N - 1} g^2\left(\frac{j}{k_N}\right).$$

$$\psi_2 = \frac{1}{12}$$

The multivariate counterpart is very similar. The estimator is called the Modulated Realized Covariance (rMRCov) and is defined as

$$\mathrm{MRC} = \frac{N}{N - k_N + 2} \frac{1}{\psi_2 k_N} \sum_{i=0}^{N-k_N+1} \bar{\boldsymbol{r}}_{\tau_i} \cdot \bar{\boldsymbol{r}}'_{\tau_i} - \frac{\psi_1^{k_N}}{\theta^2 \psi_2^{k_N}} \hat{\Psi}$$

where $\hat{\Psi}_N = \frac{1}{2N} \sum_{i=1}^{N} \boldsymbol{r}_{\tau_i} (\boldsymbol{r}_{\tau_i})'$. It is a bias correction to make it consistent. However, due to this correction, the estimator is not ensured PSD. An alternative is to slightly enlarge the bandwidth such that $\mathrm{k}_N = \lfloor \theta N^{1/2+\delta} \rfloor$. $\delta = 0.1$ results in a consistent estimate without the bias correction and a PSD estimate, in which case:

$$\mathrm{MRC}^{\delta} = \frac{N}{N - k_N + 2} \frac{1}{\psi_2 k_N} \sum_{i=0}^{N-k_N+1} \bar{\boldsymbol{r}}_i \cdot \bar{\boldsymbol{r}}'_i$$

Value

A $d \times d$ covariance matrix.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Hautsch, N., and Podolskij, M. (2013). Preaveraging-based estimation of quadratic variation in the presence of noise and jumps: theory, implementation, and empirical Evidence. *Journal of Business & Economic Statistics*, 31, 165-183.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
## Not run:
library("xts")
# Note that this ought to be tick-by-tick data and this example is only to show the usage.
a <- list(as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, MARKET)]),
          as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, STOCK)]))
rMRCov(a, pairwise = TRUE, makePsd = TRUE)


# We can also add use data.tables and use a named list to convey asset names
a <- list(foo = sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, MARKET)],
          bar = sampleOneMinuteData[as.Date(DT) == "2001-08-04", list(DT, STOCK)])
rMRCov(a, pairwise = TRUE, makePsd = TRUE)


## End(Not run)
```

---

rmTradeOutliersUsingQuotes

Delete transactions with unlikely transaction prices

Description

Function deletes entries with prices that are above the ask plus the bid-ask spread. Similar for entries with prices below the bid minus the bid-ask spread.

Usage

```
rmTradeOutliersUsingQuotes(
  tData,
  qData,
  lagQuotes = 0,
  nSpreads = 1,
  BFM = FALSE,
```

```
  backwardsWindow = 3600,
  forwardsWindow = 0.5,
  plot = FALSE,
  ...
)
```

Arguments

tData	a data.table or xts object containing the time series data, with at least the column "PRICE", containing the transaction price.
qData	a data.table or xts object containing the time series data with at least the columns "BID" and "OFR", containing the bid and ask prices.
lagQuotes	numeric, number of seconds the quotes are registered faster than the trades (should be round and positive). Default is 0. For older datasets, i.e. before 2010, it may be a good idea to set this to e.g. 2. See Vergote (2005)
nSpreads	numeric of length 1 denotes how far above the offer and below bid we allow outliers to be. Trades are filtered out if they are MORE THAN nSpread * spread above (below) the offer (bid)
BFM	a logical determining whether to conduct 'Backwards - Forwards matching' of trades and quotes. The algorithm tries to match trades that fall outside the bid - ask and first tries to match a small window forwards and if this fails, it tries to match backwards in a bigger window. The small window is a tolerance for inaccuracies in the timestamps of bids and asks. The backwards window allow for matching of late reported trades, i.e. block trades.
backwardsWindow	a numeric denoting the length of the backwards window. Default is 3600, corresponding to one hour.
forwardsWindow	a numeric denoting the length of the forwards window. Default is 0.5, corresponding to one half second.
plot	a logical denoting whether to visualize the forwards, backwards, and unmatched trades in a plot.
...	used internally

Details

Note: in order to work correctly, the input data of this function should be cleaned trade (tData) and quote (qData) data respectively. In older high frequency datasets the trades frequently lag the quotes. In newer datasets this tends to happen only during extreme market activity when exchange networks are at maximum capacity.

Value

xts or data.table object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Vergote, O. (2005). How to match trades and quotes for NYSE stocks? K.U.Leuven working paper.

Christensen, K., Oomen, R. C. A., Podolskij, M. (2014): Fact or Friction: Jumps at ultra high frequency. *Journal of Financial Economics*, 144, 576-599

`rOWCov` *Realized outlyingness weighted covariance*

Description

Calculate the Realized Outlyingness Weighted Covariance (rOWCov), defined in Boudt et al. (2008).

Let $r_{t,i}$, for $i = 1, ..., M$ be a sample of M high-frequency $(N \times 1)$ return vectors and $d_{t,i}$ their outlyingness given by the squared Mahalanobis distance between the return vector and zero in terms of the reweighted MCD covariance estimate based on these returns.

Then, the rOWCov is given by

$$\text{rOWCov}_t = c_w \frac{\sum_{i=1}^{M} w(d_{t,i}) r_{t,i} r'_{t,i}}{\frac{1}{M}\sum_{i=1}^{M} w(d_{t,i})},$$

The weight $w_{i,\Delta}$ is one if the multivariate jump test statistic for $r_{i,\Delta}$ in Boudt et al. (2008) is less than the 99.9% percentile of the chi-square distribution with N degrees of freedom and zero otherwise. The scalar c_w is a correction factor ensuring consistency of the rOWCov for the Integrated Covariance, under the Brownian Semimartingale with Finite Activity Jumps model.

Usage

```
rOWCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  seasadjR = NULL,
  wFunction = "HR",
  alphaMCD = 0.75,
  alpha = 0.001,
  ...
)
```

Arguments

`rData`	a (MxN) `xts` object containing the N return series over period t, with M observations during t.
`cor`	boolean, in case it is `TRUE`, and the input data is multivariate, the correlation is returned instead of the covariance matrix. `FALSE` by default.

alignBy	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
makeReturns	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.
seasadjR	a (MxN) `xts` object containing the seasonally adjusted returns. This is an optional argument.
wFunction	determines whether a zero-one weight function (one if no jump is detected based on $d_{t,i}$ and 0 otherwise) or Soft Rejection ("SR") weight function is to be used. By default a zero-one weight function (wFunction = "HR") is used.
alphaMCD	a numeric parameter, controlling the size of the subsets over which the determinant is minimized. Allowed values are between 0.5 and 1 and the default is 0.75. See Boudt et al. (2008) or the `covMcd` function in the **robustbase** package.
alpha	is a parameter between 0 and 0.5, that determines the rejection threshold value (see Boudt et al. (2008) for details).
...	used internally, do not change.

Details

Advantages of the rOWCov compared to the `rBPCov` include a higher statistical efficiency, positive semi-definiteness and affine equi-variance. However, the rOWCov suffers from a curse of dimensionality. The rOWCov gives a zero weight to a return vector if at least one of the components is affected by a jump. In the case of independent jump occurrences, the average proportion of observations with at least one component being affected by jumps increases fast with the dimension of the series. This means that a potentially large proportion of the returns receives a zero weight, due to which the rOWCov can have a low finite sample efficiency in higher dimensions.

Value

an $N \times N$ matrix

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Boudt, K., Croux, C., and Laurent, S. (2008). Outlyingness weighted covariation. *Journal of Financial Econometrics*, 9, 657–684.

See Also

`ICov` for a list of implemented estimators of the integrated covariance.

Examples

```
## Not run:
library("xts")
# Realized Outlyingness Weighted Variance/Covariance for prices aligned
# at 1 minutes.

# Univariate:
row <- rOWCov(rData = as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04",
                                                  list(DT, MARKET)]), makeReturns = TRUE)
row

# Multivariate:
rowc <- rOWCov(rData = as.xts(sampleOneMinuteData[as.Date(DT) == "2001-08-04",]),
               makeReturns = TRUE)
rowc

## End(Not run)
```

---

rQPVar *Realized quad-power variation of intraday returns*

Description

Calculate the realized quad-power variation, defined in Andersen et al. (2012).

Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$. Then, the rQPVar is given by

$$\text{rQPVar}_t = N * \frac{N}{N-3} \left(\frac{\pi^2}{4}\right)^{-4} (|r_{t,i}||r_{t,i-1}||r_{t,i-2}||r_{t,i-3}|)$$

Usage

```
rQPVar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE, ...)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally, do not change.

Value

- In case the input is an xts object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in xts format, an xts will be returned.
- If the input data is a data.table object, the function returns a data.table with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

See Also

IVar for a list of implemented estimators of the integrated variance.

Examples

```
qpv <- rQPVar(rData= sampleTData[, list(DT, PRICE)], alignBy= "minutes",
              alignPeriod =5, makeReturns= TRUE)
qpv
```

---

rQuar *Realized quarticity*

Description

Calculate the realized quarticity (rQuar), defined in Andersen et al. (2012).

Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$.

Then, the rQuar is given by

$$\mathrm{rQuar}_t = \frac{N}{3} \sum_{i=1}^{N} (r_{t,i}^4)$$

Usage

```
rQuar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.

Value

- In case the input is an xts object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in xts format, an xts will be returned.
- If the input data is a data.table object, the function returns a data.table with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

Examples

```
rq <- rQuar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
            alignPeriod = 5, makeReturns = TRUE)
rq
```

---

rRTSCov *Robust two time scale covariance estimation*

Description

Calculate the robust two time scale covariance matrix proposed in Boudt and Zhang (2010). Unlike the rOWCov, but similarly to the rThresholdCov, the rRTSCov uses univariate jump detection rules to truncate the effect of jumps on the covariance estimate. By the use of two time scales, this covariance estimate is not only robust to price jumps, but also to microstructure noise and non-synchronic trading.

Usage

```
rRTSCov(
  pData,
  cor = FALSE,
  startIV = NULL,
  noisevar = NULL,
  K = 300,
  J = 1,
  KCov = NULL,
  JCov = NULL,
  KVar = NULL,
  JVar = NULL,
  eta = 9,
  makePsd = FALSE,
  ...
)
```

Arguments

pData	a list. Each list-item i contains an xts object with the intraday price data of stock i for day t.
cor	boolean, in case it is TRUE, and the input data is multivariate, the correlation is returned instead of the covariance matrix. FALSE by default.
startIV	vector containing the first step estimates of the integrated variance of the assets, needed in the truncation. Is NULL by default.
noisevar	vector containing the estimates of the noise variance of the assets, needed in the truncation. Is NULL by default.
K	positive integer, slow time scale returns are computed on prices that are K steps apart.
J	positive integer, fast time scale returns are computed on prices that are J steps apart.
KCov	positive integer, for the extradiagonal covariance elements the slow time scale returns are computed on prices that are K steps apart.
JCov	positive integer, for the extradiagonal covariance elements the fast time scale returns are computed on prices that are J steps apart.
KVar	vector of positive integers, for the diagonal variance elements the slow time scale returns are computed on prices that are K steps apart.
JVar	vector of positive integers, for the diagonal variance elements the fast time scale returns are computed on prices that are J steps apart.
eta	positive real number, squared standardized high-frequency returns that exceed eta are detected as jumps.
makePsd	boolean, in case it is TRUE, the positive definite version of rRTSCov is returned. FALSE by default.
...	used internally, do not change.

Details

The rRTSCov requires the tick-by-tick transaction prices. (Co)variances are then computed using log-returns calculated on a rolling basis on stock prices that are K (slow time scale) and J (fast time scale) steps apart.

The diagonal elements of the rRTSCov matrix are the variances, computed for log-price series X with n price observations at times $\tau_1, \tau_2, \dots, \tau_n$ as follows:

$$(1 - \frac{\overline{n}_K}{\overline{n}_J})^{-1}(\{X,X\}_T^{(K)^*} - \frac{\overline{n}_K}{\overline{n}_J}\{X,X\}_T^{(J)^*}),$$

where $\overline{n}_K = (n-K+1)/K$, $\overline{n}_J = (n-J+1)/J$ and

$$\{X,X\}_T^{(K)^*} = \frac{c_\eta^*}{K} \frac{\sum_{i=1}^{n-K+1}(X_{t_{i+K}} - X_{t_i})^2 I_X^K(i;\eta)}{\frac{1}{n-K+1}\sum_{i=1}^{n-K+1} I_X^K(i;\eta)}.$$

The constant c_η adjusts for the bias due to the thresholding and $I_X^K(i;\eta)$ is a jump indicator function that is one if

$$\frac{(X_{t_{i+K}} - X_{t_i})^2}{(\int_{t_i}^{t_{i+K}} \sigma_s^2 ds + 2\sigma_{\varepsilon_X}^2)} \leq \eta$$

and zero otherwise. The elements in the denominator are the integrated variance (estimated recursively) and noise variance (estimated by the method in Zhang et al, 2005).

The extradiagonal elements of the rRTSCov are the covariances. For their calculation, the data is first synchronized by the refresh time method proposed by Harris et al (1995). It uses the function `refreshTime` to collect first the so-called refresh times at which all assets have traded at least once since the last refresh time point. Suppose we have two log-price series: X and Y. Let $\Gamma = \{\tau_1, \tau_2, \dots, \tau_{N_T^X}\}$ and $\Theta = \{\theta_1, \theta_2, \dots, \theta_{N_T^Y}\}$ be the set of transaction times of these assets. The first refresh time corresponds to the first time at which both stocks have traded, i.e. $\phi_1 = \max(\tau_1, \theta_1)$. The subsequent refresh time is defined as the first time when both stocks have again traded, i.e. $\phi_{j+1} = \max(\tau_{N^X_{\phi_j}+1}, \theta_{N^Y_{\phi_j}+1})$. The complete refresh time sample grid is $\Phi = \{\phi_1, \phi_2, ..., \phi_{M_N+1}\}$, where M_N is the total number of paired returns. The sampling points of asset X and Y are defined to be $t_i = \max\{\tau \in \Gamma : \tau \leq \phi_i\}$ and $s_i = \max\{\theta \in \Theta : \theta \leq \phi_i\}$.

Given these refresh times, the covariance is computed as follows:

$$c_N(\{X,Y\}_T^{(K)} - \frac{\overline{n}_K}{\overline{n}_J}\{X,Y\}_T^{(J)}),$$

where

$$\{X,Y\}_T^{(K)} = \frac{1}{K} \frac{\sum_{i=1}^{M_N-K+1} c_i(X_{t_{i+K}} - X_{t_i})(Y_{s_{i+K}} - Y_{s_i}) I_X^K(i;\eta) I_Y^K(i;\eta)}{\frac{1}{M_N-K+1}\sum_{i=1}^{M_N-K+1} I_X^K(i;\eta) I_Y^K(i;\eta)},$$

with $I_X^K(i;\eta)$ the same jump indicator function as for the variance and c_N a constant to adjust for the bias due to the thresholding.

Unfortunately, the rRTSCov is not always positive semidefinite. By setting the argument `makePsd = TRUE`, the function `makePsd` is used to return a positive semidefinite matrix. This function replaces the negative eigenvalues with zeroes.

Value

an $N \times N$ matrix

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Boudt K. and Zhang, J. 2010. Jump robust two time scale covariance estimation and realized volatility budgets. Mimeo.

Harris, F., McInish, T., Shoesmith, G., and Wood, R. (1995). Cointegration, error correction, and price discovery on informationally linked security markets. *Journal of Financial and Quantitative Analysis*, 30, 563-581.

Zhang, L., Mykland, P. A., and Ait-Sahalia, Y. (2005). A tale of two time scales: Determining integrated volatility with noisy high-frequency data. *Journal of the American Statistical Association*, 100, 1394-1411.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
## Not run:
library(xts)
set.seed(123)
start <- strptime("1970-01-01", format = "%Y-%m-%d", tz = "UTC")
timestamps <- start + seq(34200, 57600, length.out = 23401)

dat <- cbind(rnorm(23401) * sqrt(1/23401), rnorm(23401) * sqrt(1/23401))

dat <- exp(cumsum(xts(dat, timestamps)))
price1 <- dat[,1]
price2 <- dat[,2]
rcRTS <- rRTSCov(pData = list(price1, price2))
# Note: List of prices as input
rcRTS

## End(Not run)
```

---

rRVar *An estimator of realized variance.*

Description

Calculates the daily Realized Variance. Let $r_{t,i}$ be an intraday return vector with $i = 1, ..., M$ number of intraday returns.

Then, the realized variance is given by

$$\mathrm{RVar}_t = \sum_{i=1}^{M} r_{t,i}^2$$

Usage

```
rRVar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE, ...)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally, do not change.

Value

- In case the input is an xts object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in xts format, an xts will be returned.
- If the input data is a data.table object, the function returns a data.table with the same column names as the input data, containing the date and the realized measures.

See Also

IVar for a list of implemented estimators of the integrated variance.

Examples

```
rv <- rRVar(sampleOneMinuteData, makeReturns = TRUE)
plot(rv[, DT], rv[, MARKET], xlab = "Date", ylab = "Realized Variance", type = "l")
```

rSemiCov *Realized semicovariance*

Description

Calculate the Realized Semicovariances (rSemiCov). Let $r_{t,i}$ be an intraday MxN return matrix and $i = 1, ..., M$ the number of intraday returns. Then, let $r^+_{t,i} = max(r_{t,i}, 0)$ and $r^-_{t,i} = min(r_{t,i}, 0)$.

Then, the realized semicovariance is given by the following three matrices:

$$\text{pos}_t = \sum_{i=1}^{M} r^+_{t,i} r^{+'}_{t,i}$$

$$\text{neg}_t = \sum_{i=1}^{M} r^-_{t,i} r^{-'}_{t,i}$$

$$\text{mixed}_t = \sum_{i=1}^{M} (r^+_{t,i} r^{-'}_{t,i} + r^-_{t,i} r^{+'}_{t,i})$$

The mixed covariance matrix will have 0 on the diagonal. From these three matrices, the realized covariance can be constructed as $pos + neg + mixed$. The concordant semicovariance matrix is $pos + neg$. The off-diagonals of the concordant matrix is always positive, while for the mixed matrix, it is always negative.

Usage

```
rSemiCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE
)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`cor`	boolean, in case it is `TRUE`, and the input data is multivariate, the correlation is returned instead of the covariance matrix. `FALSE` by default.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

Details

In the case that cor is TRUE, the mixed matrix will be an $N \times N$ matrix filled with NA as mapping the mixed covariance matrix into correlation space is impossible due to the 0-diagonal.

Value

In case the data consists of one day a list of five $N \times N$ matrices are returned. These matrices are named `mixed`, `positive`, `negative`, `concordant`, and `rCov`. The latter matrix corresponds to the realized covariance estimator and is thus named like the function `rCov`. In case the data spans more than one day, the list for each day will be put into another list named according to the date of the estimates.

Author(s)

Emil Sjoerup.

References

Bollerslev, T., Li, J., Patton, A. J., and Quaedvlieg, R. (2020). Realized semicovariances. *Econometrica*, 88, 1515-1551.

See Also

`ICov` for a list of implemented estimators of the integrated covariance.

Examples

```
# Realized semi-variance/semi-covariance for prices aligned
# at 5 minutes.

# Univariate:
rSVar = rSemiCov(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
                   alignPeriod = 5, makeReturns = TRUE)
rSVar
## Not run:
library("xts")
# Multivariate multi day:
rSC <- rSemiCov(sampleOneMinuteData, makeReturns = TRUE) # rSC is a list of lists
# We extract the covariance between stock 1 and stock 2 for all three covariances.
mixed <- sapply(rSC, function(x) x[["mixed"]][1,2])
neg <- sapply(rSC, function(x) x[["negative"]][1,2])
pos <- sapply(rSC, function(x) x[["positive"]][1,2])
covariances <- xts(cbind(mixed, neg, pos), as.Date(names(rSC)))
colnames(covariances) <- c("mixed", "neg", "pos")
# We make a quick plot of the different covariances
plot(covariances)
addLegend(lty = 1) # Add legend so we can distinguish the series.

## End(Not run)
```

rSkew *Realized skewness*

Description

Calculate the realized skewness, defined in Amaya et al. (2015).

Assume there are N equispaced returns in period t. Let $r_{t,i}$ be a return (with $i = 1, \ldots, N$) in period t. Then, `rSkew` is given by

$$\text{rSkew}_t = \frac{\sqrt{N} \sum_{i=1}^{N} (r_{t,i})^3}{\left(\sum r_{i,t}^2\right)^{3/2}}.$$

Usage

```
rSkew(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Amaya, D., Christoffersen, P., Jacobs, K., and Vasquez, A. (2015). Does realized skewness and kurtosis predict the cross-section of equity returns? *Journal of Financial Economics*, 118, 135-167.

Examples

```
rs <- rSkew(sampleTData[, list(DT, PRICE)],alignBy ="minutes", alignPeriod =5,
           makeReturns = TRUE)
rs
```

---

rSV — *DEPRECATED*

Description

DEPRECATED USE `rSVar`

Usage

```
rSV(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	DEPRECATED USE `rSVar`
`alignBy`	DEPRECATED USE `rSVar`
`alignPeriod`	DEPRECATED USE `rSVar`
`makeReturns`	DEPRECATED USE `rSVar`

---

rSVar — *Realized semivariance of highfrequency return series*

Description

Calculate the realized semivariances, defined in Barndorff-Nielsen et al. (2008).

Function returns two outcomes:

1. Downside realized semivariance
2. Upside realized semivariance.

Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$.

Then, the `rSVar` is given by

$$\mathrm{rSVardownside}_t = \sum_{i=1}^{N} (r_{t,i})^2 \; \times \; I[r_{t,i} < 0]$$

$$\mathrm{rSVarupside}_t = \sum_{i=1}^{N} (r_{t,i})^2 \; \times \; I[r_{t,i} > 0]$$

Usage

```
rSVar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE, ...)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example to aggregate. based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally

Value

list with two entries, the realized positive and negative semivariances

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., Kinnebrock, S., and Shephard N. (2010). *Measuring downside risk: realised semivariance*. In: Volatility and Time Series Econometrics: Essays in Honor of Robert F. Engle, (Edited by Bollerslev, T., Russell, J., and Watson, M.), 117-136. Oxford University Press.

See Also

IVar for a list of implemented estimators of the integrated variance.

Examples

```
sv <- rSVar(sampleTData[, list(DT, PRICE)], alignBy = "minutes",
          alignPeriod = 5, makeReturns = TRUE)
sv
```

rThresholdCov *Threshold Covariance*

Description

Calculate the threshold covariance matrix proposed in Gobbi and Mancini (2009). Unlike the rOWCov, the rThresholdCov uses univariate jump detection rules to truncate the effect of jumps on the covariance estimate. As such, it remains feasible in high dimensions, but it is less robust to small cojumps.

Let $r_{t,i}$ be an intraday $Nx1$ return vector of N assets where $i = 1, ..., M$ and M being the number of intraday returns.

Then, the k, q-th element of the threshold covariance matrix is defined as

$$\text{thresholdcov}[k,q]_t = \sum_{i=1}^{M} r_{(k)t,i} 1_{\{r^2_{(k)t,i} \leq TR_M\}} \; r_{(q)t,i} 1_{\{r^2_{(q)t,i} \leq TR_M\}},$$

with the threshold value TR_M set to $9\Delta^{-1}$ times the daily realized bi-power variation of asset k, as suggested in Jacod and Todorov (2009).

Usage

```
rThresholdCov(
  rData,
  cor = FALSE,
  alignBy = NULL,
  alignPeriod = NULL,
  makeReturns = FALSE,
  ...
)
```

Arguments

rData	an xts or data.table object containing returns or prices, possibly for multiple assets over multiple days.
cor	boolean, in case it is TRUE, and the input data is multivariate, the correlation is returned instead of the covariance matrix. FALSE by default.
alignBy	character, indicating the time scale in which alignPeriod is expressed. Possible values are: "ticks", "secs", "seconds", "mins", "minutes", "hours"
alignPeriod	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set alignPeriod = 5 and alignBy = "minutes".
makeReturns	boolean, should be TRUE when rData contains prices instead of returns. FALSE by default.
...	used internally, do not change.

Value

in case the input is and contains data from one day, an $N \times N$ matrix is returned. If the data is a univariate `xts` object with multiple days, an `xts` is returned. If the data is multivariate and contains multiple days (`xts` or `data.table`), the function returns a list containing $N \times N$ matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Barndorff-Nielsen, O. and Shephard, N. (2004). Measuring the impact of jumps in multivariate price processes using bipower covariation. Discussion paper, Nuffield College, Oxford University.

Jacod, J. and Todorov, V. (2009). Testing for common arrival of jumps in discretely-observed multidimensional processes. *Annals of Statistics*, 37, 1792-1838.

Mancini, C. and Gobbi, F. (2012). Identifying the Brownian covariation from the co-jumps given discrete observations. *Econometric Theory*, 28, 249-273.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
# Realized threshold  Variance/Covariance:
# Multivariate:
## Not run:
library("xts")
set.seed(123)
start <- strptime("1970-01-01", format = "%Y-%m-%d", tz = "UTC")
timestamps <- start + seq(34200, 57600, length.out = 23401)

dat <- cbind(rnorm(23401) * sqrt(1/23401), rnorm(23401) * sqrt(1/23401))

dat <- exp(cumsum(xts(dat, timestamps)))
rcThreshold <- rThresholdCov(dat, alignBy = "minutes", alignPeriod = 1, makeReturns = TRUE)
rcThreshold

## End(Not run)
```

---

rTPQuar *Realized tri-power quarticity*

Description

Calculate the rTPQuar, defined in Andersen et al. (2012).

Assume there are N equispaced returns $r_{t,i}$ in period t, $i = 1, \ldots, N$. Then, the rTPQuar is given by

$$\text{rTPQuar}_t = N \frac{N}{N-2} \left(\frac{\Gamma(0.5)}{2^{2/3}\Gamma(7/6)} \right)^3 \sum_{i=3}^{N} (|r_{t,i}|^{4/3} |r_{t,i-1}|^{4/3} |r_{t,i-2}|^{4/3})$$

Usage

```
rTPQuar(rData, alignBy = NULL, alignPeriod = NULL, makeReturns = FALSE)
```

Arguments

`rData`	an `xts` or `data.table` object containing returns or prices, possibly for multiple assets over multiple days.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`.
`alignPeriod`	positive numeric, indicating the number of periods to aggregate over. For example, to aggregate based on a 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`makeReturns`	boolean, should be `TRUE` when `rData` contains prices instead of returns. `FALSE` by default.

Value

- In case the input is an `xts` object with data from one day, a numeric of the same length as the number of assets.
- If the input data spans multiple days and is in `xts` format, an `xts` will be returned.
- If the input data is a `data.table` object, the function returns a `data.table` with the same column names as the input data, containing the date and the realized measures.

Author(s)

Giang Nguyen, Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2012). Jump-robust volatility estimation using nearest neighbor truncation. *Journal of Econometrics*, 169, 75-93.

Examples

```
tpq <- rTPQuar(rData = sampleTData[, list(DT, PRICE)], alignBy = "minutes",
               alignPeriod = 5, makeReturns = TRUE)
tpq
```

rTSCov *Two time scale covariance estimation*

Description

Calculate the two time scale covariance matrix proposed in Zhang et al. (2005) and Zhang (2010). By the use of two time scales, this covariance estimate is robust to microstructure noise and non-synchronic trading.

Usage

```
rTSCov(
  pData,
  cor = FALSE,
  K = 300,
  J = 1,
  KCov = NULL,
  JCov = NULL,
  KVar = NULL,
  JVar = NULL,
  makePsd = FALSE,
  ...
)
```

Arguments

pData	a list. Each list-item i contains an `xts` object with the intraday price data of stock i for day t.
cor	boolean, in case it is `TRUE`, and the input data is multivariate, the correlation is returned instead of the covariance matrix. `FALSE` by default.
K	positive integer, slow time scale returns are computed on prices that are `K` steps apart.
J	positive integer, fast time scale returns are computed on prices that are `J` steps apart.
KCov	positive integer, for the extradiagonal covariance elements the slow time scale returns are computed on prices that are `K` steps apart.
JCov	positive integer, for the extradiagonal covariance elements the fast time scale returns are computed on prices that are `J` steps apart.
KVar	vector of positive integers, for the diagonal variance elements the slow time scale returns are computed on prices that are `K` steps apart.
JVar	vector of positive integers, for the diagonal variance elements the fast time scale returns are computed on prices that are `J` steps apart.
makePsd	boolean, in case it is `TRUE`, the positive definite version of `rTSCov` is returned. `FALSE` by default.
...	used internally, do not change.

Details

The rTSCov requires the tick-by-tick transaction prices. (Co)variances are then computed using log-returns calculated on a rolling basis on stock prices that are K (slow time scale) and J (fast time scale) steps apart.

The diagonal elements of the rTSCov matrix are the variances, computed for log-price series X with n price observations at times $\tau_1, \tau_2, \ldots, \tau_n$ as follows:

$$(1 - \frac{\overline{n}_K}{\overline{n}_J})^{-1}([X,X]_T^{(K)} - \frac{\overline{n}_K}{\overline{n}_J}[X,X]_T^{(J))}$$

where $\overline{n}_K = (n-K+1)/K$, $\overline{n}_J = (n-J+1)/J$ and

$$[X,X]_T^{(K)} = \frac{1}{K}\sum_{i=1}^{n-K+1}(X_{t_{i+K}} - X_{t_i})^2.$$

The extradiagonal elements of the rTSCov are the covariances. For their calculation, the data is first synchronized by the refresh time method proposed by Harris et al (1995). It uses the function `refreshTime` to collect first the so-called refresh times at which all assets have traded at least once since the last refresh time point. Suppose we have two log-price series: X and Y. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_{N_T^X}\}$ and $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{N_T^Y}\}$ be the set of transaction times of these assets. The first refresh time corresponds to the first time at which both stocks have traded, i.e. $\phi_1 = \max(\tau_1, \theta_1)$. The subsequent refresh time is defined as the first time when both stocks have again traded, i.e. $\phi_{j+1} = \max(\tau_{N^X_{\phi_j}+1}, \theta_{N^Y_{\phi_j}+1})$. The complete refresh time sample grid is $\Phi = \{\phi_1, \phi_2, ..., \phi_{M_N+1}\}$, where M_N is the total number of paired returns. The sampling points of asset X and Y are defined to be $t_i = \max\{\tau \in \Gamma : \tau \le \phi_i\}$ and $s_i = \max\{\theta \in \Theta : \theta \le \phi_i\}$.

Given these refresh times, the covariance is computed as follows:

$$c_N([X,Y]_T^{(K)} - \frac{\overline{n}_K}{\overline{n}_J}[X,Y]_T^{(J)}),$$

where

$$[X,Y]_T^{(K)} = \frac{1}{K}\sum_{i=1}^{M_N-K+1}(X_{t_{i+K}} - X_{t_i})(Y_{s_{i+K}} - Y_{s_i}).$$

Unfortunately, the rTSCov is not always positive semidefinite. By setting the argument makePsd = TRUE, the function `makePsd` is used to return a positive semidefinite matrix. This function replaces the negative eigenvalues with zeroes.

Value

in case the input is and contains data from one day, an N by N matrix is returned. If the data is a univariate `xts` object with multiple days, an `xts` is returned. If the data is multivariate and contains multiple days (`xts` or `data.table`), the function returns a list containing N by N matrices. Each item in the list has a name which corresponds to the date for the matrix.

Author(s)

Jonathan Cornelissen, Kris Boudt, and Emil Sjoerup.

References

Harris, F., McInish, T., Shoesmith, G., and Wood, R. (1995). Cointegration, error correction, and price discovery on informationally linked security markets. *Journal of Financial and Quantitative Analysis*, 30, 563-581.

Zhang, L., Mykland, P. A., and Ait-Sahalia, Y. (2005). A tale of two time scales: Determining integrated volatility with noisy high-frequency data. *Journal of the American Statistical Association*, 100, 1394-1411.

Zhang, L. (2011). Estimating covariation: Epps effect, microstructure noise. *Journal of Econometrics*, 160, 33-47.

See Also

ICov for a list of implemented estimators of the integrated covariance.

Examples

```
# Robust Realized two timescales Variance/Covariance
# Multivariate:
## Not run:
library(xts)
set.seed(123)
start <- strptime("1970-01-01", format = "%Y-%m-%d", tz = "UTC")
timestamps <- start + seq(34200, 57600, length.out = 23401)

dat <- cbind(rnorm(23401) * sqrt(1/23401), rnorm(23401) * sqrt(1/23401))

dat <- exp(cumsum(xts(dat, timestamps)))
price1 <- dat[,1]
price2 <- dat[,2]
rcovts <- rTSCov(pData = list(price1, price2))
# Note: List of prices as input
rcovts

## End(Not run)
```

RV *DEPRECATED DEPRECATED USE* rRVar

Description

DEPRECATED DEPRECATED USE rRVar

Usage

```
RV(rData)
```

Arguments

rData	DEPRECATED USE rRVar

salesCondition *salesCondition is deprecated. Use tradesCondition instead.*

Description

salesCondition is deprecated. Use tradesCondition instead.

Usage

```
salesCondition(
  tData,
  validConds = c("", "@", "E", "@E", "F", "FI", "@F", "@FI", "I", "@I")
)
```

Arguments

tData	salesCondition is deprecated. Use tradesCondition instead.
validConds	salesCondition is deprecated. Use tradesCondition instead.

sampleMultiTradeData *Multivariate tick by tick data*

Description

Cleaned Tick by tick data for a sector ETF, called `ETF` and two stock components of that ETF, these stocks are named `AAA` and `BBB`.

Usage

```
sampleMultiTradeData
```

Format

A `data.table` object

`sampleOneMinuteData` *One minute data*

Description

One minute data price of one stock and a market proxy. This is data from the US market.

Usage

```
sampleOneMinuteData
```

Format

A `data.table` object

`sampleQData` *Sample of cleaned quotes for stock XXX for 2 days measured in microseconds*

Description

A `data.table` object containing the quotes for the pseudonymized stock XXX for 2 days. This is the cleaned version of the data sample `sampleQDataRaw`, using `quotesCleanup`.

Usage

```
sampleQData
```

Format

data.table object

Examples

```
## Not run:
# The code to create the sampleQData dataset from raw data is
sampleQData <- quotesCleanup(qDataRaw = sampleQDataRaw,
                                         exchanges = "N", type = "standard", report = FALSE)

## End(Not run)
```

sampleQDataRaw *Sample of raw quotes for stock XXX for 2 days measured in microseconds*

Description

A `data.table` object containing the raw quotes the pseudonymized stock XXX for 2 days, in the typical NYSE TAQ database format.

Usage

```
sampleQDataRaw
```

Format

data.table object

sampleTData *Sample of cleaned trades for stock XXX for 2 days*

Description

A `data.table` object containing the trades for the pseudonymized stock XXX for 2 days, in the typical NYSE TAQ database format. This is the cleaned version of the data sample `sampleTDataRaw`, using `tradesCleanupUsingQuotes`.

Usage

```
sampleTData
```

Format

A data.table object.

Examples

```
## Not run:
# The code to create the sampleTData dataset from raw data is
sampleQData <- quotesCleanup(qDataRaw = sampleQDataRaw,
                                     exchanges = "N", type = "standard", report = FALSE)

tradesAfterFirstCleaning <- tradesCleanup(tDataRaw = sampleTDataRaw,
                                          exchanges = "N", report = FALSE)

sampleTData <- tradesCleanupUsingQuotes(
  tData = tradesAfterFirstCleaning,
  qData = sampleQData,
```

```
  lagQuotes = 0)[, c("DT", "EX", "SYMBOL", "PRICE", "SIZE")]
# Only some columns are included. These are the ones that were historically included.

# For most applications, we recommend aggregating the data at a high frequency
# For example, every second.
aggregated <- aggregatePrice(sampleTData[, list(DT, PRICE)],
              alignBy = "seconds", alignPeriod = 1)
acf(diff(aggregated[as.Date(DT) == "2018-01-02", PRICE]))
acf(diff(aggregated[as.Date(DT) == "2018-01-03", PRICE]))

signature <- function(x, q){
res <- x[, (rCov(diff(log(PRICE), lag = q, differences = 1))/q), by = as.Date(DT)]
return(res[[2]])
}
rvAgg <- matrix(nrow = 100, ncol = 2)
for(i in 1:100) rvAgg[i, ] <- signature(aggregated, i)
plot(rvAgg[,1], type = "l")
plot(rvAgg[,2], type = "l")

## End(Not run)
```

`sampleTDataEurope` *European data*

Description

Trade data of one stock on one day in the European stock market.

Usage

```
sampleTDataEurope
```

Format

A `data.table` object

`sampleTDataRaw` *Sample of raw trades for stock XXX for 2 days*

Description

An imaginary `data.table` object containing the raw trades the pseudonymized stock XXX for 2 days, in the typical NYSE TAQ database format.

Usage

```
sampleTDataRaw
```

Format

A data.table object.

`selectExchange` *Retain only data from a single stock exchange*

Description

Filter raw trade data to only contain specified exchanges

Usage

```
selectExchange(data, exch = "N")
```

Arguments

`data`	an `xts` or `data.table` object containing the time series data. The object should have a column "EX", indicating the exchange by its symbol.
`exch`	The (vector of) symbol(s) of the stock exchange(s) that should be selected. By default the NYSE is chosen (`exch = "N"`). Other exchange symbols are: • A: AMEX • N: NYSE • B: Boston • P: Arca • C: NSX • T/Q: NASDAQ • D: NASD ADF and TRF • X: Philadelphia • I: ISE • M: Chicago • W: CBOE • Z: BATS

Value

`xts` or `data.table` object depending on input.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

`spotDrift` *Spot Drift Estimation*

Description

Function used to estimate the spot drift of intraday (tick) stock prices/returns

Usage

```
spotDrift(
  data,
  method = "mean",
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = NULL,
  ...
)
```

Arguments

`data`	Can be one of two input types, `xts` or `data.table`. It is assumed that the input comprises prices in levels.
`method`	Which method to be used to estimate the spot-drift. Currently, three methods are available, rolling mean and median as well as the kernel method of Christensen et al. (2018). The kernel is a left hand exponential kernel that will weigh newer observations more heavily than older observations.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	How often should the estimation take place? If `alignPeriod` is 5 the estimation will be done every fifth unit of `alignBy`.
`marketOpen`	Opening time of the market, standard is "09:30:00".
`marketClose`	Closing time of the market, standard is "16:00:00".
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the `DT` column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`.
`...`	Additional arguments for the individual methods. See 'Details'.

Details

The additional arguments for the mean and median methods are:

- `periods` for the rolling window length which is 5 by default.

- `align` controls the alignment. The default is `"right"`.

For the kernel mean estimator, the arguments `meanBandwidth` can be used to control the bandwidth of the drift estimator and the `preAverage` argument, which can be used to control the pre-averaging horizon. These arguments default to 300 and 5 respectively.

The following estimation methods can be specified in `method`:

Rolling window mean (`"mean"`)

Estimates the spot drift by applying a rolling mean over returns.

$$\hat{\mu_t} = \sum_{t=k}^{T} \text{mean}\left(r_{t-k:t}\right),$$

where k is the argument `periods`. Parameters:

- `periods` how big the window for the estimation should be. The estimator will have `periods` `NA`s at the beginning of each trading day.
- `align` alignment method for returns. Defaults to `"left"`, which includes only past data, but other choices, `"center"` and `"right"` are available. Warning: These values includes future data.

Outputs:

- `mu` a matrix containing the spot drift estimates

Rolling window median (`"median"`)

Estimates the spot drift by applying a rolling mean over returns.

$$\hat{\mu_t} = \sum_{t=k}^{T} \text{median}\left(r_{t-k:t}\right),$$

where k is the argument `periods`. Parameters:

- `periods` How big the window for the estimation should be. The estimator will have `periods` `NA`s at the beginning of each trading day.
- `align` Alignment method for returns. Defaults to `"left"`, which includes only past data, but other choices, `"center"` and `"right"` are available. These values includes FUTURE DATA, so beware!

Outputs:

- `mu` a matrix containing the spot drift estimates

kernel spot drift estimator (`"kernel"`)

$$dX_t = \mu_t dt + \sigma_t dW_t + dJ_t,$$

where μ_t, σ_t, and J_t are the spot drift, the spot volatility, and a jump process respectively. However, due to microstructure noise, the observed log-price is

$$Y_t = X_t + \varepsilon_t$$

In order robustify the results to the presence of market microstructure noise, the pre-averaged returns are used:

$$\Delta_i^n \overline{Y} = \sum_{j=1}^{k_n-1} g_j^n \Delta_{i+j}^n Y,$$

where $g(\cdot)$ is a weighting function, $min(x, 1-x)$, and k_n is the pre-averaging horizon. The spot drift estimator is then:

$$\hat{\bar{\mu}}_t^n = \sum_{i=1}^{n-k_n+2} K\left(\frac{t_{i-1}-t}{h_n}\right) \Delta_{i-1}^n \overline{Y},$$

The kernel estimation method has the following parameters:

- `preAverage` a positive `integer` denoting the length of pre-averaging window for the log-prices. Default is 5
- `meanBandwidth` an `integer` denoting the bandwidth for the left-sided exponential kernel for the mean. Default is `300L`

Outputs:

- `mu` a matrix containing the spot drift estimates

Value

An object of class `"spotDrift"` containing at least the estimated spot drift process. Input on what this class should contain and methods for it is welcome.

Author(s)

Emil Sjoerup.

References

Christensen, K., Oomen, R., and Reno, R. (2020) The drift burst hypothesis. Journal of Econometrics. Forthcoming.

Examples

```
# Example 1: Rolling mean and median estimators for 2 days
meandrift <- spotDrift(data = sampleTData, alignPeriod = 1)
mediandrift <- spotDrift(data = sampleTData, method = "median",
                         alignBy = "seconds", alignPeriod = 30, tz = "EST")
plot(meandrift)
plot(mediandrift)
## Not run:
# Example 2: Kernel based estimator for one day with data.table format
price <- sampleTData[as.Date(DT) == "2018-01-02", list(DT, PRICE)]
kerneldrift <- spotDrift(sampleTDataEurope, method = "driftKernel",
                         alignBy = "minutes", alignPeriod = 1)
plot(kerneldrift)

## End(Not run)
```

spotVol *Spot volatility estimation*

Description

Estimates a wide variety of spot volatility estimators.

Usage

```
spotVol(
  data,
  method = "detPer",
  alignBy = "minutes",
  alignPeriod = 5,
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  tz = "GMT",
  ...
)
```

Arguments

`data`	Can be one of two input types, `xts` or `data.table`. It is assumed that the input comprises prices in levels. Irregularly spaced observations are allowed. They will be aggregated to the level specified by parameters `alignBy` and `alignPeriod`.
`method`	specifies which method will be used to estimate the spot volatility. Valid options are `"detPer"`, `"stochPer"` `"kernel"` `"piecewise"` `"garch"`, `"RM"` ,`"PARM"` See 'Details' below for explanation and parameters to use in each of the methods.
`alignBy`	character, indicating the time scale in which `alignPeriod` is expressed. Possible values are: `"ticks"`, `"secs"`, `"seconds"`, `"mins"`, `"minutes"`, `"hours"`
`alignPeriod`	positive integer, indicating the number of periods to aggregate over. For example, to aggregate an `xts` object to the 5-minute frequency, set `alignPeriod = 5` and `alignBy = "minutes"`.
`marketOpen`	the market opening time. This should be in the time zone specified by `tz`. By default, `marketOpen = "09:30:00"`.
`marketClose`	the market closing time. This should be in the time zone specified by `tz`. By default, `marketClose = "16:00:00"`.
`tz`	fallback time zone used in case we we are unable to identify the timezone of the data, by default: `tz = NULL`. We attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use `tz` if specified, and if it is not specified, we use `"UTC"`
`...`	method-specific parameters (see 'Details' below).

Details

The following estimation methods can be specified in `method`:

Deterministic periodicity method (`"detPer"`)

Parameters:

- `dailyVol` A string specifying the estimation method for the daily component s_t. Possible values are `"rBPCov", "rRVar", "rMedRVar"`. `"rBPCov"` by default.
- `periodicVol` A string specifying the estimation method for the component of intraday volatility, that depends in a deterministic way on the intraday time at which the return is observed. Possible values are `"SD", "WSD", "TML", "OLS"`. See Boudt et al. (2011) for details. Default = `"TML"`.
- `P1` A positive integer corresponding to the number of cosine terms used in the flexible Fourier specification of the periodicity function, see Andersen et al. (1997) for details. Default = 5.
- `P2` Same as `P1`, but for the sine terms. Default = 5.
- `dummies` Boolean: in case it is `TRUE`, the parametric estimator of periodic standard deviation specifies the periodicity function as the sum of dummy variables corresponding to each intraday period. If it is `FALSE`, the parametric estimator uses the flexible Fourier specification. Default is `FALSE`.

Outputs (see 'Value' for a full description of each component):

- `spot`
- `daily`
- `periodic`

Let there be T days of N equally-spaced log-returns $r_{i,t}$, $i = 1, \ldots, N$ and $i = 1, \ldots, T$. In case of `method = "detPer"`, the returns are modeled as

$$r_{i,t} = f_i s_t u_{i,t}$$

with independent $u_{i,t} \sim \mathcal{N}(0, 1)$. The spot volatility is decomposed into a deterministic periodic factor f_i (identical for every day in the sample) and a daily factor s_t (identical for all observations within a day). Both components are then estimated separately, see Taylor and Xu (1997) and Andersen and Bollerslev (1997). The jump robust versions by Boudt et al. (2011) have also been implemented.

If `periodicVol = "SD"`, we have

$$\hat{f}_i^{SD} = \frac{SD_i}{\sqrt{\frac{1}{\lfloor \lambda/\Delta \rfloor} \sum_{j=1}^{N} SD_j^2}}$$

with $\Delta = 1/N$, cross-daily averages $SD_i = \sqrt{1/T \sum_{i=t}^{T} r_{i,t}^2}$, and λ being the length of the intraday time intervals.

If `periodicVol = "WSD"`, we have another nonparametric estimator that is robust to jumps in contrast to `periodicVol = "SD"`. The definition of this estimator can be found in Boudt et al. (2011, Eqs. 2.9-2.12).

The estimates when periodicVol = "OLS" and periodicVol = "TML" are based on the regression equation

$$\log \left| 1/T \sum_{t=1}^{T} r_{i,t} \right| - c = \log f_i + \varepsilon_i$$

with *i.i.d.* zero-mean error term ε_i and $c = -0.63518$. periodicVol = "OLS" employs ordinary-least-squares estimation and periodicVol = "TML" truncated maximum-likelihood estimation (see Boudt et al., 2011, Section 2.2, for further details).

Stochastic periodicity method ("stochPer")

Parameters:

- P1: A positive integer corresponding to the number of cosine terms used in the flexible Fourier specification of the periodicity function. Default = 5.
- P2: Same as P1, but for the sine terms. Default = 5.
- init: A named list of initial values to be used in the optimization routine ("BFGS" in optim). Default = list(sigma = 0.03, sigma_mu = 0.005, sigma_h = 0.005, sigma_k = 0.05, phi = 0.2, rho = 0.98, mu = c(2, -0.5), delta_c = rep(0, max(1,P1)),delta_s = rep(0, max(1,P2))). The naming of the parameters follows Beltratti and Morana (2001), the corresponding model equations are listed below. init can contain any number of these parameters. For parameters not specified in init, the default initial value will be used.
- control: A list of options to be passed down to optim.

Outputs (see 'Value' for a full description of each component):

- spot
- par

This method by Beltratti and Morana (2001) assumes the periodicity factor to be stochastic. The spot volatility estimation is split into four components: a random walk, an autoregressive process, a stochastic cyclical process and a deterministic cyclical process. The model is estimated using a quasi-maximum likelihood method based on the Kalman Filter. The package FKF is used to apply the Kalman filter. In addition to the spot volatility estimates, all parameter estimates are returned.

The model for the intraday change in the return series is given by

$$r_{t,n} = \sigma_{t,n} \varepsilon_{t,n},\ t = 1, \ldots, T;\ n = 1, \ldots, N,$$

where $\sigma_{t,n}$ is the conditional standard deviation of the n-th interval of day t and $\varepsilon_{t,n}$ is a *i.i.d.* mean-zero unit-variance process. The conditional standard deviations are modeled as

$$\sigma_{t,n} = \sigma \exp\left(\frac{\mu_{t,n} + h_{t,n} + c_{t,n}}{2}\right)$$

with σ being a scaling factor and $\mu_{t,n}$ is the non-stationary volatility component

$$\mu_{t,n} = \mu_{t,n-1} + \xi_{t,n}$$

with independent $\xi_{t,n} \sim \mathcal{N}(0, \sigma_\xi^2)$. $h_{t,n}$ is the stochastic stationary acyclical volatility component

$$h_{t,n} = \phi h_{t,n-1} + \nu_{t,n}$$

with independent $\eta_{t,n} \sim \mathcal{N}(0, \sigma^2_\eta)$ and $|\phi| \leq 1$. The cyclical component is separated in two components:

$$c_{t,n} = c_{1,t,n} + c_{2,t,n}$$

The first component is written in state-space form,

$$\begin{pmatrix} c_{1,t,n} \\ c^*_{1,t,n} \end{pmatrix} = \rho \begin{pmatrix} \cos\lambda & \sin\lambda \\ -\sin\lambda & \cos\lambda \end{pmatrix} \begin{pmatrix} c_{1,t,n-1} \\ c^*_{1,t,n-1} \end{pmatrix} + \begin{pmatrix} \kappa_{1,t,n} \\ \kappa^*_{1,t,n} \end{pmatrix}$$

with $0 \leq \rho \leq 1$ and $\kappa_{1,t,n}, \kappa^*_{1,t,n}$ are mutually independent zero-mean normal random variables with variance σ^2_κ. All other parameters and the process $c^*_{1,t,n}$ in the state-space representation are only of instrumental use and are not part of the return value which is why we won't introduce them in detail in this vignette; see Beltratti and Morana (2001, pp. 208-209) for more information.

The second component is given by

$$c_{2,t,n} = \mu_1 n_1 + \mu_2 n_2 + \sum_{p=2}^{P} (\delta_{cp} \cos(p\lambda) + \delta_{sp} \sin(p\lambda n))$$

with $n_1 = 2n/(N+1)$ and $n_2 = 6n^2/(N+1)/(N+2)$.

Nonparametric filtering (`"kernel"`)

Parameters:

- `type` String specifying the type of kernel to be used. Options include `"gaussian"`, `"epanechnikov"`, `"beta"`. Default = `"gaussian"`.
- `h` Scalar or vector specifying bandwidth(s) to be used in kernel. If `h` is a scalar, it will be assumed equal throughout the sample. If it is a vector, it should contain bandwidths for each day. If left empty, it will be estimated. Default = `NULL`.
- `est` String specifying the bandwidth estimation method. Possible values include `"cv"`, `"quarticity"`. Method `"cv"` equals cross-validation, which chooses the bandwidth that minimizes the Integrated Square Error. `"quarticity"` multiplies the simple plug-in estimator by a factor based on the daily quarticity of the returns. `est` is obsolete if `h` has already been specified by the user. `"cv"` by default.
- `lower` Lower bound to be used in bandwidth optimization routine, when using cross-validation method. Default is $0.1n^{-0.2}$.
- `upper` Upper bound to be used in bandwidth optimization routine, when using cross-validation method. Default is $n^{-0.2}$.

Outputs (see 'Value' for a full description of each component):

- `spot`
- `par`

This method by Kristensen (2010) filters the spot volatility in a nonparametric way by applying kernel weights to the standard realized volatility estimator. Different kernels and bandwidths can be used to focus on specific characteristics of the volatility process.

Estimation results heavily depend on the bandwidth parameter h, so it is important that this parameter is well chosen. However, it is difficult to come up with a method that determines the optimal bandwidth for any kind of data or kernel that can be used. Although some estimation methods are provided, it is advised that you specify h yourself, or make sure that the estimation results are appropriate.

One way to estimate h, is by using cross-validation. For each day in the sample, h is chosen as to minimize the Integrated Square Error, which is a function of h. However, this function often has multiple local minima, or no minima at all ($h \to \infty$). To ensure a reasonable optimum is reached, strict boundaries have to be imposed on h. These can be specified by `lower` and `upper`, which by default are $0.1n^{-0.2}$ and $n^{-0.2}$ respectively, where n is the number of observations in a day.

When using the method `"kernel"`, in addition to the spot volatility estimates, all used values of the bandwidth h are returned.

A formal definition of the estimator is too extensive for the context of this vignette. Please refer to Kristensen (2010) for more detailed information. Our parameter names are aligned with this reference.

Piecewise constant volatility (`"piecewise"`)

Parameters:

- `type` string specifying the type of test to be used. Options include `"MDa", "MDb", "DM"`. See Fried (2012) for details. Default = `"MDa"`.
- `m` number of observations to include in reference window. Default = `40`.
- `n` number of observations to include in test window. Default = `20`.
- `alpha` significance level to be used in tests. Note that the test will be executed many times (roughly equal to the total number of observations), so it is advised to use a small value for `alpha`, to avoid a lot of false positives. Default = `0.005`.
- `volEst` string specifying the realized volatility estimator to be used in local windows. Possible values are `"rBPCov", "rRVar", "rMedRVar"`. Default = `"rBPCov"`.
- `online` boolean indicating whether estimations at a certain point t should be done online (using only information available at $t-1$), or ex post (using all observations between two change points). Default = `TRUE`.

Outputs (see ‘Value’ for a full description of each component):

- `spot`
- `cp`

This nonparametric method by Fried (2012) is a two-step approach and assumes the volatility to be piecewise constant over local windows. Robust two-sample tests are applied to detect changes in variability between subsequent windows. The spot volatility can then be estimated by evaluating regular realized volatility estimators within each local window. `"MDa", "MDb"` refer to different test statistics, see Section 2.2 in Fried (2012).

Along with the spot volatility estimates, this method will return the detected change points in the volatility level. When plotting a `spotVol` object containing `cp`, these change points will be visualized.

GARCH models with intraday seasonality (`"garch"`)

Parameters:

- `model` string specifying the type of test to be used. Options include `"sGARCH", "eGARCH"`. See `ugarchspec` in the `rugarch` package. Default = `"eGARCH"`.

- `garchorder` numeric value of length 2, containing the order of the GARCH model to be estimated. Default = `c(1,1)`.
- `dist` string specifying the distribution to be assumed on the innovations. See `distribution.model` in `ugarchspec` for possible options. Default = `"norm"`.
- `solver.control` list containing solver options. See `ugarchfit` for possible values. Default = `list()`.
- `P1` a positive integer corresponding to the number of cosine terms used in the flexible Fourier specification of the periodicity function. Default = 5.
- `P2` same as `P1`, but for the sinus terms. Default = 5.

Outputs (see 'Value' for a full description of each component):

- `spot`
- `ugarchfit`

Along with the spot volatility estimates, this method will return the `ugarchfit` object used by the `rugarch` package.

In this model, daily returns r_t based on intraday observations $r_{i,t}, i = 1, \ldots, N$ are modeled as

$$r_t = \sum_{i=1}^{N} r_{i,t} = \sigma_t \frac{1}{\sqrt{N}} \sum_{i=1}^{N} s_i Z_{i,t}.$$

with $\sigma_t > 0$, intraday seasonality $s_i > 0$, and $Z_{i,t}$ being a zero-mean unit-variance error term.

The overall approach is as in Appendix B of Andersen and Bollerslev (1997). This method generates the external regressors s_i needed to model the intraday seasonality with a flexible Fourier form (Andersen and Bollerslev, 1997, Eqs. A.1-A.4). The `rugarch` package is then employed to estimate the specified intraday GARCH(1,1) model on the residuals $r_{i,t}/s_i$.

Realized Measures (`"RM"`)

This estimator takes trailing rolling window observations of intraday returns to estimate the spot volatility.

Parameters:

- `RM` string denoting which realized measure to use to estimate the local volatility. Possible values are: `"rBPCov"`, `"rMedRVar"`, `"rMinRVar"`, `"rCov"`, `"rRVar"`. Default = `"rBPCov"`.
- `lookBackPeriod` positive integer denoting the amount of sub-sampled returns to use for the estimation of the local volatility. Default is `10`.
- `dontIncludeLast` logical indicating whether to omit the last return in the calculation of the local volatility. This is done in Lee-Mykland (2008) to produce jump-robust estimates of spot volatility. Setting this to `TRUE` will then use `lookBackPeriod - 1` returns in the construction of the realized measures. Default = `FALSE`.

Outputs (see 'Value' for a full description of each component):

- `spot`
- `RM`
- `lookBackPeriod`

This method returns the estimates of the spot volatility, a string containing the realized measure used, and the lookBackPeriod.

(Non-overlapping) Pre-Averaged Realized Measures (`"PARM"`)

This estimator takes rolling historical window observations of intraday returns to estimate the spot volatility as in the option `"RM"` but adds return pre-averaging of the realized measures. For a description of return pre-averaging see the details on `spotDrift`.

Parameters:

- `RM` String denoting which realized measure to use to estimate the local volatility. Possible values are: `"rBPCov", "rMedRVar", "rMinRVar", "rCov", and "rRVar"`. Default = `"rBPCov"`.
- `lookBackPeriod` positive integer denoting the amount of sub-sampled returns to use for the estimation of the local volatility. Default = 50.

Outputs (see 'Value' for a full description of each component):

- `spot`
- `RM`
- `lookBackPeriod`
- `kn`

Value

A `spotVol` object, which is a list containing one or more of the following outputs, depending on the method used:

- `spot`

  An `xts` or `matrix` object (depending on the input) containing spot volatility estimates $\sigma_{t,i}$, reported for each interval i between `marketOpen` and `marketClose` for every day t in `data`. The length of the intervals is specified by `alignPeriod` and `alignBy`. Methods that provide this output: All.

  `daily` An `xts` or `numeric` object (depending on the input) containing estimates of the daily volatility levels for each day t in `data`, if the used method decomposed spot volatility into a daily and an intraday component. Methods that provide this output: `"detPer"`.
- `periodic`

  An `xts` or `numeric` object (depending on the input) containing estimates of the intraday periodicity factor for each day interval i between `marketOpen` and `marketClose`, if the spot volatility was decomposed into a daily and an intraday component. If the output is in `xts` format, this periodicity factor will be dated to the first day of the input data, but it is identical for each day in the sample. Methods that provide this output: `"detPer"`.
- `par`

  A named list containing parameter estimates, for methods that estimate one or more parameters. Methods that provide this output: `"stochper", "kernel"`.
- `cp`

  A vector containing the change points in the volatility, i.e. the observation indices after which the volatility level changed, according to the applied tests. The vector starts with a 0. Methods that provide this output: `"piecewise"`.

- `ugarchfit`

  A `ugarchfit` object, as used by the `rugarch` package, containing all output from fitting the GARCH model to the data. Methods that provide this output: `"garch"`.

  The `spotVol` function offers several methods to estimate spot volatility and its intraday seasonality, using high-frequency data. It returns an object of class `spotVol`, which can contain various outputs, depending on the method used. See 'Details' for a description of each method. In any case, the output will contain the spot volatility estimates.

  The input can consist of price data or return data, either tick by tick or sampled at set intervals. The data will be converted to equispaced high-frequency returns $r_{t,i}$ (read: the i-th return on day t).

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Andersen, T. G. and Bollerslev, T. (1997). Intraday periodicity and volatility persistence in financial markets. *Journal of Empirical Finance*, 4, 115-158.

Beltratti, A. and Morana, C. (2001). Deterministic and stochastic methods for estimation of intraday seasonal components with high frequency data. *Economic Notes*, 30, 205-234.

Boudt K., Croux C., and Laurent S. (2011). Robust estimation of intraweek periodicity in volatility and jump detection. *Journal of Empirical Finance*, 18, 353-367.

Fried, R. (2012). On the online estimation of local constant volatilities. *Computational Statistics and Data Analysis*, 56, 3080-3090.

Kristensen, D. (2010). Nonparametric filtering of the realized spot volatility: A kernel-based approach. *Econometric Theory*, 26, 60-93.

Taylor, S. J. and Xu, X. (1997). The incremental volatility information in one million foreign exchange quotations. *Journal of Empirical Finance*, 4, 317-340.

Examples

```
## Not run:
init <- list(sigma = 0.03, sigma_mu = 0.005, sigma_h = 0.007,
             sigma_k = 0.06, phi = 0.194, rho = 0.986, mu = c(1.87,-0.42),
             delta_c = c(0.25, -0.05, -0.2, 0.13, 0.02),
             delta_s = c(-1.2, 0.11, 0.26, -0.03, 0.08))

# Next method will take around 370 iterations
vol1 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)], method = "stochPer", init = init)
plot(vol1$spot[1:780])
legend("topright", c("stochPer"), col = c("black"), lty=1)
## End(Not run)

# Various kernel estimates
## Not run:
h1 <- bw.nrd0((1:nrow(sampleOneMinuteData[, list(DT, PRICE = MARKET)]))*60)
vol2 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)],
```

```
                method = "kernel", h = h1)
vol3 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)],
                method = "kernel", est = "quarticity")
vol4 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)],
                method = "kernel", est = "cv")
plot(cbind(vol2$spot, vol3$spot, vol4$spot))
xts::addLegend("topright", c("h = simple estimate", "h = quarticity corrected",
                     "h = crossvalidated"), col = 1:3, lty=1)

## End(Not run)

# Piecewise constant volatility
## Not run:
vol5 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)],
                method = "piecewise", m = 200, n  = 100, online = FALSE)
plot(vol5)
## End(Not run)

# Compare regular GARCH(1,1) model to eGARCH, both with external regressors
## Not run:
vol6 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)], method = "garch", model = "sGARCH")
vol7 <- spotVol(sampleOneMinuteData[, list(DT, PRICE = MARKET)], method = "garch", model = "eGARCH")
plot(as.numeric(t(vol6$spot)), type = "l")
lines(as.numeric(t(vol7$spot)), col = "red")
legend("topleft", c("GARCH", "eGARCH"), col = c("black", "red"), lty = 1)

## End(Not run)

## Not run:
# Compare realized measure spot vol estimation to pre-averaged version
vol8 <- spotVol(sampleTDataEurope[, list(DT, PRICE)], method = "RM", marketOpen = "09:00:00",
                marketClose = "17:30:00", tz = "UTC", alignPeriod = 1, alignBy = "mins",
                lookBackPeriod = 10)
vol9 <- spotVol(sampleTDataEurope[, list(DT, PRICE)], method = "PARM", marketOpen = "09:00:00",
                marketClose = "17:30:00", tz = "UTC", lookBackPeriod = 10)
plot(zoo::na.locf(cbind(vol8$spot, vol9$spot)))

## End(Not run)
```

---

spreadPrices *Convert to format for realized measures*

Description

Convenience function to split data from one `xts` or `data.table` with at least `"DT"`, `"SYMBOL"`, and `"PRICE"` columns to a format that can be used in the functions for calculation of realized measures. This is the opposite of `gatherPrices`.

Usage

```
spreadPrices(data)
```

Arguments

data An xts or a data.table object with at least "DT", "SYMBOL", and "PRICE" columns. This data should already be cleaned.

Value

An xts or a data.table object with columns "DT" and a column named after each unique entrance in the "SYMBOL" column of the input. These columns contain the price of the associated symbol. We drop all other columns, e.g. SIZE.

Author(s)

Emil Sjoerup.

See Also

gatherPrices

Examples

```
## Not run:
library(data.table)
data1 <- copy(sampleTData)[,  `:=`(PRICE = PRICE * runif(.N, min = 0.99, max = 1.01),
                                   DT = DT + runif(.N, 0.01, 0.02))]
data2 <- copy(sampleTData)[, SYMBOL := 'XYZ']

dat <- rbind(data1, data2)
setkey(dat, "DT")
dat <- spreadPrices(dat)

rCov(dat, alignBy = 'minutes', alignPeriod = 5, makeReturns = TRUE, cor = TRUE)

## End(Not run)
```

SPYRM *SPY realized measures*

Description

Realized measures for the SPY ETF calculated at 1 and 5 minute sampling.

Usage

```
SPYRM
```

Format

A data.table object

Note

The CLOSE column is NOT the official close price, but simply the last recorded price of the day. Thus, this may be slightly different from other sources.

`summary.HARmodel` *Summary for* `HARmodel` *objects*

Description

Summary for `HARmodel` objects

Usage

```
## S3 method for class 'HARmodel'
summary(object, ...)
```

Arguments

`object`	An object of class `HARmodel`
`...`	pass `lag` to determine the maximum order of the Newey West estimator. Default is `22`

Value

A modified `summary.lm`

`tradesCleanup` *Cleans trade data*

Description

This is a wrapper function for cleaning the trade data of all stock data inside the folder dataSource. The result is saved in the folder dataDestination.

In case you supply the argument `rawtData`, the on-disk functionality is ignored. The function returns a vector indicating how many trades were removed at each cleaning step in this case. and the function returns an `xts` or `data.table` object.

The following cleaning functions are performed sequentially: `noZeroPrices`, `autoSelectExchangeTrades` or `selectExchange`, `tradesCondition`, and `mergeTradesSameTimestamp`.

Since the function `rmTradeOutliersUsingQuotes` also requires cleaned quote data as input, it is not incorporated here and there is a separate wrapper called `tradesCleanupUsingQuotes`.

Usage

```
tradesCleanup(
  dataSource = NULL,
  dataDestination = NULL,
  exchanges = "auto",
  tDataRaw = NULL,
  report = TRUE,
  selection = "median",
  validConds = c("", "@", "E", "@E", "F", "FI", "@F", "@FI", "I", "@I"),
  marketOpen = "09:30:00",
  marketClose = "16:00:00",
  printExchange = TRUE,
  saveAsXTS = FALSE,
  tz = NULL
)
```

Arguments

`dataSource` character indicating the folder in which the original data is stored.

`dataDestination` character indicating the folder in which the cleaned data is stored.

`exchanges` vector of stock exchange symbols for all data in `dataSource`, e.g. `exchanges = c("T","N")` retrieves all stock market data from both NYSE and NASDAQ. The possible exchange symbols are:

- A: AMEX
- N: NYSE
- B: Boston
- P: Arca
- C: NSX
- T/Q: NASDAQ
- D: NASD ADF and TRF
- X: Philadelphia
- I: ISE
- M: Chicago
- W: CBOE
- Z: BATS

The default value is `"auto"` which automatically selects the exchange for the stocks and days independently using the `autoSelectExchangeTrades`

`tDataRaw` `xts` object containing raw trade data. This argument is `NULL` by default. Enabling it means the arguments `from`, `to`, `dataSource` and `dataDestination` will be ignored (only advisable for small chunks of data).

`report` boolean and `TRUE` by default. In case it is true the function returns (also) a vector indicating how many trades remained after each cleaning step.

`selection` argument to be passed on to the cleaning routine `mergeTradesSameTimestamp`. The default is "median".

validConds — character vector containing valid sales conditions. Passed through to tradesCondition.

marketOpen — character in the format of "HH:MM:SS", specifying the opening time of the exchange(s).

marketClose — character in the format of "HH:MM:SS", specifying the closing time of the exchange(s).

printExchange — Argument passed to autoSelectExchangeTrades indicates whether the chosen exchange is printed on the console, default is TRUE. This is only used when exchanges is "auto"

saveAsXTS — indicates whether data should be saved in xts format instead of data.table when using on-disk functionality. FALSE by default.

tz — fallback time zone used in case we we are unable to identify the timezone of the data, by default: tz = NULL. With the non-disk functionality, we attempt to extract the timezone from the DT column (or index) of the data, which may fail. In case of failure we use tz if specified, and if it is not specified, we use "UTC". In the on-disk functionality, if tz is not specified, the timezone used will be the system default.

Details

Using the on-disk functionality with .csv.zip files from the WRDS database will write temporary files on your machine in order to unzip the files - we try to clean up after it, but cannot guarantee that there won't be files that slip through the crack if the permission settings on your machine does not match ours.

If the input data.table does not contain a DT column but it does contain DATE and TIME_M columns, we create the DT column by REFERENCE, altering the data.table that may be in the user's environment.

Value

For each day an xts or data.table object is saved into the folder of that date, containing the cleaned data. This procedure is performed for each stock in "ticker". The function returns a vector indicating how many trades remained after each cleaning step.

In case you supply the argument rawtData, the on-disk functionality is ignored and the function returns a list with the cleaned trades as xts object (see examples).

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., and Shephard, N. (2009). Realized kernels in practice: Trades and quotes. *Econometrics Journal*, 12, C1-C32.

Brownlees, C.T. and Gallo, G.M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, 51, 2232-2245.

Examples

```
# Consider you have raw trade data for 1 stock for 2 days
head(sampleTDataRaw)
dim(sampleTDataRaw)
tDataAfterFirstCleaning <- tradesCleanup(tDataRaw = sampleTDataRaw,
                                         exchanges = list("N"))
tDataAfterFirstCleaning$report
dim(tDataAfterFirstCleaning$tData)
# In case you have more data it is advised to use the on-disk functionality
# via "dataSource" and "dataDestination" arguments
```

`tradesCleanupUsingQuotes`

Perform a final cleaning procedure on trade data

Description

Function performs cleaning procedure `rmTradeOutliersUsingQuotes` for the trades of all stocks data in "dataDestination". Note that preferably the input data for this function is trade and quote data cleaned by respectively e.g. `tradesCleanup` and `quotesCleanup`.

Usage

```
tradesCleanupUsingQuotes(
  tradeDataSource = NULL,
  quoteDataSource = NULL,
  dataDestination = NULL,
  tData = NULL,
  qData = NULL,
  lagQuotes = 0,
  nSpreads = 1,
  BFM = FALSE,
  backwardsWindow = 3600,
  forwardsWindow = 0.5,
  plot = FALSE
)
```

Arguments

`tradeDataSource`
: character indicating the folder in which the original trade data is stored.

`quoteDataSource`
: character indicating the folder in which the original quote data is stored.

`dataDestination`
: character indicating the folder in which the cleaned data is stored, folder of `dataSource` by default.

`tData` data.table or xts object containing trade data cleaned by tradesCleanup. This argument is NULL by default. Enabling it, means the arguments from, to, dataSource and dataDestination will be ignored (only advisable for small chunks of data).

`qData` data.table or xts object containing cleaned quote data. This argument is NULL by default. Enabling it means the arguments from, to, dataSource, dataDestination will be ignored (only advisable for small chunks of data).

`lagQuotes` numeric, number of seconds the quotes are registered faster than the trades (should be round and positive). Default is 0. For older datasets, i.e. before 2010, it may be a good idea to set this to, e.g., 2 (see, Vergote, 2005).

`nSpreads` numeric of length 1 denotes how far above the offer and below bid we allow outliers to be. Trades are filtered out if they are MORE THAN nSpread * spread above (below) the offer (bid)

`BFM` a logical determining whether to conduct "Backwards - Forwards matching" of trades and quotes. The algorithm tries to match trades that fall outside the bid - ask and first tries to match a small window forwards and if this fails, it tries to match backwards in a bigger window. The small window is a tolerance for inaccuracies in the timestamps of bids and asks. The backwards window allow for matching of late reported trades, i.e. block trades.

`backwardsWindow` a numeric denoting the length of the backwards window used when BFM = TRUE. Default is 3600, corresponding to one hour.

`forwardsWindow` a numeric denoting the length of the forwards window used when BFM = TRUE. Default is 0.5, corresponding to one half second.

`plot` a logical denoting whether to visualize the forwards, backwards, and unmatched trades in a plot. Passed on to rmTradeOutliersUsingQuotes

Details

In case you supply the arguments tData and qData, the on-disk functionality is ignored and the function returns cleaned trades as a data.table or xts object (see examples).

When using the on-disk functionality and tradeDataSource and quoteDataSource are the same, the quote files are all files in the folder that contains 'quote', and the rest are treated as containing trade data.

Value

For each day an xts object is saved into the folder of that date, containing the cleaned data.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A., and Shephard, N. (2009). Realized kernels in practice: Trades and quotes. *Econometrics Journal*, 12, C1-C32.

Brownlees, C.T., and Gallo, G.M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, 51, 2232-2245.

Christensen, K., Oomen, R. C. A., Podolskij, M. (2014): Fact or Friction: Jumps at ultra high frequency. *Journal of Financial Economics*, 144, 576-599

Examples

```
# Consider you have raw trade data for 1 stock for 2 days
tDataAfterFirstCleaning <- tradesCleanup(tDataRaw = sampleTDataRaw,
                                          exchanges = "N", report = FALSE)
qData <- quotesCleanup(qDataRaw = sampleQDataRaw,
                       exchanges = "N", report = FALSE)
dim(tDataAfterFirstCleaning)
tDataAfterFinalCleaning <-
  tradesCleanupUsingQuotes(qData = qData[as.Date(DT) == "2018-01-02"],
                           tData = tDataAfterFirstCleaning[as.Date(DT) == "2018-01-02"])
dim(tDataAfterFinalCleaning)
# In case you have more data it is advised to use the on-disk functionality
# via the "tradeDataSource", "quoteDataSource", and "dataDestination" arguments
```

`tradesCondition` *Delete entries with abnormal trades condition.*

Description

Delete entries with abnormal trades condition

Usage

```
tradesCondition(
  tData,
  validConds = c("", "@", "E", "@E", "F", "FI", "@F", "@FI", "I", "@I")
)
```

Arguments

`tData`	an `xts` or `data.table` object containing the time series data, with one column named `"COND"` indicating the Sale Condition.
`validConds`	a character vector containing valid sales conditions defaults to `c('', '@', 'E', '@E', 'F', 'FI', '@F', '@FI', 'I', '@I')`. See details.

Details

To get more information on the sales conditions, see the NYSE documentation. Section about Daily TAQ Trades File. The current version (as of May 2020) can be found online at NYSE's webpage

Value

`xts` or `data.table` object depending on input.

Note

Some CSV readers and the WRDS API parses empty strings as NAs. We transform `NA` values in COND to `""`.

Author(s)

Jonathan Cornelissen, Kris Boudt, Onno Kleen, and Emil Sjoerup.

Index