Package ‘highfrequency’

January 15, 2020

Version 0.6.3
Date 2020-01-14
Title Tools for Highfrequency Data Analysis
Maintainer Kris Boudt
  <kris.boudt@ugent.be>
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.
License GPL (>= 2)
Encoding UTF-8
LazyData true
Depends R (>= 3.5.0)
Imports xts, zoo, Rcpp, RcppArmadillo, graphics, methods, stats, utils, grDevices, robustbase, cubature, mvtnorm, data.table (>= 1.12.0), RcppRoll, lubridate, readr
LinkingTo Rcpp, RcppArmadillo
Suggests covr, FKF, BMS, rugarch, testthat, knitr, rmarkdown
RoxygenNote 7.0.1
VignetteBuilder knitr
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>)
Repository CRAN
Date/Publication 2020-01-15 00:00:02 UTC
R topics documented:

- highfrequency-package .................................................. 3
- aggregateQuotes .......................................................... 4
- aggregateTrades .......................................................... 5
- aggregateats .............................................................. 7
- AJjumptest ................................................................. 8
- autoSelectExchangeQuotes ............................................... 11
- autoSelectExchangeTrades ................................................ 12
- BNSjumptest ............................................................... 13
- exchangeHoursOnly ...................................................... 15
- getLiquidityMeasures .................................................... 16
- getPrice ................................................................. 20
- getTradeDirection ....................................................... 20
- harModel ................................................................. 21
- hasQty ................................................................. 25
- heavyModel ............................................................... 25
- ivInference ............................................................... 28
- JQjumptest ............................................................... 30
- listAvailableKernels .................................................... 32
- lltc ................................................................. 32
- makePsd .............................................................. 33
- makeReturns ............................................................ 34
- matchTradesQuotes ..................................................... 34
- medRQ .............................................................. 35
- medRV .............................................................. 36
- mergeQuotesSameTimestamp ........................................... 37
- mergeTradesSameTimestamp ........................................... 38
- minRQ .............................................................. 39
- minRV .............................................................. 40
- MRC .............................................................. 41
- noZeroPrices .......................................................... 43
- noZeroQuotes .......................................................... 43
- quotesCleanup .......................................................... 44
- rAVGCov ............................................................ 46
- rBeta .............................................................. 47
- rBPCov .............................................................. 49
- rCov .............................................................. 50
- realized_library ......................................................... 51
- refreshTime ............................................................ 52
- rHYCov .............................................................. 53
- rKernelCov ............................................................ 54
- rKurt .............................................................. 55
- rmLargeSpread ........................................................ 56
- rmNegativeSpread ..................................................... 57
- rmOutliersQuotes ...................................................... 58
- rMPV .............................................................. 59
- rmTradeOutliers ....................................................... 60
The highfrequency package provides numerous tools for analyzing financial (highfrequency) data. It has five main goals:

- 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.

To learn more about highfrequency, start with the vignettes: ‘browseVignettes(package = "highfrequency")’

Author(s)
Kris Boudt, Jonathan Cornelissen, Scott Payseur
Maintainer: Kris Boudt <Kris.Boudt@econ.kuleuven.be>
Contributors: Giang Nguyen, Onno Kleen
Thanks: We would like to thank Brian Peterson, Chris Blakely, Eric Zivot and Maarten Schermer. We are also grateful to Dirk Eddelbuettel for his extraordinary support as a mentor during the Google Summer of Code 2019. Moreover, we thank Emil Sjørup for implementing additional options in the harModel function.

aggregateQuotes

Aggregate a data.table or xts object containing quote data

Description

Function returns a data.table or xts object containing the aggregated quote data with columns "SYMBOL", "EX", "BID", "BIDSIZ", "OFR", "OFRSIZ". See sample_qdata for an example of the argument qdata.

Usage

aggregateQuotes(
    qdata,
    on = "minutes",
    k = 5,
    marketopen = "09:30:00",
    marketclose = "16:00:00",
    tz = "GMT"
)

Arguments

qdata  data.table or xts object to be aggregated, containing the intraday quote data of a stock for one day.
on character, indicating the time scale in which "k" is expressed. Possible values are: "secs", "seconds", "mins", "minutes", "hours". xts object to the 5 minute frequency, set k=5 and on = "minutes".
k positive integer, indicating the number of periods to aggregate over. E.g. to aggregate an object to the 5 minute frequency set k = 5 and on = "minutes".
marketopen the market opening time, by default: marketopen = "09:30:00".
marketclose the market closing time, by default: marketclose = "16:00:00".
tz time zone used, by default: tz = "GMT".
aggregateTrades

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

An data.table or xts object containing the aggregated time series.

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

Author(s)

Jonathan Cornelissen, Kris Boudt and Onno Kleen

Examples

# aggregate quote data to the 30 second frequency
qdata_aggregated <- aggregateQuotes(sample_qdata, on = "seconds", k = 30)
head(qdata_aggregated)

aggregateTrades  Aggregate a data.table or xts object containing trades data

Description

Function returns new 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.

Usage

aggregateTrades(
    tdata,
    on = "minutes",
    k = 5,
    marketopen = "09:30:00",
    marketclose = "16:00:00",
    tz = "GMT"
)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tdata</td>
<td>data.table or xts object to be aggregated, containing the intraday price series of a stock for possibly multiple days.</td>
</tr>
<tr>
<td>on</td>
<td>character, indicating the time scale in which &quot;k&quot; is expressed. Possible values are: &quot;secs&quot;, &quot;seconds&quot;, &quot;mins&quot;, &quot;minutes&quot;, &quot;hours&quot;.</td>
</tr>
<tr>
<td>k</td>
<td>positive integer, indicating the number of periods to aggregate over. E.g. to aggregate an object to the 5 minute frequency set k = 5 and on = &quot;minutes&quot;.</td>
</tr>
<tr>
<td>marketopen</td>
<td>the market opening time, by default: marketopen = &quot;09:30:00&quot;.</td>
</tr>
<tr>
<td>marketclose</td>
<td>the market closing time, by default: marketclose = &quot;16:00:00&quot;.</td>
</tr>
<tr>
<td>tz</td>
<td>time zone used, by default: tz = &quot;GMT&quot;.</td>
</tr>
</tbody>
</table>

Details

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

In case of previous tick aggregation or on = "seconds"/"minutes"/"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.

Value

An data.table or xts object containing the aggregated time series.

Author(s)

Jonathan Cornelissen, Kris Boudt and Onno Kleen.

Examples

```r
# aggregate trade data to 5 minute frequency
tdata_aggregated <- aggregateTrades(sample_tdata, on = "minutes", k = 5)
head(tdata_aggregated)
```
**aggregatets**  
*Aggregate a time series*

**Description**

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

**Usage**

```r
aggregatets(
  ts,
  FUN = "previoustick",
  on = "minutes",
  k = 1,
  weights = NULL,
  dropna = FALSE
)
```

**Arguments**

- **ts**
  - xts 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.
- **on**
  - character, indicating the time scale in which "k" is expressed. Possible values are: "secs", "seconds", "mins", "minutes", "hours", "days", "weeks".
- **k**
  - positive integer, indicating the number of periods to aggregate over. For example, to aggregate an xts object to the five-minute frequency set k = 5 and on = "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 timestamps 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).

**Details**

The timestamps of the new time series are the closing times and/or days of the intervals. E.g. for a weekly aggregation the new timestamp is the last day in that particular week (namely sunday).

In case of previous tick aggregation, for on = "seconds"/"minutes"/"hours", the element of the returned series with e.g. timestamp 09:35:00 contains the last observation up to that point, excluding 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 NA's by the function `na.locf` (last observation carried forward) from the zoo package.

**Value**

An xts object containing the aggregated time series.

**Author(s)**

Jonathan Cornelissen and Kris Boudt

**Examples**

```r
#load sample price data
ts <- sample_tdata$PRICE

#Previous tick aggregation to the 5-minute sampling frequency:
tsagg5min <- aggregatets(ts, on = "minutes", k = 5)
head(tagg5min)

#Previous tick aggregation to the 30-second sampling frequency:
tsagg30sec <- aggregatets(ts, on = "seconds", k = 30)
tail(tagg30sec)
```

---

**Description**

This test examines the presence of jumps in high-frequency price series. It is based on the theory of Ait-Sahalia and Jacod (2009) (AJ). It consists in comparing the multipower variation of equispaced 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:

\[
AJjumptest_{t,N} = \frac{S_t(p, k, h) - k^{p/2-1}}{\sqrt{V_{t,N}}}
\]

in which,

\[
S_t(p, k, h) = \frac{PV_{t,M}(p, kh)}{PV_{t,M}(p, h)}
\]

\[
PV_{t,N}(p, kh) = \sum_{i=1}^{N/k} |y_{t,i}|^p
\]

\[
PV_{t,N}(p, h) = \sum_{i=1}^{N} |r_{t,i}|^p
\]

\[
V_{t,N} = \frac{N(p, k)A_{t,N(2p)}}{NA_{t,N(p)}}
\]

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

\[
A_{t,n(2p)} = \left(\frac{1}{N}\right)^{(1-p/2)} \frac{1}{\mu_p} \sum_{i=1}^{N} |r_{t,i}|^p \text{ for } |r_j| < \alpha(1/N)^w
\]

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

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

**Usage**

```r
AJjumptest(
pdata,
p = 4,
k = 2,
align.by = NULL,
align.period = NULL,
alpha.multiplier = 4,
makeReturns = FALSE,
...
)
```
Arguments

pdata  a zoo/xts object containing all prices in period t for one asset.
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.
align.by a string, align the tick data to "seconds","minutes","hours"
align.period an integer, align the tick data to this many [seconds,minutes,hours].
alpha.multiplier   alpha multiplier
makeReturns boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.
...   additional arguments.

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:

$$X_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u + Z_t$$

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

$$Z_t = \sum_{j=1}^{N_t} k_j$$

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

The Ait-Sahalia and Jacod test is that: 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& Zikes(2009))

Value

list

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


autoSelectExchangeQuotes

Examples

AJjumptest(sample_tdata$PRICE, p = 2, k = 3, align.by = "seconds",
align.period = 5, makeReturns = TRUE)

Description

Function returns an xts object containing only observations of the exchange with highest value for
the sum of "BIDSIZ" and "OFRSIZ", i.e. the highest quote volume.

Usage

autoSelectExchangeQuotes(qdata)

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. The chosen exchange is printed on the console.
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 and Onno Kleen
Examples

autoSelectExchangeQuotes(sample_qdataraw_microseconds)

---

autoSelectExchangeTrades

*etain only data from the stock exchange with the highest trading volume*

Description

Function returns a data.table or xts object containing only observations of the exchange with the highest value for the variable "SIZE", i.e. the highest trade volume.

Usage

autoSelectExchangeTrades(tdata)

Arguments

tdata  
an xts object with at least a column "EX", indicating the exchange symbol and "SIZE", indicating the trade volume. The chosen exchange is printed on the console.
  
  • 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 and Onno Kleen

Examples

autoSelectExchangeTrades(sample_tdataraw_microseconds)

Description

This test examines the presence of jumps in high-frequency price series. It is based on theory of Barndorff-Nielsen and Shephard (BNS). The null hypothesis is no jumps. Depending on the choice of estimator (integrated variance (IVestimator), integrated quarticity (IQestimator)), mechanism (linear, ratio) and adjustment (logarith), the function returns the result. Function returns three outcomes: 1. z-test value 2. critical value (with confidence level of 95%) and 3. p-value of the test. Assume there is N equispaced returns in period t.

Assume the Realized variance (RV), IVestimator and IQestimator are based on N equispaced returns.

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

Then the BNSjumptest is given by:

$$BNSjumptest = \frac{RV - IVestimator}{\sqrt{(\theta - 2) \frac{1}{N} IQestimator}}$$

in which, $IVestimator$ can be: bipower variance (BV), minRV, medRV. $IQestimator$ can be: tripower quarticity (TP), quadpower quarticity (QP), minRQ, medRQ.

$\theta$: depends on IVestimator (Huang and Tauchen (2005)).

Usage

```r
BNSjumptest( rdata, IVestimator = "BV", IQestimator = "TP", type = "linear", logtransform = FALSE, max = FALSE, align.by = NULL, align.period = NULL, makeReturns = FALSE )
```

Arguments

- **rdata** a zoo/xts object containing all returns in period t for one asset.
- **IVestimator** can be chosen among jump robust integrated variance estimators: BV, minRV, medRV and corrected threshold bipower variation (CTBV). If CTBV is chosen, an argument of startV, start point of auxiliary estimators in threshold estimation (Corsi et al. (2010) can be included. BV by default.
IQuestimator can be chosen among jump robust integrated quarticity estimators: TP, QP, minRQ and medRQ. TP 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 logarith form. FALSE by default.

max boolean, should be TRUE when max adjustment in SE. FALSE by default.

align.by a string, align the tick data to "seconds"/"minutes"/"hours".

align.period an integer, align the tick data to this many [seconds|minutes|hours].

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

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:

$$X_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u + Z_t$$

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

$$Z_t = \sum_{j=1}^{N_t} k_j$$

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 $RV_{t,N}$ and the IVestimator captures the jump part only, and this observation underlines the BNS test for jumps. (Theodosiou & Zikes(2009))

Value

list

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


**exchangeHoursOnly**

**Examples**

```r
BNSjumptest(sample_tdata$PRICE, IEstimator= "minRV",
            IQestimator= "medRQ", type= "linear", makeReturns = TRUE)
```

---

**Description**

The function returns data within exchange trading hours "daybegin" and "dayend". By default, daybegin and dayend are set to "09:30:00" and "16:00:00" respectively (see Brownlees and Gallo (2006) for more information on good choices for these arguments).

**Usage**

```r
exchangeHoursOnly(data, daybegin = "09:30:00", dayend = "16:00:00")
```

**Arguments**

- **data**: a data.table or xts object containing the time series data. Multiple days of input are allowed.
- **daybegin**: character in the format of "HH:MM:SS", specifying the starting hour, minute and second of an exchange trading day.
- **dayend**: character in the format of "HH:MM:SS", specifying the closing hour, minute and second of an exchange trading day.

**Value**

xts or data.table object depending on input

**Author(s)**

Jonathan Cornelissen, Kris Boudt and Onno Kleen.

**References**


**Examples**

```r
exchangeHoursOnly(sample_tdata$microseconds)
```
getLiquidityMeasures

Compute Liquidity Measure Function returns an xts or data.table object containing 23 liquidity measures. Please see details below. Note that this assumes a regular time grid. The Lee + Ready measure uses two lags for the Tick Rule.

Description

Compute Liquidity Measure

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

Note that this assumes a regular time grid. The Lee + Ready measure uses two lags for the Tick Rule.

Usage

ggLiquidityMeasures(tqdata, win = 300, type = NULL)

Arguments

tqdata A data.table or xts object as in the highfrequency merged trades and quotes data (that is included).

win A windows length for the forward-prices used for ‘realized’ spread

type Legacy option. Default is to return all liquidity measures.

Details

NOTE: xts or data.table should only contain one day of observations

The respective liquidity measures are defined as follows:

- effectiveSpread

\[
eff_{\text{effective spread}} = 2 \times D_t \times (\text{PRICE}_t - \frac{(\text{BID}_t + \text{OFR}_t)}{2}),
\]

where \( D_t \) is 1 (-1) if \( \text{trade}_t \) 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

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

where \( D_t \) is 1 (-1) if \( \text{trade}_t \) 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

\[\text{trade value}_t = \text{SIZE}_t \times \text{PRICE}_t.\]
getLiquidityMeasures

- **signedValueTrad**: signed trade value

\[ \text{signed trade value}_t = D_t \times (\text{SIZE}_t \times \text{PRICE}_t), \]

where \( D_t \) is 1 (-1) if trade \( t \) was buy (sell) (see Boehmer (2005), Bessembinder (2003)).

- **depthImbalanceDifference**: depth imbalance (as a difference)

\[ \text{depth imbalance (as difference)}_t = \frac{D_t \times (\text{OFRSIZ}_t - \text{BIDSIZ}_t)}{(\text{OFRSIZ}_t + \text{BIDSIZ}_t)}, \]

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 where the time indication refers to (and thus not to the registered quote timestamp).

- **depthImbalanceRatio**: depth imbalance (as ratio)

\[ \text{depth imbalance (as ratio)}_t = \left( \frac{\text{OFRSIZ}_t}{\text{BIDSIZ}_t} \right)^{D_t}, \]

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 where 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 where 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 where the time indication refers to (and thus not to the registered quote timestamp).

- **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

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

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

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

where \( D_t \) is 1 (-1) if \( \text{trade}_t \) 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

\[ \text{proportional half traded spread}_t = \frac{\text{half traded spread}_t}{\text{OFR}_t + \text{BID}_t}. \]

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

\[ \text{proportional quoted spread}_t = \frac{\text{quoted spread}_t}{\text{OFR}_t + \text{BID}_t}. \]

(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

\[ \text{log quoted spread}_t = \log\left(\frac{\text{OFR}_t}{\text{BID}_t}\right) \]

(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).
getLiquidityMeasures

- quotedSlope: quoted slope
  \[ \text{quoted slope}_t = \frac{\text{quoted spread}_t}{\log \text{quoted size}_t} \]
  (Hasbrouck and Seppi, 2001).
- logQslope: log quoted slope
  \[ \log \text{quoted slope}_t = \frac{\log \text{quoted spread}_t}{\log \text{quoted size}_t} \]
- 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
  \[ \text{signed trade size}_t = \text{D}_t \ast \text{SIZE}_t, \]
  where \( \text{D}_t \) is 1 (-1) if \( \text{trade}_t \) was buy (sell).

Value

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

References


Examples

tqdata <- matchTradesQuotes(sample_tdata, sample_qdata)
res <- getLiquidityMeasures(tqdata)
head(res)
getPrice

Get price column(s) from a timeseries

Description

Will attempt to locate price column(s) from a time series with rational defaults.

Usage

getPrice(x, symbol = NULL, prefer = NULL)

Arguments

x
A data object with columns containing data to be extracted

symbol
text string containing the symbol to extract

prefer
preference for any particular type of price, see Details

Details

May be subset by symbol and preference. prefer Preference will be for any commonly used financial time series price description, e.g. 'trade', 'close', 'bid', 'ask' with specific tests and matching for types and column names currently supported in R, but a default grep match will be performed if one of the supported types doesn’t match.

The functionality was taken from the quantmod-package

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 \texttt{matchTradesQuotes})

Details

NOTE: The value of the first (and second) observation of the output should be ignored if price == midpoint for the first (second) observation.
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 and Onno Kleen. Special thanks to Dirk Eddelbuettel.

References


Examples

```r
# generate matched trades and quote data set
tqdata <- matchTradesQuotes(sample_tdata, sample_qdata)
directions <- getTradeDirection(tqdata)
head(directions)
```

harModel

**HAR model estimation (Heterogeneous Autoregressive model for Realized volatility)**

Description

Function returns the estimates for the Heterogeneous Autoregressive model 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. Consult the vignette for more information.

Usage

```r
harModel(
data,
periods = c(1, 5, 22),
periodsJ = c(1, 5, 22),
periodsQ = c(1),
leverage = NULL,
RVest = c("rCov", "rBPCov", "rQuar"),
type = "HARRV",
inputType = "RM",
jumptest = "ABDJumptest",
alpha = 0.05,
h = 1,
transform = NULL,
...
)
```
Arguments

data
an xts object containing either: intra-day (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.

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 = "HARRVJ", type = "HARRVCJ", type = "HARRVQJ" the second element refers to the name of the function to estimate the continuous component of daily volatility (jump robust). If type = "HARRVQ", the second element refers to the name of the function used to estimate the integrated quarticity. If type = "HARRVQJ" 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 = "HARRV", the most basic model. Other valid options are type = "HARRVJ", type = "HARRVCJ", type = "HARRVQ", type = "HARRVQJ", type = "CHARRV", or type = "CHARRVQ".

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).

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”.

extra arguments for jump test.

Value

The function outputs an object of class harModel and lm (so harModel is a subclass of lm).

Author(s)

Jonathan Cornelissen and Kris Boudt

References


Examples

##### Example 1: HARRVCJ #####

dat <- sample_5minprices_jumps$stock1
dat <- makeReturns(dat) #Get the high-frequency return data

x <- harModel(dat, periods = c(1,5,10), periodsJ = c(1,5,10),
RVest = c("rCov","rBPCov"),
type = "HARRVCJ", transform = "sqrt", inputType = "returns")

# Estimate the HAR model of type HARRVCJ
class(x)
x
# plot(x)
predict(x)

##### Example 2: HARRV #####

# Forecasting daily Realized volatility for the S&P 500 using the basic harModel: HARRV
library(xts)
RV_SP500 <- as.xts(realized_library$rv5, order.by = realized_library$date)

x <- harModel(data = RV_SP500 , periods = c(1,5,22), RVest = c("rCov"),
type = "HARRV", h = 1, transform = NULL, inputType = "RM")

class(x)
x
summary(x)
```
# Example 3: HARRVQ
dat <- sample_5minprices_jumps$stock1
dat <- makeReturns(dat) # Get the high-frequency return data
x <- harModel(dat, periods = c(1, 5, 10),
               periodsJ = c(1, 5, 10),
               periodsQ = c(1),
               RVest = c("rCov", "rQuar"),
               type = "HARRVQ", inputType = "returns")
# Estimate the HAR model of type HARRVQ
class(x)
x
# plot(x)
# predict(x)

# Example 4: HARRVQJ with already computed realized measures
dat <- SP500RM[, c("RV", "BPV", "RQ")]
x <- harModel(dat, periods = c(1, 5, 22), periodsJ = c(1),
               periodsQ = c(1), type = "HARRVQ")
# Estimate the HAR model of type HARRVQJ
class(x)
x
# plot(x)
predict(x)

# Example 5: CHARRV with already computed realized measures
dat <- SP500RM[, c("RV", "BPV")]
x <- harModel(dat, periods = c(1, 5, 22), type = "CHARRV")
# Estimate the HAR model of type CHARRV
class(x)
x
# plot(x)
predict(x)

# Example 6: CHARRVQ with already computed realized measures
dat <- SP500RM[, c("RV", "BPV", "RQ")]
x <- harModel(dat, periods = c(1, 5, 22), periodsQ = c(1), type = "CHARRVQ")
# Estimate the HAR model of type CHARRVQ
class(x)
x
# plot(x)
predict(x)

# Example 7: HARRV
# Forecasting weekly Realized volatility for the S&P 500 using the basic harModel: HARRV
library(xts)
RV_SP500 <- as.xts(realized_library$rv5, order.by = realized_library$date)
x <- harModel(data = RV_SP500, periods = c(1, 5, 22), RVest = c("rCov"),
               periodsQ = c(1, 5, 10),
               RVest = c("rCov", "rQuar"),
               type = "HARRVQ", inputType = "returns")
```
hasQty

    type = "HARRV", h = 5, transform = NULL, inputType = "RM")

    class(x)
    x
    summary(x)
    plot(x)
    predict(x)

---

**hasQty**

*Check for Trade, Bid, and Ask/Offer (BBO/TBBO), Quantity, and Price data*

**Description**

A set of functions to check for appropriate TBBO/BBO and price column names within a data object, as well as the availability and position of those columns.

**Usage**

    hasQty(x, which = FALSE)

**Arguments**

- **x** : data object
- **which** : display position of match

---

**heavyModel**

*HEAVY Model estimation*

**Description**

This function calculates the High frEquency bAsed Volatility (HEAVY) model proposed in Shephard and Sheppard (2010). This function is used as a predictive volatility model built to exploit high-frequency data.

**Usage**

    heavyModel(     
    data,     
    p = matrix(c(0, 0, 1, 1), ncol = 2),     
    q = matrix(c(1, 0, 0, 1), ncol = 2),     
    startingvalues = NULL,     
    LB = NULL,     
    UB = NULL,     
    backcast = NULL,     
    compconst = FALSE     
)
Arguments

**data**
a (T x K) matrix containing the data, with T the number of days. For the traditional HEAVY model: K = 2, the first column contains the squared daily demeaned returns, the second column contains the realized measures.

**p**
a (K x K) matrix containing the lag length for the model innovations. Position (i, j) in the matrix indicates the number of lags in equation i of the model for the innovations in data column j. For the traditional heavy model p is given by matrix(c(0,0,1,1), ncol = 2) (default).

**q**
a (K x K) matrix containing the lag length for the conditional variances. Position (i, j) in the matrix indicates the number of lags in equation i of the model for conditional variances corresponding to series j. For the traditional heavy model introduced above q is given by matrix( c(1,0,0,1),ncol=2 ) (default).

**startingvalues**
a vector containing the starting values to be used in the optimization to find the optimal parameters estimates.

**LB**
a vector of length K indicating the lower bounds to be used in the estimation. If NULL it is set to a vector of zeros by default.

**UB**
a vector of length K indicating the upper bounds to be used in the estimation. If NULL it is set to a vector of Inf by default.

**backcast**
a vector of length K used to initialize the estimation. If NULL the unconditional estimates are taken.

**compconst**
a boolean variable. In case TRUE, the omega values are estimated in the optimization. In case FALSE, volatility targeting is done and omega is just 1 minus the sum of all relevant alpha’s and beta’s multiplied by the unconditional variance.

Details

Assume there are T daily returns and realized measures in the period t. Let r_i and RM_i be the i\text{th} daily return and daily realized measure respectively (with i = 1, \ldots, T).

The most basic heavy model is the one with lag matrices p of \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} and q of \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. This can be represented by the following equations:

\[
\begin{align*}
\text{var}(r_t) &= h_t = w + \alpha RM_{t-1} + \beta h_{t-1}; w, \alpha \geq 0, \beta \in [0, 1] \\
E(RM_t) &= \mu_t = w_R + \alpha_R RM_{t-1} + \beta_R \mu_{t-1}; w_R, \alpha_R, \beta_R \geq 0, \alpha_R + \beta_R \in [0, 1]
\end{align*}
\]

Equivalently, they can be presented in terms of matrix notation as below:

\[
\begin{pmatrix} h_t \\ \mu_t \end{pmatrix} = \begin{pmatrix} w \\ w_R \end{pmatrix} + \begin{pmatrix} 0 & \alpha \\ 0 & \alpha_R \end{pmatrix} \begin{pmatrix} r_{t-1}^2 \\ RM_{t-1} \end{pmatrix} + \begin{pmatrix} \beta & 0 \\ 0 & \beta_R \end{pmatrix} \begin{pmatrix} h_{t-1} \\ \mu_{t-1} \end{pmatrix}
\]

In this version, the parameters vector to be estimated is \((w, w_R, \alpha, \alpha_R, \beta, \beta_R)\).

In terms of startingvalues, Shephard and Sheppard recommend for this version of the Heavy model to set \(\beta\) be around 0.6 and sum of \(\alpha + \beta\) to be close to but slightly less than one. In general, the lag length for the model innovation and the conditional covariance can be greater than 1. Consider,
for example, matrix p is \(
\begin{pmatrix}
0 & 2 \\
0 & 1
\end{pmatrix}
\) and matrix q is the same as above. Matrix notation will be as below:

\[
\begin{pmatrix}
h_t \\
\mu_t
\end{pmatrix}
= \begin{pmatrix}
w \\
w_R
\end{pmatrix} + \begin{pmatrix}
0 & \alpha_1 \\
0 & \alpha_R
\end{pmatrix} \begin{pmatrix}
r^2_{t-1} \\
RM_{t-1}
\end{pmatrix} + \begin{pmatrix}
0 & \alpha_2 \\
0 & 0
\end{pmatrix} \begin{pmatrix}
r^2_{t-2} \\
RM_{t-2}
\end{pmatrix} + \begin{pmatrix}
\beta & 0 \\
0 & \beta_R
\end{pmatrix} \begin{pmatrix}
h_{t-1} \\
\mu_{t-1}
\end{pmatrix}
\]

In this version, the parameters vector to be estimated is \((w, w_R, \alpha_1, \alpha_R, \alpha_2, \beta, \beta_R)\).

**Value**

A list with the following values: (i) loglikelihood: the log likelihood evaluated at the parameter estimates. (ii) likelihoods: an xts object of length T containing the log likelihoods per day. (iii) condvar: a \((T \times K)\) xts object containing the conditional variances (iv) estparams: a vector with the parameter estimates. The order in which the parameters are reported is as follows: First the estimates for omega then the estimates for the non-zero alpha’s with the most recent lags first in case \(\max(p) > 1\), then the estimates for the non-zero beta’s with the most recent lag first in case \(\max(q) > 1\). (v) convergence: an integer code indicating the successfulness of the optimization. See optim for more information.

**Author(s)**

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

**References**


**Examples**

```r
# Implementation of the heavy model on DJI:
returns <- realized_library$open_to_close
bv <- realized_library$bv
returns <- returns[!is.na(bv)]
bv <- bv[!is.na(bv)] # Remove NA's
data <- cbind(returns^2, bv) # Make data matrix with returns and realized measures
backcast <- matrix(c(var(returns), mean(bv)), ncol = 1)

# For traditional (default) version:
startvalues <- c(0.004, 0.02, 0.44, 0.41, 0.74, 0.56) # Initial values
output <- heavyModel(data = as.matrix(data, ncol = 2), compconst = FALSE, startingvalues = startvalues, backcast = backcast)

# For general version:
startvalues <- c(0.004, 0.02, 0.44, 0.4, 0.41, 0.74, 0.56) # Initial values;
p <- matrix(c(2, 0, 0, 1), ncol = 2)
q <- matrix(c(1, 0, 0, 1), ncol = 2)

heavy_model <- heavyModel(data = as.matrix(data, ncol = 2), p = p, q = q, compconst = FALSE, startingvalues = startvalues, backcast = backcast)
```
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, minRV, medRV. 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:

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

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

in which,

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

$cv$ : critical value.

$se$ : standard error.

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

$\hat{IQ}$ integrated quarticity estimator.

Usage

ivInference(
  rdata,
  IVestimator = "RV",
  IQestimator = "rQuar",
  confidence = 0.95,
  align.by = NULL,
  align.period = NULL,
  makeReturns = FALSE,
  ...
)

Arguments

rdata : zoo/xts object containing all returns in period $t$ for one asset.

IVestimator : can be chosen among integrated variance estimators: RV, BV, TV, minRV or medRV. RV by default.
Inference

IQestimator can be chosen among integrated quarticity estimators: rQuar, realized tri-power quarticity (TPQ), quad-power quarticity (QPQ), minRQ or medRQ. TPQ by default.

confidence confidence level set by users. 0.95 by default.

align.by a string, align the tick data to "seconds"|"minutes"|"hours"

align.period an integer, align the tick data to this many [seconds|minutes|hours].

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:

$$X_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u$$

where $a$ is the drift term, $\sigma$ denotes the spot volatility 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


Examples

ivInference(sample_tdata$PRICE, IVestimator= "minRV", IQestimator = "medRQ", confidence = 0.95, makeReturns = TRUE)
**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 JOjump test is given by:

\[
\text{JOjump}_{t, N} = \frac{N \text{BV}_t}{\sqrt{\Omega_{SwV}}} \left(1 - \frac{\text{RV}_t}{\text{SwV}_t}\right)
\]

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

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

\[
\Omega_{SwV} = \frac{\mu_6}{9} n_{N-p-1} \prod_{k=1}^{p} |r_{t,i+k}|^{6/p}
\]

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

\( U \): independent standard normal random variables

\( p \): parameter (power).

**Usage**

\[
\text{JOjump}(\text{pdata, power = 4, ...})
\]

**Arguments**

- \( \text{pdata} \) a zoo/xts object containing all prices in period \( t \) for one asset.
- \( \text{power} \) can be chosen among 4 or 6 by default.
- \( \ldots \) additional arguments.
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:

$$X_t = \int_0^t a_u du + \int_0^t \sigma_u dW_u + Z_t$$

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

$$Z_t = \sum_{j=1}^{N_t} k_j$$

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

The Jiang and Oomen test is that: in the absence of jumps, the accumulated difference between the simple return and the log return captures one half of the integrated variance. (Theodosiou & Zikes (2009))

Value

list

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

JOjumptest(sample_5minprices_jumps$stock1, power = 6)
### listAvailableKernels

#### Available Kernels

**Description**

Returns a vector of the available kernels.

**Usage**

```r
listAvailableKernels()
```

**Value**

character vector

**Author(s)**

Scott Payseur

**References**


**Examples**

```r
listAvailableKernels
```

---

### lltc

#### LLTC Data

**Description**

Tick data for LLTC 2011/07/01, cleaned with tradesCleanup.

**Usage**

```r
lltc
```

**Format**

xts object

**Examples**

```r
data(lltc)
plot(lltc)
```
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 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 $\Gamma$ be the orthogonal matrix consisting of the $p$ eigenvectors of $S$. Denote $\lambda_1^+, \ldots, \lambda_p^+$ its $p$ eigenvalues, whereby the negative eigenvalues have been replaced by zeroes. Under this approach, the positive semi-definite projection of $S$ is $S^+ = \Gamma' \text{diag}(\lambda_1^+, \ldots, \lambda_p^+) \Gamma$.

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

Value

An xts object containing the aggregated trade data.

Author(s)

Jonathan Cornelissen and Kris Boudt

References

**makeReturns**  
*Compute log returns*

**Description**  
Function returns an xts object with the log returns as xts object.

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

**Usage**  
makeReturns(ts)

**Arguments**  
- **ts**  
  xts object

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

**Value**  
an xts object containing the log returns.

**Author(s)**  
Jonathan Cornelissen and Kris Boudt

---

**matchTradesQuotes**  
*Match trade and quote data*

**Description**  
Function matches the trades and quotes and returns an xts-object containing both.

**Usage**  
matchTradesQuotes(tdata, qdata, adjustment = 2)

**Arguments**  
- **tdata**  
  data.table or xts-object containing the trade data (multiple days possible).
- **qdata**  
  data.table or xts-object containing the quote data (multiple days possible).
- **adjustment**  
  numeric, number of seconds the quotes are registered faster than the trades (should be round and positive). Based on the research of Vergote (2005), we set 2 seconds as the default.
**medRQ**

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

**Value**

data.table or xts-object containing the matched trade and quote data

**Author(s)**

Jonathan Cornelissen, Kris Boudt and Onno Kleen

**References**


**Examples**

```r
# match the trade and quote data
tqdata <- matchTradesQuotes(sample_tdata, sample_qdata)
head(tqdata)
# multi-day input allowed
tqdata <- matchTradesQuotes(sample_tdata_microseconds, sample_qdata_microseconds)
```

**Description**

Function returns the medRQ, defined in Andersen et al. (2012).

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

Then, the medRQ is given by

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

**Usage**

```r
medRQ(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```

**Arguments**

- `rdata` a zoo/xts object containing all returns in period $t$ for one asset.
- `align.by` a string, align the tick data to “seconds”/“minutes”/“hours”.
- `align.period` an integer, align the tick data to this many [seconds/minutes/hours].
- `makeReturns` boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.
Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

```r
## Not run:
medRQ(rdata = sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)
medRQ

## End(Not run)
```

Description

Function returns the medRV, defined in Andersen et al. (2009).

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

Then, the medRV is given by

\[
medRV_t = \frac{\pi}{6 - 4\sqrt{3} + \pi} \left( \frac{M}{M - 2} \right) \sum_{i=2}^{M-1} \text{med}(|r_{t,i-1}|, |r_{t,i}|, |r_{t,i+1}|)^2
\]

Usage

```r
medRV(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```

Arguments

- `rdata`: a zoo/xts object containing all returns in period \( t \) for one asset.
- `align.by`: a string, align the tick data to “seconds”/“minutes”/“hours”.
- `align.period`: an integer, align the tick data to this many [seconds/minutes/hours].
- `makeReturns`: boolean, should be TRUE when `rdata` contains prices instead of returns. FALSE by default.
Details

The medRV 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 \( t \). medRV is designed to be robust to price jumps. The difference between RV and medRV 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. (2007) and Corsi et al. (2010).

Value

numeric

Author(s)

Jonathan Cornelissen and Kris Boudt

References


Examples

data(sample_tdata);
medrv <- medRV(rdata = sample_tdata$PRICE, align.by = "minutes",
align.period = 5, makeReturns = TRUE)
medrv

mergeQuotesSameTimestamp

Merge multiple quote entries with the same time stamp

Description

Function replaces multiple quote entries that have the same time stamp by a single one and returns an xts 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 and 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

xts or data.table object depending on input

Author(s)

Jonathan Cornelissen and Kris Boudt

mergeTradesSameTimestamp

Merge multiple transactions with the same time stamp

Description

Function replaces multiple transactions that have the same time stamp by a single one and returns an xts or 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.
**minRQ**

**Value**

data.table or xts object depending on input

**Author(s)**

Jonathan Cornelissen and Kris Boudt

---

**Description**

Function returns the minRQ, defined in Andersen et al. (2012).

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

Then, the minRQ is given by

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

**Usage**

```r
minRQ(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```

**Arguments**

- `rdata` a zoo/xts object containing all returns in period $t$ for one asset.
- `align.by` a string, align the tick data to "seconds""minutes""hours"
- `align.period` an integer, align the tick data to this many [seconds|minutes|hours].
- `makeReturns` boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

**Value**

numeric

**Author(s)**

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

**References**

Examples

```r
## Not run:
minRQ(rdata = sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)
minRQ

## End(Not run)
```

minRV

Description

Function returns the minRV, defined in Andersen et al. (2009).

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

Then, the minRV is given by

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

Usage

```r
minRV(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```

Arguments

- `rdata`: a zoo/xts object containing all returns in period \( t \) for one asset.
- `align.by`: a string, align the tick data to "seconds"/"minutes"/"hours".
- `align.period`: an integer, align the tick data to this many [seconds/minutes/hours].
- `makeReturns`: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

Value

numeric

Author(s)

Jonathan Cornelissen and Kris Boudt

References

Examples

```r
data(sample_tdata)
minrv <- minRV(rdata = sample_tdata$PRICE, align.by = "minutes",
               align.period = 5, makeReturns = TRUE)
minrv
```

R

Modulated Realized Covariance (MRC): Return univariate or multivariate preaveraged estimator.

Description

Function returns univariate or multivariate preaveraged estimator, as defined in Hautsch and Podolskij (2013).

Usage

```r
MRC(pdata, pairwise = FALSE, makePsd = FALSE)
```

Arguments

- `pdata` a list. Each list-item contains an xts 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 MRC is returned. FALSE by default.

Details

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

$$ Y_\tau = X_\tau + \epsilon_\tau $$

where, the observed $d$ dimensional log-prices are the sum of underlying Brownian semimartingale process $X$ and a noise term $\epsilon_\tau$.

$\epsilon_\tau$ 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 $\tau$ of a list (after refreshing data). Let $r_{\tau_i}$ be a return (with $i = 1, \ldots, N$) of an asset in period $\tau$. Assume there is $d$ assets.
In order to define the univariate pre-averaging estimator, we first define the pre-averaged returns as

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

where \( g \) is a non-zero real-valued function \( g : [0, 1] \rightarrow \mathbb{R} \) given by \( g(x) = \min(x, 1 - x) \). \( k_N \) is a sequence of integers satisfying \( k_N = \lfloor \theta N^{1/2} \rfloor \). We use \( \theta = 0.8 \) as recommended in Hautsch & 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

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

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 (MRC) and is defined as

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

where \( \hat{\Psi}_N = \frac{1}{2N} \sum_{i=1}^{N} r_{\tau_i} (r_{\tau_i})' \). 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 \( k_N = \lfloor \theta N^{1/2 + \delta} \rfloor \). \( \delta = 0.1 \) results in a consistent estimate without the bias correction and a PSD estimate, in which case:

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

Value

an \( d \times d \) matrix

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt
References

Examples
```r
a <- list(sample_5minprices_jumps["2010-01-04",1], sample_5minprices_jumps["2010-01-04",2])
MRC(a, pairwise = TRUE, makePsd = TRUE)
```

---

**noZeroPrices**

*Delete the observations where the price is zero*

**Description**
Function deletes the observations where the price is zero.

**Usage**
```r
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**
```r
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

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 steps are performed sequentially: noZeroQuotes, selectExchange, rm-LargeSpread, mergeQuotesSameTimestamp, rmOutliersQuotes.

Usage

quotesCleanup(
  datasource = NULL,
  datadestination = NULL,
  exchanges,
  qdataraw = NULL,
  report = TRUE,
  selection = "median",
  maxi = 50,
  window = 50,
  type = "advanced",
  rmoutliersmaxi = 10,
  saveasxts = TRUE
)

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.
quotesCleanup

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

qdataraw xts or data.table object containing (ONE stock only) raw quote 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 quotes remained after 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 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.

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

saveasxts indicates whether data should be saved in xts format instead of data.table when using on-disk functionality. TRUE by default.

Value

The function converts every csv 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 and Onno Kleen.
References

Examples
# Consider you have raw quote data for 1 stock for 2 days
head(sample_qdata_raw_microseconds)
dim(sample_qdata_raw_microseconds)
qdata_aftercleaning <- quotesCleanup(qdata = sample_qdata_raw_microseconds, exchanges = "N")
qdata_aftercleaning$report
dim(qdata_aftercleaning$qdata)

# In case you have more data it is advised to use the on-disk functionality
# via "from","to","datasource",etc. arguments

---

rAVGCov

Realized Covariance: Average Subsample

Description
Realized Covariance using average subsample.

Usage
rAVGCov(
  rdata,  
cor = FALSE,  
align.by = "minutes",  
align.period = 5,  
makeReturns = FALSE
)

Arguments
rdata a \((M.x.N)\) matrix/zoo/xts object containing the \(N\) return series over period \(t\), with \(M\) observations during \(t\).
cor boolean, in case it is TRUE, the correlation is returned. FALSE by default.
align.by Align the tick data to seconds/min/minutes/hours
align.period Align the tick data to this many [seconds/min/minutes/hours]
makeReturns Prices are passed make them into log returns
Value

Realized covariance using average subsample.

Author(s)

Scott Payseur

References


Michiel de Pooter, Martin Martens, and Dick van Dijk. Predicting the daily covariance matrix for S&P100 stocks using intraday data - but which frequency to use? *Econometric Reviews*, 2008.

Examples

# Average subsampled realized variance/covariance aligned at one minute returns at
# 5 subgrids (5 minutes).

# Univariate
rvSub <- rAVGCov(rdata = sample_tdata$PRICE, align.by = "minutes",
                      align.period = 5, makeReturns = TRUE)
rvSub

# Multivariate:
rcovSub <- rAVGCov(rdata = cbind(lltc, sbux, fill = 0), align.by = "minutes",
                      align.period = 5, makeReturns = FALSE)
rcovSub

---

**rBeta**

*Realized beta: a tool in measuring risk with respect to the market.*

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

\begin{verbatim}
rBeta(  rdata,  rindex,  RCOVestimator = "rCov",  RVestimator = "RV",  makeReturns = FALSE )
\end{verbatim}

Arguments

- **rdata**: a zoo/xts object containing all returns in period t for one asset.
- **rindex**: a zoo/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: RV, minRV and medRV. RV 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.

Details

Suppose there are \( N \) equispaced returns on day \( t \) for the asset \( j \) and the index \( m \). Denote \( r_{(j)i,t} \) and \( r_{(m)i,t} \) as the \( i \)th 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:

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

(Barndorff & Shephard (2004)).

Note: It is worth to note that the function does not support to calculate for data of multiple days.

Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References

Examples

\begin{verbatim}
a <- sample_5minprices_jumps['2010-01-04', 1]
b <- sample_5minprices_jumps['2010-01-04', 2]
rBeta(a, b, RCOVestimator = "rBPCov", RVestimator = "minRV", makeReturns = TRUE)
\end{verbatim}

Description

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

Let \( r_{t,i} \) be an intraday \( N \times 1 \) 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

\begin{verbatim}
rBPCov(rdata, cor = FALSE, align.by = NULL, align.period = NULL, makeReturns = FALSE, makePsd = FALSE)
\end{verbatim}

Arguments

- **rdata**: a \((M \times N)\) matrix/zoo/xts object containing the \( N \) return series over period \( t \), with \( M \) observations during \( t \).
- **cor**: boolean, in case it is TRUE, the correlation is returned. FALSE by default.
- **align.by**: a string, align the tick data to "seconds"|"minutes"|"hours".
- **align.period**: an integer, align the tick data to this many [seconds|minutes|hours].
- **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.
Value
an $N \times N$ matrix or a list of matrices if the time period spans multiple days

Author(s)
Jonathan Cornelissen and Kris Boudt

References

Examples
# Realized Bipower Variance/Covariance for CTS aligned
# at 5 minutes.

# Univariate:
rbpv <- rBPCov(rdata = sample_tdata$PRICE, align.by = "minutes",
               align.period = 5, makeReturns = TRUE)
rbpv

# Multivariate:
rbpc <- rBPCov(rdata = sample_5minprices_jumps['2010-01-04'], makeReturns = TRUE, makePsd = TRUE)
rbpc

---

rCov

Realized Covariance

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

Then, the rCov is given by

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

Usage
rCov(
    rdata,
    cor = FALSE,
    align.by = NULL,
    align.period = NULL,
    makeReturns = FALSE
)
Arguments

rdata: a \((M \times N)\) matrix/zoo/xts object containing the \(N\) return series over period \(t\), with \(M\) observations during \(t\). In case of a matrix, no multi-day adjustment is possible.

cor: boolean, in case it is TRUE, the correlation is returned. FALSE by default.

align.by: a string, align the tick data to "seconds","minutes","hours".

align.period: an integer, align the tick data to this many [seconds|minutes|hours].

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

Value

an \(N \times N\) matrix

Author(s)

Jonathan Cornelissen and Kris Boudt

Examples

# Realized Variance/Covariance for CTS aligned
# at 5 minutes.
data(sample_tdata)
data(sample_5minprices_jumps)

# Univariate:
rv = rCov(rdata = sample_tdata$PRICE, align.by = "minutes",
     align.period = 5, makeReturns = TRUE)
rv

# Multivariate:
rc = rCov(rdata = sample_5minprices_jumps['2010-01-04'], makeReturns=TRUE)
rc

realized_library  The realized library from the Oxford-Man Institute of Quantitative Finance

Description

A data.frame object containing the daily open-close returns, daily realized variances based on five-minute intraday returns and daily realized kernels ranging from 2000-01-03 up to 2019-06-10 for the S&P 500. Use colnames(realized_library) to see which realized measures. The full library of the Oxford-Man Institute of Quantitative Finance can be found on their website: http://realized.oxford-man.ox.ac.uk.
refreshTime

Usage

realized_library

Format

data.frame

References


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)

Arguments

pdata | a list. Each list-item contains an xts object containing the original time series (one day only and typically a price series).

Value

An xts object containing the synchronized time series.

Author(s)

Jonathan Cornelissen and Kris Boudt

References

Examples

```r
# 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))
```

---

**rHYCov**

*Hayashi-Yoshida Covariance*

---

**Description**

Calculates the Hayashi-Yoshida Covariance estimator

**Usage**

```r
rHYCov(
  rdata, cor = FALSE, period = 1, align.by = "seconds",
  align.period = 1, makeReturns = FALSE, makePsd = TRUE
)
```

**Arguments**

- `rdata`: a possibly multivariate xts object.
- `cor`: boolean, in case it is TRUE, the correlation is returned. FALSE by default.
- `period`: Sampling period
- `align.by`: Align the tick data to seconds|minutes|hours
- `align.period`: Align the tick data to this many [seconds|minutes|hours]
- `makeReturns`: Prices are passed make them into log returns
- `makePsd`: boolean, in case it is TRUE, the positive definite version of rHYCov is returned. FALSE by default.
rKernelCov

Author(s)

Scott Payseur

# Average Hayashi-Yoshida Covariance estimator is calculated on five-minute returns
# Multivariate: # realized_cov <- rHYCov(rdata = cbind(lltc, sbux, fill = 0), period = 5, align.by = "minutes", # align.period = 5, makeReturns = FALSE) # realized_cov Note: for the diagonal elements the rCov is used.

References


---

rKernelCov

Realized Covariance: Kernel

Description

Realized covariance calculation using a kernel estimator.

Usage

rKernelCov(
  rdata,
  cor = FALSE,
  align.by = "seconds",
  align.period = 1,
  makeReturns = FALSE,
  kernel.type = "rectangular",
  kernel.param = 1,
  kernel.dofadj = TRUE
)

Arguments

rdata a (MxN) matrix/zoo/xts object containing the N return series over period t, with M observations during t.

cor boolean, in case it is TRUE, the correlation is returned. FALSE by default.

align.by Align the tick data to seconds/minutes/hours

align.period Align the tick data to this many [seconds/minutes/hours]

makeReturns Convert to Returns

kernel.type Kernel name (or number)

kernel.param Kernel parameter (usually lags)

kernel.dofadj Kernel Degree of freedom adjustment
Details

The different types of kernels can be found using `listAvailableKernels`.

Value

Kernel estimate of realized covariance.

Author(s)

Scott Payseur and Onno Kleen

References


Examples

```r
# Univariate:
rvKernel <- rKernelCov(rdata = sample_tdata$PRICE, align.by = "minutes",
                        align.period = 5, makeReturns = TRUE)
rvKernel

# Multivariate:
rcKernel <- rKernelCov(rdata = cbind(lltc, sbux, fill = 0), align.by = "minutes",
                        align.period = 5, makeReturns = FALSE)
rcKernel
```

---

**rKurt**

*Realized kurtosis of high-frequency return series.*

Description

Function returns Realized kurtosis, defined in Amaya et al. (2011).

Usage

```r
rKurt(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```
Arguments

rdata a zoo/xts object containing all returns in period t for one asset.
align.by a string, align the tick data to "seconds"/"minutes"/"hours".
align.period an integer, align the tick data to this many [seconds/minutes/hours].
makeReturns boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

Details

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

Then, the rKurt is given by

\[
rKurt_t = \frac{1}{N} \sum_{i=1}^{N} (r_{t,i})^4
\]

in which \( RV_t \): realized variance

Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

data(sample_tdata)
rKurt(sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)

---

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)
**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.

**Value**

xts or data.table object depending on input.

**Author(s)**

Jonathan Cornelissen and Kris Boudt

---

**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(sample_qdataraw_microseconds)`
Delete entries for which the mid-quote is outlying with respect to surrounding entries

Description

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" observations
3. Rolling median of the previous "window" 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.

Usage

rmOutliersQuotes(qdata, maxi = 10, window = 50, type = "advanced")

Arguments

qdata: an 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 description).

Details

NOTE: This function works only correct if supplied input data consists of 1 day.

Value

xts object or data.table depending on type of input

Author(s)

Jonathan Cornelissen and Kris Boudt
References

rMPV
Realized multipower variation (MPV), an estimator of integrated power variation.

Description
Function returns the rMPV, defined in Andersen et al. (2012).
Assume there is $N$ equispaced returns in period $t$. Let $r_{t,i}$ be a return (with $i = 1, \ldots, N$) in period $t$.
Then, the rMPV is given by
$$ r_{MPV}^{N(m,p)} = d_{m,p} \frac{Np/2}{N - m + 1} \sum_{i=1}^{N-m+1} |r_{t,i}|^{p/m} \ldots |r_{t,i+m-1}|^{p/m} $$
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
```r
rMPV(
  rdata,
  m = 2,
  p = 2,
  align.by = NULL,
  align.period = NULL,
  makeReturns = FALSE
)
```

Arguments
- **rdata**: a zoo/xts object containing all returns in period $t$ for one asset.
- **m**: the window size of return blocks. 2 by default.
- **p**: the power of the variation. 2 by default.
- **align.by**: a string, align the tick data to "seconds""minutes""hours".
- **align.period**: an integer, align the tick data to this many [seconds|minutes|hours].
- **makeReturns**: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.
Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

data(sample_tdata)
rMPV(sample_tdata$PRICE, m = 2, p = 3, align.by = "minutes", align.period = 5, makeReturns = TRUE)

---

rmTradeOutliers  Delete transactions with unlikely transaction prices

Description

Deprecated - use rmTradeOutliers instead.

Usage

rmTradeOutliers(tdata, qdata)

Arguments

tdata a data.table or xts object containing the time series data, with at least the column "PRICE", containing the transaction price (ONE DAY ONLY).

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 (ONE DAY ONLY).

Value

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

```r
rmTradeOutliersUsingQuotes(tdata, qdata)
```

**Arguments**

- `tdata`: a data.table or xts object containing the time series data, with at least the column "PRICE", containing the transaction price (ONE DAY ONLY).
- `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 (ONE DAY ONLY).

**Details**

Note: in order to work correctly, the input data of this function should be cleaned trade (tdata) and quote (qdata) data respectively.

**Value**

xts or data.table object depending on input

**Author(s)**

Jonathan Cornelissen, Kris Boudt and Onno Kleen

---

**rOWCov**

**Realized Outlyingness Weighted Covariance**

**Description**

Function returns the Realized Outlyingness Weighted Covariance, 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

$$
r_{tOWCov} = c_w \frac{N}{M} \sum_{i=1}^{M} w(d_{t,i}) r_{t,i} r_{t,i}'
$$
The weight $w_{ij,\Delta}$ is one if the multivariate jump test statistic for $r_{ij,\Delta}$ 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

```r
rOWCov(
  rdata,
  cor = FALSE,
  align.by = NULL,
  align.period = NULL,
  makeReturns = FALSE,
  seasadjR = NULL,
  wfunction = "HR",
  alphaMCD = 0.75,
  alpha = 0.001
)
```

Arguments

- **rdata**: a (MxN) matrix/zoo/xts object containing the $N$ return series over period $t$, with $M$ observations during $t$.
- **cor**: boolean, in case it is TRUE, the correlation is returned. FALSE by default.
- **align.by**: a string, align the tick data to "seconds""|"minutes"|"hours".
- **align.period**: an integer, align the tick data to this many [seconds|minutes|hours].
- **makeReturns**: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.
- **seasadjR**: a (MxN) matrix/zoo/xts object containing the seasonaly adjusted returns. This is an optional argument.
- **wfunction**: determines whether a zero-one weight function (one if no jump is detected based on $d_{ij}$ 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 en 0.5, that determines the rejection threshold value (see Boudt et al. (2008) for details).

Details

Advantages of the rOWCov compared to the rBPCov include a higher statistical efficiency, positive semidefiniteness and affine equivariance. However, the rOWCov suffers from a curse of dimensionality. Indeed, 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.
**rQPVar**

**Value**

an $N \times N$ matrix

**Author(s)**

Jonathan Cornelissen and Kris Boudt

**References**


**Examples**

```r
# Realized Outlyingness Weighted Variance/Covariance for CTS aligned
# at 5 minutes.

# Univariate:
routw <- rOWCov(rdata = sample_tdata$PRICE, align.by = "minutes",
                 align.period = 5, makeReturns = TRUE)
routw

# Multivariate:
routw <- rOWCov(rdata = sample_5minprices_jumps['2010-01-04'], makeReturns = TRUE)
routw
```

---

**Description**

Function returns the realized quad-power variation, defined in Andersen et al. (2012).

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

Then, the rQPVar is given by

$$rQPVar_t = N \times \frac{N}{N-3} \left( \frac{\pi^2}{4} \right)^{-4} (|r_{t,i}| |r_{t,i-1}| |r_{t,i-2}| |r_{t,i-3}|)$$

**Usage**

```r
rQPVar(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```
Arguments

rdata a zoo/xts object containing all returns in period t for one asset.
align.by a string, align the tick data to "seconds"|"minutes"|"hours".
align.period an integer, align the tick data to this many [seconds|minutes|hours].
makeReturns boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

Value
numeric

Author(s)
Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

data(sample_tdata)
rQPVar(rdata= sample_tdata$PRICE, align.by= "minutes", align.period =5, makeReturns= TRUE)

rQuar
Realized quarticity of highfrequency return series.

Description

Function returns the rQuar, defined in Andersen et al. (2012).

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

Then, the rQuar is given by

\[
\text{rQuar}_t = \frac{N}{3} \sum_{i=1}^{N} (r_{t,i}^4)
\]

Usage

rQuar(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
**Arguments**

- **rdata**: a zoo/xts object containing all returns in period \( t \) for one asset.
- **align.by**: a string, align the tick data to "seconds""minutes""hours".
- **align.period**: an integer, align the tick data to this many [seconds|minutes|hours].
- **makeReturns**: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

**Value**

numeric

**Author(s)**

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

**References**


**Examples**

```r
## Not run:
data(sample_tdata)
rQuar(rdata = sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)
## End(Not run)
```

---

**rRTSCov**

*Robust two time scale covariance estimation*

**Description**

Function returns 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**

```r
rRTSCov(
  pdata,
  cor = FALSE,
  startIV = NULL,
  noisevar = NULL,
  K = 300,
)```
J = 1,  
K_cov = NULL,  
J_cov = NULL,  
K_var = NULL,  
J_var = 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, the correlation is returned. 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.
K_cov  
positive integer, for the extradiagonal covariance elements the slow time scale returns are computed on prices that are K steps apart.
J_cov  
positive integer, for the extradiagonal covariance elements the fast time scale returns are computed on prices that are J steps apart.
K_var  
vector of positive integers, for the diagonal variance elements the slow time scale returns are computed on prices that are K steps apart.
J_var  
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.

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 $\tau_1, \tau_2, \ldots, \tau_n$ as follows:

$$(1 - \frac{nK}{nJ})^{-1} \{(X, X)^{(K)}_T - \frac{nK}{nJ} (X, X)^{(J)}_T\},$$
where $\bar{n}_K = (n - K + 1)/K$, $\bar{n}_J = (n - J + 1)/J$ and

$$\{X, X\}^{(K)}_T = \frac{c_\eta}{K} \sum_{i=1}^{n-K+1} (X_{t_i+K} - X_{t_i})^2 I_X^K(i; \eta).$$

The constant $c_\eta$ adjusts for the bias due to the thresholding and $I_X^K(i; \eta)$ is a jump indicator function that is one if

$$\frac{(X_{t_{i+K}} - X_{t_i})^2}{(\int_{t_i}^{t_{i+K}} \sigma^2 ds + 2\sigma^2 \varepsilon)^2} \leq \eta$$

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 $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_{N_X}\}$ and $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{N_Y}\}$ 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. $\phi_{j+1} = \max(\tau_{N_X+1}, \theta_{N_Y+1})$. The complete refresh time sample grid is $\Phi = \{\phi_1, \phi_2, \ldots, \phi_{MN+1}\}$, where $MN$ 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:

$$c_N(\{X, Y\}^{(K)}_T) = \frac{\bar{n}_K}{\bar{n}_J} \{X, Y\}^{(J)}_T,$$

where

$$\{X, Y\}^{(K)}_T = \frac{1}{K} \sum_{i=1}^{MN-K+1} c_i(X_{t_{i+K}} - X_{t_i})(Y_{s_{i+K}} - Y_{s_i}) I_X^K(i; \eta) I_Y^K(i; \eta),$$

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 and Kris Boudt
**References**


**Examples**

```r
# Robust Realized two timescales Variance/Covariance
data(sample_tdata)
# Univariate:
rvRTS <- rRTSCov(pdata = sample_tdata$PRICE)
# Note: Prices as input
rvRTS

# Multivariate:
rcRTS <- rRTSCov(pdata = list(cumsum(lltc) + 100, cumsum(sbux) + 100))
# Note: List of prices as input
rcRTS
```

---

**rSkew**

*Realized skewness of highfrequency return series.*

**Description**

Function returns Realized skewness, defined in Amaya et al. (2011).

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

Then, the rSkew is given by

$$
rSkew_t = \frac{\sqrt{N} \sum_{i=1}^{N} (r_{t,i})^3}{RV_t^{3/2}}
$$

in which $RV_t$: realized variance

**Usage**

```r
rSkew(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```
Arguments

rdata a zoo/xts object containing all returns in period t for one asset.
align.by a string, align the tick data to "seconds"|"minutes"|"hours".
align.period an integer, align the tick data to this many [seconds|minutes|hours].
makeReturns boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

Value

numeric

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

data(sample_tdata)
rSkew(sample_tdata$PRICE, align.by ="minutes", align.period =5, makeReturns = TRUE)

Description

Function returns realized semivariances, defined in Barndorff-Nielsen et al. (2008).
Function returns two outcomes: 1. Downside realized semivariance and 2. Upside realized semivariance.
Assume there is \( N \) equispaced returns in period \( t \). Let \( r_{t,i} \) be a return (with \( i = 1, \ldots, N \)) in period \( t \).
Then, the rSV is given by

\[
\text{rSV}_{\text{downside}}_t = \sum_{i=1}^{N} (r_{t,i})^2 \times I[r_{t,i} < 0]
\]

\[
\text{rSV}_{\text{upside}}_t = \sum_{i=1}^{N} (r_{t,i})^2 \times I[r_{t,i} > 0]
\]
Usage

\texttt{rSV(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)}

Arguments

- \texttt{rdata}:
  a zoo/xts object containing all returns in period \( t \) for one asset.
- \texttt{align.by}:
  a string, align the tick data to "seconds"|"minutes"|"hours".
- \texttt{align.period}:
  an integer, align the tick data to this many [seconds|minutes|hours].
- \texttt{makeReturns}:
  boolean, should be \texttt{TRUE} when \texttt{rdata} contains prices instead of returns. \texttt{FALSE} by default.

Value

list with to arguments. The realized positive and negative semivariance.

Author(s)

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

References


Examples

```r
## Not run:
data(sample_tdata)
rSV(sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)
## End(Not run)
```

---

**rThresholdCov**

**Threshold Covariance**

Description

Function returns the threshold covariance matrix proposed in Gobbi and Mancini (2009). Unlike the \texttt{rOWCov}, the \texttt{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 \( N \times 1 \) return vector and \( i = 1, \ldots, M \) the number of intraday returns.

Then, the \( k, q \)-th element of the threshold covariance matrix is defined as

\[
\text{thresholdcov}[k, q]_t = \sum_{i=1}^{M} r_{(k)_{t,i}} 1\{r_{(k)_{t,i}}^2 \leq TR_M\} r_{(q)_{t,i}} 1\{r_{(q)_{t,i}}^2 \leq TR_M\},
\]

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).
**rThresholdCov**

**Usage**

```r
rThresholdCov(
  rdata,
  cor = FALSE,
  align.by = NULL,
  align.period = NULL,
  makeReturns = FALSE
)
```

**Arguments**

- `rdata`: a \((M \times N)\) matrix/zoo/xts object containing the \(N\) return series over period \(t\), with \(M\) observations during \(t\).
- `cor`: boolean, in case it is TRUE, the correlation is returned. FALSE by default.
- `align.by`: a string, align the tick data to "seconds","minutes","hours".
- `align.period`: an integer, align the tick data to this many [seconds|minutes|hours].
- `makeReturns`: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

**Value**

an \(N \times N\) matrix

**Author(s)**

Jonathan Cornelissen and Kris Boudt

**References**


**Examples**

```r
# Realized threshold Variance/Covariance:
# Multivariate:
rcThreshold <- rThresholdCov(cbind(lltc, sbux), align.by = "minutes", align.period = 1)
rcThreshold
```
<table>
<thead>
<tr>
<th><strong>rTPVar</strong></th>
<th>Realized tri-power variation estimator of quarticity for a high-frequency return series.</th>
</tr>
</thead>
</table>

**Description**

Function returns the rTPVar, defined in Andersen et al. (2012).

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

Then, the rTPVar is given by

$$
rTPVar_t = N \frac{N}{N-2} \left( \frac{\Gamma(0.5)}{2^{2/3} \Gamma(7/6)} \right)^{2/3} \sum_{i=3}^{N} (|r_{t,i}|^{4/3}|r_{t,i-1}|^{4/3}|r_{t,i-2}|^{4/3})
$$

**Usage**

```r
rTPVar(rdata, align.by = NULL, align.period = NULL, makeReturns = FALSE)
```

**Arguments**

- `rdata`: a zoo/xts object containing all returns in period $t$ for one asset.
- `align.by`: a string, align the tick data to "seconds"|"minutes"|"hours".
- `align.period`: an integer, align the tick data to this many [seconds|minutes|hours].
- `makeReturns`: boolean, should be TRUE when rdata contains prices instead of returns. FALSE by default.

**Value**

numeric

**Author(s)**

Giang Nguyen, Jonathan Cornelissen and Kris Boudt

**References**


**Examples**

```r
data(sample_tdata)
rTPVar(rdata = sample_tdata$PRICE, align.by = "minutes", align.period = 5, makeReturns = TRUE)
rTPVar
```
RTQ Calculate the realized tripower quarticity

Description

Calculate the realized tripower quarticity

Usage

RTQ(rdata)

Arguments

rdata a zoo/xts object containing all returns in period t for one asset.

Value

numeric

rTSCov Two time scale covariance estimation

Description

Function returns 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,
K_cov = NULL,
J_cov = NULL,
K_var = NULL,
J_var = 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, the correlation is returned. 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.
- `K_cov`: positive integer, for the extradiagonal covariance elements the slow time scale returns are computed on prices that are K steps apart.
- `J_cov`: positive integer, for the extradiagonal covariance elements the fast time scale returns are computed on prices that are J steps apart.
- `K_var`: vector of positive integers, for the diagonal variance elements the slow time scale returns are computed on prices that are K steps apart.
- `J_var`: 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.

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:

$$(1 - \frac{\pi_K}{\pi_J})^{-1}([X, X]_T^{[K]} - \frac{\pi_K}{\pi_J} [X, X]_T^{[J]}),$$

where $\pi_K = (n - K + 1)/K$, $\pi_J = (n - J + 1)/J$ and

$$[X, X]_T^{[K]} = \frac{1}{K} \sum_{i=1}^{n-K+1} (X_{i+K} - X_i)^2.$$

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 $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_{N^X}\}$ and $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{N^Y}\}$ 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. $\phi_{j+1} = \max(\tau_{N^X_{\phi_j}}, \theta_{N^Y_{\phi_j} + 1})$. The complete refresh time sample grid is $\Phi = \{\phi_1, \phi_2, \ldots, \phi_{M_N+1}\}$, 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:

\[ c_N (\langle X, Y \rangle_T^{(K)}) - \frac{\pi K}{\pi_J} (X, Y \rangle_T^{(J)}) \],

where

\[ [X, Y]_T^{(K)} = \frac{1}{K} \sum_{i=1}^{M_N - K + 1} (X_{t+i} - X_t) (Y_{s+i} - Y_s) \].

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**

an \( N \times N \) matrix

**Author(s)**

Jonathan Cornelissen and Kris Boudt

**References**


**Examples**

# Robust Realized two timescales Variance/Covariance for CTS

# Univariate:
rvts <- rTSCov(pdata = sample_tdata$PRICE)

# Note: Prices as input
rvts

# Multivariate:
rcovts <- rTSCov(pdata = list(cumsum(lltc) + 100, cumsum(sbux) + 100))

# Note: List of prices as input
rcovts
**RV**

*An estimator of realized variance.*

**Description**

An estimator of realized variance.

**Usage**

RV(rdata)

**Arguments**

- rdata: a zoo/xts object containing all returns in period t for one asset.

**Value**

numeric

---

**salesCondition**

*Delete entries with abnormal Sale Condition.*

**Description**

Function deletes entries with abnormal Sale Condition: trades where column "COND" has a letter code, except for "E" and "F".

**Usage**

salesCondition(tdata)

**Arguments**

- tdata: an xts or data.table object containing the time series data, with one column named "COND" indicating the Sale Condition.

**Value**

xts or data.table object depending on input

**Author(s)**

Jonathan Cornelissen and Kris Boudt
sample_5minprices

Description

Ten artificial time series for the NYSE trading days during January 2010

Ten hypothetical price series were simulated according to the factor diffusion process discussed in Barndorff-Nielsen et al. We assume that prices are only observed when a transaction takes place. The intensity of transactions follows a Poisson process and consequently, the inter-transaction times are exponentially distributed. Therefore, we generated the inter transaction times of the price series by an independent exponential distributions with lambda = 0.1, which we keep constant over time. This means we expect one transaction every ten seconds. In a final step, the time series were aggregated to the 5-minute frequency by previous tick aggregation.

Usage

sample_5minprices

Format

xts object

References


sample_5minprices_jumps

Description

Ten artificial time series (including jumps) for the NYSE trading days during January 2010

Ten simulated price series for the 19 trading days in January 2010: Ten hypothetical price series were simulated according to the factor diffusion process discussed in Barndorff-Nielsen et al. On top of this process we added a jump process, with jump occurrences governed by the Poisson process with 1 expected jump per day and jump magnitude modelled as in Boudt et al. (2008). We assume that prices are only observed when a transaction takes place. The intensity of transactions follows a Poisson process and consequently, the inter transaction times are exponentially distributed. Therefore, we generated the inter transaction times of the price series by an independent exponential distributions with lambda = 0.1, which we keep constant over time. This means we expect one transaction every ten seconds. In a final step, the time series were aggregated to the 5-minute frequency by previous tick aggregation.
Usage

sample_5minprices_jumps

Format

xts object

References


---

**sample_qdata**

*Sample of cleaned quotes for stock XXX for 1 day*

Description

An xts object containing the raw quotes for the imaginary stock XXX for 1 day, in the typical NYSE TAQ database format. This is the cleaned version of the data sample `sample_qdataraw`, using `quotesCleanup`.

Usage

`sample_qdata`

Format

`xts object`

---

**sample_qdataraw**

*Sample of raw quotes for stock XXX for 1 day*

Description

An imaginary xts object containing the raw quotes for stock XXX for 1 day, in the typical NYSE TAQ database format.

Usage

`sample_qdataraw`

Format

`xts object`
sample_qdataraw_microseconds

Sample of raw quotes for stock XXX for 2 days measured in microseconds

---

Description

An imaginary xts object containing the raw quotes for stock XXX for 2 days, in the typical NYSE TAQ database format.

Usage

```
sample_qdataraw_microseconds
```

Format

data.table object

---

sample_qdata_microseconds

Sample of cleaned quotes for stock XXX for 2 days measured in microseconds

---

Description

A data.table object containing the raw quotes for the imaginary stock XXX for 2 days. This is the cleaned version of the data sample `sample_qdataraw_microseconds`, using quotesCleanup.

Usage

```
sample_qdata_microseconds
```

Format

data.table object
**sample_real5minprices**  
Sample of imaginary price data for 61 days

**Description**  
An xts object containing the 5-min aggregated imaginary price series for the trading days between 2005-03-04 and 2005-06-01.

**Usage**  
sample_real5minprices

**Format**  
xts object

---

**sample_returns_5min**  
Sample returns data

**Description**  
EUR/USD returns from January to September 2004

**Usage**  
sample_returns_5min

**Format**  
A large xts object.

---

**sample_tdata**  
Sample of cleaned trades for stock XXX for 1 day

**Description**  
An xts object containing the trades for the imaginary stock XXX for 1 day, in the typical NYSE TAQ database format. This is the cleaned version of the data sample sample_tdataraw, using tradesCleanup.

**Usage**  
sample_tdata

**Format**  
A large xts object.
Sample of raw trades for stock XXX for 1 day

Description

An imaginary xts object containing the raw trades for stock XXX for 1 day, in the typical NYSE TAQ database format.

Usage

sample_tdataraw

Format

A large xts object.

Sample of raw trades for stock XXX for 2 days

Description

An imaginary data.table object containing the raw trades for stock XXX for 2 days, in the typical NYSE TAQ database format.

Usage

sample_tdataraw_microseconds

Format

A data.table object.
sample_tdata_microseconds

*Sample of cleaned trades for stock XXX for 2 days*

### Description

An data.table object containing the trades for the imaginary stock XXX for 2 days, in the typical NYSE TAQ database format. This is the cleaned version of the data sample `sample_tdataraw_microseconds`, using `tradesCleanup`.

### Usage

```
sample_tdata_microseconds
```

### Format

A data.table object.

---

**sbux**

*Starbucks Data*

### Description

Tick data for Starbucks 2011/07/01, cleaned with `tradesCleanup`.

### Usage

```
sbux
```

### Format

xts object

### Examples

```
data(sbux)
plot(sbux)
```
selectExchange

Retain only data from a single stock exchange

Description

Function returns an xts object containing the data of only 1 stock exchange.

Usage

```r
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 and Kris Boudt
SP500RM

SP500 Realized Measures calculated with 5 minute sampling

Description

Realized measures from the SP500 index from April 1997 to August 2013.

Usage

SP500RM

Format

A large xts object.

Source


References


spotDrift

Spot Drift Estimation

Description

Function used to estimate the spot drift of intraday (tick) stock prices/returns

Usage

spotDrift(
  data,
  method = "driftMean",
  ...,  
  on = "minutes",
  k = 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.

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.

... Additional arguments for the individual methods. See details

on What time-frame should the estimator be applied? Accepted inputs are "milliseconds", "seconds" and "secs" for seconds, "minutes" and "mins" for minutes, and "hours" for hours. Standard is minutes

k How often should the estimation take place? If k is 5 the estimation will be done every fifth unit of on.

marketopen Opening time of the market, standard is "09:30:00"

marketclose Closing time of the market, standard is "16:00:00"

tz Time zone, standard is "GMT"

Details

The additional arguments for the mean and median methods are: periods for the rolling window length which is 5 by standard and align to allow for control of the alignment, should one wish to do so, the standard is "right"

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


Examples

# Example 1: Rolling mean and median estimators for 2 days
meandrift <- spotDrift(data = sample_tdata.microseconds, k = 1, tz = "EST")
mediandrift <- spotDrift(data = sample_tdata.microseconds, method = "driftMedian",
on = "seconds", k = 30, tz = "EST")
plot(meandrift)
plot(mediandrift)

# Example 2: Kernel based estimator for one day
price <- sample_tdata$PRICE
storage.mode(price) <- "numeric"
spotvol <- spotDrift(price, method = "driftKernel", on = "minutes", k = 1)
#plot(kerneldrift)

---

spotvol

Spot volatility estimation

Description

Spot volatility estimation

Usage

```r
spotvol(
  data,
  method = "detper",
  ..., 
  on = "minutes",
  k = 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 on and k.

method specifies which method will be used to estimate the spot volatility. Options include "detper" and "stochper". See 'Details'.

... method-specific parameters (see 'Details').

on string indicating the time scale in which k is expressed. Possible values are: "secs", "seconds", "mins", "minutes", "hours".

k positive integer, indicating the number of periods to aggregate over. E.g. to aggregate an xts object to the 5 minute frequency, set k = 5 and on = "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 string specifying the time zone to which the times in data and/or marketopen/marketclose belong. Default = "GMT".
Details

The following estimation methods can be specified in method:

**Deterministic periodicity method** ("detper")

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dailyvol</td>
<td>A string specifying the estimation method for the daily component (s_t). Possible values are &quot;bipower&quot;, &quot;rv&quot;, &quot;medrv&quot;. Default = &quot;bipower&quot;.</td>
</tr>
<tr>
<td>periodicvol</td>
<td>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 &quot;TML&quot;, &quot;SD&quot;, &quot;WSD&quot;, &quot;OLS&quot;. See Boudt et al. (2011) for details. Default = &quot;TML&quot;.</td>
</tr>
<tr>
<td>P1</td>
<td>A positive integer corresponding to the number of cosinus terms used in the flexible Fourier specification of the periodic function. Default = 5.</td>
</tr>
<tr>
<td>P2</td>
<td>Same as P1, but for the sinus terms. Default = 5.</td>
</tr>
<tr>
<td>dummies</td>
<td>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 = FALSE.</td>
</tr>
</tbody>
</table>

Outputs (see 'Value' for a full description of each component):

- spot
- daily
- periodic

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. For more details, see Taylor and Xu (1997) and Andersen and Bollerslev (1997). The jump robust versions by Boudt et al. (2011) have also been implemented.

**Stochastic periodicity method** ("stochper") Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>A positive integer corresponding to the number of cosinus terms used in the flexible Fourier specification of the periodic function. Default = 5.</td>
</tr>
<tr>
<td>P2</td>
<td>Same as P1, but for the sinus terms. Default = 5.</td>
</tr>
<tr>
<td>init</td>
<td>A named list of initial values to be used in the optimization routine (&quot;BFGS&quot; 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))). See Beltratti &amp; Morana (2001) for a definition of each parameter.</td>
</tr>
<tr>
<td>control</td>
<td>A list of options to be passed down to optim.</td>
</tr>
</tbody>
</table>

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.

**Nonparametric filtering** ("kernel")

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String specifying the type of kernel to be used. Options include &quot;gaussian&quot;, &quot;epanechnikov&quot;, &quot;beta&quot;. Default = &quot;gaussian&quot;.</td>
</tr>
<tr>
<td>h</td>
<td>Scalar or vector specifying bandwidth(s) to be used in kernel. If h is a scalar, it will be assumed equal throughout the sample.</td>
</tr>
<tr>
<td>est</td>
<td>String specifying the bandwidth estimation method. Possible values include &quot;cv&quot;, &quot;quarticity&quot;. Method &quot;cv&quot; equals cross-validation, which chooses the bandwidth that minimizes the Integrated Square Error.</td>
</tr>
<tr>
<td>lower</td>
<td>Lower bound to be used in bandwidth optimization routine, when using cross-validation method. Default is (0.1n^{-0.2}).</td>
</tr>
<tr>
<td>upper</td>
<td>Upper bound to be used in bandwidth optimization routine, when using cross-validation method. Default is (n^{-0.2}).</td>
</tr>
</tbody>
</table>
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 (\( h \rightarrow \infty \)). 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.

**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). Default = 0.005.
- volest: String specifying the realized volatility estimator to be used in local windows. Possible values are "bipower", "rv", "medrv". Default = "bipower".
- 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) 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.

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.
spotvol

**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 cosinus 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**

This method generates the external regressors needed to model the intraday seasonality with a Flexible Fourier form. The rugarch package is then employed to estimate the specified GARCH(1,1) model.

Along with the spot volatility estimates, this method will return the `ugarchfit` object used by the rugarch package.

**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 `k` and `on`. 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 $r_{i,t}$ (read: the $i$th return on day $t$).

References


Examples

# Default method, deterministic periodicity
vol1 <- spotvol(sample_real5minprices)
plot(vol1)

# Compare to stochastic periodicity
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 110 iterations
vol2 <- spotvol(sample_real5minprices, method = "stochper", init = init)
plot(as.numeric(vol1$spot[1:780]), type="l")
lines(as.numeric(vol2$spot[1:780]), col="red")
legend("topright", c("detper", "stochper"), col = c("black", "red"), lty=1)

# Various kernel estimates
h1 <- bw.nrd0((1:nrow(sample_real5minprices))*(5*60))
vol3 <- spotvol(sample_real5minprices, method = "kernel", h = h1)
vol4 <- spotvol(sample_real5minprices, method = "kernel", est = "quarticity")
vol5 <- spotvol(sample_real5minprices, method = "kernel", est = "cv")
plot(vol3, length = 2880)
```r
lines(as.numeric(t(vol4$spot))[1:2880], col = "red")
lines(as.numeric(t(vol5$spot))[1:2880], col = "blue")
legend("topright", c("h = simple estimate", "h = quarticity corrected", "h = crossvalidated"), col = c("black", "red", "blue"), lty=1)

# Piecewise constant volatility, using an example from Fried (2012)

simdata <- matrix(sqrt(5/3)*rt(3000, df = 5), ncol = 500, byrow = TRUE)
simdata <- c(1, 1, 1.5, 1.5, 2, 1)*simdata
# the volatility of the simulated now changes at 1000, 2000 and 2500
vol6 <- spotvol(simdata, method = "piecewise", m = 200, n = 100,
online = FALSE)
plot(vol6)

# Compare regular GARCH(1,1) model to eGARCH, both with external regressors

vol7 <- spotvol(sample_real5minprices, method = "garch", model = "sGARCH")
vol8 <- spotvol(sample_real5minprices, method = "garch", model = "eGARCH")
plot(as.numeric(t(vol7$spot)), type = "l")
lines(as.numeric(t(vol8$spot)), col = "red")
legend("topleft", c("GARCH", "eGARCH"), col = c("black", "red"), lty=1)
```

---

**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, selectExchange, salesCondition, 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**

```r
tradesCleanup(
    datasource = NULL,
    datadestination = NULL,
    exchanges,
    tdataraw = NULL,
    report = TRUE,
    selection = "median",
)```
saveasxts = TRUE
)

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
tdataraw xts object containing (for ONE stock only) 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".
saveasxts indicates whether data should be saved in xts format instead of data.table when using on-disk functionality. TRUE by default.

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 and Kris Boudt
tradesCleanupUsingQuotes

References


Examples

# Consider you have raw trade data for 1 stock for 2 days
head(sample_tdataraw_microseconds)
dim(sample_tdataraw_microseconds)
tdata_afterfirstcleaning <- tradesCleanup(tdataraw = sample_tdataraw, exchanges = list("N"))
tdata_afterfirstcleaning$report
dim(tdata_afterfirstcleaning$tdata)

# In case you have more data it is advised to use the on-disk functionality
# via "from","to","datasource",etc. 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(
  datasource = NULL,
  datadestination = NULL,
  tdata = NULL,
  qdata = 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, folder of datasource by default.

tdata data.table or xts object containing (ONE day and for ONE stock only) 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)
tradesCleanupUsingQuotes

qdata data.table or xts object containing (ONE day and for ONE stock only) 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)

Value

For each day an xts object is saved into the folder of that date, containing the cleaned data.

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).

Author(s)

Jonathan Cornelissen, Kris Boudt and Onno Kleen.

References


Examples

# Consider you have raw trade data for 1 stock for 2 days
 tdata_afterfirstcleaning <- tradesCleanup(tdataraw = sample_tdataraw_microseconds, exchanges = "N", report = FALSE)

#
 qdata <- quotesCleanup(qdataraw = sample_qdataraw_microseconds, exchanges = "N", report = FALSE)

 dim(tdata_afterfirstcleaning)
 tdata_afterfinalcleaning <- tradesCleanupUsingQuotes(qdata = qdata[as.Date(DT) == "2018-01-02"],
 tdata = tdata_afterfirstcleaning[as.Date(DT) == "2018-01-02"])

 dim(tdata_afterfinalcleaning)
 #In case you have more data it is advised to use the on-disk functionality
 #via "from","to","datasource", etc. arguments
Index

*Topic **AJjumptest**
AJjumptest, 8

*Topic **BNSjumptest**
BNSjumptest, 13

*Topic **Drift**
spotDrift, 84

*Topic **JOjumptest**
JOjumptest, 30

*Topic **RV**
RV, 76

*Topic **cleaning**
autoSelectExchangeQuotes, 11
autoSelectExchangeTrades, 12
exchangeHoursOnly, 15
mergeQuotesSameTimestamp, 37
mergeTradesSameTimestamp, 38
noZeroPrices, 43
noZeroQuotes, 43
quotesCleanup, 44
rLargeSpread, 56
rmNegativeSpread, 57
rmOutliersQuotes, 58
rmTradeOutliersUsingQuotes, 61
selectExchange, 83
tradesCleanup, 91
tradesCleanupUsingQuotes, 93

*Topic **datasets**
lltc, 32
realized_library, 51
sample_5minprices, 77
sample_5minprices_jumps, 77
sample_qdata, 78
sample_qdata_microseconds, 79
sample_qdata_raw, 78
sample_qdata_raw_microseconds, 79
sample_real5minprices, 80
sample_returns_5min, 80
sample_tdata, 80
sample_tdata_microseconds, 82
sample_tdata_raw, 81
sample_tdata_raw_microseconds, 81
sbux, 82
SP500RM, 84

*Topic **data**
aggregateQuotes, 4
aggregateTrades, 5
dataFrames, 13
makePsd, 33
matchTradesQuotes, 34
refreshTime, 52

*Topic **forecasting**
harModel, 21

*Topic **highfrequency**
AJjumptest, 8
BNSjumptest, 13
ivInference, 28
JOjumptest, 30
medRQ, 35
MRC, 41
rBeta, 47
rKurt, 55
rMPV, 59
rQPVar, 63
rQuar, 64
rSkew, 68
rSV, 69
rTPVar, 72
RV, 76

*Topic **ivInference**
ivInference, 28

*Topic **leaning**
salesCondition, 76

*Topic **liquidity**
getTradeDirection, 20

*Topic **manipulation**
aggregateQuotes, 4
aggregateTrades, 5
datasets, 52
makePsd, 33
matchTradesQuotes, 34
refreshTime, 52
*Topic medRQ
medRQ, 35
*Topic preaveraging
MRC, 41
*Topic rBeta
rBeta, 47
*Topic rKurt
rKurt, 55
*Topic rMPV
rMPV, 59
*Topic rQPVar
rQPVar, 63
*Topic rQuar
rQuar, 64
*Topic rSV
rSV, 69
*Topic rSkew
rSkew, 68
*Topic rTPVar
rTPVar, 72
*Topic volatility
listAvailableKernels, 32
medRQ, 36
minRQ, 40
rAVGCov, 46
rBPCov, 49
rCov, 50
rHYCov, 53
rKernelCov, 54
rOWCov, 61
rRTSCov, 65
rThresholdCov, 70
rTSCov, 73
aggregateQuotes, 4, 7
aggregateTrades, 5, 7
aggregatets, 7
AJjumptest, 8
autoSelectExchangeQuotes, 11
autoSelectExchangeTrades, 12
BNSjumptest, 13
exchangeHoursOnly, 15
getLiquidityMeasures, 16
getePrice, 20
getTradeDirection, 20
harModel, 21
hasQty, 25
heavyModel, 25
highfrequency (highfrequency-package), 3
highfrequency-package, 3
ivInference, 28
JOjumptest, 30
listAvailableKernels, 32, 55
lltc, 32
lm, 23
makePsd, 33, 67, 75
makeReturns, 34
matchTradesQuotes, 20, 34
medRQ, 35
medRV, 36
mergeQuotesSameTimestamp, 37, 44, 45
mergeTradesSameTimestamp, 38, 91, 92
minRQ, 39
minRV, 40
MRC, 41
noZeroPrices, 43, 91
noZeroQuotes, 43, 44
quotesCleanup, 44, 93
rAVGCov, 46
rBeta, 47
rBPCov, 49, 62
rCov, 50
realized_library, 51
refreshTime, 52, 67, 74
rHYCov, 53
rKernelCov, 54
rKurt, 55
rlLargeSpread, 56
rlnNegativeSpread, 57
rmOutliersQuotes, 44, 45, 58
rMPV, 59
rmTradeOutliers, 60
rmTradeOutliersUsingQuotes, 61, 91, 93
rOWCov, 61, 65, 70
rQPVar, 63
INDEX

rQuar, 64
rRTSCov, 65
rSkew, 68
rSV, 69
rThresholdCov, 65, 70
rTPVar, 72
RTQ, 73
rTSCov, 73
RV, 76

salesCondition, 76, 91
sample_5minprices, 77
sample_5minprices_jumps, 77
sample_qdata, 4, 78
sample_qdata_microseconds, 79
sample_qdataraw, 78, 78
sample_qdataraw_microseconds, 79, 79
sample_real5minprices, 80
sample_returns_5min, 80
sample_tdata, 80
sample_tdata_microseconds, 82
sample_tdataraw, 80, 81
sample_tdataraw_microseconds, 81, 82
sbux, 82
selectExchange, 44, 83, 91
SP500RM, 84
spotDrift, 84
spotvol, 86

tradesCleanup, 91, 93
tradesCleanupUsingQuotes, 91, 93