Highfrequency: Toolkit for the analysis of highfrequency financial data in R.

Kris Boudt
VU University Amsterdam,
Lessius and K.U.Leuven

Jonathan Cornelissen
K.U.Leuven

Scott Payseur
UBS Global Asset Management

Maarten Schermer
GSoC Student

Abstract

The highfrequency package contains an extensive toolkit for the use of highfrequency financial data in R. It offers functionality to manage, clean and match highfrequency trades and quotes data. Furthermore, it enables users to: calculate easily various liquidity measures, estimate and forecast volatility, and investigate microstructure noise and intraday periodicity. See also: `http://highfrequency.herokuapp.com` In a couple of months I'll post an interactive tutorial on: `http://www.datamind.org`

Keywords: High-frequency data, Liquidity, Quotes, R software, Realized volatility, Trades.

1. General information

The economic value of analyzing *high-frequency financial* data is now obvious, both in the academic and financial world. It is the basis of intraday and daily risk monitoring and forecasting, an input to the portfolio allocation process, and also for high-frequency trading. For the efficient use of high-frequency data in financial decision making, the `highfrequency` package implements state of the art techniques for cleaning and matching for trades and quotes, as well as the calculation and forecasting of liquidity, realized and correlation measures based on high-frequency data. The `highfrequency` package is the outcome of a Google summer of code project and an improved and updated version of the 2 packages: `RTAQ` (Cornelissen and Boudt 2012) and `realized` (Payseur 2008).

Handling high-frequency data can be particularly challenging because of the specific characteristics of the data, as extensively documented in Yan and Zivot (2003). The `highfrequency` package offers high-level tools for the analysis of high-frequency data. These tools tackle three typical challenges of working with high-frequency data. A first specific challenge is the enormous number of observations, that can reach heights of millions observations per stock per day. Secondly, transaction-by-transaction data is by nature irregularly spaced over time. Thirdly, the recorded data often contains errors for various reasons.

The `highfrequency` package offers a (basic) interface to manage highfrequency trades and

quotes data. Furthermore and foremost, it offers easy-to-use tools to clean and aggregate high-frequency data, to calculate liquidity measures and, measure and forecast volatility. The tools in `highfrequency` are designed in such a way that they will work with most data input types. The package contains an extensive collection of functions to calculate realized measures, since it arose by merging the package `RTAQ` (Cornelissen and Boudt 2012) from the TradeAnalytics project and the package `realized` (Payseur 2008).

`highfrequency` strongly builds on the functionality offered by the `xts` package (Ryan and Ulrich 2009). You can use the functionality in the `highfrequency` package for trades and quotes objects from the most popular vendors, as long as your objects are stored as xts objects.

The `highfrequency` package is hosted on R-forge and the latest version of the package can be downloaded through the following command:

```
install.packages("highfrequency", repos="http://R-Forge.R-project.org")
```

2. Organizing highfrequency data

Users that have their own system set up to store their highfrequency data as xts objects can skip this section. The functions to calculate liquidity, etc. in the `highfrequency` package can take highfrequency data of various data vendors as input: NYSE TAQ, WRDS TAQ, Reuters, Bloomberg, etc. To achieve this flexibility, we rely on the functionality provided by the excellent `quantmod` package (Ryan 2011). Additionally, `highfrequency` provides functionality to convert raw data from several data providers into `xts` objects. The latter is discussed and illustrated in this section.

Highfrequency financial data is typically subdivided into two files, containing information on the trades and quotes respectively. `highfrequency` can currently parse three types of input data into xts objects: (i) ".txt" files extracted from the NYSE TAQ database, (ii) ".csv" "files extracted from the WRDS database, (iii) ".asc" files from `http://www.tickdata.com`. For each of these input data types, we briefly discuss how the `convert` function can be used to convert the txt/csv/asc files into xts objects (Ryan and Ulrich 2009) that are saved on-disk in the "RData" format. We opt to store the data as xts objects because this type of object can be indexed by an indicator of time and date.[1]

Parsing raw NYSE TAQ data into xts:
Suppose the folder `~/raw_data` on your drive contains the two folders: "2008-01-02" and "2008-01-03". Suppose these folders each contain the files `AAPL_trades.txt` and `AA_trades.txt` with the daily TAQ data bought from the New York Stock Exchange (NYSE) for the `AAPL` and `AA` stock. The raw data can then easily be converted into on-disk xts objects as follows:

```
from = "2008-01-02";
to = "2008-01-03";
datasource = "~/raw_data";
datadestination = "~/xts_data";
```

[1] Examples of the use of xts objects can be found on `http://www.quantmod.com/examples/data/`.

```
convert(from, to, datasource, datadestination, trades=TRUE,
          quotes=FALSE,ticker=c("AA","AAPL"), dir=TRUE, extension="txt",
          header=FALSE,tradecolnames=NULL,quotecolnames=NULL,
          format="%Y%m%d %H:%M:%S");
```

At this point, the folder `~/xts_data` will contain two folders named 2008-01-02 and 2008-01-03 containing the files `AAPL_trades.RData` and `AAPL_trades.RData` which contain the respective daily trade data. The data can now be loaded with the `TAQLoad` function (see below).

Parsing raw WRDS TAQ data into xts:
Suppose the folder `~/raw_data` on your drive contains the files `IBM_quotes.csv` and `IBM_trades.csv` acquired from Wharton Research Data Service (WRDS).[2] Both files contain respectively the trade and quote data for the "IBM" stock for "2011-12-01" up to "2011-12-02". The data can then easily be converted into on-disk xts objects as follows:

```
from = "2011-12-01";
to = "2011-12-02";
datasource = "~/raw_data";
datadestination = "~/xts_data";
convert( from=from, to=to, datasource=datasource,
              datadestination=datadestination, trades = T,  quotes = T,
              ticker="IBM", dir = TRUE, extension = "csv",
              header = TRUE, tradecolnames = NULL, quotecolnames = NULL,
              format="%Y%m%d %H:%M:%S", onefile = TRUE )
```

Now, the folder `~/xts_data` contains two folders, one for each day in the sample, i.e. "2011-12-01" up to "2011-12-02". Each of these folders contains a file named `IBM_trades.RData` in which the trades for that day can be found and a file `IBM_quotes.RData` in which the quotes for that day can be found, both saved as xts objects. As a final step, the function `TAQLoad` can be used to load the on-disk data into your R workspace. You can for example get the trades for this sample in your workspace as follows:

```
>   xts_data = TAQLoad( tickers="IBM", from="2011-12-01",
+              to="2011-12-02",trades=F,
+              quotes=TRUE, datasource=datadestination)
>   head(xts_data)
                     SYMBOL EX  BID      BIDSIZ OFR      OFRSIZ MODE
2011-12-01 04:00:00 "IBM"  "P" "176.85" "1"    "188.00" " 1"   "12"
2011-12-01 04:00:17 "IBM"  "P" "185.92" "1"    "187.74" " 1"   "12"
2011-12-01 04:00:18 "IBM"  "P" "176.85" "1"    "187.74" " 1"   "12"
2011-12-01 04:00:25 "IBM"  "P" "176.85" "1"    "187.73" " 1"   "12"
2011-12-01 04:00:26 "IBM"  "P" "176.85" "1"    "188.00" " 1"   "12"
2011-12-01 04:00:26 "IBM"  "P" "176.85" "1"    "187.74" " 1"   "12"
```

[2] http://wrds-web.wharton.upenn.edu/wrds

Parsing raw Tickdata.com data into xts:
Suppose the folder `~/raw_data` on your drive contains the files `GLP_quotes.asc` and `GLP_trades.asc` acquired from www.tickdata.com. Both files contain respectively the trade and quote data for the "GLP" stock for "2011-01-11" up to "2011-03-11". The data can then easily be converted into on-disk xts objects as follows:

```
from = "2011-01-11";
to     = "2011-03-11";
datasource = "~/raw_data";
datadestination = "~/xts_data";
convert(from=from, to=to, datasource=datasource,
             datadestination=datadestination, trades = TRUE,
             quotes = TRUE, ticker="GLP", dir = TRUE, format = "%d/%m/%Y %H:%M:%OS",
             extension = "tickdatacom", header = TRUE,  onefile = TRUE );
```

At this point, the folder `~/xts_data` now contains three folders, one for each day in the sample, i.e. "2011-01-11", "2011-02-11", "2011-03-11". Each of these folders contains a file named `GLP_trades.RData` in which the trades for that day can be found and a file `GLP_quotes.RData` in which the quotes for that day can be found, both saved as xts object. Notice the `format = '%d/%m/%Y %H:%M:%OS'` argument that allows for timestamps more accurate than seconds, e.g. milliseconds in this case. As a final step, the function `TAQLoad` can be used to load the on-disk data into your R workspace. You can for example get the trade data for the first day of the sample as follows:

```
>  options("digits.secs"=3); #Show milliseconds
>  xts_data = TAQLoad(tickers="GLP", from="2011-01-11", to="2011-01-11",
+              trades=T, quotes=F,
+              datasource=datadestination)
>  head(xts_data)
                        SYMBOL EX  PRICE     SIZE   COND  CORR G127
2011-01-11 09:30:00.338 "GLP"  "T" "18.0700" " 500" "O X" "0"  ""
2011-01-11 09:30:00.338 "GLP"  "T" "18.0700" " 500" "Q"   "0"  ""
2011-01-11 09:33:49.342 "GLP"  "T" "18.5000" " 150" "F"   "0"  ""
2011-01-11 09:39:29.280 "GLP"  "N" "19.2000" "4924" "O"   "0"  ""
2011-01-11 09:39:29.348 "GLP"  "D" "19.2400" " 500" "@"   "0"  "T"
2011-01-11 09:39:29.411 "GLP"  "N" "19.2400" " 200" "F"   "0"  ""
```

Typical trades and quotes: data description:
Both trades and quotes data should thus be xts objects (Ryan and Ulrich 2009), if you want to use the functionality of `highfrequency`. Table 1 reports the information (i.e. columns) typically found in trades and quotes data objects.

3. Manipulation of highfrequency data

In this section we discuss two very common first steps in the manipulation of highfrequency financial data: (i) the cleaning and (ii) aggregation of the data.

Table 1: Elements of trade and quote data

All data	
SYMBOL	The stock's ticker
EX	Exchange on which the trade/quote occurred
Trade data	
PRICE	Transaction price
SIZE	Number of shares traded
COND	Sales condition code
CORR	Correction indicator
G127	Combined "G", Rule 127, and stopped stock trade
Quote data	
BID	Bid price
BIDSIZ	Bid size in number of round lots (100 share units)
OFR	Offer price
OFRSIZ	Offer size in number of round lots (100 share units)
MODE	Quote condition indicator

3.1. Cleaning of highfrequency data

For various reasons, raw trade and quote data contains numerous data errors. Therefore, the data is not suited for analysis right-away, and data-cleaning is an essential step in dealing with tick-by-tick data (Brownlees and Gallo 2006). `highfrequency` implements the step-by-step cleaning procedure proposed by Barndorff-Nielsen *et al.* (2008). Table 2 provides an overview of the cleaning functions. A user can either use a specific cleaning procedure or a wrapper function that performs multiple cleaning steps. The wrapper functions offer on-disk functionality: i.e. load on-disk raw data and save the clean data back to your hard disk. This method is advisable in case you have multiple days of data to clean. Of course, the data on your disk should be organized as discussed in the previous section to benefit from this functionality. More specifically, these functions expect that the data is saved as xts objects saved in a folder for each day (as the `TAQLoad` function does), which will always be the case if you used the `convert` function to convert the raw data into xts objects.

To maintain some insight in the cleaning process, the functions `tradesCleanup` and `quotesCleanup` report the total number of remaining observations after each cleaning step:

```
> data("sample_tdataraw");
> dim(sample_tdataraw);
[1] 48484     7
> tdata_afterfirstcleaning = tradesCleanup(tdataraw=sample_tdataraw,exchanges="N");
> tdata_afterfirstcleaning$report;
      initial number       no zero prices      select exchange
                48484                48479                20795
     sales condition merge same timestamp
                20135                 9105
> dim(tdata_afterfirstcleaning$tdata)
```

Table 2: Cleaning functions

Function	Function Description
	All Data:
ExchangeHoursOnly	Restrict data to exchange hours
selectexchange	Restrict data to specific exchange
	Trade Data:
noZeroPrices	Delete entries with zero prices
autoSelectExchangeTrades	Restrict data to exchange with highest trade volume
salesCondition	Delete entries with abnormal Sale Condition
mergeTradesSameTimestamp	Delete entries with same time stamp and use median price
rmTradeOutliers	Delete entries with prices above/below ask/bid +/- bid/ask spread
	Quote Data:
noZeroQuotes	Delete entries with zero quotes
autoSelectExchangeQuotes	Restrict data to exchange with highest bidsize + offersize
mergeQuotesSameTimestamp	Delete entries with same time stamp and use median quotes
rmNegativeSpread	Delete entries with negative spreads
rmLargeSpread	Delete entries if spread > maxi*median daily spread
rmOutliers	Delete entries for which the mid-quote is outlying with respect to surrounding entries
Wrapper cleanup functions (perform sequentially the following for on-disk data)	
tradesCleanup	noZeroPrices, selectExchange, salesCondition, mergeTradesSameTimestamp.
quotesCleanup	noZeroQuotes, selectExchange, rmLargeSpread, mergeQuotesSameTimestamp rmOutliers
tradesCleanupFinal	rmTradeOutliers (based on cleaned quote data as well)

```
[1] 9105    7
```

3.2. Aggregation of highfrequency data

Prices are typically not recorded at equispaced time points, while e.g. many realized volatility measures rely on equispaced returns. Furthermore, prices are often observed at different points in time for different assets, while e.g. most multivariate realized volatility estimators rely on synchronized data (see e.g. Table 3). There several ways to force these asynchronously and/or irregularly recorded series to a synchronized and/or equispaced time grid.

The most popular method, previous tick aggregation, forces prices to an equispaced grid by taking the last price realized before each grid point. `highfrequency` provides users with the `aggregatets` function for fast and easy previous tick aggregation:

```
> library("highfrequency");
> # Load sample price data
> data("sample_tdata");
> ts = sample_tdata$PRICE;
> # Previous tick aggregation to the 5-minute sampling frequency:
> tsagg5min = aggregatets(ts,on="minutes",k=5);
> head(tsagg5min);

                     PRICE
2008-01-04 09:35:00 193.920
```

```
2008-01-04 09:40:00 194.630
2008-01-04 09:45:00 193.520
2008-01-04 09:50:00 192.850
2008-01-04 09:55:00 190.795
2008-01-04 10:00:00 190.420
```

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

```
                     PRICE
2008-01-04 15:57:30 191.790
2008-01-04 15:58:00 191.740
2008-01-04 15:58:30 191.760
2008-01-04 15:59:00 191.470
2008-01-04 15:59:30 191.825
2008-01-04 16:00:00 191.670
```

In the example above the prices are forced to a regular equispaced time grid of respectively 5 minutes and 30 seconds. Furthermore, the `aggregatets` function is build into all realized measures (see Section 4) and can be called by setting the arguments `align.by` and `align.period`. In that case, first the price(s) are forced the an equispaced regular time grid and then the realized measure is calculated based on the returns over these regular time periods to which the observations were forced. This has the advantage that the user can input the original price series into the realized measures without having to worry about the asynchronicity or the irregularity of the price series.

Another synchronization method (not integrated into the realized measures) is refresh time, initially proposed by Harris *et al.* (1995) and recently advocated in Barndorff-Nielsen *et al.* (2011). The function `refreshTime` in `highfrequency` can be used to force time series (very fast) to a synchronized but not necessarily equispaced time grid. The so-called refresh times are the time points at which all assets have traded at least once since the last refresh point. More specifically, 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 been traded again. This process is repeated until the end of one time series is reached.

Illustration on price aggregation with refresh time and subsequent volatility calculation:

```
> data("sample_tdata");
> data("sample_qdata");
> #We assume that stock1 and stock2 contain price data on imaginary stocks:
> stock1 = sample_tdata$PRICE;
> stock2 = sample_qdata$BID;
> #Previous-tick aggregation to one minute:
> mPrice_1min = cbind(aggregatePrice(stock1),aggregatePrice(stock2));
> #Refresh time aggregation:
> mPrice_Refresh = refreshTime(list(stock1,stock2));
> #Calculate a jump robust volatility measures
```

```
> #based on synchronized data:
> rbpcov1 = rBPCov(mPrice_1min,makeReturns=TRUE);
> rbpcov2 = rBPCov(mPrice_Refresh,makeReturns=TRUE);
> #Calculate a jump and microstructure noise robust volatility measure
> #based on nonsynchronous data:
> rtscov = rTSCov(list(stock1,stock2));
```

4. Realized volatility measures

The availability of high-frequency data has enabled researchers to estimate the ex post realized volatility based on squared intraday returns (Andersen *et al.* 2003). In practice, the main challenges in univariate volatility estimation are dealing with (i) jumps in the price level and (ii) microstructure noise. Multivariate volatility estimation is additionally callenging because of (i) the asynchronicity of observations between assets and (ii) the need for a positive semidefnite covariance matrix estimator. The `highfrequency` package implements many recently proposed realized volatility and covolatility measures[3]

An overview of the univariate and multivariate volatility estimators implemented in `highfrequency` is given in Table 3. The first two columns indicate whether the estimator can be applied to univariate or multivariate price series. The following two columns indicate whether the estimator is robust with respect to jumps and microstructure noise respecitively. The next column reports whether asynchronic price series can/should be used as input. The last column indicates whether the estimator always yields a positive semidefinite matrix in the multivariate case. All realized measures have (at least) the following arguments:

- `rdata`: The return data.
  - In the univariate case: an xts object containing the (tick) data for one day.
  - In the multivariate case:
    - In case of synchronized observations: a (M x N) matrix/zoo/xts object containing the N return series over period t, with M observations during t.
    - In the case of asynchronous observations: a list. Each list-item i contains an xts object with the intraday data of stock i for day t.
- `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.
- Arguments relevant for multivariate case (i.e. realized covolatility measures):
  - `cor`: boolean, in case it is TRUE, the correlation is returned. FALSE by default.

[3] Throughout we use the terms realized volatility and realized variance interchangeably to denote high-frequency data based estimators for the daily variation in the returns. All outputted values should be interpreted as variances, unless they clearly indicate standard deviations, e.g. in the spotvol function.

Table 3: Overview of Volatility estimators

Estimator	Univariate	Multivariate	Jump robust	Microstructure noise robust	Tick-by-tick returns as input	Positive semidefinite
medRV (Andersen *et al.* 2012)	x		x			/
minRV (Andersen *et al.* 2012)	x		x			/
rCov (Andersen *et al.* 2003)	x	x				x
rBPCov (Barndorff-Nielsen and Shephard 2004)	x	x	x			
rOWCov (Boudt *et al.* 2011a)	x	x	x			x
rThresholdCov (Gobbi and Mancini 2009)		x	x			x
rTSCov (Zhang 2011)	x	x		x	x	
rRTSCov (Boudt and Zhang 2010)	x	x	x	x	x	
rAVGCov (Ait-Sahalia *et al.* 2005)	x	x		x	x	x
rKernelCov (Barndorff-Nielsen *et al.* 2004)	x	x		x	x	x
rHYCov (Hayashi and Yoshida 2005)		x			x	

  - `makePsd`: boolean. Non positive semidefinite estimates can be transformed into a positive semidefinite matrix by setting this argument to TRUE. The eigenvalue method is used.

- ...: additional arguments depending on the type of realized measure.

5. Spot volatility estimation

While realized measures can be used to estimate the volatility over a specified time period, it is also interesting to estimate the 'instantaneous' volatility at any given moment. This is called the spot volatility. The `spotvol` function offers several methods to estimate spot volatility and its intraday seasonality, using high-frequency data. The spot volatility $\sigma_{t,n}$ can be described as the the volatility of returns on a certain moment n during day t. These returns $r_{t,n}$ are commonly modeled as

$$r_{t,n} = \sigma_{t,n}\varepsilon_{t,n}, \tag{1}$$

where $\varepsilon_{t,n} \sim \text{IID}(0, \frac{1}{N})$ is a white noise process. N denotes the number of intraday periods, and we will denote the number of days with T.

5.1. General usage

The spot volatility estimates $\hat{\sigma}_{t,n}$ can be estimated from these returns using the `spotvol` function. The estimates are stored as the variable `spot` in a `spotvol` object, which is a list containing certain outputs of the function, depending on the method used. These following example shows how to call the `spotvol` function, using sample data of 5-minute prices, that the class of the output is indeed '`spotvol`', and how to assign the spot volatility estimates `spot` to a new variable:

```
library(highfrequency)
data(sample_real5minprices)
out <- spotvol(sample_real5minprices)
class(out)
```

```
## [1] "spotvol"

sigma_hat <- out$spot
```

Input data: prices or returns

The `spotvol` function accepts two sorts of input:

1. An `xts` object containing price data.
2. A `matrix` containing return data.

The first can be used for raw price data, i.e. from the NYSE TAQ database. The `highfrequency` package offers several methods to process data files into `xts` format. The `spotvol` function will then aggregate this price data to the frequency specified by the user and calculate the return series.

```
str(sample_real5minprices)

## An 'xts' object on 2005-03-04 09:30:00/2005-06-01 16:00:00 containing:
##   Data: num [1:4819, 1] 105 105 105 105 104 ...
##   Indexed by objects of class: [timeDate] TZ:
##   xts Attributes:
##  NULL

out <- spotvol(sample_real5minprices)
```

The second way of inputting data is through a `matrix`, which can be convenient if your data has already been processed. No further permutations will be made to the data, as it is assumed to be in the right format. Therefore, it should satisfy the following restrictions:

- The returns should be equispaced, i.e. the time between each pair of observations should be equal.
- Each row of the matrix should correspond to a day.
- Each column of the matrix should correspond to an intraday time interval.

For example, a matrix containing 5-minute returns of 60 days of 24h exchange rate data should have 60 rows and 288 columns:

```
data(sample_returns_5min)
str(sample_returns_5min)

##  num [1:60, 1:288] 0.00404 -0.02825 0.03223 0.00806 0.04056 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:288] "X2" "X3" "X4" "X5" ...

out <- spotvol(sample_returns_5min)
```

Market opening times and time zones

Working with `xts` objects means that every observation in your data should have a timestamp. This is convenient, but can also pose some problems (e.g. with timezones). The `spotvol` function does not require the data to be timestamped: you can also supply raw return data in matrix form.

Market opening and closing times can be specified by the arguments `marketopen` and `marketclose`. They should be entered in 24h format and in the timezone specified by `tz`, which should also be the timezone of your `xts` data. If your `xts` object does not have a timezone attribute specified, it will be assigned value `tz`. Note that all of your data must be in the same timezone. Problems can arise when using a daylight saving dependent timezone, such as CET, which changes to CEST during summer. It is advised to use a timezone which is invariant to daylight saving, such as GMT/UTC.

The time entered as `marketopen` should always be before `marketclose`. This can be a problem in case of 24h data, which lacks open and close times, or the opening time would be the same as the closing time. As the `spotvol` function processes the data on a daily basis, some problems arise when a 24h-period of data spans multiple days. There are two ways around this: either have your data always start at 0:00:00, or avoid time issues by entering your data as a `matrix`.

Method specific parameters

The `spotvol` function offers several methods to estimate the spot volatility. These are all listed in the help pages of the function and package. The user can specify the method to be used with the `method` argument, which accepts the following values:

- `"detper"` (default)
- `"stochper"`
- `"kernel"`
- `"piecewise"`

Each of these has different parameters to be specified by the user. These can all be entered as arguments to the `spotvol` function, although they will only be accepted if they are actually used by the specified method. The 'Details' section of the `spotvol` help page lists which parameters are used by each method. It is not required to specify all parameters; the function will even work if none are entered, by using default values.

Output

The `spotvol` function returns an object of the `spotvol` class. This is a list of which the contents depend on the used method. In any case, it includes the variable `spot`, which contains all spot volatility estimates as an `xts` or `matrix` object (depending on the input). The following example shows the different outputs for methods `"detper"` and `"stochper"`:

```
out1 <- spotvol(sample_real5minprices, method = "detper")
out2 <- spotvol(sample_real5minprices, method = "stochper")
```

```
names(out1)

## [1] "spot"     "daily"    "periodic"

names(out2)

## [1] "spot" "par"
```

Plotting

To get a quick overview of the output, the `plot` function can be called on `spotvol` objects. The S3 method `plot.spotvol` has been implemented to show the spot volatility estimates, as well as any other components of the `spotvol` object that can be visually represented, depending on the used method. An additional argument that can be passed to `plot.spotvol` is `length`, which can limit the amount of data points to be plotted. Examples are shown in section 5.2.

5.2. Methods

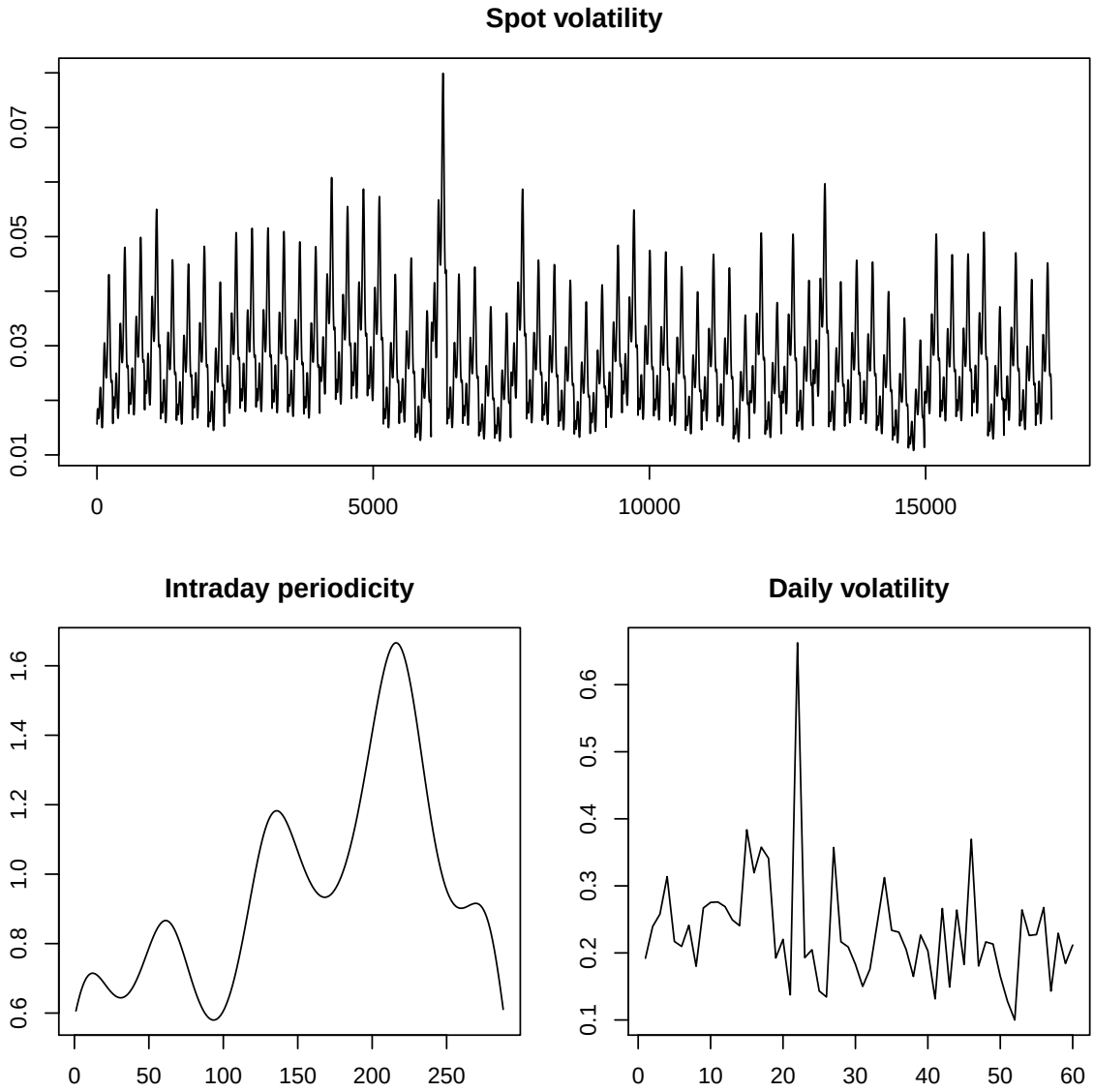
Each method implemented in the `spotvolatility` package has been proposed in an academic paper. For detailed descriptions, refer to these papers written by the original authors of the methods. References can be found at the end of this document or in the help page of this package, where you can also find a short overview of each method. In this section, we show some examples for each method.

Deterministic periodicity - `"detper"`

This is the default method. 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.* (2011b) have also been implemented.

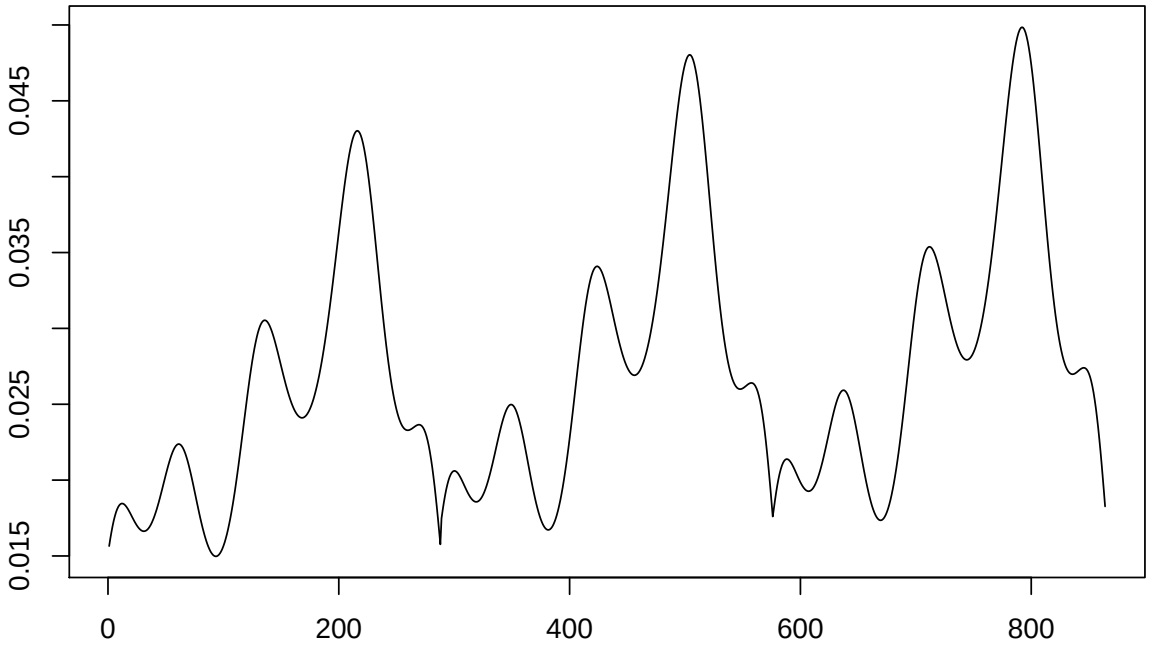
First, we can use the `plot` function to get a quick view of the results. This will call the `plot.spotvolatility` function, which automatically shows all relevant results contained in the `spotvol` object. When estimated by the deterministic method, it plots the general spot volatility estimates in the top panel, and the intraday periodicity pattern and daily volatilies in the bottom panels:

```
data(sample_returns_5min)
vol_detper <- spotvol(sample_returns_5min, method = "detper")
plot(vol_detper)
```



Now, to get a better look at the spot volatility estimates, we need to zoom in. This time, we only plot the first 3 days of the spot volatility. Recall that since we entered the input as a matrix, the output `vol_detper$spot` is also a matrix, with each row representing 1 day of estimates:

```
plot(as.numeric(t(vol_detper$spot[1:3,])), type = "l")
```



Now, it is easier to see how this method works. The estimated periodicity pattern (the second panel in the previous plot) is repeated every day, and it is multiplied by the estimate of the daily volatility (the third panel in the previous plot). To get the plot to zoom in, we also could have run

```
plot(vol_detper, length = 3*288)
```

Stochastic periodicity - `"stochper"`

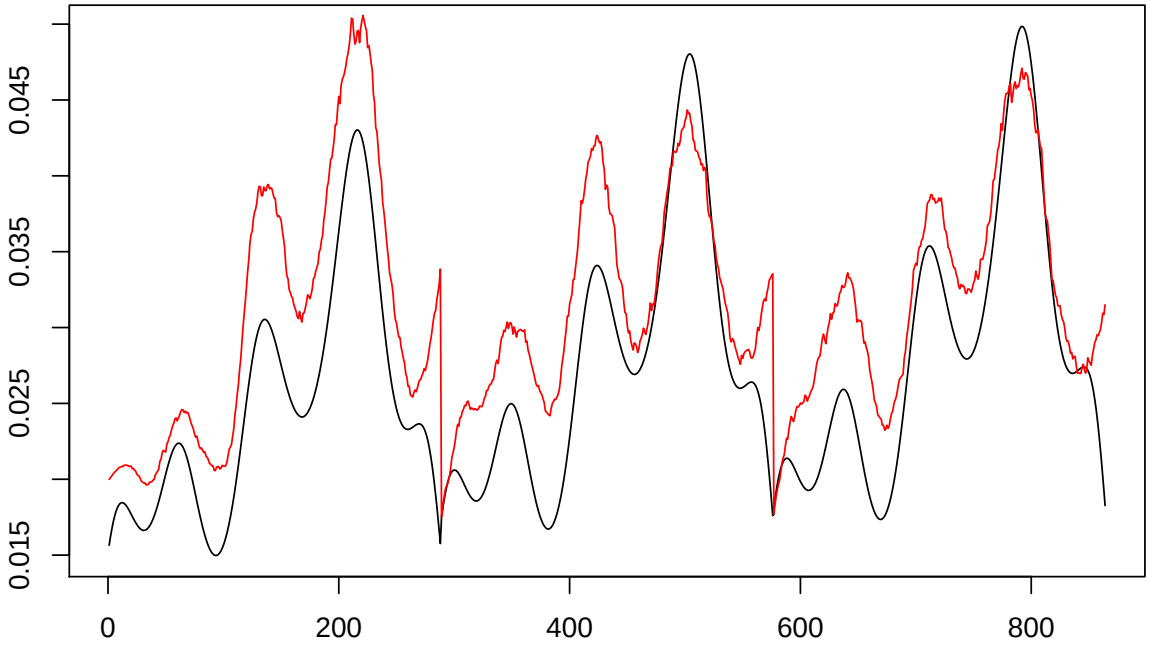
This method by Beltratti and Morana (2001) assumes the periodicity factor to be stochastic. The spot volatility estimation is split into four components: a random walk, an autoregressive process, a stochastic cyclical process and a deterministic cyclical process. The model is estimated using a quasi-maximum likelihood method based on the Kalman Filter. The package **FKF** (Luethi *et al.* 2014) is used to apply the Kalman filter. In addition to the spot volatility estimates, all parameter estimates are returned.

Like most stochastic volatility models, this model can be difficult to estimate. Optimization can take a long time, and results can depend on the initial values used in the estimation of the parameters. Therefore, these values can be specified by the user, as well as optimization restrictions:

```
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))
vol_stochper <- spotvol(sample_returns_5min, method = "stochper",
    init = init, control = list(maxit = 20))
```

The `control` argument will be passed down to `optim`, so it will accept all options listed in `?optim`. We can now compare the spot volatility estimates of the stochastic method to those calculated previously by the deterministic model:

```
plot(as.numeric(t(vol_detper$spot[1:3,])), type = "l")
lines(as.numeric(t(vol_stochper$spot[1:3,])), col = "red")
```



Nonparametric filtering - `"kernel"`

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 appropiate.

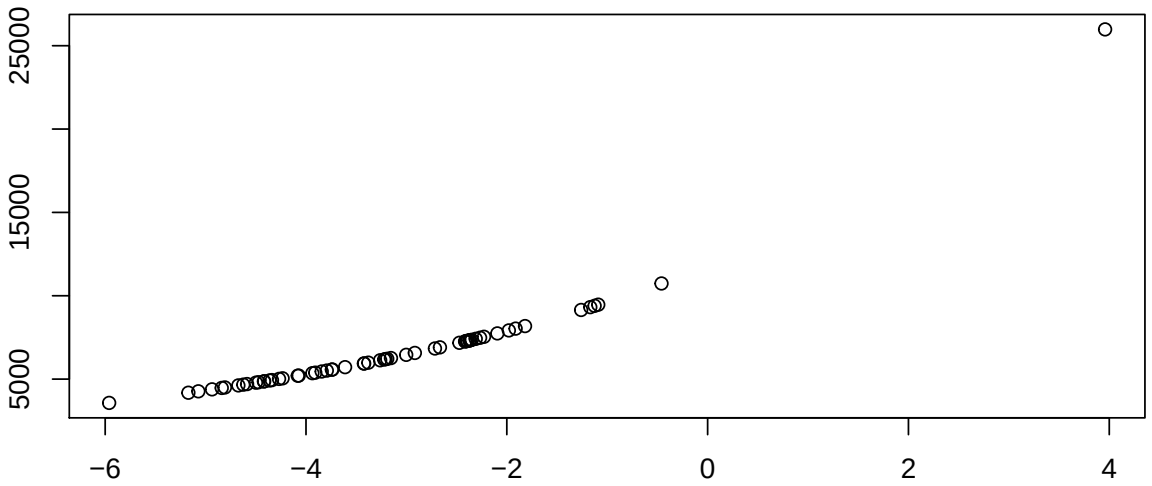
An easy way to get a quick estimate of h is to use the R function `bw.nrd0`. The kernels will be calculated on a daily basis using the time index of each data point. Therefore, the bandwidth should be calculated from the same data, i.e. a vector of intraday times, for example in seconds. For data consisting of 5-minute returns, this would be

```
h1 = bw.nrd0((1:nrow(sample_returns_5min))*(5*60))
vol3 <- spotvol(sample_returns_5min, method = "kernel", h = h1)
```

Another quick estimator has been implemented, one that multiplies the above simple estimate by the quarticity of the returns on each day. It can be used by the option `est =`

`"quarticity"`. The quarticity is a measure of the variance of the spot volatility. Days with higher quarticity will be assigned a larger bandwidth, to reduce the influence of single large returns. When using the method "kernel", in addition to the spot volatility estimates, all used values of the bandwidth h are returned, so we can compare them to the quarticity values:

```
quarticity = (nrow(sample_returns_5min)/3) * rowSums(sample_returns_5min^4)
vol4 <- spotvol(sample_returns_5min, method = "kernel", est = "quarticity")
plot(log(quarticity), vol4$par$h)
```

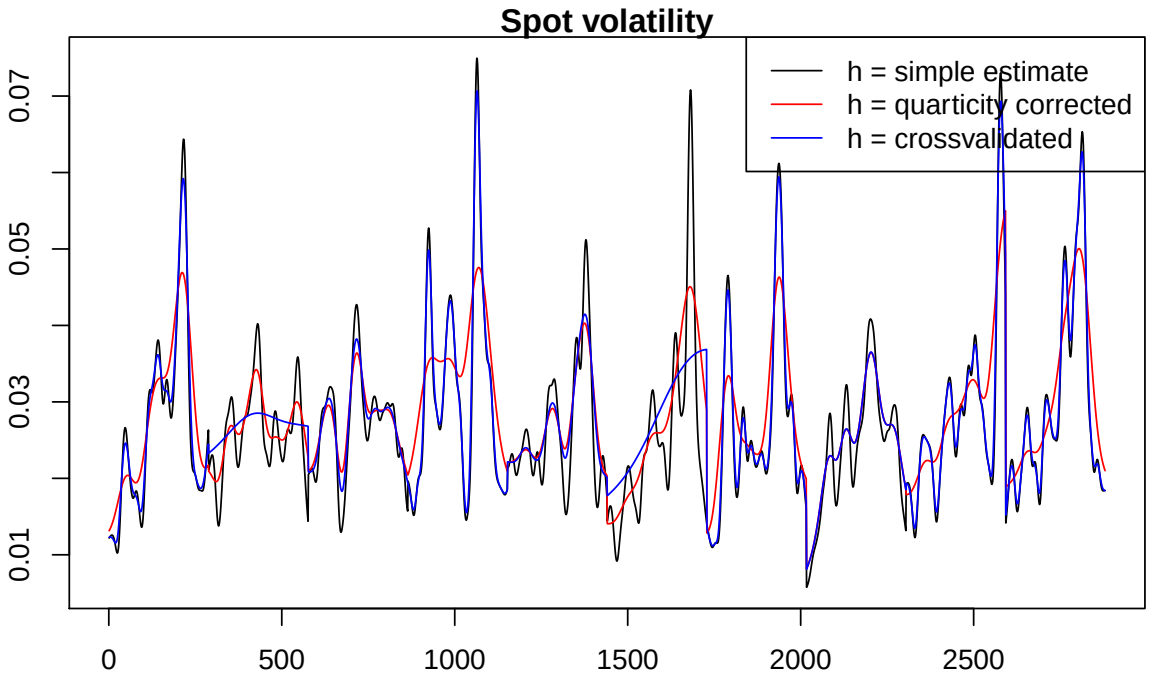


One way to estimate h, is by using cross-validation. For each day in the sample, h is chosen as to minimize the Integrated Square Error, which is a function of h. However, this function often has multiple local minima, or no minima at all ($h \to \infty$). To ensure a reasonable optimum is reached, strict boundaries have to be imposed on h. These can be specified by lower and upper, which by default are $0.1n^{-0.2}$ and $n^{-0.2}$ respectively, where n is the number of observations in a day. Crossvalidation can be used by specifying the option `est = "cv"`:

```
vol5 <- spotvol(sample_returns_5min, method = "kernel", est = "cv")
```

Now, we can easily compare the estimates of each bandwidth selection method by adding them to the same plot (`plot.spotvol` does not show any additional plots besides the spot volatility estimates for this method):

```
plot(vol3, length = 2880)
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)
```



This plot compares the spot volatility estimates of the first 10 days. The simple estimator usually yields the smallest bandwidth, which makes it more sensitive to shocks. On the 2nd and 6th day (observations 289-576 and 1441-1728 in the plot, respectively), the crossvalidation estimate has hit its upper boundary, meaning the minimization problem had no feasible solution on those days.

Piecewise constant volatility - `"piecewise"`

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.

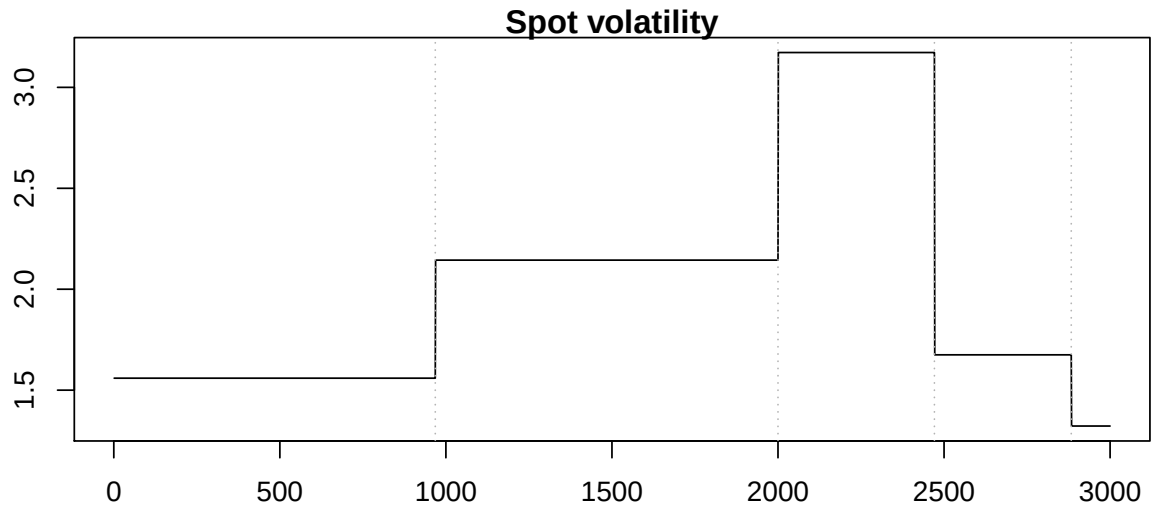
An example from Fried (2012) consists of simulated data from the student's t-distribution. The volatility is equal to $\sqrt{(5/3)}$ for the first 1000 observations, $1.5\sqrt{(5/3)}$ from 1001 to 2000, $2\sqrt{(5/3)}$ from 2001 to 2500, and $\sqrt{(5/3)}$ again for the last 500 observations. We organize these simulated values in a matrix containing 500 observations per row:

```
simdata <- matrix(sqrt(5/3) * rt(3000, df = 5), ncol = 500, byrow = TRUE)
simdata <- c(1, 1, 1.5, 1.5, 2, 1) * simdata
```

When we apply the piecewise constant volatility method the resulting `spotvol` object will contain both the spot volatility estimates and the detected change points in the volatility level. When plotting it, these change points will be visualized:

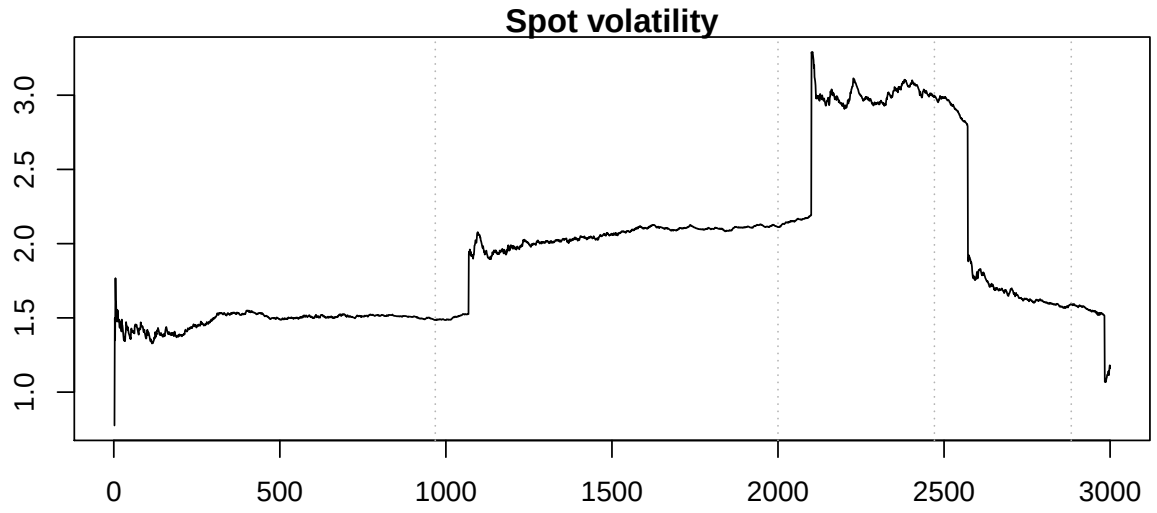
```
vol6 <- spotvol(simdata, method = "piecewise", m = 200, n = 100,
    online = FALSE)

## Detecting change points...
```

```
## Change detected at observation 968 ...
## Change detected at observation 2000 ...
## Change detected at observation 2471 ...
## Change detected at observation 2883 ...

plot(vol6)
```



Note that these results depend on randomly generated values, which can vary when you repeat this example. Here, we applied ex post estimation, using all observations between two change points to estimate the volatility. Another option is to apply online estimation, where only observations up to point t are used in the estimation of the volatility at point t. This results in slowly changing estimates, depending on the lengths m and n of the reference and test windows:

```
vol7 <- spotvol(simdata, method = "piecewise", m = 200, n = 100,
    online = TRUE, volest = "tau")

## Detecting change points...
## Change detected at observation 968 ...
## Change detected at observation 2000 ...
## Change detected at observation 2471 ...
## Change detected at observation 2883 ...

plot(vol7)
```



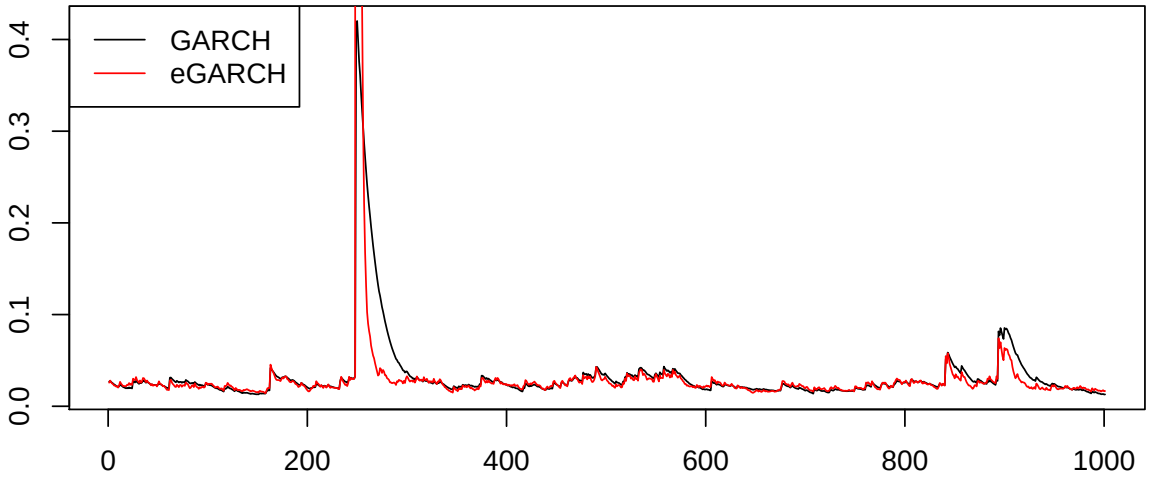
This can lead to a few unreliable estimates after each change point, when only a couple of observations are available. We also used a different estimator of the volatility for this plot. It can be specified by the option `volest`.

GARCH with intraday seasonality - `"garch"`

The package also includes an option to apply GARCH models, implemented by the **rugarch** package (Ghalanos 2014), to estimate spot volatility from intraday data. This is done by including external regressors in the GARCH model. These regressors are based on a flexible Fourier form, which was also used in the stochastic and deterministic periodicity estimation methods.

The **rugarch** package offers several GARCH specifications, which can be chosen from using the `model` argument in the `spotvol` function. Depending on the data, some models will not reach convergence in the minimization algorithm though, so it is advised to try multiple specifications. As an example, we will compare a standard GARCH(1,1) model to an eGARCH(1,1) specification:

```
vol8 <- spotvol(sample_returns_5min, method = "garch", model = "sGARCH",
    solver.control = list(maxeval = 1000))

## Fitting sGARCH model...

vol9 <- spotvol(sample_returns_5min, method = "garch", model = "eGARCH",
    solver.control = list(maxeval = 1000))

## Fitting eGARCH model...

plot(as.numeric(t(vol8$spot))[6000:7000], type = "l")
lines(as.numeric(t(vol9$spot))[6000:7000], col = "red")
legend("topleft", c("GARCH", "eGARCH"), col = c("black", "red"),
    lty = 1)
```



This plot highlights the huge effect an outlier can have on the estimation. The GARCH models seem particulary susceptible to these, the eGARCH in this case even more so than the standard GARCH model.

6. Volatility forecasting

It is broadly accepted by academic researchers that, when appropriately managed, access to high-frequency data leads to a comparative advantage in better predicting the volatility of future price evolution and hence the generation of alpha through quantitative investment models. Already in 2003 Fleming *et al.* (2003) estimated that an investor would be willing to pay 50 to 200 basis points per year to capture the gains in portfolio performance, from using high-frequency returns instead of daily returns for volatility forecasting. His findings were later confirmed in a somewhat different setting by the research of De Pooter *et al.* (2008).

In this section we discuss 2 classes of univariate forecasting models implemented in the `highfrequency` package that have highfrequency data as input: (i) the Heterogeneous Autoregressive model for Realized Volatility (HAR) discussed in Andersen *et al.* (2007) and Corsi (2009a) and (ii) the HEAVY model (High frEquency bAsed VolatilitY) introduced in Shephard and Sheppard (2010). The goal of both models is roughly speaking to predict the next days volatility based on the highfrequency price evolutions prior to that day.

Although the HAR and the HEAVY model have the same goal, i.e. modeling the conditional volatility, they take a different approach. Whereas the HAR model is focused on predicting the open-to-close variation, the HEAVY model has two equations: one focused on the open-to-close variation, one focused on the close-to-close variation. The main advantages of the HAR model are among other things, that it is simple and easy to estimate (since it is essentially a special type of linear model that can be estimated by least squares), even though it still manages to reproduce typical features found in volatility such as the long memory, fat tails, etc. The main advantages of the HEAVY model are among other things, that it models both the close-to-close conditional variance, as well as the conditional expectation of the open-to-close variation. Furthermore, the HEAVY model has momentum and mean reversion effects, and it adjusts quickly to structural breaks in the level of the volatility process. In contrast to the HAR model, the estimation of the HEAVY model is done by Guassian quasi-maximum

likelihood, for both equations in the model separately.

The next two sections review the HAR model and HEAVY model each in more detail, and of course discuss and illustrate how `highfrequency` can be used to estimate these models.

6.1. The HAR model

Corsi (2009b) proposed the Heterogeneous Autoregressive (HAR) model for Realized Volatility. As a first step, intraday returns are used to measure the daily volatility using e.g. simply Realized Volatility and typically ignoring the overnight return. As a second step, the realized volatility is parameterized as a linear function of the lagged realized volatilities over different horizons. Typically, the daily, weekly and monthly horizons are chosen to aggregate the realized volatility over. The simplest version of the model, as proposed in Corsi (2009b), is referred to as the HAR-RV model. In an interesting paper, Andersen *et al.* (2007) extend the model of Corsi (2009b) by explicitly including the contribution of jumps to the daily Realized Volatility in the model. Their analysis suggests that the volatility originating from jumps in the price level is less persistent than the volatility originating from the continuous component of Realized Volatility, and that separating the rough jump moves from the smooth continuous moves enables better forecasting results. They propose mainly two ways to model this idea, and both are implemented in the `highfrequency` package. In what follows, we first discuss the different versions of the HAR model in more detail, and then we discuss and illustrate how `highfrequency` can be used to estimate them.

The `harModel` function in `highfrequency` implements the Heterogeneous Autoregressive model for Realized Volatility discussed in Andersen *et al.* (2007) and Corsi (2009a). Let $r_{t,i}$ the i-th intraday return on day t, for $i = 1, \ldots, M$ with M the number of intraday returns per day. The realized volatility is then given by $RV_t = \sum_{i=1}^{M} r_{t,i}^2$. Denote by

$$RV_{t,t+h} = h^{-1}[RV_{t+1} + RV_{t+2} + \ldots + RV_{t+h}],$$

the Realized volatility aggregate over h days. Note that by definition $RV_{t,t+1} \equiv RV_{t+1}$.

The HARRV model

The most basic volatility model discussed Andersen *et al.* (2007) and Corsi (2009a) is given by:

$$RV_{t,t+h} = \beta_0 + \beta_D RV_t + \beta_W RV_{t-5,t} + \beta_M RV_{t-22,t} + \epsilon_{t,t+h},$$

for $t = 1, 2, \ldots, T$. Typically $h = 1$, and the model simplifies to

$$RV_{t+1} = \beta_0 + \beta_D RV_t + \beta_W RV_{t-5,t} + \beta_M RV_{t-22,t} + \epsilon_{t+1}.$$

The HARRVJ model

In the presence of jumps in intraday price process, the traditional realized volatility measures are dominated by the contribution of the jumps to volatility. Therefore, robust measure of volatility were developed in order to estimate only the "smooth" price variation (roughly speaking). The contribution of the jumps to the quadratic price process can then be estimated by $J_t = \max[RV_t - BPV_t, 0]$. The HAR-RV-J model is then given by

$$RV_{t,t+h} = \beta_0 + \beta_D RV_t + \beta_W RV_{t-5,t} + \beta_M RV_{t-22,t} + J_t + \epsilon_{t,t+h}.$$

The HARRVCJ model

Andersen *et al.* (2007) argue that it may be desirable to treat small jumps as measurement errors, or part of the continuous sample path variation process, associating only large values of $(RV_t - BPV_t)$ with the jump component. Therefore, they define Z_t as

$$Z_t = (1/n)^{-1/2} \times \frac{(RV_t - BPV_t)RV_t^{-1}}{[(\mu_1^{-4} + 2\mu_1^{-2} - 5)\max\{1, TQ_t BPV_t^{-2}\}]^{1/2}}, \tag{2}$$

where $TQ_t = M \times \mu_{4/3}^{-3} \sum_{j=3}^{M} |r_{t,j}|^{4/3} |r_{t,(j-1)}|^{4/3} |r_{t,(j-2)}|^{4/3}$, with $\mu_p = E(|z|^p)$. Suppose you identify the "significant" jumps by the realizations of Z_t in excess of some critical value, say Φ_α,

$$J_t = I[Z_t < \Phi_\alpha] RV_t + I[Z_t \leq \Phi_\alpha] BPV_t,$$

where $I[\cdot]$ denotes the indicator function. To ensure that the estimated continuous sample path component variation and jump variation sum to the total realized variation, we follow Andersen *et al.* (2007) in estimating the former component as the residual,

$$C_t = I[Z_t \leq \Phi_\alpha] RV_t + I[Z_t > \Phi_\alpha] BPV_t.$$

The HAR-RV-CJ model is then given by:

$$RV_{t,t+h} = \beta_0 + \beta_{CD} C_t + \beta_{CW} C_{t-5,t} + \beta_{CM} C_{t-22,t} + J_t + \epsilon_{t,t+h}.$$

Usage and arguments

The `harModel` function in `highfrequency` takes as `data` input an xts object containing the intraday returns. You can specify the type of HAR model that you want to estimate by setting the `type` argument. By default, the most basic model is chosen `type='HARRV'`. Other valid options are `type='HARRVJ'` or `type='HARRVCJ'` (see above for a discussion of these models).

Based on the intraday returns, daily realized measures are calculated. With the argument `RVest`, you can set which volatility estimators should be used to estimate both the daily integrated variance (non-jump-robust) and the continuous component of volatility. By default, total daily volatility is measured by Realized Volatility and the continuous component is measured by the Realized Bipower Variation, hence `RVest=c(rCov, rBPCov)`, but users can set estimators to their liking (see Section 4 for a discussion on the implemented Realized Volatility measures).

The daily volatility measures are subsequently aggregated over different time horizons. Use the arguments to `periods` and `periodsJ` to specify the horizons over which the continuous/total volatility and the jump component should be aggregated. Typically, the daily, weekly and monthly frequency are used, i.e. one, five and twenty-two days which translates into the default setting: `periods=c(1,5,22)` and `periodsJ=c(1,5,22)`.

Corsi and Reno (2012) propose to further extend the HAR model with a leverage component (Leverage Heterogeneous Auto-Regressive model). This allows to take into account that negative returns affect future volatility differently than positive returns. By default, the leverage effect is not taken into account and the argument `leverage=NULL`. In case you want to mimic the analysis in Corsi and Reno (2012), with leverage components aggregated on the daily, weekly and monthly frequency, just set `leverage=c(1,5,22)`.

In the `HARRVCJ` a test determines whether the contribution of the jumps is statistically significant. The argument `jumptest` should be set to the function name of the test to determine whether the jump variability is significant that day. By default `jumptest='ABDJumptest'`, hence using the test statistic in Equation (2) (or Equation (18) of Andersen *et al.* (2007)). The argument `alpha` can then be used to set the confidence level used in testing for jumps, which is set to the traditional " 5%" by default.

Often, the object will be to model the volatility for the next day, but this needn't be the case. Use the argument `h` to set the time horizon over which you want to aggregate the dependent variable, e.g. `h=5` in case you are interested in modeling the weekly realized volatility.

To conclude, we note that in applications of the HAR model sometimes the dependent and explanatory variables are transformed by taking the logarithm or the square root. Set the argument `transform='log'` or `transform='sqrt'` or any other function to transform all variables right before the linear model is estimated. By default, no transformation is done and `transform=NULL`.

Examples

Fitting the HARRV model on the Dow Jones Industrial Average in 2008
As a first step, we load the daily realized volatility for the Dow Jones Industrial average and select only 2008. The sample dataset `realized_library` in `highfrequency` contains a very rich set of realized volatility measures computed by Heber *et al.* (2009). `highfrequency` only contains a subset of the full realized library, which can be found on their website: `http://realized.oxford-man.ox.ac.uk`.

```
> data(realized_library); #Get sample daily Realized Volatility data
> DJI_RV = realized_library$Dow.Jones.Industrials.Realized.Variance; #Select DJI
> DJI_RV = DJI_RV[!is.na(DJI_RV)]; #Remove NA's
> DJI_RV = DJI_RV['2008'];
```

As a second step, we compute the traditional Heterogeneous AutoRegressive model (HAR). The output of the model is an S3 object. Since the HAR-model is simply a special type of linear model, it is also implemented that way: the output of the `harModel` function is a sublclass `harModel` of `lm`, the standard class for linear models. Figure 1 plots the output object of the `harModel` function, which has time on the horizontal axis and the observed realized volatility and the forecasted realized volatility on the vertical axis (of course this analysis is in-sample, but the estimated coefficients of the model can be used for real out-of-sample forecasts obviously). It is clear from a visual inspection of Figure 1 that one of the features of the `harModel` is that it can adapt relatively fast to changes in the volatility level, which explains its popularity in recent years.

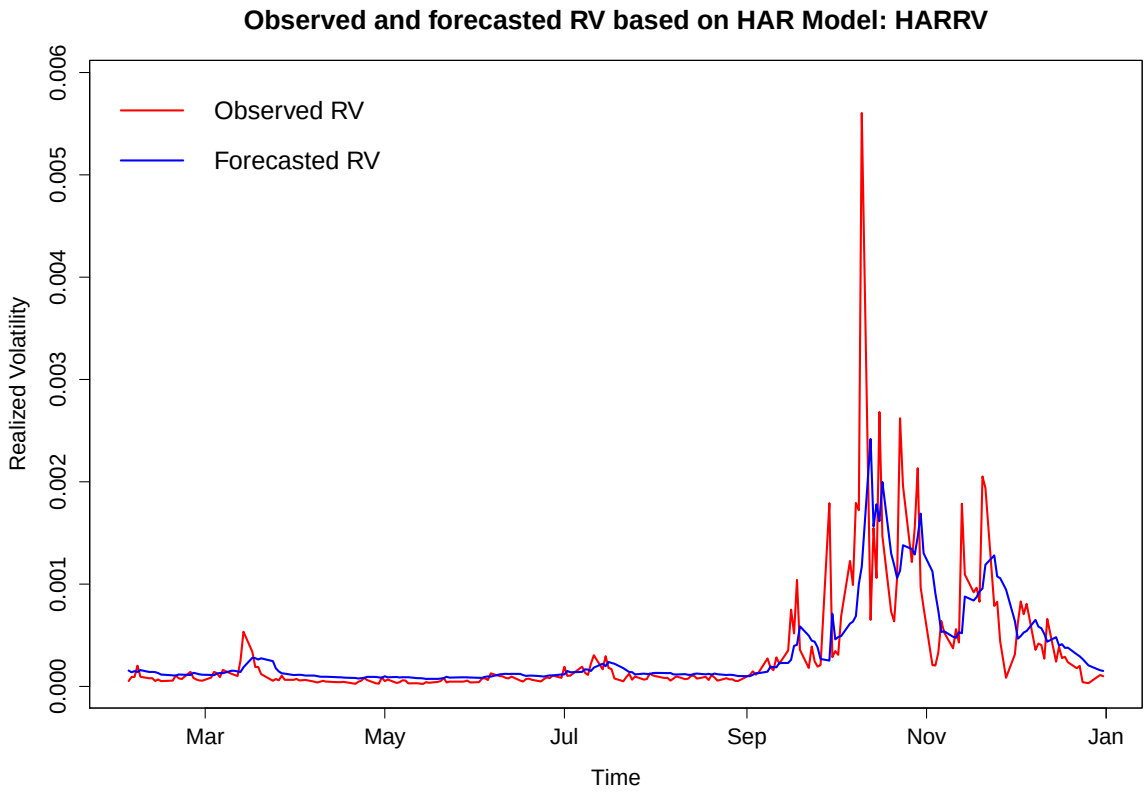
```
> x = harModel(data=DJI_RV , periods = c(1,5,22), RVest = c("rCov"),
+                          type="HARRV",h=1,transform=NULL);
> class(x);

[1] "harModel" "lm"

> x;
```

```
Model:
RV1 = beta0  +  beta1 * RV1 +  beta2 * RV5 +  beta3 * RV22

Coefficients:
    beta0      beta1      beta2      beta3
4.432e-05  1.586e-01  6.213e-01  8.721e-02


    r.squared  adj.r.squared
       0.4679         0.4608

> summary(x);

Call:
"RV1 = beta0  +  beta1 * RV1 +  beta2 * RV5 +  beta3 * RV22"

Residuals:
       Min         1Q     Median         3Q        Max
-0.0017683 -0.0000626 -0.0000427 -0.0000087  0.0044331

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
beta0 4.432e-05  3.695e-05   1.200   0.2315
beta1 1.586e-01  8.089e-02   1.960   0.0512 .
beta2 6.213e-01  1.362e-01   4.560 8.36e-06 ***
beta3 8.721e-02  1.217e-01   0.716   0.4745
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 0.0004344 on 227 degrees of freedom
Multiple R-squared: 0.4679,        Adjusted R-squared: 0.4608
F-statistic: 66.53 on 3 and 227 DF,  p-value: < 2.2e-16

> plot(x);
```

Fitting the HARRVCJ model on small example dataset
Using `highfrequency`, it is also very easy to estimate more sophisticated versions of the harModel. For example, the HARRVCJ model discussed in Andersen *et al.* (2007) can be easily estimated with a small example dataset as follows:

```
> data("sample_5minprices_jumps");
> data = sample_5minprices_jumps[,1];
> data = makeReturns(data); #Get the high-frequency return data
> x      = harModel(data, periods = c(1,5,10), periodsJ=c(1,5,10),
+                   RVest = c("rCov","rBPCov"), type="HARRVCJ",
+                   transform="sqrt");
> x
```

Figure 1: HAR-model for the Realized Volatility of the DJIA in 2008




```
Model:
sqrt(RV1) = beta0  +  beta1 * sqrt(C1) +  beta2 * sqrt(C5) +  beta3 * sqrt(C10)
   +  beta4 * sqrt(J1) +  beta5 * sqrt(J5) +  beta6 * sqrt(J10)

Coefficients:
   beta0     beta1     beta2     beta3     beta4     beta5
 -0.8835    1.1957  -25.1922   38.9909   -0.4483    0.8084
   beta6
 -6.8305

    r.squared  adj.r.squared
       0.9915         0.9661
```

This last example shows how easily a special type of HAR model can be estimated having solely the intraday returns as input. Note that this last example purely serves a didactic purpose: the high R-squared can be explained by the small number of observastions (i.e. days) in the dataset.

6.2. The HEAVY model

Shephard and Sheppard (2010) introduced the HEAVY model (High frEquency bAsed VolatilitY) to leverage highfrequency data for volatility modeling. Essentially, the model boils down to two equations: Equation (3) models the close-to-close conditional variance, while Equation (4) models the conditional expectation of the open-to-close variation. Denote by $\mathcal{F}_{t-1}^{HF}$ the past high frequency data (i.e. all intraday returns) and by $\mathcal{F}_{t-1}^{LF}$ the past low frequency data (i.e. daily returns). The HEAVY model is now based on $\mathcal{F}_{t-1}^{HF}$ in contrast to traditional GARCH models that use only $\mathcal{F}_{t-1}^{LF}$.

Denote the daily returns:

$$r_1, r_2, \ldots, r_T.$$

Based on intraday data, one can calculate the daily realized measures:

$$RM_1, RM_2, \ldots, RM_T,$$

with T the total number of days in the sample. The heavy model is given by:

$$\mathrm{Var}(r_t|\mathcal{F}_{t-1}^{HF}) = h_t = \omega + \alpha RM_{t-1} + \beta h_{t-1}, \quad \omega, \alpha \geq 0 \text{ and } \beta \in [0,1], \tag{3}$$

$$\mathrm{E}(RM_t|\mathcal{F}_{t-1}^{HF}) = \theta_t = \omega_R + \alpha_R RM_{t-1} + \beta_R \theta_{t-1}, \quad \omega_R, \alpha_R, \beta_R \geq 0 \text{ and } \alpha_R + \beta_R \in [0,1]. \tag{4}$$

Equation (3) models the close-to-close conditional variance, and equation (4) models the conditional expectation of the open-to-close variation. In matrix notation, the model becomes:

$$\begin{pmatrix} h_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} \omega \\ \omega_R \end{pmatrix} + \begin{pmatrix} 0 & \alpha \\ 0 & \alpha_R \end{pmatrix} \begin{pmatrix} r_{t-1}^2 \\ RM_{t-1} \end{pmatrix} + \begin{pmatrix} \beta & 0 \\ 0 & \beta_R \end{pmatrix} \begin{pmatrix} h_{t-1} \\ \theta_{t-1} \end{pmatrix}. \tag{5}$$

The function `heavyModel` in the `highfrequency` package implements the above model in very general way. It enables the user the obtain the estimates for the parameters $(\omega, \omega_R, \alpha; \alpha_R, \beta, \beta_R)$ using Quasi-maximum likelihood.[4]

Usage and arguments

The `heavyModel`[5] takes a (T x K) matrix containing the data as input, with T the number of days. For the traditional HEAVY model: $K = 2$, the first column contains the squared demeaned daily returns, i.e. $r_1^2, \ldots, r_T^2$ and the second column contains the realized measures, i.e $RM_1, \ldots, RM_T$.

As you can see in Equation (5), the matrix notation of the HEAVY model contains two matrices with parameters named α and named β respectively. This matrix structure allows to see that the traditional HEAVY model given in (5) is just a special case of a larger set of models that can be written this way. You can use the arguments `p` and `q` from the `heavyModel` function to specify for each of these two matrices: (i) which parameters in the matrix should be estimated by setting a non-zero integer in that matrix cell (ii) how many lags should be included in the model. More formally: `p` is a $(K \times K)$ matrix containing the lag length for the

[4]The implementation is (loosely) based on the matlab code from Kevin Sheppard: http://www.kevinsheppard.com/wiki/MFE_Toolbox

[5]The implementation of the heavyModel is not completely finished. For the moment only bound constraints on the parameters are imposed in the optimization. Future developments also include outputting standard errors, and a c-implementation of the likelihood function to speed up the QML estimation.

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 introduced above p is given by (cfr. the matrix with the αs in Equation (5)):

$$p = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

Similarly to the matrix `p`, the matrix `q` is a $(K \times K)$ 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 (cfr. the matrix with the βs in Equation (5)):

$$q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The starting values that will be used in the optimization can be set by the argument `startingvalues`. The arguments `LB` and `UB` can be used to set the Upper and Lower Bound for the parameters. By default, the lower bound is set to zero and the upperbound to infinity. To specify how the estimation should be initialized you can use the `backcast` argument, which will be set to the unconditional estimates by default. Finally, the `compconst` argument is a boolean indicating how the ωs should be estimated. In case it is TRUE, ωs are estimated in the optimization, and in case it is FALSE volatility targeting is done and ω is just 1 minus the sum of all relevant αs and βs multiplied by the unconditional variance.

Output

The output of the `heavyModel` is a list. Most interestingly, the list-item `estparams` contains a matrix with the parameter estimates and their names. The order in which the parameters are reported is as follows: first the estimates for ω, then the estimates for the non-zero αs with the most recent lags first in case $\max(p) > 1$, then the estimates for the non-zero βs with the most recent lag first in case $\max(q) > 1$.

Whereas the `loglikelihood` list-item reports the total log likelihood, we offer users a bit more insight with the list-item `likelihoods` containing an xts-object with the daily log likelihood evaluated at the parameters. The list-item `convergence` provides more information on the convergence of the optimization, see `optim` for more information.

The list-item `condvar` is a $(T \times K)$ xts-object containing the conditional variances. For the traditional HEAVY model, Figure 2 plots for example the conditional close-to-close variance for the DJIA for 1996 up to 2009 (see next subsection for more information).

Examples

Fitting the HEAVY model on the Dow Jones Industrial Average from 1996 up to 2009
As a first step, we load the `realized_library` (Heber *et al.* 2009) which contains information on the Dow Jones Industrial Average. We then select from this library the daily returns and the daily Realized Kernel estimates (Barndorff-Nielsen *et al.* 2004). The data matrix that serves as an input for the `heavyModel` now has the returns as the first column and the Realized Kernel estimates in the second column. We further set the argument `backcast` to the variance of the daily returns and the average realized kernel over the sample period.We

now have all ingredients to estimate the HEAVY Model. Based on the output of the model, Figure 2 plots the conditional open-to-close variance, as estimated by the second equation in the model. Note the striking peak in 2008 at the start of the financial crisis.

```
>  # Implementation of the heavy model on DJIA:
>  data("realized_library");
>  returns  =  realized_library$Dow.Jones.Industrials.Returns;
>  rk       =  realized_library$Dow.Jones.Industrials.Realized.Kernel;
>  returns  = returns[!is.na(rk)];  rk = rk[!is.na(rk)]; # Remove NA's
>  data     = cbind( returns^2, rk );
>  backcast = matrix( c(var(returns),mean(rk)) ,ncol=1);
>
>  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);
>  output$estparams
             [,1]
omega1 0.01750506
omega2 0.06182249
alpha1 0.45118753
alpha2 0.41204541
beta1  0.73834594
beta2  0.56367558
```

7. Liquidity

7.1. Matching trades and quotes

Trades and quotes are often supplied as separate data objects. For many research and practical questions related to transaction data, one needs to merge trades and quotes. Since trades and quotes can be subject to different reporting lags, this is not a straightforward operation (Lee and Ready 1991). The function `matchTradesQuotes` can be used for matching trades and quotes. One should supply the number of seconds quotes are registered faster than trades. Based on the research of Vergote (2005), we set 2 seconds as the default.

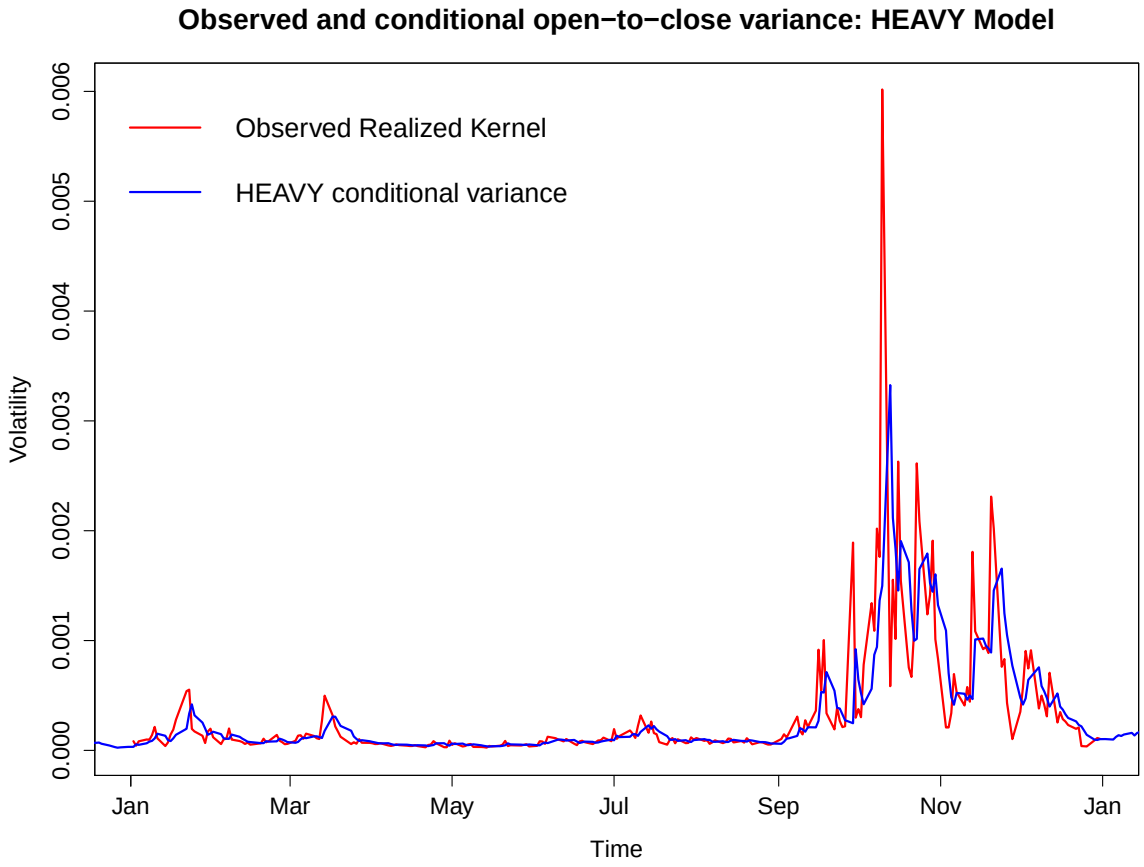
7.2. Inferred trade direction

Many trades and quotes databases do not indicate whether individual trades are market buy or market sell orders. `highfrequency` implements with `getTradeDirection` the Lee-Ready rule (Lee and Ready 1991) to infer the trade direction based on the matched trades and quotes.

7.3. Liquidity measures

Numerous liquidity measures can be calculated based on matched trade and quote data, using the function `tqLiquidity` (see Bessembinder (2003), Boehmer (2005), Hasbrouck and

Figure 2: HEAVY Model for the Realized Volatility of the DJIA in 2008.



Seppi (2001) and Venkataraman (2001) for more information on the implemented liquidity measures). The main implemented liquidity measures are listen in Table 4, and can be used as arguments of the function `tqLiquidity`.

The example below illustrates how to: (i) match trades and quotes, (ii) get the trade direction and (iii) calculate liquidity measures.

```
> #Load data samples
> data("sample_tdata");
> data("sample_qdata");
> #Match the trade and quote data
> tqdata = matchTradesQuotes(sample_tdata,sample_qdata);
> #Display information in tqdata
> colnames(tqdata)[1:6];

[1] "SYMBOL" "EX"     "PRICE"  "SIZE"   "COND"   "CORR"

> colnames(tqdata)[7:12];
```

Table 4: Overview of Liquidity measures

Argument(s)	Liquidity measures
es, rs	Compute effective (realized) spread
value_trade	Compute trade value (price × size)
signed_value_trade	Compute signed trade value
signed_trade_size	Compute signed trade size
di_diff, di_div	Compute depth imbalance
pes, prs	Compute proportional effective and realized spread
price_impact, prop_price_impact	Compute price impact
tspread, pts	Compute half traded and proportional half-traded spread
qs, logqs	Compute quoted spread
qsslope, logqslope	Compute quoted slope

```
[1] "G127"   "BID"    "BIDSIZ" "OFR"    "OFRSIZ" "MODE"
```

```
> #Get the inferred trade direction according to the Lee-Ready rule
> x = getTradeDirection(tqdata);
> #Calculate the proportional realized spread:
> prs = tqLiquidity(tqdata,sample_tdata,sample_qdata,type="prs");
> #Calculate the effective spread:
> es = tqLiquidity(tqdata,type="es");
```

8. Acknowledgements:

We thank Google for the financial support through the Google summer of code 2012 program. Furthermore, we thank the participants of the 1st R/Rmetrics Summer School and 4th User/Developer Meeting on Computational Finance and Financial Engineering, Switzerland (2010) and the "R/Finance: Applied finance with R" conference, USA, Chicago (2010), for their comments and suggestions. In particular, we thank Christophe Croux, Brian Peterson and Eric Zivot for their insights and help in developing this package. Financial support from the Flemish IWT (Institute for Science and Innovation), the National Bank of Belgium and the Research Fund K.U.Leuven is gratefully acknowledged.

References

Ait-Sahalia Y, Mykland PA, Zhang L (2005). "A Tale of Two Time Scales: Determining Integrated Volatility With Noisy High-Frequency Data." *Journal of the American Statistical Association*, **100**, 1394–1411.

Andersen TG, Bollerslev T (1997). "Intraday periodicity and volatility persistence in financial markets." *Journal of Empirical Finance*, **4**, 115–158.

Andersen TG, Bollerslev T, Diebold F (2007). "Roughing it up: including jump components in the measurement, modelling and forecasting of return volatility." *The Review of Economics and Statistics*, **89**(4), 701–720.

Andersen TG, Bollerslev T, Diebold F, Labys P (2003). "Modeling and forecasting realized volatility." *Econometrica*, **71**, 579–625.

Andersen TG, Dobrev D, Schaumburg E (2012). "Jump-Robust Volatility Estimation using Nearest Neighbor Truncation." *Journal of Econometrics*, **169**, 75–93.

Barndorff-Nielsen OE, Hansen PR, Lunde A, Shephard N (2004). "Regular and modified kernel-based estimators of integrated variance: The case with independent noise." *Working paper*.

Barndorff-Nielsen OE, Hansen PR, Lunde A, Shephard N (2008). "Realised Kernels in Practice: Trades and Quotes." *Econometrics Journal*, **4**, 1–32.

Barndorff-Nielsen OE, Hansen PR, Lunde A, Shephard N (2011). "Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading." *Journal of Econometrics*, **162**, 149–169.

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

Beltratti A, Morana C (2001). "Deterministic and Stochastic Methods for Estimation of Intra-day Seasonal Components with High Frequency Data." *Economic Notes*, **30**(2), 205 – 234.

Bessembinder H (2003). "Issues in Assessing Trade Execution Costs." *Journal of Financial Markets*, **6**, 223–257.

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

Boudt K, Croux C, Laurent S (2011a). "Outlyingness weighted covariation." *Journal of Financial Econometrics*, **9**, 657–684.

Boudt K, Croux C, Laurent S (2011b). "Robust estimation of intraweek periodicity in volatility and jump detection." *Journal of Empirical Finance*, **18**, 353–367.

Boudt K, Zhang J (2010). "Jump robust two time scale covariance estimation and realized volatility budgets." *Mimeo*.

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

Cornelissen J, Boudt K (2012). *RTAQ: Tools for the analysis of trades and quotes in R*. R package version 0.2, URL http://CRAN.R-project.org/package=RTAQ.

Corsi F (2009a). "A Simple Approximate Long Memory Model of Realized Volatility." *Journal of Financial Econometrics*, **7**, 174–196.

Corsi F (2009b). "A Simple Long Memory Model of Realized Volatility." *Journal of Financial Econometrics*, **7**, 174–196.

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

De Pooter M, Martens MP, Van Dijk DJ (2008). "Predicting the Daily Covariance Matrix for S&P 100 Stocks using Intraday Data - But Which Frequency to Use?" *Econometric Reviews*, **27**, 199–229.

Fleming J, Kirby C, Ostdiek B (2003). "The Economic Value of Volatility Timing Using "Realized" Volatility." *Journal of Financial Economics*, **67**, 473–509.

Fried R (2012). "On the Online Estimation of Local Constant Volatilities." *Computation Statistics and Data Analysis*, **56**, 3080–3090.

Ghalanos A (2014). *rugarch: Univariate GARCH models.* R package version 1.3-3., URL http://CRAN.R-project.org/package=rugarch.

Gobbi F, Mancini C (2009). "Identifying the covariation between the diffusion parts and the co-jumps given discrete observations." *Mimeo*.

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

Hasbrouck J, Seppi DJ (2001). "Common Factors in Prices, Order Flows and Liquidity." *Journal of Financial Economics*, **59**, 383–411.

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

Heber G, Lunde A, Shephard N, Sheppard K (2009). "Oxford-Man Institute's realized library, version 0.1." *", Oxford-Man Institute, University of Oxford*.

Kristensen D (2010). "Nonparametric Filtering of the Realized Spot Volatility: A Kernel-based Approach." *Econometric Theory*, **26**, 60–93.

Lee CMC, Ready MJ (1991). "Inferring Trade Direction from Intraday Data." *Journal of Finance*, **46**, 733–746.

Luethi D, Erb P, Otziger S (2014). *FKF: Fast Kalman Filter.* R package version 0.1.3, URL http://CRAN.R-project.org/package=FKF.

Payseur S (2008). *realized: Realized.* R package version 0.81, URL http://cran.r-project.org/web/packages/realized/realized.pdf.

Ryan JA (2011). *quantmod: Quantitative Financial Modelling Framework.* R package version 0.3-17, URL http://CRAN.R-project.org/package=quantmod.

Ryan JA, Ulrich JM (2009). *xts: Extensible Time Series.* R package version 0.6-7, URL http://CRAN.R-project.org/package=xts.

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

Taylor S, Xu X (1997). "The Incremental Volatility Information in One Million Foreign Exchange Quotations." *Journal of Empirical Finance*, **4**, 317 – 340.

Venkataraman K (2001). "Automated Versus Floor Trading: An Analysis of Execution Costs on the Paris and New York Exchanges." *The Journal of Finance*, **56**, 1445–1485.

Vergote O (2005). "How to Match Trades and Quotes for NYSE Stocks?" *K.U.Leuven working paper*.

Yan B, Zivot E (2003). "Analysis of High-Frequency Financial Data with S-Plus." *UWEC-2005-03*. URL http://ideas.repec.org/p/udb/wpaper/uwec-2005-03.html.

Zhang L (2011). "Estimating covariation: Epps effect, microstructure noise." *Journal of Econometrics*, **160**, 33–47.

Affiliation:

Kris Boudt
VU University Amsterdam, Lessius and K.U.Leuven, Belgium
E-mail: kris.boudt@kuleuven.be

Jonathan Cornelissen
K.U.Leuven, Belgium
E-mail: jonathan.cornelissen@kuleuven.be

Scott Payseur
Director at UBS Global Asset Management
E-mail: scott.payseur@gmail.com

Maarten Schermer
GSoC Student
E-mail: m.j.a.schermer@gmail.com