




Analyzing Intraday Financial Data in R: The `highfrequency` Package

Kris Boudt 
Ghent University
Vrije Universiteit Brussel
Vrije Universiteit Amsterdam

Onno Kleen 
Erasmus University
Rotterdam

Emil Sjørup

Abstract

The `highfrequency` package for the R programming language provides functionality for pre-processing financial high-frequency data, analyzing intraday stock returns, and forecasting stock market volatility. For academics and practitioners alike, it provides a tool chain required to work with such datasets and to conduct statistical analyses dedicated to spot volatility, jumps, realized measures, and many more. We showcase our implemented routines and models on raw high-frequency data from large stock exchanges.

Keywords: financial markets, high-frequency data, jumps, realized measures, R.

1. Introduction

During the last 30 years, more and more financial intraday data has become available and employed by academics, regulators, and financial firms alike. This led to the creation of a large branch in the financial econometrics and time series analysis literature, namely high-frequency econometrics. However, working with high-frequency data has many pitfalls. For example, the trades and quotes data by the New York Stock Exchange (NYSE) needs extensive cleaning, the data arrives at irregular time intervals, and the statistical tools developed for analyzing the cleaned data are often non-standard. With this package, we enable users of high-frequency data to focus on conducting analyses by aligning the process in two dimensions: (1) The data preparation process which is tedious but necessary is streamlined and becomes less prone to error. (2) We provide a wide array of estimators and procedures developed in the financial econometrics literature in one statistical language. Our value proposition is strong because we provide tools which can work as a pipeline from data ingestion all the way to analysis output, or anywhere in between.

The **highfrequency** package (Boudt, Cornelissen, Payseur, Kleen, and Sjørup 2022) for the R programming language (R Core Team 2022) provides numerous data handling and visualization tools along with many estimators of various measures and forecasting methods. These functions cover a wide array of overall topics in the high-frequency econometrics literature and are based on a wide variety of the most influential papers within these topics. Package **highfrequency** is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=highfrequency>.

While a limited number of estimators implemented in the **highfrequency** package are available elsewhere, there is to the best of our knowledge no software library with comparable functionality publicly available in one statistical environment. For downloading financial intraday data, there is the `getSymbols` function in the **quantmod** package (Ryan and Ulrich 2022), which enables users to download open-high-low-close data, or the **alphavantage** package (Dancho and Vaughan 2020), which enables users to download financial data up to the 1-minute frequency. The **HighFreq** package (Pawlowski 2021) provides functionality for aggregating intraday data to a lower frequency. Similarly, the **tidyquant** package (Dancho and Vaughan 2022) can handle intraday data but is mostly dedicated to work with lower-frequency data. The realized generalized autoregressive conditional heteroskedasticity model implemented in the **rugarch** package (Ghalanos 2022) employs daily volatility measures based on intraday data but the data need to be already aggregated to a daily frequency.

Because the **highfrequency** package covers such an extensive set of features, we present only selected ones in this paper such that the setting broadly resembles an actual data analysis.

A significant contribution of the **highfrequency** package are routines for estimating realized measures of the return distribution. In the univariate case, the most popular type of estimators are realized variance estimators; discussed by, among many other, Andersen, Bollerslev, Diebold, and Labys (2001), Barndorff-Nielsen, Hansen, Lunde, and Shephard (2008), and Barndorff-Nielsen and Shephard (2004b). Many different estimators have been proposed to deal with peculiarities of high-frequency financial data such as uneven sampling rate, jumps, and measurement errors. The class of realized volatility estimators have helped to improve the understanding of the dynamics of volatility of securities prices as well as contributed to improve forecast accuracy. Estimators of higher moments of the return distribution, in particular skewness and kurtosis, were introduced in Amaya, Christoffersen, Jacobs, and Vasquez (2015). In the multivariate case there are also several covariance estimators, for example those introduced by Barndorff-Nielsen and Shephard (2004a), Hayashi and Yoshida (2005), and Zhang, Mykland, and Ait-Sahalia (2005). All of these can be used to estimate daily realized covariance matrices. Such (possibly inverted) covariance matrices are very important in portfolio allocation and risk management.

Considering variance on a daily time-frame is only appropriate for actors who have an investment horizon which comes close to or exceeds one day. However, many actors in the financial markets have a horizon much, much shorter than these, ordering in the minutes, seconds, or below. From these actors stems a need to monitor the instantaneous volatility, also called the spot volatility. A non-parametric estimator of the spot volatility was introduced by Kristensen (2010). Since then, numerous estimators of the spot volatility have been introduced, many focusing on exploiting daily and weekly recurring patterns in the spot volatility (see, e.g., Boudt, Croux, and Laurent 2011).

Similarly, many tests have been developed for the presence of jumps that are discontinuous

elements in high-frequency price paths (see, among others, [Barndorff-Nielsen and Shephard 2006](#)). Jumps occur most often when new information about the underlying asset becomes available. A large number of jump tests are formed by constructing a test statistic that depends on a jump robust measure of integrated daily volatility and a non-robust counterpart. These tests are then used to identify whether a jump occurred during a whole trading day. A more granular counterpart is the test type introduced in [Lee and Mykland \(2008\)](#), who construct a test that is able to infer whether a single high-frequency return observation contains a jump. This type of test is based on spot estimators of the volatility and also optionally of the drift.

Another area of the literature is focused on modeling and forecasting realized measures, specifically realized measures of the variance. In the **highfrequency** package, we include two models for such purposes. The first model of the two is the heterogeneous autoregressive (HAR) model of [Corsi \(2009\)](#), which models the realized volatility through a restricted autoregressive model. There are numerous extensions to this model, and we include many of these. The second model is the high-frequency based volatility (HEAVY) model of [Shephard and Sheppard \(2010\)](#). This model combines return volatility and realized measures to improve volatility modeling. Evaluating forecast performance of these models can also be seen as a way to validate the practical importance of high-frequency based volatility estimates.

The rest of the paper is structured as follows. Section 2 presents high-frequency financial datasets and the particulars that separate these from “regular” financial datasets, including empirically motivated cleaning methods and data handling techniques. Section 3 introduces a method for estimating whether a buyer or a seller is the “aggressor” in a trade. This method is useful for estimators of the liquidity of an asset, which will round out the section. In Section 4, a theoretical framework for the high-frequency price process of a financial asset is presented. This price process will be used as the basis for the following sections. Section 5 introduces realized measures of the return distribution with a focus on the variance of asset returns. Then, we extend the price process of the previous section to the multivariate case and introduce estimators of the realized covariance. Throughout this section, we focus on pitfalls analysts will be facing when conducting univariate and multivariate analyses. In Section 6, we introduce and show example usage of estimators of the spot volatility. We tie the two previous sections together with Section 7, where we introduce jump tests based on both realized measures for an entire day and spot measures of volatility. In Section 8, we introduce two models used to forecast realized variances and compare their forecast performance in an empirical example. Section 9 concludes.

2. Processing raw high-frequency data

In this paper, we focus on the empirical aspects of high-frequency econometrics using the sample datasets included in the **highfrequency** package. Our datasets comprise both raw (i.e., “tick-by-tick”) and cleaned trade and quote data. The processed datasets contain cleaned univariate and multivariate tick-by-tick data, aggregated univariate and multivariate one-minute price data, and realized measures of volatility for the SPDR S&P500 ETF Trust (with ticker symbol SPY), which tracks the Standard and Poor’s (S&P) 500 stock market index.

In this context, trade data refers to asset prices of actual transactions and quote data refers to the best bid and ask prices as either reported by participants or observed in the centralized order book.

2.1. Raw tick-by-tick datasets

To show an example of raw trade data included in the package, we print the first and last four rows of the sample dataset `sampleTDataRaw`. The source of this dataset is the Daily TAQ (Trade and Quote) database run by the NYSE which includes every trade and every quote reported to the consolidated tape. The dataset `sampleTDataRaw` contains data for a single pseudonymized stock on 2–3 January 2018 downloaded via the Wharton Research Data Service (<https://wrds-www.wharton.upenn.edu/>) and it is supplied in the format of a `data.table` from the `data.table` package (Dowle and Srinivasan 2021). Note that for all data wrangling tasks and realized measure calculations, our package allows the data to be stored as `data.table` or `xts` (Ryan and Ulrich 2020). Especially for the data cleaning tasks, there is a gain of using the `data.table` package in terms of memory efficiency and speed.

The raw trade and quote data contain observations both before the market opens and after it closes. Pre-market trading in the United States, in terms of stocks, usually runs between 4:00 and 9:30 eastern standard time (EST) and after-hours trading typically runs from 16:00 to 20:00 EST. The United States stock exchanges are open from 9:30 to 16:00 EST. When applicable, time zones of all the data we include in the `highfrequency` package is local to the respective exchange(s) of the dataset.

```
R> options("digits" = 5)
```

```
R> data("sampleTDataRaw", package = "highfrequency")
R> print(sampleTDataRaw, topn = 4)
```

	DT	EX	SYMBOL	COND	SIZE	PRICE	CORR
1:	2018-01-02 05:01:21.479	P	XXX	FTI	2	157.80	0
2:	2018-01-02 05:23:50.188	P	XXX	FTI	3	157.80	0
3:	2018-01-02 05:23:50.236	P	XXX	FTI	1	157.80	0
4:	2018-01-02 07:11:54.065	P	XXX	T	130	158.00	0

77260:	2018-01-03 19:09:49.230	P	XXX	FT	100	157.25	0
77261:	2018-01-03 19:10:48.529	D	XXX	TI	20	157.47	0
77262:	2018-01-03 19:36:55.309	D	XXX	TI	5	157.15	0
77263:	2018-01-03 19:55:37.789	D	XXX	TI	15	157.45	0

The `SYMBOL` column contains a string identifying the symbol of the trade, the `DT` column represents date and time and contains a `POSIXct` timestamp, the `PRICE` column contains the prices of the trades. The `SIZE` column shows the number of shares traded. The `COND` column contains the sales condition of the corresponding trade as defined by the NYSE. The characters F, T, and I in our data example above indicate the trade being an intermarket sweep order, an extended hours trade, and/or an odd lot trade respectively.¹ the `EX` column shows the exchange of the trade, and `CORR` is a correction indicator. These columns are all used in the `highfrequency` package during data cleaning procedures. For an example of quote data included, we show the first and last four observations during market opening of the `sampleQDataRaw` dataset which contains raw quote data. The dataset is the quote counterpart to the trade data in `sampleTDataRaw`.

¹For the definition of all possible sale condition indicators, see <https://www.nyse.com/market-data/historical/daily-taq>.

```
R> data("sampleQDataRaw", package = "highfrequency")
R> print(exchangeHoursOnly(sampleQDataRaw), topn = 4)
```

	DT	EX	BID	BIDSIZ	OFR	OFRSIZ	SYMBOL
1:	2018-01-02 09:30:00.042	K	158.00	3	158.50	1	XXX
2:	2018-01-02 09:30:00.092	P	158.01	1	158.39	20	XXX
3:	2018-01-02 09:30:00.094	Z	158.25	1	158.80	5	XXX
4:	2018-01-02 09:30:00.115	N	158.39	1	158.50	18	XXX

130615:	2018-01-03 15:59:59.730	N	157.26	1	157.28	3	XXX
130616:	2018-01-03 15:59:59.809	N	157.26	1	157.28	4	XXX
130617:	2018-01-03 15:59:59.940	N	157.26	1	157.28	19	XXX
130618:	2018-01-03 15:59:59.950	N	157.26	1	157.28	20	XXX

The `SYMBOL` column contains a string identifying the symbol of the quote, the `DT` column represents date and time and contains a `POSIXct` timestamp, the `BID` column contains the bid of the quote, which is the price that the market participant posting the quote is willing to buy shares at, and the `BIDSIZ` column shows the number of shares that are available to be sold at the corresponding bid price. The `OFR` column contains the offer of the quote, which is the price that the participant posting the quote is willing to sell shares at, and the `OFRSIZ` column shows the number of shares that are available to be bought at the corresponding offer price.

2.2. Preparing raw price and quote data for analysis

The structure of raw high-frequency datasets depends on the type of data as well as numerous other factors. The exact nomenclature and format of columns also varies depending on the vendor. In the following, we assume the columns to be (re)named as in our sample datasets shown above.

Depending on the data vendor of the high-frequency data and the instrument type, the cleaning of the data required to conduct meaningful analysis varies greatly. Whether data cleaning is needed for the specific vendor and type of data typically becomes immediately apparent when looking at a plot of the data. Figure 1 shows the (mostly) raw trade and quote data from the pseudonymized datasets provided in the `highfrequency` package, `sampleTDataRaw` and `sampleQDataRaw`.² In the plot it becomes evident that the bid and offer prices (black dots) deviate substantially from the actual trade prices (red line). These observed prices clearly need to be cleaned. Additionally, we observe a distinct irregular glitch in the price process around 11:38 among other small outliers in the trade data. The former glitch turns out to still be present in our trade data after initial (trade data only) cleaning steps. Hence, this remaining outlier shows the need for filtering trade data after matching them to the prevailing quotes for ensuring that trades do not happen “too” far away from the bid and offer. The classical approach for cleaning trade and quote data is outlined in [Barndorff-Nielsen, Hansen, Lunde, and Shephard \(2009\)](#). This method seeks to retain as much data as possible while eliminating outliers. In the `highfrequency` package, functions are available to conduct

²The only data cleaning step applied is to remove all 64,579 quotes with a price equal zero – otherwise the plot becomes unreadable due to the wide range of the y -axis.

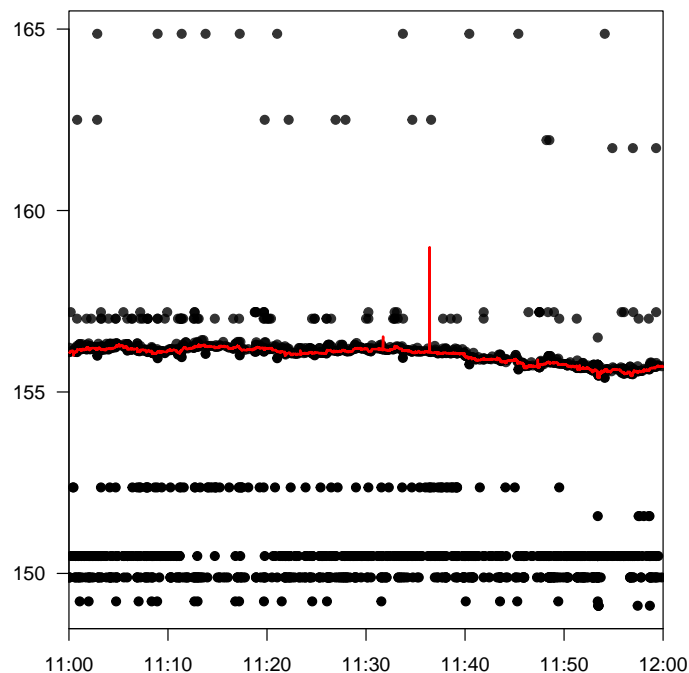


Figure 1: Raw trade and quote data. The plot shows 60 minutes of trading in the `sampleTDataRaw` dataset and the accompanying quotes from the `sampleQDataRaw` dataset. The black dots are bids and offers, while the red line represents transaction prices.

each step of the cleaning procedure separately, but we also provide convenience functions that take care of the entire procedure.

In order to clean trade data, users can apply the functions `noZeroPrices`, `exchangeHoursOnly`, `autoSelectExchangeTrades` (or `selectExchange`), `tradesCondition`, and `mergeTradesSameTimestamp` to their raw trade datasets. These functions delete entries with zero prices, retain only data during the official opening hours of the exchanges, select a single exchange either manually or automatically, and merge trades that occur at the same time, respectively. When these functions are applied to a raw trade dataset, the user has what we call a pre-cleaned trade dataset. The pre-cleaned trade dataset will in most cases have the most egregious outliers removed and therefore be relatively clean. Users can call the function `tradesCleanup` for applying the listed cleaning rules in consecutive order. However, in many cases additional cleaning is required to remove trades that happened outside the bid-offer band. Such trades are unreasonable if they happen “too far” below the bid or above the offer. To determine this we naturally need to clean the quote data too.

The quote data can be cleaned using the functions `noZeroQuotes`, `exchangeHoursOnly`, `autoSelectExchangeQuotes` (or `selectExchange`), `rmNegativeSpread`, `rmLargeSpread`, `mergeQuotesSameTimestamp`, `rmOutliersQuotes`. These functions remove entries with bid or offer prices of zero, delete entries outside the official opening hours of the exchanges, retain only data from one exchange chosen automatically or manually, delete entries with negative spread and entries with large spreads, merge quotes with identical timestamps, and remove outliers. When these functions are applied to a raw quote dataset, the user has a cleaned quote dataset.

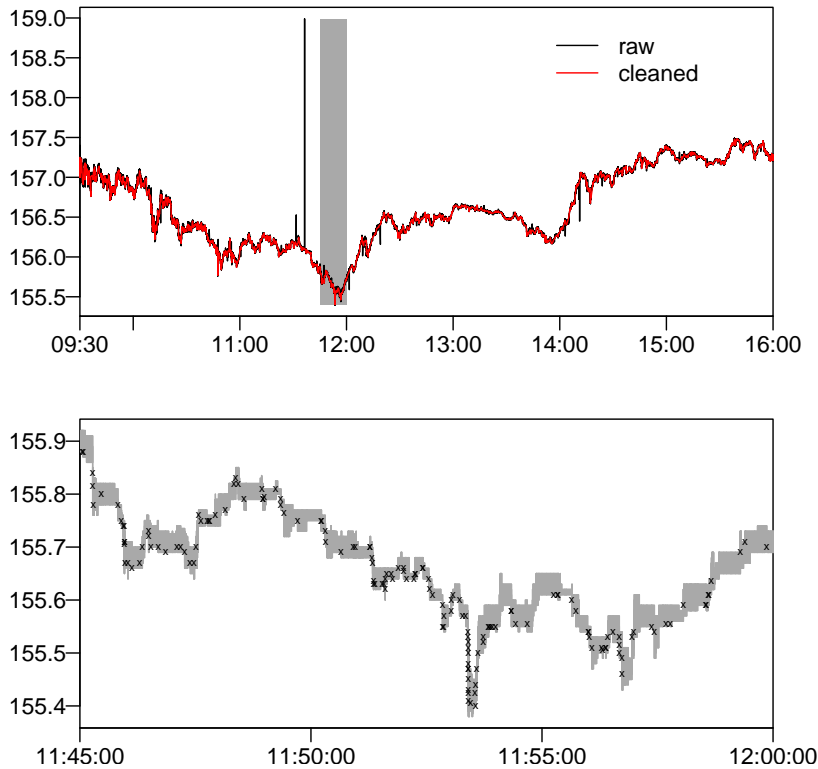


Figure 2: The upper panel displays the raw and cleaned transaction prices on the third of January of the `sampleTDataRaw` dataset. The lower panel shows 15 minutes of trading where the gray area is the bid and ask. The area shown in the lower subplot is shaded in gray.

Then, with the pre-cleaned trade dataset and the cleaned quote dataset, the final cleaning step is implemented in `rmTradeOutliersUsingQuotes`, which excludes trades that happens “too far” outside the prevailing bid and offer. Figure 2 shows the raw trades of the previously shown raw data along with the cleaned trade data in the upper subplot. In the lower subplot, the shaded area on the upper subplot is shown in a zoomed-in manner, showcasing 15 minutes of trading, with the gray shaded area denoting the bid-offer spread.

The function `matchTradeQuotes` is used to match trades and quotes. Unfortunately, this is not as straightforward as it may seem. In financial markets, trades and quotes may be subject to small reporting lags and, moreover, these reporting lags have changed over time. In the early days of tick-by-tick datasets, specialists and their clerks would report quotes electronically, while trades were reported at exchanges manually on a card through an optical reader. If the clerks were faster, the quotes would thus be registered faster than the trades, inducing a lag. In light of this lag, [Lee and Ready \(1991\)](#) and [Vergote \(2005\)](#) suggest lagging quotes by two seconds. Fortunately, in recent times the reporting lags have significantly decreased such that two seconds is not reasonable anymore. In fact, the reporting differences has almost completely disappeared, only causing real problems in times of peak market activity and extreme volatility.

Therefore, we implement the backwards-forwards matching (BFM) algorithm of [Christensen, Oomen, and Podolskij \(2014\)](#) for reintegration of otherwise erroneously deleted outliers in the trade datasets. The algorithm seeks to reintroduce trade outliers caused by inaccuracies in

timestamps and late reporting of block trades. This is achieved through attempting to match trades that occur outside the bid-offer interval first through a small window in the future, and then a larger window in the past. Any observation that does not lie between the bid and the offer either in the forward or the backward window is then deleted. For further information, see their appendix D. Note that when using the BFM algorithm it must be done in the data cleaning steps, not post-cleaning.

For convenience, we provide the functions `quotesCleanup`, and `tradesCleanupUsingQuotes` which together with `tradesCleanup` serve to automate the entire cleaning procedure.

The cleaning of the raw trade dataset included in the **highfrequency** package is done as follows to yield the `sampleTData` dataset.

```
R> sampleQData <- quotesCleanup(qDataRaw = sampleQDataRaw,
+   exchanges = "N", type = "standard", report = FALSE)
R> tradesAfterFirstCleaning <- tradesCleanup(
+   tDataRaw = sampleTDataRaw, exchanges = "N", report = FALSE)
R> sampleTData <- tradesCleanupUsingQuotes(tData = tradesAfterFirstCleaning,
+   qData = sampleQData,
+   lagQuotes = 0)[, c("DT", "EX", "SYMBOL", "PRICE", "SIZE")]
R> print(sampleTData, topn = 4)
```

	DT	EX	SYMBOL	PRICE	SIZE
1:	2018-01-02 09:30:00.125	N	XXX	158.50	50
2:	2018-01-02 09:30:00.145	N	XXX	158.50	1805
3:	2018-01-02 09:30:00.259	N	XXX	158.49	4
4:	2018-01-02 09:30:00.259	N	XXX	158.49	1

7165:	2018-01-03 15:59:59.299	N	XXX	157.28	400
7166:	2018-01-03 15:59:59.309	N	XXX	157.28	200
7167:	2018-01-03 15:59:59.329	N	XXX	157.28	200
7168:	2018-01-03 15:59:59.349	N	XXX	157.28	200

2.3. Aggregating high-frequency data

In high-frequency econometrics, data handling is a large part of conducting analyses. For example, many high-frequency-based estimators require data sampled at regular intervals. In the **highfrequency** package we provide different functions to handle cleaned data. The functions `aggregateTrades`, `aggregateQuotes`, `aggregatePrice`, and `aggregateTS` can be used to aggregate data based on time. The former two provide functionality specific to trades and quotes, whereas the latter two work on time series more generally. For example, we can aggregate trade data with the function `aggregateTrades`. This function takes a unit of time, seconds, minutes, or hours as argument `alignBy` and a (fractional) number of units as argument `alignPeriod`. In the **highfrequency** package, whenever data are aggregated based on time, we use the scheme of having units of time called `alignBy` aggregated by a number of units called `alignPeriod`.

```
R> agg <- aggregateTrades(sampleTData[, list(DT, PRICE, SIZE, SYMBOL)],
+   marketOpen = "09:30:00", marketClose = "16:00:00",
+   alignBy = "minutes", alignPeriod = 5)
```



```
R> print(agg, digits = 6, topn = 4)
      DT PRICE SIZE SYMBOL VWPRICE
1: 2018-01-02 09:30:00 158.50 50 XXX 158.500
2: 2018-01-02 09:35:00 158.85 25009 XXX 158.722
3: 2018-01-02 09:40:00 158.89 14472 XXX 158.999
4: 2018-01-02 09:45:00 158.47 10537 XXX 158.754
---
155: 2018-01-03 15:45:00 157.42 7040 XXX 157.388
156: 2018-01-03 15:50:00 157.24 8246 XXX 157.376
157: 2018-01-03 15:55:00 157.35 11405 XXX 157.362
158: 2018-01-03 16:00:00 157.28 56598 XXX 157.265
```

On the other hand, “calendar time” is not the only criterion on which to aggregate data, there are also so called business time measures. We include the `businessTimeAggregation` function which can do aggregation based on volume, volatility, and the intensity measure presented in [Oomen \(2006\)](#). The function samples a specified number of observations that are equally spaced based on the chosen measure.

One common problem faced by analysts is how to handle multiple tick-by-tick datasets at once when conducting multivariate analysis because securities do not trade on a fixed grid, trades and quotes do not arrive in a synchronous manner. This means with sufficient timestamp granularity and without any intervention, every observation in one instrument will be associated with missing observations in all other instruments. For example, assume that as in [Figure 3](#) an analyst is working on a dataset with three assets, stocks A, B, and C and that the timestamps are recorded with millisecond resolution. The first trade in the 15 seconds window for asset A and B occur after around 0.5 seconds but the first trade in C only occurs after around 1 second. Moreover, even though the first occurring trade execution time for asset A and B are close, they are not exactly the same. As a consequence, an aggregated dataset would list a trade in A at around 0.5 seconds and missing observations in B and C at that particular point in time. In light of this observation, some sort of synchronization must be done in order to conduct multivariate high-frequency analysis. This synchronization can be done by time using the `aggregate` family of functions, or by using functions like `merge.data.table`, and specifying the merge to be a rolling merge, which will roll missing observations forwards. However, rolling missing observations forward every time a new trade happens in any other instruments may be misleading because this may create a false impression of liquidity in an illiquid security. In the `highfrequency` package we implement the refresh time algorithm of [deB. Harris, McInish, Shoemith, and Wood \(1995\)](#) which synchronizes trades for an arbitrary number of securities by sampling only the latest trades from each security whenever newer trades have happened in all securities. [Figure 3](#) illustrates how the `refreshTime` algorithm works. The figure shows three assets. The vertical lines denote a trade arriving, and the numbers denote the number of the trade, i.e., the first trade is marked 1, and the second is marked 2 and so on. On the upper subplot, the right-arrows denote the timestamp where the given observations are moved to. So the first observations that arrive in assets A and B are moved to where the first observation in asset C arrives. The lower subplot shows the arrival of the trades post alignment using the `refreshTime` function. The refresh time algorithm is used to mitigate the Epps effect ([Epps 1979](#)), which is the effect of asynchronous trading driving down estimates of the covariance as a non-zero return in one asset is always paired with zero return(s) in other asset(s) when looking at timescales granular enough.

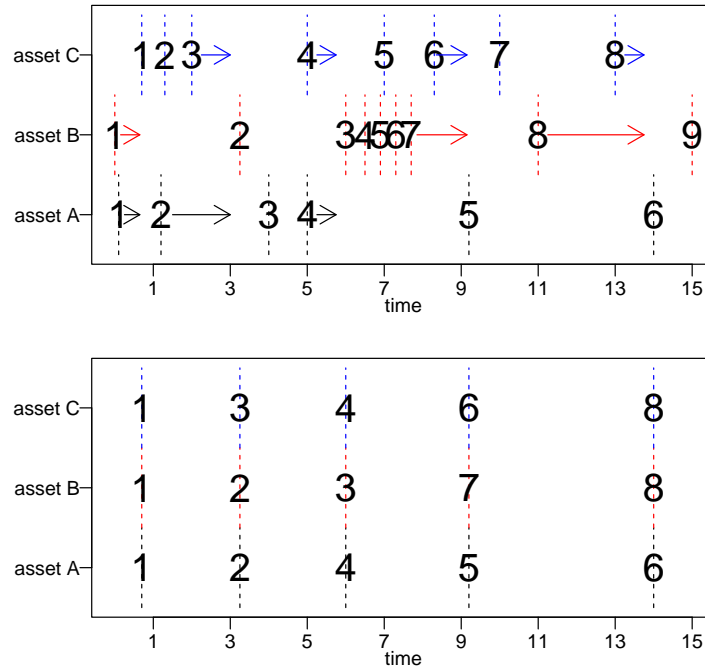


Figure 3: Visualization of the refresh time algorithm. The x -axes denote a window of time corresponding to 15 seconds. The vertical lines denote arrival of trades in the three different assets. The upper panel shows the inputs before the synchronization, the big numbers show the observation number of the observation. The rightwards pointing arrows show that the trade at the left is carried to the ending point to the right of the arrow. The lower panel shows the synchronized trades, which has all the observations aligned on the same timestamps. Note that a burst of activity happens in asset B which consists of many trades in rapid succession, but only one of these trades is retained after the synchronization.

3. Trade direction and liquidity measures

It is a stylized fact that financial markets show a diurnal pattern of activity where, on average, more trades occur in the morning around opening time and in the evening around closing time than around lunch. This amount of changing activity implies a changing amount of liquidity which is the ability to trade large quantities of stock or other assets quickly and with little impact on the price. The changing activity level also implies a change in the information flow to and from the market.

Many proxies of liquidity require knowing the direction of a trade. The direction of a trade is an indicator whether the buyer or the seller of the trade initiated the trade. Initiation happens through either a market order or a marketable limit order. Not all markets publish this information in their datasets and therefore this indicator will have to be inferred if it is not present. The function `getTradeDirection` implements the Lee-Ready algorithm of [Lee and Ready \(1991\)](#) which can be used to (imperfectly) infer the direction of trades. In most functions the quotes and trades of assets are passed as separate objects. However, in order to estimate the direction of a trade `getTradeDirection` function requires the trades and quotes to be passed as a single object. This should be done using the `matchTradeQuotes` function which is discussed in [Section 2.2](#).

Once the quotes and trades have been correctly merged, the issue of identifying which actor is the initiator of a trade is up next. In the `getTradeDirection` function we implement the Lee-Ready algorithm which is a simple three step procedure.

- If the transaction price is higher (lower) than the midquote, the transaction is marked as buyer (seller) initiated.
- If the transaction price is equal to the midquote, the transaction is marked as buyer (seller) initiated if the transaction price is higher (lower) than the previous transaction price.
- If the transaction price is equal to both the previous transaction price and the midquote, the transaction is marked buyer (seller) initiated if its transaction price is higher (lower) than the transaction price of the transaction prior to the previous transaction.

The liquidity of an asset plays an important role in many aspects of financial markets, such as analysis of jump events, news releases, and asset allocation. Therefore, it is not surprising that numerous measures of the liquidity of assets are available. We implement 23 of these measures in the function `getLiquidityMeasures`. Because 23 estimators are too many to go into detail with, a list of the estimators including the formulae is provided in Table ?? in Appendix A. [Boudt and Petitjean \(2014\)](#) examine the liquidity around news releases and jump events.

4. Theoretical price model

In the remainder of the paper, we will focus on tools for doing inference on the price process over the period $[0, T]$. The length of one day is normalized to unity.

For simplicity in notation, we consider the univariate case for now and assume to have one day with $n + 1$ observations observed on a time grid $t_0, t_1, t_2, \dots, t_n$. We denote the observed log-price by Y_{t_i} . The tick-by-tick high-frequency log-return is given by the difference in log-prices:

$$\Delta Y_{t_i} = Y_{t_i} - Y_{t_{i-1}}.$$

The observed price equals an efficient price process X_{t_i} plus an error term ε_{t_i} :

$$Y_{t_i} = X_{t_i} + \varepsilon_{t_i}. \quad (1)$$

In the literature, the error-term ε_{t_i} is referred to as market microstructure noise.

We also assume that the efficient, unobserved price process follows a continuous-time Itô semimartingale,

$$X_t = X_0 + \int_0^t \mu_s ds + \int_0^t \sigma_s dW_s + \sum_{i=1}^{N_t} J_i, \quad (2)$$

where X_t is the efficient log price of an asset, μ_s is the drift process, σ_s is the volatility process, W_s is a Wiener process, and J_i is the i th price jump in the interval $[0, t]$. Its arrival is governed by the count process N_t . Therefore the continuous-time price process in Equation 2 has two continuous parts; that, is a predictable drift process and a diffusive volatility process, and

two discrete parts; that is, the starting value (which is irrelevant when looking at log-returns) and the jump component.

We will explain the tools to detect price jumps, estimate the spot volatility, and also the total variability of the efficient price process over a day. The latter is the so-called quadratic variation (QV) of the price process, which can be decomposed in the integrated variance (IV) and the sum of squared intraday jumps:

$$QV = \underbrace{\int_0^T \sigma_s^2 ds}_{IV} + \underbrace{\sum_{i=1}^{N_T} J_i^2}_{\text{Jump variation}} .$$

5. Realized measures of variance and covariance

In the high-frequency econometrics literature, a great amount of attention has been paid to estimating the variance as well as covariances of high-frequency price processes. The main challenges faced when conducting univariate analyses are market microstructure noise, e.g., the restriction of prices to fixed multiples of cents, and discontinuous price processes caused by jumps, i.e., changes in prices that are “too large” for a continuous process. Both market microstructure noise and jumps cause upward bias in non-robust estimation methods. When conducting multivariate analyses, an analyst is additionally faced with the fact that instruments trade in an asynchronous manner. This means that using a method of alignment is necessary when estimating covariances. This then invariably results in loss of data as mentioned in Section 2.

5.1. Univariate realized measures

In this subsection we will use the sample dataset `sampleOneMinuteData`. This dataset contains 22 days of one-minute data for a pseudonymized exchange-traded-fund (ETF), called `MARKET` and a stock pseudonymized as `STOCK`.

```
R> data("sampleOneMinuteData", package = "highfrequency")
```

In Figure 4 we plot the last four trading days worth of data, which will be the subset of data we will use in this illustration. In the code used throughout this section, we call this subsample `oneMinute`. Each of the four days are shown separately, the red dashed line is the stock and the black solid line is the ETF.

Visually and while focusing only on the stock, it seems that in the upper two and the lower right subplots most of the variation in the price stems from pronounced jumps in the price. The lower left subplot suggests that the ETF had a much lower proportion of the variation that stems from jumps. The upper right subplot shows that both the ETF and stock has a jump in the upwards direction just around 14:00. To highlight the properties of the estimators and the impact of jumps and market microstructure noise on the estimators we use the same subsample for multiple estimators.

The RV estimator is simply defined as

$$RV = \sum_{i=1}^n \Delta Y_{t_i}^2.$$

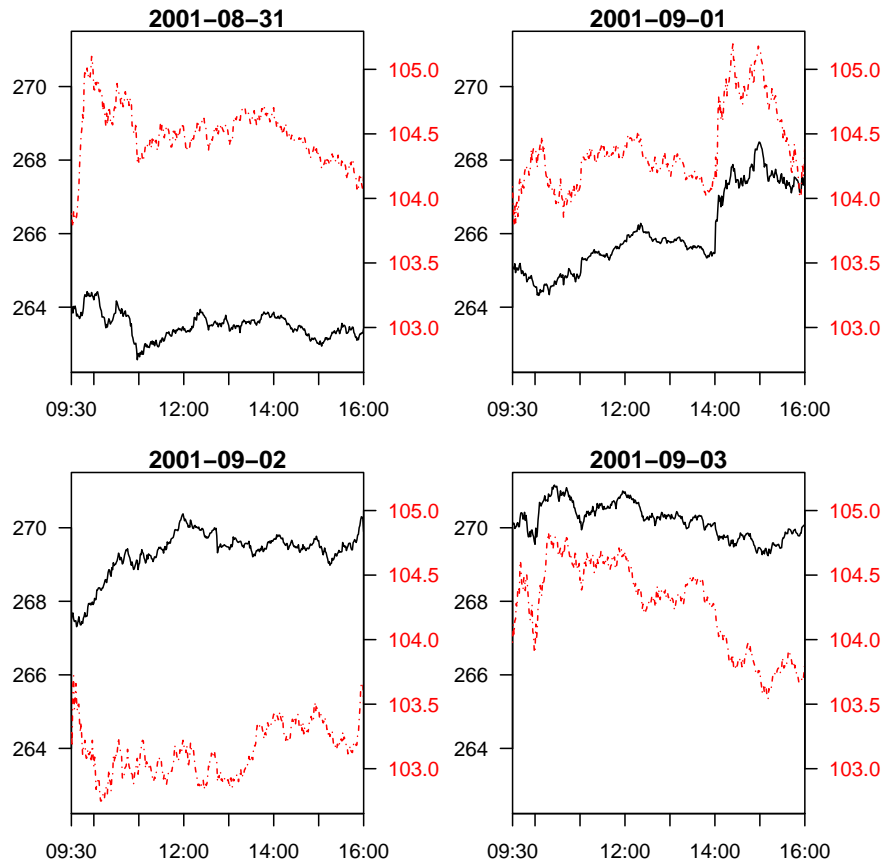


Figure 4: The last four trading days of one-minute data in the `sampleOneMinuteData` dataset included in the `highfrequency` package. In black with a solid line we depict the pseudonymized market return, and the red dashed line depicts the pseudonymized stock.

The realized variance is usually used with equidistant sampling. With a one-minute sampling grid, we have that $n = 390$ for a usual 6.5 hour trading day on the NYSE. The function `rCov` can be used to estimate the realized variance. The `alignBy` and `alignPeriod` arguments are used to control the aggregation of prices to arbitrary minute, second, or hour intervals. Naturally, in this case these options are limited as we are already using data aggregated on a one-minute grid. The `makeReturns` argument is used to denote that we are passing prices as the first argument. An illustration of RV estimation with one-minute returns is shown below.

```
R> oneMinute <- sampleOneMinuteData[as.Date(DT) > "2001-08-30"]
R> RV1 <- rCov(oneMinute[, list(DT, MARKET)], makeReturns = TRUE,
+   alignBy = "minutes", alignPeriod = 1)
R> print(RV1)
```

	DT	RV
1:	2001-08-31	0.0000322
2:	2001-09-01	0.0000605
3:	2001-09-02	0.0000368
4:	2001-09-03	0.0000397

Supplying the arguments `alignBy` and `alignPeriod` is only done for illustrative purposes as this data is already sampled at a one-minute frequency. The realized variance estimator measures the total variance of the price process, not just the integrated variance. Therefore, if the object of interest is the integrated variance, the realized variance is only consistent when we observe the efficient price process free of jumps.

Due to the inconsistency of the RV estimator as an estimator of the integrated variance in the presence of jumps, the need for a jump robust estimator is clear. One of the jump robust estimators implemented in the **highfrequency** package is the bipower variation (BPV) estimator, proposed in [Barndorff-Nielsen and Shephard \(2004b\)](#). This estimator is defined as

$$BPV = \frac{\pi}{2} \sum_{i=1}^{n-1} |\Delta Y_{t_i}| \cdot |\Delta Y_{t_{i+1}}|.$$

Here, the effect of a jump in a single return observation gets smoothed out by multiplying with the subsequent return if the latter is not affected by a jump itself. This change makes the BPV estimator consistent in the presence of jumps as the sampling window shrinks and the number of observations in each window increases. The bipower variation converges to the integrated volatility even in the presence of jumps. Since we have a jump-robust and a non-robust estimator, we can estimate the size of the jump variation (JV) by taking $JV = RV - BPV$. Due to estimation errors in BPV , in practice JV is not guaranteed to be positive. This is typically dealt with by setting negative estimates to 0. In order to apply the BPV estimator, we can use the `rBPCov` function in the same way as we used `rCov` in the previous example.

```
R> BPV1 <- rBPCov(oneMinute[, list(DT, MARKET)], makeReturns = TRUE)
R> print(BPV1)
```

```
      DT      BPV
1: 2001-08-31 0.0000312
2: 2001-09-01 0.0000474
3: 2001-09-02 0.0000332
4: 2001-09-03 0.0000399
```

Choosing the correct period for aligning returns, be it in calendar or business time, is an important factor of realized variance estimation. Empirically, the estimated volatility is typically higher when a higher frequency sampling is used. This is because the effect of market microstructure noise. One cause of this microstructure noise are discrete price intervals for trade prices. In the following, we will aggregate the realized covariance across different aggregation periods both in calendar time and business time.

```
R> data("sampleMultiTradeData", package = "highfrequency")
R> nums <- c(1, 2, 5, 10, 20, 30, 60, 90, 180, 360, 600)
R> rvAgg <- matrix(nrow = length(nums), ncol = 2)
R> for(i in 1:nrow(rvAgg)) {
+   rvAgg[i, 1] <-
+     rCov(sampleMultiTradeData[SYMBOL == 'AAA', list(DT, PRICE)],
+         alignBy = "ticks", alignPeriod = nums[i],
+         makeReturns = TRUE)[[2]]
```

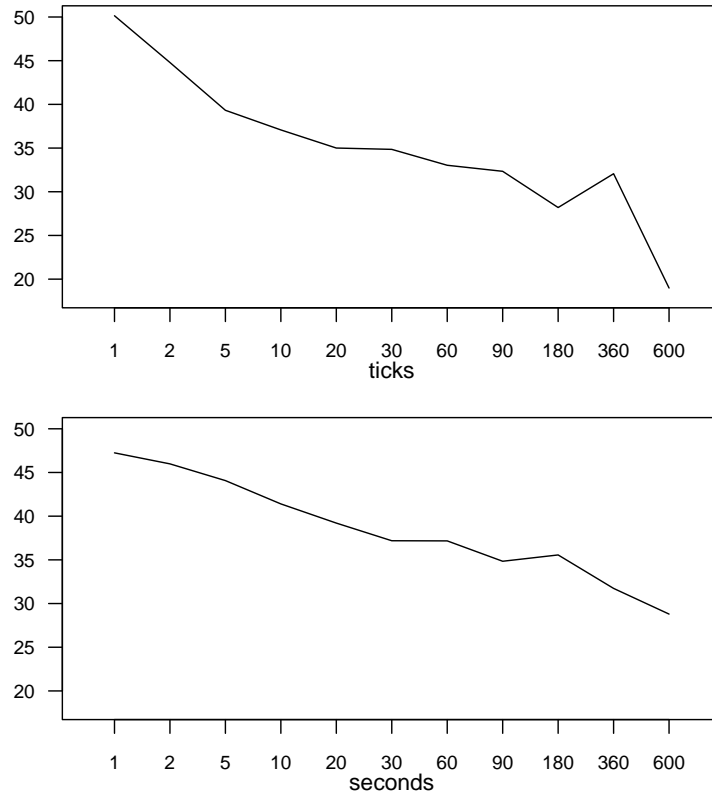


Figure 5: The volatility signature plot for the AAA stock in the sample dataset `sampleMultiTradeData`, the upper plot shows the percentage annualized realized volatility, with the price aligned by an increasing number of ticks. In the lower subplot the price is aligned by seconds.

```
+   rvAgg[i, 2] <-
+     rCov(sampleMultiTradeData[SYMBOL == 'AAA', list(DT, PRICE)],
+         alignBy = "seconds", alignPeriod = nums[i],
+         makeReturns = TRUE)[[2]]
+ }
```

Figure 5 shows the corresponding volatility signature plot for the stock named AAA in the `sampleMultiTradeData` dataset. The upper subplot depicts the estimated realized variance when the price is sampled at increasing tick intervals, while the lower subplot shows the estimated realized variance when the price is sampled at increasing second intervals instead. We observe a downward-sloping trend in both subplots which is evidence for microstructure noise present in the data.

Barndorff-Nielsen *et al.* (2009) propose the realized kernels estimator, which use kernel methods to combat market microstructure noise. The realized kernel estimator takes form of

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

$$\gamma_h = \sum_{i=|h|+1}^n \Delta Y_{t_i} \Delta Y_{t_{i-|h|}},$$

where $k(\cdot)$ is a kernel function. The kernel function should satisfy the smoothness condition, $k'(0) = k'(1) = 0$ and H is a kernel tuning parameter.

`listAvailableKernels()` lists the kernels that are implemented. The kernels correspond to those of Barndorff-Nielsen *et al.* (2008), specifically Table 1 and Table 2 in the aforementioned paper, along with the Modified Tukey-Hanning kernel.

```
R> listAvailableKernels()
```

```
[1] "Rectangular"      "Bartlett"
[3] "Second"           "Epanechnikov"
[5] "Cubic"            "Fifth"
[7] "Sixth"            "Seventh"
[9] "Eighth"           "Parzen"
[11] "TukeyHanning"     "ModifiedTukeyHanning"
```

For calculating the ex-post variation by means of kernel estimation at one-minute intervals, we use the function `rKernelCov`. To select the Parzen kernel we set `kernelType = "Parzen"`.

```
R> RK1 <- rKernelCov(oneMinute[, list(DT, MARKET)], kernelType = "Parzen",
+   makeReturns = TRUE)
R> print(RK1)
```

```
          DT          RK
1: 2001-08-31 0.0000294
2: 2001-09-01 0.0000584
3: 2001-09-02 0.0000379
4: 2001-09-03 0.0000360
```

An alternative approach to dealing with market microstructure noise is to use the pre-averaging technique. We illustrate this estimator in the multivariate setting in the next subsection.

5.2. Multivariate realized measures

Variance estimates are not the only component relevant to risk management, but measures of dependency are also very important inputs in portfolio construction and asset allocation decisions. There are numerous estimators of dependency based on high-frequency data and we implement many of these estimators. In this section, we will cover the solutions for problems that arise when going from a univariate setting to a multivariate setting.

Naturally, we need a multivariate price process, and we use a generalization of the univariate model presented in Equation 1 and Equation 2:

$$\begin{aligned} Y_t &= X_t + \varepsilon_t, \\ X_t &= X_0 + \int_0^t \mu_s ds + \int_0^t \Omega_s dW_s + \sum_{i=1}^{N_t} J_i. \end{aligned}$$

The multivariate analogue of the integrated variance is the integrated covariance matrix

$$ICov = \int_0^T \Omega_s \Omega_s^\top ds$$

We carry over the notation for the observed prices but instead they are now vectors such that, with d assets, $Y_t = (y_{1,t}, y_{2,t}, \dots, y_{d,t})$, and the returns are now a vector of returns $\Delta Y_{t_i} = Y_{t_i} - Y_{t_{i-1}} = (y_{1,t} - y_{1,t-1}, y_{2,t} - y_{2,t-1}, \dots, y_{d,t} - y_{d,t-1})$.

The realized covariance estimator of [Barndorff-Nielsen and Shephard \(2004a\)](#) is just a straightforward multivariate extension of the RV estimator, namely the sum of the outer product of high-frequency returns:

$$RC = \sum_{i=1}^n \Delta Y_{t_i} \Delta Y_{t_i}^\top.$$

The realized covariance estimator yields the realized variance estimates on the diagonal and the covariance(s) between assets on the off-diagonals. Therefore, we can use the same function that we used for the realized variance estimation, `rCov`. This time, the function will return a `list` object with one $d \times d$ matrix per day. The `cor` argument can be used to transform the covariance matrices into correlations instead, where the diagonals will be equal to 1 and the off-diagonals are correlation coefficients.

```
R> rCov(oneMinute, makeReturns = TRUE) [1:2]
```

```
$`2001-08-31`
```

	STOCK	MARKET
STOCK	0.0000792	0.0000340
MARKET	0.0000340	0.0000322

```
$`2001-09-01`
```

	STOCK	MARKET
STOCK	0.0001313	0.0000649
MARKET	0.0000649	0.0000605

When moving to tick-by-tick data, the issue of synchronization arises. Due to the earlier mentioned Epps effect, the covariance estimate is biased towards 0. With the realized covariance estimator, in the presence of microstructure noise we have that the diagonals are biased upwards, and the off-diagonals are biased downwards as the sampling frequency increases. These two biases can lead to severely distorted empirical correlation coefficients.

One method of dealing with the Epps-effect is to use the modulated realized covariance, which makes use of pre-averaging techniques. Let $\overline{\Delta Y}_{t_i} = \sum_{h=1}^{k_n-1} g\left(\frac{h}{k_n}\right) \Delta Y_{t_i+h}$ be the pre-averaged returns, where $g(x) = \min(x, 1-x)$, $k_n = \lfloor \theta \sqrt{N} \rfloor$ and θ is a tuning parameter controlling the pre-averaging horizon with N being the number of observations after applying the refresh time algorithm. The pre-averaging reduces the effect of market microstructure noise, but requires a bias correction term.

The modulated realized covariance is defined as

$$MRC = \frac{N}{N - k_n - 2} \frac{1}{\phi_2 k_n} \sum_{i=0}^{N-k_n+1} \overline{\Delta Y}_{t_i} \overline{\Delta Y}_{t_i}^\top - \frac{\phi_1^{k_n}}{\theta^2 \phi_2^{k_n}} \hat{\Psi},$$

where we have that $\hat{\Psi} = \frac{1}{2N} \sum_{i=1}^N \Delta Y_{t_i} \Delta Y_{t_i}^\top$, $\phi_1^{k_n} = k_n \sum_{j=1}^{k_n} \left(g\left(\frac{j+1}{k_n}\right) - g\left(\frac{j}{k_n}\right) \right)^2$, and $\phi_2^{k_n} = \frac{1}{k_n} \sum_{j=1}^{k_n} g^2\left(\frac{j}{k_n}\right)$. The modulated realized covariance can be estimated using the function

`rMRCov`, the data must be prices in levels, and the argument `theta` controls the pre-averaging horizon.

```
R> rMRCov(
+   list("ETF" = sampleMultiTradeData[SYMBOL == "ETF", list(DT, PRICE)],
+       "AAA" = sampleMultiTradeData[SYMBOL == "AAA", list(DT, PRICE)]),
+   theta = 0.1)
```

```
      ETF      AAA
ETF 0.000301 0.000309
AAA 0.000309 0.000589
```

```
R> rCov(spreadPrices(sampleMultiTradeData[SYMBOL %chin% c("ETF", "AAA")]),
+   alignBy = "ticks", alignPeriod = 1, makeReturns = TRUE)
```

```
      ETF      AAA
ETF 0.000283 0.000000
AAA 0.000000 0.000998
```

As can be seen, the modulated realized covariance overcomes the asynchronicity problem and produces realistic estimates, which imply a relatively high correlation with the market. On the other hand, the realized covariance on the tick-by-tick data estimates the off-diagonal elements to be zero due to the Epps effect.

Table 2 provides a comprehensive list of the realized measures implemented in the **high-frequency** package. Most functions work with multivariate data over multiple days, but some functions only work with data over a single day, these functions are `rMRCov`, `rCholCov`, `rBACov`, `rOWCov`, `rTSCov`, and `rRTSCov`. Most, but not all realized measure functions have the arguments `alignBy`, `alignPeriod`, and `makeReturns`, the two former arguments can be used to control the alignment of prices or returns in case the input does not match the wanted granularity. The latter argument is used to designate whether the input data are returns or prices. For example, if we want to calculate five-minute semi-covariances, but the data we have is comprised of one-minute returns, we would set `alignBy = "minutes"`, `alignPeriod = 5`, and `makeReturns = FALSE`. As a convenience to developers and analysts, we provide additional documentation pages for `help("ICov")` and `help("IVar")` which serve as alternatives to Table 2 in Appendix B to look for estimators of the integrated covariance and the integrated variance respectively.

In addition to estimators of the integrated variance and covariance, we provide a number of estimators of higher moments of the return distributions; for example, kurtosis, skewness and quarticity. The latter has a nice interpretation and usage because it is a measure of the volatility of the volatility and it can be used to estimate the measurement error of realized volatility estimates.

5.3. Noise variance estimation

As mentioned earlier, market microstructure noise is an inescapable problem when dealing with high-frequency data. Market microstructure noise can have rather rich dynamics including serial correlation. Moreover previous research suggests that these dynamics may change over time, see [Hansen and Lunde \(2006\)](#).

The realized moments of disjoint increments (REMEDI) estimator of [Li and Linton \(2021\)](#) can be used to estimate the dynamics of market microstructure. More specifically, it allows for estimation of the l^{th} -order auto-covariance of the market microstructure noise,

$$\hat{R}_l = \frac{1}{n} \sum_{i=2k_n}^{n-k_n-l} \left(Y_{t_{i+l}} - Y_{t_{i+l+k_n}} \right) \left(Y_{t_i} - Y_{t_{i-2k_n}} \right),$$

where n is the number of observations and k_n is a tuning parameter.

The REMEDI estimator is implemented in the `ReMeDI` function. We also implement the functions `knChooseReMeDI` and `ReMeDIAsymptoticVariance` to estimate the optimal k_n parameter and the asymptotic variance of the REMEDI estimation, respectively.

To estimate the optimal k_n tuning parameter, the microstructure noise, and the asymptotic (co)-variance of the microstructure noise, we use three functions, `ReMeDI`, `knChooseReMeDI`, and `ReMeDIAsymptoticVariance`.

```
R> stockData <- sampleMultiTradeData[
+   SYMBOL == "AAA", list(DT, PRICE = log(PRICE))]
R> kn <- knChooseReMeDI(stockData)
R> remedi <- ReMeDI(stockData, kn = kn, lags = 0:15)
R> remedi

[1] 4.85e-08 1.84e-08 1.08e-08 1.06e-08 1.15e-08 1.56e-08
[7] 1.32e-08 1.11e-08 5.20e-09 -1.84e-09 -8.06e-09 -1.04e-08
[13] -7.51e-09 -6.08e-09 -2.80e-09 -1.91e-09

R> asympVar <- ReMeDIAsymptoticVariance(
+   stockData, kn = kn, lags = 0:15, phi = 0.2, i = 2)
R> asympVar

$ReMeDI
[1] 4.85e-08 1.84e-08 1.08e-08 1.06e-08 1.15e-08 1.56e-08
[7] 1.32e-08 1.11e-08 5.20e-09 -1.84e-09 -8.06e-09 -1.04e-08
[13] -7.51e-09 -6.08e-09 -2.80e-09 -1.91e-09

$asympVar
[1] 1.84e-12 1.06e-12 6.93e-13 4.17e-13 2.02e-13 1.76e-14 1.12e-14
[8] 2.54e-14 1.34e-13 1.56e-13 9.22e-14 1.74e-14 2.54e-14 5.57e-15
[15] 1.52e-14 2.69e-14

$lags
[1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

attr(,"class")
[1] "asympVarReMeDI"
```

We refer to [Li and Linton \(2021\)](#) for further details.

6. Spot estimation and intraday periodicity

While realized volatility estimates the integrated variance, there are also estimators that seek to estimate the spot volatility, σ_t . The spot volatility is sometimes also referred to as the instantaneous volatility. In the **highfrequency** package we implement multiple spot volatility estimators in the `spotVol` function.

As noted in Section 3, the activity on the financial markets measured in both volume traded and number of transactions show a diurnal pattern. This pattern shows up in the instantaneous volatility too. The increased volatility in the morning is likely due to increased amounts of price discovery happening. The news that ticked in since markets closed a day earlier are incorporated in the price of an asset during market opening times. Before markets close, the increased volatility comes from, among others, investors who seek to reduce their exposure overnight. Due to the distinct diurnal pattern, many estimators incorporate this effect; for example, by letting the spot volatility be determined by a constant daily factor, which may be estimated by a daily realized measure, and a seasonality factor that incorporates the diurnal pattern into the spot estimates. One such estimator of the spot volatility is the deterministic periodicity class of estimators defined as

$$\hat{\sigma}_t = \hat{d}_t \cdot \hat{\sigma}_C$$

where $\hat{\sigma}_C$ is a day-to-day constant level of volatility, which may change between days, and \hat{d}_t is the periodicity factor which is the same across days but varies during the day; for

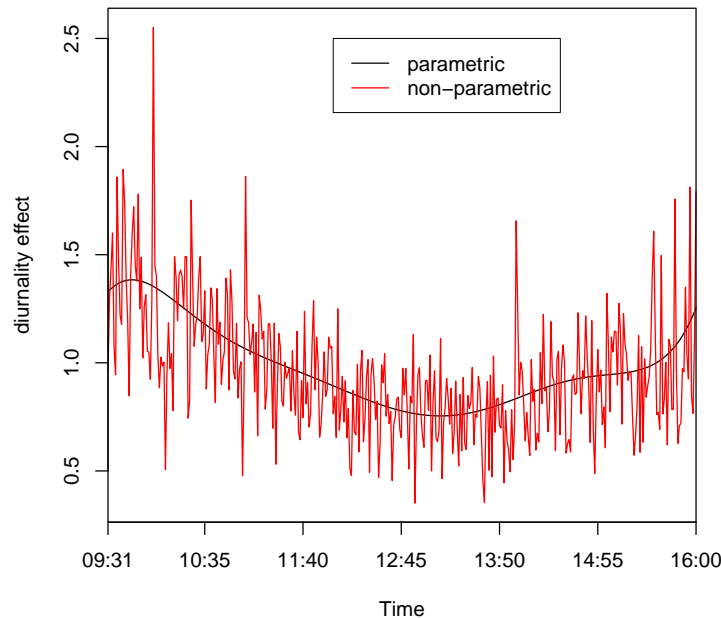


Figure 6: Spot volatility estimation the smooth line shows the intraday periodicity estimate from the parametric spot volatility estimator. The jagged line shows the intraday periodicity estimate from the non-parametric estimator. Note that the jagged-ness is exacerbated by the relatively low number of observations in the sample dataset but that the estimate will likely be smoother with a larger dataset.

example, \hat{d}_t is the same on day one and day two at 09:45:00, but its value is different from the value at 09:50:00. Using the function `spotVol`, we estimate the spot volatility of the MARKET ETF in the `sampleOneMinuteData` dataset. We apply the deterministic periodicity estimator which is a class of spot volatility estimators where we supply four different implementations. We use the "TML" and the "WSD" settings for the Truncated Maximum Likelihood and the Weighted Standard Deviation estimation method of [Boudt *et al.* \(2011\)](#). These estimators are parametric and non-parametric, respectively. For definitions and introductions to these estimators, see [Boudt *et al.* \(2011\)](#).

```
R> parametric <- spotVol(
+   data = sampleOneMinuteData[, list(DT, PRICE = MARKET)],
+   periodicVol = "TML", P1 = 2, P2 = 2, alignPeriod = 1)
R> nonParametric <- spotVol(
+   data = sampleOneMinuteData[, list(DT, PRICE = MARKET)],
+   periodicVol = "WSD", alignPeriod = 1)
```

We plot the resulting estimates of the recurring periodicity in [Figure 6](#). Unsurprisingly, the parametric estimator is much smoother than the non-parametric estimator which shows considerable variability in the periodicity estimated. We can see that the volatility is higher in the beginning and in the end of the trading day than in the middle of the trading day.

7. Testing for the presence of jumps

In general there are two approaches to jump testing in the literature. First, there are daily jump tests which are tests with the null hypothesis that no jump occurred during a day. Second, there are intraday jump tests which are multiple tests for the presence of jumps in small blocks of returns. We call these tests LM-type tests because the tests we implement are based on the procedure proposed in [Lee and Mykland \(2008\)](#).

Since we have the RV estimator which converges to the QV in the presence of jumps, and the BPV estimator which converges to the IV, we can estimate the jump variation by simply taking $JV = RV - BPV$.

If RV and BPV coincide with each other, we know that we have no jumps, and we can test this null hypothesis. The test of [Barndorff-Nielsen and Shephard \(2006\)](#) (henceforth BNS) does this by estimating the QV and IV, along with the integrated quarticity (IQ), which is defined as $IQ = \int_0^T \sigma_s^4 ds$. The IQ is estimated under the alternative that the jump variation is not equal to zero in order to increase the test's power. The BNS test comes in two forms, one version based on a difference and one version based on a ratio. The difference version is given by

$$\hat{G} = \frac{\hat{IV} - RV}{\sqrt{(\theta - 2) \frac{1}{n} \hat{IQ}}} \xrightarrow{L} \mathcal{N}(0, 1),$$

where θ denotes the coefficient multiplying the variance of \hat{IV} . We can reject the null hypothesis of a continuous price process only if \hat{G} is significantly negative, as a significantly positive value would imply a negative contribution to the variance from jumps. However, such observations stem only from small-sample biases present in many jump robust estimators.

The Lee-Mykland (LM) test takes a form similar to that of a classical t -test. It is given by:

$$\mathcal{L}_{t_i} = \frac{(Y_{t_i} - Y_{t_{i-1}}) - \hat{\mu}_{t_i}}{\hat{\sigma}_{t_i}},$$

where $\hat{\mu}_{t_i}$ and $\hat{\sigma}_{t_i}$ are estimates of the instantaneous mean and volatility of the continuous part of $Y_{t_i} - Y_{t_{i-1}}$. The null hypothesis of no jump of the LM test is rejected if $|\mathcal{L}| > C_n + \beta^* \cdot S_n$ where $C_n = \frac{\sqrt{2 \log n}}{c} - \frac{\log \pi + \log \log n}{\sqrt{2c(2 \log n)}}$ and $S_n = \frac{1}{\sqrt{c(2 \log n)}}$, with $c = \frac{\sqrt{2}}{\sqrt{\pi}}$ and n is the number of observations in the entire sample. Lastly, $\beta^* = -\log(-\log(\alpha))$, where α is the significance level of the test.

The daily jump tests in the **highfrequency** package cannot be used to detect the number of jumps in the price path of one period. As discussed in Lee and Mykland (2008), this is due to the usage of integrated measures in this type of jump test. In contrast, the LM test is able to determine the number of jumps in a price path, and even detect which return contains a jump, not just the presence of non-zero jump variation in the entire day.

In the following, we will use the function `BNSjumpTest` function to apply the BNS jump test on the market ETF in `sampleOneMinuteData`. Thereafter we will apply the LM test by using the `intradayJumpTest` function. For the latter, we focus our attention on the day with the lowest p -value in the BNS jump test and ascertain the number of jumps on this date.

```
R> daily <- BNSjumpTest(
+   sampleOneMinuteData[, list(DT, MARKET)],
+   makeReturns = TRUE)
```

The BNS jump test provides a critical value for each of the 22 days, so we extract the p -values in order to find a suitable date to conduct our LM test on.

```
R> pValues <- sapply(daily, function(x) x[["pvalue"]])
R> pValues[which.min(pValues)]
```

```
2001-08-26
 3.38e-08
```

We can see that the largest test statistic and, hence, the lowest p -value realized on 26 August 2001. Therefore, we choose this day for plotting our intraday LM test procedure.

```
R> intraday <- intradayJumpTest(
+   sampleOneMinuteData[as.Date(sampleOneMinuteData$DT) == "2001-08-26",
+   list(DT, PRICE = MARKET)],
+   makeReturns = TRUE, alignBy = "minutes", alignPeriod = 1)
```

In Figure 7 we show the results of the daily BNS jump test as well as the LM test on 26 August 2001. In the upper subplot, we depict the test statistics per day. In the lower subplot, we plot the intraday price process and indicate timestamps at which the LM test rejects the null hypothesis of no instantaneous jumps at the 5% significance level with a red vertical line. The daily test has a p -value that leads to a rejection of the null hypothesis of no jumps on 26 August 2001. The intraday jump test follows suit and detects 13 jumps throughout the day.

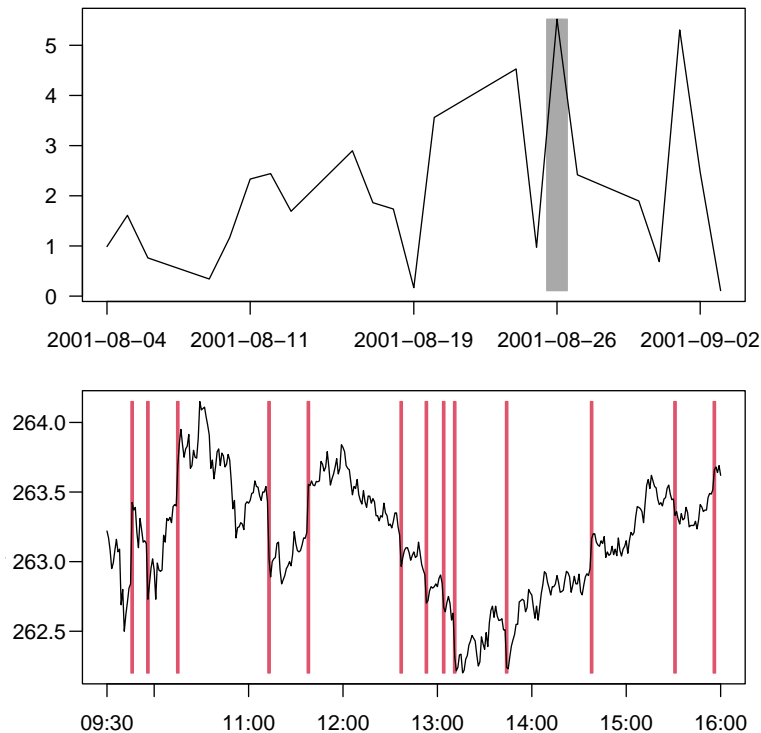


Figure 7: The top subplot shows the absolute value of the daily BNS jump test statistics. The gray shaded area highlights the day that contains the maximum value, which is also the day on which the LM test (bottom subplot) is applied. The bottom subplot shows the one-minute intraday prices on 26 August 2001. The areas shaded in red denote the times where the LM test exceeds the 95% critical value, which implies that a jump is detected.

8. Modeling and forecasting realized variances

So far, we have focused on data cleaning, deriving realized measures of intraday variation, testing for the presence of jumps in prices, and estimating the spot volatility. All these analyses have one thing in common; they're all ex-post based, meaning that they describe what has already happened. Now, we will turn to an ex-ante topic: forecasting realized measures.

The realized measures are employed both in forming expectations about the future and in forecast evaluation. In the **highfrequency** package, we implement two models for forecasting volatility, i.e., the HEAVY model of Shephard and Sheppard (2010) and the HAR model (and extensions hereof) introduced by Corsi (2009).

These models are typically estimated using time series aggregated on a daily frequency, e.g., by using `rCov`. For this reason, the `SPYRM` dataset included in the **highfrequency** package contains realized measures of the SPDR S&P500 ETF Trust with the ticker `SPY` which tracks the S&P500 index. The dataset contains data from 2 January 2014 to 31 December 2019 and consists of six realized measures each computed at a one- and five-minute frequency. In addition, the closing prices of the ETF is also included.

8.1. HAR model

The HAR model of Corsi (2009) is a simple model used for forecasting realized measures. The model is motivated heuristically by the presence of heterogeneous traders, who trade on a daily, weekly, and monthly basis. The HAR model comes in several different extensions; for example, correcting for jumps and measurement error of the realized variance. Let τ be the daily index. In the following, RV_τ will be used to refer to estimators that converge to the quadratic variation for a given day and BPV_τ will be used to denote jump robust estimators of the integrated variance. Any estimator of the quadratic variation can be plugged in instead of RV_τ and likewise for the integrated variance and quarticity instead of BPV_τ and RQ_τ .

The original HAR model is specified as

$$RV_\tau = \beta_0 + \beta_1 RV_{\tau-1} + \beta_2 RV_{\tau-1}^w + \beta_3 RV_{\tau-1}^m + \varepsilon_\tau,$$

where $RV_{\tau-1}^w = \frac{RV_{\tau-1} + RV_{\tau-2} + \dots + RV_{\tau-5}}{5}$, $RV_{\tau-1}^m = \frac{RV_{\tau-1} + RV_{\tau-2} + \dots + RV_{\tau-22}}{22}$, and ε_τ is the error term.

In the **highfrequency** package the HAR model is implemented such that both realized measures and high-frequency returns can be supplied. The `HARmodel` function accepts both prices and pre-computed realized measures. The `inputType` argument is used to toggle between pre-computed realized measures (`inputType = "RM"`) and high-frequency returns when `inputType` is different from "RM".

The usage for basic HAR estimation based on realized measures is simply:

```
R> RV <- as.xts(SPYRM[, list(DT, RV5)]) * 10000
R> model <- HARmodel(data = RV, periods = c(1, 5, 22),
+                   type = "HAR", inputType = "RM")
R> summary(model)
```

Call:

```
"RV1 = beta0 + beta1 * RV1 + beta2 * RV5 + beta3 * RV22"
```

Residuals:

```
   Min      1Q  Median      3Q      Max
-4.798 -0.152 -0.098  0.020 23.148
```

Coefficients:

```
      Estimate Std. Error t value Pr(>|t|)
beta0  0.1160     0.0429   2.71 0.00689 **
beta1  0.2953     0.0981   3.01 0.00265 **
beta2  0.2813     0.0773   3.64 0.00028 ***
beta3  0.1472     0.0622   2.37 0.01808 *
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.747 on 1469 degrees of freedom

Multiple R-squared: 0.25, Adjusted R-squared: 0.248

F-statistic: 163 on 3 and 1469 DF, p-value: <2e-16

The `HARmodel` function returns an object of type `HARmodel`, which has the following methods: `summary`, `plot`, `print`, and `predict`.

Many extensions of the HAR model have been proposed. The model types available in the **highfrequency** package are `HAR`, `HARJ`, `HARCJ`, `HARQ`, `HARQJ`, `CHAR`, `CHARQ`. Selecting between these models is done by using the `type` argument.

The “J” models denote a model with a jump component. The jump component is, as described in Section 5.1, estimated as a an estimate of the quadratic variation minus an estimate of the integrated variance. For example, the realized variance and the bipower variation can be used for calculating $J_\tau = RV_\tau - BPV_\tau$. In the HARJ model, the jump component is added as a regressor. Thus, the model reads:

$$RV_\tau = \beta_0 + \beta_1 RV_{\tau-1} + \beta_2 RV_{\tau-1}^w + \beta_3 RV_{\tau-1}^m + \beta_4 J_{\tau-1} + \varepsilon_\tau.$$

Andersen, Bollerslev, and Diebold (2007) compares the above HAR model that includes only a jump component (HARJ) with their proposed HAR model including a continuous and a jump component (HARCJ). The HARCJ model reads:

$$RV_\tau = \beta_0 + \beta_1 C_{\tau-1} + \beta_2 C_{\tau-1}^w + \beta_3 C_{\tau-1}^m + \beta_4 J_{\tau-1} + \varepsilon_\tau,$$

where C_τ is defined as $I[Z_\tau \leq \Phi_\alpha] \cdot RV_\tau + I[Z_\tau > \Phi_\alpha] \cdot BPV_\tau$, where $I[\cdot]$ is the indicator function, Z_τ is the test statistic of a jump test, and Φ is the critical value of that jump test. Simply put, the HARCJ replaces the non-robust estimator with a jump-robust estimator when a jump test shows that a jump has occurred. To control the jump test applied in the HARCJ type model, the argument `jumpTest` can be used. The available options are the names of the functions mentioned in Section 7.

Andersen *et al.* (2007) discussed including the continuous and jump components as separate explanatory variables in the HAR model. For example, a continuous HAR (CHAR) uses only the continuous part of the quadratic variation to forecast the quadratic variation.

$$RV_\tau = \beta_0 + \beta_1 BPV_{\tau-1} + \beta_2 BPV_{\tau-1}^w + \beta_3 BPV_{\tau-1}^m + \varepsilon_\tau,$$

Bollerslev, Patton, and Quaadvlieg (2016) introduce the HAR model with quarticity (HARQ) model, which incorporates an interaction term of realized volatility and realized quarticity. The reason for this interaction term is that the realized quarticity is a measure for the estimation error in the realized variance proxy on a particular day. The typically negative interaction term downweights the predictive weight on RV observations that have a large measurement error,

$$RV_\tau = \beta_0 + \left(\beta_1 + \beta_{1Q} RQ_\tau^{1/2} \right) RV_{\tau-1} + \beta_2 RV_{\tau-1}^w + \beta_3 RV_{\tau-1}^m + \varepsilon_\tau,$$

where RQ_τ is an estimate of the integrated quarticity.

In the **highfrequency** package, we also provide two extensions of the models above. The HARQJ model is a HARQ model augmented by a jump component and the CHARQ model is a CHAR model augmented by a RQ interaction term.

In addition, we provide non-linear transformations as is done in Corsi (2009), the standard deviation form and the logarithmic form:

$$\sqrt{RV_\tau} = \beta_0 + \beta_1 \sqrt{RV_{\tau-1}} + \beta_2 \sqrt{RV_{\tau-1}^w} + \beta_3 \sqrt{RV_{\tau-1}^m} + \beta_4 \sqrt{J_{\tau-1}} + \varepsilon_\tau,$$

and

$$\log RV_\tau = \beta_0 + \beta_1 \log RV_{\tau-1} + \beta_2 \log RV_{\tau-1}^w + \beta_3 \log RV_{\tau-1}^m + \beta_4 \log(1 + J_{\tau-1}) + \varepsilon_\tau.$$

These transformations can be accessed through the `transform` argument.

The aggregation of the realized measures into weekly and monthly values is implemented in a flexible manner such that one can specify arbitrary lags and an arbitrary number of them. The arguments to control the aggregation are `periods`, `periodsJ`, and `periodsQ` for the different components inside a HAR model. Additionally, as a simple extension, all models can be augmented with an external regressor through the `externalRegressor` argument. This external regressor can be aggregated like the other regressors through the `periodsExternal` argument.

8.2. HEAVY model

The HEAVY model relies both on high-frequency data and on low-frequency data. The daily returns are denoted by r_τ , and the realized measure is denoted by RM_τ . Following the notation from Shephard and Sheppard (2010), the HEAVY model is written as

$$\begin{aligned} \text{VAR}\left(r_\tau | \mathcal{F}_{\tau-1}^{\text{HF}}\right) &= h_\tau = \omega + \alpha RM_{\tau-1} + \beta h_{\tau-1}, \quad \omega, \alpha \geq 0, \quad \beta \in [0, 1), \\ \text{E}\left(RM_\tau | \mathcal{F}_{\tau-1}^{\text{HF}}\right) &= \mu_\tau = \omega_R + \alpha_R RM_{\tau-1} + \beta_R \mu_{\tau-1}, \quad \omega_R, \alpha_R, \beta_R \geq 0, \alpha_R + \beta_R \in [0, 1), \end{aligned}$$

where $\mathcal{F}_\tau^{\text{HF}}$ denotes the information that contains all observations before and including day τ . The HEAVY model can be estimated via the `HEAVYmodel` function. The main input is an `xts` time series that contains returns in the first column and the realized measure of interest in the second column. One example of estimating a HEAVY model based on log-returns and the five-minute RV estimates is the following:

```
R> logReturns <- 100 * makeReturns(SPYRM$CLOSE)[-1]
R> dataSPY <- xts(cbind(logReturns, RV = SPYRM$RV5[-1] * 10000),
+   order.by = SPYRM$DT[-1])
R> model <- HEAVYmodel(data = dataSPY[, c("logReturns", "RV")])
R> summary(model)
```

	Estimate	Std. Error	t value	Pr(> t)
omega	0.0401	0.01060	3.78	1.58e-04
alpha	0.1904	0.03222	5.91	3.44e-09
beta	0.7557	0.03044	24.82	4.99e-136
omegaR	0.0300	0.00513	5.84	5.11e-09
alphaR	0.7314	0.07902	9.26	2.13e-20
betaR	0.2298	0.06139	3.74	1.82e-04

The log-likelihoods are:

Variance equation: -1692.795

RM equation: -1630.093

In this full sample estimation, we observe that all parameters, both in the variance and in the measurement equation, are highly significant.

8.3. Forecasting results

We now turn to a pseudo real-time forecasting comparison of the HAR and HEAVY model. We mimic the information set at time τ and estimate the models based on a rolling window of 1000 observation and make a one-step-ahead out-of-sample forecast. The evaluation sample consists of the data in the SPYRM dataset from 2018–2019. In total, we have 494 out-of-sample observations.

```
R> dataSPY$HARfcst <- NA
R> dataSPY$HEAVYfcst <- NA
R> for (i in 1:494) {
+   HAREstimated <- HARmodel(data = dataSPY[i:(i + 999)], "RV"),
+   periods = c(1, 5, 22), type = "HAR",
+   inputType = "RM")
+   dataSPY$HARfcst[i+1000] <- predict(HAREstimated)
+   HEAVYEstimated <- HEAVYmodel(
+   data = dataSPY[i:(i + 999)], c("logReturns", "RV"))
+   dataSPY$HEAVYfcst[i+1000] <-
+   predict(HEAVYEstimated, stepsAhead = 1)[,"CondRM"]
+ }
```

In Figure 8 we depict the realized variances among the predictions derived from the HAR and HEAVY model.

Considering the forecasts graphically, both models seem to predict the observed values reasonably well, but we can identify some broad trends. The HAR model seems to over-predict the realized volatility on low-volatility days, while the HEAVY model does not show this behavior. The HAR model is less sensitive to volatility spikes and shows a lower willingness to adapt to spikes in volatility, while the HEAVY model changes much more rapidly.

The HEAVY model almost always provides a lower forecast than the HAR model. The picture is not clear-cut, therefore we cannot conclude which model performs best in this circumstance. Hence, we turn to forecast evaluation via the root-mean-square error (RMSE):

```
R> sqrt(mean((dataSPY$RV - dataSPY$HEAVYfcst)^2, na.rm = TRUE))
```

```
[1] 0.62
```

```
R> sqrt(mean((dataSPY$RV - dataSPY$HARfcst)^2, na.rm = TRUE))
```

```
[1] 0.701
```

If we look at the RMSE of the two models, we observe that in our test sample, the HEAVY model has a RMSE that is around 12% smaller than the RMSE of the HAR model. This suggests that the HEAVY model performs better than the HAR model on average. However, this is not the entire story. The performance in times of low volatility may be less important to risk advisers than how well the models perform in times of high volatility. Therefore, we isolate the days with the ex-post 10% highest realized volatility in the evaluation sample and compare the RMSEs on this subsample separately.

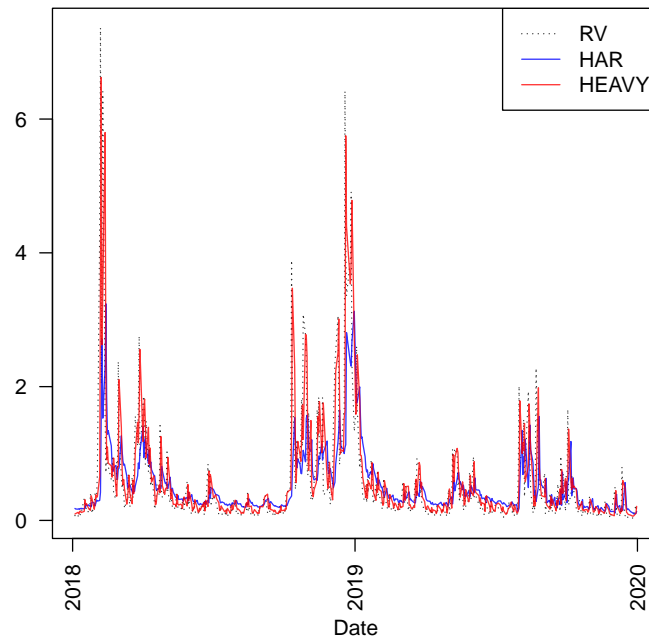


Figure 8: RV realizations in the out-of-sample forecasts for the S&P 500 and corresponding forecasts of a HAR and a HEAVY model. The depicted out-of-sample period spans Jan 2018–Dec 2019. The forecasts are based on a rolling window comprising 1000 daily observations each. Daily scale.

```
R> idx <- dataSPY$RV > quantile(dataSPY$RV[-c(1:1000)], 0.90)
R> sqrt(mean((dataSPY$RV[idx] - dataSPY$HEAVYfcst[idx])^2, na.rm = TRUE))
```

```
[1] 1.72
```

```
R> sqrt(mean((dataSPY$RV[idx] - dataSPY$HARfcst[idx])^2, na.rm = TRUE))
```

```
[1] 2.02
```

The HEAVY model has a RMSE that is about 14% lower than the HAR model on days of high volatility. Most financial actors will happily choose a model which performs better in high-volatility environments instead of a model which performs better in low-volatility environments. Therefore, we can conclude that the HEAVY model is more attractive than the HAR model based on our sample data.

9. Conclusion

In this article we have introduced the R package **highfrequency** for high-frequency data analysis. The package provides valuable tools for practitioners, academics, and regulators alike to conduct their applied research in a user-friendly manner.

We have introduced several major topics in the literature: data cleaning and handling, measures of liquidity and estimating the direction of a trade, realized measures of the return distribution, spot volatility estimation, testing for the presence of jumps in the price process, and modeling and forecasting realized volatility. All of these topics are important parts of the literature and the provided tools can be used to conduct many different analyses and tasks such as data cleaning, risk management, visualization, and more.

If you use the R package **highfrequency** in publications, please cite the software using the following command `citation(package = "highfrequency")`.

Acknowledgement

The authors acknowledge Google for financial support via the Google Summer of Code initiative in the years 2012, 2013, 2014, 2019, and 2020. In addition, we thank Chris Blakely, Nabil Bouamara, Jonathan Cornelissen, Dirk Eddelbuettel, Giang Nguyen, Scott Payseur, Brian Peterson, Maarten Schermer, and Eric Zivot for their support in the development of the **highfrequency** package. Moreover, we thank Andreas Alfons for giving valuable feedback.

References

- Allaire JJ, Xie Y, McPherson J, Luraschi J, Ushey K, Atkins A, Wickham H, Cheng J, Chang W, Iannone R (2022). **rmarkdown**: *Dynamic Documents for R*. R package version 2.14, URL <https://CRAN.R-project.org/package=rmarkdown>.
- Amaya D, Christoffersen P, Jacobs K, Vasquez A (2015). “Does Realized Skewness and Kurtosis Predict the Cross-Section of Equity Returns?” *Journal of Financial Economics*, **118**(1), 135–167. doi:10.1016/j.jfineco.2015.02.009.
- Andersen TG, Bollerslev T, Diebold FX (2007). “Roughing It Up: Including Jump Components in the Measurement, Modeling and Forecasting of Return Volatility.” *The Review of Economics and Statistics*, **89**(4), 701–720. doi:10.1162/rest.89.4.701.
- Andersen TG, Bollerslev T, Diebold FX, Labys P (2001). “The Distribution of Realized Exchange Rate Volatility.” *Journal of the American Statistical Association*, **96**(453), 42–55. doi:10.1198/016214501750332965.
- Andersen TG, Dobrev D, Schaumburg E (2012). “Jump-Robust Volatility Estimation Using Nearest Neighbor Truncation.” *Journal of Econometrics*, **169**(1), 75–93. doi:10.1016/j.jeconom.2012.01.011.
- Barndorff-Nielsen OE, Hansen PR, Lunde A, Shephard N (2008). “Designing Realized Kernels to Measure the Ex Post Variation of Equity Prices in the Presence of Noise.” *Econometrica*, **76**(6), 1481–1536. doi:10.3982/ecta6495.
- Barndorff-Nielsen OE, Hansen PR, Lunde A, Shephard N (2009). “Realized Kernels in Practice: Trades and Quotes.” *The Econometrics Journal*, **12**(3), 1–32. doi:10.1111/j.1368-423x.2008.00275.x.

- Barndorff-Nielsen OE, Shephard N (2004a). “Econometric Analysis of Realized Covariation: High Frequency Based Covariance, Regression, and Correlation in Financial Economics.” *Econometrica*, **72**(3), 885–925. doi:10.1111/j.1468-0262.2004.00515.x.
- Barndorff-Nielsen OE, Shephard N (2004b). “Power and Bipower Variation with Stochastic Volatility and Jumps.” *Journal of Financial Econometrics*, **2**(1), 1–37. doi:10.1093/jjfinec/nbh001.
- Barndorff-Nielsen OE, Shephard N (2006). “Econometrics of Testing for Jumps in Financial Economics Using Bipower Variation.” *Journal of Financial Econometrics*, **4**(1), 1–30. doi:10.1093/jjfinec/nbi022.
- Bollerslev T, Li J, Patton AJ, Quaadvlieg R (2020). “Realized Semicovariances.” *Econometrica*, **169**(1), 75–93. doi:10.3982/ecta17056.
- Bollerslev T, Patton AJ, Quaadvlieg R (2016). “Exploiting the Errors: A Simple Approach for Improved Volatility Forecasting.” *Journal of Econometrics*, **192**(1), 1–18. doi:10.1016/j.jeconom.2015.10.007.
- Boudt K, Cornelissen J, Payseur S, Kleen O, Sjørup E (2022). *highfrequency: Tools for Highfrequency Data Analysis*. R package version 1.0.0, URL <https://CRAN.R-project.org/package=highfrequency>.
- Boudt K, Croux C, Laurent S (2008). “Outlyingness Weighted Covariation.” *Journal of Financial Econometrics*, **9**(4), 249–273. doi:10.1093/jjfinec/nbr003.
- Boudt K, Croux C, Laurent S (2011). “Robust Estimation of Intraday Periodicity in Volatility and Jump Detection.” *Journal of Empirical Finance*, **18**(2), 353 – 367. doi:10.1016/j.jempfin.2010.11.005.
- Boudt K, Dragun K, Sauri O, Vanduffel S (2021). “Beta-Adjusted Covariance Estimation.” *SSRN Electronic Journal*, pp. 1–50. doi:10.2139/ssrn.3768326.
- Boudt K, Laurent S, Lunde A, Quaadvlieg R, Sauri O (2017). “Positive Semidefinite Integrated Covariance Estimation, Factorizations and Asynchronicity.” *Journal of Econometrics*, **196**(2), 347–367. doi:10.1016/j.jeconom.2016.09.016.
- Boudt K, Petitjean M (2014). “Intraday Liquidity Dynamics and News Releases Around Price Jumps: Evidence from the DJIA Stocks.” *Journal of Financial Markets*, **17**, 121–149. doi:10.1016/j.finmar.2013.05.004.
- Boudt K, Zhang J (2010). “Jump Robust Two Time Scale Covariance Estimation and Realized Volatility Budgets.” *Quantitative Finance*, **15**(6), 1041–1054. doi:10.1080/14697688.2012.741692.
- Christensen K, Oomen R, Podolskij M (2014). “Fact or Friction: Jumps at Ultra High Frequency.” *Journal of Financial Economics*, **114**(3), 576–599. doi:10.1016/j.jfineco.2014.07.007.
- Corsi F (2009). “A Simple Approximate Long-Memory Model of Realized Volatility.” *Journal of Financial Econometrics*, **7**(2), 174–196. doi:10.1093/jjfinec/nbp001.

- Dancho M, Vaughan F (2020). **alphavantager**: *Lightweight R Interface to the Alpha Vantage API*. R package version 0.1.2, URL <https://CRAN.R-project.org/package=alphavantager>.
- Dancho M, Vaughan F (2022). **tidyquant**: *Tidy Quantitative Financial Analysis*. R package version 1.0.5, URL <https://CRAN.R-project.org/package=tidyquant>.
- deB Harris FH, McInish TH, Shoesmith GL, Wood RA (1995). “Cointegration, Error Correction, and Price Discovery on Informationally Linked Security Markets.” *The Journal of Financial and Quantitative Analysis*, **30**(4), 563–579. doi:10.2307/2331277.
- Dowle M, Srinivasan A (2021). **data.table**: *Extension of data.frame*. R package version 1.14.2, URL <https://CRAN.R-project.org/package=data.table>.
- Eddelbuettel D, François R (2011). “**Rcpp**: Seamless R and C++ Integration.” *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.
- Eddelbuettel D, Sanderson C (2014). “**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra.” *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.
- Epps TW (1979). “Comovements in Stock Prices in the Very Short Run.” *Journal of the American Statistical Association*, **74**(366), 291–298. doi:10.1080/01621459.1979.10482508.
- Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2021). **mvtnorm**: *Multivariate Normal and t Distributions*. R package version 1.1-3, URL <https://CRAN.R-project.org/package=mvtnorm>.
- Ghalanos A (2022). **rugarch**: *Univariate GARCH Models*. R package version 1.4-8, URL <https://CRAN.R-project.org/package=rugarch>.
- Ghalanos A, Theussl S (2015). **Rsolnp**: *General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16, URL <https://CRAN.R-project.org/package=Rsolnp>.
- Gilbert P, Varadhan R (2019). **numDeriv**: *Accurate Numerical Derivatives*. R package version 2016.8-1.1, URL <https://CRAN.R-project.org/package=numDeriv>.
- Hansen PR, Lunde A (2006). “Realized Variance and Market Microstructure Noise.” *Journal of Business & Economic Statistics*, **24**(2), 127–161. doi:10.1198/073500106000000071.
- Hautsch N, Podolskij M (2013). “Preaveraging-Based Estimation of Quadratic Variation in the Presence of Noise and Jumps: Theory, Implementation, and Empirical Evidence.” *Journal of Business & Economic Statistics*, **31**(2), 165–183. doi:10.1080/07350015.2012.754313.
- Hayashi T, Yoshida N (2005). “On Covariance Estimation of Non-Synchronously Observed Diffusion Processes.” *Bernoulli*, **11**(2), 359–379. doi:10.3150/bj/1116340299.
- Hester J (2020). **covr**: *Test Coverage for Packages*. R package version 3.5.1, URL <https://CRAN.R-project.org/package=covr>.
- Kristensen D (2010). “Nonparametric Filtering of the Realized Spot Volatility: A Kernel-Based Approach.” *Econometric Theory*, **26**(1), 60 – 93. doi:10.1017/s0266466609090616.

- Lee CMC, Ready MJ (1991). “Inferring Trade Direction from Intraday Data.” *The Journal of Finance*, **46**(2), 733–46. doi:10.1111/j.1540-6261.1991.tb02683.x.
- Lee SS, Mykland PA (2008). “Jumps in Financial Markets: A New Nonparametric Test and Jump Dynamics.” *The Review of Financial Studies*, **21**(6), 2535–2563. doi:10.1093/rfs/hhm056.
- Li M, Linton O (2021). “A ReMeDI for Microstructure Noise.” *Econometrica*, **90**(1). doi:10.3982/ecta17505.
- Luethi D, Erb P, Otziger S, McDonald D, Smith P (2021). **FKF: Fast Kalman Filter**. R package version 0.2.3, URL <https://CRAN.R-project.org/package=FKF>.
- Maechler M, Rousseeuw P, Croux C, Todorov V, Ruckstuhl A, Salibian-Barrera M, Verbeke T, Koller M, Conceicao ELT, Anna di Palma M (2022). **robustbase: Basic Robust Statistics**. R package version 0.95-0, URL <https://CRAN.R-project.org/package=robustbase>.
- Mancini C, Gobbi F (2012). “Identifying the Brownian Covariation from the Co-Jumps Given Discrete Observations.” *Econometric Theory*, **28**(2), 249–273. doi:10.1017/s0266466611000326.
- Oomen R (2006). “Properties of Realized Variance Under Alternative Sampling Schemes.” *Journal of Business & Economic Statistics*, **24**(2), 219–237. doi:10.1198/073500106000000044.
- Pawlowski J (2021). **HighFreq: High Frequency Time Series Management**. R package version 0.1, URL <https://github.com/algoquant/HighFreq>.
- R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Ryan JA, Ulrich JM (2020). **xts: eXtensible Time Series**. R package version 0.12.1, URL <https://CRAN.R-project.org/package=xts>.
- Ryan JA, Ulrich JM (2022). **quantmod: Quantitative Financial Modelling Framework**. R package version 0.4.20, URL <https://CRAN.R-project.org/package=quantmod>.
- Sarkar D (2008). **lattice: Multivariate Data Visualization with R**. Springer-Verlag, New York. doi:10.1007/978-0-387-75969-2. ISBN 978-0-387-75968-5.
- Shephard N, Sheppard K (2010). “Realising the Future: Forecasting with High-Frequency-Based Volatility (HEAVY) Models.” *Journal of Applied Econometrics*, **25**, 197–231. doi:10.1002/jae.1158.
- Ushey K (2018). **RcppRoll: Efficient Rolling/Windowed Operations**. R package version 0.3.0, URL <https://CRAN.R-project.org/package=RcppRoll>.
- Vergote O (2005). “How to Match Trades and Quotes for NYSE Stocks?” *SSRN Electronic Journal*. doi:10.2139/ssrn.808984.
- Wickham H (2011). “**testthat: Get Started with Testing**.” *The R Journal*, **3**, 5–10. doi:10.32614/RJ-2011-002.

- Wickham H, Seidel D (2022). *scales: Scale Functions for Visualization*. R package version 1.2.0, URL <https://CRAN.R-project.org/package=scales>.
- Xie Y (2022). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.39, URL <https://CRAN.R-project.org/package=knitr>.
- Zeileis A, Grothendieck G (2005). “**zoo**: S3 Infrastructure for Regular and Irregular Time Series.” *Journal of Statistical Software*, **14**(6), 1–27. doi:10.18637/jss.v014.i06.
- Zeileis A, Köll S, Graham N (2020). “Various Versatile Variances: An Object-Oriented Implementation of Clustered Covariances in R.” *Journal of Statistical Software*, **95**(1), 1–36. doi:10.18637/jss.v095.i01.
- Zhang L, Mykland PA, Ait-Sahalia Y (2005). “A Tale of Two Time Scales: Determining Integrated Volatility with Noisy High-Frequency Data.” *Journal of the American Statistical Association*, **100**(472), 1394–1411. doi:10.1198/016214505000000169.

A. Table of liquidity measures

Name of estimator	Estimation formula
Effective Spread	$ES_t = 2D_t \times [PRICE_t - (BID_t + OFR_t)/2]$
Realized Spread	$RS_t = 2D_t \times [PRICE_t - (BID_{t+300} + OFR_{t+300})/2]$
Trade value	$TV_t = SIZE_t \times PRICE_t$
Signed trade value	$STV_t = D_t \times (SIZE_t \times PRICE_t)$
Depth imbalance (difference)	$DI_d_t = D_t \times (OFRSIZ_t - BIDSIZ_t) / (OFRSIZ_t + BIDSIZ_t)$
Depth imbalance (ratio)	$DI_r_t = (OFRSIZ_t / BIDSIZ_t)^{D_t}$
Proportional effective spread	$PES_t = ES_t / [(OFR_t + BID_t) / 2]$
Proportional realized spread	$PRS_t = RS_t / [(OFR_t + BID_t) / 2]$
Price impact	$PI_t = (ES_t - RS_t) / 2$
Proportional price impact	$PPI_t = [(ES_t - RS_t) / 2] / [(OFR_t + BID_t) / 2]$
Half traded spread	$HTS_t = D_t \times [PRICE_t - (BID_t + OFR_t) / 2]$
Proportional half traded spread	$PHTS_t = HTS_t / [(OFR_t + BID_t) / 2]$
Squared log returns	$SLR_t = [\log(PRICE_t) - \log(PRICE_{t-1})]^2$
Absolute log returns	$ALR_t = \log(PRICE_t) - \log(PRICE_{t-1}) $
Quoted spread	$QSpr_t = OFR_t - BID_t$
Proportional quoted spread	$PQS_t = QSpr_t / [(OFR_t + BID_t) / 2]$
Log quoted spread	$LQSpr_t = \log(OFR_t) - \log(BID_t)$
Log quoted size	$LQSiz_t = \log(OFRSIZ_t) + \log(BIDSIZ_t)$
Quoted slope	$QSlo_t = QSpr_t / LQSiz_t$
Log quoted slope	$LQSlo_t = LQSpr_t / LQSiz_t$
Midquote squared return	$MQSR_t = [\log(MIDQUOTE_t) - \log(MIDQUOTE_{t-1})]^2$
Midquote absolute return	$MQAR_t = \log(MIDQUOTE_t) - \log(MIDQUOTE_{t-1}) $
Signed trade size	$STS_t = D_t \times SIZE_t$

Table 1: Formulae for the liquidity measures provided in the `getLiquidityMeasures` function. The variable D_t is the sign of the trade, indicating whether the buyer (seller) is the initiator of the trade if the buyer (seller) is the initiator $D_T = 1$ ($D_T = -1$). This column can be provided if the data is available (for example futures data) otherwise it is automatically estimated by the Lee-Ready algorithm.

B. Table of realized measures

Estimator	Citation	Function	Jump	Noise	Tick
<i>Multivariate estimators</i>					
Realized Variance	Andersen <i>et al.</i> (2001)	rCov	no	no	no
Bipower Variation	Barndorff-Nielsen and Shephard (2004b)	rBPCov	yes	no	no
Realized kernel covariance	Barndorff-Nielsen <i>et al.</i> (2009)	rKernelCov	no	yes	no
Twoscale covariance	Zhang <i>et al.</i> (2005)	rTSCov	no	yes	yes
Robust twoscale covariance	Boudt and Zhang (2010)	rRTSCov	yes	yes	yes
Threshold covariance	Mancini and Gobbi (2012)	rThresholdCov	yes	no	no
Semi-covariance	Bollerslev, Li, Patton, and Quaechvlieg (2020)	rSemiCov	no	no	no
Cholesky Covariance	Boudt <i>et al.</i> (2017)	rCholCov	yes	yes	yes
Outlyingness weighted covariance	Boudt, Croux, and Laurent (2008)	rOWCov	yes	no	no
Hayashi-Yoshida covariance	Hayashi and Yoshida (2005)	rHYCov	yes	no	no
Average subsample covariance	Zhang <i>et al.</i> (2005)	rAVGCov	no	yes	no
Modulated realized covariance	Hautsch and Podolskij (2013)	rMRCov	no	yes	yes
Beta-Adjusted covariance	Boudt, Dragun, Sauri, and Vanduffel (2021)	rBACov	yes	yes	yes
<i>Univariate estimators</i>					
Realized Variance	Andersen <i>et al.</i> (2001)	rRVar	no	no	no
Median Realized variance	Andersen, Dobrev, and Schaumburg (2012)	rMedRV	yes	no	no
Median Realized quarticity	Andersen <i>et al.</i> (2012)	rMedRQ	yes	no	no
Minimum Realized Variance	Andersen <i>et al.</i> (2012)	rMinRV	yes	no	no
Minimum Realized Quarticity	Andersen <i>et al.</i> (2012)	rMinRQ	yes	no	no
Realized Quadpower variation	Andersen <i>et al.</i> (2012)	rQPVar	yes	no	no
Realized Quarticity	Andersen <i>et al.</i> (2012)	rQuar	no	no	no
Realized tripower quarticity	Andersen <i>et al.</i> (2012)	rTPQuar	yes	no	no
Realized Kurtosis	Amaya <i>et al.</i> (2015)	rKurt	no	no	no
Realized Skewness	Amaya <i>et al.</i> (2015)	rSkew	no	no	no

Table 2: Table containing the function names of the realized measures available in the **high-frequency** package. The last two columns denote whether estimators are robust (yes) or not robust (no) to jumps and market microstructure noise, respectively. All the estimators can handle tick-by-tick input, the last column is used to distinguish functions that necessitate tick-by-tick data and do not work as expected with aggregated one-minute data.

C. Computational details

The results in this paper were obtained using R version 4.2.1 with the following packages: **highfrequency** version 1.0.0 of Boudt *et al.* (2022), **Rcpp** version 1.0.9 of Eddelbuettel and François (2011), **xts** version 0.12.1 of Ryan and Ulrich (2020), **zoo** version 1.8-10 of Zeileis and Grothendieck (2005), **RcppAmadillo** version 0.11.2.0.0 of Eddelbuettel and Sanderson (2014), **robustbase** version 0.95-0 of Maechler *et al.* (2022), **data.table** version 1.14.2 of Dowle and Srinivasan (2021), **RcppRoll** version 0.3.0 of Ushey (2018), **quantmod** version 0.4.20 of Ryan and Ulrich (2022), **sandwich** version 3.0-2 of Zeileis, Köll, and Graham (2020), **numDeriv** version 2016.8-1.1 of Gilbert and Varadhan (2019), **lattice** version 0.20-45 of Sarkar (2008), **scales** version 1.2.0 of Wickham and Seidel (2022), and **Rsolnp** version 1.16 of Ghalanos and Theussl (2015). Computations were performed on macOS 12.4 with an Intel® Core™ i5-1038NG7 CPU @ 2GHz × 4 processor. Suggestions of the **highfrequency** package are the packages **mvtnorm** (Genz *et al.* 2021), **covr** (Hester 2020), **FKF** (Luethi, Erb, Oztiger, McDonald, and Smith 2021), **rugarch** (Ghalanos 2022), **testthat** (Wickham 2011), **knitr** (Xie 2022), and **rmarkdown** (Allaire *et al.* 2022).

The code used in the main paper is available in the R script `run_vignette.R` located in the examples folder on the dedicated **highfrequency** GitHub repository at <https://github.com/jonathancornelissen/highfrequency>. R, **highfrequency**, and all other packages are available from the Comprehensive R Archive Network at <http://CRAN.R-project.org/>. Any version under development will be available on our GitHub repository.

Affiliation:

Kris Boudt
 Department of Economics
 Ghent University, Belgium
and
 Solvay Business School
 Vrije Universiteit Brussel, Belgium
and
 Econometrics and Data Science Department
 Vrije Universiteit Amsterdam, The Netherlands
 E-mail: kris.boudt@ugent.be

Onno Kleen
 Erasmus School of Economics
 Erasmus University Rotterdam, The Netherlands
 E-mail: kleen@ese.eur.nl