RTAQ
Tools for the analysis of trades and quotes in R

Kris Boudt
Lessius and K.U.Leuven

Jonathan Cornelissen
K.U.Leuven

Abstract

The Trades and Quotes data of the New York Stock Exchange is a popular input for the implementation of intraday trading strategies, the measurement of liquidity and volatility and investigation of the market microstructure, among others. This package contains a collection of R functions to carefully clean and match the trades and quotes data, calculate ex post liquidity and volatility measures and detect price jumps in the data.

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

1. General information

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 `RTAQ` 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 `RTAQ` package offers a (basic) interface to manage Trades and Quotes (TAQ) data. Furthermore and foremost, it offers easy-to-use tools to clean and aggregate high-frequency data, and to calculate volatility and liquidity measures. The tools for data management and cleaning in `RTAQ` are specifically designed for data from the New York Stock Exchange (NYSE) TAQ database, in constrast to the liquidity and volatility functions which will work with most types of data input. On the topic of volality measurement `RTAQ` wants to be complementary to the `realized` package (Payseur 2008) by focussing on robust and recently introduced volalitity measures. The function `as.realizedObject` serves as a bridge between both packages.

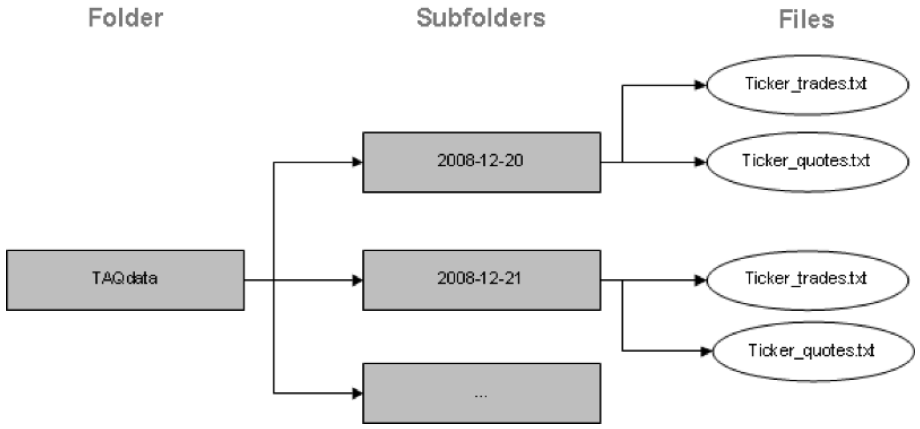
`RTAQ` strongly builds on the functionality offered by the `xts` package (Ryan and Ulrich 2009) and the `timeDate` (Wuertz and Chalabi 2009) package.

The `RTAQ` package is part of the TradeAnalytics project on R-forge and the latest version of the package can be downloaded through the following command:

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

The latest stable and fully documented version can be downloaded from CRAN by:

Figure 1: Structure of raw data on hard disk.




```
install.packages("RTAQ")
```

2. TAQ data management essentials

The TAQ database of the NYSE contains the intraday (high-frequency) information for all listed stocks on both the transactions and the best quotes of the designated market maker (formerly called specialist). This enables researchers and practitioners to measure volatility as wel as liquidity more precisely, and to investigate microstructure issues, which explains the popularity of the TAQ database. The data is typically subdivided into two files, containing information on the trades and quotes respectively.

2.1. Raw data conversion

Raw TAQ data can be obtained in several ways.[1] Therefore, we need to set a "standard" way to store and organise the data, before the functions for the analysis can be presented. We recommend to save the raw data in txt format and structure it as illustrated by Figure 1. This means that the folder "TAQdata" contains a number of folders (one for each trading day in the sample), while each of these folders contains two txt files for each stock, containing the information on trades and quotes respectively.

In a following step, the function `convert` can be used to convert the txt files into xts objects (Ryan and Ulrich 2009) that are saved on-disk in the "RData" format. The function `convert` maintains the same folder and name structure as in Figure 1 (except for the fact that the data is now stored as ".RData" obviously). As a next step, the function `TAQLoad` can be used to load the on-disk data into your R workspace.

We opt to store the data as xts objects because this type of object can be indexed by an indica-

[1]The NYSE itself delivers historical data on monthly DVD's or for recent data through the "TAQ web". TAQ data can also be obtained through external vendors such as the Wharton Research Data Services.

tor of time and date. We choose an indicator of class "timeDate" (Wuertz and Chalabi 2009). [2]

2.2. Trades and quotes data description

Both trades and quotes data are thus saved as xts objects (Ryan and Ulrich 2009), using e.g. the `convert` function. Table 1 reports the information (i.e. columns) in trades and quotes data objects and the package contains two sample datasets in the data format it assumes:

```
> library(RTAQ);
> data("sample_tdataraw");
> head(sample_tdataraw);

                    SYMBOL EX  PRICE    SIZE     COND CR  G127
2008-01-04 09:30:26 "XXX"  "N" "193.76" "345050" "O"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "100"    "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "400"    "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"

> data("sample_qdataraw");
> head(sample_tdataraw);

                    SYMBOL EX  PRICE    SIZE     COND CR  G127
2008-01-04 09:30:26 "XXX"  "N" "193.76" "345050" "O"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "100"    "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "400"    "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"
2008-01-04 09:30:27 "XXX"  "N" "193.82" "50"     "E"  "0" "0"
```

2.3. Data cleaning

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). `RTAQ` 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 should be organised as described above to benefit from this functionality.

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

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

```
> 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)
[1] 9105    7
```

3. Liquidity

3.1. Mathing trades and quotes

Trades and quotes are supplied by the NYSE as separate data objects. For many research 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.

3.2. Inferred trade direction

The NYSE TAQ database does not indicate whether individual trades are market buy or

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)

market sell orders. `RTAQ` implements with `getTradeDirection` the Lee-Ready rule (Lee and Ready 1991) to infer the trade direction based on the matched trades and quotes.

3.3. 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 Seppi (2001) and Venkataraman (2001) for more information on the implemented liquidity measures). The main implemented liquidity measures are listen in Table 3, 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] "SYMBOL" "EX"     "PRICE"  "SIZE"   "COND"   "CORR"   "G127"   "BID"
 [9] "BIDSIZ" "OFR"    "OFRSIZ" "MODE"
```

Table 3: 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

```
> #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");
```

4. Aggregation and volatility

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. While numerous realized (co)volatility estimators have been implemented in the `realized` package (Payseur 2008), `RTAQ` also implements jump-robust and recently introduced (co)volatility estimators. The function `as.realizedObject` can be used to convert the xts objects used in `RTAQ`, to realized objects.

An overview of the univariate and multivariate volatility estimators implemented in `RTAQ` is given in Table 4. 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. Non positive semidefinite estimates can be transformed into a positive semidefinite matrix with the function `makePsd`, which uses the eigenvalue method.

While most multivariate estimators rely on synchronized data (see e.g. Table 4), prices are typically recorded at different points in time for different assets. There several ways to force these asynchronally recorded series to a synchronized and/or equispaced time grid. The most

Table 4: Overview of Volatility estimators

Estimator	Univariate	Multivariate	Jump robust	Microstructure noise robust	Tick-by-tick returns as input	Positive semidefinite
MedRV (Andersen *et al.* 2010)	x		x			/
MinRV (Andersen *et al.* 2010)	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
thresholdCov (Gobbi and Mancini 2009)		x	x			x
TSCov (Zhang 2011)	x	x		x	x	
RTSCov (Boudt and Zhang 2010)	x	x	x	x	x	

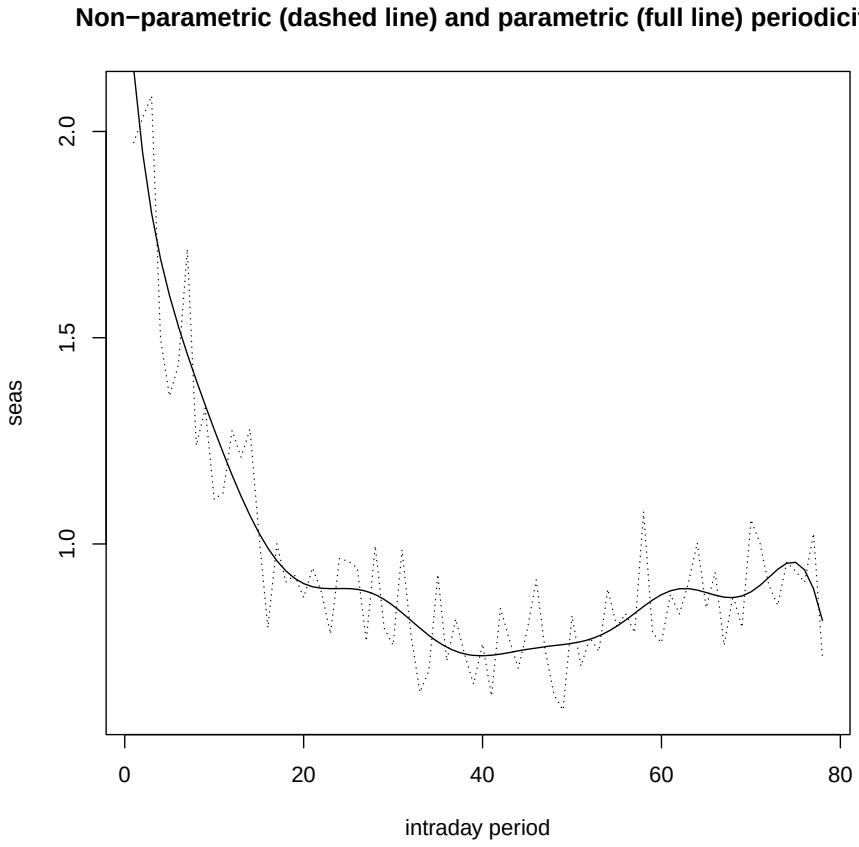
popular method, previous tick aggregation, forces prices to an equispaced grid by taking the last price realized before each grid point. The function `aggregatePrice` is useful for previous tick aggregation of a price series, while the `aggregateTrades` and `aggregateQuotes` can be used for aggregating trade and quote data respectively. Another synchronization method is refresh time, initially proposed by Harris and Wood (1995) and recently advocated in Barndorff-Nielsen *et al.* (2011). The function `refreshTime` can be used to force time series 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. For example, the first refresh time corresponds to the first time at which all stocks have traded. The subsequent refresh time is defined as the first time when all stocks have been traded again. This process is repeated untill the end of one time series is reached.

Illustration on price aggregation and 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));
```

A last interesting feature of high-frequency data is the periodicity in the volatility of high-frequency returns induced by opening, lunch and closing of financial markets (Andersen and Bollerslev 1997). `RTAQ` implements both the classic intraday periodicity estimation method of Andersen and Bollerslev (1997) and a jump robust version proposed by Boudt *et al.* (2011b). These estimation methods assume that the intraday volatility (called `vol` in output) can be decomposed in a daily volatility factor (constant for all intraday returns observed on the

Figure 2: Estimated periodicity pattern using spotVol function.



same day, called `dailyvol`) and an intraday factor (deterministic function of the intraday time, called `periodicvol`). The estimates are either non-parametric (based on a scale estimator) or parametric (based on regression estimation of a flexible specification for the intraday factor). The sample code below illustrates the estimation of intraday periodicity as well as Figure 2:

```
> data("sample_real5minprices");
>
> #compute and plot intraday periodicity
> out = spotVol(sample_real5minprices,P1=6,P2=4,periodicvol="TML",k=5,
+ dummies=FALSE);
> head(out);
                        returns         vol    dailyvol periodicvol
2005-03-04 09:35:00 -0.0010966963 0.004081072 0.001896816    2.151539
2005-03-04 09:40:00 -0.0005614217 0.003695715 0.001896816    1.948379
2005-03-04 09:45:00 -0.0026443880 0.003417950 0.001896816    1.801941
```

Acknowledgements: We would like to thank the participants of the 1st R/Rmetrics Sum-

mer 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 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

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, Labys P (2003). "Modeling and forecasting realized volatility." *Econometrica*, **71**, 579–625.

Andersen TG, Dobrev D, Schaumburg E (2010). "Jump-Robust Volatility Estimation using Nearest Neighbor Truncation." *NBER Working Paper No. 15533*.

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*.

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, forthcoming*.

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.

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

Harris F TMGS, 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.

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

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

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

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*.

Wuertz D, Chalabi Y (2009). *timeDate: Rmetrics - Chronological and Calendarical Objects*. R package version 2100.86, URL http://CRAN.R-project.org/package=timeDate.

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
Lessius and K.U.Leuven, Belgium
E-mail: kris.boudt@econ.kuleuven.be

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