RTAQ: Tools for the analysis of trades and quotes in R

Jonathan Cornelissen[*] Kris Boudt[†]

01-03-2010

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 document describes 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.
Latest version of the package: http://r-forge.r-project.org/R/?group_id=316

[*]K.U.Leuven, Belgium.
[†]Lessius University College and K.U.Leuven, Belgium.
Correspondence to: Kris Boudt, Faculty of Business and Economics, 69 Naamsestraat, B-3000 Leuven, Belgium. E-mail: kris.boudt@econ.kuleuven.be Tel: +32 16 326728 Fax: +32 16 326624
Financial support from the National Bank of Belgium is gratefully acknowledged.
NOTE: This is work in progress. If interested, please contact the authors for the latest version of the documentation.

1 Introduction

Handling high-frequency data is particularly challenging because of the specific characteristics of the data, as extensively documented in Bingcheng and Zivot (2003). A first major difficulty is the enormous number of observations, that can reach heights of millions observations per stock per day. Secondly, the recorded data often contains errors for various reasons. Third, transaction-by-transaction data is by nature irregularly spaced over time.

In this document we present functions to clean and analyse this type of data, using the software package **R 2.10.1**. To facilitate further analysis, the data objects are stored in xts (extensible time series) format and the index of each observation is given by a timeDate object. In order for these functions to work properly, the packages "xts" and "timeDate" should be installed and loaded.

The remainder of this paper is organized as follows. The next section gives a description of the data and how to load it into R. Subsequently, section 3 presents the code to clean the raw high-frequency data. In sections 4 to 6 we discuss the functions to calculate liquidity measures, aggregate the data and calculate volatility measures respectively.

2 Data description

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 investigate microstructure issues and to measure volatility as wel as liquidity more precisely, which explains its popularity as a data source. The data is typically subdivided into two files, containing information on the trades and quotes respectively.

1. Raw **trade** data: 9 columns: see Table 1

2. Raw **quote** data: 9 columns: see Table 2

Table 1:
Elements of raw trade data

SYMBOL	The stock's ticker
DATE	Date on which the trade was registered the function convert (section 2.1.1) assumes dd/mm/yyyy format
EX	Exchange on which the trade occurred see section 3.1.2
TIME	Time on which trade was registered function convert (section 2.1.1) assumes hh:mm:ss format
PRICE	Transaction price
SIZE	Number of shares traded
COND	Sales condition code
CR	Correction indicator
G127	Combined "G", Rule 127, and stopped stock trade indicator

The raw TAQ data can be obtained in several ways.[1] Therefore, we are forced to determine a "standard" way to store and organise the data, before the functions for the analysis can be presented. We propose 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 txt files are loaded into R, the data is converted into an xts-object[2] and subsequently stored in the "RData" format. The function convert() presented in section 2.1.1 performs this convertion while maintaining the same folder and name structure as in Figure 1 (except for the fact that the data is now stored as ".RData" obviously). After this process, the converted data can be used as input for the functions proposed in this document.

We opt to store the data as xts-objects because this type of object can be indexed by an indication of time and date, in this case from the class "time-

[1]The NYSE itself delivers historical data on monthly DVD's (A procedure to automatically extract the desired data can be found in Appendix A) or for recent data through the "TAQ web". TAQ data can also be obtained through external vendors such as the Wharton Research Data Services.

[2]More information can be found in the user guide of the package xts (Ryan and Ulrich, 2009).

Table 2:
Elements of raw quote data

SYMBOL	The stock's ticker
DATE	Date on which the trade was registered the function convert (section 2.1.1) assumes dd/mm/yyyy format
EX	Exchange on which the quote occurred. see section 3.1.2
TIME	Time on which trade was registered function convert (section 2.1.1) assumes hh:mm:ss format
BID	Bid price
BIDSIZE	Bid size in number of round lots (100 share units)
OFFER	Offer price
OFFERSIZE	Offer size in number of round lots (100 share units)
MODE	Quote condition

Date".[3] This allows the user to conveniently make use of the functionality of xts and timeDate. For example:[4]

- To select observations from April 2008: xtsobject["2008-04"];
- To select observations from April 1 until april 10, 2008: xtsobject["2008-04-01::2008-04-10"]

This means the DATE and TIME column from the raw data are used to compute the timeDate index of the format "CCYY-MM-DD HH:MM:SS" of each observation and can then be removed. Consequently:

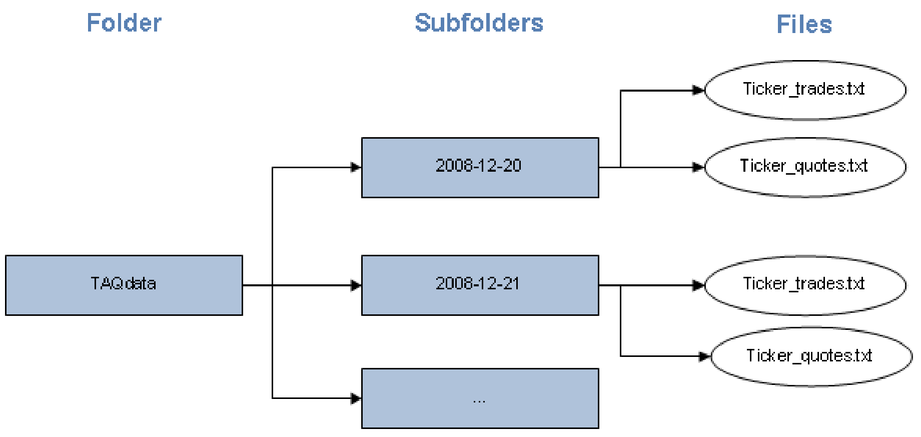
1. The trade data xts-object contains 7 columns and index of the timeDate class: SYMBOL/EX/PRICE/SIZE/COND/CR/G127

2. The quote data xts-object contains 7 columns and an index of the timeDate class: SYMBOL/EX/BID/BIDSIZE/OFFER/OFFERSIZE/MODE

[3]More information can be found in the user guide of the timeDate package (Wuertz and Chalabi, 2009).

[4]More examples of the subsetting of xts objects can be found on "http://www.quantmod.com/examples/data/".

Figure 1: Structure of raw data on hard disk.



2.1 Loading TAQ data in R

2.1.1 convert(from,to,datasource,datadestination,trades=T,quotes=T, ticker,dir=F)

- Function to convert both trade and quote data stored as “txt” files and structured as illustrated in Figure 1 into xts-objects and store them in the “RData” format. Figure 2 illustrates the structure of files after conversion.

- from: first date to convert e.g. “2008-01-30”

- to: last date to convert e.g. “2008-01-31”

- Datasource: folder in which the original data is stored

- Datadestination: folder in which the converted data should be stored

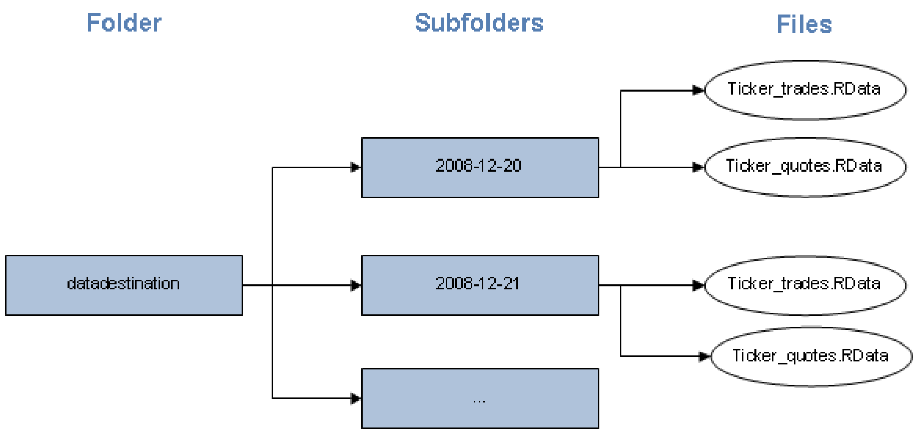
- trades: determines whether trades are converted

- quotes: determines whether quotes are converted

- ticker: vector with tickers to be converted

- dir: if TRUE the datadestination folder and subfolders will be created automatically

Figure 2: Structure of converted data on hard disk.



2.1.2 TAQload(tickers,from,to,trades=T,quotes=F, datasource=NULL,variables=NULL)

- Function to load the taq data from a certain stock. If only the trades (or quotes) should be loaded the function returns them directly as xts object. If both trades and quotes should be extracted the function returns a list with two items named "trades" and "quotes" respectively.
- Given that the files are ordered in folders per day, and the folders contain: ticker_trades.RData or ticker_quotes.RData. In these files the xts object is stored.
- tickers: the ticker(s) to be loaded. It is recommended that you use only 1 ticker as input in case of non-synchronic observations. For synchronic data a vector of tickers can be used.
- from: first day to load e.g. "2008-01-30"
- to: last day to load e.g. "2008-01-30"
- trades: determines whether trades are extracted
- quotes: determines whether quotes are extracted
- datasource: path to folder in which the files are contained
- variables: a character (or character vector) containing the name(s) of the variable(s) that should be loaded, e.g. c("SYMBOL","PRICE"). By default all data is loaded.

2.1.3 matchtq(tdata,qdata,adjustment=2)

- Function matches the trades and quotes and returns xts object containing both tdata (qdata) is an xts object with containing the trades (quotes)
- Adjustment = number of seconds the quotes are registered faster than the trades (should be round and positive). Based on the research of Vergote (2005), we set 2 seconds as the default.

```
matchtq = function(tdata,qdata,adjustment=2){
  tt = dim(tdata)[2];
  index(qdata) = index(qdata) + adjustment;

  #merge:
  merged = merge(tdata,qdata);

  ##fill NA's:
  merged[,((tt+1):dim(merged)[2])] = na.locf(as.zoo(merged[,((tt+1):dim(merged)[2])]), na.rm=FALSE);

  #Select trades:
  index(tdata)=as.POSIXct(index(tdata));
  index(merged)=as.POSIXct(index(merged));
  merged = merged[index(tdata)];

  #return useful parts:
  merged = merged[,c((1:tt),((tt+3):(dim(merged)[2])))];

  ##a bit rough but otherwise opening price disappears...
  merged = as.xts(na.locf(as.zoo(merged),fromLast=TRUE));

  index(merged) = as.timeDate(index(merged));
  return(merged)
}
```

Code snippet 1.
Function to match trades and quotes.

3 Data cleanup functions

Data-cleaning is an essential step in dealing with tick-by-tick data. We follow the step-by-step procedure proposed by Barndorff-Nielsen et al. (2008). The functions to perform the clean-up are presented in schematic way in the following subsections. We refer to "tdata" and "qdata" as the xts-objects that contain the trades and quotes respectively. "alldata" is used when the function works on both tdata and qdata.
First, we present the individual cleanup functions for trades and quotes respectively. Secondly, the wrapper functions are presented that perform all the subsequent cleaning procedures for a number of stocks over a certain time interval. Of course, the data should be organised as described above to benefit from this functionality.

3.1 All data

3.1.1 ExchangeHoursOnly(alldata, daybegin = "09:30:00",dayend="16:00:00")

- Delete entries with a time stamp outside the daybegin-dayend window (9:30-16:00 by default: the NYSE opening hours).
- This function is not used anymore because the TAQ3 software can do this faster.

3.1.2 selectexchange(alldata,exch="N")

- Retain entries from a single exchange (NYSE)
- By default the NYSE is chosen (exch="N")
- The symbols for the other exchanges are:
  - A: AMEX
  - N: NYSE
  - B: Boston
  - P: Arca
  - C: NSX
  - T/Q: NASDAQ
  - D: NASD ADF and TRF
  - X: Philadelphia
  - I: ISE
  - M: Chicago
  - W: CBOE
  - Z: BATS

3.2 Trade data

3.2.1 nozeroprices(tdata)

- Delete entries with a transaction price = 0.

3.2.2 autoselectexchange(tdata)

- Automatically selects observations from the exchange with the highest value for the variable "SIZE", i.e. the highest trade volume.
- The name of the selected exchange is printed on the console.

3.2.3 Directly via TAQ3 software

- No input of corrected trades (Trades with a Correction Indicator, CORR $\neq 0$).

3.2.4 salescond(tdata)

- Delete entries with abnormal Sale Condition. (Trades where COND has a letter code, except for "E" and "F"). [5]

3.2.5 mergesametimestamp(tdata,selection="median")

- If multiple transactions have the same time stamp: use the median price.
- alternatively:
  - selection = "maxvolume": use the price of the transaction with largest volume.
  - selection = "weightedaverage": use take the weighted average of all prices.

3.2.6 rmtradeoutliers(tdata,qdata)

- Delete entries with prices that are above the ask plus the bid-ask spread. Similar for entries with prices below the bid minus the bid-ask spread.
- Note: in order to work correctly, the input of this function should be cleaned trade and quote data respectively.

3.3 Quote data

3.3.1 nozeroquotes(qdata)

- Delete entries with a bid or ask = 0.

3.3.2 autoselectexchangeq(qdata)

- Automatically selects only observations of the exchange with highest value for "BIDSIZE + OFFERSIZE", i.e. the highest volume.
- The name of the selected exchange is printed on the console.

[5]See the TAQ3 (version 1.1.9) User's Guide for additional details about sale conditions.

3.3.3 mergequotessametimestamp(qdata)

- When multiple quotes have the same timestamp, we replace all these with a single entry with the median bid and median ask price.

3.3.4 rmnegspread(qdata)

- Delete entries for which the spread is negative.

3.3.5 rmlargespread(qdata,maxi=50)

- Delete entries for which the spread is more than "maxi" times the median spread on that day.

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

- if type = "standard":
  Delete entries for which the mid-quote deviated by more than "maxi" median absolute deviations[6] from a rolling centered median (excluding the observation under consideration) of "window" observations.
- if type = "advanced":
  Remove entries for which the mid-quote deviates by more than "maxi" median absolute deviations from the value closest to the midquote of these three options:
  1. Rolling centered median (excluding the observation under consideration)
  2. Rolling median of the following "window" observations
  3. Rolling median of the previous "window" observations

  The advantage of this procedure compared to the "standard" proposed by Barndorff-Nielsen et al. (2008) is that it will not incorrectly remove large price jumps. Therefore this procedure has been set as the default for removing outliers.

3.4 Total cleanup wrappers

3.4.1 tradescleanup(from,to,datasource,datadestination,ticker)

- Function performs all cleaning procedures mentioned for "all" and "trade" data for all stocks in "ticker" over the interval [from,to] and saves the

[6]On this point we deviate from Barndorff-Nielsen et al. (2008) who propose the "mean" absolute deviation, because their method is sensitive to outlier masking.

result in the folder datadestination. Since the function “rmtradeoutliers” also requires cleaned quote data as input, it is not incorporated here and there is a seperate wrapper (see section 3.4.3).

- from: first date to clean e.g. “2008-01-30”
- to: last date to clean e.g. “2008-01-31”
- datasource: folder in which the original data is stored
- datadestination: folder in which the cleaned data should be stored
- ticker: vector of tickers for which the data should be cleaned
- NOTE: In case trades for a certain stock on a certain date are missing, the corresponding folder (see Figure 1) should contain a file “missing_ticker.RData” to indicate that it can be skipped. If you have used the function “convert”, this will automatically be the case.

3.4.2 quotescleanup(from,to,datasource,datadestination,ticker)

- Function performs all cleaning procedures mentioned for “all” and “quote” data for all stocks in “ticker” over the interval [from,to] and saves the result in datadestination
- from: first date to clean e.g. “2008-01-30”
- to: last date to clean e.g. “2008-01-31”
- datasource: folder in which the original data is stored
- datadestination: folder in which the cleaned data should be stored
- ticker: vector of tickers for which the data should be cleaned
- NOTE: In case quotes for a certain stock on a certain date are missing, the corresponding folder (see Figure 1) should contain a file “missingquotes_ticker.RData” to indicate that it can be skipped. If you have used the function “convert”, this will automatically be the case.

3.4.3 tradescleanup_finalop(from,to,datasource,datadestination,ticker)

- Function performs cleaning procedure "rmtradeoutliers" for the trades of all stocks in "ticker" over the interval [from,to] and saves the result in datadestination. Note that also (cleaned) quotes should be available in "datasource".
- from: first date to clean e.g. "2008-01-30"
- to: last date to clean e.g. "2008-01-31"
- datasource: folder in which the original data is stored
- datadestination: folder in which the cleaned data should be stored
- ticker: vector of tickers for which the data should be cleaned
- NOTE: In case trades (quotes) for a certain stock on a certain date are missing, the corresponding folder (see Figure 1) should contain a file "missing_ticker.RData" ("missingquotes_ticker.RData") to indicate that it can be skipped. If you have used the function "convert", this will automatically be the case.

4 Liquidity measurement

The functions presented in this section compute liquidity measures for each and every trade (for which a quote has been found and that has not been remove by cleaning). Thus, the output represents the "spot" liquitidity. In the next section, the tools to obtain aggregated liquidity measures will be presented.
A good review of liquidity measures and terminology can be found in "TAQ Project - Database User Guide" by Renaud Beaupain. The same terminology and function names are used here. We compute these liquidity measure before aggregating the data. In a second step the liquidity measures can be aggregated which is discussed in the next section.

The liquidity functions make make use of the following data input:

1. data: xts-object containing the joined trades and quotes
2. tdata: xts-object containing the trades
3. qdata: xts-object containing the quotes

4.1 gettradedir(data)

- Function returns a vector with the inferred trade direction which is determined using the Lee and Ready algorithym (Lee and Ready, 1991):
- data = xts object containing joined trades and quotes (e.g. using matchtq())
- NOTE: the value of the first (and second) should be ignored if price=midpoint for the first (second) observation.

```
gettradedir = function(data){
##Variable declaration:
  bid = as.numeric(data$BID);
  offer = as.numeric(data$OFFER);
  midpoints = (bid + offer)/2;
  price = as.numeric(data$PRICE);

  buy1 = price > midpoints; #definitely a buy
  equal = price == midpoints;

#Is it an up-tick?
  dif1 = c(TRUE,0 < price[2:length(price)]-price[1:(length(price)-1)]);
#Tick is equal to previous tick?
  equal1 = c(TRUE,0 == price[2:length(price)]-price[1:(length(price)-1)]);
  dif2 = c(TRUE,TRUE,0 < price[3:length(price)]-price[1:(length(price)-2)]);

  buy = buy1 | (dif1 & equal) | (equal1 & dif2 & equal);

  buy[buy==TRUE]=1;
  buy[buy==FALSE]=-1;

  return(buy);
}
```

Code snippet 2.
Implementation of the Lee and Ready (1991) algorithm to get the inferred trade direction.

4.2 liquidity(data,tdata,qdata)

- Function returns all liquidity measures mentioned below (from section 4.3 to 4.25) as xts object
- column names are the same as function names:
  e.g.: object = liquidity(data,tdata,qdata)
  object$es will give you the effective spread
  object$price_impact will give you the price impact
  etc.

4.3 es(data)

- Function returns the effective spread as xts object.

$$\text{Effective Spread}_t = 2 * D_t * (\text{PRICE}_t - \frac{(\text{BID}_t + \text{OFFER}_t)}{2}),$$

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

4.4 rs(data,tdata,qdata)

- Function returns the realized spread as an xts object.
- Please note that the returned object can contain less observations than the original "data" because of the need to find quotes that match the trades 5-min ahead.

$$\text{Realized Spread}_t = 2 * D_t * (\text{PRICE}_t - \frac{(\text{BID}_{t+300} + \text{OFFER}_{t+300})}{2}),$$

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

4.5 value_trade(data)

- Returns the trade value as xts object.

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

4.6 signed_value_trade(data)

- Returns the signed trade value as xts object.

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

4.7 signed_trade_size(data)

- Returns the signed size of the trade as xts object.

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

4.8 di_diff(data)

- Returns the depth imbalance (as a difference) as xts object.

$$\text{depth imbalace (as difference)}_t = \frac{D_t * (\text{OFFERSIZE}_t - \text{BIDSIZE}_t)}{(\text{OFFERSIZE}_t + \text{BIDSIZE}_t)},$$

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

4.9 di_div(data)

- Returns the depth imbalance (as a ratio) as xts object.

$$\text{depth imbalace (as ratio)}_t = (\frac{D_t * \text{OFFERSIZE}_t}{\text{BIDSIZE}_t})^{D_t},$$

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

4.10 pes(data)

- Returns the Proportional Effective Spread as xts object.

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

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

4.11 prs(data)

- Returns the Proportional Realized Spread as xts object.

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

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

4.12 price_impact(data)

- Returns the price impact as xts object.

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

(Boehmer, 2005; Bessembinder, 2003).

4.13 prop_price_impact(data)

- Returns the Proportional Price impact as xts object.

$$\text{proportional price impact}_t = \frac{100 * \frac{(\text{effective spread}_t - \text{realized spread}_t)}{2}}{\frac{\text{OFFER}_t + \text{BID}_t}{2}}$$

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

4.14 tspread(data)

- Returns the half traded spread as xts object.

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

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

4.15 pts(data)

- Returns the proportional half traded spread as xts object.

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

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

4.16 p_return_sqr(data)

- Returns the squared log return on Trade prices as xts object.

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

  Note: the first observation is set to zero.

4.17 p_return_abs(data)

- Returns the absolute log return on Trade prices as xts object.

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

  Note: return of first observation is set to zero.

4.18 qs(data)

- Returns the quoted spread as xts object.

  $$\text{quoted spread}_t = \text{OFFER}_t - \text{BID}_t.$$

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

4.19 pqs(data)

- Returns the proportional quoted spread as xts object.

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

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

4.20 logqs(data)

- Returns the logarithm of the quoted spread as xts object.

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

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

4.21 logsize(data)

- Returns the log quoted size as xts object.

$$\text{log quoted size}_t = \log(\text{OFFERSIZE}_t) - \log(\text{BIDSIZE}_t)$$

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

4.22 qslope(data)

- Returns the quoted slope as xts object.

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

(Hasbrouck and Seppi, 2001).

4.23 logqslope(data)

- Returns the log quoted slope as xts object.

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

4.24 mq_return_sqr(data)

- Returns midquote squared returns as xts object.

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

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

4.25 mq_return_abs(data)

- Returns absolute midquote returns slope as xts object.

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

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

5 Aggregation functions

5.1 General

5.1.1 aggregatets(ts, FUN=previoustick, on="minutes", k=1, weights=F)

- Returns aggregated time-series as xts object.
- Valid values for the argument "on" include: secs, seconds, mins, minutes,hours, days, weeks.
- Aggregates over on*k intervals using FUN on every interval. By default previous-tick aggregation is done, which means that the last observation in each interval is selected.
- Function can handle irregularly spaced Timeseries.
- The timestamps of the new time series are the closing times and/or days of the intervals. E.g. for a weekly aggregation the new timestamp is the last day in that particular week (namely sunday).
- By default weights=F and no weighting scheme is used. When you assign an xts object with wheights to the argument "weights", a weighted mean is taken over each interval. Of course, the weights should have the same timestamps as the supplied time-series.

```
aggregatets = function(ts, FUN=previoustick, on="minutes", k=1, weights=F){

  #Without weights:
  if(weights[1]==F){
  ep = endpoints(ts, on, k);
  ts2 = period.apply(ts,ep,FUN);
  }

  #With weights:
  if(weights[1]!=F){
  tsb = cbind(ts,weights);
  ep = endpoints(tsb, on, k);
  ts2 = period.apply(tsb,ep,FUN=weightedaverage);
  }


  if(on=="minutes"|on=="mins"|on=="secs"|on=="seconds"){
  if(on=="minutes"|on=="mins"){secs = k*60;}
  if(on=="secs"|on=="seconds"){secs = k}
  a = .index(ts2) + (secs-.index(ts2) %\% secs);
  ts3 = .xts(ts2,a);
  }

  if(on=="hours"){
  secs = 3600;
  a = .index(ts2) + (secs-.index(ts2) \%\% secs);
  ts3 = .xts(ts2,a);
  }


  if(on=="days"){
  secs = 24*3600;
  a = .index(ts2) + (secs-.index(ts2) \%\% secs) - (24*3600);
  ts3 = .xts(ts2,a);
  }


  if(on=="weeks") {
  secs = 24*3600*7;
  a = (.index(ts2) + (secs-(.index(ts2) + (3L * 86400L)) \%\% secs))-(24*3600);
  ts3 = .xts(ts2,a);
}

  #return to timeDate timestamps
  index(ts3) = as.timeDate(index(ts3));

  return(ts3);
}
```

Code snippet 3.
Very general aggregation function that can handle irregularly spaced time series and applies FUN on every time interval.

5.1.2 agg_price(ts,FUN = previoustick,on="minutes",k=5)

- Returns aggregated times-series as xts object.
- The first observation of new object is the opening price.
- Aggregates over on*k intervals using FUN on every interval.
- Valid arguments for on: see function "aggregate".
- The following observations are the closing prices over their respective intervals, and consequently the last observation is the closing price of that day if FUN=previoustick.

5.2 Trades and Quotes

5.2.1 agg_trades(tdata,on="minutes",k=5)

- Aggregates an entire trades xts object (tdata) over a on*k interval.
- Returned xts-object contains: SYMBOL,EX,PRICE,SIZE.
- Variables COND, CR, G127 are dropped because aggregating them makes no sense.
- The first observation is always included (to have the exact opening price).

5.2.2 agg_quotes(qdata,on="minutes",k=5)

- Aggregates an entire quotes xts object (qdata) object over a on*k interval.
- Returned xts-object contains: SYMBOL,EX,BID,BIDSIZE,OFFER,OFFERSIZE.
- Variable MODE is dropped because aggregation makes no sense.
- The first observation is always included (to have the exact opening quotes).

5.3 Liquidity

In Section "Liquidity Functions", several functions were presented to calculate the spot liquidity. We can use the aggregation functions presented to calculate "aggregated liquidity measures". Generally, there are two options:

1. The first option is to take an equally weighted mean over each subinterval. If you would like to get the effective spread aggregated over 30-min intervals for example:

   ```
   Effective_spread = es(data);
   agg_eff_spread = aggregatets(Effective_spread,k=30,FUN=mean);
   ```

2. The second and probably more interesting option is to take a weighted mean of the liquidity measure over a certain interval. Typically, the trade size is taken as weighting criterium. For example, to calculate the aggregated (30-minute) effective spread using this option:

   ```
   Effective_spread = es(data);
   agg_eff_spread =
   aggregatets(Effective_spread,weights = data$SIZE,k=30)
   ```

6 Daily volatility measurement

In this section we will present functions to calculate univariate and multivariate measures of volatility. Apart from the standard estimators (such as Realized

Volatility (RV) and Realized Covariation (RCov)), we focus on estimators of volatility that are robust to jumps in the price level. In that sense, this section is complementary to the R package “Realized” of Payseur (2008) which has functions to compute (non jump-robust) volatility measures.
The input for the multivariate volatility measures should be equispaced timeseries, with all timeseries aggregated to the same time grid obviously. Section 5 describes the tools to aggregate timeseries to equispaced intervals. In what follows, we denote by M the number of observations for a certain time period t (i.e. the number of “gridpoints”). The number of timeseries is represented by N.

6.1 Univariate measures

6.1.1 RV(returnseries)

- Returns the Realized Variance (RV).
- returnseries is a vector/zoo/xts object containing all returns in period t for one asset.
- Let $r_{t,i}$ be a return (with $i = 1, \dots, M$) in period t. Then the Realized Variance is given by the sum of the squared intraday returns

$$\mathrm{RV}_t = \sum_{i=1}^{M} r_{t,i}^2$$

6.1.2 ROWVar(returnseries, seasadjR = NULL, wfunction = ”HR” , alphaMCD = 0.5, alpha = 0.001)

- Returns the Realized Outlyingness Weighted Variance (ROWVar) (Boudt et al., 2008).
- returnseries is a vector/zoo/xts object containing all returns in period t for one asset.
- seasadjR is a matrix/zoo/xts object containing the seasonaly adjusted returns in period t for one asset. This is an optional argument.
- wfunction: Determines whether a Hard Rejection (”HR”) or Soft Rejection (”SR”) weight function is to be used. By default a Hard Rejection (wfunction = ”HR”) function is used.

- alphaMCD is a numeric parameter controlling the size of the subsets over which the determinant is minimized. Allowed values are between 0.5 and 1 and the default is 0.5. See Boudt et al. (2008) or "?covMcd" in the robustbase package.

- alpha is a parameter between 0 and 1, that determines the rejection threshold value (see Boudt et al. (2008) for details).

- Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t and $d_{t,i}$ a measure for the local outlyingness of that return, as defined in Boudt et al. (2008). The ROWVar is then given by

$$\text{ROWVar}_t = c_w \frac{\sum_{i=1}^{M} w(d_{t,i}) r_{t,i}^2}{\frac{1}{M}\sum_{i=1}^{M} w(d_{t,i})}.$$

6.1.3 RBPVar(returnseries)

- Returns the Realized BiPower Variation (RBPVar) (Barndorff-Nielsen and Shephard, 2004b).

- returnseries is a vector/zoo/xts object containing all returns in period t for one asset.

- Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t. Then, the RBPVar is given by

$$\text{RBPVar}_t = \frac{\pi}{2} \sum_{i=2}^{M} |r_{t,i}||r_{t,i-1}|$$

6.1.4 MinRV(returnseries)

- Returns the MinRV (Andersen et al., 2009).

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

- Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t. Then, the MinRV is given by

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

6.1.5 MedRV(returnseries)

- Returns the MedRV (Andersen et al., 2009).
- returnseries is a zoo/xts object containing all returns in period t for one asset.
- Let $r_{t,i}$ be a return (with $i = 1, \ldots, M$) in period t. Then, the MedRV is given by

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

6.2 Multivariate measures

6.2.1 RCov(Nreturnseries)

- Returns the Realized Covariation (RCov).
- Nreturnseries is a $(M \times N)$ matrix/zoo/xts object containing the N return series over period t.
- Let $r_{t,i}$ be an intraday (Nx1) return vector and $i = 1, ..., M$ the number of intraday returns. Then the RCov is given by

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

6.2.2 ROWCov(Nreturnseries, seasadjR = NULL, wfunction = "HR" , alphaMCD = 0.5, alpha = 0.001)

- Returns the Realized Outlyingness Weighted Covariation (ROWCov) (Boudt et al., 2008).
- Nreturnseries is a $(M \times N)$ matrix/zoo/xts object containing the N return series over period t.
- seasadjR is a $(M \times N)$ matrix/zoo/xts object containing the seasonaly adjusted returns. This is an optional argument.
- wfunction: Determines whether a Hard Rejection ("HR") or Soft Rejection ("SR") weight function is to be used. By default a Hard Rejection (wfunction = "HR") function is used.

- alphaMCD is a numeric parameter controlling the size of the subsets over which the determinant is minimized. Allowed values are between 0.5 and 1 and the default is 0.75. See Boudt et al. (2008) or "?covMcd" in the robustbase package.

- alpha is a parameter between 0 and 1, that determines the rejection threshold value (see Boudt et al. (2008) for details).

- Let $r_{t,i}$, for $i = 1, \dots, M$ be a sample of M high-frequency (Nx1) return vectors and $d_{t,i}$ their outlyingness given by the squared Mahalanobis distance between the return vector and zero in terms of the reweighted MCD covariance estimate based on these returns. The Realized Outlyingness Weighted Covariation is defined as

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

  where $w(\cdot)$ is a hard rejection weight function and c_w the corresponding correction factor assuming the return follow a normal distribution , as described in Boudt et al. (2008).

6.2.3 RBPCov(Nreturnseries)

- Returns the Realized BiPower Covariation (RBPCov) (Barndorff-Nielsen and Shephard, 2004a).

- Nreturnseries is a $(M \times N)$ matrix/zoo/xts object containing the N return series over period t.

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

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

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

6.2.4 tresholdcov(Nreturnseries)

- Returns the treshold covariance matrix proposed in Gobbi and Mancini (2009).

- Nreturnseries is a $(M \times N)$ matrix/zoo/xts object containing the N return series over period t.
- The treshold value TR_M is taken as suggested in Jacod and Todorov (2009).

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

6.2.5 RCor(Nreturnseries)

- Returns the Realized Correlation (RCor).
- Nreturnseries is a $(M \times N)$ matrix/zoo/xts object containing the N return series over period t.
- The (k,q)th element of the daily Realized correlation matrix on day t is defined as

$$\text{RCor}[k,q]_t = \frac{\sum_{i=1}^{n} r_{(k)t,i} * r_{(q)t,i}}{\sqrt{\sum_{i=1}^{n} r^2_{(k)t,i} \sum_{i=1}^{n} r^2_{(q)t,i}}}$$

  where $r_{(k)t,i}$ is the k-th component of the return vector $r_{t,i}$ and n the number of observations per day.

7 Jump detection

In this section we will present functions to test the presence of a jump component in the daily volatility and in the intraday return series. However, this is work in progress.

References

Andersen, T. G., D. Dobrev, and E. Schaumburg (2009, November). Jump-robust volatility estimation using nearest neighbor truncation. *working paper* (15533).

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

Barndorff-Nielsen, O. and N. Shephard (2004b). Power and bipower variation with stochastic volatility and jumps. *Journal of Financial Econometrics 2*(1), 1–37.

Barndorff-Nielsen, O. E., P. R. Hansen, A. Lunde, and N. Shephard (2008). Realised kernels in practice: Trades and quotes. *Econometrics Journal 4*, 1–32.

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

Bingcheng, Y. and E. Zivot (2003). Analysis of high-frequency financial data with S-Plus. *UWEC-2005-03*.

Boehmer, E. (2005). Dimensions of execution quality: Recent evidence for us equity markets. *Journal of Financial Economics 78 (3)*, 553–582.

Boudt, K., C. Croux, and S. Laurent (2008). Outlyingness weighted quadratic covariation. *Mimeo*.

Gobbi, F. and C. Mancini (2009, October). Identifying the covariation between the diffusion parts and the co-jumps given discrete observations. *working paper* (math/0610621).

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

Jacod, J. and V. Todorov (2009). Testing for common arrival of jumps in discretely-observed multidimensional processes. *Annals of Statistics 37*, 1792–1838.

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

Payseur, S. (2008). *realized: Realized.* R package version 0.81.

Ryan, J. A. and J. M. Ulrich (2009). *xts: Extensible Time Series.* R package version 0.6-7.

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. and Y. Chalabi (2009). *timeDate: Rmetrics - Chronological and Calendarical Objects.* R package version 2100.86.

A Extracting TAQ data from DVD's

In this section we discuss a practical method to extract historical TAQ data in case it has been delivered on DVD's. Other delivery methods are neglected for now.

The obvious first step is to get the data on hard disk. A storage capacity of roughly 400 GB per year of data for e.g. the S&P100 stocks is required. We store the zipped data in a folder structure with each folder indicating the date of one trading day. This requires a simple tool *(file: folders_dates.R)*, because the names of zipfiles on DVD do not directly indicate for which trading days the file contains data.[7]

The amount of data that needs to be unzipped is enormous. Therefore, we propose to use a "client server architecture". After unzipping the raw data of one trading day on a certain client computer, the needed data is extracted using the TAQ3 software and saved back on the server. We opt not to keep the unzipped raw data stored, since the required storage capacity would be very high. The entire process is automatically performed by the code in the *File: "unzipper.R"*. This file makes use of the files: *"getsize.cmd"* and *"FileSize.bat"* to check for empty extracted files, *"tickerlist.txt"* contains the list of tickers of the stocks to be extracted and *"basefile.txt"* consists of a standard jobfile for the TAQ3 software. Since the unzip function of R 2.9.1 on a standard computer is not suited to unzip files larger than 2 GB, we opt for the "command line version" of 7-zip to do the job. For each trading day the following operations are performed chronologically:

1. Unzips the raw data of a certain day (stored on client);
2. Creates jobfiles for TAQ3 Selection based on exchange software (stored on client);
3. Pushes jobfile for each stock's ticker in the vector "ticker" to the TAQ3 software which saves the extracted data to the external HD;
4. Deletes the jobfiles;
5. Deletes the unzipped raw data.

This file is executed on each client-pc for a certain number of days. On a standard computer (windows vista, 3.16 Ghz processor and 4GB RAM) this process takes up to 50 hours per month of data for recent periods.

[7]The name structure of these zipfiles is as follows: CDA.zip for the first day of the month, CDB.zip for the second and so on.

```
extract = function(from="2008-01-01",to="2009-05-31",ticker=ticker,trades=TRUE,quotes=TRUE,
zipdir = "R:\\rawdata\\",exdir="R:\\extracteddata\\", tempdir= "c:\\rawdata\\localdrive\\",
taqfolder="C:\\TAQWIN32\\"){

library("timeDate")
dates = timeSequence(from,to, format = "%Y-%m-%d", FinCenter = "GMT")
dates = dates[isBizday(dates, holidays = holidayNYSE(2004:2010))];

for(qq in 1:length(dates)){
##Here starts a loop over the requested time frame.
##Every day in this frame is the "currentdate" at some point.
currentdate = as.character(dates[qq]);

setwd(taqfolder);
## 1. To unzip a file for the "currentday"
#To unzip a file for the "currentday"
alphabet = c("A","B","C","D","E",...,"S","T","U","V","W","X","Y","Z");
zipnames = paste("CD",alphabet,".zip",sep="");
unzip_locations = paste(zipdir,currentdate,"\\",zipnames,sep="")
condition = file.exists(unzip_locations);
unzip_location = unzip_locations[condition];
chosencharacter = alphabet[condition];
ex_location = tempdir;

command = paste("\"c:\\Program Files\\7Zip\\7z.exe\" e -o",tempdir," \"",unzip_location,"\"",sep="")

system(command, intern = FALSE, ignore.stderr = TRUE,
       wait = TRUE, input = NULL, show.output.on.console = TRUE,
       minimized = FALSE, invisible = FALSE);

##2. Create jobs for the taq3 software:
currentdate_taq = paste(as.character(as.real(unlist(strsplit(as.character(currentdate),"-"))[2])),
as.character(as.real(unlist(strsplit(as.character(currentdate),"-"))[3])),
unlist(strsplit(as.character(currentdate),"-"))[1],sep="/");

## Each of these jobfiles will contain the commands
## for the taq3 software to extract data
## for exactly 1 stock and 1 day ("currentday")

setwd(paste(taqfolder,"JOBS",sep=""));
jobfile = read.table("basefile.txt", sep="\n");
jobfile = as.character(as.vector(jobfile[,1]));
jobfile[30] = paste("START DATE=",currentdate_taq,sep="");
jobfile[31] = paste("END DATE=",currentdate_taq,sep="");

#Create directory for data to be saved into:
dirname = paste(exdir,currentdate,sep="")
dir.create(dirname)

if(trade){
##3. Execute the taq jobfiles for all stocks in ticker:
##Start trades extraction
for(i in 1:length(ticker))  {

  jobfile[2] = paste("JOB DESCRIPTION= extraction_of_",ticker[i],sep="");
  jobfile[4] = paste("QUOTES FILE NAME=",dirname,"\\",ticker[i],"_quotes.txt",sep="");
  jobfile[5] = paste("TRADES FILE NAME=",dirname,"\\",ticker[i],"_trades.txt",sep="");
  jobfile[34] = paste("SYMBOL/CUSIP LIST=",ticker[i],sep="");
  jobfile1 = as.factor(jobfile)

  jobfile_name = paste(ticker[i],"_jobfile.tjf",sep="");
  write.table(jobfile1, file = jobfile_name, append = FALSE, quote = FALSE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = FALSE,
            col.names = FALSE);

command = paste(taqfolder,"taq3.exe ",taqfolder,"JOBS\\",ticker[i],"_jobfile.tjf",sep="")
system(command, intern = FALSE, ignore.stderr = TRUE,
       wait = TRUE, input = NULL, show.output.on.console = TRUE,
       minimized = FALSE, invisible = FALSE);

##4.Delete job file after execution of job:
unlink(paste(ticker[i],"_jobfile.tjf",sep=""), recursive = FALSE)
                            }

}
##5. Delete the unzipped-files' folder:
unlink(ex_location, recursive = TRUE);
}
}
```

Code snippet 4.
The above code illustrates the part of the function "extract" for the extraction of trades for a certain timeframe ("from" until "to") and all stocks in the vector "ticker". The logical arguments "trades" and "quotes" determine whether or not to extract trades and/or quotes. The raw zipfiles should be stored in "zipdir", the raw unzipped data will temporarily be stored in "tempdir" and the extracted data in "exdir". The TAQ3 software and other helpfiles should be stored in "taqfolder".