

Nonparametric Volatility Estimation: A realized variance implementation in R and VBA

22S:166 Fall 2007 - Term Project

David Schreindorfer Joe Bawazer

Abstract

This work can be considered a non-technical introductory to realized variance. After an overview of the role of volatility in finance, we explain how realized variance (RV) gained in popularity through its use in the evaluation of GARCH models. The paper then proceeds to a brief introduction to the theoretical results in stochastic processes underlying the quantity of interest. Lastly, we implement a simple RV calculation in R as well as in VBA, for which, to our best knowledge, no packages are available thus far.

1 Introduction

This paper was written as part of the requirements of a class in Statistical Computing and we therefore consider the target audience to consist mainly of statisticians. Because the importance of the realized variance (RV) statistic is based on its applications in finance, however, we start with a relatively detailed section explaining the important role of volatility in finance. Furthermore, we outline how RV helped to "save the GARCH model" and how it can be used to improve the choice of order of this Nobel price-winning model. This in turn leads to improved volatility forecasts.

1.1 Motivation

Much of finance is concerned with the variability of possible payoffs of an investment, for example in the stock of a company. These payoffs are usually described by returns, which is a term for the relative (percentage) change of the price of an investment. If we let S_t denote the stock price at time t , a return over the period from $t - 1$ to t would be calculated as $\ln(S_t/S_{t-1}) = \ln(S_t) - \ln(S_{t-1})$. This calculation assumes continuous compounding, as opposed to annual or semiannual compounding which are probably more familiar to people who took out a loan or a mortgage before. The higher the variance of these future returns, the higher the range of possible outcomes in terms of the investor's total profit or loss. Economic theory suggest that investors like to avoid this kind of risk (they are 'risk averse'). In other words, they prefer steady returns without big time-to-time losses and are willing to accept lower expected returns for investments whose expected

volatility is lower. Therefore, they are naturally interested in forecasting this risk, which is most typically measured by variance, of their investment over future periods.

The immense importance of variance forecasts for many financial applications explains why Robert Engle, who first found a time series model to successfully predict future variances, was awarded the Nobel price in economics in 2003. Engle came up with a way of predicting variances of future periods based upon variances observed in the recent past. The most common use of his autoregressive conditional heteroskedasticity (ARCH) model is to forecast variances of daily stock returns, i.e. to produce a variance forecast for tomorrow (and the days thereafter) based on the return variance observed over the last couple of days. For a long time, finance researchers had empirically documented that stock return variances are heteroscedastic, i.e. that returns go through volatile and less volatile periods. Engles' way of using only information from the recent past to make predictions enables his model to correctly reflect this heteroscedastic behavior. A voluminous literature on ARCH and GARCH (G for generalized) models has emerged since Engles's original 1982 paper, producing a vast number of very sophisticated forecasting methods. Since the main purpose of this project is not to implement GARCH models, we do not go into technical details about these models at this point and refer the interested reader to the time series literature. Good reference texts in this area include Hamilton (1994) and Tsay (2005).

In contrast to 'classical' statistical models, a model whose purpose is primarily to produce an accurate out-of-sample forecast, should not be judged with respect to the significance of its parameters or its in-sample explanatory power (R^2). Rather, it should be evaluated out-of-sample, i.e. its predictions should be evaluated against future realizations to calculate an 'out-of-sample R^2 '. Unfortunately, the variance of a stock return is a latent variable, which means that it is not observable. To escape this misery, researchers have used proxies instead of the unobservable variance for their out-of-sample evaluations. Since the expected daily return for a stock is very close to zero, a natural proxy for the return variance is the squared return (since $Var[r] = E[r^2] - E[r]^2$). This evaluation method nearly led to the conclusion that GARCH models are useless for any practical application. People found that the predictive power of GARCH models when measured against the very noisy squared return proxy (expected returns are close to zero but actual returns are not) is almost always below 10%, which means that the forecasts are useless. This is the sad conclusion many authors drew after conducting surveys of the ARCH/GARCH literature.

Toward the end of the 1990s, intra-daily financial data (see section ??) became available. Instead of recording only the stock's closing price at a particular trading day, researchers could now look at all the price movements within a day, often tens of thousands of observations within any one day. In 1998, T. Andersen and T. Bollerslev (a former PhD student of Engle who generalized his ARCH model) published a paper entitled 'Answering the Skeptics: Yes, Standard Volatility Models do provide accurate Forecasts'. Using this new type of data, a new measure based on the statistical theory on quadratic variation was proposed that could be used instead of the usual squared return proxy. This measure, commonly referred to as realized variance, overcomes the noise inherent in squared returns by using much more data to produce an estimate. More precisely, it uses thousands of observations

instead of just one!. Using RV in out-of-sample evaluations, it has been shown that lower orders GARCH models often perform significantly better than higher order models out of sample. The leading article in this area is Hansen and Lunde (2005).

While RV does, in theory, produce unbiased estimates of the true return variance, many practical issues, usually summarized by the term 'Market Microstructure Effects', related to the way stocks are traded in practice, introduce a bias. This fact, which mainly manifests itself in the form of strong negative serial correlation in the data, has been shown in multiple papers. A partial solution to this problem is to sample infrequently, i.e. to use only parts of the data. While there are many different ways of implementing this idea, we chose the method of first converting the series with unequally spaced observation times (the 'heterogeneous series') into one with equal spacing between the data points (a 'homogeneous series') and then skipping parts of the observations to create our used sample. We implement the computation of RV in R and VBA (the common language in the field is MatLab) and show its bias for high sampling frequencies.

1.2 Outline

The paper proceeds as follows. We start with a description of the continuous time setup that is typically assumed in the context of realized variance estimation in section ???. Section ??? describes the used data and points out difficulties that may arise in its use. In section ???, we explain how the computations were implemented both in R and in VBA. Here, we also explain some of the functions that helped to overcome the critical issues in the implementation in the respective languages. The text proceeds to the results (???) and concludes with suggestions for further work (??). All code can be found in the appendix.

2 Continuous time setup

In this part, we describe the theoretical framework we are operating in. Since the focus of this paper is the computational implementation, however, the section is rather brief and does not establish a complete treatment of the results. The material, based on results in stochastic processes (this subfield is usually termed 'Stochastic calculus'), is relatively technical and a reader who is more interested in the implementations may safely skip to section ???. A very good theoretical introduction to stochastic calculus is Shreve (2004).

We let S_t denote the efficient price process of the stock, which can be thought of as the 'true', but unobservable price (note: the notation was used differently above). It is then reasonable to assume an Itô process for the log-price $X_t = \ln S_t$, which provides a relatively flexible framework. This process takes the form

$$dX_t = \mu_t dt + \sigma_t dW_t \tag{1}$$

where W_t is a standard Brownian motion. The drift process μ_t and the instantaneous variance process σ_t are adapted stochastic processes, i.e. they are both measurable with respect to the information set at time t. The difference in log-prices dX_t equals the return

of the efficient price process at time t , i.e. the process of interest. Unfortunately, we do not observe X_t , but rather the log transaction price

$$Y_{t_i} = X_{t_i} + \epsilon_{t_i} \quad (2)$$

which includes the observation error ϵ . Here, the t_i 's denote the observation times from the actual sample. We would like to estimate the continuous quadratic variation of the efficient price process, namely

$$\langle X, X \rangle_T = \int_0^T \sigma_t^2 dt. \quad (3)$$

Could one observe the efficient price process, one could compute the realized variance (also called the observed quadratic variation) as $\sum_{t_i} (X_{t_{i+1}} - X_{t_i})^2$, which converges to the sought quantity in probability. Because we have huge amounts of data available (see section ??), we would therefore have a very precise estimator of the continuous quadratic variation. Since the X_t process is unobservable, however, one is stuck with having to find an estimator that uses the observed log transaction price process Y_t . Zhang et al. (2005) show that if one computes an estimate of $\langle X, X \rangle_T$ by simply using the returns computed from all observed log transaction prices, one ends up with

$$[Y, Y]_T^{(all-obs.)} = \sum_{t_i, t_{i+1} \in [0, T]} (Y_{t_{i+1}} - Y_{t_i})^2 = 2nE[\epsilon^2] + \mathbf{O}(n^{1/2}) \quad (4)$$

and

$$E([Y, Y]_T^{(all-obs.)} | X_{process}) = [X, X]_T^{(all-obs.)} + 2nE[\epsilon^2] \quad (5)$$

where n is the number of sampling intervals over $[0, T]$ and \mathbf{O} is the 'big O' notation from computational complexity theory. In words, for a large n , one ends up with something related to the variance of the noise as opposed to the true integrated volatility asymptotically. The understanding here is that RV is calculated from an equally spaced series. The results are confirmed by our empirical results below. For a very high sampling frequency, RV becomes more and more affected by the bias caused by the noise.

3 The Data

The Ticks and Quotes (TAQ) database of the New York Stock Exchange (NYSE) contains both transaction and quote data (see explanation below) for all stocks traded on the exchange. The NYSE is the largest stock exchange in the world by dollar volume and, with 2,764 listed securities, has the second most securities of all stock exchanges. While the TAQ is a fee-based database, we obtain access to it through the Wharton Research Data Services (WRDS) subscription of the finance department at the University of Iowa. We gather 6 month (July 2005 through December 2005) of intra-daily transaction data for International Business Machines Corp. (IBM). This type of data, which is also commonly referred to as 'high frequency data' or 'tick data', describes how many shares exchange hands ('volume') at what price and at what time on the NYSE. Quotes data, on the other

hand, would describe all bid and ask prices quoted by dealers on the exchange, that is, the prices they are willing to conduct transactions at. Table ?? shows a sample of our data for the first 5 observations in October. For simplicity, we ignore the volume variable in our analysis.

SYMBOL	DATE	TIME	PRICE	CORR
IBM	20051003	9:30:07	80.22	0
IBM	20051003	9:30:07	80.20	0
IBM	20051003	9:30:07	80.22	0
IBM	20051003	9:30:07	80.22	0
IBM	20051003	9:30:07	80.22	0

Table 1: A VERY small subset of the data

Several characteristics immediately point to the difficulties associated with the data: (1) While the exchange opens at 9:30am and closes at 4.00pm, trading does not take place at every possible point in time. This means that almost none of the tools of 'classical time series analysis' can be employed to the data since these all build upon the notion of the 'back shift operator'. In other words, methods in time series analysis were typically developed for equally spaced data. (2) It frequently happens that multiple trades at various prices are recorded for one time point. In the above data sample for example, it does not seem clear what one should use as the 9:30:07 price observation. (3) The column labeled 'CORR' contains a correction factor for different kinds of miss-recorded observations. The question arises of whether one can just disregard these 'outliers'. (4) The time associated with each observation consists of two parts, a date and a daytime, both stored as character strings. These need to be combined somehow to clearly differentiate between multiple days if one wishes to work with longer time periods.

Month	# Observations	# nonzero CORR
July	187345	91
August	163432	75
September	175850	76
October	197692	70
November	159660	101
December	184491	86
Total	1068470	499

Table 2: Number of Observations

As observable from Table ??, another challenge of high-frequency data is their (surprise surprise) high frequency. The arbitrary 6 month period under investigation contains over 1 million observations, which makes it very difficult to successfully detect any bugs in the computer code by just 'looking at the results'.

4 Implementation

The computations are done in both R and VBA. Since mistakes are hard to discover given the huge amount of data, we consider this a good way of ensuring accuracy. Also, our analysis shows the relative efficiency of the two languages of dealing with these computationally intensive tasks. While the complete programs can be found in the appendix, the following sections describe the process of obtaining our final estimate and an explanation of how some difficulties were overcome, in cases where we feel like an elaboration on the documentation found in the code is helpful.

4.1 Data cleaning

We decide to disregard the observations corresponding to nonzero correction factors, i.e. those observations that were miss-recorded for various reasons, because they mainly correspond to large 'spikes' in the prices. Therefore, they don't appear to be reasonable observations at the respective times, i.e. we don't believe that they reflect the underlying efficient price process well. Further, this step seems justifiable in our eyes since the ignored observations represent less than 0.05% of the total data.

4.1.1 Coding in VBA

The VBA program is setup to read in .txt. files with lines separated by the Linefeed Character. (In ASCII this corresponds to Chr(10).) The this line is inputted into an array. Then, using the Linefeed Character as a delimiter, each individual line is read into another array. The number of indices in this array is equal to the number of individual lines in the original .txt file. Essentially each line is one of three types: a Title line, a Data line, or a Blank line. To further complicate the matter, Title lines would show up sporadically in between Data lines. So the next step was to loop through the array of individual lines and delete those that were not needed. These amounted to Blank lines, Title lines, and Data lines with a nonzero correction factor. Blank lines could be easily detected as a Null string. Title lines could also be detected as the first character is always "S." The correction factors were a little trickier. If the correction factor was zero then the 47th and 48th characters of each data line would be '0'. The macro would look for these characteristics in a line and if any of the three were present the line would not be reread into the array. The end result is a compressed array containing only the data we wish to analyze.

4.2 Creating a Homogeneous Series

Since our first goal in this project is to create a homogeneous series with one observation per second, we proceed to tackle the problem of multiple observations at one time point that occurs very frequently in the data. While one could produce a weighted average that takes the different trading volumes into consideration, we decide to avoid this approach to make the algorithm more efficient, and compute simple arithmetic averages instead. An advantage of using the weighted average would have been to reduce the magnitude of irrational trades in small quantities (which would receive small weights) that might be caused

by uninformed trading or in an attempt by some large financial institutions to add noise to the data. To obtain price observations for the unobserved times, we interpolate linearly between neighboring observations. While this sounds like an easy step, the code turned out to be rather involved (see appendix), containing numerous for-loops as well as extensive use of sub-indices. Another conflict arises with respect to the suggested underlying process. While the Ito-process for the efficient price is a martingale, we technically use information we don't possess yet at a particular point in time when doing the linear interpolation. Nevertheless, we decide to proceed with this method as it is the one most widely applied in the literature. The only feasible alternative of using the previously observed price for the 'empty times' leads to the problem of large price jumps for periods with few observations. We avoid a detailed explanation of the associated problems at this point and refer the interested reader to Dacorogna et al. (2001). The last step to obtain a homogeneous series with one-second intervals is to fill-in prices for times before the first price observation and after the last price observation each day. We chose to replicate the respective first and last price observation of the trading day for these times. This avoids complications that arise from price jumps between trading days when using price observations from neighboring days to do a linear interpolation. These jumps would lead to a higher volatility estimate for each day. However, our objective is to measure the volatility for a particular day, and this does not include volatility caused by jumps between days.

4.2.1 Coding in R

To simplify the identification of unique trading times (consisting of a date and a time), we make use of the *as.POSIXct()* function in R, which creates a 'time-object' consisting of data, time, and time-zone. While this makes the analysis simpler at this point, it later creates the problem that R shifts the times by one hour when the change from daylight savings time occurs (in October). For the observations that are affected by this, we add 3600 seconds (one hour) back to achieve the correct trading hours. The time object is also crucial for the next step of 'filling in the blanks'.

4.2.2 Coding in VBA

The process in VBA makes no use of a time object. Each line is read at the position corresponding to the Date and Time. Each Date and Time value is inspected to see if it is unique or if there are other Date and Time pairs of equal value. It should be clarified here that VBA reads these times as a string. So if the next line in the array has the same string value at this position then it is grouped as having an equal time. The prices for unique times are set equal to that unique price value. The price of a time with more than one observation is computed as the arithmetic average of the prices for that time. The details of the averaging in VBA was somewhat complicated (see code), but it was consistent with those values obtained from the calculations in R.

4.3 Choosing a frequency and computing Realized Variance

The last step in our programs is to choose a sampling frequency and to compute the RV statistic for the resulting series. We chose intervals of 1, 2, 3, 4, 5, 10, 20, 30, 60, 120, 180, 240, 300, 600, and 1800 seconds. Each one of these estimates is supposed to estimate the same quantity, the continuous quadratic variation of the process. Without what we termed 'market microstructure effects', we would expect more observations to lead to a more precise estimate, i.e. to a reduced variance and an estimate that converges to the sought quantity at a higher sampling frequency. However, our results show that the estimate increases sharply for the highest frequencies, impressively showing the pronounced bias caused by the increased noise. An example of the aforementioned microstructure noise is the so called 'bid-ask bounce', which describes the frequent up and down movements of the transaction price between the bid and the ask price, depending on which kind of transaction is being conducted. Several models have been proposed to show how this effect introduces negative serial correlation to the data.

4.3.1 Coding in VBA

The realized variance calculations in VBA are done in what one may call a 'brute force' method. Each day's stock prices are used to compute the realized variance based on the specific time lag for the return calculation. The major downside of these calculation in this VBA program is that it is hard coded to compute variances for only the 1, 2, 3, 4, 5, 10, 20, 30, 60, 120, 180, 240, 300, 600, and 1800 second intervals. Thus if a user would like to implement a different sized interval it would require them to go into the code and make the appropriate changes. However, it would be possible to set up a user interface where he or she enters the time interval to be computed. This would also compress the code into a less drawn out procedure.

5 Results

Consistent with the results predicted by theory, the realized variance statistic shows a higher bias for high sampling frequencies. The volatility of returns for time lags between one and ten seconds are very high compared with those of longer lags. After about ten seconds, the variance steeply declines in for these high-frequency calculations until they settled into more consistent estimates from a lag of about five minutes and above.

The choice of R as a computational device was due to its' ease of use in Statistical computation. The entire program comprises roughly three pages of code. This contrasts sharply with the VBA code which spans over 20 pages in length. The terse coding also enables a user with R experience to make adjustments to the program rather easily. The major disadvantage of using R was the run-time of the program. Perform the calculations on a one month data set takes approximately one hour.

The choice of Visual Basic in Excel is due to it's extensive use in Business and Finance, as well as the easy of use that comes with it's graphical interface options. VBA comes with a

convenient "step through" option which eases the task of debugging. Also this allows the user to watch the program perform each action of the macro. The end result is an Excel Workbook that allows the user to carry out realized variance calculation through the use of buttons. Hence, no knowledge of the aforementioned processes are needed in order to obtain the estimates of quadratic variation. Since these calculations are hard coded over numerous pages, a user would need to spend much more time customize the application to suit his or her needs. An unexpected benefit of using VBA was the speed with which the calculations were carried out for a given month. The most computationally intensive procedures of data cleaning and creating a homogeneous time series took about 7.5 minutes on a computer with a 1.86 GHz Pentium processor. On a machine with a Dual 2.20 GHz Pentium processor this run time was reduced to about 2.5 minutes.

The following graph shows the estimates for all considered sampling frequencies. To get a smoother picture that avoids some of the noise of shorter time periods, the plotted RVs are averages over the 6 month from June 2005 to December 2005.

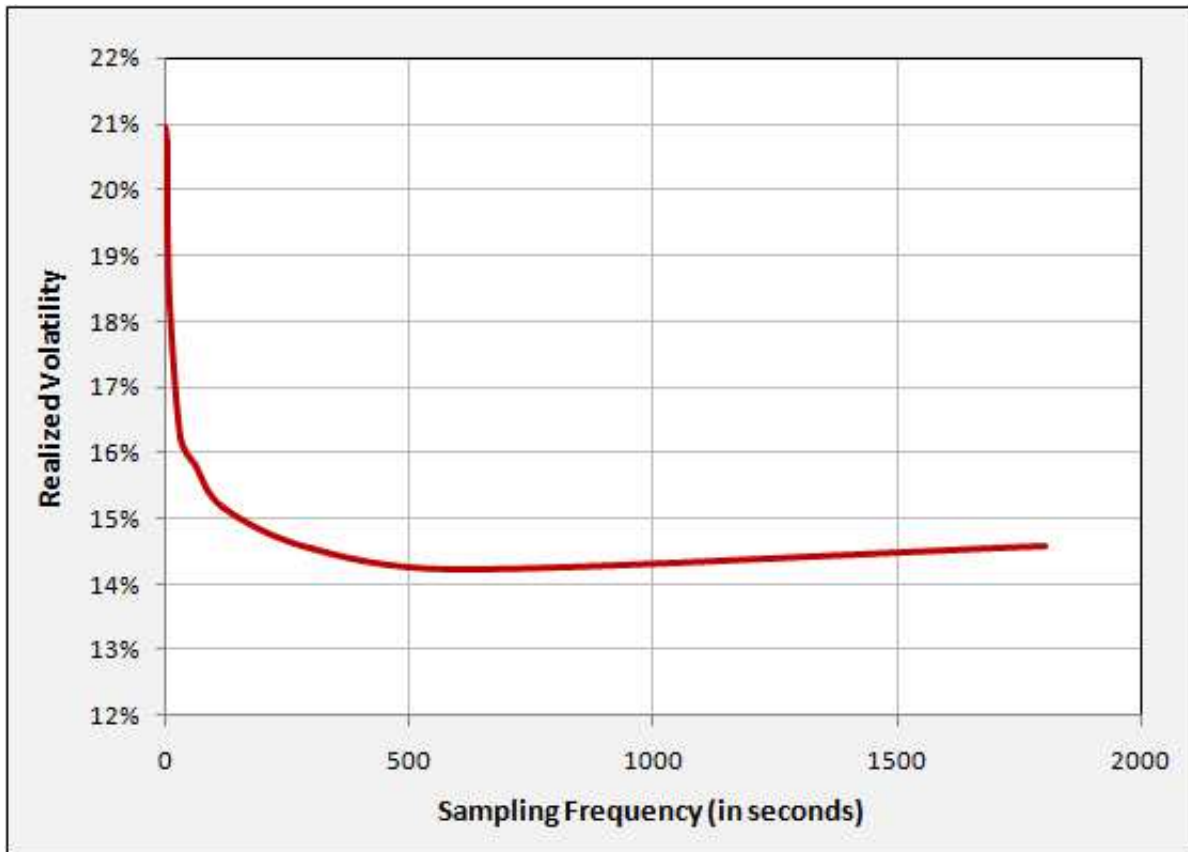


Figure 1: Annualized RV vs. Sampling Frequency, averaged over 6 month period

6 Conclusion and suggestions for further work

While no new theoretical results were proposed in this project, we showed how to handle the relatively involved computations associated with RV in R and in VBA. While R is an environment preferred by many statisticians, our results show that EXCEL is much better equipped to handle the long loops in the code, resulting in a much shorter run-time. While the VBA program results in an easy-to-use interface that can be employed by anyone wanting to estimate RV without cranking through the details of the code, the R code might be considered more elegant by statisticians who would like to understand the underlying calculations.

It should be emphasized that this paper has not given any suggestions on how to optimally sample to minimize the microstructure bias, a problem that has been solved in the literature already. Furthermore, Zhang et al. (2005), among others, have proposed methods to make use of all of the data while sampling at 'sparse' frequencies. Also, numerous bias-reduction methods have been proposed in the literature. Therefore, further work could use the code provided here and extend it to sample optimally and correct for the remaining bias to produce a more comprehensive set of functions for RV calculations. We consider continuing this work by incorporating extensions of this kind.

References

- [1] Andersen, T.G., and Bollerslev, T. (1998). Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International Economic Review*, vol 39 issue 4, pages 885-905.
- [2] Dacorogna, M.M., Gençay, R., Müller, U., Olsen, R.B., and Pictet, O.V. (2001). An introduction to high-frequency finance. *Academic Press*.
- [3] Hamilton, J. (1994). Time Series Analysis. *Princeton: Princeton University Press*.
- [4] Hansen, P.R., and Lunde, A. (2005). A forecast comparison of volatility models: does anything beat a GARCH(1,1)?. *Journal of Applied Econometrics*, vol 20 issue 7, pages 873-889.
- [5] Shreve, S.E. (2004). Stochastic Calculus for Finance II. *Springer*.
- [6] Tsay ,R.S. (2005). Analysis of Financial Time Series. *John Wiley & Sons, Inc.*
- [7] Zhang, L., Mykland, P.A., and Ait-Sahalia, Y. (2005). A tale of two time scales: Determining integrated volatility with noisy high-frequency data. *Journal of the American Statistical Association*, vol 100 issue 472, pages 1394-1411.

A R code

```
### -----
### (I) CREATE HOMOGENEOUS SERIES
### -----

### READ IN & CLEAN UP DOWNLOADED DATA (CREATE HETEROGENEOUS SERIES)
# read in data and remove observations with nonzero correction factors
data <- read.csv("ibm.csv",header=TRUE)
data.clean <- data[data$CORR==0,c(DATE,TIME,PRICE)]
# create vector of observed dates in the YYYY/MM/DD format (same length as cleaned data)
DATE <- as.matrix(paste(substr(data.clean$DATE,1,4),"/",substr(data.clean$DATE,5,6),"/",substr(data.clean$DATE,7,8),sep=""))
# create vector of dates and times
TIMES <- as.POSIXct(paste(DATE,data.clean$TIME,sep= ),tz="EST5EDT")
TIMES[as.POSIXlt(TIMES)$isdst==0] <- TIMES[as.POSIXlt(TIMES)$isdst==0]+3600
# create data frame with date/time in column one and price in column two
HETERO <- data.frame(cbind(rep(NA,length(TIMES)),rep(NA,length(TIMES))))
HETERO[,1] <- TIMES
HETERO[,2] <- data.clean$PRICE
names(HETERO) <- c("TIME","PRICE")

### CREATE HETEROGENEOUS SERIES WITH ONE PRICE PER TIME POINT IN SAMPLE (by averaging any multiple values)
# set up matrix with unique observed times in column 1 and NAs in column 2
SINGLE.TIMES <- data.frame(cbind(rep(NA,length(unique(TIMES))),rep(NA,length(unique(TIMES)))))
SINGLE.TIMES[,1] <- as.POSIXct(paste(unique(TIMES)+3600,tz="EST5EDT") #add 3600 because unique() changes time zone
names(SINGLE.TIMES) <- c("TIME","PRICE")
# create vector with indices of observed times that are unique (first observation for any observed time)
index_A <- c(1,which(diff(TIMES)!=0)+1,length(TIMES))
# fill in average prices for observed times
SINGLE.TIMES[length(unique(TIMES)),2] <- mean(HETERO$PRICE[index_A[length(index_A)]:length(TIMES)])
for(i in 1:(length(index_A)-1)){SINGLE.TIMES[i,2] <- mean(HETERO$PRICE[index_A[i]:(index_A[i+1]-1)])}

### SET UP MATRIX FOR HOMOGENEOUS SERIES
# vector of 1-second time intervals for one day (created in EXCEL)
sec <- as.matrix(as.character(read.csv("times.csv",header=TRUE)$SEC))
# number of trading days in observed month
days = length(unique(DATE))
# create vector of dates/times for observed month
unique.dates <- unique(DATE)
dates = matrix(nrow=(23401*days),ncol=1)
for(i in 1:days){dates[((i-1)*23401+1):(i*23401)] <- rep(unique.dates[i],23401)}
times = as.matrix(rep(sec,days))
TIME = as.POSIXct(paste(as.character(dates),as.character(times)),tz="EST5EDT")
TIME[as.POSIXlt(TIME)$isdst==0] <- TIME[as.POSIXlt(TIME)$isdst==0]+3600
# create matrix with times (observed days and a second intervals) and empty price column
TIME = as.data.frame(TIME)
NAprices = as.data.frame(rep(NA,23401*days))
empty = data.frame(cbind(TIME,NAprices))
names(empty) <- c("TIME","PRICE")

### FILL MATRIX FOR HOMOGENEOUS SERIES 1-sec-series
# create series with 1 second spacing, observed prices for observed times, and NAs for unobserved times
library(zoo)
joint <- merge(SINGLE.TIMES,empty,by.x="TIME",by.y="TIME",all.y=TRUE)
joint$PRICE.x[is.na(joint$PRICE.x)] <- 0
joint$PRICE.y[is.na(joint$PRICE.y)] <- 0
HOMO <- data.frame(cbind(rep(NA,length(empty$TIME)),rep(NA,length(empty$TIME))))
names(HOMO) <- c("TIME","PRICE")
HOMO[,1] <- joint$TIME
HOMO[,2] <- joint$PRICE.x + joint$PRICE.y
# get numbers corresponding to index of observed days of the year (Jan 01 equals 1, Dec 31 equals 365)
ydays <- unique(as.POSIXlt(SINGLE.TIMES$TIME)$yday)
# set 09:30:00 prices of 1-sec-series equal to first price observation of the respective days
index.min.SINGLE.TIMES <- numeric(length(ydays))
for(i in 1:length(ydays)){index.min.SINGLE.TIMES[i] <- min(SINGLE.TIMES$TIME[as.POSIXlt(SINGLE.TIMES$TIME)$yday==ydays[i]])}
for(i in 1:length(ydays)){HOMO$PRICE[(i-1)*23401+1] <- SINGLE.TIMES$PRICE[SINGLE.TIMES$TIME==index.min.SINGLE.TIMES[i]]}
# set 16:00:00 price of 1-sec-series equal to last price observation of the day
```

```

index.max.SINGLE.TIMES <- numeric(length(ydays))
for(i in 1:length(ydays)){index.max.SINGLE.TIMES[i] <- max(SINGLE.TIMES$TIME[as.POSIXlt(SINGLE.TIMES$TIME)$yday==ydays[i]])}
for(i in 1:length(ydays)){HOMO$PRICE[i*23401] <- SINGLE.TIMES$PRICE[SINGLE.TIMES$TIME==index.max.SINGLE.TIMES[i]]}
# interpolate for missing prices (at unobserved times)
HOMO$PRICE[HOMO$PRICE==0] <- NA
HOMO[,2] <- na.approx(zoo(HOMO))[,2]

### -----
### (II) CREATE SERIES WITH SPECIFIC FREQUENCY AND COMPUTE RV
### -----

### -----
### function to create series with delta t (in seconds) spacing and to calculate RV for this spacing
### arguments: one.sec series with 1 second sampling frequency
### del - desired sampling frequency (in seconds) for new series
### -----
rv.delta.t <- function(one.sec, del)
{
# determine the number of trading days in 1 second series
number.days <- length(unique(as.POSIXlt(one.sec$TIME)$yday))
# determine # of intervals per day for a series with spacing del (cut off remainder)
number.intervals <- floor(23400/del)
# create series with spacing del
index <- numeric()
for(i in 1:number.days)
{for(j in 0:number.intervals){index <- c(index,((i-1)*23401+1)+j*del)}}
del.space <- one.sec[index,]
# calculate RV for each day in sample
number.obs.day <- number.intervals +1
del.rv <- numeric(number.days)
for(i in 1:number.days)
{del.rv[i] <- sum((diff(log(as.numeric(del.space$PRICE[((i-1)*number.obs.day+1):(i*number.obs.day)]))))^2)}
# calculate average RV for all days in sample
del.RV <- mean(del.rv)
# return Variables
list(del.rv=del.rv,del.RV=del.RV)
}

### compute realized variance for various frequencies
deltas <- c(1,2,3,4,5,10,20,30,60,120,180,240,300,600,1800)
RV <- numeric(length(deltas))
for(i in 1:length(deltas)){RV[i] <- rv.delta.t(one.sec=HOMO,del=deltas[i])$del.RV}
realized.volatility <- sqrt(RV*252)

### plot RV versus the sampling frequency
plot(x=deltas, y=realized.volatility, type="l", xlab="Sampling Frequency (in seconds)", ylab="Realized Volatility", main="Re

```

B VBA code

```
Sub f1Btn_Click()
    btnRun.Visible = False
    btnItp.Visible = False
    btnRv.Visible = False

    Dim f1 As Variant, s1 As Variant
    s1 = "#1: Choose the .txt file which contains montly data for the stock price."

    f1 = Application _
        .GetOpenFilename("TextFiles(*.txt),*.txt", , s1)
    txt2.Text = f1

    If (txt2.value = False) Then
        btnRun.Visible = False
        MsgBox ("Please Select a Text File.")
        GoTo q:
    Else
        btnRun.Visible = True
    End If
q:

End Sub

Sub btnRun_Click()
    btnItp.Visible = False
    btnRv.Visible = False
    Dim f1 As String

    f1 = txt2.Text
    Call doIt(f1)

End Sub
```

```

Sub doIt(f1 As String)
    Dim stime1
    stime1 = Time
    MsgBox (stime1)
    btnItp.Visible = False
    btnRv.Visible = False

    Dim txt
'These integers just counters
    Dim i As Long, j As Long, k As Long, l As Long, trial As Long

    Dim ua As Long, ub As Long
Dim sec() As Variant
    Dim a As Variant
    Dim b() As Variant
    Dim c() As Variant

'For this code to work the "times.txt" file must in in the folder
'specified below
    Open "C:\Users\IowaJB\Desktop\times.txt" For Input As #1
    Open f1 For Input As #2

    i = 0
'Reads seconds "9:30:00" to "16:00:00" into the array sec()
    Do Until EOF(1)
        Line Input #1, txt
        ReDim Preserve sec(i)
        sec(i) = txt
        i = i + 1
    Loop

    Dim usec As Long
    usec = UBound(sec)

'This file is interpreted as one line in Excel. This was very problematic
    Line Input #2, txt

'This splits the "One Line" into subgroups delimited by the linefeed
'character- Chr(10)
    a = Split(txt, Chr(10))
'this is the highest index of a(). Hence a() has indicies 1:ua
    ua = UBound(a)

    j = 0
    ReDim Preserve b(0)
    b(0) = ""
    For i = 1 To ua

'Lines of labels would sporadically appear among the data.
'Also cases of extreme outliers were marked with a code other than " 0"
'at character 72
'This ensures that only lines that are not a label and that were not
'outliers are kept
        If (Left(a(i), 1) <> ("S")) And (Mid(a(i), 47, 2) = " 0") Then
            ReDim Preserve b(j)
            b(j) = a(i)
            j = j + 1
        End If
    End If
    Next i
ub = UBound(b)

'We now create vectors of just the "Time"-t() and "Price"-p() from vector b().
'Also created is a vector price() which will contain the numeric conversions
'of the strings from p()
'The reason is so we can find all the Times which are equal and average the
'prices for that time.
'This average will be the single unique price for this time

```

```

Dim d() As String
Dim t() As String
Dim p() As String

For i = 0 To ub - 1
    ReDim Preserve d(i)
    ReDim Preserve t(i)
    ReDim Preserve p(i)
    d(i) = Mid(b(i), 11, 8)
    t(i) = Mid(b(i), 23, 8)
    p(i) = Mid(b(i), 35, 7)
Next i

Dim ud As Long
ud = UBound(d)

Dim dates()
Dim maxx()
Dim numdays As Integer
For i = 0 To ud - 1
    If d(i) = d(i + 1) Then
        numdays = numdays
    Else
        ReDim Preserve maxx(numdays)
        maxx(numdays) = i

        ReDim Preserve dates(numdays)
        dates(numdays) = d(i)
        numdays = numdays + 1
    End If
Next i

ReDim Preserve dates(numdays)
dates(numdays) = d(i)
ReDim Preserve maxx(numdays)
maxx(numdays) = ud

Sheets("Sheet1").Cells.Delete Shift:=xlUp

Dim priceFin()
Dim urep As Long
Dim times()
Dim prices()

For l = 0 To numdays
    trial = 2 * l + 1
    Dim replicates()
    Dim c1 As Long
    Dim c2 As Long
    Dim c3 As Long
    c1 = 0
    c2 = 0

    If l = 0 Then
        If 0 = maxx(l) Then
            ReDim replicates(0)
            replicates(0) = maxx(l)
        Else
            For i = 0 To maxx(l) - 1
                If t(i) = t(i + 1) Then
                    ReDim Preserve replicates(c2)
                    replicates(c2) = i
                Else
                    ReDim Preserve replicates(c2)
                    replicates(c2) = i
                    c2 = c2 + 1
                End If
            Next i
        End If
    End If
Next l

```



```

        Next i
    End If

'Need to take sum of prices at replicated times
    urep = UBound(replicates)

    ReDim Preserve prices(0)
    prices(0) = 0
    c1 = maxx(1 - 1) + 1
    For i = 0 To urep
        ReDim Preserve prices(i)
        For j = c1 To replicates(i)
            prices(i) = prices(i) + val(p(j))
        Next j
        c1 = j
    Next i

'Now average the prices
    prices(0) = prices(0) / (val(replicates(0)) - val(maxx(1 - 1)))

    For i = 1 To urep
        prices(i) = prices(i) / (val(replicates(i)) - _
            val(replicates(i - 1)))
    Next i

'combine seconds with prices
    usec = UBound(sec)

    ReDim Preserve priceFin(0)
    priceFin(0) = Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & _
        Chr(32) & Chr(32) & Chr(32) & "Price"
    j = 0
    For i = 0 To usec - 1
        If j <= urep Then
            If sec(i + 1) = t(replicates(j)) Then
                ReDim Preserve priceFin(i + 1)
                priceFin(i + 1) = prices(j)
                j = j + 1
            Else
                ReDim Preserve priceFin(i + 1)
                priceFin(i + 1) = 0
            End If
        Else
            ReDim Preserve priceFin(i + 1)
            priceFin(i + 1) = 0
        End If
    Next i
End If

'Combine Date and Time as Single String of form "YYYYmmdd-H:MM:SS"
    Dim dateFin() As Variant
    ReDim Preserve dateFin(0)
    dateFin(0) = Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & "Date"
    Dim s()
    ReDim Preserve s(0)
    s(0) = 0
    For i = 0 To usec - 1
        ReDim Preserve dateFin(i + 1)
        ReDim Preserve s(i + 1)
        s(i + 1) = sec(i + 1)
        If Left(s(i + 1), 1) = " " Then
            s(i + 1) = Right(s(i + 1), 7)
        End If
        dateFin(i + 1) = dates(1) & Chr(45) & s(i + 1)
    Next i

```

```

Worksheets("Sheet1").Cells(1, trial).value = dateFin(0)
Worksheets("Sheet1").Cells(1, trial + 1).value = priceFin(0)

If l = 0 Then
    For i = 1 To usec
        Worksheets("Sheet1").Cells(i + 1, trial).value = dateFin(i)
        Worksheets("Sheet1").Cells(i + 1, trial + 1).value = _
            val(priceFin(i + 1))
    Next i
Else
    For i = 1 To usec
        Worksheets("Sheet1").Cells(i + 1, trial).value = dateFin(i)
        Worksheets("Sheet1").Cells(i + 1, trial + 1).value = _
            val(priceFin(i))
    Next i
End If
Sheets("Sheet1").Columns(trial).EntireColumn.AutoFit
Sheets("Sheet1").Columns(trial + 1).EntireColumn.AutoFit
Sheets("Sheet1").Columns(trial + 1).EntireColumn.NumberFormat = _
    "0.0000"

Next l

Close #1
Close #2
btnRun.Visible = False
btnItp.Visible = True

Dim stime2
stime2 = Time
MsgBox (stime2)

End Sub

```

```

Sub btnItp_Click()
    Dim stime1
    stime1 = Time
    MsgBox (stime1)

    btnRv.Visible = False

    Dim i As Long
    Dim j As Long
    Dim col As Long
    Dim num As Long
    Dim v() As Long
    Dim uv As Long
    Dim diff As Long
    Dim itp As Double

col = 2
Do While Worksheets("Sheet1").Cells(1, col) = Chr(32) & Chr(32) & _
Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & "Price"
num = 0
    For i = 2 To 23402
        If Worksheets("Sheet1").Cells(i, col).value > 0 Then
            ReDim Preserve v(num)
            v(num) = i
            num = num + 1
        End If
    Next i
    uv = UBound(v)
    If v(0) <> 2 Then
        For i = 2 To v(0) - 1
            Worksheets("Sheet1").Cells(i, col).value = _
            Worksheets("Sheet1").Cells(v(0), col).value
        Next i
    End If

    If v(uv) <> 23402 Then
        For i = v(uv) + 1 To 23402
            Worksheets("Sheet1").Cells(i, col).value = _
            Worksheets("Sheet1").Cells(v(uv), col).value
        Next i
    End If
    If v(0) <> v(uv) Then
        For j = 0 To uv - 1
            diff = v(j + 1) - v(j)
            If diff > 1 Then
                For i = 1 To diff - 1
                    itp = (v(j + 1) - v(j) - i) / (v(j + 1) - v(j)) * _
                    (Worksheets("Sheet1").Cells(v(j), col).value) + i _
                    / (v(j + 1) - v(j)) * _
                    (Worksheets("Sheet1").Cells(v(j + 1), col).value)

                    Worksheets("Sheet1").Cells((v(j) + i), col).value _
                    = itp
                Next i
            End If
        Next j
    End If
    col = col + 2
Loop

    btnItp.Visible = False
    btnRv.Visible = True

    Dim stime2
    stime2 = Time
    MsgBox (stime2)

End Sub

```



```

    For i = 0 To uStore
        rv = rv + store(i)
    Next i

    Worksheets("Sheet1").Cells(23415, col - 1).value = "1800sec"
    Worksheets("Sheet1").Cells(23415, col).value = rv

    col = col + 2
Loop
' compute daily averages for each time
Worksheets("Sheet1").Cells(23417, 1).value = "1"
Worksheets("Sheet1").Cells(23418, 1).value = "2"
Worksheets("Sheet1").Cells(23419, 1).value = "3"
Worksheets("Sheet1").Cells(23420, 1).value = "4"
Worksheets("Sheet1").Cells(23421, 1).value = "5"
Worksheets("Sheet1").Cells(23422, 1).value = "10"
Worksheets("Sheet1").Cells(23423, 1).value = "30"
Worksheets("Sheet1").Cells(23424, 1).value = "60"
Worksheets("Sheet1").Cells(23425, 1).value = "120"
Worksheets("Sheet1").Cells(23426, 1).value = "300"
Worksheets("Sheet1").Cells(23427, 1).value = "600"
Worksheets("Sheet1").Cells(23428, 1).value = "1800"

col = 2
Dim num As Long
Dim rvTime As Double
Dim rvAvg() As Double
Dim secCounter

Do While Worksheets("Sheet1").Cells(1, col) = Chr(32) & Chr(32) & _
Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & Chr(32) & "Price"

    Dim n As Long

    rvTime = rvTime + Worksheets("Sheet1").Cells(23404, col).value
    n = n + 1
    col = col + 2
Loop
    Worksheets("Sheet1").Cells(23417, 2).value = Sqr(rvTime / n * 252)

num = n
Dim row As Long
row = 1

For secCounter = 23418 To 23428
    col = 2
    rvTime = 0

    For i = 1 To num
        rvTime = rvTime + _
        Worksheets("Sheet1").Cells(23404 + row, col).value
        col = col + 2
    Next i

    Worksheets("Sheet1").Cells(23417 + row, 2).value = _
    Sqr(rvTime / n * 252)
    row = row + 1
Next secCounter

Dim stime2
stime2 = Time
MsgBox (stime2)
btnRv.Visible = False

Sheets("Sheet1").Select

End Sub

```