Analysis of High Frequency Financial Data: Models, Methods and Software. Part I: Descriptive Analysis of High Frequency Financial Data with S-PLUS.

Eric Zivot*

July 4, 2005.

1 Introduction

High-frequency financial data are observations on financial variables taken daily or at a finer time scale, and are often irregularly spaced over time. Advances in computer technology and data recording and storage have made these data sets increasingly accessible to researchers and have driven the data frequency to the ultimate limit for some financial markets: time stamped transaction-by-transaction or tick-by-tick data, referred to as ultra-high-frequency data by Engle (2000). For equity markets, the Trades and Quotes (TAQ) database of the New York Stock Exchange (NYSE) contains all recorded trades and quotes on NYSE, AMEX, NASDAQ, and the regional exchanges from 1992 to present. The Berkeley Options Data Base recorded similar data for options markets from 1976 to 1996. In foreign exchange markets, Olsen Associates in Switzerland maintains a data base of indicative FX spot quotes for many major currency pairs published over the Reuters' network since the mid 1980's.

These high-frequency financial data sets have been widely used to study various market microstructure related issues, including price discovery, competition among related markets, strategic behavior of market participants, and modeling of real-time market dynamics. Moreover, high-frequency data are also useful for studying the statistical properties, volatility in particular, of asset returns at lower frequencies. Excellent surveys on the use of high-frequency financial data sets in financial econometrics are provided by Andersen (2000), Campbell, Lo and MacKinlay (1997), Dacarogna et. al. (2001), Ghysels (2000), Goodhart and O'Hara (1997), Gouriéroux and Jasiak (2001), Lyons (2001), Tsay (2001), and Wood (2000).

*Parts of these notes are based on the unpublished paper "Analysis of High Frequency Financial Data with S-PLUS" by Bingchen Yan and Eric Zivot. Data and S-PLUS scripts are available at http:\\faculty.washington.edu\ezivot\ezresearch.htm.

High-frequency financial data possess unique features absent in data measured at lower frequencies, and analysis of these data poses interesting and unique challenges to econometric modeling and statistical analysis. First, the number of observations in high-frequency data sets can be overwhelming. The average daily number of quotes in the USD/EUR spot market could easily exceed 20,000, and the average daily number of observations of an actively traded NYSE stock can be even higher. Second, data are often recorded with errors and need to be cleaned and corrected prior to direct analysis. For various reasons, high-frequency data may contain erroneous observations, data gaps and even disordered sequences. Third, transaction-by-transaction data on trades and quotes are, by nature, irregularly spaced time series with random daily numbers of observations. Moreover, trades and quotes on multiple assets seldom occur at the same time, and trading activity varies considerably across assets. Fourth, high-frequency data typically exhibit periodic (intra-day and intra-week) patterns in market activity. It is well known that trading activities at the NYSE are more dense in the beginning and closing of the trading day than in the lunch hours. FX trading activities also systematically vary as the earth sequentially passes through the business hours of geographical trading centers. Furthermore, discrete price movements, nonsynchronous trading, and bid-ask bounce may distort inferences based on standard statistical models.

The above characteristics of high frequency financial data substantially complicate the process of econometric and statistical analysis, and typical statistics and econometrics software do not contain the tools necessary to properly handle and analyze high frequency data. S-PLUS, with its rich and flexible object oriented statistical modeling language and graphical facilities, is ideally suited for the analysis of high frequency data. This part of the lecture illustrates how to process and descriptively analyze high-frequency financial data using the S-PLUS statistical modeling language and the S+FinMetrics module for the analysis of time series data. The goal are (1) to provide a practical guide to high-frequency financial data analysis, from getting raw data into the software program, to preparing data for analysis and creating relevant variables, and to performing basic descriptive and graphical analysis; (2) to illustrate the basic characteristics of high frequency financial time series, and to motivate the statistical modeling of high frequency data. Three example data sets are used to demonstrate the applications of techniques and tools discussed, two from equity markets (TAQ data) and one from FX markets (Olsen data). The lectures make use of the S-PLUS library HF developed by Bingchen Yan and Eric Zivot, which contains a collection of functions specially designed for high-frequency financial data analysis.

The organization of the lecture is as follows. Section 1 gives a brief overview of the S-PLUS library HF. Section 3 introduces three example data sets and describes how to load and process the data for further analysis. Section 4 deals with basic data manipulations, such as creating various market variables, performing summary statistics, regularizing unequally spaced data. It also illustrates some empirical characteristics of high-frequency data using basic descriptive statistics and graphical techniques.

2 Overview of the S-PLUS HF library

The S-PLUS HF library is a collection of S-PLUS functions written by Bingchen Yan and Eric Zivot[1]. Table 1 gives a brief summary of the main functions in the library. The functions make use of the proprietary "timeDate" and "timeSeries" classes in S-PLUS, version 6.0 and higher, that can be used to characterize irregularly spaced, intra-day high frequency time series. Functions are included to load data from the TAQ and Olsen data, to perform data manipulation and descriptive analysis over specified trading periods, and to construct variables frequently used in the analysis of high frequency time series.

The following sections illustrate the descriptive analysis of high frequency financial time series using S-PLUS and the functions in the HF library.

3 Data Processing

3.1 Data Sets

The data sets used in this lecture are trades and quotes data for Microsoft and GE (05/01/1997–05/15/1997) and USD/EUR spot rate quotes (03/11/2001–03/17/2001). The trades and quotes data for Microsoft are saved in the ASCII files"trade_msft.txt" and "quote_msft.txt", while similar data for GE are saved in "trade_ge.txt" and "quote_ge .txt". These data sets contain standard and complete information from the TAQ database[2]. For example, the first six rows of trade_msft.txt are:

```
cond |ex |symbol |corr |g127 |price |siz |tdate |tseq |ttim |
T  |T  |MSFT  |0  |0  |121.125  |1500  |01MAY1997  |0  |28862  |
T  |T  |MSFT  |0  |0  |121.5625  |500  |01MAY1997  |0  |28944  |
T  |T  |MSFT  |0  |0  |121.5625  |1000  |01MAY1997  |0  |29000  |
T  |T  |MSFT  |0  |0  |121.5625  |1200  |01MAY1997  |0  |29002  |
T  |T  |MSFT  |0  |0  |121.625  |1000  |01MAY1997  |0  |31095  |
```

The trades data have 10 columns separated by "|". The most important columns are "symbol" for stock symbol (e.g. "GE" or "MSFT"), "price" for transaction prices (e.g. 110.625), "size" for traded size in number of shares (e.g. 100), "tdate" for date of the trade (e.g. "01MAY1997"), and "ttime" for time of the trade in seconds since the midnight of the day (e.g. 34220). The time used in the TAQ database is recorded in

[1]The library FH was developed by the authors and is available for download at http:\\faculty.washington.edu\ezivot\splus.htm. The library was created using S-PLUS 6.2. The library is currently being updated to incorporate the big data features of S-PLUS 7.0. Wolfgang Breymann also has an S-PLUS library of functions for the analysis of high frequency foreign exchange rate data available at http://www.math.ethz.ch/~breymann/.

[2]For a detailed explanation of the complete fields in the TAQ database, see the online TAQ2 user's guide at http://nyse.com/marketinfo/taqdatabase.html.

Function	Description
	Data loading
`TAQLoad`	Load TAQ data into timeSeries
`OlsenLoad`	Load Olsen data into timeSeries
	Time series and data manipulation
`reorderTS`	Correct ordering of dates in timeSeries
`plotByDays`	Plot timeSeries by days
`is.tsBW`	Determine if timeSeries lie in specified interval
`tsBW`	Extract timeSeries within interval
`ExchangeHoursOnly`	Restrict timeSeries to exchange hours
`FxBizWeekOnly`	Restict dates business week
`diff.withinDay`	Take difference within 1 day period
`diff.withinWeek`	Take difference with 1 week period
`align.withinWeek`	Align to regular clock within a week
`align.withinDay`	Align to regular clock within a day
`aggregateSeriesHF`	Faster version of aggregateSeries
`SmoothAcrossIntervs`	Smooth data in intervals across days or weeks
	Variable construction
`DurationInInterv`	Compute time between trades
`PriceChgInInterv`	Compute price change in interval
`getSpread`	Compute bid/ask spread
`getMidQuote`	Compute midquote
`aggregateTradeTypes`	Aggregate trade direction indicator over interval
`DetermInterp`	Interpolate across interval
`naDuration`	Count number of sequential NA values
`numNAs`	Determine of number of NA values
`TradeDirec`	Determine if transaction is buy or sell

Table 1: Summary of S-PLUS HF library functions.

US Eastern time accommodating the daylight saving time. The quotes data have 11 "|" separated columns, the most important of which are "symbol" for stock symbol, "bid" for bid prices (e.g. 121.5), "bidsiz" for bid size in number of round lots, i.e. 100 share units (e.g. 11), "ofr" for ask prices (e.g. 121.625), "ofrsiz" for ask size in number of round lots (e.g. 11), "qdate" for date of the quote, and "qtime" for time of the quote in seconds since midnight of the day.

The USD/EUR quotes data are saved in the ASCII file "eur_usd.txt" and each record contains 4 or 5 white space separated fields: date, time in GMT, ask quote, bid quote and quoting institution. For FX quotes data, the date and time are directly expressed in conventional format, e.g. "04.03.2001 14:41:30" for "dd.mm.yyyy HH:MM:SS" (European time-date format). For example, the first five rows of eur_usd.txt are

```
11.03.2001 01:07:46 0.93370 0.93420 AREX
11.03.2001 01:07:52 0.93360 0.93410 AREX
11.03.2001 01:07:57 0.93340 0.93390 AREX
11.03.2001 01:08:04 0.93370 0.93420 AREX
11.03.2001 05:42:35 0.93300 0.93400 CMBK
```

3.2 Data Loading

All data sets are in ASCII format and have to be loaded into S-PLUS for further analysis. The functions `TAQLoad( )` and `OlsenLoad( )` in the HF library take the TAQ data and Olsen's FX quote data in their standard formats and save the resulting data as an S version 4 (SV4) "timeSeries" object. Assuming the data sets are saved in the directory `\C:\HFAnalysis\"`, the Microsoft trade data are loaded using `TAQLoad( )` as follows:

```
> msftt.ts = TAQLoad(file = "C:\\HFAnalysis\\trade_msft.txt",
+                    type = "trade", sep = "|", skip = 1)
```

The function `TAQLoad( )`takes the path and name of the data file through the argument `file`; the argument `type` specifies if the data is `trade` or `quote`; `sep` specifies the delimiter/separator between fields used in the data file; `skip` tells the loading function how many rows to skip before starting to read in data.

The remaining TAQ data can be loaded similarly: `msftq.ts` for the Microsoft quotes data, `get.ts` for the GE trades data, and `geq.ts` for the GE quotes data:

```
> msftq.ts = TAQLoad(file = "C:\\HFAnalysis\\quote_msft.txt",
+                    type = "quote", sep = "|", skip = 1)
> get.ts = TAQLoad(file = "C:\\HFAnalysis\\trade_ge.txt",
+                  type = "trade", sep = "|", skip = 1)
> geq.ts = TAQLoad(file = "C:\\HFAnalysis\\quote_ge.txt",
+                  type = "quote", sep = "|", skip = 1)
```

The first 5 rows of the Microsoft trades data can be viewed by typing

```
> msftt.ts[1:5,  ]
        Positions Cond Ex Symbol Corr G127    Price Size Seq
 5/1/1997 8:01:02 T    T  MSFT   0    0    121.1250 1500 0
 5/1/1997 8:02:24 T    T  MSFT   0    0    121.5625  500 0
 5/1/1997 8:03:20 T    T  MSFT   0    0    121.5625 1000 0
 5/1/1997 8:03:22 T    T  MSFT   0    0    121.5625 1200 0
 5/1/1997 8:38:15 T    T  MSFT   0    0    121.6250 1000 0
```

The first 5 rows of the Microsoft quotes data are:

```
> msftq.ts[1:5,  ]
        Positions Ex MMID Symbol     Bid BidSize Mode     Ask AskSize Seq
 5/1/1997 8:17:24 T       MSFT   121.500 11      12   121.625 11      0
 5/1/1997 9:00:44 T       MSFT   121.750 10      12   121.625 11      0
 5/1/1997 9:07:27 T       MSFT   121.750 10      12   121.625 10      0
 5/1/1997 9:16:30 T       MSFT   121.875 10      12   121.625 10      0
 5/1/1997 9:20:29 T       MSFT   121.875 10      12   121.625  3      0
```

Notice that the displayed trades and quotes data only have 9 and 10 columns respectively, rather 10 and 11 columns that appear in the text files. The reason is that the loading function combines the date and time information in the text file into an SV4 "`timeDate`" object represented in the "`Positions`" column.

Any time series in S-PLUS may be represented by an SV4 "timeSeries" object, which contains two basic parts: time date information and data series information. These two parts, together with other attributes of the object, are constructed as slots to the object, the name of which can be viewed using function `slotNames( )`. For example, the slots of the "timeSeries" object `msftt.ts` are

```
> slotNames(msftt.ts)
 [1] "data"              "positions"         "start.position"
 [4] "end.position"      "future.positions"  "units"
 [7] "title"             "documentation"     "attributes"
[10] "fiscal.year.start" "type"
```

The slots `data` and `positions` contain the fundamental data and time date information of a "timeSeries" object, and can be accessed by the @ operator or the extractor functions `seriesData()` and `positions()`. For example, the first 5 rows of the contents of the `data` slot to `msftt.ts` are

```
> msftt.ts@data[1:5,  ]
  Cond Ex Symbol Corr G127    Price Size Seq
1    T  T   MSFT    0    0 121.1250 1500   0
2    T  T   MSFT    0    0 121.5625  500   0
```

```
3    T  T   MSFT    0    0 121.5625 1000   0
4    T  T   MSFT    0    0 121.5625 1200   0
5    T  T   MSFT    0    0 121.6250 1000   0
```

The command `seriesData(msftt.ts)[1:5,]` gives the same result. To access the first 5 rows of the `positions` slot use

```
> msftt.ts@positions[1:5]
[1] 5/1/1997 8:01:02 5/1/1997 8:02:24
[3] 5/1/1997 8:03:20 5/1/1997 8:03:22
[5] 5/1/1997 8:38:15
```

or `positions(msftt.ts)[1:5,]`. Note that these time date records are in US Eastern time. For full details on the "`timeSeries`" object, see the online help for `class.timeSeries`.

To load the USD/EUR data, use the loading function `OlsenLoad( )` as follows:

```
> eurusd.ts = OlsenLoad(file = "C:\\HFAnalysis\\eur_usd.txt")
```

The function `OlsenLoad( )` takes a `file` argument for the path and name of the data file and creates the "timeSeries" object `eurusd.ts`. The standard format of Olsen's data is used as the default specification of the loading function, and the function automatically transforms European time date format to American time date format. The first 5 rows of `eurusd.ts` are

```
> eurusd.ts[1:5,  ]
         Positions    Bid    Ask Institution
 3/11/2001 1:07:46 0.9337 0.9342 AREX
 3/11/2001 1:07:52 0.9336 0.9341 AREX
 3/11/2001 1:07:57 0.9334 0.9339 AREX
 3/11/2001 1:08:04 0.9337 0.9342 AREX
 3/11/2001 5:42:35 0.9330 0.9340 CMBK
```

The time and date of `eurusd.ts` are in Greenwich Mean Time (GMT), which is typical for FX data. The contents of a particular column, say the bid quotes, can be viewed by typing

```
> eurusd.ts[1:5, "Bid"]
         Positions    Bid
 3/11/2001 1:07:46 0.9337
 3/11/2001 1:07:52 0.9336
 3/11/2001 1:07:57 0.9334
 3/11/2001 1:08:04 0.9337
 3/11/2001 5:42:35 0.9330
```

Notice that S-PLUS allows using row or column names to subscript data. Another way to subscript data using logical values will be discussed below.

To summarize, using the data loading functions `TAQLoad( )` and `OlsenLoad( )`from the HF library, five "timeSeries" objects, `msftt.ts`, `msftq.ts`, `get.ts`, `geq.ts`, and `eurusd.ts`, have been created.

3.3 Data Examination and Cleaning

Raw high-frequency financial data can have a variety of problems. Some of these problems may result from human input errors, like typing errors leading to data outliers. Other errors may result from computer system errors, such as transmission failures leading to data gaps, and database bugs leading to mis-ordered time series observations. Failure to recognize and account for these data problems may cause misleading results in subsequent statistical analysis. The HF library offers several routines to discover and correct these data problems.

Mis-ordered time series refers to the situation in which observations still carry correct time stamp information but they are not in a strict increasing time order. For example, one observation has a time stamp of 10:05:34 and the next observation has a time stamp of 10:05:29. This type of data problem can be prevented by filtering with the HF library function `reorderTS( )`. For example, to correct any mis-ordered data in `msft.ts`, `msftq.ts`, `get.ts`, `geq.ts`, and `eurusd.ts` use

```
> msftt.ts = reorderTS(msftt.ts)
> msftq.ts = reorderTS(msftq.ts)
> get.ts = reorderTS(get.ts)
> geq.ts = reorderTS(geq.ts)
> eurusd.ts = reorderTS(eurusd.ts)
```

The function `reorderTS( )` takes in a "timeSeries" and outputs a correctly ordered version of the "timeSeries" based on the re-ordering the positions information of the "timeSeries".

For data problems like data outliers and gaps, the most effective diagnostic method is to plot the data. However, due to the potentially enormous amount of observations in high-frequency data, plotting all available data may overwhelm the plotting device and produce an unreadable plot. The HF function `plotByDays( )`separates data from a high frequency "timeSeries" into daily intervals and produces separate time plots for each day. This makes it easier to visually inspect the data for problems. To illustrate, time plots for seven days of Microsoft traded prices and USD/EUR bid quotes are created using

```
> par(mfrow = c(3, 3))
> plotByDays(ts = msftt.ts, coltoplot = "Price", days.max = 7)
> par(mfrow = c(3, 3))
> plotByDays(ts = eurusd.ts, coltoplot = 1, days.max = 7)
```

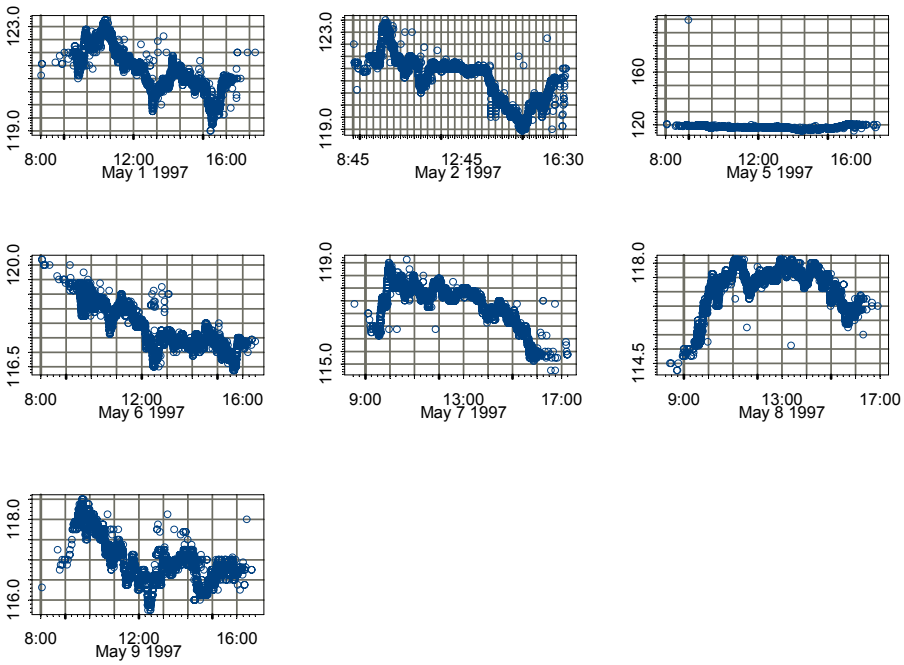

Figure 1: Seven days of intra-day price data for Microsoft stock.

and are illustrated in Figures 1 and 2. The required argument `ts` accepts the "time-Series" to plot. The `coltoplot` argument specifies which column(s) of the series to plot. The argument `days.max = 7` limits plots to the first seven days. The default is to plot the first column of the supplied series for all days with two or more observations.

In Figure 1, the plotting period is from 05/01/1997, Monday, through 05/09/1997, Friday, excluding the weekend of 05/03/ 1997 - 05/04/1997. The plots suggest that the trading activity of Microsoft's stock was very dense during the sample. In addition, the diagnostic plots identify a data outlier around 9:00 on 05/05/1997, with trading price more than 180, significantly higher than surrounding observations and highly unlikely. Furthermore, the plots reveal that, although the official trading hours of the NYSE are from 9:30 to 16:00 Eastern Time, data outside this interval are frequently observed.

The plotting period for the FX quotes in Figure 2 is from 03/11/2001, Sunday, through 03/17/2001, Saturday, one whole week. These plots appear free from outliers. This is to be expected since Olsen & Associates pre-processes the raw data using a sophisticated data cleaning filter (see Dacarogna et. al. (2001)) as the raw data from the Reuters network is prone to outliers and other data errors. The plots suggest that the business week in the FX market starts from early Monday, trading

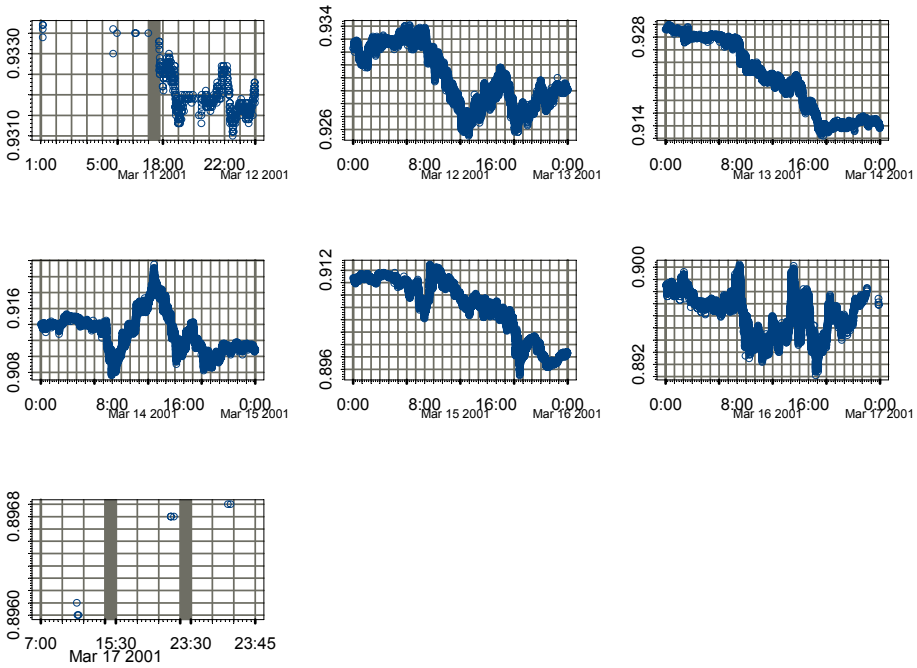

Figure 2: Seven days of intra-day quote data on Eur/$ exchange rate.

in Australia and New Zealand (still Sunday in GMT), and ends with late Friday, trading in North America (before Friday midnight in GMT) without any interruption between weekdays. The FX market is generally quiet over the weekend.

Often there is the need to limit the analysis to data within certain time intervals. For example, observations with time stamps outside the official business hours of the NYSE, as exemplified by the plots in Figure 1, are typically considered erroneous and therefore should be excluded. For foreign exchange markets, the plots in Figure 2 show that the weekend period from late Friday to Sunday night is inactive and data during this period behaves much differently than the active within week period. For modeling FX data, the convention is to have the business week run from Sunday 22:00 GMT to Friday 22:00 GMT and exclude observations with time stamp outside of this time range.

The HF library function `ExchangeHoursOnly( )` can impose business hours restrictions to data from exchanges with well defined daily opening and closing hours. For example, to eliminate or filter out the observations of Microsoft trades outside the NYSE official hours use:

```
> msftt.ts = ExchangeHoursOnly(ts = msftt.ts,
+                   exch.hours = c("9:30", "16:00"))
```

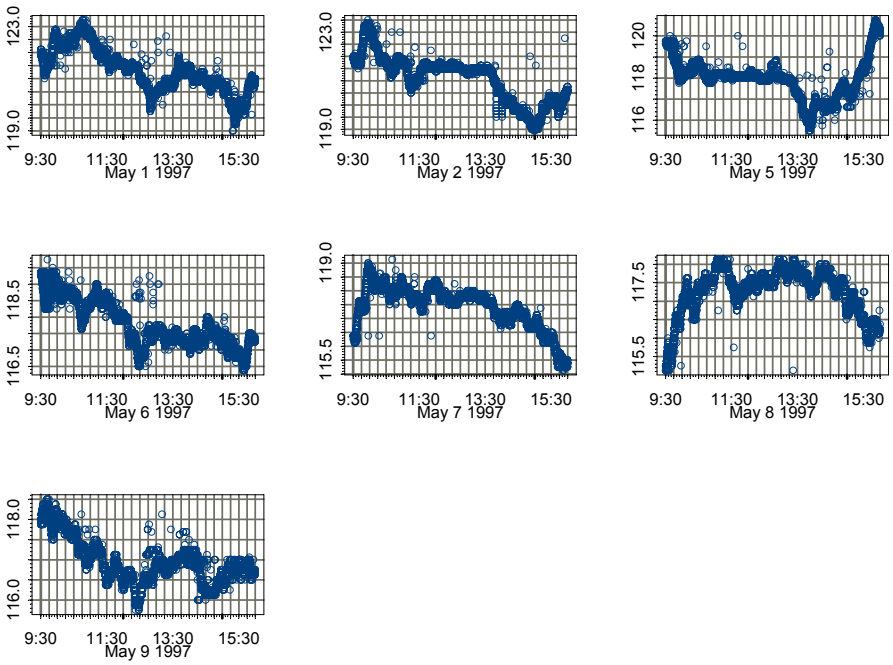

Figure 3: Seven days of intra-day price data for Microsoft stock restricted to NYSE exchange hours.

where the argument `exch.hours` takes a two-element character vector specifying the opening and closing hours of the exchange. Other data series from the TAQ database can be filtered in similar fashion. The business hours filtered Microsoft trades prices (first 7 days) are plotted in Figure 3, for comparison with the plots in Figure 1. Clearly, the data points earlier than 9:30 and later than 16:00 have been successfully removed.

For the data from the FX markets, the HF library function `FxBizWeekOnly( )` can impose business week restrictions in a flexible manner. Assuming that the business week is periodic, i.e. 5 days, and from Sunday 22:00 GMT to Friday 22:00 GMT, the weekend observations of the USD/EUR data can be filtered out using

```
> eurusd.ts = FxBizWeekOnly(ts = eurusd.ts,
+                           bizweek.hours = c("22:00", "22:00"))
```

where the `bizweek.hours` argument determines the specification of the business week beginning hour and closing hour. The definition of the business week is quite flexible and a non-periodic business week specification is also allowed. Figure 4 presents the plots of business week filtered USD/EUR bid quotes. Notice that the data points

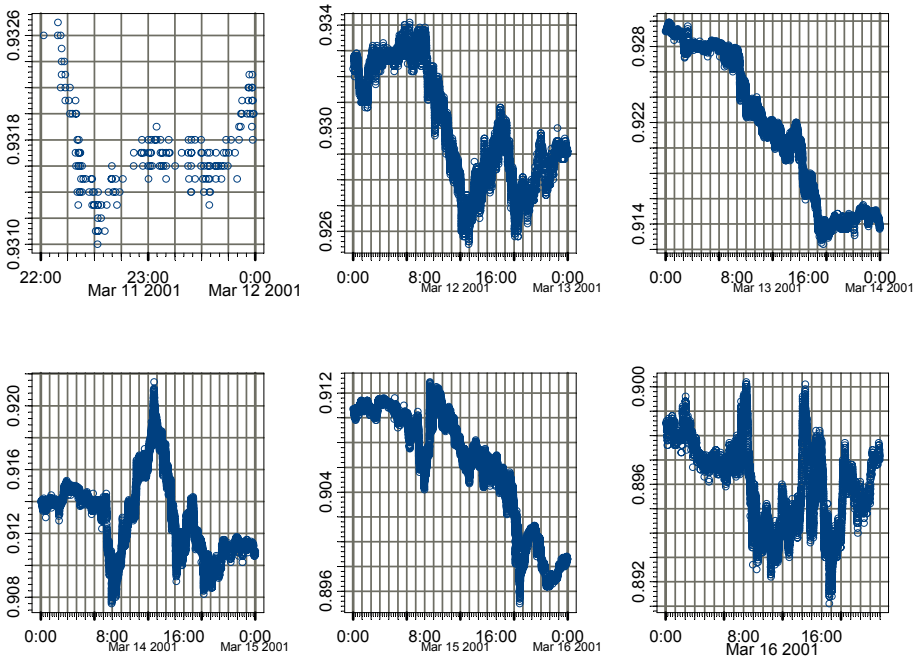

Figure 4: Seven days of intra-day quote data on Eur/$ exchange rate restricted to business week.

earlier than 22:00:00 03/11/2001, Sunday, and later than 22:00:00 03/16/2001, Friday, have been filtered out and the whole Saturday activity on 03/17/2001 has been eliminated.

4 Data Manipulation and Characterization

4.1 Construction of Market Variables

This subsection discusses how to construct market variables that are of direct interest to high-frequency financial data analysis. These variables include price change from transaction to transaction (or from quote to quote for FX data since transaction data are rarely available), duration or time span between trades or quotes, and spread between bid and ask quotes. The HF library offers several functions to conveniently construct these variables.

4.1.1 Price Change

The price change is the amount of price movement from transaction to transaction. It may be expressed on the same scale as the price level; e.g., the price change of 0.125 from \$120.00 to \$120.125 is expressed in dollars. Alternatively, due to the fact that the prices of almost all financial prices move in minimum increments, or tick sizes, the price change may also be expressed in terms of the number of ticks, where the tick size may vary from asset to asset and may even change over time for the same asset. For example, the minimum price movement for USD/EUR is 0.0001 and is 0.01 for JPY/USD. For stocks traded on the NYSE, the tick size was \$1/8 (\$0.125) before June 24, 1997, \$1/16 (\$0.0626) until January 29, 2001, and afterwards the tick size of all stocks traded on the NYSE and AMEX became \$0.01. The tick size for the sample TAQ data used in this lecture (5/1–15, 1997) is \$1/8.

The price change is computed by differencing the price level data. However, stock exchanges have well-specified daily business hours and the FX markets effectively operate on a business week basis. Stoll and Whaley (1990) showed that overnight stock returns differ substantially from intraday returns. Therefore, it is necessary to separate the price changes between trading days and over weekends from others within normal trading intervals when computing price changes. The HF function `PriceChgInInterv( )` is designed to perform this task, and computes price changes of high-frequency price data only within the specified trading intervals. The arguments accepted by the function are

```
> args(PriceChgInInterv)
function(x, ticksize = NULL, interv.type = "daily",
         bound.hours = c("9:30", "16:00"))
```

The function accepts the price level time series through the argument `x`. The optional argument `ticksize` will output the price changes as the number of ticks; otherwise the price changes are in the same scale as price levels. The argument `interv.type` specifies the trading session type, "`daily`" for exchange markets with daily business hours, or "`weekly`" for the FX markets. The `bound.hours` specifies the daily opening and closing hours of exchanges or the starting hour of one business week on Sunday and the closing hour of the business week on Friday.

To illustrate, the price changes in multiples of ticks of MSFT trading prices are computed using[3]

```
> pcTicks.msft = PriceChgInInterv(x = msftt.ts[, "Price"], ticksize = 1/8,
              interv.type = "daily", bound.hours = c("9:30", "16:00"))
> pcTicks.msft[1:5]
        Positions Price
 5/1/1997 9:30:06  1
```

[3] show data

```
 5/1/1997 9:30:09 -1
 5/1/1997 9:30:10  0
 5/1/1997 9:30:14  1
 5/1/1997 9:30:14 -1
```

Similarly, the price changes in multiples of ticks of USD/EUR bid quotes are

```
> pcTicks.eurusd = PriceChgInInterv(eurusd.ts[, "Bid"], ticksize = 0.0001,
            interv.type = "weekly", bound.hours = c("22:00", "22:00"))
> pcTicks.eurusd[1:5]
          Positions Bid
 3/11/2001 22:01:37  0
 3/11/2001 22:09:34  0
 3/11/2001 22:09:36  1
 3/11/2001 22:11:08 -5
 3/11/2001 22:11:10  4
```

4.1.2 Duration

A basic property of high-frequency financial data is that the transaction times of trades and quotes are unequally spaced, which implies that the time duration between transactions is not constant[4]. Duration can be computed by differencing the times at which successive trades or quotes occurred. However, like price changes, durations between trading days and over weekends should not be treated the same way as durations within the normal business hours. The HF function `DurationInInterv( )` computes durations between high-frequency observations only within specified trading intervals, and exclude the durations across trading intervals. The arguments expected by `DurationInInterv` are

```
> args(DurationInInterv)
function(x, units = "seconds", interv.type = "daily",
          bound.hours = c("9:30", "16:00"))
```

The argument `x` takes a "timeSeries" object or positions of a "timeSeries" object to compute the durations between successive observations. The `units` argument specifies the time units to express the durations. The default is to express durations in seconds. Other valid choices are "`milliseconds`" and "`minutes`". The arguments `interv.type` and `bound.hours` have the same meaning as in the function `PriceChgInInterv( )`.

For example, to compute the durations expressed in seconds between MSFT trades within the daily trading interval and the NYSE trading hours use

[4]Easley and O'Hara (1992) proposed a theoretical model in which varying durations carry different information content about price movements. Engle and Russell (1998) modeled the duration process with an Autoregressive Conditional Duration (ACD) model.

```
> duration.msftt = DurationInInterv(x = msftt.ts, units = "seconds",
            interv.type = "daily", bound.hours = c("9:30", "16:00"))
> duration.msftt[1:5,  ]
        Positions Duration.in.seconds
 5/1/1997 9:30:06 4
 5/1/1997 9:30:09 3
 5/1/1997 9:30:10 1
 5/1/1997 9:30:14 4
 5/1/1997 9:30:14 0
```

To compute the duration in seconds between USD/EUR quotes with a weekly trading interval and the business week from Sunday 22:00 to Friday 22:00, GMT, use

```
> duration.eurusd = DurationInInterv(x = eurusd.ts, units = "seconds",
            interv.type = "weekly", bound.hours = c("22:00", "22:00"))
> duration.eurusd[1:5,  ]
          Positions Duration.in.seconds
 3/11/2001 22:01:37   2
 3/11/2001 22:09:34 477
 3/11/2001 22:09:36   2
 3/11/2001 22:11:08  92
 3/11/2001 22:11:10   2
```

4.1.3 Spread

The bid-ask spread is computed by subtracting bid quotes, the prices at which the market maker or other traders in the market will be willing to buy assets, from ask quotes, the prices at which the market maker or other traders in the market will be willing to sell assets. The determination of bid-ask spread is one of the most successful areas of the market microstructure research. A rich literature has developed to explain how inventory controls, asymmetric information, and processing costs may affect bid-ask spread, as well as the relationship between the spread with other financial variables like asset return volatility.

Because prices of financial assets move in multiples of ticks, the bid-ask spread also moves in multiples of ticks. The HF function `getSpread( )` computes the spread from a time series of ask quotes and a time series of bid quotes, with the option to express the spread in term of tick size. The arguments expected by `getSpread( )` are

```
> args(getSpread)
function(ask, bid, ticksize = NULL)
```

The arguments `ask` and `bid` take time series of ask and bid quotes separately. The optional argument `ticksize` specifies the size of the minimum price movement and

expresses the spread in multiples of ticks; otherwise, the spread is expressed in the same units as the bid/ask price.

To compute the spread of MSFT bid and ask quotes in multiples of ticks with ticksize \$1/8 use

```
> spread.msft = getSpread(ask = msftq.ts[, "Ask"],
                    bid = msftq.ts[, "Bid"], ticksize = 1/8)
> spread.msft[1:5,  ]
        Positions Spread
 5/1/1997 9:30:14 1
 5/1/1997 9:30:17 2
 5/1/1997 9:30:17 1
 5/1/1997 9:30:21 1
 5/1/1997 9:30:57 1
```

Similarly, the spread of USD/EUR bid and ask quotes in multiples of ticks with ticksize 0.0001 may be computed using

```
> spread.eurusd = getSpread(ask = eurusd.ts[, "Ask"],
                      bid = eurusd.ts[, "Bid"], ticksize = 0.0001)
> spread.eurusd[1:5,  ]
          Positions Spread
 3/11/2001 22:01:35 4
 3/11/2001 22:01:37 5
 3/11/2001 22:09:34 5
 3/11/2001 22:09:36 5
 3/11/2001 22:11:08 5
```

4.1.4 Trade Direction

With high frequency equity data, transaction prices are often observed but the direction of trade is generally not observed[5]. Although each trade involves a buyer and a seller, whether the trade is market buyer-initiated or market seller-initiated has direct impact on price movements and is of particular importance to many market microstructure studies. One naive approach to infer trade direction might be to match the trade price with the contemporaneous quote prices. If the trade occurs on the bid price, the trade is initiated by a market sell order; if the trade occurs on the ask price, the trade is initiated by a market buy order. However, Lee and Ready (1991) showed that equity quotes are often recorded on the Consolidated Tape ahead of the trade that triggered them, and so they proposed to match trades to quotes that are set at least five seconds prior to the transaction. Furthermore, the standing

[5]Since the trade data are rarely available and not covered in the sample data, computing trade direction is only limited to the equity sample data.

orders held by floor brokers on the NYSE often drive transaction prices within the bid-ask spread. See Lee and Ready (1991) and Hausman, Lo, and MacKinlay (1992) for further discussion and references therein.

The HF library function `tradeDirec()` computes trade direction using the algorithm in Hausman, Lo, and MacKinly (1992): classify a trade as a "buy" if the transaction price is higher than the mid-quote of the prevailing bid-ask quotes (the most recent quote that is at least five seconds older than the trade); classify as a sell if the price is lower than the mid-quote; classify as an "indeterminate" trade if the price equals the mid-quote. The arguments of the function are:

```
> args(tradeDirec)
function(trade, mq, timeLag = "5s")
```

The function accepts a "`timeSeries`" of transaction prices through the `trade` argument, and a "`timeSeries`" of mid quotes through the `mq` argument. The `timeLag` component specifies the time lag used to match trades with quotes. The default is 5 seconds.

To illustrate the function, consider computing the trade direction of the MSFT trades. As input to `tradeDirec()`, the mid-quote of the MSFT quotes can be computed using the HF library function `getMidQuote()`:

```
> mq.msft = getMidQuote(ask = msftq.ts[,"Ask"],
  bid = msftq.ts[, "Bid"])
```

The trade direction of the MSFT trades is then estimated using:

```
> trade.direc.msft = tradeDirec(trade = msftt.ts[, "Price"],
                   mq = mq.msft)
> trade.direc.msft[1:5,  ]
        Positions BuySellDirec
 5/1/1997 9:30:02 0
 5/1/1997 9:30:06 1
 5/1/1997 9:30:09 0
 5/1/1997 9:30:10 0
 5/1/1997 9:30:14 1
```

4.1.5 Realized Volatility

Volatility of asset returns is central to modern finance theory, and is widely used in asset pricing, portfolio selection, and risk management. Unlike other market variables like price changes and spread, the volatility is not directly observable and there is no unique method for estimating it. Model-based volatility measures include GARCH models, stochastic volatility models, or the volatility implied by options or other

derivatives prices. Andersen et. al. (2001) propose an alternative mode-free approach to estimate ex post realized volatility from squared returns within the volatility horizon. They prove that as the sampling frequency of returns approaches infinity, realized volatility measures are asymptotically free of measurement error. For daily volatility, they use 5-minute returns to construct daily realized volatilities. The 5-minute horizon is short enough to have the underlying asymptotic work well, but long enough to mitigate the autocorrelation distortion caused by market microstructure frictions.

The HF function `Genr.RealVol()` provides a flexible and convenient tool to compute realized volatilities at different frequencies from high-frequency data. The arguments expected by `Genr.RealVol()` are

```
> args(Genr.RealVol)
function(ts, interv.type = "daily", bound.hours = c("9:30", "16:00"),
rv.span, rt.span, how.align = "before")
```

The `ts` argument accepts a one-column high-frequency logarithmic price series; `NA`s are allowed in the price series, but are effectively purged before further processing; the arguments `interv.type` and `bound.hours` have similar interpretations as in `PriceChgInInterv()`; `rv.span` takes value of a `timeSpan` object specifying the time span over which realized volatilities are defined, e.g. `timeSpan("1d")` for daily realized volatility; `rt.span` is also a `timeSpan` parameter specifying the time span of returns used to compute realized volatility, e.g. `timeSpan("5m")` for 5-minute returns; the argument `how.align` determines how to sample irregularly spaced high-frequency data at an equally spaced time scale. The default value `\before"` will impute the most recent observation within the trading session or the nearest observation in case of no observation before the sampling position available. For more options on alignment, see online help for `align()`.

In addition, `Genr.RealVol()` processes high-frequency data by trading sessions, and will generate extra 0 returns if a price series is short to full trading sessions. In other words, for each trading session, `Genr.RealVol()` first generates a full trading session of `rt.span` returns, including artificially generated 0 returns, and then computes `rv.span` realized volatilities. This rule provides the greatest possible accommodation to various types and formats of high-frequency data, but requires users' discretions on input choices.

To illustrate, the daily realized volatilities (in percentages) of MSFT trade prices based on 5-minute returns are computed using

```
> rvDaily.msft = Genr.RealVol(ts = log(msftt.ts[, "Price"])*100,
  interv.type = "daily", bound.hours = c("9:30", "16:00"),
  rv.span = timeSpan("6h30m"), rt.span = timeSpan("5m"))
> rvDaily.msft[1:5,  ]
         Positions RealizedVol
```

```
 5/1/1997 16:00:00 2.149662
 5/2/1997 16:00:00 1.869500
 5/5/1997 16:00:00 2.357502
 5/6/1997 16:00:00 2.232159
 5/7/1997 16:00:00 2.215329
```

Notice that the effective "day" for MSFT trading only lasts 6 hours and 30 minutes.

For the Fx data, the mid quote will be used as a substitute for the transaction price in equity data:

```
> mq.eurusd = getMidQuote(ask = eurusd.ts[, "Ask"], bid = eurusd.ts[, "Bid"])
```

To compute daily realized volatilities (in percentage) of USD/EUR based on 5-minute returns, use

```
> rvDaily.eurusd = Genr.RealVol(ts = log(mq.eurusd)*100,
  interv.type = "weekly", bound.hours = c("22:00", "22:00"),
  rv.span = timeSpan("24h"), rt.span = timeSpan("5m"))
> rvDaily.eurusd[1:5,  ]
          Positions RealizedVol
 3/12/2001 22:00:00 0.6819636
 3/13/2001 22:00:00 0.7758647
 3/14/2001 22:00:00 0.9544073
 3/15/2001 22:00:00 1.1861121
 3/16/2001 22:00:00 1.0198233
```

A 24-hour day is used to reflect the around-the-clock operation of the Fx markets.

4.2 Statistical Properties of Market Variables

A variety of stylized facts about the statistical properties of high frequency data have been documented in the literature. See Campbell, Lo and MacKinlay (1997) and Tsay (2001) for a review and references. Some of the more notable facts are:

1. Price changes of transaction prices and quotes are discrete valued variables, only taking values of the multiples of tick sizes. In addition, a significant proportion of observations are without price change and the majority of price changes are within the range of limited ticks, e.g. ± 3 ticks.

2. There is tendency for price reversal, or bid-ask bounce in transaction price changes. When transactions randomly occur at the bid or ask quotes, the price changes in transaction prices can exhibit reversal which produces a negative first-order autocorrelation. This negative autocorrelation appears even if the "true" process for price changes lacks autocorrelation.

3. As price and quote changes are discrete, spreads of bid-ask quotes are also discrete and only take a limited set of values. Furthermore, spreads have been found, both for equity markets and FX markets, to cluster on some conventional values.

4. Typically during active trading periods, several trades or quotes may appear to occur at the "same" time and share the same time stamp. That is, it is possible that the mostly used commonly time unit, seconds, is not fine enough to capture the exact trade times during extremely rapid market activity. Consequently, there may be a significant fraction of transactions with zero durations.

S-PLUS and the S+FinMetrics module offer convenient tools for uncovering these properties of high frequency data. The following sections illustrate the use of these tools for uncovering some stylized facts regarding high frequency price changes, spreads and duration

4.2.1 Price Changes

For summarizing time series data, the S+FinMetrics function `summaryStats( )` computes quantile and moment information. For example,

```
> summaryStats(pcTicks.msft)

Sample Quantiles:
 min 1Q median 3Q max
 -40  0      0  0  42

Sample Moments:
       mean   std skewness kurtosis
 -0.0003604 1.165    0.135    181.2

Number of Observations:  97116

> summaryStats(pcTicks.eurusd)

Sample Quantiles:
 min 1Q median 3Q max
 -15 -1      0  1  22

Sample Moments:
      mean  std skewness kurtosis
 -0.002788 1.97 0.003182    5.429

Number of Observations:  126987
```

The summary statistics reveal that both price change variables highly concentrate on 0 and roughly symmetric distributions. However, the high kurtosis values indicate that prices from transaction to transaction may occasionally experience extreme movements.

Since price changes only take values on a discrete grid, their distributional properties may be better illustrated through histogram plots. The following commands, utilizing the S-PLUS function `cut()`, classify the MSFT price changes into seven categories: smaller than or equal to -3 ticks, -2 through 2 ticks, and greater than or equal to 3 ticks, and put the USD/EUR bid quote changes into thirteen categories: smaller than or equal to -6 ticks, -5 through 5 ticks, and greater than or equal to 6 ticks:

```
> pcFactor.msft = cut(seriesData(pcTicks.msft), breaks = c(min(pcTicks.msft),
      -3:2, max(pcTicks.msft)), include.lowest = T, labels = c("<= -3",
      paste(-2:2), ">= 3"), factor.result = T)
> pcFactor.eurusd = cut(seriesData(pcTicks.eurusd),
      breaks = c(min(pcTicks.eurusd),-6:5, max(pcTicks.eurusd)),
      include.lowest = T, labels = c("<= -6", paste(-5:5), ">= 6"),
      factor.result = T)
```

Next, the S-PLUS function `hist.factor( )` can be used to plot the histograms of these categorized price changes. The histogram plots of the two price change variables are shown in Figure 5.

```
> par(mfrow = c(1, 2))
> hist.factor(pcFactor.msft, prob = T, xlab = "Price Changes.MSFT",
      main = "Histogram of MSFT\nTrading Price Changes in Ticks")
> hist.factor(pcFactor.eurusd, prob = T, xlab = "Price Changes.USD/EUR",
      main = "Histogram of USD/EUR\nBid Quote Changes in Ticks")
```

The plots verify the results from the summary statistics. The distributions of two price change variables are essentially symmetric and centered at zero. For the left panel on MSFT data, more than 60% of price changes are 0; about 25% price changes are in 1 tick; around 5% of price changes are in 2 ticks; only 2% of price changes are in 3 ticks or more. In the right panel, USD/EUR bid quote changes are less concentrated around 0; about 25% of quote changes are 0 and changes in 3 ticks still make up about 10% of total observations. One potential reason is that the tick size of USD/EUR, 0.0001, is much smaller than that of MSFT stock price, 1/8 or 0.125, and the price is more likely to change with smaller tick size.

The price reversal property or bid-ask bounce in transaction prices can be illustrated by computing the sample autocorrelations of the transaction price changes of the MSFT stock[6]. The following commands first construct a lead price change series

[6]Show the autocorrelations

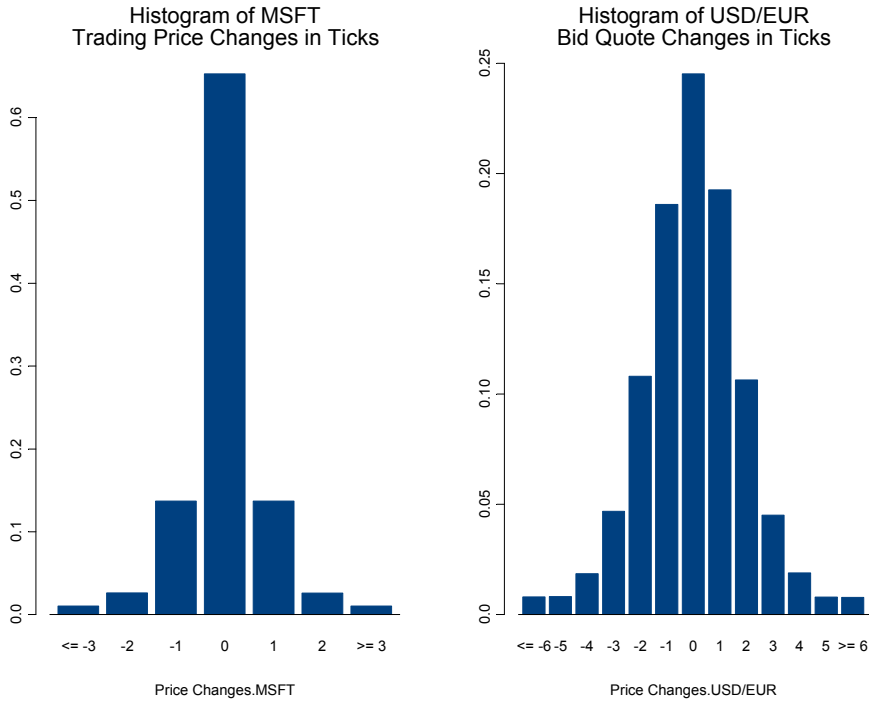

Figure 5: Histograms of intra-day price and quote changes.

and a lagged price change series from the original MSFT price change "timeSeries", so that the two new series have the same length and the i^{th} trades in the lagged price change series and in the lead price change series correspond to the i^{th} and $i+1^{th}$ trades, respectively, in the original MSFT price change series:

```
> pcLag.msft = pc.msft[1:(nrow(pc.msft) - 1),  ]
> pcLead.msft = pc.msft[2:nrow(pc.msft),  ]
```

Each newly constructed price change series is then classified into three categories: "down", "unchanged", and "up", with labels as ("Lead.PC" or "Lag.PC") "-", "0", and "+".

```
> pcLagFactor.msft = cut(seriesData(pcLag.msft), breaks =
      c(min(pcLag.msft), -1:0, max(pcLag.msft)), include.lowest = T,
      labels = paste("Lag.PC", c("-", "0", "+")), factor.result = T)
> pcLeadFactor.msft = cut(seriesData(pcLead.msft), breaks =
      c(min(pcLead.msft), -1:0, max(pcLead.msft)), include.lowest = T,
      labels = paste("Lead.PC", c("-", "0", "+")), factor.result = T)
```

The S-PLUS `table( )` command may then be used to characterize the relationship between successive price changes:

```
> table(pcLagFactor.msft, pcLeadFactor.msft)
         Lead.PC - Lead.PC 0 Lead.PC +
Lag.PC -       787      8058      8020
Lag.PC 0      8449     46869      8077
Lag.PC +      7630      8468       757
```

The resulting table shows the observation counts of each combination of lead and lagged price change categories. The table shows that consecutive up or down movements in price changes is highly unlikely; it is roughly equally likely that the price change will go up or remain unchanged from going down, or vice versa; almost half of successive price changes remain unchanged. These observations support the previous findings of price reversal or bid-ask bounce in transaction prices.

4.2.2 Bid/Ask Spread

The distributional properties of spread variables can also be characterized using `summaryStats( )` and `hist.factor( )`:

```
> summaryStats(spread.msft)

Sample Quantiles:
 min 1Q median 3Q max
   1  1      1  1   3

Sample Moments:
  mean    std skewness kurtosis
 1.198 0.4024    1.589    3.758

Number of Observations:  20203

> summaryStats(spread.eurusd)

Sample Quantiles:
 min 1Q median 3Q max
   1  3      4  5  10

Sample Moments:
  mean   std skewness kurtosis
 3.997 1.616    0.282    3.055

Number of Observations:  126988
```

Unlike price changes, bid-ask spreads are non-negative and have a minimum value of 1 tick. It seems that the spread of MSFT stock quotes is tighter than that of

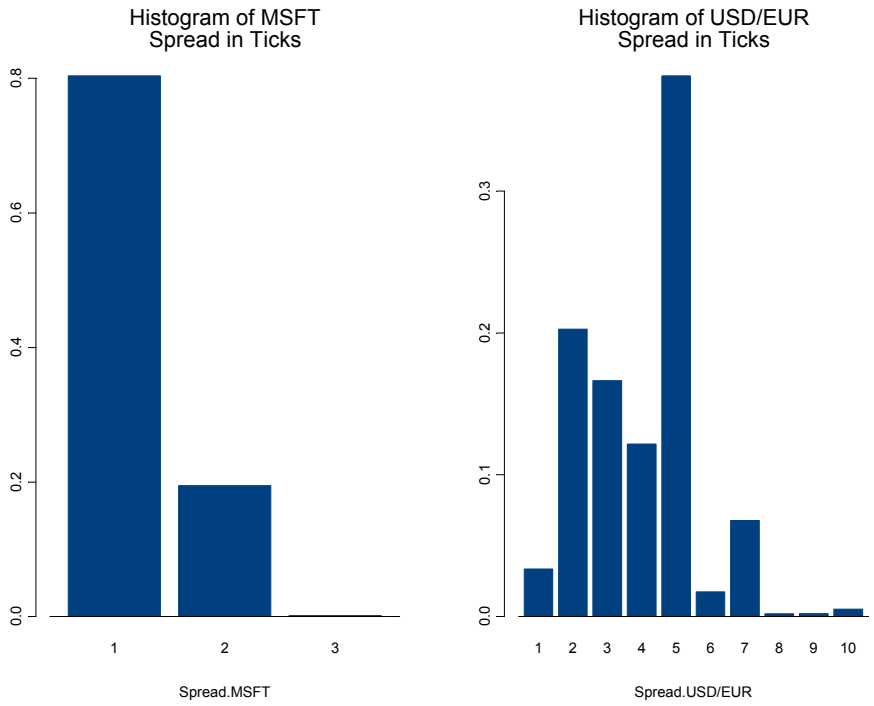

Figure 6: Histograms of intra-day bid-ask spreads.

USD/EUR, although the smaller tick size of USD/EUR quotes should be kept in mind in comparison. The bid-ask spread is discrete valued and its distribution can be visualized using a histogram plot. Since the two spread variables are in a narrow range of values (see min and max values above), no categorization is needed before calling `hist.factor`:

```
> par(mfrow = c(1, 2))
> hist.factor(seriesData(spread.msft), prob = T, xlab = "Spread.MSFT",
    main = "Histogram of MSFT\nSpread in Ticks")
> hist.factor(seriesData(spread.eurusd), prob = T, xlab =
    "Spread.USD/EUR", main = "Histogram of USD/EUR\nSpread in Ticks")
```

The resulting histograms are shown in Figure 6.
The left panel suggests that almost all spreads of MSFT quotes are 1 or 2 ticks with the majority over 80% on 1 tick. In the right panel, the spreads of USD/EUR take more diverse values. However, the spike at 5 with about 40% of total observation indicates that the spreads of USD/EUR quotes tend to cluster on this conventional value, a feature which has been widely documented in the literature.

4.2.3 Duration

Next, consider computing summary statistics and empirical distributions of duration variables:

```
> summaryStats(duration.msftt)

Sample Quantiles:
 min 1Q median 3Q max
   0  0      1  3 132

Sample Moments:
 mean   std skewness kurtosis
 2.65 4.372    5.256    61.76

Number of Observations:  97116

> summaryStats(duration.eurusd)

Sample Quantiles:
 min 1Q median 3Q max
   0  1      1  3 760

Sample Moments:
  mean   std skewness kurtosis
 3.401 10.05    16.82    597.6

Number of Observations:  126987
```

The summary statistics of durations reveal that, although the majority (up to the 75%th quantile for both duration variables) is within 3 seconds, durations between trades and quotes sometimes can be very long. In addition, the durations of MSFT trades are generally shorter than those of USD/EUR quotes, both in terms of mean and extreme values. This is partially due to the fact that stock exchanges have well specified trading hours each day, while the FX markets are essentially decentralized, around the clock markets, and the trading activity during some periods can be quite low; e.g., the period after the quit of North American trading and before the start of Asian trading.

Before plotting the histograms of duration variables, durations are first put into the following 11 categories: 0 through 9 seconds and 10 seconds or more.

```
> durFactor.msftt = cut(seriesData(duration.msftt), left.include=T,
            breaks=c(0:10, max(duration.msftt)), include.lowest =T,
            labels = c(paste(0:9), ">= 10"), factor.result=T)
```

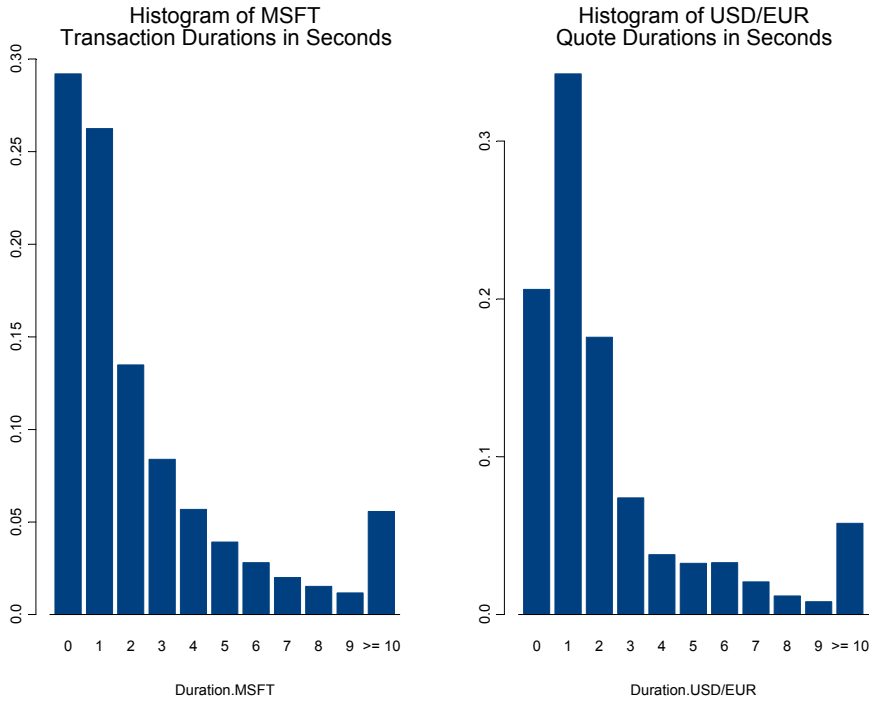

Figure 7: Histograms of intra-day durations.

```
> durFactor.eurusd = cut(seriesData(duration.eurusd), left.include=T,
            breaks=c(0:10, max(duration.eurusd)), include.lowest =T,
            labels = c(paste(0:9), ">= 10"), factor.result=T)
```

Categorized duration data are then plotted in histograms and shown in Figure 7.

```
> par(mfrow = c(1, 2))
> hist.factor(durFactor.msftt, prob = T, xlab = "Duration.MSFT",
      main = "Histogram of MSFT\nTransaction Durations in Seconds")
> hist.factor(durFactor.eurusd, prob = T, xlab = "Duration.USD/EUR",
      main = "Histogram of USD/EUR\nQuote Durations in Seconds")
```

The plots clearly indicate that a significant proportion of duration observations (about 30% for MSFT trades and 20% for USD/EUR quotes) is 0 seconds; i.e., multiple trades or quotes occurring at the same time when the observation frequency is seconds.

4.3 Calendar Patterns in Market Activities

Financial market activities can exhibit periodic calendar patterns. These calendar patterns have been found in the volatility of asset prices, the transaction volumes,

the tick frequency, the duration between ticks, and the spread of bid and ask quotes. One stylized fact about the trading activities of the NYSE is that all market variables follow a reserve J-shaped pattern except the duration, whose pattern has an opposite shape. FX trading activities also follow an intra-day calendar pattern with three peaks corresponding to the business hours of three geographical trading centers (i.e. Asian, European, and American). The strength and prevalence of these calendar patterns in market activities dictate that ignoring or failing to adjust for them may lead to misleading results in the analysis of high-frequency financial data.

This subsection illustrates the characterization of the calendar patterns in three market variables: volatility, duration, and tick frequency. The extracted calendar patterns may be used to adjust market variables before fitting econometric models to high-frequency financial data.

4.3.1 Calendar Patterns in Volatility

For simplification, the illustration here is limited to the estimation of non-model based volatility measures[7]. Specifically, realized volatilities over 5-minute intervals are computed with the HF function `Genr.RealVol()`.

To compute 5-minute realized volatilities in percentages for the MSFT trade price, use

```
> rv5min.msft = Genr.RealVol(ts = log(msftt.ts[, "Price"])*100,
                interv.type = "daily", bound.hours = c("9:30", "16:00"),
                rv.span = timeSpan("5m"), rt.span = timeSpan("1m"))
> rv5min.msft[1:5,  ]
        Positions RealizedVol
 5/1/1997 9:35:00 0.2512298
 5/1/1997 9:40:00 0.6336351
 5/1/1997 9:45:00 0.3982507
 5/1/1997 9:50:00 0.3595108
 5/1/1997 9:55:00 0.5575766
```

By specifying `rt.span = timeSpan("1m")`, the realized volatilities are computed from 1-minute price changes within each 5-minute interval[8]. The resulting "`timeSeries`" object `rv5min.msft` contains 78 5-minute realized volatilities for each of 11 trading days in the sample, 858 observations in total.

One way to identify the periodic calendar pattern in volatility is to examine the autocorrelations of the volatility measures. If a pattern exists, it should be observed in the autocorrelations. The S-PLUS functions `acf( )` and `acf.plot( )` can be used

[7]Examples of model-based volatility measures are GARCH models and stochastic volatility models.

[8]Because of the negative autocorrelations in price changes, the 5-minute realized volatilities estimated this way tend to overestimate the true 5-minute volatility.

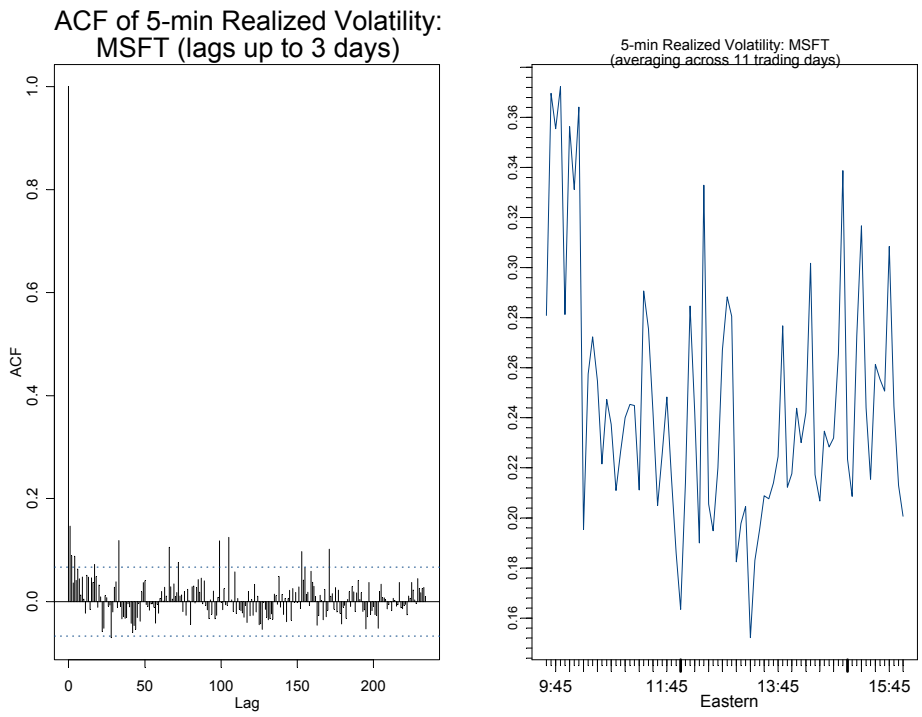

Figure 8: Intra-day seasonalities in squared returns from Microsoft stock.

to compute and plot the autocorrelations of the realized volatilities. The resulting plot is shown in the left panel of Figure 8.

```
> acf.plot(acf(rv5min.msft, lag.max = 234, plot = F), main =
     "ACF of 5-min Realized Volatility:\nMSFT (lags up to 3 days)")
```

The lags are limited to 234, which corresponds to 3 days of 78 intervals each day. A periodic pattern with periodicity 78 can be observed in the autocorrelation plot.

The calendar pattern itself can be uncovered by smoothing or averaging the volatility measures across trading days. For example, the volatility measures at 9:35 for 11 trading days will be averaged to get a smoothed measure of volatility at 9:35. The HF function `SmoothAcrossIntervs( )` can be used to average volatility measures across trading days. The arguments expected by `SmoothAcrossIntervs( )`are

```
> args(SmoothAcrossIntervs)
function(ts, n.subinterv, n.interv, FUN)
```

The `ts` argument specifies the time series to smooth, containing `n.interv` trading intervals (e.g., 11 trading days), each with `n.subinterv` subintervals (e.g. 78 intra-day subintervals); the `FUN` parameter specifies which smoothing function to use (usually

mean or median, although loess.smooth and supsmu may also be used). NAs are allowed in the input series and are removed in the function before smoothing. The output is a smoothed "timeSeries" with the length of n.subinterv. The smoothed calendar patterns from the 5-minute realized volatilities may then be estimated using

```
> rv5min.ave.msft = SmoothAcrossIntervs(ts = rv5min.msft, n.subinterv = 78,
                          n.interv = 11, FUN = mean)
> rv5min.ave.msft@title = "5-min Realized Volatility:
                           MSFT\n(averaging across 11 trading days)"
```

Notice that the title slot of the smoothed "timeSeries" result is assigned the descriptions of the series, which will be used as title in plots. The plotted calendar pattern is shown in the right panel of Figure 8.

```
> plot(rv5min.ave.msft, reference.grid = F,
     x.axis.args = list(format.label = c("%H:%02M", "%Z"),
     time.of.day.style = "24:00"))
```

The pattern plot verifies the previous findings that the price volatility of assets traded on the NYSE exhibit a reverse J-shaped pattern, with the highest volatility at the market opening, reaching the trough around the lunch hour, and picking up slightly just before the market closure.

For the FX data, 5-minute realized volatilities of USD/EUR are computed by applying Genr.RealVol() to the mid quotes with interv.type = "weekly".

```
> rv5min.eurusd = Genr.RealVol(ts = log(mq.eurusd)*100, interv.type = "weekly",
                     bound.hours = c("22:00", "22:00"), rv.span = timeSpan("5m"),
                     rt.span = timeSpan("1m"))
> rv5min.eurusd[1:5,  ]
          Positions RealizedVol
 3/11/2001 22:05:00 0.005360062
 3/11/2001 22:10:00 0.010719263
 3/11/2001 22:15:00 0.030323529
 3/11/2001 22:20:00 0.041539013
 3/11/2001 22:25:00 0.049462498
```

The number of 5-minute intervals is 288 for each of 5 trading days, or 1440 in total.

The periodic calendar pattern in volatility may be verified from the autocorrelation plot in the left panel of Figure 9.

```
> acf.plot(acf(rv5min.eurusd, lag.max = 864, plot = F),
     main = "ACF of 5-min Realized Volatility:\nUSD/EUR (lags up to 3 days)")
```

The plot suggests a periodic pattern in USD/EUR's mid quote realized volatility with the periodicity of 288, the number 5-minute intervals each trading day.

The smoothed calendar pattern can be computed using SmoothAcrossIntervs( )

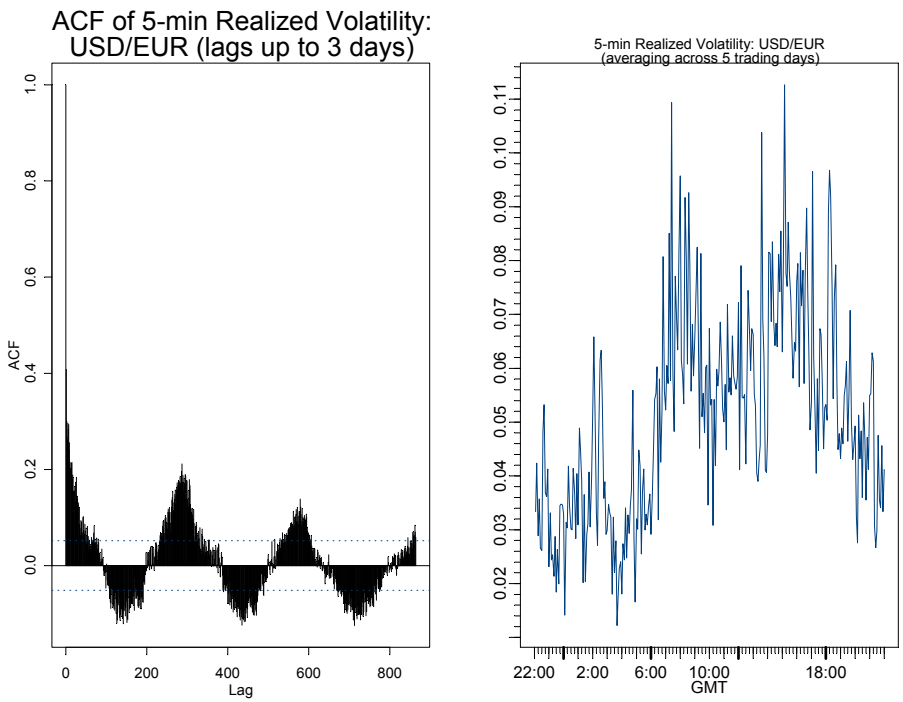

Figure 9: Intra-day seasonalities in squared returns from USD/EUR exchange rate.

```
> rv5min.ave.eurusd = SmoothAcrossIntervs(ts = rv5min.eurusd, n.subinterv = 288,
                          n.interv = 5, FUN = mean)
> rv5min.ave.eurusd@title = "5-min Realized Volatility:
     USD/EUR\n(averaging across 5 trading days)"
> plot(rv5min.ave.eurusd, reference.grid = F,
     x.axis.args = list(format.label = c("%H:%02M", "%Z"),
     time.of.day.style = "24:00"))
```

The plot of the calendar pattern is shown in the right panel of Figure 9.

The plot presents the calendar pattern in the USD/EUR mid quote volatility over a 24-hour trading day, form 22:00 to 22:00 GMT. The calendar pattern starts with a minor spike during the Asian trading hours, and then jumps to a highly elevated new level right after the European trading starts. With the American trading joining in around 13:00 GMT, the calendar pattern experiences the second major spike, followed by a drop and pick-up in volatility as European traders go to and come back from lunch. After European traders slow down around 16:00 GMT, the volatility pattern drops constantly, back to the low level at the beginning of the day.

4.3.2 Calendar Pattern in Duration

With the duration variables constructed in Section 3.1.2, the calendar pattern in durations may be characterized by the mean duration of each 5-minute interval. The HF function `aggregateSeriesHF()`, an extension of S-PLUS function `aggregateSeries()`, can be used to process data within each aggregation block.

The arguments expected by `aggregateSeriesHF()` are:

```
> args(aggregateSeriesHF)
function(ts, interv.type = "daily", bound.hours = c("9:30", "16:00"),
by, k.by, FUN, adj, drop.empty = F, ...)
```

The `ts` argument takes the time series to aggregate; the arguments `interv.type` and `bound.hours` are as introduced before; the `FUN` parameter specifies the function to be used for data in each aggregation block (typically mean or sum, though user-written functions are also accepted); the arguments `by` and `k.by` define each block in terms of the time units and the number of such units; the parameter `adj` takes value of either 0 or 1 so that the aggregated result for each block will be placed at the beginning or ending position of the block; setting the logical parameter `drop.empty` to F will retain in the output the aggregation blocks within trading sessions but with no data in the block to aggregate[9], which ensures that each trading session has the same number of observations in the aggregated series. Other supplied parameters will be passed through to `aggregateSeries()`. For more details, see the online help for `aggregateSeries()`.

The function `aggregateSeriesHF()` extends `aggregateSeries()` in two ways. First, `aggregateSeriesHF()` increases the execution speed by limiting aggregation blocks to those only within trading sessions. For example, the NYSE trading session only runs for six and a half hours out of a 24-hour day, and `aggregateSeriesHF()` takes about one-fourth of the time required by `aggregateSeries()` to process a data set. Furthermore, similar to `Genr.RealVol()`, because `aggregateSeriesHF()` processes data by trading sessions, either daily or weekly, it will generate extra aggregation blocks to make up a multiple number of trading sessions in the resulting aggregated series. For example, if a high-frequency Fx data set starts from Wednesday in the first week and ends on Tuesday in the last week, the aggregated series still starts from the business week starting hour on Sunday in the first week, and ends at the business week ending hour on Friday in the last week. The newly created blocks, together with other blocks without raw data, will have values of 0 or `NA` depending on the aggregation function used.

The mean duration variables can be created using `aggregateSeriesHF( )` with `mean` as the aggregation function.

To compute the 5-minute mean durations for MSFT trades, use

[9]The S-PLUS function `aggregateSeries()` has a bug when `drop.empty = F`. The HF library contains a corrected version of the function.

```
> meanDur5min.msft = aggregateSeriesHF(duration.msftt, interv.type = "daily",
                bound.hours = c("9:30", "16:00"), FUN = mean, by = "minutes",
                k.by = 5, adj = 1, drop.empty = F, together = T)
> meanDur5min.msft[1:5,  ]
        Positions Duration.in.seconds
 5/1/1997 9:35:00 0.7261614
 5/1/1997 9:40:00 1.1434109
 5/1/1997 9:45:00 1.5863874
 5/1/1997 9:50:00 2.0000000
 5/1/1997 9:55:00 1.0340136
```

The calendar pattern in the duration of the MSFT trades is

```
> meanDur5min.ave.msft = SmoothAcrossIntervs(ts = meanDur5min.msft,
                             n.subinterv = 78, n.interv = 11,
                              FUN = mean)
> meanDur5min.ave.msft@title = "5-min Mean Durations:
+ MSFT\n(averaging across 11 trading days)"
```

The autocorrelations and the calendar pattern plot of the 5-minute mean durations of the MSFT trades are shown in Figure 10.

```
> par(mfrow = c(1, 2))
> acf.plot(acf(meanDur5min.msft, lag.max = 234, plot = F), main =
  "ACF of 5-min Mean Durations:\nMSFT (lags up to 3 days)")
> plot(meanDur5min.ave.msft, reference.grid = F, x.axis.args =
       list(format.label = c("%H:%02M", "%Z"),
       time.of.day.style = "24:00"))
```

The strong periodicity in the autocorrelations plot panel shows the existence of a calendar pattern in the 5-minute mean duration of the MSFT trades. In the right panel, durations are quite short with the intense trading in the market opening and closing periods, and are prolonged during lunch hours. The calendar pattern in durations closely mimics the reverse of the calendar pattern in the price volatility of MSFT.

Similarly, the 5-minute mean duration and the calendar pattern in the duration of USD/EUR quotes are created below.

```
> meanDur5min.eurusd = aggregateSeriesHF(duration.eurusd, interv.type = "weekly",
                bound.hours = c("22:00", "22:00"), FUN = mean, by = "minutes",
                k.by = 5, adj = 1, drop.empty = F, together = T)
> meanDur5min.eurusd[1:5,  ]
         Positions Duration.in.seconds
```

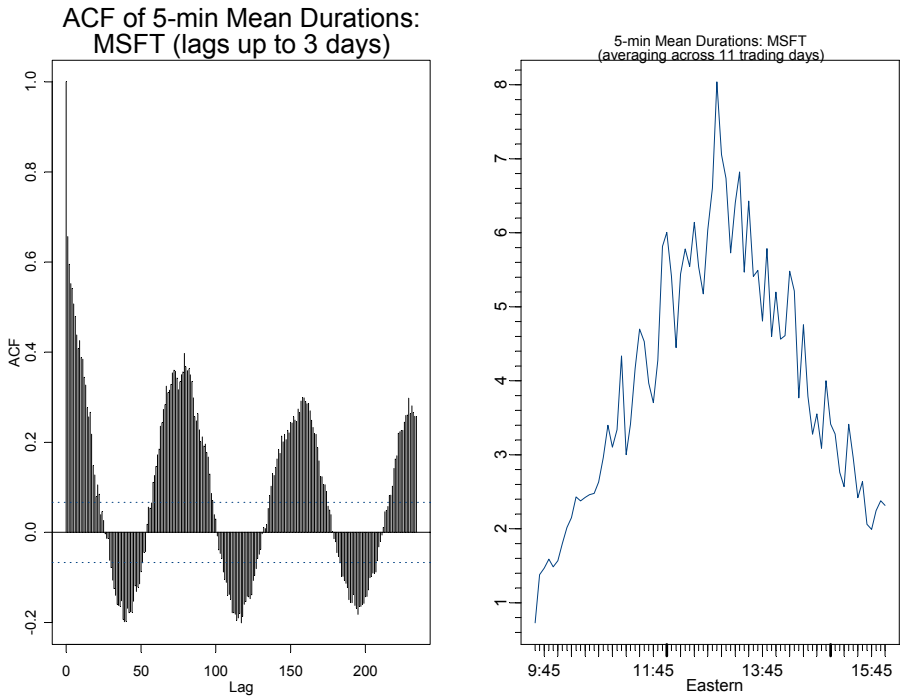

Figure 10: Intra-day seasonalities in Microsoft trade durations.

```
 3/11/2001 22:05:00   2.00000
 3/11/2001 22:10:00 239.50000
 3/11/2001 22:15:00  27.66667
 3/11/2001 22:20:00  41.00000
 3/11/2001 22:25:00  13.64706
> meanDur5min.ave.eurusd = SmoothAcrossIntervs(ts = meanDur5min.eurusd,
                                  n.subinterv = 288, n.interv = 5, FUN = mean)
> meanDur5min.ave.eurusd@title = "5-min Mean Durations:
                                     USD/EUR\n(averaging across 11 trading days)"
```

Because the aggregation function `mean` generates `NA`s for those blocks with no raw data to aggregate[10] and the option `drop.empty = F` keeps those aggregation blocks in the mean duration variable, those `NA`s have to be removed before computing the autocorrelations[11]. The `NA`s are flagged using the S-PLUS function `is.na( )`

```
> na.idx = is.na(seriesData(meanDur5min.eurusd))
```

[10] `sum()` generates 0s for those blocks.

[11] The mean durations for MSFT trades happen to be free of `NA`s.

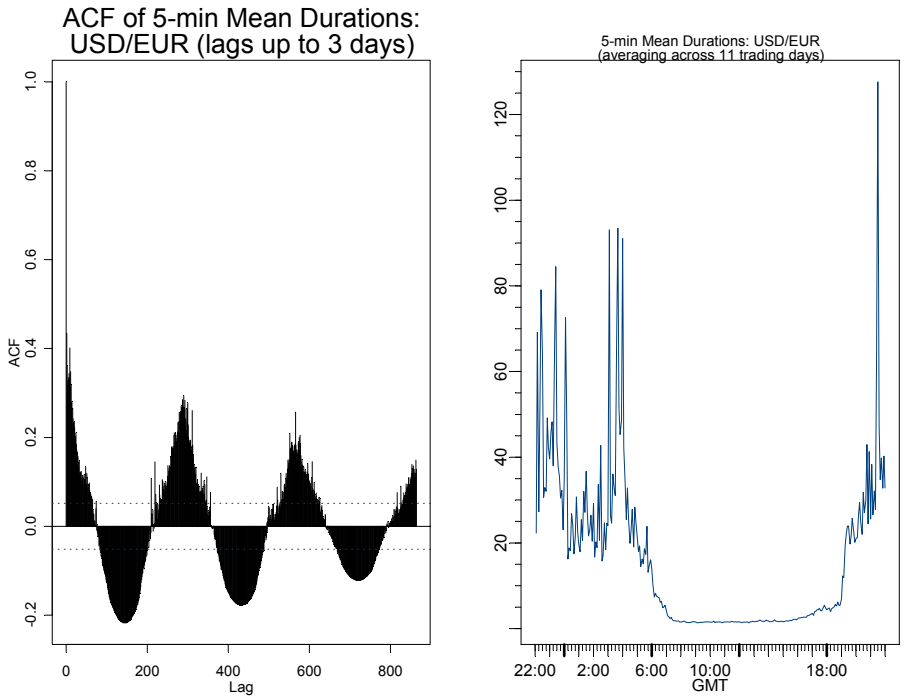

Figure 11: Intra-day seasonalities in USD/EUR quote durations.

The autocorrelations and the calendar pattern in the duration of the USD/EUR quotes are shown in Figure 11.

```
> par(mfrow = c(1, 2))
> acf.plot(acf(meanDur5min.eurusd[!na.idx, ], lag.max = 864,
    plot = F), main = "ACF of 5-min Mean Durations:\n
     USD/EUR (lags up to 3 days)")
> plot(meanDur5min.ave.eurusd, reference.grid = F,
    x.axis.args = list(format.label = c("%H:%02M", "%Z"),
    time.of.day.style = "24:00"))
```

In addition to the periodicity shown in the autocorrelations plot of the mean duration, the calendar pattern plot shows that the duration of the USD/EUR quotes are high or the market activities are low in the early trading day. With Asian traders joining the market, mainly from Tokyo, around 23:00 GMT or 8:00 JST, the mean duration drops to a low level until the lunch time of Tokyo. Afterwards, the mean duration returns to the morning active level and is further shortened soon later with the start of European trading. The rest of the trading day is dominated by the European and American trading activities and the mean duration remains at its minimum before it jumps up again in the late trading day.

4.3.3 Calendar Pattern in Tick Frequency

Tick frequency may be measured by the number of tick observations in some short time intervals. The counts of trades can be computed with an auxiliary time series with the identical time date positions as the original trades series but with the value of 1 for all observations.

For example, the auxiliary time series for the MSFT trades can be created using the S-PLUS constructor function for "timeSeries" `timeSeries( )`

```
> aux.msft = timeSeries(data.frame(Ticks = rep(1, nrow(msftt.ts))),
                        pos = msftt.ts@positions)
```

Then the number of trades in each 5-minute intervals is counted using aggregateSeriesHF( ) with `sum` as the aggregation function.

```
> trades5min.msft = aggregateSeriesHF(aux.msft, interv.type = "daily",
              bound.hours = c("9:30", "16:00"), FUN = sum, by = "minutes",
              k.by = 5, adj = 1, drop.empty = F, together = T)
> trades5min.msft[1:5,  ]
        Positions Ticks
 5/1/1997 9:35:00 410
 5/1/1997 9:40:00 258
 5/1/1997 9:45:00 191
 5/1/1997 9:50:00 149
 5/1/1997 9:55:00 294
```

Averaging the 5-minute trades frequency across trading days gives out the calendar pattern in tick frequency.

```
> trades5min.ave.msft = SmoothAcrossIntervs(ts = trades5min.msft,
                         n.subinterv = 78, n.interv = 11, FUN = mean)
> trades5min.ave.msft@title = "Number of Trades in 5-min Intervs:
                         MSFT\n(averaging across 11 trading days)"
```

The autocorrelations of the 5-minute trades frequency and the calendar pattern are plotted in Figure 12.

```
> par(mfrow = c(1,2))
> acf.plot(acf(trades5min.msft, lag.max = 234, plot = F),
   main = "ACF of Number of Trades in 5-min Intervs:\n
   MSFT (lags up to 3 days)")
> plot(trades5min.ave.msft, reference.grid = F,
     x.axis.args = list(format.label = c("%H:%02M", "%Z"),
                        time.of.day.style = "24:00"))
```

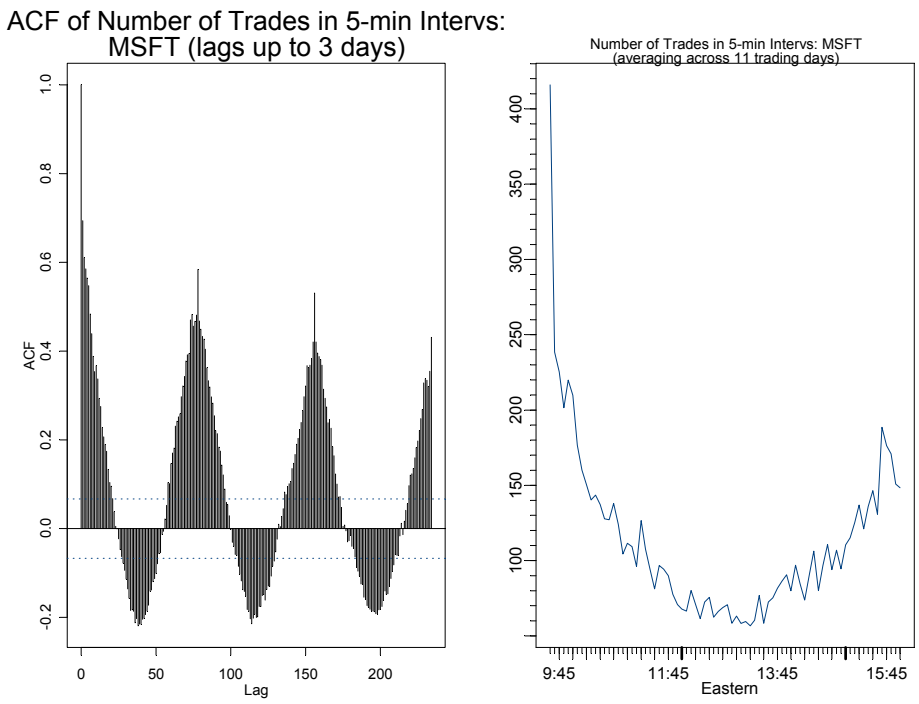

Figure 12: Intra-day seasonalities in Microsoft trades.

The calendar pattern in the frequency of the MSFT trades is very close to the pattern in the volatility of the MSFT transaction prices, a reversed J-shaped pattern. Right after the market opening, the frequency can reach more than 400 trades in an interval of 5 minutes or 300 seconds. It explains why there are many 0-second durations in the high-frequency data.

The calendar pattern in the quote frequency of USD/EUR can be characterized similarly.

```
> aux.eurusd = timeSeries(data.frame(Ticks = rep(1, nrow(eurusd.ts))),
            pos = eurusd.ts@positions)
> quotes5min.eurusd = aggregateSeriesHF(aux.eurusd, interv.type = "weekly",
              bound.hours = c("22:00", "22:00"), FUN = sum, by = "minutes",
              k.by = 5, adj = 1, drop.empty = F, together = T)
> quotes5min.eurusd[1:5,  ]
          Positions Ticks
 3/11/2001 22:05:00  2
 3/11/2001 22:10:00  2
 3/11/2001 22:15:00  9
```

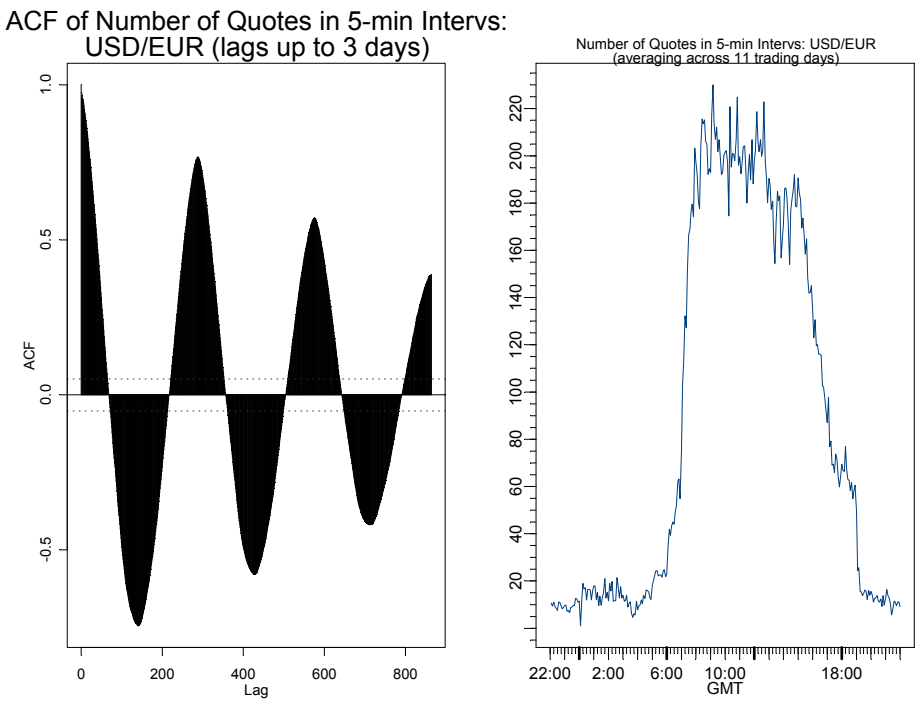

Figure 13: Intra-day seasonalities in USD/EUR quotes.

```
 3/11/2001 22:20:00  9
 3/11/2001 22:25:00 17
> quotes5min.ave.eurusd = SmoothAcrossIntervs(ts = quotes5min.eurusd,
                           n.subinterv = 288, n.interv = 5, FUN = mean)
> quotes5min.ave.eurusd@title = "Number of Quotes in 5-min Intervs:
                          USD/EUR\n(averaging across 11 trading days)"
```

The autocorrelations and the calendar pattern of the quote frequency of USD/EUR are plotted in Figure 13.

```
> par(mfrow = c(1,2))
> acf.plot(acf(quotes5min.eurusd, lag.max = 864, plot = F),
     main = "ACF of Number of Quotes in 5-min Intervs:\n
     USD/EUR (lags up to 3 days)")
> plot(quotes5min.ave.eurusd, reference.grid = F,
     x.axis.args = list(format.label = c("%H:%02M", "%Z"),
                        time.of.day.style = "24:00"))
```

The calendar pattern in the quote frequency is almost a replication of the pattern of the mid quote volatility.

References

[1] Andersen, T. G., 2000, "Some Reflections on Analysis of High-frequency Data", *Journal of Business and Economic Statistics*, 18, 146-153

[2] Andersen, T. G., Bollerslev, T., Diebold, F. X., and Ebens, H., 2001, "The Distribution of Realized Stock Return Volatility", *Journal of Financial Economics*, 61, 43-76

[3] Campbell, J. Y., Lo, A. W., and MacKinlay, A. C., 1997, *The Econometrics of Financial Markets*, Princeton University Press, New Jersey.

[4] Dacarogna, M. M., Gençay, R., Müller, U. A., Olsen, R. B., and Pictet, O. V., 2001, *An Introduction to High-Frequency Finance*, Academic Press

[5] Easley, D., and O'Hara, M, 1992, "Time and the Process of Security Price Adjustment," *Journal of Finance*, 47, 577-605

[6] Engle, R. F., and Russell, J. R., 1998, "Autoregressive Conditional Duration: A New Model for Irregularly Spaced Transaction Data", *Econometrica*, 66, 1127-1162

[7] Engle, R. F., 2000, "The Econometrics of Ultra High Frequency Data", *Econometrica*, 68, 1-22

[8] Ghysels, E., 2000, "Some Econometric Recipes for High-Frequency Data Cooking", *Journal of Business and Economic Statistics*, 18, 154-163

[9] Goodhart, C.A.E., and O'Hara, M, 1997, "High Frequency Data in Financial Markets: Issues and Applications", *Journal of Empirical Finance*, 4, 73-114

[10] Gouriéroux, C., and Jasiak, J., 2001, *Financial Econometrics: Problems, Models, and Methods*, Princeton University Press, New Jersey

[11] Hausman, J. A., Lo, A. W., and MacKinlay, A. C., 1992, "An Ordered Probit Analysis of Transaction Stock Prices", *Journal of Financial Economics*, 31, 319-379

[12] Lee, and Ready, 1991, "Inferring Trade Direction from Intraday Data," Journal of Finance, 46, 733-746.

[13] Lyons, R., 2001, *The Microstructure Approach to Exchange Rates*, MIT Press

[14] Stoll, H., and Whaley, R., 1990, "Stock Market Structure and Volatility," *Review of Financial Studies*, 3, 37-71

[15] Tsay, R. S., 2001, *Analysis of Financial Time Series*, John Wiley & Sons, Inc

[16] Wood, R. A., 2000, "Market Microstructure Research Databases: History and Projections", *Journal of Business and Economic Statistics*, 18, 140-145