Working with Time Series Data in R

Eric Zivot

Department of Economics, University of Washington

October 21, 2008

Preliminary and Incomplete

Importing Comma Separated Value (.csv) Data into R

When you download asset price data from finance.yahoo.com, it gets saved in a comma separated value (`.csv`) file. This is a text file where each value is separated (delimited) by a comma ",". This type of file is easily read into both Excel and R. Excel opens `.csv` files directly. The easiest way import data in .csv files into R is to use the R function `read.csv()`.

To illustrate, consider the monthly adjusted closing price data on Starbucks (SBUX) and Microsoft (MSFT) in the files `sbuxPrices.csv` and `msftPrices.csv`. These file are available on the class homework page. The first 5 rows of the `sbuxPrices.csv` file are

```
Date,Adj Close
3/31/1993,1.19
4/1/1993,1.21
5/3/1993,1.5
6/1/1993,1.53
```

Notice that the first row contains the names of the columns, the date information is in the first column with the format m/d/YYYY, and the adjusted closing price (close price adjusted for stock splits and dividends) is in the second column. Assume that this file is located in the directory `C:\classes\econ424\fall2008`. To read the data into R use

```
>  sbux.df = read.csv("C:/classes/econ424/fall2008/sbuxPrices.csv",
+                      header = TRUE, stringsAsFactors = FALSE)
```

Now do the same for the Microsoft data.

Remarks:

1. Note how the directory structure is specified using forward slashes "/". Alternatively, you can use double back slashes "\\" instead of a single forward slash "/".
2. The argument `header = TRUE` indicates that the column names are in the first row of the file

3. The argument `stringsAsFactors = FALSE` tells the function to treat the date information as character data and not to convert it to a factor variable.

The SBUX data is imported into `sbux.df` which is an object of class `data.frame`

```
> class(sbux.df)

[1] "data.frame"
```

A `data.frame` object is a rectangular data object with the data in columns. The column names are

```
> colnames(sbux.df)

[1] "Date"      "Adj.Close"
```

And the first 6 rows are

```
> head(sbux.df)

       Date Adj.Close
1 3/31/1993      1.19
2  4/1/1993      1.21
3  5/3/1993      1.50
4  6/1/1993      1.53
5  7/1/1993      1.48
6  8/2/1993      1.52
```

The data in the columns can be of different types. The `Date` column contains the date information as character data and the `Adj.Close` column contains the adjusted price data as numeric data. Notice that the dates are not all monthly closing dates but that the adjusted closing prices are for the last trading day of the month.

```
> class(sbux.df$Date)

[1] "character"

> class(sbux.df$Adj.Close)

[1] "numeric"
```

Representing time series data in a `data.frame` object has the disadvantage that the date index information cannot be efficiently used. You cannot subset observations based on the date index. You must subset by observation number. For example, to extract the prices between March, 1994 and March, 1995 you must use

```
> which(sbux.df$Date == "3/1/1994")
[1] 13

> which(sbux.df$Date == "3/1/1995")
[1] 25
```

```
> sbux.df[13:25,]
         Date Adj.Close
13  3/1/1994      1.52
14  4/4/1994      1.86
…
25  3/1/1995      1.50
```

In addition, the default plot method for `data.frame` objects do not utilize the date information for the x-axis. For example, the following call to `plot()` creates an error

```
> plot(sbux.df$Date, sbux.df$Adj.Close, type="l")
```

Representing Regularly Spaced Data as ts Objects

Regularly spaced time series data, data that are separated by a fixed interval of time, may be represented as objects of class `ts`. Such data are typically observed monthly, quarterly or annually. `ts` objects are created using the `ts()` constructor function (base R). For example,

```
> sbux.ts = ts(data=sbux.df$Adj.Close, frequency = 12,
             start=c(1993,3), end=c(2008,3))
> class(sbux.ts)
[1] "ts"

> msft.ts = ts(data=msft.df$Adj.Close, frequency = 12,
             start=c(1993,3), end=c(2008,3))
```

The argument `frequency = 12` specifies that that prices are sampled monthly. The starting and ending months are specified as a two element vector with the first element giving the year and the second element giving the month. When printed, `ts` objects show the dates associated with the observations.

```
> sbux.ts
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
1993             1.19  1.21  1.50  1.53  1.48  1.52  1.71  1.67  1.39
…
```

The functions `start()` and `end()` show the first and last dates associated with the data

```
> start(sbux.ts)
[1] 1993    3
> end(sbux.ts)
[1] 2008    3
```

The `time()` function extracts the time index as a `ts` object

```
> time(sbux.ts)
          Jan      Feb      Mar      Apr      May      Jun  …
1993                    1993.167 1993.250 1993.333 1993.417 …
```

The frequency per period and time interval between observations of a `ts` object may be extracted using

```
> frequency(sbux.ts)
[1] 12

> deltat(sbux.ts)
[1] 0.08333333
```

However, subsetting a `ts` object produces a numeric object

```
> tmp = sbux.ts[1:5]
> class(tmp)
[1] "numeric"

> tmp
[1] 1.19 1.21 1.50 1.53 1.48
```

To subset a `ts` object and preserve the date information use the `window()` function

```
> tmp = window(sbux.ts, start=c(1993, 3), end=c(1993,8))
> class(tmp)
[1] "ts"

> tmp
      Mar  Apr  May  Jun  Jul  Aug
1993 1.19 1.21 1.50 1.53 1.48 1.52
```

The arguments `start=c(1993, 3)` and `end=c(1993,8)` specify the beginning and ending dates of the window.

Merging ts objects

To combine the ts objects sbux.ts and msft.ts into a single object use the cbind() function

```
> sbuxmsft.ts = cbind(sbux.ts, msft.ts)
> class(sbuxmsft.ts)
[1] "mts" "ts"
```

Since `sbuxmsft.ts` contains two `ts` objects it is assigned the additional class `mts` (multiple time series). The first five rows are

```
> window(sbuxmsft.ts, start=c(1993, 3), end=c(1993,7))
         sbux.ts msft.ts
Mar 1993    1.19    2.43
Apr 1993    1.21    2.25
May 1993    1.50    2.44
Jun 1993    1.53    2.32
Jul 1993    1.48    1.95
```

Plotting `ts` objects

`ts` objects have their own plot method (`plot.ts`)

```
> plot(sbux.ts, col="blue", lwd=2, ylab="Adjusted close",
+      main="Monthly closing price of SBUX")
```

Which produces the plot in Figure 1. To plot a subset of the data use the `window()` function inside of `plot()`

```
> plot(window(sbux.ts, start=c(2000,3), end=c(2008,3)),
+      ylab="Adjusted close",col="blue", lwd=2,
+     main="Monthly closing price of SBUX")
```

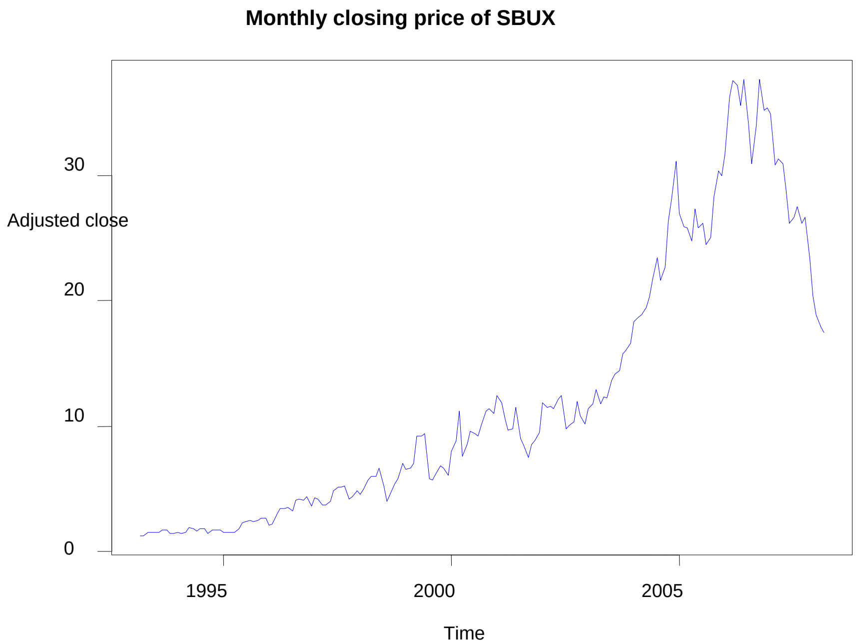


Figure 1 Plot created with `plot.ts()`

For `ts` objects with multiple columns (`mts` objects), two types of plots can be created. The first type, illustrated in Figure 2, puts each series in a separate panel

```
> plot(sbuxmsft.ts)
```

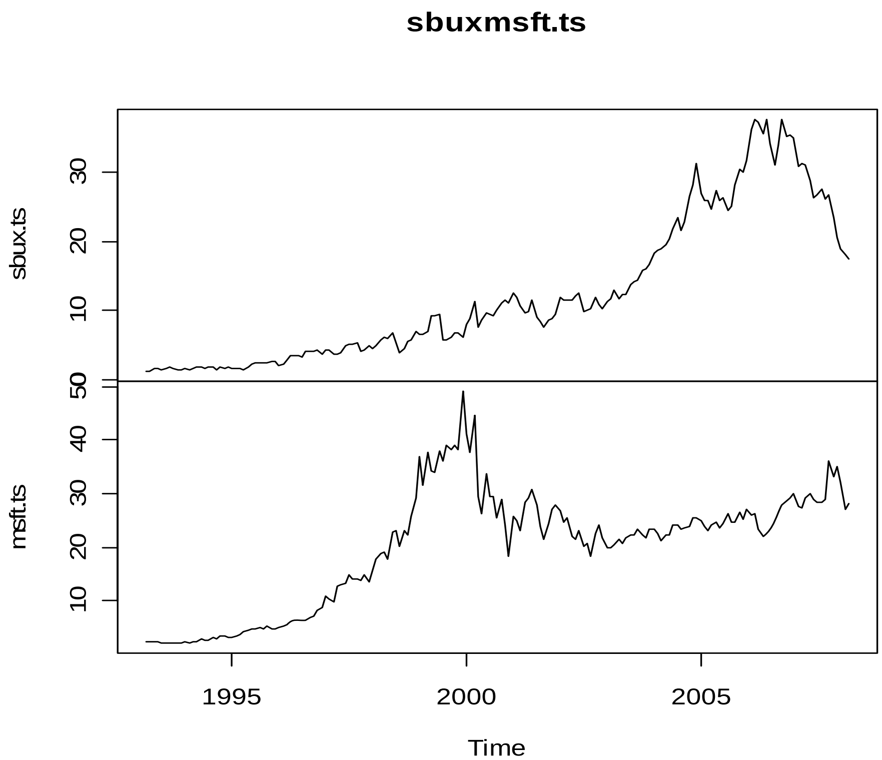


Figure 2 Multiple time series plot

The second type, shown in Figure 3, puts all series on the same plot

```
> plot(sbuxmsft.ts, plot.type="single",
+      main="Monthly closing prices on SBUX and MSFT",
+      ylab="Adjusted close price",
+      col=c("blue", "red"), lty=1:2)
> legend(1995, 45, legend=c("SBUX","MSFT"), col=c("blue", "red"),
+        lty=1:2)
```

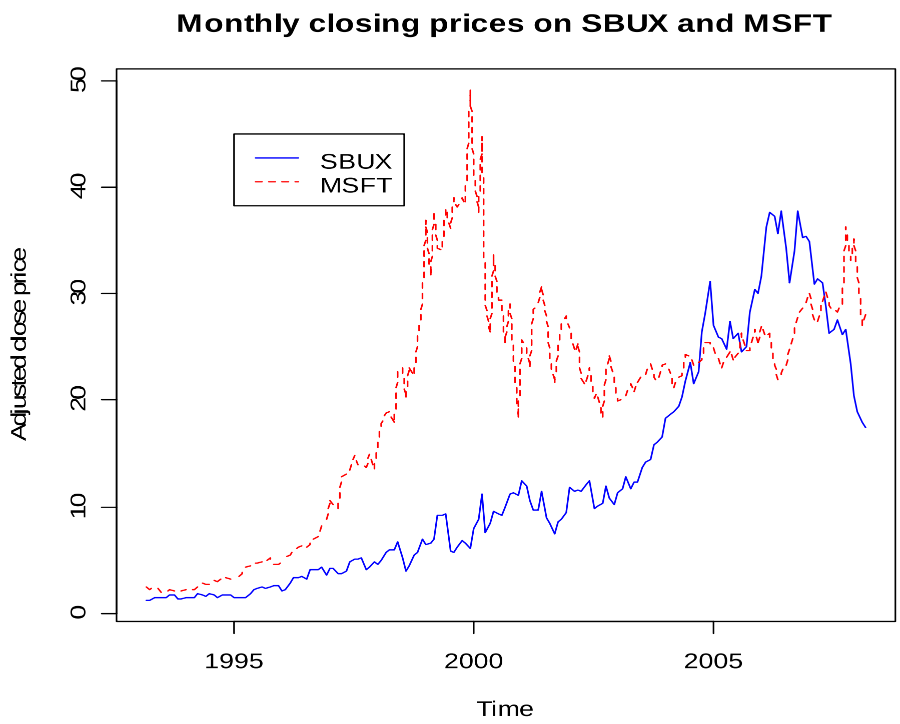


Figure 3 Multiple time series plot

Manipulating `ts` objects and computing returns

Some common manipulations of time series data involve lags and differences using the functions `lag()` and `diff()`. For example, to lag the price data in `sbux.ts` by one time period use

```
>  lag(sbux.ts)
```

To lag the price data by 12 periods use

```
>  lag(sbux.ts, k=12)
```

Notice what happens when you combine a `ts` object with its lag

```
> cbind(sbux.ts, lag(sbux.ts))
         sbux.ts lag(sbux.ts)
Feb 1993      NA         1.19
Mar 1993    1.19         1.21
Apr 1993    1.21         1.50
May 1993    1.50         1.53
Jun 1993    1.53         1.48
```

The `lag()` function shifts the time index back by an amount k. To shift the time index forward set k to a negative number

```
> lag(sbux.ts, k=-1)
> lag(sbux.ts, k=-12)
> cbind(sbux.ts, lag(sbux.ts, k=-1))
         sbux.ts lag(sbux.ts, k = -1)
Mar 1993    1.19                   NA
Apr 1993    1.21                 1.19
May 1993    1.50                 1.21
Jun 1993    1.53                 1.50
Jul 1993    1.48                 1.53
```

To compute the first difference in prices use

```
> diff(sbux.ts)
```

Notice that application of `diff()` is equivalent to

```
> sbux.ts – lag(sbux.ts, k=-1)
```

To compute a 12 lag difference (annual difference for monthly data) use

```
> diff(sbux.ts, lag=12)
```

which is equivalent to using

```
> sbux.ts – lag(sbux.ts, k=-12)
```

Notice what happens when you combine a `ts` object with its first difference

```
> cbind(sbux.ts, diff(sbux.ts))
         sbux.ts diff(sbux.ts)
Mar 1993    1.19            NA
Apr 1993    1.21          0.02
May 1993    1.50          0.29
Jun 1993    1.53          0.03
Jul 1993    1.48         -0.05
```

You can use the `diff()` and `lag()` functions together to compute the simple one period return

```
> sbuxRetSimple.ts = diff(sbux.ts)/lag(sbux.ts, k=-1)
> msftRetSimple.ts = diff(msft.ts)/lag(msft.ts, k=-1)
> window(cbind(sbuxRetSimple.ts, msftRetSimple.ts),
+        start=c(1993,4), end=c(1993,7))
         sbuxRetSimple.ts msftRetSimple.ts
Apr 1993       0.01680672      -0.07407407
May 1993       0.23966942       0.08444444
Jun 1993       0.02000000      -0.04918033
Jul 1993      -0.03267974      -0.15948276
```

Similarly, to compute the 12-period simple return use

```
> diff(sbux.ts, lag=12)/lag(sbux.ts, k=-12)
```

You can use the `log()` and `diff()` functions together to compute continuously compounded returns

```
> sbuxRet.ts = diff(log(sbux.ts))
```

```
> msftRet.ts = diff(log(msft.ts))
```

```
> window(cbind(sbuxRet.ts, msftRet.ts), start=c(1993,4),
+         end=c(1993,7))
          sbuxRet.ts  msftRet.ts
Apr 1993  0.01666705 -0.07696104
May 1993  0.21484475  0.08106782
Jun 1993  0.01980263 -0.05043085
Jul 1993 -0.03322565 -0.17373781
```

To compute the 12-period continuously compounded return use

```
> diff(log(sbux.ts), lag=12)
```

Representing Time Series Data as `zoo` objects

The `ts` class is rather limited, especially for representing financial data that is not regularly spaced. For example, the `ts` class cannot be used to represent daily financial data because such data are only observed on business days. That is, a business day time clock generally runs from Monday to Friday skipping the weekends. So data are equally spaced in time within the week but the spacing between Friday and Monday is different. This type of irregular spacing cannot be represented using the `ts` class.

A very flexible time series class is `zoo` (Zeileis' ordered observations) created by Achim Zeileis and Gabor Grothendieck and available in the package zoo on CRAN. The `zoo` class was designed to handle time series data with an arbitrary ordered time index. This index could be a regularly spaced sequence of dates, an irregularly spaced sequence of dates, or a numeric index. A zoo object essentially attaches date information with data.

Install and load the package zoo into R before completing the examples in the next sections.

```
> library(zoo)
```

Creating a time index

There are several ways to represent a time index or sequence of dates in R. Table 1 summarizes the main time index classes available in R

Table 1 Date index classes in R

Class	Package	Description

Date	Base	Represent calendar dates as the number of days since 1970-01-01
POSIXct	Base	Represent calendar dates as the (signed) number of seconds since the beginning of 1970 as a numeric vector. Supports various timezone specifications (e.g. GMT, PST, EST etc.)
yearmon	zoo	Represent monthly data. Internally it holds the data as year plus 0 for January, 1/12 for February, 2/12 for March and so on in order that its internal representation is the same as `ts` class with `frequency = 12`.
yearqtr	zoo	Represent quarterly data. Internally it holds the data as year plus 0 for Quarter 1, 1/4 for Quarter 2 and so on in order that its internal representation is the same as `ts` class with `frequency = 4`.

The Date class (Base R)

Use the Date class to represent a time index only involving dates but not times. The Date class represents dates internally as the number of days since January 1, 1970. You create Date objects from a character string representing a date using the `as.Date()` function. The default format is "`YYYY/m/d`" or "`YYYY-m-d`"" where YYYY represents the four digit year, `m` represents the month digit and `d` represents the day digit. For example,

```
> my.date = as.Date("1970/1/1")
> my.date
[1] "1970-01-01"

> class(my.date)
[1] "Date"

> as.numeric(my.date)
[1] 0
```

Use the `format` argument to specify the input format of the date if it is not in the default format

```
> as.Date("1/1/1970", format="%m/%d/%Y")
[1] "1970-01-01"

> as.Date("January 1, 1970", format="%B %d, %Y")
[1] "1970-01-01"

> as.Date("01JAN70", format="%d%b%y")
[1] "1970-01-01"
```

Notice that the output format is always in the form "`YYYY-m-d`". To change the displayed output format of a date use the `format()` function

```
> my.date
[1] "1970-01-01"

> format(my.date, "%m/%d/%Y")
```

```
[1] "01/01/1970"
```

Table 2 gives the standard date format codes

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (abbreviated)
%B	Month (full name)
%y	Year (2 digit)
%Y	Year (4 digit)

Table 2. Format codes for dates

To convert an integer variable to a Date object use the `class()` function

```
> my.date = 0
> class(my.date) = "Date"
> my.date
[1] "1970-01-01"
```

The `weekdays()`, `months()` and `quarters()` functions can be used to extract specific components of Date objects

```
> weekdays(my.date)
[1] "Thursday"

> months(my.date)
[1] "January"

> quarters(my.date)
[1] "Q1"
```

To create a sequence of `Date` objects starting in March, 1993 and ending in March, 2003 use

```
> td = seq(as.Date("1993/3/1"), as.Date("2003/3/1"), "months")

> class(td)
[1] "Date"

> head(td)
[1] "1993-03-01" "1993-04-01" "1993-05-01" "1993-06-01" "1993-07-01"
[6] "1993-08-01"
```

Having a numeric representation for dates allows for some simple date arithmetic. For example,

```
> td[2] - td[1]

Time difference of 31 days
```

creates a difftime object and shows that there are 31 days between April 1, 1993 and March 1, 1993

The POSIXct class (Base R)

To be completed.

The yearmon class (Package zoo)

To be completed.

The yearmon class (Package zoo)

To be completed.

The timeDate class (Package fCalendar)

To be completed.

Creating a zoo object

To create a `zoo` object one needs a time index and data. The time index must have the same number of rows as the data object and can be any vector containing ordered observations. Typically, the time index is an object of class `Date`, `POSIXct`, `yearmon`, `yearqtr` or `timeDate`.

Consider creating `zoo` objects from the monthly information in the `data.frame` objects `sbux.df` and `msft.df`. First, create a time index of class `Date` starting in March, 1993 and ending in March, 2003

```
> td = seq(as.Date("1993/3/1"), as.Date("2003/3/1"), "months")
> class(td)
[1] "Date"

> head(td)
[1] "1993-03-01" "1993-04-01" "1993-05-01" "1993-06-01" "1993-07-01"
[6] "1993-08-01"
```

Alternative, create a time index by coercing the character date strings in the Date column of sbux.df to objects of class date

```
> head(td2)
[1] "1993-03-31" "1993-04-01" "1993-05-03" "1993-06-01" "1993-07-01"
[6] "1993-08-02"
```

Now that we have a time index, we can create the `zoo` object by combining the time index with numeric data

```
> sbux.z = zoo(x=sbux.df$Adj.Close, order.by=td)
> msft.z = zoo(x=msft.df$Adj.Close, order.by=td)

> class(sbux.z)
[1] "zoo"

> str(sbux.z)
'zoo' series from 1993-03-01 to 2003-03-01
  Data: num [1:121] 1.19 1.21 1.5 1.53 1.48 1.52 1.71 1.67 1.39 1.39
...
```

```
  Index: Class 'Date'  num [1:121] 8460 8491 8521 8552 8582 ...

> head(sbux.z)
1993-03-01 1993-04-01 1993-05-01 1993-06-01 1993-07-01 1993-08-01
      1.19       1.21       1.50       1.53       1.48       1.52
```

The time index and data can be extracted using the `index()` and `coredata()` functions

```
> index(sbux.z)
  [1] "1993-03-01" "1993-04-01" "1993-05-01" "1993-06-01" "1993-07-01"
…
> coredata(sbux.z)
  [1]  1.19  1.21  1.50  1.53  1.48  1.52  1.71  1.67  1.39  1.39
…
```

The `start()` and `end()` functions also work for `zoo` objects

```
> start(sbux.z)
[1] "1993-03-01"

> end(sbux.z)
[1] "2003-03-01"
```

An advantage of `zoo` objects is that subsetting can be done with the time index. For example, to extract the data for March 1993 and March 2004 use

```
> sbux.z[as.Date(c("2003/3/1", "2004/3/1"))]
2003-03-01 2004-03-01
     12.88      18.93
```

The `window()` function also works with `zoo` objects

```
> window(sbux.z, start=as.Date("2003/3/1"), end=as.Date("2004/3/1"))

2003-03-01 2003-04-01 2003-05-01 2003-06-01 2003-07-01 2003-08-01
…
2003-10-01 2003-11-01 2003-12-01 2004-01-01 2004-02-01 2004-03-01
     15.80      16.08      16.58      18.31      18.70      18.93
```

Creating lags and differences works the same way for `zoo` objects as it does for `ts` objects.

Merging zoo objects

To combine the zoo objects `sbux.z` and `msft.z` into a single object use either the `cbind()` or the `merge()` functions

```
> sbuxmsft.z = cbind(sbux.z, msft.z)
```

```
> class(sbuxmsft.z)
[1] "zoo"

> head(sbuxmsft.z)
           sbux.z msft.z
1993-03-01   1.19   2.43
1993-04-01   1.21   2.25
1993-05-01   1.50   2.44
1993-06-01   1.53   2.32
1993-07-01   1.48   1.95
1993-08-01   1.52   1.98
```

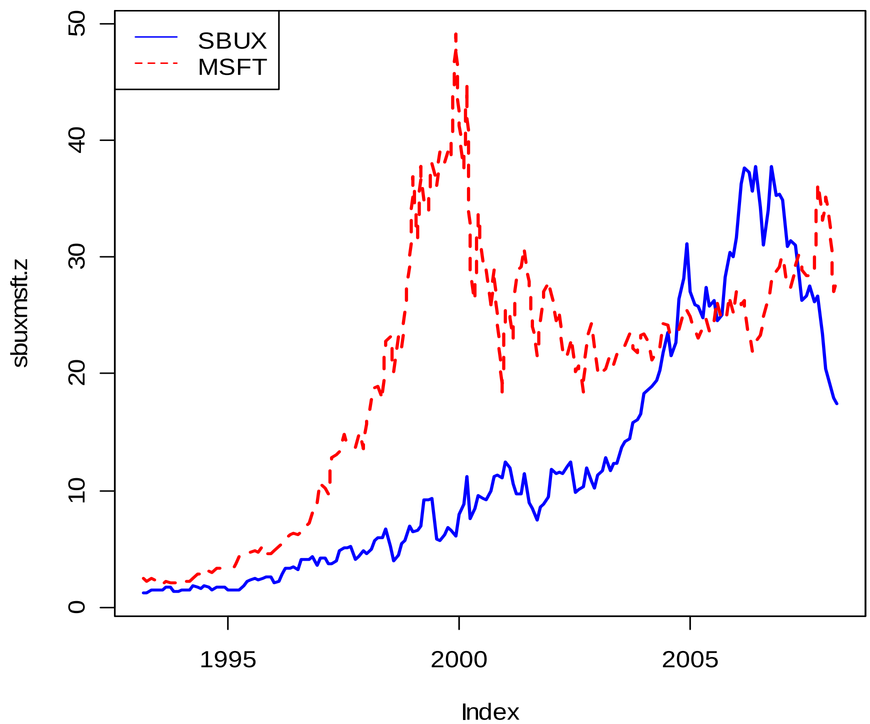
Use `cbind()` when combining `zoo` objects that have the same time index, and use `merge()` when the objects have different time indices. Note, you can only combine `zoo` objects for which the time index is of a common class (e.g. all time indices are `Date` objects).

Plotting zoo objects

The `plot()` function can be used to plot `zoo` objects, and follows a syntax similar to the `plot()` function used for plotting `ts` objects. The following commands produce the plot illustrated in Figure 4

```
> # plot one series at a time and add a legend
> plot(sbux.z, col="blue", lty=1, lwd=2, ylim=c(0,50))
> lines(msft.z, col="red", lty=2, lwd=2)
> legend(x="topleft", legend=c("SBUX","MSFT"), col=c("blue","red"),
+        lty=1:2)

> # plot multiple series at once
> plot(sbuxmsft.z, plot.type="single", col=c("blue","red"), lty=1:2,
+      lwd=2)
> legend(x="topleft", legend=c("SBUX","MSFT"), col=c("blue","red"),
+        lty=1:2)
```



Manipulating zoo objects

To be completed

There are several useful functions for manipulating zoo objects

Converting a `ts` object to a `zoo` object

To be completed.

Importing data into a `zoo` object

The function `read.zoo()` can read data from a text file stored on disk and create a zoo object. This function is based on the Base R function `read.table()` and has a similar syntax. For example, to read the date and price information in the text file `sbux.csv` and create the zoo object `sbux.z` with a time index of class yearmon use

```
> sbux.z2 = read.zoo("C:/classes/econ424/fall2008/sbuxPrices.csv",
+                     format="%m/%d/%Y", sep=",", header=T)

> # convert index to yearmon
> index(sbux.z2) = as.yearmon(index(sbux.z2))

> head(sbux.z2)
Mar 1993 Apr 1993 May 1993 Jun 1993 Jul 1993 Aug 1993
    1.19     1.21     1.50     1.53     1.48     1.52
```

Importing Data Directly from Yahoo!

The function `get.hist.quote()` in the package tseries can be used to directly import data on a single ticker symbol from `finance.yahoo.com` into a `zoo` object (multiple symbols are not supported).

To download daily adjusted closing price data on SBUX over the period March 1, 1993 through March 1, 2008 use (make sure the tseries package has been installed)

```
> library(tseries)
> SBUX.z = get.hist.quote(instrument="sbux", start="1993-03-01",
+                         end="2008-03-01", quote="AdjClose",
+                         provider="yahoo", origin="1970-01-01",
+                         compression="d", retclass="zoo")
trying URL
'http://chart.yahoo.com/table.csv?s=sbux&a=2&b=01&c=1993&d=2&e=01&f=20
08&g=d&q=q&y=0&z=sbux&x=.csv'
Content type 'text/csv' length unknown
opened URL
downloaded 179 Kb

time series ends   2008-02-29
```

The optional argument `origin=”1970-01-01”` sets the origin date for the internal numeric representation of the date index, and the argument `compression=”d”` indicates that daily data should be downloaded. The object `SBUX.z` is of class `zoo` and the date index is of class `Date`

```
> class(SBUX.z)
[1] "zoo"

> class(index(SBUX.z))
[1] "Date"

> start(SBUX.z)
[1] "1993-03-01"

> end(SBUX.z)
[1] "2008-02-29"
```