QF5210: FINANCIAL TIME SERIES: THEORY AND COMPUTATION

An introduction to R: some useful commands and financial data analysis

10 May 2019

Important: You need to type the commands by yourselves in the R working space instead of copying and pasting them from this PDF file. The single and double quotes in the PDF file are not compatible with R.

This note is a brief introduction to R. To learn more details about R, you can find many books online or from the libraries. For example, you can download the manual *An introduction to R* (link: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf) or the first chapter "Basic R Programming" of the book "Introducing Monte Carlo Methods with R" from the website of NUS Libraries (link: http://link.springer.com.libproxy1.nus.edu.sg/book/10.1007%2F978-1-4419-1576-4).

1 Some useful R commands

You can obtain help from the system. For R, sentences starting with " # " are comments.

- To launch a Web browser that allows to show the help pages type

  ```
  > help.start()
  ```

- to obtain help on particular topic (e.g. ar: to fit an autoregressive time series model to the data) type

  ```
  > ?ar # or help(ar)
  ```

- The assignment operator is <-. For instance, x <- 10 assigns the value 10 to the variable x. Here R treats x as a sequence of real numbers with the first element being 10.

  ```
  >x <- 10; x
  [1] 10
  ```

- R is an object oriented program. It handles many types of object. Objects are created and stored by name. To display the names of (most of) the objects which are currently stored within R, the R command

  ```
  > objects()
  ```

- Sometimes objects from the previous calculations need to be removed.

  ```
  > rm(obj) #to remove the object obj type
  > rm(obj1, obj2, obj3) #to remove a collection of objects
  > rm(list=ls(all=TRUE)) #to remove all objects
  > objects()
  character(0)
  ```

- To use commands (e.g. functions) stored in an external file, e.g. commands.R in the current working directory work, type

  ```
  > source(”commands.R”)
  > abcfunction() #abcfuntion() is defined and stored in commands.R
  ```

- Finally, the basic operations in R are similar to those we commonly use and the command to exit R is

  ```
  > q().
  ```

1.1 Data objects: Vectors

- A vector may be created by using concatenation function, *c*().

  ```
  > value.num <- c(3,4,2,6,20)
  > value.char <- c(”koala”,”kangaroo”,”echidna”)
  > value.logical <- c(F,F,T,T)
  > value.logical [1] FALSE FALSE TRUE TRUE
  ```

- The *rep* function replicates elements of vectors.

  ```
  > value <- rep(5,6)
  > value
  [1] 5 5 5 5 5 5
  ```

- The *seq* function creates a regular sequence of values to form a vector.

  ```
  > seq(from=2,to=12,by=2)
  [1] 2 4 6 8 10 12
  > seq(from=2,to=12,length=6)
  [1] 2 4 6 8 10 12
  > 1:5
  [1] 1 2 3 4 5
  ```

- The functions can be used in combination.

  ```
  > value <- c(1,2,5,rep(3,4),seq(from=1,to=6,by=3))
  > value
  [1] 1 2 5 3 3 3 3 1 4
  ```

- The *scan* function is used to enter data at the terminal.

  ```
  > value <- scan()
  1: 1 2 3 5 8 13 21
  8:
  ```

```
Read 7 items
> value
[1] 1 2 3 5 8 13 21
```

- Vector operations

```
> x <- runif(10) # generates random vector of length 10 independent, uniformly distributed
> x
[1] 0.4736283 0.8360173 0.1076954 0.2037522 0.3427902 0.7600275 0.2862429
[8] 0.3000142 0.6728465 0.9733413
> y <- 10*x + 1
> y
[1] 5.736283 9.360173 2.076954 3.037522 4.427902 8.600275 3.862429
[8] 4.000142 7.728465 10.733413
> z <- (x-mean(x))/sd(x)
> z
[1] -0.07442069 1.15104971 -1.31187564 -0.98704569 -0.51686859
[6] 0.89407932 -0.70809151 -0.66152173 0.59926419 1.61543063
> mean(x)
[1] 0.4956356
> sd(x)
[1] 0.2957142
```

1.2 Data objects: Matrices

- A matrix may be created from a vector by using *dim*.

```
> value <- rnorm(10) # generates random vector of length 10 independent, normal distributed
> dim(value) <- c(2,5) #2× 5 matrix
> value
[,1] [,2] [,3] [,4] [,5]
[1,] 0.4353214 -1.1757879 0.4889242 0.4890142 0.06190076
[2,] -0.8995333 0.6631052 0.2173262 -0.6255608 -0.20898896
> dim(value) <- NULL # back to vector
> value
[1] 0.43532138 -0.89953328 -1.17578789 0.66310520 0.48892421
[6] 0.21732620 0.48901417 -0.62556085 0.06190076 -0.20898896
```

- It may also be created from a vector by using *matrix*.

```
> value1 <- matrix(value,2,5); value1 #2,5 is the dimension of the matrix
[,1] [,2] [,3] [,4] [,5]
[1,] 0.4353214 -1.1757879 0.4889242 0.4890142 0.06190076
[2,] -0.8995333 0.6631052 0.2173262 -0.6255608 -0.20898896
> matrix(value,2,5,byrow=T) #type ?matrix to see the difference
[,1] [,2] [,3] [,4] [,5]
[1,] 0.4353214 -0.8995333 -1.1757879 0.66310520 0.4889242
[2,] 0.2173262 0.4890142 -0.6255608 0.06190076 -0.2089890
```

- To bind a row onto an already existing matrix, the *rbind* function can be used

```
> value2 <- rbind(value1,c(1,1,2,2,3)) # add one row
[,1] [,2] [,3] [,4] [,5]
[1,] 0.4353214 -1.1757879 0.4889242 0.4890142 0.06190076
[2,] -0.8995333 0.6631052 0.2173262 -0.6255608 -0.20898896
[3,] 1.0000000 1.0000000 2.0000000 2.0000000 3.00000000
```

- To bind a column onto an already existing matrix, the *cbind* function can be used

```
> value3 <- cbind(value2,c(1,1,2)) # add one column
> value3
[,1] [,2] [,3] [,4] [,5] [,6]
[1,] 0.4353214 -1.1757879 0.4889242 0.4890142 0.06190076 1
[2,] -0.8995333 0.6631052 0.2173262 -0.6255608 -0.20898896 1
[3,] 1.0000000 1.0000000 2.0000000 2.0000000 3.00000000 2
```

1.3 Data objects: Data frame

- The function *data.frame* converts a matrix or collection of vectors into a data frame

```
> value3 <- data.frame(value3)
> value3
X1 X2 X3 X4 X5 X6
1 0.4353214 -1.1757879 0.4889242 0.4890142 0.06190076 1
2 -0.8995333 0.6631052 0.2173262 -0.6255608 -0.20898896 1
3 1.0000000 1.0000000 2.0000000 2.0000000 3.00000000 2
> value4 <- data.frame(rnorm(3),runif(3))
> value4
rnorm.3. runif.3.
```

```
1 0.7011674 0.06922412
2 0.6156600 0.51559586
3 -0.6608378 0.13018683
```

- To view the row and column names of a data frame:

```
> names(value4)
[1] "rnorm.3." "runif.3."
> row.names(value4)
[1] "1" "2" "3"
```

- Alternative labels can be assigned by doing the following

```
> names(value4) <- c("C1","C2")
> row.names(value3) <- c("R1","R2","R3")
```

- Names can also be specified within the data.frame function itself.

```
> data.frame(C1=rnorm(3),C2=runif(3),row.names=c("R1","R2","R3"))
C1 C2
R1 -0.4812944 0.4096827
R2 -0.3381241 0.8146339
R3 -0.3751873 0.6444918
```

- The following example is to show how to access elements of a vector or matrix

```
> x <- sample(1:5, 10, rep=T) #produces a random sample of values between one and five, ten times
> x
[1] 5 1 2 5 1 1 3 1 2 2
> ones <- (x == 1) #check if all the elements of x are equal to 1
> ones
[1] FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE
> x[ones] <- 0
> x
[1] 5 0 2 5 0 0 3 0 2 2
> others <- (x > 1)
> y <- x[others] #stores the values greater than 1 into y.
> y
[1] 5 2 5 3 2 2
> which(x > 1) #finds indices of elements bigger than 1
```

```
[1] 1 3 4 7 9 10
> y <- x[-(1:5)] #copies x without the first 5 elements. To exclude values, negative index vectors are used
> y
[1] 0 3 0 2 2
```

1.4 Data sorting

The command *order* allows sorting with tie-breaking: Find an index vector that arranges the first of its arguments in increasing order. Ties are broken by the second argument and any remaining ties are broken by a third argument.

```
> x <- sample(1:5, 20, rep=T)
> y <- sample(1:5, 20, rep=T)
> z <- sample(1:5, 20, rep=T)
> xyz <- rbind(x, y, z)
> dimnames(xyz)[[2]] <- letters[1:20] #names the columns by the first 20 letters
> xyz
a b c d e f g h i j k l m n o p q r s t
x 3 4 3 2 5 5 4 5 1 2 4 4 3 3 5 5 2 5 3 4
y 1 1 2 2 4 4 5 5 1 1 2 1 4 5 3 4 2 1 1 5
z 5 1 5 4 5 4 1 2 5 1 4 1 3 4 1 3 3 1 1 4
> o <- order(x, y, z) #orders the matrix xyz first by x, then by y and at last by z
> xyz[, o]
i j q d s a c m n b l k g t r o p f e h
x 1 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5
y 1 1 2 2 1 1 2 4 5 1 1 2 5 5 1 3 4 4 4 5
z 5 1 3 4 1 5 5 3 4 1 1 4 1 4 1 1 3 4 5 2
```

1.5 Matrix calculations

- If, for example, A and B are square matrices of the same size, then

  ```
  >A*B #is the matrix of element by element products
  > A %*% B #is the matrix product.
  ```

- If x is a vector, then

  ```
  > x %*% A %*% x #is a quadratic form.
  > (mat1 <- matrix(c(1,0,1,1), nrow=2))
  [,1] [,2]
  [1,] 1 1
  [2,] 0 1
  > (mat2 <- matrix(c(1,1,0,1), nrow=2))
  ```

```
[,1] [,2]
[1,] 1 0
[2,] 1 1
> solve(mat1) # inverts the matrix
[,1] [,2]
[1,] 1 -1
[2,] 0 1
> mat1 %*% mat2 # Matrix multiplication
[,1] [,2]
[1,] 2 1
[2,] 1 1
> mat1 + mat2 # Matrix addition
[,1] [,2]
[1,] 2 1
[2,] 1 2
> t(mat1) # Matrix transposition
[,1] [,2]
[1,] 1 0
[2,] 1 1
> det(mat.1) # Matrix determinant
[1] 1
```

- *diag*() depends on its argument.
  - *diag*(v), where v is a vector, gives a diagonal matrix with elements of the vector as the diagonal entries
  - *diag*(M), where M is a matrix, gives the vector of main diagonal entries of M.
  - if k is a single integer value then *diag*(k) is the $k \times k$ identity matrix.

  Examples:

```
> A<-diag(c(1,2))
> A
[,1] [,2]
[1,] 1 0
[2,] 0 2
> diag(A)
```

```
[1] 1 2
> diag(2)
[,1] [,2]
[1,] 1 0
[2,] 0 1
```

1.6 Reading and writing data

For reading and writing in files, R uses the working directory.

There are several ways to read and load data into the R working space, depending on the data format. For simple text data, the command is *read.table*. For .csv files, the command is *read.csv*. The data file is specified in either a single or double quotes; see examples below and the R commands of Lecture 1 available on IVLE.

R treats the data as an object and refer to them by the assigned name. For both loading commands, R stores the data in a matrix framework. As such, one can use the command *dim* (i.e., dimension) to see the size of the data.

```
> mydata <- read.table(”data.dat”)
```

will create a data frame named *mydata*, and each variable (column) will be named, by default, by

$$V1,\ V2, ...$$

and can be accessed individually by mydata$V1, mydata$V2, ... or by mydata[”V1”], mydata[”V2”], ... or, still another solution, by mydata[,1], mydata[,2], ...

The command *write.table* writes in a file an object, typically a data frame but this could well be another kind of object (vector, matrix, . . .). The arguments and options are (type ?write.table to understand this command):

```
> write.table(x, file = ””, append = FALSE, quote = TRUE, sep = ” ”, eol = ”
n”, na = ”NA”, dec = ”.”, row.names = TRUE, col.names = TRUE, qmethod = c(”escape”,
”double”))
```

Examples:

```
> X<-matrix(1:6,2,3,byrow=T); X
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
> Y<-data.frame(X); Y # create data frame
X1 X2 X3
1 1 2 3
2 4 5 6
> Z<-rbind(Y, c(’a’, ’b’, ’c’)); Z # add one row
```

```
X1 X2 X3
1 1 2 3
2 4 5 6
3 a b c
> colnames(Z)<-c(’A’, ’B’,’C’); Z #name the columns
A B C
1 1 2 3
2 4 5 6
3 a b c
> filepath<-”C:/Users/.../” # path of your working directory ending by ”/”
> file.name<-”test.data” # give a name to a new file
> full.file.name<- paste(filepath, file.name, sep=’ ’) # put the working directory and the file name together
> write.table(file=full.file.name, Z) # write Z in the file
> list.files(filepath) # show all the existing files under the working directory
[16] ”hw1” ”test.data”
> file.show(full.file.name) # show the file test.data, strings are in ””
> Znew<- read.table(full.file.name); Znew # read the file test.data
A B C
1 1 2 3
2 4 5 6
3 a b c
> write.table(file=full.file.name, Z, quote=F) # many options are avialble
> file.show(full.file.name) #no””
> Znew<- read.table(full.file.name, skip=1); Znew #no column names from the file, will be given automatically # 4 columns instead of 3!
V1 V2 V3 V4
1 1 1 2 3
2 2 4 5 6
3 3 a b c
> Znew<- read.table(full.file.name, skip=1, nrows=1); Znew # skips first line and reads the second
V1 V2 V3 V4
1 1 1 2 3
```

1.7 Re-directing output and file management

By default, the output are showed in the R working space. However, you can re-direct the output to a file in your current working directory. See the following example:

```
> print(’hello’)
> sink(”out.file”)
> print(’hello’)
> sink()
> file.show(’out.file’) # shows the file
> file.remove('out.file') # removes out.file
> list.files() # no out.file any more
```

2 Financial data analysis

R contains some financial databases. For example, **EuStockMarkets** database. Run the following code to access the database, learn its mode and class, and plot the four time series.

```
> data(EuStockMarkets)
> head(EuStockMarkets)
> dimnames(EuStockMarkets)[[2]]
> mode(EuStockMarkets)
> class(EuStockMarkets)
> plot(EuStockMarkets)
```

To plot the time series in a pdf file under the working directory, one can use the following commands

```
> pdf("EuStocks.pdf",width=6,height=5)
> plot(EuStockMarkets)
> graphics.off()
```

Download the financial data of Leture 1 from IVLE, then analyse them (mean, variance, test, plot, etc) as shown on the Lecture by using the R commands also available on IVLE.