Lund Institute of Technology Centre for Mathematical Sciences Mathematical Statistics

Quantitative Risk Management Using Copulas, FMSN65/MASM33

Computer Assignment 1

This assignment is a compulsory part of the course and will be carried out by each group consisting of two students. The report (a PDF file) should be uploaded at the home page of the course in *Canvas* by the corresponding deadline.

Introduction to R

Getting Started:
If you plan to carry out the assignment on your own computer, you need to download and install `R` from http://www.r-project.org.

If you want to carry out the assignment in the computer room MH:230, you just need to start one of the PCs first. Then choose the latest version of `R` from the `Start` menu. Note that if you want to save any files, they will be saved on the hard drive in the PC. Alternatively, you can save your files to a cloud server or a memory stick.

Some tips and hints when typing in R code:

- R is case sensitive (so `LM()` and `lm()` are not equivalent).
- R is tolerant to the use of spaces, so `x <- 1` and `x<-1` are equivalent; though, the former being considered to be more readable.
- You can use the arrow keys to speed things up. The 'up' arrow gives you the previous command that you typed.
- The usual prompt sign for R is `>`. If you get a `+` prompt sign instead, it means that R is awaiting the completion of the previous command that you typed in. This can happen because you have forgotten to close parentheses, for instance. Just type in the remainder of the command.

Exercise 1: R as a Calculator

Use R to compute the following:

1. $98765 - 43210$
2. 12.3456789^2
3. $\sqrt{(1.23 + 4.56) \times (7.8 + 9.0)}$

Type the following expressions into R, and check that you understand the results.

1. `3+4*5`
2. `3/4/5`
3. `1:20/2`
4. `(1:20)/2`
5. `1:(20/2)`
6. `c(4,2,4)^2 - c(1,3)`
7. `(1:5)^(1:2)`

Exercise 2: Creating regular sequences

Write (efficient) code to create the following sequences:

1. $2, 4, 6, \dots, 100$ (even numbers up to 100)
2. $1, 2, 3, 4, 5, 4, 3, 2, 1$
3. $10, 20, 30, 40, 10, 20, 30, 40, 10, 20, 30, 40$

(HINT: `rep(x, n)` is equivalent to `c(x,x,.....,x)`, where `x` is repeated n times).

Exercise 3: Practising indexing of vectors

A sequence $y_1, y_2, \dots, y_n$ of $n = 100$ standard normal random variables can be **simulated** by `y <- rnorm(100)`

Write code to return:

1. $y_2, y_4, y_6, \dots, y_{100}$
2. all elements of y greater than 1.96
3. the indices of all negative elements of y
4. $\frac{y_{51}}{y_1}, \frac{y_{52}}{y_2}, \dots, \frac{y_{100}}{y_{50}}$

Exercise 4: Exploratory Analysis of IBM Stock Price Data

This exercise is concerned with the closing price (in U.S. dollars) of IBM stock over a sequence of 250 consecutive trading days in 1980. The dataset `ibm.txt` can be downloaded from the following locations

- The page `Datasets in R` under the `Pages` in the homepage of the course in *Canvas*,
- http://www.maths.lth.se/matstat/kurser/fmsn15masm23/datasetsR.html.

These data can be read in and assigned to the vector `ibm` by using the following command. *Please note that depending on where you have saved the file `ibm.txt` you have to modify the path for the file in the following command.*

```
ibm <- scan(file="C:/IntroR/ibm.txt")
```

Alternatively, you can read directly from the Web as follows.

```
ibm <- scan("http://www.maths.lth.se/matstat/kurser/fmsn15masm23/datasets/ibm.txt")
```

1. Take a look at the data by typing the name of the vector: `ibm`.
2. Find the mean, standard deviation, maximum and minimum of the IBM stock price data.
3. You should have found in part 1 that the maximum price is the (rather unlikely) value 6575. This is, in fact, an outlier generated by 1a typographical error. Find out which element of `ibm` corresponds to this outlier, and replace that element by the correct value which is 65.75.
4. Recalculate the summary statistics from part 1 using the corrected data set.
5. The *return* on any given day is defined to be the stock price on that day divided by the stock price on the previous day. Create a vector containing the returns.
6. It is often assumed in financial modelling that the log-returns are (approximately) independent and normally distributed. Create a vector containing the logarithms of the daily returns; call it `lreturn`.
7. Investigate whether the normality assumption seems reasonable by producing a histogram of the data with the command `hist(lreturn)`. Try `hist(lreturn,breaks=15)` to get a finer level of discretization.
8. Type `t.test(lreturn,mu=0)` and interpret the results. Find a 99% confidence interval for the (population) mean log-return by `t.test(lreturn,mu=0,conf.level=0.99)`.

Exercise 5: Plotting Functions of One Variable

The aim of this exercise is to help you learn how to plot functions of one variable in R. Consider, for example, the function

$$f(x) = x^2$$

and suppose that we wish to plot the graph of $f(x)$ for x in the range $(-5, 5)$. The way to do this in R is as follows:

(i) Create a fine grid (sequence) of x values on the interval $(-5,\ 5)$.

(ii) Evaluate the function $f(x)$ at each grid point.

(iii) Join up the resultant pairs $(x, f(x))$ with tiny straight line segments.

This is perhaps easiest understood by actually doing it!

Start by creating the grid of x values using the command

```
x <- seq(-5, 5, length=201)
```

Take a look at the object `x` (just type its name) to see what you have made. Now evaluate the function on the grid:

```
fx <- x^2
```

Finally, plot `fx` against `x` with

```
plot(x, fx, type="l")
```

The argument `type="l"` of `"plot"` tells R to connect the points with line segments (hence `"l"`). By default R would simply plot the unconnected points – see what happens if you omit the (optional) `type="l"` argument. You can beautify your graph by adding labels to the axes, by adding a title, and by changing the colour scheme. Try

```
plot(x, fx, type="l", ylab="f(x)", col="red", main="Graph of f(x)=x^2")
```

and then try some variations of your own devising.

Now try plotting the following functions:

1. $f(x) = 7\log(x) - x$ for x in the interval $(0.1,\ 20)$.
2. $f(x) = 5\log(x) + 10\log(1-x)$ for x in the interval $(0.1,\ 0.9)$.

Exercise 6: Plotting Functions of Two Variables

It is much more difficult to visualise functions of many variables than functions of one variable. Nonetheless, we can get useful displays of functions of two variables using perspective ('wire frame'), contour and image ('heat') plots. This exercise will help you learn how to generate such plots in R.

Consider the function

$$f(x, y) = e^{-(x^2+y^2)}$$

and suppose that we wish to plot its graph over the range $x \in (-3,\ 3)$ and $y \in (-3, 3)$. To do this we will:

(i) Create a fine grid of points on the x-y plane over the ranges indicated above.

(ii) Evaluate the function $f(x, y)$ at each grid point.

(iii) Supply grid and function evaluations to R's `persp`, `contour` or `image` commands.

Start by creating the locations of the grid lines on the x and y axes:

```
x <- seq(-3, 3, length=51)
y <- seq(-3, 3, length=51)
```

All the R plotting commands for functions of two variables expect the evaluations of $f(x, y)$ to be presented as a *matrix*, with elements corresponding to the appropriate grid locations. We start by creating a matrix full of zeros:

```
fxy <- matrix(0, ncol=51, nrow=51)
```

Note that the number of columns (`ncol`) and rows (`nrow`) matches the number of grid lines on each axis. We now insert the function values into this matrix using loops[1]:

```
for (i in 1:51){
  for (j in 1:51){
    fxy[i,j] <- exp(-(x[i]^2 + y[j]^2))
  }
}
```

Finally, we can get perspective, contour and image with the following syntax:

```
persp(x, y, fxy)
contour(x, y, fxy)
image(x, y, fxy)
```

If you have time, try beautifying these plots; e.g.

```
persp(x, y, fxy, col="cyan", theta=45)
contour(x, y, fxy, nlevels=25)
image(x, y, fxy, col=terrain.colors(50))
```

Use R's help system to guide you.

Exercise 7: Programming task 1

On 21 June 1995 the German press reported the following sensation: "For the first time it its history a lottery draw resulted in winning numbers that had been already drawn once. The numbers drawn on 21 June 1995, namely 15–25–27–30–42–48, were identical to those drawn on 20 December 1986. There are nearly 14 million different combinations that can be drawn."

To put this into context, the German lottery selects six numbers (without replacement) out of 49. And the draw on 21 June 1995 was the 3016th draw of the lottery.

Is this fact really sensational? What is the probability that the first repeat does not happen until after the 3016th draw?

1. Write some code that simulates 3016 draws of the German lottery and determines whether there has been one, or more, repeat draws within these 3016 draws. Execute the code repeatedly to estimate the probability that the first repeat does not happen until after the 3016th draw.

   Hint: Find a way of turning each draw into a single number (also sometimes called a hash) and store all 3016 hashes in a vector; then use the function `unique()` to check whether the 3016 draws resulted in 3016 different hashes. For this scheme to work, you need that draws of different numbers (e.g. 5–4–20–6–45–17 and 9–3–11–23–41–4) produce different hashes, but that draws of the same numbers (e.g. 24–14–21–19–13–33 and 13–24–33–14–21–19) produce the same hash.

[1]Note: Alternatively, to avoid loops we could 'vectorise' these calculations. A simple command like

```
fxy <- exp(-outer(x^2, y^2, "+"))
```

would do the same calculations. In this case, we also would not have to issue the previous command that created `fxy` and filled it with zeros.

2. It is not difficult to find the exact solution for the probability of having at least one repeated draw in 3016 draws.

   HINT: Write the probability for the event that all 3016 draws are different and take the complement of that event.

   Write a function in `R` which calculates the exact probability of having at least one repeated draw (see the function `choose()` in `R`).

Exercise 8: Programming task 2

Given a value y and a starting value x_0, the following iterative scheme implements the Newton-Raphson algorithm for finding $\sqrt[3]{y}$:

$$x_{n+1} = x_n - \frac{x_n^3 - y}{3x_n^2} = \frac{1}{3}\left(2x_n + \frac{y}{x_n^2}\right), \qquad n = 0, 1, 2, \ldots$$

Write a function that takes y as input and has two optional parameters; one optional parameter to pass a possible starting value x_0 to the function and a second optional parameter to which the user can pass a value ϵ and the above iteration stops when $|x_n^3 - y| < \epsilon$. Make sure that you assign sensible default values to the two optional parameters. The function should return $\sqrt[3]{y}$.

Finishing Off:

When you've finished, close down R by typing `q()`. Choose 'Save' when prompted as to whether you want to retain your workspace.