STATISTICAL MODELLING Part IIC, Michaelmas 2021
Practical 1: Introduction to R By courtesy of Dr. S. A. Bacallado and Dr. R. Shah

Note: Below is the practical used by last year's lecturer. A more comprehensive introduction to R can be found at `https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf`. The best way to learn R is through making errors and reading the manual/help page. You should try to solve all the exercises before the next practical session.

R as a calculator

R can be used as a calculator:

```
> (9.1^3)*sqrt(14)*exp(-5)/log(4)
```

Help on any R function can be found by typing a question mark followed by the function, e.g.

```
> ?exp > help(exp) > ??exp
```

You will need to use this help facility extensively (and get used to skim-reading to find the relevant bit!). Note that R is case-sensitive.
The `<-` symbol is the usual assignment operator in R (symbol `=` can also be used, but it has slightly different purposes so I recommend sticking to `<-`). For instance, we can assign the value `3` to the variable `x`, and then perform operations on `x`. Anything which appears after the hash symbol `#` is a comment and need not be typed.

```
> x <- 3
> round(x^2 + log10(x), 3)  # try ?round to see what it does
[1] 9.477
> 37 %/% 3  # try ?'%/%'
[1] 12
> 37 %% 3
[1] 1
```

Creating vectors

The `c` function (for 'concatenate') combines values into a vector.

```
> x <- c(3, 6, 4, 2)
> x
[1] 3 6 4 2
> length(x)
[1] 4
```

There is no such thing as a scalar in R; what one might think of as a scalar is treated as a vector of length 1. Note that R does not distinguish between row and column vectors unlike MATLAB.
You can create a vector `y` with the same entries using `y <- scan()`. Enter one component per line and leave a blank line after the last.
A sequence of equally spaced numbers can be created using the `seq` function. The `rep` function provides different ways of repeating vectors.

Operations on vectors

Operations on vectors in R are performed component by component. For example

```
> x + x
[1]  6 12  8  4
> x*x
[1]  9 36 16  4
> exp(x)
[1]  20.085537 403.428793  54.598150   7.389056
```

When operations are performed on vectors of different lengths, the shorter vector is cycled until it is the same length as the longer vector.

```
> x <- c(3, 6, 4, 2)
> y <- c(1, 2)
> x + y
[1] 4 8 5 4
> x*y
[1]  3 12  4  4
> x^y
[1]  3 36  4  4
> y <- 1:3  # same as y <- c(1, 2, 3)
> x + y
[1] 4 8 7 3
Warning message:
In x + y : longer object length is not a multiple of shorter object length
```

What are the values of `x + 2`, `3*x` and `(2 + x)^3`?

Indexing vectors

```
> x <- c(3, 6, 4, 2)
> x[2]  # 2nd component of x
[1] 6
> x[c(1, 3)]  # 1st and 3rd components of x
[1] 3 4
> x[-1]  # All of x except the 1st component
[1] 6 4 2
> x[-(1:2)]  # All of x except the 1st two components
[1] 4 2
> x[1:2] <- c(7.1, 3.4)  # We can assign values to components
> x
[1] 7.1 3.4 4.0 2.0
```

Note that after the final command, `x` has automatically transformed from a vector of integers to a vector of *floating point numbers* (these are a way of representing real numbers on computers, though of course only to a certain degree of accuracy).
We can also index components of a vector using a `TRUE / FALSE` (logical) vector.

```
> index_vec <- c(TRUE, TRUE, FALSE, TRUE)
> x[index_vec]
[1] 7.1 3.4 2.0
```

Logical vectors can also be created using the binary operator < which performs componentwise comparisons.

```
> x
[1] 7.1 3.4 4.0 2.0
> x > 3.6
[1]  TRUE FALSE  TRUE FALSE
> x[x > 3.6]
[1] 7.1 4.0
```

Matrices

We can create a matrix using the `matrix` function.

```
> A <- matrix(1:8, 2, 4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

Can you enter the terms by row instead? Rows and columns of matrices can be extracted in the following way:

```
> A[1, ]
[1] 1 3 5 7
> A[, 3]
[1] 5 6
```

Note that the rows and columns thus formed are now vectors. We can check this using the very helpful `str` (for 'structure') function.

```
> str(A[1, ])
 int [1:4] 1 3 5 7
```

Here we see that `A[1, ]` is an integer vector of length 4. To keep the 2-by-1-matrix structure-type, we use

```
> A[, 2, drop = FALSE]
     [,1]
[1,]    3
[2,]    4
```

An alternative is to do

```
> matrix(A[, 2])
     [,1]
[1,]    3
[2,]    4
```

Submatrices can be formed by e.g. `A[, 1:3]`. The diagonal can be extracted using `diag`. We can perform many standard operations on matrices.

```
> A %*% x  # matrix vector multiplication
     [,1]
[1,] 51.3
[2,] 67.8
> A*A  # componentwise multiplication
     [,1] [,2] [,3] [,4]
[1,]    1    9   25   49
[2,]    4   16   36   64
> t(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> A %*% t(A)  # matrix matrix multiplication
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

The `solve` function can be used to invert matrices and, thus, to solve linear systems:

```
> solve(A %*% t(A))
      [,1]  [,2]
[1,]  1.50 -1.25
[2,] -1.25  1.05
```

A few important functions

```
> x <- c(3, 6, 4, 2)
> sum(x)
[1] 15
> sum(x > 3)  # TRUE is treated as 1 and FALSE, 0
[1] 2
> mean(x)
[1] 3.75
> sort(x)
[1] 2 3 4 6
> sd(x)  # standard deviation
[1] 1.707825
```

How is the standard deviation being calculated? Functions for matrices:

```
> mean(A)  # mean treats A as a vector
[1] 4.5
> colMeans(A)
```

```
[1] 1.5 3.5 5.5 7.5
> rowSums(A)
[1] 16 20
```

The function `cbind` 'glues' columns of matrices together.

```
> cbind(1, A)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    3    5    7
[2,]    1    2    4    6    8
```

Generating (pseudo) random numbers

(Pseudo) independent and identically distributed sequences of random numbers are generated with commands like `rnorm, runif, rchisq` etc. (normal, uniform, χ^2). The corresponding density, cumulative distribution and quantile functions are, e.g. `dnorm, pnorm, qnorm`.

```
> x <- rnorm(1000)
> hist(x, freq = FALSE)
> x_vec <- seq(-3, 3, by = 0.1)
> lines(x_vec, dnorm(x_vec), col = "red")  # adds lines to an existing plot
```

What does the following code do?

```
> X <- matrix(runif(50*1000, min=-1, max=1), 50, 1000)
> hist(sqrt(50) * colMeans(X) / sd(X), freq = FALSE)  # sd treats X as a vector
> lines(x_vec, dnorm(x_vec), col = "red")
```

`lines` plots on top of the current plot, but if you wish to use superposition to histograms (or plots) you can use the following instruction (equivalent to to hold on in Matlab).

```
> par(new=TRUE)
```

Experiment with other distributions and other sample sizes.

Exercises

1. Let $Z \sim N(0,1)$. Estimate $\mathbb{E}(Z|\{Z \geq 1\})$ and $\mathbb{E}(Z^6)$.
2. What is the upper 5% point of a χ^2_6 distribution?
3. Use R to solve

$$\begin{aligned} 3a + 4b - 2c + d &= 9 \\ 2a - b + 7c - 2d &= 13 \\ 6a + 2b - c + d &= 11 \\ a + 6b - 2c + 5d &= 27. \end{aligned}$$

4. Two lecturers mark the same Tripos question for two randomly selected, disjoint sets of students. To make sure that neither of them is more lenient than the other, they want to test whether their average marks are equal. But they are afraid the sample size is too small to apply the Central Limit Theorem, so one of them writes the following code.

```
> grades_1 <- c(10,11,14.5,15,15,18,12,19,18.5,19,20,13)
> grades_2 <- c(12,11,14.5,13,12,11,12,14.5,20,17)
> mean(grades_1)
> mean(grades_2)
> tstat <- mean(grades_1)-mean(grades_2)
> all_grades <- c(grades_1,grades_2)
> edtstat = rep(0, 10000)
> for (i in 1:10000) {
>   perm <- sample(all_grades)
>   edtstat[i] <- mean(perm[1:length(grades_1)])-
>                   mean(perm[-(1:length(grades_1))])
> }
> p_value <- mean(abs(tstat) <= abs(edtstat))
> p_value
```

How can you interpret the output of the last line? Search for the documentation of any function you have not encountered before.