STATISTICAL MODELLING Part IIC, Michaelmas 2021
Practical 2: More on the basics of R By courtesy of Dr. S. A. Bacallado and Dr. R. Shah

Note: Below is the practical sheet used by last year's lecturer. The recording is an improvised demo of the materials below. You should still try to understand them and finish the exercises as the functions below are better written with more features.

Writing functions

When writing anything but a short algorithm, it is often easiest to edit the commands from a text file. In `Rstudio` create a new Rscript using Ctrl+Shift+N. Write out the following code:

```
f <- function (x, y) {
  z <- x^2 + y^2
  return(c(cos(z), sin(z)))
}
```

Once you have written the algorithm and saved the file as `Rubbish.R`, say, in the current working directory, you can execute the commands by typing

```
> source("Rubbish.R")
```

in the console. Typing `f` in the console will now echo the code of your function, and you can run your function by giving it the right arguments e.g. `f(2, 3)`.
Now let's write a function to simulate t-statistics,

$$\frac{\hat{\beta}_j}{\sqrt{\tilde{\sigma}^2\{(X^TX)^{-1}\}_{jj}}},$$

from a linear model. The arguments of our function will be a design matrix `X`, a vector of coefficients `Beta`, a function for generating the errors (i.e. the ε_i) `errors_gen`, and the number of repetitions of the simulation we'd like to perform, `B`. The output will be a `p` by `B` matrix where each row is an i.i.d. sample of t-statistics $T_1, \ldots, T_B$, corresponding to the relevant component of `Beta`. Here is the code:

```
LinMod_sim <- function(X, Beta = rep(0, p), errors_gen = rnorm, B = 10000) {
  n <- nrow(X)
  p <- ncol(X)
  y_mat <- as.vector(X %*% Beta) + matrix(errors_gen(n*B), n, B)
  inv_gram <- solve(t(X) %*% X)
  P <- X %*% inv_gram %*% t(X)
  Beta_hat_mat <- inv_gram %*% t(X) %*% y_mat
  resid_mat <- y_mat - P %*% y_mat
  sigma_tilde_vec <- colSums(resid_mat^2) / (n - p)
  t_stat_mat <- Beta_hat_mat / sqrt(diag(inv_gram))
  t_stat_mat <- t_stat_mat / rep(sqrt(sigma_tilde_vec), each = p)
  return(t_stat_mat)
}
```

Notice we have specified default values for some of the arguments: for example the default error generating function is `rnorm`. Go through each line of the function to make sure you understand what it is doing (this may take some time), and look up any functions that you don't know using `?`. To understand what is happening in the matrix vector addition in line 4, and the matrix vector divisions in lines 10 and 11, it may help to try the following (we have omitted the output):

```
> 1:3 + matrix(0, nrow = 3, ncol = 2)
> rep(1:2, each = 3) + matrix(0, nrow = 3, ncol = 2)
```

The `as.vector` function used in line 4 of the code coerces `X %*% Beta` into a vector so that when it is added to the matrix of errors, the correct output is produced. Once the function `LinMod_sim` has been

loaded into your workspace (either by copying and pasting, or saving the code and sourcing it), we can use it. First create a matrix of predictors of your choice (perhaps using `runif` or `rnorm`) and store it in `X`. Now try

```
> output <- LinMod_sim(X)
```

Notice we do not need to specify the arguments that have default values in order to run the function.

Q–Q plots

We know that if $\beta_j = 0$ and the design matrix X is n by p (with full column rank), provided the errors satisfy $\varepsilon \sim N_n(0, \sigma^2 I)$,

$$\frac{\hat{\beta}_j}{\sqrt{\tilde{\sigma}^2\{(X^TX)^{-1}\}_{jj}}} \sim t_{n-p}.$$

To check that this is indeed the case, we can analyse the output of our function via Quantile–Quantile, or Q–Q, plots. This involves

1. sorting the t-statistics $T_1, \ldots, T_B$ into increasing order
2. plotting them against $\{F^{-1}(\frac{k}{B+1}) : k = 1, \ldots, B\}$, where F is the t_{n-p} distribution function.

We expect an approximately straight line through the origin with gradient 1 if the T_k are distributed according to t_{n-p}. This is the case because the empirical distribution function F_B of $T_1, \ldots, T_B$, defined as

$$F_B(x) := \frac{1}{B}\sum_{k=1}^{B} \mathbb{1}_{\{T_k \leq x\}}$$

satisfies $F_B(x) \to F(x)$ (almost surely) as $B \to \infty$. Thus for large B we can expect to see that $F_B(x) \approx F(x)$. Rather than looking at the t-statistics themselves, if we are interested in studying the size of the two-sided t-test, we may more simply look at the squares of the t-statistics directly. What will their distribution be? You can use the following function to produce the Q–Q plots.

```
qqplot_F <- function(f_stat, df1, df2, conf_level = 0.95) {
  f_stat <- sort(f_stat)
  B <- length(f_stat)
  theoretical_quantiles <- qf((1:B) / (B + 1), df1, df2)
  plot(theoretical_quantiles, f_stat)
  abline(0, 1, col = "red")
  return(mean(f_stat <= qf(conf_level, df1, df2)))
}
```

As usual, use `?` to query and functions you don't know already. What does the final line of the function do? Experiment with different values of n and p and error distributions. For example you could try

```
> output2 <- LinMod_sim(X, errors_gen = rcauchy)
> output3 <- LinMod_sim(X, errors_gen =  rexp(x) - 1)
```

The reason we subtract 1 when the errors are exponential is to enforce that the errors have zero mean. Note that if the points on the Q–Q plot lie above the diagonal it suggests that the usual t-test with nominal level α will have a size greater than α (why?). On the other hand, if the points lie below the diagonal, the t-test will be conservative and have size less than α.

Lists

Lists collect together items of different types, e.g.

```
> Empl <- list(employee="Eve", spouse = "Adam", children=2, child.ages = c(4, 7))
```

Elements of a list need not be of the same length, but its components are numbered. Thus `Empl` is a list of length 4, and its components are referred to as `Empl[[1]]` etc.. Notice that `Empl[[4]]` is a vector, so `Empl[[4]][1]` is its first entry. Names of components can also be used:

```
> Empl$employee
> Empl$child.ages[2]
```

The different components of a list can be almost anything, even functions or other lists.

Exercises

1. Modify the `LinMod_sim` to study what happens when the error variances are unequal. For example, you could replace line 4 of the function by

   ```
   y_mat <- as.vector(X %*% Beta) + matrix(errors_gen(n*B), n, B) * rnorm(n)
   ```

2. Modify the `LinMod_sim` function so that

   (a) it takes an extra argument, `G_0`, which can be used to specify a subset of the variables of X;

   (b) letting the matrix formed from these variables be X_0, the function outputs a list whose first component is the existing `t_stat_mat`, and whose second component is simulated values of the F statistic

   $$\frac{\frac{1}{p-p_0}\|(P-P_0)y\|^2}{\frac{1}{n-p}\|(I-P)y\|^2},$$

   with P, P_0 etc. defined as in lectures.

   Compare the values of the simulated F statistics with the appropriate distribution using the `qqplot_F` function.