MATH3029/MATH4082: On using the command `lm` in R

The session's main purpose is to review interpreting output from `lm`. Please work through the details making sure that you understand all numbers from the output. This is not an exercise in copy+paste or typing `R` commands correctly (Note: when copy + paste from a pdf onto the `R` console, it might not recognise certain fonts!).

You can find out more about the `lm` command by typing

```
help(lm)
```

However, the full specification of this command is somewhat involved. Fortunately, in most applications one can mainly use the default options and, as we shall see, in such cases only a small number of inputs need to be specified.

Let us use the Scottish hill race data in the MASS library. Type

```
library(MASS)
names(hills) = c("distance", "height.climbed", "record")
attach(hills)
```

Read about the commands `attach` and `names` if you are unfamiliar with them.

1. Plot `record` against `distance` and `height.climbed` using `plot` command; do the relationships look approximately linear?
2. Now fit a linear model (please check syntax specifications)

   ```
   out = lm( record ~ distance+ height.climbed)
   ```

   and look at the output:

   ```
   summary(out)
   ```

   How do we interpret the number 6.217956 corresponding to `distance`?

   In order to check what all the output means, let us first calculate the fitted values from the output. Using the parameter estimates in the table, type

   ```
   fv = -8.992039 + 6.217956*distance + 0.011048*height.climbed
   ```

   We can do the same thing more conveniently by extracting the fitted values directly from `out`:

   ```
   yhat = out$fitted.values
   ```

   Convince yourself that `yhat` and `fv` are the same! What is your prediction of `record` at height of 2125 and distance of 11.8?
3. Ensure you are familiar with the interpretation of the output in the coefficients table:
   (a) the estimates are obtained by maximum likelihood, which is equivalent to least squares in this case;
   (b) Recall that $\hat{\boldsymbol{\beta}} \sim N_p(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1})$, and hence each element of $\hat{\beta}_i$ is normally distributed with mean β_i and variance given by σ^2 times the diagonal elements, say d_{ii}, of $(\mathbf{X}^\top\mathbf{X})^{-1}$.
   (c) However since σ^2 is unknown and estimated by the residuals sum of squares $\hat{\sigma}^2$, $\frac{\hat{\beta}_i-\beta_i}{\sqrt{\hat{\sigma}^2 d_{ii}}} \sim t_{n-p}$ (Why?). This should explain the t-value in the output.
   (d) the standard errors are obtained by calculating the square root of the diagonal elements of $\hat{\sigma}^2(\mathbf{X}^\top\mathbf{X})^{-1}$ (check command `vcov` for this);
   (e) and the p-value in each case is the probability of a more extreme value, without regard to sign, under the hypothesis that the relevant coefficient is 0.

   Moving down the output from `summary(out)`, the *residual standard error* is the square root of the mean residual sum of squares, and the *residual degrees of freedom* is the sample size, n, minus the number of

parameters (including the intercept term) which have been estimated. In this example, the residual degrees of freedom is $35 - 3 = 32$.

If the residuals are denoted by $r_i = y_i - \hat{y}_i$, $i = 1, \ldots, n$, and the residual degrees of freedom is d, then the residual standard error is given by

$$\sqrt{\frac{1}{d}\sum_{i=1}^{n} r_i^2}.$$

We can check this agrees with `summary(out)` by typing the following:

```
sqrt(sum(residuals*residuals)/32)
```

4. The number produced above should be the residual standard error, 14.68, produced by `summary(out)`.

   type:

5. Finally, let us consider the F-statistic on the last line of `summary(out)`. This statistic tests for the existence of regression, i.e. it tests the null hypothesis that the distance and height.climbed coefficients are both zero ($H_0 : \beta_1 = \beta_2 = 0$), using the mean model sum of squares divided by the mean residual sum of squares:

$$\frac{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2/2}{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2/32} \quad .$$

   The calculation being performed (why?) in R is

   ```
   (sum((fv-57.87571)**2)/2)/(sum(residuals**2)/32)
   ```

   where 57.87571 is the sample mean of `record`.

   Check that this agrees with the value of the F-statistic in `summary(out)`.

   How can you check if only the covariate `height.climbed` is significant using the (1) t-test and (2) F-test under nested models? (use the command `anova`)

1 Basics of R

1.1 Matrices

Let us look at some more matrix based commands

```
x = iris
```

(assigns Fisher's Iris data into x, which is a standard dataset in R).

```
y = x[, 1:4]
```

extracts the first four columns from the dataset. Note that we can add matrices $A + B$ but if we multiply matrices we need to use the special symbol $\%*\%$.

Example:

```
z=as.matrix(y) # converts dataframe to a matrix
A=t(z)%*%z
dim(A) # the dimension of A - here 4 x 4
```

where $\mathrm{t}(z)$ is the transpose of z. To get the 4 x 4 sample covariance matrix we can use

```
S = var(y)
S
```

and the correlation matrix is

```
R = cor(y)
R
```

We can find the eigenvalues and eigenvectors of S as follows

```
eigen(S,symmetric=TRUE)
```

1.2 Lists

Note that the output of this command is a list, which has two parts

```
$values    (the eigenvalues)
```

and

```
$vectors   (the eigenvectors)
```

We can create a list as follows:

```
ans = list(x=0,y=0,z="")
```

where the list contains

```
 $x (real), $y (real)
```

and

```
  $z   (character strings)
```

So, we can make an assignment

```
ans$x = c(1,2,3,4,5)
ans$y = c(4,3,2,1)
ans$z = c("cat","dog")
```

and the list and can be seen by typing

```
ans
```

Lists can be very useful in providing the output of functions.

1.3 Functions

A function is a series of commands that can be called by using a simple name and arguments. Consider the following function:

```
testfun=function(n=100){
x=rnorm(n)
xsumsq=sum(x**2)
xsum=sum(x)
name= "Normal"
par(mfrow=c(1,2))
out=list(sum=0,sumsq=0,name="")
hist(x)
hist(x**2)
out$sum = xsum
out$sumsq = xsumsq
out$name = name
out
}
```

In order to see the details of the stored function, just type the function name

```
testfun
```

Now run the function for n=200 and put the answer in temp

```
temp=testfun(200)
temp
```

Now run the function for the default value of n (100 here) and put the answer in temp2

```
temp2=testfun()
temp2
```

1.4 Loops

Loops are very useful things for repeating calculations The format is as follows for s simple loop for squaring the numbers 1-100 and printing out the values:

```
for (i in 1:100){
print(i**2)
}
```

1.5 If statements

You may need to check a logical statement with an IF statment. This loop does the same things but only for integers that are divisible by 3:

```
for (i in 1:100){
if  (i/3==trunc(i/3)){
print(i**2)
}
}
```

1.6 While statements

Another useful thing is the WHILE loop, where a statement is carried out repeatedly while the condition is TRUE. If the condition is FALSE then the command is not carried out, and the program then exits the WHILE loop.

```
sum=1
while (sum <= 100){
```

```
print(sum**2)
sum=sum+1
}
print("finished!")
```

1.7 Finishing and saving your work

To quit a session of R then type

```
q()
```

You'll be asked if you want to save the 'workspace image'. In general I would say 'no' to this question, unless you have been doing some very heavy numerical calculations that will take a long time to repeat. If you say 'yes' then next time you start R the variables will all be restored. This is fine for small jobs but when you start doing lots of different jobs it can be confusing to have unecessary variables lying around.

If you wish to save your work then I would recommend writing the commands in a text file as you go along (e.g. using wordpad or notepad) and then copying and pasting them into the R window (or using the source command - see below).