GSSE401 : STATISTICAL METHODS IN SCIENTIFIC RESEARCH

LAB NOTES AND ASSIGNMENTS

Peter Diggle and Paulo Ribeiro

Introduction

These notes describe the three lab classes which form part of the GSSE402 course, but relate to the statistical methods taught in the GSSE401 course.

The labs use the R software system, which is pre-loaded onto the machines in the lab itself. There is also an add-on package called gsse401 with data-sets and additional functions to be used during the Lab sessions. This package needs to be installed by each user individually.

Alternatively, the software can be installed onto your own computer from the CD-ROM supplied, and the package gsse401 can be downloaded from: *http://www.maths.lancs.ac.uk/~ribeiro/gsse401*.

R is a open source and freely available software. The official R web-site is: *http://www.r-project.org*.

The three labs will cover the following topics:

1. Introduction to the R computing environment: loading the software, reading data from files, simple calculations, graphics, accessing the on-line help system.
2. Standard statistical functions in R: confidence intervals and t-tests, regression modelling
3. R programming: writing your own R functions, building a personal library of R functions for non-standard analyses.

Each lab will include one or more pre-prepared examples, followed by an assignment.

Setting up your account for the computer Labs

In this session we describe how to install the package which will be used during the Lab sessions. The instructions below are necessary *only for the first* Lab Section (provided that you don't remove the installed package from your disk area).

1. Every user of the University system has a *disk space number* and a *login* name. You will need to know yours. Whenever you see `YOUR_NUMBER` and `YOUR_LOGIN` in these notes, replace them by your personal disk area number and login name, respectively.
2. Find the R icon on your desktop and click on it to start the program.
3. In the command window there will be a prompt like this:

   >

   At this prompt type the following commands:

   ```
   mypacks <- 'h:/YOUR_NUMBER/YOUR_LOGIN'

   instal.packages('gsse401', contriburl =
    'http://www.maths.lancs.ac.uk/~ribeiro/gsse401', lib=mypacks)
   ```

   This will take a while to run. Type `y` when the following text will appear on your screen:

   ```
   Delete downloaded files (y/N)?
   ```

Now the package is installed under the directory gsse401 in your disk area and available for the Lab sessions.

Getting Started

The instructions in this Section applies for all Lab sessions and/or every time you start the software R. Throughout this notes we use the `typewriter font like this` to indicate commands to be typed at the prompt, names of R objects and/or output or messages issued by the system.

1. **Starting R:** To run the R system, click on the R icon on your desktop. This will open an R *command window*, at which (as its name implies) you enter R commands. Note that R is *not* a menu-driven software.

2. **Changing the working directory:** When working in the Lab you will need to change the working directory in order to be able to store data and save results. To do that go to the top menu and click on

   File → *Change dir*.

   In the dialog window change the directory to your work area in the University system. In the Lab this will be mapped on the drive *h:* and you have to know your disk space area number and your login name. You will have to enter something like:

   ```
   h:\YOUR_NUMBER\YOUR_LOGIN
   ```

3. **Getting help:** at least for your first few R sessions, you are advised to begin by typing the command

   ```
   help.start()
   ```

   which invokes the on-line help facilities for R commands, functions and data-sets. The help facilities are web-based, and you are advised to use `netscape` as the web-browser.

4. **Loading the gsse401 package:** this teaching supporting package is an add-on to R and contains data-sets and functions which will be used during the Lab sessions. The package should have been installed in the beginning of the first Lab session. To load it type:

   ```
   library(gsse401, lib.loc=".")
   ```

   Nothing will be returned but this command makes the functions and data-sets available for further use during the current session.

   To find out which data-sets included in the package type at the prompt:

   ```
   gsse401.data()
   ```

   and to list the functions included in the package type:

   ```
   gsse401.functions()
   ```


5. **Quitting R:** to finish the session type:

   ```
   q()
   ```

   The following prompt will appear:

   ```
   Save workspace image?  [y/n/c]:
   ```

   Type `y` to save all the objects in the current session. Doing that, all the existing objects will be available in new session to be started later at the current directory.
   Typing `n` the objects will be deleted and no longer available.
   Type `c` to cancel quitting and go back to the current session.

Lab 1. Simple calculations and graphics

1.1 Reading data

There are three basic ways to load data in R: entering data directly in R, reading from a file or using an already existing data-set. We now describe each one.

1.1.1 Keyboard input

Small sets of data can be input directly and stored in *objects*. Two basic types of objects are vectors (a single sequence of numbers) and data-frames (an array of rows and columns, where each row typically corresponds to a replicate of a basic experiment or observation, and each column corresponds to a variable).

Example 1.1.1.1

```
x <- c(1.4, 2.3, 5.1, 3.3, 6.1)
x
```

In this example we say that the *object* `x` is a vector with the numbers seen above. Consider now an example of a data-frame:

Example 1.1.1.2

```
dat <- data.frame(animal=c('cow','goat','cat'),weight=c(465,27.6,3.3))
dat
```

1.1.2 Input from a file

Larger sets of data will usually be read from the user's own files. Typically, the file is organized as a two-dimensional array (i.e. in rows and columns), and the data-frame used to store the data within the R environment will mirror this structure

Example 1.1.2.1

1. open a text editor (e.g. *Notepad* in Windows) and type in the following data in a two column format:

   ```
   1   160
   1   171
   1   180
   1   165
   2   190
   2   182
   2   172
   2   185
   2   180
   ```


2. save it as a text file named `mydata.txt` and close the editor.

3. to read the data in R go back to R session and type at the prompt:

   ```
   mydat <- read.table('mydata.txt')
   mydat
   ```

In the example above you could have used a spreadsheet (e.g. Excel) or any other program to enter the data. The example is still valid provided that the data are saved as a text file.

1.1.3 Loading an existing data-set

The R system has a number of data-sets included automatically. You can use the documentation of each package to find out which data-sets are available.

Example 1.1.3.1

The data set `cars` is included in the R distribution. To load, view and learn more about this data-set type:

```
data(cars)
cars
help(cars)
```

The last commands will show a description the data in a help window or in your browser (if you have typed `help.start()` before).

Example 1.1.3.2

To load and inspect the gravity data seen in the first week's lecture make sure that you have loaded the package `gsse401` before using `library(gsse401)` and type:

```
data(gravity)
gravity
help(gravity)
```

1.2 Simple calculations

R includes all the usual standard mathematical functions. When applied to vectors or arrays, they operate element-by-element. Type the commands below and inspect the results.

Example 1.2.1

```
1 + 5
```

```
y <- 1 + 5 * (3+7)/2
y
exp(1.3)
sqrt(9)
4**3
4^2
x
x2 <- x * x
x2
lx <- log(x)
lx
```

If you want to identify individual elements of vectors or arrays, you use the square bracket notation.

Example 1.2.2

```
x
z <- x[2:5]
z

w <- x[x > 4]
w
```

There are shortcuts to create sequences of numbers.

Example 1.2.3

```
x1 <- seq(0, 1,  l=11)
x1

x2 <- rep(1, 5)
x2

x3 <- rep(c(1, 2), c(3, 5))
x3

x4 <- rep(1:3, rep(5,3))
x4
```

Now consider another type of object a *matrix* and its generalisation, an *array*.

Example 1.2.4

```
z <- matrix(1:12, ncol = 4, nrow = 3)
```



```
z
z[2,3]
z[1:3, 2:3]
z[,2]
z[3,]
is.matrix(z)

z1 <- array(1:24, dim=c(3,4,2))
z1[,,2]
z1[1:2,,2]
z1[2,3:4,]
is.matrix(z1)
is.array(z)
```

Rows of data-frames are identified by numbers, just as matrices in Example 1.2.4 above. Columns of data-frames can be identified by numbers or by names using the notation $.

Example 1.2.5

```
df <- data.frame(group=rep(1:3, 4), sex=rep(1:2, c(6,6)), number=1:12)
df[4:6, 2:3]
df[,1]
df$group
df$number
df[df$group == 1,]
df[df$sex == 2, 3]
```

Another type of object is a *list*. Lists are useful to store objects of different types:

Example 1.2.6

```
ml <- list(A=1:10, B='THIS IS A MESSAGE', C=matrix(1:9, ncol=3))
ml
ml$A
ml$B
ml[[3]]
```

While typing the commands above several objects are being created in your workspace. To list the existing objects type

```
ls()
```

and to remove objects, e.g. `z1` and `l1`, type

```
rm(z1, l1)
```

1.3 Plotting

The simplest form of plotting command in R is `plot(x,y)`, where `x` and `y` are vectors of the same length. The result is a scatterplot of `x` on the horizontal axis against `y` on the vertical axis.

Example 1.3.1

```
x <- 1:10
y <- 5 + x*x
plot(x, y)
```

The `plot()` function has many optional additional arguments which allow you to customise your plots. The `par()` function allows you to control the general layout of plots. To list all of the graphical parameters type:

```
par()
```

Look at the help pages (`help(par)` to find out how these work, and note the following example, which illustrates some of the options.

Example 1.3.2

```
par(mfrow=c(3,1))
plot(x, y)
plot(x, y, type='l', xlab='this is x', ylab='and this is y')
plot(x, y, cex=4)
lines(x, y)
title('this is a plot')
```

There is also a 'demo' function which shows some of the graphical capabilities of R:

```
demo(graphics)
```

After typing the command above you will be promptedto press <return>. Each time you do so a new plot will be shown on the graphical window and the commands used to generate the plot will be shown in your command window.

Example 1.3.3

```
demo(graphics)
```


Assignment 1

1.1 Generate a sample of size 30 from the Normal distribution with mean zero and standard deviation 1, using the `rnorm()` function. For details on how the function `rnorm()` works try `args(rnorm)` and/or `help(rnorm)`. Store your sample as an array with 10 rows and 3 columns. Calculate:
(a) the sum of all 10 elements in the first column.
(b) the mean of all 3 elements in the first row.
(c) the exponential of the mean of all 30 elements.

1.2 Use the on-line help system to find out about the function `hist()`. Access the `ugclass` data-set in the gsse401 package, and produce the following four plots on a single page:
(a) a histogram of heights,
(b) a histogram of weights,
(c) a scatterplot of heights on the horizontal against weights on the vertical, using different plotting symbols for females and males,
(d) the same as (c), except using log-transformed heights and weights.

Lab 2. Standard statistical functions

R includes many functions to perform standard statistical (and other kinds of) calculations.

2.1 Means, variances and standard deviations

Example 2.1.1

```
x <- 1:9
mean(x)
var(x)
sqrt(var(x))
```

2.2 Confidence intervals, t-statistics and p-values

We shall construct an analysis of the rubber abrasion data using standard R functions (in the third lab, we'll show you how to make this easier for repeated use by writing your own functions).

Example 2.2.1

```
data(rubber)
rubber

x <- rubber$untreated
y <- rubber$treated
z <- y - x
zbar <- mean(z)
zbar
se <- sqrt(var(z)/10)
se
crit <- qt(0.975, df=9)
crit
ci <- c(zbar - crit*se, zbar + crit*se)
ci
tstat <- abs(zbar/se)
tstat
pval <- 2*(1 - pt(tstat, df=9))
pval
```


2.3 Linear regression modelling

The `lm()` function allows you to fit multiple linear regression models. Look at the help pages for details (type `help(lm)`). Its simplest use is to fit a straight line to data in the form of pairs (x, y), where x is the explanatory variable and y is the response.

Example 2.3.1

```
x <- 1:10
y <- 1 + 2*x + 0.2*rnorm(10)
plot(x,y)
result <- lm(y~x)
result
abline(result)
names(result)
result$coef
result$resid
```

Assignment 2

2.1 Experiment with using the `clt()` function on the class heights from the `ugclass` data.

(a) Are heights Normally distributed? How big a sample would you want to take before you could be confident that the *average* of your sample would be approximately Normally distributed?
(b) Repeat for log-transformed heights.

2.2 Use the `lm()` function to fit a linear regression model to the transformed `gravity` data, using time as the response and square-root-distance as the explanatory variable. Hence estimate the gravitational constant, g, and compare your answer with the one you obtained in Assignment 1 of GSSE401 (the correct value is approximately $g = 981 cm/sec^2$).

Lab 3. Writing your own functions

3.1 Why write your own functions?

An R function is just a sequence of R commands which can be invoked by typing the *name* of the function and the names of its *arguments* (the data on which the function will perform its calculations. For example, `var(x)`, `plot(x,y)`, etc.

By writing your own functions, you can store a sequence of R commands for future use, and apply these commands to future data. The recommended practice is to write the R functions in a plain text file on your machine, for example a file called `Rfunctions.R`. To open this file from the R prompt using a default text editor type:

```
edit(file = 'Rfunctions.R')
```

Then type in your functions and/or commands. When you finish typing, save and close the file. The functions/commands in the file will be automatically loaded in your R session and the file will be saved in the working directory. All functions defined within the `Rfunctions.R` file are now available for use in the current R sessionx. If there is any syntax error an message will be issued in your screen.

The object-oriented nature of R makes it very flexible. For example, R functions can be used as commands within another function, and the result of one function can be passed directly as an argument to another function without storing the intermediate result. The next example illustrate this showing three commands being concatenated in a single one.

Example 3.1.1

```
x <- 1:5
y <- exp(x)
z <- mean(y)
z

mean(exp(1:5))
```

Now, a "silly" example, to introduce you to some R features which will be useful in writing more complicated functions.

Example 3.1.2

Type `edit(file = 'Rfunctions.R')` to open the editor and into it type the following function:


```
mymean <- function(x) {
   n <- length(x)
   result <- 0
   for (i in 1:n) {
      result <- result+x[i]
   }
   result <- result/n
   result
}
```

Save and quit to go back to R and try

```
x <- 1:9
mymean(x)
mean(x)
```

You can store all your functions in the same `Rfunctions.R` file if you want to, but you must reload it, as above, each time you change the contents of `Rfunctions.R` - which you will want to do to add new functions, or (sadly) to correct mistakes in your initial attempts to define a function of your own.

3.2 Analysis of a paired experiment

We'll now write a function which allows us to carry out a paired sample analysis by a single command, rather than by a succession of commands as in Example 2.2.1.

Open your file `Rfunctions.R` again by typing

```
edit(file='Rfunctions.R')
```

and type the function below.
NOTE: The lines which begin with a 'hash' sign are comments, and whilst it is recommended that you include explanatory comments in your own functions, you can omit them for the purposes of this exercise if you like.

```
ttest <- function(xy, confidence=0.95) {
#
# xy is a two-column array containing the paired data-values
# to be analysed
#
# the second argument gives the desired confidence level,
# and is set to 0.95 by default
#
z <- xy[,2]-xy[,1]
n <- length(z)
zbar <- mean(z)
se <- sqrt(var(z)/n)
```

```
crit <- pt(n-1, 1 - (1-confidence)/2)
ci <- c(zbar-crit*se, zbar+crit*se)
tstat <- abs(zbar/se)
pval <- 2*(1-pt(n-1,tstat))
#
# the results are returned as a list, with separate components
# for the observed mean difference between the two treatments,
# the standard error of this mean difference, the test statistic
# for the hypothesis that the population mean difference is zero,
# the degrees of freedom for the test (one less than the sample
# size), and the p-value of a (two-sided) test of significance.
#
list(meandiff=zbar,se=se,test=tstat,df=n-1,p=pval)
}
```

Now we use this function to perform the t-test for the rubber data

```
data(rubber)
rubber
res <- ttest(rubber[,2:3])
res
```


Assignment 3

3.1 Experiment with the `reg()` function to gain some feel for how residual analysis can be used to detect lack of fit between a linear regression model and a set of data when the true generating mechanism for the data involves a non-linear relationship. Summarise your experiment with two or three graphs and *at most* half a page of text.

NOTE: The function `reg()` requires the package scatterplot3d. To install and load this package type:

```
instal.packages('scatterplot3d', lib=mypacks)
library(scatterplot3d, lib.loc=".")
```

if this does not work try:

```
instal.packages('scatterplot3d', contriburl =
     'http://www.maths.lancs.ac.uk/~ribeiro/gsse401', lib=mypacks)
library(scatterplot3d, lib.loc=".")
```

3.2 Write a function to fit a harmonic regression model to time series data with a known periodicity (for example, hourly or daily temperature measurements.

The mathematical model is

$$y(t) = \mu + \alpha\cos(2\pi t/\theta) + \beta\sin(2\pi t/\theta) + z(t)$$

where $y(t)$ denotes the datum at time t, θ is the known periodicity, in units of t (so, for example, for hourly data with a periodicity of one day, $\theta = 24$), μ, α and β are parameters to be estimated and $z(t)$ is random error.

The arguments (inputs) to your function should be a vector `y` containing the time-series of data, and a scalar `theta`, the known periodicity. The result (output) from your function should include the estimated values of μ, α and β, and a vector `fv` containing the values of the fitted harmonic regression curve at each time,

$$fv(t) = \hat{\mu} + \hat{\alpha}\cos(2\pi t/\theta) + \hat{\beta}\sin(2\pi t/\theta)$$

You might also like to include an option to produce automatically a plot of the data with the fitted curve superimposed.