

## Lab 1: Introduction, Plotting, Data manipulation

If you have never used R before, check out these texts and help pages;

For help with general computing and basic stats; Modern applied statistics with Splus (Venables and Ripley) is a good text. Phil Spector (<http://www.stat.berkeley.edu/users/spector>) has an on-line introduction to R and Splus. Other online tutorials are linked from the class homepage. For help with installing R on your computer, check out <http://cran.us.r-project.org/>. You can download R for windows, linux and unix at this site, as well as the add-on packages. There is an extensive manual text (pdf-file), though I would regard this as a reference, not a text to study. R is installed on the lab computers and on stat. This lab is meant to help you get started. I have included most of the commands you will need. Try to figure out the rest yourself by reading the help files and the online tutorials.

### 1 Getting Started

**START:** To start R on the stat computers or in windows, find the R icon and click on it. Note, on the stat computers R is running from the G drive, but your homedirectory is on the X drive. To be able to save your work, go to the file menu and scroll down to *change directories*. First create a new directory for the lab called e.g. LAB1. Then change the directory to X:\LAB1, or you can browse until you find the directory you want to work from. If you're running R on your own windows machine, just *change directories* after creating one for the class/lab. If you're running on linux or unix, simply call R from the prompt. R will be running from the current directory. It's a good idea to create different directories for each lab.

If you choose to work from home, I can only help you with the programming - not the set-up. Feel free to ask, I may be able to help or find out for you. Installing R on windows is easy though, so if you have a computer at home it's probably worth doing this.

If you want to use results from a previous lab you can access this by attaching the old workspace to the current one. From the file menu go to *load workspace* and choose the workspace you want to attach. In unix/linux you have to issue the command `attach(X:\OLDDIR/.RData)` at the prompt.

**HELP:** You can always get information about a command by typing `help(command)` at the prompt. If you don't know the command name, try `help.search('phrase')`. If you have used Splus before, but not R, most command names are the same, but some are not which can be confusing.

**EDITORS:** For the projects, and the labs, it is best to use an editor so you don't have to retype the commands. You can always cut-and-paste into the R command window, though this is not always possible in the windows environment on the lab computers. An alternative is to write a series of commands in your editor of choice, and save the file as "file.q". You can execute the commands in the file by writing

```
source('file.q')
```

at the prompt. Note, this will only run the commands if file.q is in your current directory.

On the lab computers, open wordpad or emacs. If you use emacs, you can name the extension of the file yourself (e.g. file.q above). In wordpad the extension is .txt if you save as a text file.

Make sure you have the correct extension when you call the file. If you call a file from a different directory than the current you need to specify this in source:

```
source('X:\\Anotherfolder/anotherfile.q')
```

**PLOTS AND GRAPHICS:** R under linux/unix opens the graphics window with the `x11()` command. Under windows you don't have to open the graphics window, it will open automatically when you plot something. If you want to display multiple plots in the graphics window, use commands `par(mfrow=c(m,n))` or `split.screen(c(m,n))` to divide into m by n small graphics displays. Read the help files on these commands.

You can save figures as postscript files with the command `dev.print(file='myplots.ps')`, or in windows by going to "save as" in the file menu. Please include figures in your reports directly, not in an appendix.

**PACKAGES:** R consists of a base package and various add-ons. For the first few labs the base package contains the necessary libraries of functions already. However, you have to call the libraries from within R to make these functions available. We will use functions in the `stats` and `MASS` libraries so start each session with the command

```
library(stats)
```

```
library(MASS)
```

To see which functions are available in each library use `library(help=stats)`.

To install add-on packages, simply go to the R cran website, identify the name (e.g. "foo") of the package you want and issue the command `install.packages('foo')` at the prompt.

**ENDING THE SESSION:** You quit R by typing `q()` at the prompt, or in windows by choosing exit from the file menu. You can elect to save the workspace in which case all functions you've created during the lab will be available when you start a new session. Note, you can only save the workspace if your current directory is under X. The results and functions are stored in a file called `.RData`. To review the commands you issued in a session look at the file `.Rhistory`. When you start another session you can launch R from the directory of choice, or change to this directory once you've got R running. To access an old workspace you need to first make dot-files visible by changing the preference in the "My documents" display to show hidden files. In R, go to the file menu and load workspace. You can browse to the directory of choice and mark the `.RData` file you wish to load. Note, you can save workspaces this way too. Note also that you can load multiple workspaces to combine work from different directories and sessions.

## 2 Data Manipulation

Let's start by exploring a data set that describes the sleeping pattern of different animals. The data is available at <http://www.stat.rutgers.edu/~rebecka/TSclass> under "Labs". The file name is `sleeptab`, which you can save as a tab-del or text file. Check if an extension to the file name like `.txt` is added when you save.

There are several variables: species name, body (bwt) and brain weight (brwt) (in kilograms and grams respectively), hours spent sleeping *not* dreaming (each 24 hours), hours dreaming, hours of sleep total, maximum lifespan, gestation, predindex (an index denoting if the animal is a predator (low) or prey (high)), sleepexp (sleep exposure index, high=exposed, low=not exposed), a combination of the latter two into a "danger index", low=not in danger from other animals, high=in danger from other animals.

Load this data set into R by calling the function `read.table`. If you open the data in an editor

first (e.g. `emacs sleeptab`) you can see that the first row of the data table consists of variable names. To take this into account when loading the data use the option `header=T`. At the R prompt call

```
sleeptab<-read.table('sleeptab',header=T)
names(sleeptab)
```

(note; maybe “sleeptab.txt” above if you saved as a text file). The last function call gives you the name of the data matrix. You can see that I used various abbreviations for the variables. Note that missing values are denoted by -999. (Before continuing, review the help files for `read.table`, `read`, `scan`).

To access a variable in a data frame like “sleeptab” you use the dollar sign `$`.

```
bwt<-sleeptab$bwt
brwt<-sleeptab$brwt
par(mfrow=c(2,1))
hist(bwt,xlab="Body weight, kg (log)")
hist(brwt,xlab="Brain weight, g (log)")
```

The command `x11()` opens a graphics window that you can split into multiple rows and columns using the `par(mfrow=c(m,n))` command. Here I split the window into 2 rows, 1 column. The histograms of body weight and brain weight look very skewed. Why is that?

Try transforming the data to (i) compress the scale of the data and (ii) symmetrize it. Taking logs is a standard approach.

```
par(mfrow=c(2,1))
hist(log(bwt),xlab="Body weight, kg")
hist(log(brwt),xlab="Brain weight, g")
```

The log transform symmetrized the data somewhat, in fact it looks almost normal. You can check normality using a QQplot.

```
par(mfrow=c(1,1))
qqnorm(log(brwt),main="QQplot of log brain weight")
qqline(log(brwt))
```

Comment on the appearance of the QQplot (long tailed/short tailed compared to normal).

Let’s simulate some normal data and check the QQplot. Read the help file on `rnorm`. Simulate three data sets, with 50, 100 and 500 observations each. Plot a histogram and a QQplot for each and comment. Now try a t-distribution with three degrees of freedom (`rt`). What do the QQplots indicate now?

### 3 Summarizing data

You should be aware of several numerical summaries: mean, median, trimmed mean, etc. These are examples of location estimates. To summarize the spread in the data we use the standard deviation (SD), or IQR (interquartile range), MAD (median absolute deviation). We will return to robust statistics later in the class, but let’s try things out using the brain weight data. We will contaminate the data, one observation only, and see how this affects the data summaries.

Let's first copy the brain weight data.

```
brwt2<-brwt
```

Now contaminate the first observation by setting it to 100 times the maximum value of the brain weight data. In order to do this you need to understand how to manipulate vectors in R. A vector element can be called with square brackets and renamed by the assign `<-` operator.

```
brwt2[1]<-max(brwt)*100
```

(Before you go on; figure out how you would assign new values to the first 3 components of the vector, how about all the odd components? `[1:3]`, `[seq(1,length(brwt2),by=2)]`)

Now compute the mean and SD of the two brain weight data sets (`brwt`, `brwt2`) using the functions `mean` and `sd`. The median is a more robust location summary, as are the `mad` and `IQR` of the spread. Read the help files for these functions, apply them and discuss the results.

## 4 Regression

Let's try our hand at linear modelling. Plot the brain weight on body weight and comment on the results:

```
par(mfrow=c(1,1))
```

```
plot(log(bwt),log(brwt),xlab="Body weight (log)",ylab="Brain weight (log)")
```

Looks nice and linear. Let's try modelling this data set using least squares (or normal error likelihood, more later in class). We use the function `lm` for linear models. Read the help file on `lm`.

```
mod1<-lm(log(brwt)~ log(bwt))
```

```
summary(mod1)
```

```
lines(log(bwt),mod1$fitted)
```

What do you think of the fit of the model? The line is the fitted values, which come pretty close to the observed data (open circles). A good diagnostic tool is the residual plot:

```
par(mfrow=c(2,1))
```

```
plot(log(bwt),mod1$res,xlab="Body weight(log)",ylab="Residuals",main="Diagnostic plots")
```

```
plot(log(brwt),mod1$res,xlab="Brain weight(log)",ylab="Residuals")
```

You want to plot the residuals against the response variable, the fitted values and predictors to see if all the dependency structure has been accounted for.

Single observations can also influence the fitting of a linear model. Which observations play a major role? It's often informative to identify outliers by name. Plot brain weight on body weight again.

```
plot(log(bwt),log(brwt))
```

```
identify(log(bwt),log(brwt),sleeptab[,1])
```

Use the left mouse button to identify points of interest and the right to quit. See any interesting observations? Which animals stand out?

## 5 Boxplots

Boxplots are very useful for (i) comparing variables, (ii) summarizing data that are a mix of continuous and categorical.

```

dream<-sleeptab$dream
totsleep<-sleeptab$totsleep
danger<-sleeptab$dang
propdream<-dream/totsleep
propdream[dream==-999 | totsleep==-999]<--999
par(mfrow=c(1,1))
boxplot(propdream[propdream!=-999] ~ danger[propdream!=-999],xlab="Danger
index",ylab="Prop of sleep hrs dream")

```

Here I've plotted a boxplot of the proportion of hours an animal dreams out of the total number of hours it sleeps, categorized by the danger index. A regular boxplot that doesn't split the data this way is obtained by simply calling `boxplot(propdream[propdream!=-999])`.

Note, `[propdream!=-999]` is a vector command that identifies the part of the vector `propdream` that doesn't have an entry equal to -999 (the `!=` command).

Read the help file on `boxplot`.

## 6 Time series data, sampling and aliasing

To create a time series object in R you use the command `ts()`. Let's simulate a data set of size 250, with a simple dependency structure:  $X_t = e_t + .8 * e_{t-1}$ , where  $e_t$  is iid  $N(0, 1)$ .

```

et<-rnorm(251)
xt<-et[1:250]+.8*et[2:251]

```

Let's assume  $X_t$  denotes montly observations of some kind, i.e. the sampling frequency is 12, and that the first observation occured in march 1980. To create a time series objects do

```
xt<-ts(xt,start=1980.25,deltat=1/12)
```

To plot the time series object and check it's dependency structure graphically

```
ts.plot(xt)
acf(xt)
```

, where `acf` shows you the estimated pairwise correlations of  $X_t, X_{t-h}$  as a function of  $h$ .

What do you notice about this time series?

You will work with the *sunspots data* for the remainder of this section. You can download the sunspots data from the class home page. The sunspots data set consists of the monthly averages of sunspots observed, starting January 1749 until March 1977. There are 2739 observations. The data set is accessible directly from within R: simply issue the command `data(sunspots)` to retrieve it. Plot the sunspots data using the `ts.plot` command. Use the `split.screen` or `par(mfrow)` commands to plots the entire data set, and a subset of the data set in the same graphics window. You should see a periodicity in the data set. What is it (approximately)? Use the `locator`, `identify` commands to help you answer the question. You can also plot the autocorrelation function (`acf`) of the sunspots data using `acf`. Read the help file. Adjust the option `lag.max` so you can see the periodicity. Note, it is several years so the `lag.max` has to be set to `12*yr`s, where `yr`s is the number of years you think safely covers the periodicity in the data.

Now sample the data set with a different frequency. Plot the data sampled at every 10 years,

every 4 years and every 7 years. What do you see? Plot the acf of the sampled data as well. Look at the data sampled at every 7 years. What is the periodicity in this data set (approximately)? Why? Draw a sketch to explain. This phenomenon is referred to as *aliasing*. Example:

```
tsamp7<-seq(1,length(sunspots),by=12*7)
newsunspots7<-ts(sunspots[tsamp7],start=1749,deltat=7)
ts.plot(newsunspots7)
```

Simulate a periodic function like `sin` or `cos`, choose an appropriate frequency and number of observations. Convert the simulated data into a time series object and plot it. Play around with different sampling schemes to make the periodicity disappear, remain and to see the aliasing effect (a periodic signal of another frequency than the one you simulated). What is the rule for sampling frequency?