THE UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF STATISTICS

STAT 447B Methods for Statistical Learning (2014/15 Term 1)
Lab 0

10/11 September, 2014

This course makes intensive use of computer software R. We assume that you have had introduction to R in previous statistics courses such as STAT 305 or STAT 306. This lab recaps some of the procedures in the R programming environment that you will find useful in this course.

1 Introduction

1.1 What we expect you to know how to do in R

- Writing for-loops and functions
- Reading data from text files and csv files (e.g. `read.csv`, `read.table`)
- Accessing and manipulating vectors, matrices, and data frames
- Basic math functions such as computing the row means of a matrix (e.g. `rowMeans`, `colMeans`)
- Matrix algebra such as dot product and transposing and inverting a matrix (e.g. `solve`)
- Installing packages, using packages, and reading package documentations (e.g. `library`, `?`)
- Basic plotting functions (e.g. `plot`, `hist`)
- Fitting models, for example, using formula (e.g. `lm(y~x1+x2+x3)`)

If you are not familiar with any of the topics above, it is expected that you will bring yourself up to speed rather quickly as the labs and the assignments will assume prior knowledge of these topics. You are welcome to talk to us if you need help on any of the topics above during the office hours.

1.2 What we will introduce in this lab

- Installing, loading packages and using predefined functions
- Manipulating matrices
- Writing your own functions (and running them!)
- Variable scope
- A bit of debugging

Refer to the R file of this lab on how to do these. Make sure you understand them since we will be using R very frequently, and these basic commands are crucial throughout this course.

1.3 Advanced R topics that we will NOT introduce

- Object-Oriented Programming in R
- Interface R with Python, C/C++, Fortran, or other languages
- Parallel programming in R
- How to write your own packages

This list serves to introduce you to possibilities of R, which you can explore in your own time. The list is not exhaustive by any means.

2 Review of Basic R Programming

2.1 Installing and loading packages, and running predefined functions

Packages or libraries usually contain functions for a specific area, and sometimes datasets as well. For instance, the package `SemiPar` implements semiparametric regression. Some other packages useful later in this course include `MASS`, `splines`, `tree`, `e1071`, among others. Note that these names are case-sensitive.

Before you use a package, you need to install it first - basically it means copying the relevant files to your computer. You can either use the "Packages" command in the tool bar or the `install.packages` command in the console. Choose a mirror closest to your physical location to save download time (for larger packages).

After installation, the package is in your computer but not yet invoked. At the start of every R session you should load necessary packages so that you can use functions contained therein. To load a package, it is probably faster to use the command `library`.

To use the functions just type the function name and the necessary arguments. It is extremely important that you consult the manual by using the command `?` or `help`. For each function there is a detailed description of its use and parameters accepted.

2.2 Manipulating matrices

Vectors and matrices are important objects that you will come across in R every now and then. Vectors are usually generated via the commands `c`, `rep` or `seq` for combining objects, generating repeated values and sequences respectively. Matrices are usually generated via `matrix`, `cbind` or `rbind`, with the first command generating matrix from scratch while the other two combing vectors or matrices into larger matrices.

It is important to remember that `*` performs elementwise multiplication in R. If you want to do matrix multiplication, you need the command `%*%` instead. Some other useful commands are `t` for transpose, `solve` for solving system of linear equations or finding the inverse of a matrix, and `rowSums` and `rowMeans` for the obvious functions and their column counterparts.

One of the reasons why R is so popular in the statistical community is its ability to extract elements from a matrix easily. Some examples are given in the R file.

2.3 Writing and running your own functions

You can define your own function, for example the following:

```
myfunc <- function(A,B,C){ #Solve equations of the form Ax^2+Bx+C=0
    discrim <- B^2-4*A*C;
    if(discrim < 0) stop("No real roots!");
```

```
    r1 <- (-B-sqrt(discrim))/2/A;
    r2 <- (-B+sqrt(discrim))/2/A;
    return(c(r1,r2));
}
```

Such constructs usually consist of the following:

- `myfunc`: The name of your function
- `function`: Declares that `myfunc` is a function
- `A,B,C`: Input parameters of your function
- `return`: A statement to indicate your outputs

After defining a function you can use it just like the built-in ones. For example, `myfunc(2,3,-5)` gives you a vector $(-2.5, 1.0)$. Notice that you can return more than one number in the form of a vector, matrix or list.

2.4 Variable scope

Bear in mind that the variables involved within a function are local to the function only, i.e. they cannot be called from outside. If a variable is referenced in a function yet is undefined inside the function, R will search for the variable in the parent environment. Mistakes are easily made when the R file grows and gets complicated. It is therefore desirable to check that variables used inside a function are either passed as an argument or defined within the function.

On the other hand, the fact that variables defined are confined to the function's environment allows you to recycle variables across different functions. For example, you can use i as the dummy variable for the counter in loops for different functions without worrying that the counters get mixed up when you call them.

2.5 Debugging

The function `print` displays the value of the argument variable on the console. It is very useful to debug your program when you want to know which variable does not perform as expected. You should uncheck the option "Buffered output" in the "Misc" menu so that items to print are not buffered; i.e. they are printed to the console when executed, especially when you print something in the middle of a long loop.

Meanwhile, the function `debug` helps you debug a function line-by-line. You can view the contents of a variable at any time and can therefore check if your program is running correctly.

3 Make your own `lm` function

Here you will try making your (reduced version of the) `lm` function for the linear model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. This function should accept a design matrix $\boldsymbol{X}$ and a response vector $\boldsymbol{y}$ as arguments, and return the fitted parameters $\hat{\boldsymbol{\beta}}$. Your function should look like this:

```
linear <- function(y,X){
    beta <- ???
    return(beta)
}
```

Replace `???` above by the actual matrix manipulation for computing $\hat{\boldsymbol{\beta}}$. When you complete this, try to modify the code such that the function returns both $\hat{\boldsymbol{\beta}}$ and $\hat{\sigma}$, the estimated standard deviation of the random error.

Extra questions

Submit the R code to the following questions on *Connect:*

1. What is your code to replace the `???` in the function above?

2. Consider the `trade.union` data set used in the lecture slides (the data set is in the `SemiPar` library). Suppose we want to learn the effect of age and years of education on wage, and we want to fit a linear regression with wage being the response and both age and years of education as covariates (with intercept). How would you do that using the function `lm` and your own function `linear`? In addition to the R code, submit also the parameter estimates of this linear regression.

Note: You do not need to submit the code on calculating the estimated standard deviation of the random error.