Math 324 Fall 2004
Computing Guide: Introduction to R

Dr Ben Bolstad
bolstad_math324@bmbolstad.com
http://math324sfsu.bmbolstad.com

Introduction to R

R is a free, GPL licensed, implementation of the S statistical programming language. It provides a rich environment for carrying out statistical analysis and creating your own functions. In syntax it is close to the S-plus commercial implementation, so in general, documentation about S-plus applies also to R.

The recommended procedure is you first type your code into a text file and then copy and paste it into your R process to have it evaluated. The text editor you use would depend on your operating system (note Microsoft Word is not a text editor).

The first step is to launch R. How you do this will depend on your operating system (you will need to either click something or type R at a command prompt). Once inside of R, the default prompt is ">". A good place to start is with the internal help system. This is available with the command

```
> help.start()
```

Assignments are done with the two symbols "<-". In other words, a less-than sign immediately followed by a dash. For instance, the following command assigns a vector of 10 numbers to the variable a

```
> a <- c(2:8,6,6,7)
```

Simply type the variable name to see its value. The function `c()` is for concatenate, and is used to create vectors. The expression `2:8` does the same thing as `c(2,3,4,5,6,7,8)`. Let's do some simple arithmetic. Try the following:

```
> sum(a)
> b <- 5*(a+1) - 7
> d <- b / a^2
> a>5
> e <- a[a>5]
```

Notice that when no assignment is made the results of the computation are print to the screen. Such results are stored temporarily (until the next computation) in the variable `.Last.value`. Notice that the standard arithmetic operations are performed component-wise when applied to a vector (also true for matrices), and the standard order of operations is followed. The operation > is a logical comparison, and the final expression will store in the variable e those elements of a that exceed 5 . The square brackets are used to index elements of a vector. To see how many elements the vector `e` has, use `length(e)`. To see all of the variables in your work area type `ls()` or `objects()` for a list. Note what happens if the parenthesis are omitted.

Here are some examples of matrices:

```
> x <- matrix(1:12,3,4,byrow=T)
> y <- matrix(c(7,3,4),2,3)
> z <- matrix(rnorm(30),ncol=3)
> w <- diag(rep(1,4))
```

Notice what happens if `byrow=T` is omitted from the first example. As for the second, if the number of values divides the number of entries in the matrix evenly R will cycle through until the matrix is full. This can lead to some nasty bugs. The third makes a 10 by 3 matrix of random standard normals. The fourth uses the useful `rep()` function to make a 4 dimensional identity matrix. Matrix multiplication is done with the symbol `%*%` and the function `t()` transposes a matrix. A very useful function is `apply()`. It applies a function to rows or columns of a matrix. For example,

```
> apply(z,2,mean)
```

Takes the means of the three columns of the matrix `z`. See also `tapply()`, `sapply()`, and `lapply()` for cousins of this function.

Let's talk graphics. To get a histogram of `d`, we first need to open a graphics device.

```
> hist(d)
```

The brackets `()` denote that the object is a function. The first one above does not require any arguments. One of the powerful features of R is the ease with which you can write your own functions. For example, here is a simple function that returns the standard deviation of a vector of numbers.

```
> sd <- function(x) {sqrt(var(x))}
```

With more complicated functions you will want to create them in an editor and then load them into R using the R function `source()` or by copying and pasting them into . We can now call this function just as if it was any other function. For example,

```
> sd(a)
```

finds the SD of the vector a.

Here is an example of a more complicated analysis. Suppose that a data file called "fish.dat" exists in the same directory that I am working in and contains two columns of data. The first is the weight for 68 trout caught in Bear Lake, and the second is the length. We want to perform a linear regression of weight on length and produce residual plots all on the same figure. Use your editor to generate your own data if you want to practice this exercise. The symbol # is the comment symbol. Use `help.start` for details regarding the functions (note it is not important that you understand right now what all the following functions do) .

```
> fish <- read.table("fish.dat") # read in data
> dimnames(fish) <- list(NULL, c("weight","length")) # give labels
> fish <- as.data.frame(fish) # makes life easier below
> fit1 <-lm(weight ~ length, data=fish) # fit the linear model
> fit1.sum <- summary(fit1) # get results from model fit
> sink("fish.fit") # output fit results to a UNIX
> print(fit1.sum) # file called "fish.fit"
> sink() # close the file when done.
> par(mfrow=c(2,2), oma=c(0,0,2,0)) # partition graphics region
> plot(fish$length, fish$weight) # scatter plot
> abline(fit1) # add regression line
> plot(fit1$fitted, fit1$resid) # residual plots
> plot(fish$length, fit1$resid) # more residual plots
> qqnorm(fit1$resid) # normal quantile plot
> qqline(fit1$resid) # best line for quantile plot
> mtext(outer=T,side=3,"Linear Regression of Weight and Height") # title
```

To leave R, use

```
> q()
```

Reading in data

Typically, you would begin your analysis by reading the data in to R. The general function for reading text files containing data with variables stored in columns and cases (individuals) stored in rows is `read.table`. For instance to read a text file *excitingdata.txt* and store it in a variable names `Exciting` type

```
Exciting <- read.table("exciting.data")
```

If the first row of your data file contains column names you should instead type

```
Exciting <- read.table("exciting.data",header=TRUE)
```

and if it is tab-delimited them use

```
Exciting <- read.table("exciting.data",header=TRUE,sep="\t")
```

Histograms, stem-plots, boxplots and scatterplots

The command in R for drawing histograms is `hist()`. Suppose that x is a vector containing numerical data. Then

```
> hist(x)
```

will draw a histogram. If we wanted to specify a particular number of cells (histogram bins) then we would use

```
> hist(x,breaks=10)
\end{verbatim)
Another way to do this would be to specify where the breaks should fall
\begin{verbatim}
> hist(x,breaks=seq(min(x),max(x),length=11)
```

and of course you can find more about drawing histograms using *?hist*.

Stem plots are created using `stem()`. For example

```
> stem(x)
```

will create a stem plot of the data vector x.

Suppose that x contains a quantitative variable and y contains a categorical variable. Each x, y pair is a recording for a single individual. To create a boxplot of the quantitative variable type

```
> boxplot(x)
```

and to create boxplots of the quantitative variable for different values of the categorical variable

```
> boxplot(x ~ y)
```

A scatterplot is useful for comparing two quantitative variables (each measured on the same individuals). Suppose x,z contain quantitative variables each measured on the same set of individuals. Then

```
> plot(x,y)
```

will plot the y variable on the vertical axis against the x variable on the horizontal axis.

Simulating random data

To simulate random normal distribution data use the `rnorm` function. The command

```
> x <- rnorm(100)
```

will generate 100 random standard normal distribution values (mean 0, sd 1) and store it in the variable x. The command

```
> x <- rnorm(1000,mean=10, sd=5)
```

generates 1000 random numbers from a normal distribution with mean 10 and sd 5.

To simulate random numbers from the exponential distribution use the `rexp` function. eg

```
> y <- rexp(50)
```

generates 50 random numbers from the exponential distribution with mean 1.

A few hints for working with the cereals.dat data

Once you've read the data into R, you might want to use the `attach()` function. Suppose that `cereals` is the name of the variable into which you read the data. Then try

```
ls()
attach(cereals)
ls()
```

notice something? You should now have direct access to the each of the variables stored by name. This will make it a lot easier to create plots of the data. Another way to access the data stored in a `data.frame` or `matrix` is to use subscripting. For example

```
cereals[,2]
```

would return the second column of the `cereals` dataframe. While

```
cereals[15,]
```

would return the fifthteenth row of the `cereals` dataframe.

You will probably want to use `freq=FALSE` as a parameter to your call to `hist` in the simulation to ensure that you get plots of relative frequencies (ie probabilities).

Where to get more help

The documentation for R contains a lot of specifics on each command. On the R website there is also lots of documentation. I recommend the manual "An Introduction to R". It should contain sufficient detail for this class.