# An R Tutorial

## 1. Starting Out

R is an interactive environment for statistical computing and graphics. This tutorial will assume usage of R 2.0.0 on a PC. However, except in rare situations, these commands will work in R on UNIX and Macintosh machines as well as in S-Plus on any platform. R can be freely downloaded at `http:\\r-project.org\`.

Before reviewing R commands, it is important for you to set up a working directory for your R work. (Particularly if you plan to use R in the public computer labs.) In order to set up R, follow these steps:

(a) Create a folder called Statistics in your userspace.

(b) i. Open R (Start → All Programs → R → R 2.0.0)
ii. In R go to File → Change dir... → `H:\Statistics` → OK
iii. In R go to File → Save Workspace... → `H:\Statistics\.RData` → Save
iv. In R, type:
```
> q( )
```
v. You will be prompted to save your workspace image. Answer "yes".

(c) In your userspace open the Statistics folder. Double click on .RData to open R.

Once you enter R, you will receive the following prompt: >

To exit R, issue the following command:

```
> q()
```

Note: You should always type your code in a **separate** text file. (Notepad is preferable to Word because it won't try to auto-correct your code.) You can cut and paste from notepad into R and keep a copy of everything you've done in the text file. It is good practice to put your data, written code, and .RData file all in the same folder (e.g. "Statistics".)

## 2. Basics

# Here are some illustrations of R's interactive nature:

```
> 5          # you type in a 5 at the prompt (note that anything on a line following
             # a '#' sign is considered a comment and will not be processed)
[1] 5        # → here a 5 is returned

> 5 + 4      # adding two numbers
[1] 9        # → the sum is returned

> 5^3        # will compute 5^3
[1] 125
```

```
 > pi          # seeing R's on-screen representation of pi (rounded, of course)
[1] 3.141593

# To create a vector of numbers:
 > c(3,4,7)  # concatenate
 [1] 3 4 7

# To create an object which will be stored in your directory:
> x<- c(3,4,7)

# To list the objects in your directory:
> ls()

# To remove an object (e.g., an object called temp) from your directory:
> rm(temp)

# To view x:
> x
[1] 3 4 7

# To view the second element of x:
> x[2]          # note that in general the square brackets [ ] are used to
                # pull out an entry from a vector or matrix
[1] 4

# To view the first and third elements of x:
> x[c(1,3)]  # note that x[1,3] would fail
             # in general, the round brackets ( ) are used in functions like c()
[1] 3 7


# An example of how to create a 2 × 3 matrix:
> y <- matrix(c(1,2,3,4,3,4),2,3)

# The last two arguments of the matrix function are the number of rows and columns,
# respectively. If you would like the numbers to be entered by row instead of the default
# by column, add an additional argument, byrow=T.
# matrix(vector of data, numrows, numcols, byrow=T)

# To view the matrix, y:
> y
      [,1] [,2] [,3]
[1,]   1    3    3
[2,]   2    4    4
```

```
# To view the first row of y:
> y[1,]
[1] 1 3 3

# To view the second and third columns of y:
> y[,c(2,3)]
      [,1] [,2]
[1,]   3    3
[2,]   4    4

# To find help on any R function:
> help(function)

# For example, to find help on the function 'matrix':
> help(matrix)

# You can create your own functions using the following format:
> newfunc <- function(args)
             {
              commands
             }

# An example of a user written function to find the approximate derivative of a function f:
> derivf <- function(x)
            {
             (f(x+0.001) - f(x))/0.001
            }

# Let's say that f has been defined as:
> f <- function(x)
       {
        x^3 + x^2 + x + 10
       }

# The derivative of f is obviously 3x^2 + 2x + 1.
# Let's evaluate this true derivative and the approximate derivative (using
# the function derivf) for several values of x:

> x <- c(1:6)          # this produces the vector (1,2,3,4,5,6)
> 3*x^2 + 2*x + 1      # this is the exact derivative
[1] 6 17 34 57 86 121
> derivf(x)            #this is the approximate derivative
[1] 6.004001 17.007001 34.010001 57.013001 86.016001 121.019001
```

## 3. Some Useful Features and Functions

```
# Let's create our original object vector, x:
> x <- c(3,4,7)
> x
[1] 3 4 7

> x[-3]          # this will provide all numbers in a vector except for the third
[1] 3 4

> x[x >=5]       # this will produce all numbers in x that are greater than or equal to 5
[1] 7

# Notice that arithmetic operations are applied to each element in a vector. For example:
> x+3
[1] 6 7 10

# Let z be a new vector:
> z <- c(1.5, 1/6,1/3)

# View only the first two decimal places of z:
> round(z,2)
[1] 1.50 0.17 0.33

# View z in reverse order:
> rev(z)
[1] 0.3333333 0.1666667 1.5000000

# Order z from smallest to largest:
> sort(z)
[1] 0.1666667 0.3333333 1.5000000

# Determine the length of the vector z:
> length(z)
[1] 3

# Find the maximum value of z in two ways:
> max(z)
[1] 1.5
> sort(z)[length(z)]
[1] 1.5

# Identify the ordering of z:
> order(z)
[1] 2 3 1      # i.e., the 2nd number is the min and the 1st number is the max
```

```
# Now, create a sequence of numbers:
> seq(1,3,length=5)
[1] 1.0 1.5 2.0 2.5 3.0

# Create the same sequence in a slightly different way:
> seq(1,3,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0

# Create another sequence by going from 3 to 1:
> seq(3,1,by= -0.5)
[1] 3.0 2.5 2.0 1.5 1.0

# To randomly sample from an existing vector:
> sample(x,10,replace=T)
[1] 3 4 7 4 4 3 4 3 7 4

# Or to randomly sample from a sequence of numbers from 1 to 500:
> sample(1:500,10,replace=F)
[1] 302 479 356 438 386 413 90 86 124 456

# Missing values are represented by NA:
> w<-c(3,4,NA,5)
> w
[1] 3 4 NA 5

# Some functions ignore NA's:
> sort(w)
[1] 3 4 5

# Other function, by default, do not:
> mean(w)
[1] NA

# Two ways to determine the mean of the non-missing values are:
> mean(w,na.rm=T)
[1] 4
> mean(w[!is.na(w)])      # where ! indicates 'not' and is.na subsets only NA values
[1] 4

# Some functions, use 'na.omit=T' instead of 'na.rm=T'

# Create a new vector by concatenating x and z:
> c(x,z)
[1] 3.0000000 4.0000000 7.0000000 1.5000000 0.1666667 0.3333333
```

```
# Create a matrix by treating x and z as rows:
> rbind (x,z)              # rbind binds by rows
   [,1]     [,2]          [,3]
x  3.0      4.0000000     7.0000000
z  1.5      0.1666667     0.3333333

# Create the same matrix, but, first treat x and z as columns:
> t(cbind(x,z))           # cbind binds by columns
   [,1]     [,2]          [,3]
x  3.0      4.0000000     7.0000000
z  1.5      0.1666667     0.3333333
```

## 4. Distributional Functions

```
# Determine the height of a normal(0,2) random variable at x=1:
> dnorm(1,mean=0, sd=2)
[1]  0.1760327

# Determine the area to the left of 0 for a normal (0,1) random variable:
> pnorm(0)        # for the normal dist, the default mean is 0 and sd is 1
[1] 0.5

# Determine the quantile corresponding to the probability of 0.9 for a
# normal(0,1) random variable:
> qnorm(0.9)
[1] 1.281552

# Generate three pseudo-random numbers from a normal(0,3) distribution:
>rnorm(3,0,3)
[1]  2.5656294  2.3807115 -0.7589694

# Functions for other distributions work in a similar way. Remember that any manual
# can be accessed using the help( ) function. For example, if you want the syntax for
# the t distribution function, try help(rt)
```

## 5. Plotting Data

# R has some very nice graphics capabilities. We'll only cover a minimal amount of
# information here.

# Let's create a mock simple linear regression data matrix:

```
> regdata <- matrix(c(1:6,c(10,15,19,26,32,37)),6,2)
> regdata
      [,1]  [,2]
[1,]    1     10
[2,]    2     15
[3,]    3     19
[4,]    4     26
[5,]    5     32
[6,]    6     37
```

# Here, the first column is the explanatory variable and the second column is the
# response variable.

# To plot the data points:

```
> plot(regdata[,1], regdata[,2])    # the x-axis variable is the first argument
```

# To plot the data points and have the points connected by lines:

```
> plot(regdata[,1], regdata[,2], type='b')
```

# To plot only a line:

```
> plot(regdata[,1], regdata[,2], type='l')
```

# To add meaningful axis labels and a title to the scatterplot:

```
> plot(regdata[,1], regdata[,2], xlab='expl var', ylab='response',
      main='Regression data scatterplot')
```

# To set your own limits for the y-axis:

```
> plot(regdata[,1], regdata[,2], ylim=c(0,40))
```

# To plot a histogram of uniform(0,1) random data:

```
> hist(runif(1000, min=0, max=1))
> hist(rnorm(1000, mean=0, sd=1))
```

# The function *hist* allows for several useful options. Type '*help(hist)*' for details.

# To plot the pdf of a normal(0,1) random variable:

```
> plot(seq(-4,4,0.001), dnorm(seq(-4,4,0.001)), type='l')
```

# To view two graphs on the same plot region (i.e., page), one on top of the other:
```
> par(mfrow=c(2,1))
> plot(seq(-4,4,0.001), dnorm(seq(-4,4,0.001)), type='l')
> plot(seq(-4,4,0.001), dnorm(seq(-4,4,0.001), mean=0, sd=0.75), type='l')
```

# To add a plot to a current figure use *points* or *lines*:
```
> plot(seq(-4,4,0.001), dnorm(seq(-4,4,0.001)), type='l')
> points(seq(-4,4,0.001), dnorm(seq(-4,4,0.001), mean=0, sd=0.75))
> lines(seq(-4,4,0.001), dnorm(seq(-4,4,0.001), mean=0, sd=1.25))
```

# Enter '*help(par)*' to see the many options available for dealing with graphics. Many
# of the options (e.g., xlim, ylim) can be used directly with the plotting functions.

## 6. Dealing with Data and Data Files

# To enter in a data vector at the keyboard:
```
> u<- scan()
1: 5 3.2 5.1
2: 2.1 5 7.2
7:
> u
[1] 5.0 3.2 5.1 2.1 5.0 7.2
```

# To read in an external dataset called data.txt:
```
> tempfile <-scan('data.txt')
```
# Notice that scan brings the data into R in a vector format regardless of the original format.
# To read in an external dataset in a more flexible format use *read.table( )*.
# Check *help(read.table)*. Also, note that both *read.table()* and *scan()*
# have several options which are useful for datasets with interesting features, such as the
# column names on the first row, spacing by special symbols, etc.

# Writing data to a file, tempout1, by column:
```
> write(u, file='tempout1')
```

# Writing data to a file, tempout2, by row:
```
> write(t(u), file='tempout2')
```

# Writing another vector to the same file by row:
```
> write(t(x), file='tempout2', append=T)
```

# To read in R code from an external file, code.r:
```
> source('code.r')
```

## 7. Other Useful Functions

There are hundreds of internal R functions not discussed in this tutorial. Some which you may find useful include:

`if, ifelse, set.seed, array, sample, deriv, integrate, gamma, cumsum, diff, sign, summary, median, quantile, cut, table, tabulate, cor, outer, ceiling, floor, prod, diag, dist, rep, paste, substring, cat, print, unlist, data.dump, data.restore, options, density, scatter.smooth, stem, boxplot, qqnorm, qqplot, arrows, legend, text, mtext, abline, pairs, persp, contour, postscript.`

Also, there are hundreds of statistical functions to perform hypothesis testing, linear and non-linear model analyses, generalized linear model and survival analyses, contingency table analyses, multivariate analyses, nonparametric analyses, classification trees, time series analyses, clustering, bootstrap and jackknife techniques, etc.

## 8. Useful References


Dalgaard P. (2002). *Introductory Statistics with R.* New York: Springer-Verlag.

Ripley B.D., Venables W.N. (2002). *Modern Applied Statistics with S (4ed).* New York: Springer-Verlag.

Spector P. (1994) *An Introduction to S and S-Plus.* Belmont, CA: Duxbury Press.

`www.r-project.org`

`http://cran.r-project.org/doc/contrib/refcard.pdf`