An Introduction to R

To get R:

1. Download and install R (it's free) from the website http://cran.r-project.org There are versions for Linux, Windows and Mac.
2. Extra tutorials for R at http://cran.r-project.org/doc/manuals but start here first.

After you call up the program you will see a window with a "command prompt" which looks like

```
>
```

This is where you will type your commands. First try using R as a calculator by typing the following expressions (followed by a return)

```
> 5+3         # anything after the ''#'' is a comment
> 10*10       # ''*'' is multiplication
> (5+4)/3     # you can use parentheses to ''chain'' operations together
> 2^3         # 2^3 = 2*2*2
> sqrt(100)   # the square root of 100
```

R has most any mathematical function you can think of such as sqrt(), sin() ... mostly with easily guessable names. Expressions using the logical operators ==, !=, <, > give Boolean values (T,F)

```
> 4 > 3       # this evaluates to T (true)
> 4 < 3       # this evaluates to F (false)
> 1 == (4/4)  # this evaluates to T
> 1 != (4/4)  # this evaluates to F
```

It is possible to have *variables* that hold values in your program. Most strings beginning with an alphabetic character will be treated as variables by R. Try typing the following lines in succession

```
> x = 3       # the variable x now holds the value of 3
> y = x*x+x   # the variable y now holds x*x+x = 12
> y           # print the value of y
```

Vectors

Vectors are collections of numbers rather than single numbers (variables). You can think of a vector as a row of boxes with each box containing a number. One of the nicest aspects of R is the way it handles vectors. Here are a several ways to create vectors:

```
> x = 1:100                  # x is now the vector (1,2,...,100)
> y = seq(-1,1,length=100)   # y consists of 100 evenly spaced values from -1 to 1
> z = c(1,4,8,20)            # z is the vector (1,4,8,20)
```

If you want to see the inidividual *components* of a vector use the square braces:

```
> y[1]          # the first component of y (= -1)
> y[20]         # the 20th component of y
```

R can perform operations on entire vectors at once (when they make sense)

```
> z = 4*x   # z is now a vector of the same length as x (100).  z[1] = 4*x[1], z[2] = 4*x[2] etc.
> z = x+5   # z is x with 5 added to each component
> a = x+y   # vectors of same length can be added: a[1] = x[1]+y[1], a[2] = x[2]+y[2], etc.
> a = x*y   # or multiplied, subtracted, or divided
```

Plotting Try the following

```
> x = seq(0,1,length=100)
> y = x^2                    # y = x squared
> plot(x,y)                  # plot with points (x[1],y[1]) \ldots, (x[100],y[100])
> plot(y)                    # same as plot(1:length(y),y)
> plot(ht,low_hz)            # where ht are heights in inches and low_hz is lowest note in hz
```

Source Files You will be given assignments to write simple programs in R and this usually requires some trial, error and iteration. I recommend the following procedure: Create a "source" file in any text editor containing your R commands. This could be emacs or the Windows "Notepad" or whatever you are comfortable using. Do not use a word processor such as "Word." Suppose you create the following file named "myprog.r"in your editor:

```
x = seq(0,20,length=100)
y = x*sin(x)
title("my function")
print("values are: ")
print(y)
```

You can now run your program from R simply by using:

```
> source(``myprog.r'')
```

This technique allows you to write a program in the usual incremental way by repeatedly making minor changes to your file and "sourcing" the file. If you want to get a hard copy of the printout and the plot (for example, to submit as your homework), do the following

```
> postscript("myplot.ps")   # direct future plots to postscript file ``myplot.ps''
> sink("myout.txt")         # write future text output to ``myout.txt''
> source("myprog.R")        # run the program you created
> dev.off()                 # redirect plots to screen.  Don't forget this!
> sink()                    # redirect output to screen. ditto.
```

Quitting and help

```
> help("plot")  # gives help for the plot function.  Of couse this works for other functions.
> q()           # to quit the program. Hope you had fun.
```