[What follows is sort of a mish-mash of R commands shortened for this course thrown together to make a lecture. Some more will come later as we need them. R is very similar to MATLAB in usage. The syntax differs though: use ( ) for functions and [ ] for data. Some more details on using R at an elementary level can be found on the website http://www.math.csi.cuny.edu/verzani/tutorials/simpleR/]

1 Using R at CSI

R is the software we will use in the computer lab. R is free software. Yuu can download a copy and run it anywhere you like. To do so, you might wish to download a copy from http://cran.us.r-project.org/ (more information is on http://www.r-project.org), burn a copy to CD and take this with you.

2 starting R

R is started (on windows) by double clicking the R icon. This will open up a window where your R windows will reside. There are 3 main types: **the command line**, *the help windows* and the *plot windows*. The command line is the first one open and allows you to *type* your commands (unlike MINITAB, the main interface is through the keyboard and not the mouse). The command line has a start up message that looks something like

```
R : Copyright 2003, The R Development Core Team
Version 1.7.1  (2003-06-16)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

>
```

That *prompt* (>) is where we type our commands. For example, let's see if R is able to add

```
> 2+2
[1] 4
```

You hit enter after typing 2+2 and R responds with the answer of 4 as expected. You don't type the prompt although it is printed above.

3 Entering data in R

R has many add on packages that can be loaded with the `library()` command. FOr example, `library(MASS)` will load the external `MASS` package. Each package can contain built-in data sets for examples. These may be read in with the `data()` command as in `data(faithful)`.

However, most data sets will not be built-in. We may have to type them in. For this there are several commands to learn. We also may read them in from other sources. That is covered later.

3.1 using `c()`

The basic means to read in data is the `c()` function which combines data. If the data is

```
74, 122, 235, 111, 292, 111, 211, 133, 156, 79
```

To store this data in R we use a *data vector*. These can be made with the `c()` function which **c**ombines its arguments as follows

```
> whales = c(74, 122, 235, 111, 292, 111, 211, 133, 156, 79)
```

The `c()` function also can combine a vector and a number intelligently as in

```
> whales
 [1]  74 122 235 111 292 111 211 133 156  79
> whales = c(whales,90, 91,110)
> whales
 [1]  74 122 235 111 292 111 211 133 156  79  90  91 110
```

If you would prefer to type your data into a file and read it in, then the `scan()` function will work. If your file is `data.txt` and contains just the numbers

```
  74 122 235 111 292 111 211 133 156  79
```

Then the command `scan(file="data.txt")` will work. Alternately, a file can be browsed with `scan(file=file.choose())`

3.2 regular data

Arithmetic sequences can be generated easily.

The colon operator `a:b` will generate values $a, a+1, ..., a+k$ where $b \leq a+k < b+1$.

```
> 1:5
[1] 1 2 3 4 5
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
```

If we wish to skip by a value different from 1, we can use the `seq()` function

```
> seq(10,100,by=10)             # skip by 10
 [1]  10  20  30  40  50  60  70  80  90 100
> seq(10,100,length=5)          # 5 numbers
[1]  10.0  32.5  55.0  77.5 100.0
```

3.3 random data

Random data is generated with the "r"-functions. For example, `rnorm`, `runif`, `rexp`. The typical usage is `rnorm(n)` but the default parameter values can be changed easily as in `rnorm(n,mean=5,sd=56)`.

```
> rnorm(5)
[1] -0.1033496 -0.2584673  1.4156679  1.1250157  0.3642154
> rnorm(5,mean=10,sd=2)
[1] 10.468320 10.036919  9.001509  9.210126  9.338071
```

3.4 using `data.entry()`

A spread sheet interface exists, but is primitive. It will let you modify a dataset, but not make a new one. To edit the value 156 above to 165 we can type

```
> data.entry(whales)
```

Make the change and then quit the program. YOur changes are recorded into the variable `whales`.

If you have a big dataset in spreadsheet format, it is better to export to CSV format, and read in with `read.csv()`.

3.5 using `read.table()`

Finally, you may wish to type in data involving multiple variables. If the data is rectangular, then you can read it in with `read.table()` For example, if the file `atable.txt` contains

```
texas  florida
74      89
122    254
235    306
111    292
292    274
111    233
211    294
133    204
156    204
79      90
```

Then the command

```
> read.table(file="atable.txt",header=T)
```

will read it in with the header. (By default, `read.table` assumes no headers.

4 Applying functions in R

Using R is like MATLAB: you have a bunch of functions which act on "vectors". Here the vectors are datasets and may be rectangular in shape like a matrix, or collections of variables of different lengths.

4.1 Basic usage

The basic syntax is of the form

```
functionname(variablename)
```

For example, if the variable whales is defined the functions `mean()`, `median()` and `var()` work as expected

```
> whales
 [1]  74 122 235 111 292 111 211 133 156  79  90  91 110
> mean(whales)
[1] 139.6154
> median(whales)
[1] 111
> var(whales)
[1] 4446.423
```

Other useful functions are `length()` for the length, `names` for the names.

```
> length(whales)
[1] 13
> names(whales) = 1990:2002
> whales
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002
  74  122  235  111  292  111  211  133  156   79   90   91  110
> names(whales)                 # named them above
 [1] "1990" "1991" "1992" "1993" "1994" "1995" "1996" "1997"
 [9] "1998" "1999" "2000" "2001" "2002"
```

4.2 Extra arguments

Sometimes extra arguments are needed. FOr example, to generate 100 normal random numbers with mean 0 and variance 1 we can use `rnorm()` as

```
> rnorm(100)                    # not shown
```

To have these have mean 5 and variance 56 we can specify these as extra arguments as in

```
> rnorm(100,5,56)               # not shown
```

4.2.1 by name or position

The extra arguments in the command `rnorm(100,5,56)` are specified by position. YOu can also specify by name which allows you to bypass values. For example

```
> rnorm(5,1,2)                  # mean=1. sd=2
[1] 6.4713184 0.2412013 0.7046756 2.6932730 2.3173852
> rnorm(5,mean=1,sd=2)          # same only explicit
[1]  0.3892987  0.9283646  1.0183805  1.1978390 -2.2888802
> rnorm(5,sd=2)                 # used default of mean=0
[1] -0.3081788 -1.5453654  0.7539096  0.1511485 -2.5806210
```

5 Some statistical functions

R, like MATLAB is vectorized which allows to find many statistics quite easily. For example, the mean and standard deviation of the whales data is given by

```
> n = length(whales)
> xbar = sum(whales)/n          # the mean
> xbar
[1] 139.6154
> s2 = 1/(n-1)*sum( (whales -xbar)^2 ) # the sample variance
> sqrt(s2)
[1] 66.6815
```

One reason to use R over MATLAB is the number of built-in statistical functions and models. These are also available in MATLAB but in R they are more at the front in R.

5.1 `table()`,`summary()`

We can summarize data with the `summary` command

```
> summary(whales)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   74.0    91.0   111.0   139.6   156.0   292.0
```

For categorical data, the `table` command can be used. For example, suppose we first "cut" the data into 3 bins, then we can tabulate

```
> whales.cat  = cut(whales,c(0,100,200,300))
> whales.cat
 [1] (0,100]   (100,200] (200,300] (100,200] (200,300] (100,200]
 [7] (200,300] (100,200] (100,200] (0,100]   (0,100]   (0,100]
[13] (100,200]
Levels: (0,100] (100,200] (200,300]
> table(whales.cat)
whales.cat
  (0,100] (100,200] (200,300]
        4         6         3
```

Table will also do *n*-way contingency tables.

5.2 `mean()`,`median()`,`quantile()`,`sd()`

The standard mathematical functions are already built in and named pretty much what you might expect. There are a few extra arguments for some, for example the `mean()` function takes an option of `trim=` which will trim a certain percent of the data before finding the mean. For example

```
> mean(whales)                  # the familiar mean
[1] 139.6154
> mean(whales,trim = 0.25)      # the ''mid-mean''
```

```
[1] 119.1429
> mean(whales,trim = 0.5)       # only the middle (median) left
1996
 111
> median(whales)
1996
 111
> quantile(whales)              # gives quartiles as default
  0%  25%  50%  75% 100%
  74   91  111  156  292
> quantile(whales,0:5/5)        # the quintiles
   0%   20%   40%   60%   80%  100%
 74.0  90.4 110.8 124.2 189.0 292.0
> quantile(whales,0.95)         # a specific quantile
  95%
257.8
> var(whales)                   # variance
[1] 4446.423
> sd(whales)                    # also sqrt(var(whales))
[1] 66.6815
```

5.3 stem(),hist(),density(),boxplot()

The familiar graphical summaries are available. In addition, density estimates for numerical data are found with the `density()` function. These densities may be added to a histogram as illustrated.

```
> stem(whales)                  # stem an dleaf

  The decimal point is 2 digit(s) to the right of the |

  0 | 7899
  1 | 11123
  1 | 6
  2 | 14
  2 | 9

> hist(whales)                  # basic histogram
> hist(whales,prob=T)           # total area=1
> lines(density(whales))        # add density with lines()
> boxplot(whales)               # the boxplot
```

5.4 plot(),curve()

Mathematical plots can be made as in MATLAB: define values for `x`, then figure out `y` and plot with `plot(x,y)`. By default, a scatterplot is made. Use the argument `type="l"` to plot a line. For functions, the `curve()` command can plot functions from *a* to *b*. For example, all of these plot the sine curve from 0 to 2*pi:

```
> x = seq(0,2*pi,length=100)
> plot(x,sin(x))
```

```
> plot(x,sin(x),type="l")
> curve(sin(x),0,2*pi)
```

To add to the current plot use `points` with the same syntax. To add lines (same as `points(...,type="l")`) use `lines`. The command `abline()` will draw straight line $y = a + bx$ on the current plot window.

```
> points(x,cos(x),type="l",col="red",lwd=5) # draw derivative
> x0 = pi/4; fp = cos(x0)
> abline(sin(x0) - fp*x0,fp)    # add tangent line at pi/4
```

Contour plots are available with the `contour()` function. This example may be cut and pasted in from the help page `?contour`.

```
>      x <- y <- seq(-4*pi, 4*pi, len = 27)
>      r <- sqrt(outer(x^2, y^2, "+"))
>      opar <- par(mfrow = c(2, 2), mar = rep(0, 4))
>      for(f in pi^(0:3))
+        contour(cos(r^2)*exp(-r/f),
+                drawlabels = FALSE, axes = FALSE, frame = TRUE)
```

(The `<-` is an alternate assignment operator, `outer()` is an outer product, `r` has size 27 by 27/)

6 manipulating data in R

Just like in R, data may be manipulated by index or by logical vectors. This allows for very simple, yet useful views of a dataset. The square brackets, `[,]` are used for data extraction and assignment, in contrast to parentheses which are for functions.

6.1 vectors

The `whales` variable is an example of a vector. It has length and value found by

```
> whales
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002
  74  122  235  111  292  111  211  133  156   79   90   91  110
```

We can access or set the data in `whales` by slicing by name, index or with a logical vector.

For example,

```
> whales['1999']                # by name. NOte quotes
1999
  79
> whales[10]                    # by index (10th entry)
1999
  79
> whales[1:5]                   # a slice of 5 at a time
1990 1991 1992 1993 1994
  74  122  235  111  292
> whales[c(1,3,5,7,9,11)]       # odd indices
1990 1992 1994 1996 1998 2000
  74  235  292  211  156   90
```

Logical vectors make this extraction very useful. When you ask a logical expression of your variable using <, <=, >, >= or == (double equals!) the answer is a comparison element by element. For example

```
> whales > 100 & whales < 200
 1990  1991  1992  1993  1994  1995  1996  1997  1998  1999
FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
 2000  2001  2002
FALSE FALSE  TRUE
> whales == 111
 1990  1991  1992  1993  1994  1995  1996  1997  1998  1999
FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
 2000  2001  2002
FALSE FALSE FALSE
```

Notice the second line used & for logical and. THe corresponding or is found with |. Use ! for negation.

The sum function and others coerce these values to 0 for FALSE and 1 for TRUE. This makes it easy to find proportions or counts

```
> sum(whales > 150)
[1] 4
> sum(whales > 150)/length(whales)
[1] 0.3076923
```

More importantly, when you use these as indices, only the index corresponding to true are returned:

```
> whales[whales > 150]          # which value are bigger than 150
1992 1994 1996 1998
 235  292  211  156
> mean(whales[whales < 200])    # mean for smaller years
[1] 107.7
```

6.2 data frames and lists

When we have more than one variable from a dataset, we can store it in a rectangular way like a spreadsheet. Such data stores are called data frames in R. They can be accessed by row and column index just as a vector. For example, the datasets cats can be loaded and viewed as follows

```
> library(MASS)
> data(cats)
> cats
    Sex Bwt  Hwt
1     F 2.0  7.0
2     F 2.0  7.4
3     F 2.0  9.5
4     F 2.1  7.2
5     F 2.1  7.3
...
```

This data frame contains gender, birthweight and heart rate for several cats. (See ?cats for details). The data may be accessed as follows

```
> cats[1,2]                     # row 1 column 2
> cats[1,]                      # first row (blank column)
> cats[1:5,]                    # rows 1:5
> cats[1:5,2:3]                 # only cols 2,3 or rows 1:5
> cats[,3]                      # just 3rd column
```

The individual variables are named. They can be accessed by name as with

```
cats[,'Hwt']                    # case sensitive
```

A shortcut for this is using R's dollar-sign notation

```
> cats$Hwt
```

6.2.1 attaching a data frame or list

If you just want to play around with the data, and not change values. You can attach the names of the data frame so that instead of typing `cats$Hwt` You can simply type `Hwt`. This is done with the `attach()` function (use `detach()` to clean up.)

```
> Hwt
Error: Object "Hwt" not found
> attach(cats)
> Hwt                           # now its present
  [1]  7.0  7.4  9.5  7.2  7.3  7.6  8.1  8.2  8.3  8.5  8.7  9.8
 [13]  7.1  8.7  9.1  9.7 10.9 11.0  7.3  7.9  8.4  9.0  9.0  9.5
...
```

6.3 manipulating a data frame or list

Logical extraction works with data.frames too. To get just the Males of the cat data we could do

```
> cats[cats$Sex=="M",]
...
```

Other conveniences exists (`split`,`formula`) to be described at a later time.

7 Defining functions

You can define your own functions in R at the command line or in an external file (which must be sourced in). A function in R is defined by the following pieces

```
function ( arguments ) body
```

For example,

```
ourhist <- function(x) {
  hist(x,breaks="Scott",probability=TRUE)
  lines(density(x))
}
```

Will plot a histogram and add a density plot for you.

You can add extra arguments with defaults and named as in

```
ourhist <- function(x,breaks="Scott",col="purple") {
  hist(x,breaks=breaks,probability=TRUE,col=col)
  lines(density(x))
}
```

This can be called as

```
> ourhist(x)
> ourhist(x,"Sturges")          # use different bin rul
> ourhist(x,"Sturges","green")  # green before purple
> ourhist(x,"green")            # Oops
Error in match.arg(tolower(breaks), c("sturges", "fd", "freedman-diaconis",  :
        ARG should be one of sturges, fd, freedman-diaconis, scott
```

We see we can make changes to the defaults quite easily. The third line uses the Sturges rule and green as the color. (How R uses colors is described in the help page for `par()`.)

8 Linear models

One can make linear models (others are similarly done) quite easily with R. To look at the relationship of the Weight and heart Weight of cats, can be done as follows

```
> plot(Hwt ~ Bwt,data=cats)     # uses model formula
> tmp = lm(Hwt ~ Bwt,data=cats)
> tmp

Call:
lm(formula = Hwt ~ Bwt, data = cats)

Coefficients:
(Intercept)          Bwt
    -0.3567       4.0341

> abline(tmp)
```

The "tilde" notation is for modeling and is read "Hwt is modeled by Bwt" in this case, the notation says a linear model as in the familiar

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i. \tag{1}$$

Diagnostics are readily available. For example, the residuals are accessed by `resid(tmp)` as used in

```
> plot( resid(tmp))
```

9 Reading in external data sets

9.1 `read.csv()`

If you have spreadsheet data and want to read it into R then it may be easiest to export the data as a CSV file, and read it in with the R function `read.csv()`. (More sophisticated methods are available.) The syntax is similar to `read.table()`. You have many options such as the separator, whether headers are there or not etc.

9.2 `read.fwf()`

Many large datasets are available in fixed-width format. This too can be read in easily enough. Fixed width format needs to have the widths of the fields specified. The `meadsubset.txt` data has this format

```
FLOR  1990  B.  ACUTOROSTRATA
FLOR  1991  B.  ACUTOROSTRATA
FLOR  1993  B.  ACUTOROSTRATA
FLOR  1940  B.  ACUTOROSTRATA
FLOR  1979  B.  ACUTOROSTRATA
FLOR  1981  B.  ACUTOROSTRATA
FLOR  1952  B.  ACUTOROSTRATA
FLOR  1953  B.  ACUTOROSTRATA
FLOR  1998  B.  ACUTOROSTRATA
MAIN  1889  B.  ACUTOROSTRATA
MAIN  1991  B.  ACUTOROSTRATA
MAIN  1989  B.  ACUTOROSTRATA
MAIN  1991  B.  ACUTOROSTRATA
...
```

As it is easier to find the column numbers in my editor, I use the `diff()` function to find the differences between the column numbers. First, you need to download the file `meadsubset.txt` from the website `http://www.math.csi.cuny.edu/verzani/classes/804/computer` and store it somewhere on your computer (as a text file).

Then read it into `x` with

```
x = read.fwf(file=file.choose(),widths=diff(c(0,4,6,10,12,30)))
```

`file.choose()` will let you browse for the file.

Alternately, if you like typing you can specify the file directly as in

```
f = "http://www.math.csi.cuny.edu/verzani/classes/804/computer/meadsubset.txt"
x = read.fwf(file=url(f),widths=diff(c(0,4,6,10,12,30)))
```