Running R: simply click on the R icon.

Basic manipulations:

```
> 2+3
> x <- 2+3           [answer is stored in the vector x]
> x                  [displays the contents of x]
> x <- c(10.1, -3.2,4.5, 6.7, 8.9, -1.2)
                     [overwrites x as a vector containing 6 numbers]
> x[3]               [displays the third entry of x]
> x<0                [displays TRUE if x<0, FALSE otherwise]
> x[x<0]             [displays negative values of x]
> mean(x)            [calculates the mean of x]
> x/var(x)^.5        [divides each element of x by standard deviation]
> y <- 1/x           [stores the reciprocal of x in the vector y]
> z <- 2*x +y        [computes the vector z consisting of 2*x + y]
> z                  [displays z]
> 1:10               [the integers 1 to 10]
> sum(1:100)         [a quick way to sum the integers 1 to 100]
> prod(5:9)          [product of the integers 5 through 9]
> sum(1:100) ; prod(5:9)
                     [multiple commands separated by semicolons]

> x<-matrix(1:6,3,2); x
                     [a 3x2 matrix with elements 1, 2, 3, 4, 5, 6]
> x[2,2]             [the element in the 2nd row, 2nd column of x]
> x[3,]              [all elements in the 3rd row of x]
> x[,2]              [all elements in the 2nd column of x]
```

Loops:

```
> p<-0                                  [initializes p to 0]
> for (n in 1:10) p<-p+(1/2)^n          [compute 1/2+1/4+1/8+…+1/1024]
> p                                     [displays the result]
```

Plots:

```
> plot(sin(seq(0,pi,.1)))
        [plots sin(x) for x from 0 to pi in increments of .1]
> plot(seq(0,2*pi,.1), sin(seq(0,2*pi,.1)))
        [plots (x, sin(x)) for x from 0 to pi]
> plot(seq(0,2*pi,.01), sin(seq(0,2*pi,.01)),type="l")
        [plot with just lines]
> lines(seq(0,2*pi,.01), cos(seq(0,2*pi,.01)))
        [appends plot with another]
```

Random numbers:

```
> set.seed(321)           [set the random number seed to 321]
> runif(10)               [10 uniform(0,1) random numbers]
> sample(1:6,10,replace=T) [10 discrete uniform{1,…,6} random numbers]
```

Functions/Small programs:

```
> myfunction<-function(maxn=10) {
>   p<-0
>  for (n in 1:maxn) {
>     p<-p+(1/2)^n
>  }
>        return(p)
> }
```

```
> myfunction                [lists the program]
> myfunction()              [runs the program with default maxn, 10]
> myfunction(20)            [runs the program with maxn=20]
```

Something to Try:

Simulate tosses of a fair coin.

a). Set the random number seed to 321 using set.seed(321). Use runif to simulate 20 tosses of a fair coin. Use the "sum" command to count the number of heads.

b). Reset the random number seed to 321. Use sample to simulate 20 tosses of a fair coin. Compare to (a).

c). Repeat (a) but with random number seed 169 and 2000 tosses of a fair coin.

Basic matrix operations in R:

A *matrix* is a rectangular array of numbers. Its *dimension* is described as "(number of horizontal rows) by (number of vertical columns)". Elements of a matrix are specified in row/column coordinates. For example, in a 5x4 matrix, the element *x[5,4]* is in the 5th row, 4th column, which is the bottom right-hand corner; *x[5,1]* is in the 5th row, 1st column, which is the bottom left-hand corner; and so on. A matrix is *square* if the number of rows and number of columns are equal. A 1x1 matrix is a single number, usually called a *scalar*. A 1x*n* or *n*x1 matrix is called a *vector*.

A *zero matrix* is a matrix in which each element is zero. Two matrices are equal if they have the same dimensions and the corresponding elements in each matrix are equal.

R has a number of facilities for handling matrices. Some knowledge of manipulating matrices will be necessary.

Task 1. Create the following matrices.

a). A 5x5 zero matrix: `Z<-matrix(0,5,5)`

b). Some 2x2 matrices, using various input techniques:

```
A<-cbind(c(0,0),c(1,2))
B<-rbind(c(1,1),c(3,4))
D<-matrix(c(2,3,5,4),2,2)
E<-matrix(0,2,2)
E[1,1]<-3
E[1,2]<-7
```

c). Miscellaneous matrices:

```
one<-rbind(c(1,1))
two<-rbind(1:4,2:5)
identity<-diag(c(1,1))
```

d). Check the dimensions of a matrix: `dim(A)`

Task 2. Matrix arithmetic.

a). Scalar multiplication. What is `3*B`?

b). Matrix addition. What is `Z+A`? What is `B+A`? What is `A+B`? What is `A-B`?

c). Matrix multiplication (denoted `%*%` in R). Compute the following:

```
Z%*%A
A%*%B
one%*%B
identity%*%B
identity%*%D
identity%*%E
B%*%two
two%*%B
```

Task 3. Recall that for scalars *a* and *e*, *ae=0* implies that *a=0* or *e=0*. Also, the *cancellation law* says that if *a* is not equal to zero, then *ab=ad* implies that *b=d*. The *commutative law for multiplication* says that *ab=ba*. Consider the matrices you created in Task 1.

a). What is AE? How does this compare to the scalar result?

b). Does AB=AD? How does this compare to the scalar result?

c). Does AB=BA? How does this compare to the scalar result?

Task 4. Solving systems of linear equations. A system of linear equations can be rewritten in matrix form to facilitate its solution. For example, the system

x+y+2z=9
2x+4y-3z=1
3x+6y-5z=0

can be rewritten as

G (x,y,z)'=b

where the apostrophe ' is read *transpose.* The transpose converts the 1x3 vector into a 3x1 vector. The matrix *G* in the above expression is

```
G<-rbind(c(1,1,2),c(2,4,-3),c(3,6,-5))
```

and the vector *b* is

```
b<-cbind(c(9,1,0))
```

We can solve for *(x,y,z)'* in the above equation by computing the *inverse* of the matrix *G* and multiplying this inverse by *b. (*A matrix must be square in order to have this kind of inverse, but not all square matrices have inverses.) This is analogous to dividing in the scalar case. The inverse of the matrix *G* is computed in R as `solve(G).`

What is `solve(G)%*%G`? What is `G%*%solve(G)`? Find the solution to the above system of equations. What is `solve(A)` where *A* is defined in Task 1?

Task 5. It will be convenient if we have a function in R that will raise the power of a matrix. Copy this function and try the following commands.

```
mtxpow<-function(X,n) {
# raise the matrix X to power n
# assume n>=2
  temp<-X
  for (i in 1:(n-1)) {
    temp<-temp%*%X
  }
  return(temp)
}

mtxpow(A,2)
mtxpow(A,5)
```