

# An Introduction to R

## To get R:

1. Download R (it's free) from the website <http://cran.r-project.org> There are versions for Linux, Windows and Mac.
2. R will be installed in the public computing cluster in room 109 in the Informatics building.

## R can be used as a calculator:

Try typing the following expressions at the command line (followed by return): (> is the command prompt).

```
> 5+3
> 10*10
> log(9.4)
> exp(exp(exp(20))) # R is only human! (anything following '#' is a comment)
```

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)
> 1 == exp(0) # so does this
> 1 != exp(0) # this evaluates to F (false)
```

It is possible to have variables that hold values in your program. Most strings beginning with an alphabet character will be treated as variables. The assignment operator is `<-` (a “less than” followed by “minus”). Try typing the following lines in succession

```
> x <- 3
> y <- x*x+x
> y # print the value of y
```

## Vectors

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(-pi,pi,length=100) # y consists of 100 evenly spaced values from -pi to pi
> z <- c(1,4,8,20) # z is the vector (1,4,8,20)
> a <- x+y # vectors of same length can be added, multiplied, etc.
> b <- 4*x # this is interpreted correctly too
```

## Random Number Generation

R has lots of built-in functions for doing things with random numbers. For instance

```
> x <- runif(100) # creates a vector of 100 (uniformly distributed) random numbers between 0 and 1.
> punif(v) # is the probability that a Unif(0,1) rand number is less than v
> qunif(u) # gives the uth quantile of a Unif(0,1). More on this later.
```

There are similar functions for a variety of other distributions including the normal(0,1) (`rnorm`,`pnorm`,`qnorm`) Cauchy (`rcauchy`, `pcauchy`, `qcauchy`), Exponential, Binomial, Poisson, and others.

## Subsets

```
> x <- runif(100) # creates a vector of 100 Unif(0,1) random numbers
> x[1] # the first element of x
> x[c(1,3,5)] # a vector containing 1st, 3rd and 5th elements of x
> y <- x > .5 # a 100-long vector of Boolean values y[i] is T iff x[i] > .5
> z <- x[x>.5] # the ‘x’s’ that are greater than 5
```

**Plotting** Try the following

```
> x <- seq(0,1,length=100)
> y <- x^2                # y = x squared
> plot(x,y)              # plot with (x[1],y[1]) \ldots, (x[100],y[100])
> plot(y,x)
> plot(y)                # same as plot(1:length(y),y)
```

**Source Files** You will want to write simple programs in R and this always 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. Suppose you create the following file named “myprog.R” in your editor:

```
len <- 100
x <- runif(len,-.5,.4)
y <- cumsum(x) # y[1] = x[1], y[2] = x[1]+x[2], etc.
plot(exp(y))
title("my stock price")
print("history is: ")
print(y)
```

This technique allows you to write a program in the usual incremental way. 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") # write plot in the postscript file ‘myplot.ps’
> sink("myout.txt")       # write 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.
```

**A fun example** Suppose two decks of cards are shuffled. The first deck is arranged in a line, face up, and the second deck is arranged in a similar line right below. Count the number of places where the two decks have the same card. What is the probability that there are no matches between the two decks? (This is a hard calculation to do). One way to estimate this probability would be to perform the experiment many times and observe the proportion of times the event occurs.

```
> trials <- 1000          # number of trials
> zeros <- 0              # counts the number of times the ‘no match’ event occurs
> for (i in 1:trials) {   # all statements in the ‘for’ loop are executed ---
                          # once for each value of i in 1:trials
  > x <- 1:52;
  > deck1 <- sample(x,52,replace=F) # a random permutation of the ‘cards’
  > deck2 <- sample(x,52,replace=F) # another random permutation
  > matches <- sum(deck1==deck2)    # number of matches
  > if (matches == 0) zeros <- zeros+1 # if (matches == 0) we had a no matches so count it
  > }
> print("my estimate is:")
> print(zeros/trials);    # the proportion of times the ‘no match’ event occurred
```

**Quitting and help**

```
> help("rnorm") # gives information about the function rnorm. Of course this works
>               # for other functions too.
> q() # quitting the program. Hope you had fun.
```