Introduction to R

We will use R extensively in this course. R is a high level language especially designed for statistical calculations. R is free. You can get it at:

http://www.cran.r-project.org/

There are versions for Unix, Linux, Windows and Mac. There is a similar program called Splus. The commands are virtually identical. Splus has more stuff in it but R is free and it is faster. If you want to use Splus, it is available on Andrew (just type Splus) or you can purchase a copy from Insightful at http://www.splus.mathsoft.com/.

On my website I will often post R programs and examples.

Getting started

In Unix or Linux, you start R by typing: R. In windows, click on the R icon. You can now use R interactively. Just start typing commands. You can also use R in Batch mode. To do this, store your R commands in a file, say, file.r. In R type: source(“file.r”) which will execute the commands in file.r. In Unix, you can also do the following:

    R BATCH file.r file.out &

which will execute the commands and store them in file.out.

IMPORTANT! Use the command: q() to quit from R.

Some Basics of R

Here is a simple R session. You should try these commands out. Also, feel free to experiment a bit. Note: the # symbol means “comment.” R ignores any command after #. I have added lots of comments below to explain what is going on. You do not need to type the comments.

    x <- 5           ### assign x the value 5
    x
    print(x)    ### another way to print x
    x_5
    ### you can also use _ to make assigments but <- is better
    x
    y <- "Hello there"
    y
    y <- sqrt(10)
    z <- x+y
    z
    q()             ### Use this to quit

 Scalars are treated by S-plus as vectors of length 1. That is why they print with a leading “[1]” indicating that we are at the first element of a vector.
- Vectors can be created using the `c()` command. `c()` stands for concatenate. Square brackets are used to get subsets of a vector. The colon is used for sequences. Start up R again then do this:

```r
x <- 1:5
print(x)
x[1] <- 17
print(x)
x[1] <- 1
x[3:5] <- 0
print(x)
w <- x[-3]
print(w)
y <- c(1, 5, 2, 4, 7)
y
y[2]
y[-3]
y[c(1, 4, 5)]
i <- (1:3)
z <- c(9, 10, 11)
y[i] <- z
print(y)

y <- y^2
print(y)
y <- 1:10
y <- log(y)
y
y <- exp(y)
y
x <- c(5, 4, 3, 2, 1, 5, 4, 3, 2, 1)
z <- x + y
z
```

```r
### R carries out operations on vectors, element by element

```r
x <- 1:10
y <- c(5, 4, 3, 2, 1, 5, 4, 3, 2, 1)
x <- 2  # this is a logical vector
z <- (x == 2)
print(z)
z <- (x>5); print(z)  # You can out two commands on a line if you use a semi-colon
x[x<5] <- y[x<5]  # do you see what this is doing?
print(x)
```

2
- Two expressions can be written on the same line if separated by a semicolon. One expression can be written over several lines as long as a valid expression does not end a line.

- To create a “matrix”, use the matrix() function as follows:

```
junk <- c(1,2,3,4,5, 0.5, 2, 6, 0, 1, 1, 0)
m <- matrix(junk, ncol=3)
print(m)
m <- matrix(junk, ncol=3, byrow=T)
print(m)                     # see the difference?
dim(m)
y <- m[,1]                   # y is column 1 of m
y
x <- m[2,]                   # x is row 2 of m
x
z <- m[1,2]
print(z)
zz <- t(z)                   # take the transpose
zz
new <- matrix(1:9, 3, 3)
print(new)
hello <- z + new
print(hello)
```

The square brackets are used to refer to the rows and columns of a matrix, similar to the way they are used for vectors.

```
> m[1,3]
[1] 1
> subm <- m[2:3, 2:4]
> m[1,]
[1] 1 2 3 4
> m[2,3] <- 7
> m[,c(2,3)]
[1] 2 3 4
```

```
x <- runif(100,0,1)         # generate 100 numbers randomly between 0 and 1
mean(x)
y <- mean(x)
print(y)
help(mean)
min(x)
```
max(x)
summary(x)
help(summary)

- Lists are used to combine data of various types.

who <- list(name="Joe", age=45, married=T)
print(who)
print(who$name)
print(who[[1]])
print(who$age)
print(who[[2]])
print(who$married)
print(who[[3]])
names(who)
who$name <- c("Joe","Steve","Mary")
who$age <- c(45,23)
who$married <- c(T,F,T)
who

- A for loop is a statement that is used to repeat commands.

for(i in 1:10){
    print(i+1)
}

x <- 101:200
y <- 1:100
z <- rep(0,100)  # rep means repeat
help(rep)
for(i in 1:100){
    z[i] <- x[i] + y[i]
}

w <- x + y
print(w-z)

### As this example shows, we can often avoid using loops since
### R works directly with vectors.
### Loops can be slow so avoid them if possible.
for(i in 1:10){
    for(j in 1:5){
        print(i+j)
    }
}

- **Reading in commands** To read in commands or functions from a file rather than typing them in, use `source()`. Put some R commands into a file called `hello`. Try “`source('hello')`”

  **Functions** You can create your own functions in R. Here is an example.

  ```r
  my.fun <- function(x,y) {
      ###### This function takes x and y as input.
      ###### It returns the mean of x minus the mean of y
      a <- mean(x)-mean(y)
      return(a)
  }
  
  x <- runif(50,0,1)
  y <- runif(50,0,3)
  output <- my.fun(x,y)
  print(output)
  
  You can return more than one thing in a function. If you do, you should return a list.

  ```r
  my.fun <- function(x,y) {
      mx <- mean(x)
      my <- mean(y)
      d <- mx-my
      out <- list(meanX=mx,meanY=my,difference=d)
      return(out)
  }
  ```

  x <- runif(50,0,1)
  y <- runif(50,0,3)
  output <- my.fun(x,y)
  print(output)
  names(output)
  output$difference
  output[[3]]
  ```

- Here are some more R examples
### if statements

for(i in 1:10){
    if( i == 3) print(i)
}
for(i in 1:10){
    if( i != 3) print(i)        ### != means 'not equal to'
}
for(i in 1:10){
    if( i < 3) print(i)
}
for(i in 1:10){
    if( i <= 3) print(i)
}
for(i in 1:10){
    if( i >= 3) print(i)
}

### Plots

x <- 1:10
y <- 1 + x + rnorm(10,0,1)    ### rnorm(10,0,1) means 10 random Normals,
                                ### mean 0, standard deviation 1
plot(x,y)
plot(x,y,type="h")
plot(x,y,type="l")
plot(x,y,type="l",lwd=3)
plot(x,y,type="l",lwd=3,col=6)
plot(x,y,type="l",lwd=3,col=6,xlab="x",ylab="y")

par(mfrow=c(3,2))        ### put 6 plots per page, in a 3 by 2 configuration
for(i in 1:6){
    plot(x,y+i,type="l",lwd=3,col=6,xlab="x",ylab="y")
}

postscript("plot.ps")    ### put the plots into a postscript file
## you have to do this if you use BATCH
plot(x,y,type="l",lwd=3,col=6,xlab="x",ylab="y")
dev.off()                  ### Now you can print the file or view it with
## a ghostview previewer

par(mfrow=c(1,1))        ### return to 1 plot per page

6
x <- rnorm(100,0,1)  ### 100 random normals, mean 0, st.dev .1
y <- rpois(500,4)    ### 500 random Poisson(4)
hist(y)              ### histogram
hist(y,nclass=50)

pnorm(2,0,1)         ### P(Z < 2) where Z ~ N(0,1)
qnorm(.3,0,1)        ### find x such that P(Z < x)=.3 where Z ~ N(0,1)

x <- seq(-3,3,length=1000)  ### make a sequence of numbers
f <- dnorm(x,0,1)     ### normal density
plot(x,f,type="l",lwd=3,col=4)