STATS 406 F15: Lab 02

1 Text Data I/O Manipulation

When reading in a rectangular data set from a file you can use the read.table command. The first thing is to make sure the file is in your working directory. You can get your working directory by typing **getwd()**. If the data set is on the desktop, for example, you can appropriately change your directory by typing

```
setwd("users/{your uniqname}/Desktop")
```

Download the high way data set ("flow-occ-table.txt") from C-tools. 1. Read in the data set. 2. Change the column names so that they read F1, O1, F2, O2, F3, O3 instead of Flow1, Occ1, Flow2, Occ2, Flow3, Occ3. 3. Create a new data frame that only has two columns:

- column 1 is the maximum of Flow1, Flow2, Flow3
- column 2 is the value of Occ corresponding to which Flow was the maximum

4. Write the resulting data frame to a tab delimited file call "flow-occ-table-clean.txt".

Solution

```
## Read data
Traffic <- read.table('flow-occ-table.txt', header=T, sep=",")
## Check the column names of the data
colnames(Traffic)
## Set new column names
colnames(Traffic) = c("O1", "F1", "O2", "F2", "O3", "F3")
## Check it again
colnames(Traffic)

## Initialize the new matrix to record the final result
newTraffic <- matrix(0, nrow(Traffic), 2)
## Read the rows one by one
for(i in 1:nrow(Traffic))
{
   ## Extract the i-th row
   traffic.i <- Traffic[i, ]
   ## Find the maximal value of the three
   maxflow <- max( traffic.i[c(2,4,6)] )
   ## Find which is the maximal value of the three
   idx_maxflow <- which.max(traffic.i[c(2,4,6)])
   ## Extract the Occ columns 1, 3, 5
   traffic.i.occ <- traffic.i[c(1,3,5)]
   ## Extract the Occ value correponding to the maximal flow
   maxocc <- as.numeric(traffic.i.occ[idx_maxflow])
   ## Record the resutl
   newTraffic[i,] <- c(maxflow, maxocc)
}
## Convert matrix to data frame
newTraffic <- as.data.frame(newTraffic) ## Difference between matrix and data
##frame? All entries in matrices are of the same type. Data frames let you
##refer to column by names.
## Set column names
colnames(newTraffic) = c("maxflow", "maxocc")
## Write the data frame into a text file
write.table(newTraffic, file="flow-occ-table_clean.txt", sep="\t", col.names=T)
```

2 An Example

A gambler bets on a sequence of fair coin flips. For each coin flip, he wins $1 if heads appears and loses $1 if tails appears. He stops when he has won 2 consecutive flips. Use R to simulate his average total winnings (assume that he executed this strategy many times and take the average of his total winnings). Note that negative winnings are losses.

```
#Initialize the vector to store winnings
winnings <- NULL
# Game loop
for (i in 1:niter)
{
  #initialize first winning, last flip result, and first flip result
  lastflip <- 0
  x <- (runif(1)<=.5)
  winning <- (x==1)-(x==0)
  while((lastflip!=1)|(x!=1))
  {
     # last flip becomes this flip
     lastflip <- x
     # a new flip
     x <- (runif(1)<.5)
```

```
    winning <- winning+(x==1)-(x==0)
  }
  winnings <- c(winnings,winning)
}

mean(winnings)
```

3 Plotting

To demonstrate some basic plotting commands in R we will complete the following exercises using the traffic data set. Since this data set is so large (over 1700 observations) we will only look at the first 30 observations for the purposes of demonstrating plotting commands:

```
Y <- read.table(flow-occ-table.txt, sep=",", header=T)
Y <- Y[1:30,]
```

To demonstrate various plotting tools we will complete each of the following tasks: 1. Make scatterplots of **Occ1**, **Occ2**, **Occ3** all on the same axes. To disinguish between them make sure they are different colors. 2. The default plotting characters are hollow circles. For **Occ1** use + as the plotting character; for **Occ2** use squares, and for **Occ3** use filled in circles. 3. Make a legend to indicate which color/plotting character is which variable.

A few simple examples first.

```
# only a single input
v <- seq(0, 1, length=10)
plot(v)
# two inputs
v <- seq(0, 1, length=10)
u <- seq(3, 4, length=10)
plot(u,v)
```

To begin the example, we need to choose one variable to plot first, then add the points for the other variables using the points command. The colors are controlled using the **col** tag:

```
plot(Y$Occ1, col=2, ylim=c(0, .04))
points(Y$Occ2, col=3)
points(Y$Occ3, col=4)
```

If you do not include the **ylim** tag in your initial call to plot, then you will find that the automatically selected y-axis limits will result in some of the points being cut off this happens frequently so you will usually have to tweak **ylim**. To change the plotting character you use the **pch** tag in your calls to to plot and points. There are many different characters you can use with each number corresponding to a different one for example **pch=2** will plot little hollow triangles instead of hollow circles. For this example you would use:

```
# type this before you make the plot to save it as a pdf
pdf("plot.pdf")
plot(Y$Occ1, col=2, ylim=c(0, .04))
points(Y$Occ2, col=3)
points(Y$Occ3, col=4)
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3) ## pch=3 makes + signs
points(Y$Occ2, col=3, pch=0) ## pch=0 makes squares
points(Y$Occ3, col=4, pch=16) ## pch=16 makes filled in circles
```

To attach the points in each variable with a line you can use the lines command. Note that the lines and points commands can only be used when a plot is already open, otherwise an error will appear. In our example connecting the lines is done by

```
lines(Y$Occ1, col=2)
lines(Y$Occ2, col=3)
lines(Y$Occ3, col=4)
```

There are many options available with the legend command but we will only cover the most basic. Basically, **legend(x, y, names, pch=plotchars, col=colors, lty=1)** will create a legend beginning located on the graph at the point (x,y), with a line in the legend for each entry of names, labeling the points by their respectively plotting characters and colors, stored in plotchars and colors. The **lty=1** tag just tells R to put a line through the points in the legend.

```
# The points correspond to "Occ1", "Occ2", and "Occ3"
# The colors we used were 2,3,4
# The plotting characters where 3, 0, 16
legend(25, .035, c("Occ1", "Occ2", "Occ3"), pch=c(3,0,16), col=c(2:4), lty=1)
# type this after youve made the plot
dev.off()
```

4 Functions

In R, functions are a symbolic representation of an operation to be carried out on variables known as inputs. Functions are useful when you plan to apply a certain operation to a range of inputs which are cumbersome to be declared beforehand. The syntax for declaring a function can be best understood with an example:

Without using dnorm(), write your own function to compute the normal density function. The density function of normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$$

Plot the densities with respect to a grid of points between -3 and 3, compare the result with R internal function dnorm().

Solution for Density Function

```
## Define the normal density function
normal_density_function <- function(x, mean, sd)
{
    d <- 1 / sqrt(2*pi*sd^2) * exp(-(x - mean)^2 / (2*sd^2))
    return(d)
}

## Generate a grid with 30 points between -3 and 3
x <- seq(-3, 3, length=30)
## Compute the corresponding densities
y1 <- normal_density_function(x=x, mean=1, sd=2)
## Compute the densities from R internal function dnorm()
y2 <- dnorm(x=x, mean=1, sd=2)
## Plot the result of function normal_density_function() with points
plot(x, y1, main='Density of normal distribution', xlab='x', ylab='Density', type='p')
## Plot the result of function normal_density_function() with curve
lines(x, y2, lty=1)
```

The return statement within a function is used when you want some value returned by the function. Sometimes you just want a function to do something (say print/plot something), but not return a value. There is no need to use the return() command in those situations. Also, the variables used in a function are local and not accessible unless you are returning them as output.

5 The apply command

Frequently you will want to call the same function across a grid of values for the input. One way to do this would be to use a for loop, but in many cases the apply command offers a more computationally efficient alternative. The basic syntax is apply(A, m, f) where A is a matrix or data frame containing the grid of values for your inputs, f is the function to be applied, and m is 1 if you are applying f to the the rows and 2 if columns.

In the following example we have a data matrix that is 200 rows each containing 10 i.i.d. Binomial(50, .6) random variables:

```
A = matrix( rbinom(2000, 50, .6), 200, 10)
```

The following code obtains the third smallest entry in each of the 200 rows both using a loop and using apply:

```
# f to calculate the 3rd smallest entry of a vector x
f <- function(x) sort(x)[3]
# Storage for the function output
```

```
OUT <- rep(0, 200)
# Using a loop
for(i in 1:200)
{
OUT[i] <- f( A[i,] )
}

# Using apply
OUT <- apply(A, 1, f)
```

How to model an SIRS model?

```
# to estimate the average proportion
p_recover = 0.5;
p_loose_immunity = 0.2;
P_ever = 0
## Initialize the population.
Q = numeric(1000)
Q[1:10] = 1
Q_ever = Q
p_transmit = 0.3

## Figure out which infected people recover and become immune.
    infected = which(Q == 1)
    for (j in infected) {
      if (runif(1) < p_recover) { Q[j] = 2 }
    }
## Figure out which immune people lose their immunity.
    immune = which(Q == 2)
    for (j in immune) {
      if (runif(1) < p_loose_immunity) { Q[j] = 0 }
    }
```