0 Law of large numbers

Suppose you have i.i.d. data $X_1, X_2, ..., X_n$ each with CDF F. Recall the CDF is defined as $F(x) = P(X \leq x)$. Define the *empirical CDF* based on a sample of size n to be

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{I}(X_i \leq x)$$

which is just the sample proportion of values in the data set that are less than or equal to x. One implication of the strong long of large numbers is that the bigger n is, the better $F_n(x)$ performs as an estimator of $F(x)$. In fact, as $n \to \infty$, $F_n(x)$ "converges" to $F(x)$ for any value of x. This means you can use the sample proportions to estimate the true probabilities, and this estimate will be good when n is big. Here are a number of examples where we estimate and calculate directory $P(0 \leq X \leq 1)$:

```
### X ~ N(0,1) (as in the homework)
# exact probability
pnorm(1) - pnorm(0)
# simulated
X <- rnorm(10000)
mean(X <= 1) - mean(X <= 0)

### X ~ exponential(1)
# exact probability
pexp(1) - pexp(0)
# simulated
X <- rexp(10000)
mean(X <= 1) - mean(X <= 0)

### X ~ Uniform(0,3)
#exact
punif(1,a=0,b=3) - punif(0,a=0,b=3)
# simulated
X <- runif(10000,a=0,b=3)
mean(X <= 1) - mean(X <= 0)

### X ~ chisquare(df=2)
# exact
pchisq(1,df=2) - pchisq(0,df=2)
# simulated
X <- rchisq(10000,df=2)
mean(X <= 1) - mean(X <= 0)
```

The convergence of the empirical CDF also implies convergence of the corresponding densities and all moments $(E(X), E(X^2), E(X^3), ...)$, which is why estimating parameters looking at the sample quantities is a reasonable thing to do.

1 Reading in data from a file

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/jasoneg/Desktop")
```

In the first example we are reading in a file called `testdata.txt`, which has 10 rows, labeled by the numbers 1 through 10, and 4 columns labeled by `"y", "x1", "x2", "status"`. We can read in this data and store it into the variable `Y` by typing

```
Y <- read.table('testdata.txt', header=T, row.names=1)
```

The tag `header=T` tells R that the first row should be interpreted as column names and not the first row of the data set. The `row.names=1` similarly tells R the first column is not data, but are labels for the rows. `Y` is a data frame which is similar to a matrix and can be indexed in the same way, but can also be indexed by use of the `$` operator.

```
Y$status ## equivalent to Y[,4]
 [1] treatment treatment treatment treatment treatment control   control
 [8] control   control   control
Levels: control treatment
```

In the next example, we use the highway dataset `flow-occ-table.txt`, available from the ctools site. By visually inspecting the data set there are no row names and the values are separated by commas, so this has to be encorporated into the `read.table` call.

```
A <- read.table('flow-occ-table.txt', header=T, sep=",")
```

In this data set the commas are referred to as *delimiters*. Things such as tabs - `"\t"`, white space - `""`, semi colons - `";"`, are common delimiters in data sets that can be accounted for with the `sep` argument.

When a data set is not rectangular– that is, there are not the same number of columns in each row, reading in data can require a bit of programming. Usually this will entail use of the `scan` command coupled with some manual data manipulation. For this example you will need the data set `testdata2.txt` available at my website. Each row are observations on one unit and are observed until one exceeds 3 in absolute value. The following code will read in this data set

```
# read in the data as one long vector
V <- scan("testdata2.txt", sep=",")

# storage for the data and the number of observations per unit
X <- list()

# the number of unique records in the data set
N <- sum( abs(V) > 3 )

# storage for the maximum record length
```

```
M <- 0

for(n in 1:N)
{

   # find the next value bigger than 3
   w <- min( which( abs(V) > 3) )

   # extract subject n data and save it into X
   X[[n]] <- V[1:w]

   # throw out the data we just recorded
   V <- V[-c(1:w)]

   # see if this is the longest record yet
   if( w > M )
   {

      M = w

   }

}

# Record the data set into a matrix with
# NAs after the record has ended for each

D <- matrix(NA, N, M)

for(i in 1:N) D[i,1:length(X[[i]])] <- X[[i]]
```

Each of the above examples involved reading in data from an existing file. By using the `load` command you can load a previously saved R environment, so that all variables used in that session will be retained. (An example of this will be given in the next section)

2 Saving data

To save a data frame directly to a file for potential use outside of R, the `write.table` command can be used. The first example data set we used, `testdata.txt` was generated by the following code:

```
# generate some random numbers
v <- rnorm(30)

# convert the 30 length vector to a 10x3 matrix
X <- matrix(v, 10, 3)
```

```
# convert to data frame
X <- as.data.frame(X)

# a categorical variable in the data frame
X <- cbind(X, c( rep("treatment", 5), rep("control", 5)) )

# names for the variables
colnames(X) = c("y", "x1", "x2", "status")

# write the data to the file testdata.txt
write.table(X, file="testdata.txt", row.names=T, col.names=T)
```

The command `matrix(v, n, m)` converts `v` to a an `n` by `m` matrix. The first column of the resulting matrix is the first `n` elements of `v`. The second column is the second `n` elements of `v`, and so on. The length of `v` had better be `n*m`. Additionally if you wanted to make `testdata.txt` comma or tab delimited you would add a `sep=","` or `sep="\t"` tag to the `write.table` call.

To save the R environment for later use you use the `save` command. For example, suppose we were using R and made the following declarations

```
x1 <- matrix(0, 20, 20)
x2 <- c("Treatment", "Control")
x3 <- c(1:10)
x4 <- matrix(2, 13, 4)
```

If you were to close R without saving then all of these variables would be gone. However, if you type

```
save("x1","x2","x3","x4",file="variables.Rdata")
```

then the variables specified in the `save` statement will be recalled in R when you type

```
load("variables.Rdata")
```

3 Data manipulation example

Using the highway data set again from ctools to perform the following tasks

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 the a tab delimited file called `flow-occ-table_clean.txt`.

Recalling the previous section we can read in the data by typing

```
Traffic <- read.table('flow-occ-table.txt', header=T, sep=",")
```

The column names of a data frame are retrieved and modified using the `colnames()` command. For example,

```
colnames(Traffic)
[1]  "Occ1"  "Flow1" "Occ2"  "Flow2" "Occ3"  "Flow3"

colnames(Traffic) = c("O1", "F1", "O2", "F2", "O3", "F3")
colnames(Traffic)
[1]  "O1" "F1" "O2" "F2" "O3" "F3"
```

The following code completes the third task:

```
# storage for the new data frame
newTraffic <- matrix(0, nrow(Traffic), 2)

# Loop through the rows of Traffic
for(i in 1:nrow(Traffic))
{

   # Get the row of data
   traffic.i <- Traffic[i,]

   # Get the maximum value of flow
   maxflow <- max( traffic.i[c(2,4,6)] )

   # Get the corresponding value of occ
   maxocc <- as.numeric( traffic.i[c(1,3,5)][which.max(traffic.i[c(2,4,6)])] )

   # record the values into the new data frame
   newTraffic[i,] <- c(maxflow, maxocc)

}

# convert to a data frame
newTraffic <- as.data.frame(newTraffic)

# column names for the new data frame
colnames(newTraffic) = c("maxflow", "maxocc")

# write the new data set to a file
write.table(newTraffic, file="flow-occ-table_clean.txt", sep="\t", header=T,
   row.names=1)
```

4 A plotting example

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,]
```

If you include a line that says `attach(Y)` then you can access the variables in `Y` just by typing their name. For example, to get `Occ1` you would type `Occ1` instead of `Y$Occ1` or `Y[,1]`. To detach the data set after you are finished you type `detach(Y)`. 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 the plotting characters 2/3 the size they are by default

4. Attach the points for each variable with lines

5. Duplicate the above plot but also label the y-axis with `y-axis variable`, the x-axis with `x-axis variable`, and the heading with `Example plot`.

6. Increase the font of the axis labels by 25% and the heading by 50%.

7. Modify the x-axis tick mark labels so that the they are `a,b, ..., g` instead of 0, 5, ..., 30. Also modify the y-axis tick marks so that they read

   `very low, low, medium, high, very high`

8. Make a legend to indicate which color/plotting character is which variable

9. Place the word "text" written five times the default size in the color yellow at the ordered pair (15, .035)

Before beginning note that the `plot` command takes a minimum of one input. If only a single input is given, call it `v`, then that is treated as the variable on the y-axis and the x-axis variable is automatically generated as the numbers `1:length(v)`. If two inputs are given, then the first one is taken as the x-axis variable and the second is the y-axis variable:

```
# 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:

```
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
```

Most modifications to sizes of things in R plots will involve some variant of the tag `cex`. The default setting for `cex` is 1, and non-default values should be interpreted as a multiple of the default size. For example, `cex = .5` will make the plotting character 1/2 the size; `cex = 3` will be them 3 times larger. So for our example, this amounts to:

```
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3, cex=(2/3))
points(Y$Occ2, col=3, pch=0, cex=(2/3))
points(Y$Occ3, col=4, pch=16, cex=(2/3))
```

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)
```

The default axis labels in R are the variable names. To change them you use the `ylab` and `xlab` tags in your call to `plot`. To change the heading you use the `main` tag, all of which take strings as their inputs:

```
# the only change occurs here
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3, cex=(2/3), xlab="x-axis label",
   ylab="y-axis label", main="Example plot")

points(Y$Occ2, col=3, pch=0, cex=(2/3))
points(Y$Occ3, col=4, pch=16, cex=(2/3))
lines(Y$Occ1, col=2)
lines(Y$Occ2, col=3)
lines(Y$Occ3, col=4)
```

To change the size of the axis labels you use a `cex.lab` tag, which works basically the same as the `cex` command, except it only applies to the axis labels. Similarly, you can use `cex.main` to change the size of the heading. In this example we would modify the code to be:

```
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3, cex=(2/3), xlab="x-axis label",
   ylab="y-axis label", main="Example plot", cex.lab=1.25, cex.main=1.5)
points(Y$Occ2, col=3, pch=0, cex=(2/3))
points(Y$Occ3, col=4, pch=16, cex=(2/3))
lines(Y$Occ1, col=2)
lines(Y$Occ2, col=3)
lines(Y$Occ3, col=4)
```

To modify the tick marks on a plot is a bit more involved; you need to manually turn off the x and y axis and customize it after the plot has been made. You begin by making the plot as you normally would, but add the tag `axes=FALSE`, in:

```
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3, cex=(2/3), xlab="x-axis label",
   ylab="y-axis label", main="Example plot", , cex.lab=1.25, cex.main=1.5, axes=FALSE)
points(Y$Occ2, col=3, pch=0, cex=(2/3))
points(Y$Occ3, col=4, pch=16, cex=(2/3))
lines(Y$Occ1, col=2)
lines(Y$Occ2, col=3)
lines(Y$Occ3, col=4)
```

Now you can customize each axis by using the `axis` command. The basic way it works is that `axis(w, x, L)` will modify the x-axis if `w=1` and the y-axis if `w=2`, and will place the labels contained in `L` at the points defined by `x`, so `length(x)` had better equal `length(L)`. In our example this works out as:

```
# x-axis
axis(1, seq(0, 30, length=7), c("a", "b", "c", "d", "e", "f", "g"))

#y-axis
axis(2, seq(0, .04, length=5), c("very low", "low", "medium", "high", "very high"))

# places a box around the plot as in a normal plot
box()
```

By default the tick mark labels appear horizontally, but you can make them vertical by adding a `las=2` tag to the calls to `axis`. Now we want to make a legend, which will require a call to `legend`. There are **many** options available with the legend command but I 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 through the points in the legend. In our example this amounts to

```
plot(Y$Occ1, col=2, ylim=c(0, .04), pch=3, cex=(2/3), xlab="x-axis label",
   ylab="y-axis label", main="Example plot", , cex.lab=1.25, cex.main=1.5, axes=FALSE)
points(Y$Occ2, col=3, pch=0, cex=(2/3))
points(Y$Occ3, col=4, pch=16, cex=(2/3))
lines(Y$Occ1, col=2)
lines(Y$Occ2, col=3)
lines(Y$Occ3, col=4)
axis(1, seq(0, 30, length=7), c("a", "b", "c", "d", "e", "f", "g"))
axis(2, seq(0, .04, length=5), c("very low", "low", "medium", "high", "very high"))
box()

# (25, .035) seems like a reasonable place to put 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)
```

You can use the `text(x,y,string)` command to place text in arbitrary spots in the plot. When calling `text`, the first argument is the x-coordinate, the second is the y-coordinate, and the third is the text you want to appear there. You can also modify the size and color and such using the `cex` and `col` tags as usual. In this example we would write add to all of the code above the line

```
text(15, .035, "text", cex=5, col=7) # 7 is the code for yellow
```

To further customize plots there is the command `par` that can be used set various graphical parameters–only a few will be mentioned here. You can set this before the plot has been called and will be remembered until you close the plot. You can use this to preset all of the `cex` parameters, as well as background colors, and to set up charts of plots. The following sets up a plotting environment for a 2-by-2 grid of plots with green background, blue axis labels, red tick mark labels, black main heading labels, and modified sizes on the labels.

```
plot.env <- par(bg="green", mfrow=c(2,2), cex.main=1.125, cex.lab= 1.25,
   col.axis="red", col.lab="blue", col.main="black")
plot( rnorm(100), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(0,1) variables")
plot( rnorm(100,mean=.5), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(.5,1) variables")
plot( rnorm(100,mean=1), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(1,1) variables")
plot( rnorm(100,mean=1.5), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(1.5,1) variables", cex.main=5)
```

You can still make specifications within the `plot` command that will override what was specified by `par`. One last tip about plotting is how to save your plots on the fly. For example, suppose are about to generate the plot above but want to saved to a file called !plot.pdf!. You can do this using the `pdf` command.

```
# type this before you make the plot
pdf("plot.pdf")

# make the plot
plot.env <- par(bg="green", mfrow=c(2,2), cex.main=1.125, cex.lab= 1.25,
   col.axis="red", col.lab="blue", col.main="black")
plot( rnorm(100), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(0,1) variables")
plot( rnorm(100,mean=.5), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(.5,1) variables")
plot( rnorm(100,mean=1), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(1,1) variables")
plot( rnorm(100,mean=1.5), xlab="Index", ylab="y-axis",
   main="Scatterplot of 100 N(1.5,1) variables")

# type this after you've made the plot
dev.off()
```

Now there should be the specified file in your current directory. Note that when you write the plot directly to a file it does not appear on the screen first, so you will want to make sure the plot looks the way you want before saving it in this way.