Final

Mathieu Sauterey (mjs2364)

Section 002

Instructions (Read this completely first)

You should complete the exam by editting this file directly. Please knit the file often, so that if you make a mistake you catch it before the end of the exam. You will have exactly 160 minutes from your start time to complete the exam. **At the end you must turn in your knitted .pdf file and raw .Rmd file on Courseworks.**

When the time is up, you must shut your computer immediately. We will take off points from anyone whose computer is still open after time is up.

You may use your class notes for the exam, but not the internet. You absolutely may not communicate with anyone else during the exam. Doing so will result in an F in this class and likely result in termination from the MA program.

Question 0 (5 points)

a. (0.5 points) Place your section number as the date of the document. If you don't know your section number, you can determine it below based on when your lab meets.

- Section 002 Lab meets TR 7:40pm-8:55pm
- Section 003 Lab meets TR 11:40am-12:55pm
- Section 004 Lab meets MW 8:40am-9:55am
- Section 005 Lab meets TR 8:40am-9:55am

b. (0.5 points) Write your name and UNI as the author of the document.

c. (4 points) Please present your answers in a readable format. This includes things like indenting your code and generally presenting easy-to-read code. Presentation of the overall Markdown document will be considered as well.

Question 1: Fitting Data (49 points)

(a) (4 points) You have an urn with 30 balls -- 10 are red, 10 are blue, and 10 are green. Write a single line of code to simulate randomly picking 400 balls from the urn with replacement. Create a variable `num_green` that records the number of green balls selected in the 400 draws.

```
set.seed(1)
```

```r
ball.sim <- sample(1:3,400, replace = TRUE) # 1 for red, 2 blue, 3 green
num_green <- sum(ball.sim == 3)
```

(b) (4 points) Now repeat the above experiment 1000 times. Create a vector `data`, such that each element in `data` is the result (counting the number of green balls) from an independent trial like that described in 1.a.

```r
set.seed(2)

data <- c()
for (i in 1:1000){
  ball.sim <- sample(1:3,400, replace = TRUE)
  data[i] <- sum(ball.sim == 3)
}

# If you can't produce a data vector, uncomment the following line of code
# and use it for the rest of the questions:

# data <- rnorm(1000, 133, 9)
```

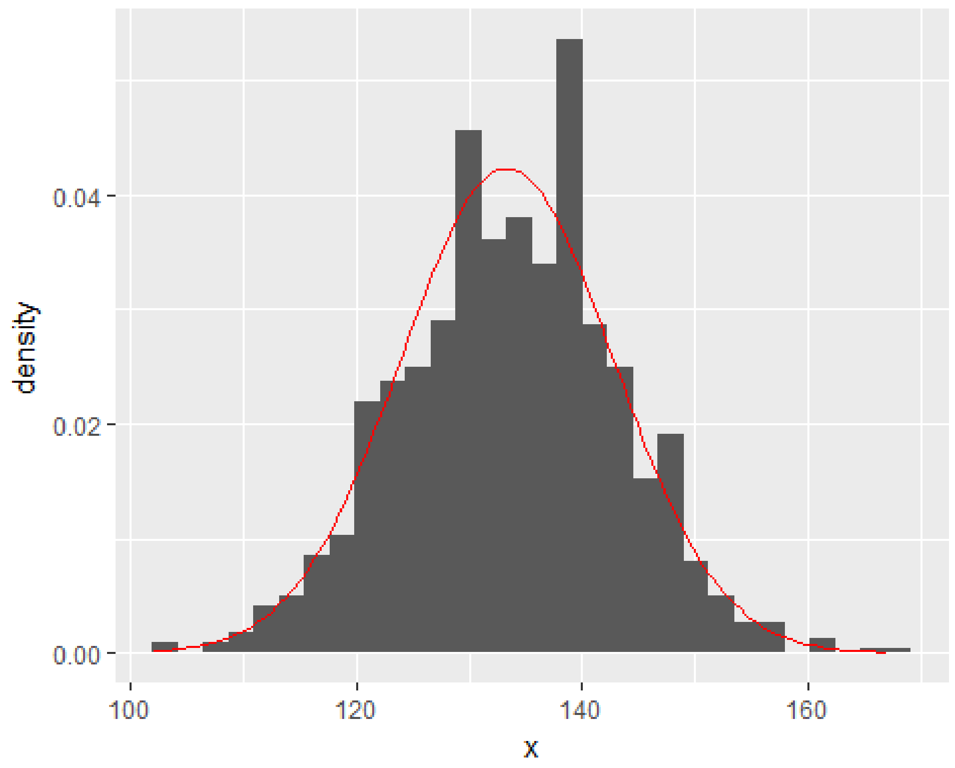
(c) (6 points) Note that if a random variable X is the number of green balls selected in 400 draws with replacement from the urn, then X follows a binomial distribution, namely $X \sim bin(n, p)$ where p is the probability of selecting a green ball from the urn in a single draw, n is the total number of draws, and

$$Pr(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \quad \text{for} \quad x = 0,1,\dots,400,$$

(d)

with $\mathbb{E}[X] = np$. Recall that the binomial distribution is well-approximated by the normal distribution. To see that this is a good approximation, plot a histogram of your data from 1.b along with a normal density curve colored red having mean $np = 400 * (1/3)$ and variance $np(1-p) = 400 * (1/3) * (2/3)$.

```r
library(ggplot2)
ggplot(data.frame(x=data))+
  geom_histogram(aes(x=x, y=..density..))+
  stat_function(aes(x=x), fun=dnorm, args=list(mean=(400/3), sd=sqrt(800/9))
,color="red")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



(d) (5 points) Give the proportion of values in your data vector that are less than or equal to 100 or greater than or equal to 150. Using `R` functions for probability distributions, compare this proportion to the probability that a normal random variable having mean $np = 400 * (1/3)$ and variance $np(1-p) = 400 * (1/3) * (2/3)$ is less than 100 or greater than 150.

```
sum(data <= 100 | data >= 150)/length(data) #proportion

## [1] 0.046

gt150 <- (1-pnorm(150, mean=(400/3), sd=sqrt(800/9))) # greater than 150
lt100 <- pnorm(100, mean=(400/3), sd=sqrt(800/9)) # Less than 100
gt150 + lt100 # Less than $100$ or greater than $150$.

## [1] 0.03875341
```

About 4.1% of values in the data vector that are less than or equal to 100 or greater than or equal to 150. Using R functions we find a similar proportion of 3.88%

(e) (5 points) Write a function `MomentEstimator` that takes two input: `data`, a vector containing the number of green balls selected in each experiment, and `n` the total number of balls selected in each experiment (in 1.a, $n = 400$ but we write the function where this could change) and returns a single output value `phat` that is the method of moments estimate of the probability p. After the function is written run the code `MomentEstimator(data, 400)` to see the method of moment estimator from your simulated data in 1.b and the code `MomentEstimator(80, 100)` to check the functionality.

```r
MomentEstimator <- function(data, n){
  m <- mean(data)
  phat <- m/n
  return(phat)
}

MomentEstimator(data, 400)

## [1] 0.3344825

MomentEstimator(80, 100)

## [1] 0.8
```

(f) (7 points) If num is the number of experiments run (i.e. in 1.b num is 1000) and x_i is the number of green balls selected in experiment $i = 1,2, \ldots, num$, then the log-likelihood for the binomial distribution is given by the following:

$$\ell(p) = \sum_{i=1}^{num} \left(\log \binom{n}{x_i} + x_i \log p + (n - x_i)\log(1-p) \right).$$

(g) Find the MLE estimate by writing a function that calculates the negative log-likelihood and then using `nlm()` to minimize it. Find the MLE estimate in this way on your data from part 1.b. Use an initial guess of $p = 0.5$.

```r
neg.ll <- function(p, vec){

  n <- length(vec)
  l<- log(choose(n,vec))+vec*log(p)+log(1-p)*(n-vec)
  return(-sum(l))
}

(nlm(neg.ll, 0.3, vec=data))$estimate

## [1] 0.133793
```

(g) (10 points) Use the bootstrap procedure to estimate the variance of your method of moments estimator. Use 5000 bootstrap resamples of the data (stored in vector `data`) you calculated in 1.b. The actual variance of the method of moments estimator is $5.56e-07$.

```r
set.seed(3)

boot <- c()

for (b in 1:5000){
  # resamp <- sample(data, 1000, replace=TRUE)
  # boot[b] <- (mean(resamp)/length(resamp))

  ball.sim <- sample(1:3,400, replace = TRUE)
  data[b] <- sum(ball.sim == 3)
  boot[b] <- MomentEstimator(data[b], 400)
}

var(boot)

## [1] 0.0005640775
```

(h) (8 points) Use simulation to provide evidence that the method of moments estimate is consistent (meaning that as the sample size increases *num*, the estimator converges to the population value).

```
set.seed(4)

#Let's try with 100,000 and we name the new vector data2

data2 <- c()
for (i in 1:100000){
  ball.sim <- sample(1:3,400, replace = TRUE)
  data2[i] <- sum(ball.sim == 3)
}

MomentEstimator(data2, 400)

## [1] 0.3333133
```

Previously with 1000 we had a p estimator equal to 0.3335125, and it is equal to 0.333342, which is closer to the actual probability of 1/3. Therefore as the sample size increases, the estimator converges to the population value.

Question 2: Transforming Data (46 points)

Gross domestic product (GDP) is a measure of the total market value of all goods and services produced in a given country in a given year. The percentage growth rate of GDP in year t is

$$100 \times \left(\frac{GDP_{t+1} - GDP_t}{GDP_t}\right) - 100$$

An important claim in economics is that the rate of GDP growth is closely related to the level of government debt, specifically with the ratio of the government's debt to the GDP. The file `debt.csv` contains measurements of GDP growth and of the debt-to-GDP ratio for twenty countries around the world, from the 1940s to 2010. Note that not every country has data for the same years, and some years in the middle of the period are missing data for some countries but not others.

```
debt <- read.csv("debt.csv", as.is = TRUE)
dim(debt)

## [1] 1171    4
```

```
head(debt)

##     Country Year    growth     ratio
## 1 Australia 1946 -3.557951 190.41908
## 2 Australia 1947  2.459475 177.32137
## 3 Australia 1948  6.437534 148.92981
## 4 Australia 1949  6.611994 125.82870
## 5 Australia 1950  6.920201 109.80940
## 6 Australia 1951  4.272612  87.09448
```

(a) (5 points) Calculate the average GDP growth rate for each year (averaging over countries). This is a classic split/apply/combine problem, and you should use `split()` and a function form the apply family of functions to solve it. You should not need to use a loop to do this. (The average growth rates for 1972 and 1989 should be 5.63 and 3.19, respectively. Print these values in your output.)

```
avg.growth <- function(df){
  return(mean(df$growth))
}

growth.split <- split(debt, debt$Year)
growth.year <- sapply(growth.split, avg.growth)

this.growth <- names(growth.year) %in% c("1972","1989")
growth.year[this.growth]

##     1972     1989
## 5.629986 3.186842
```

(b) (5 points) Calculate the average GDP growth rate for each year (averaging over countries). This is a classic split/apply/combine problem, and you should use `ddply()` to solve it. Save your output as `year.avgs` and change the column names to be `Year` and `AverageGrowth`. You should not need to use a loop to do this. (The average growth rates for 1972 and 1989 should be 5.63 and 3.19, respectively. Print these values in your output.)

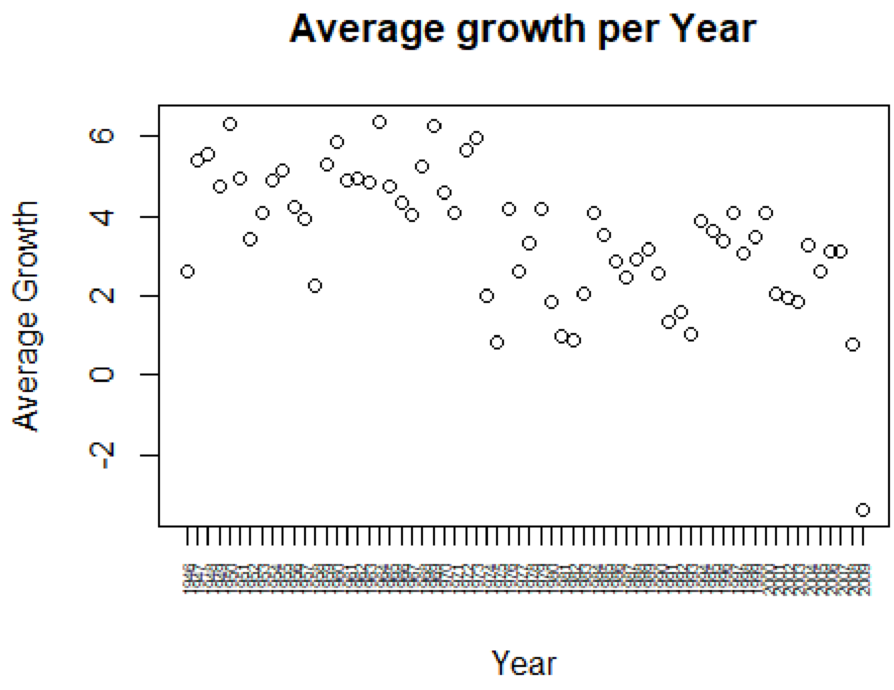
```
library(plyr)
year.avgs <- ddply(debt, .(Year), avg.growth)
colnames(year.avgs) <- c("Year","AverageGrowth")
this.growth2 <- year.avgs$Year %in% c("1972","1989")
year.avgs[this.growth2,2]

## [1] 5.629986 3.186842
```

(c) (4 points) The `year.avgs` dataframe from 2.b will be sorted by Year, meaning row 1 corresponds to 1946, row 2 to 1947, and so on with row 64 corresponding to 2009. Produce a dataframe that instead has row 1 corresponding to the year with the largest average growth, row 2 to the year with the second largest average growth, and so on with row 64 corresponding to the year with the smallest average growth.

```
ord.growth <- order(year.avgs$AverageGrowth, decreasing = TRUE)
year.avgs.ord <- year.avgs[ord.growth,]
```

(d) (3 points) Make a plot of the growth rates (y-axis) versus the year (x-axis) using your results from either 2.a or 2.b (they should be the same). Make sure the axes are labeled appropriately.

```
plot(1:nrow(year.avgs), year.avgs[,2], xlab = "Year", xaxt="n",
     ylab="Average Growth", main = "Average growth per Year")
axis(side = 1, at = 1:nrow(year.avgs), labels=year.avgs[,1], las =2, cex.axis
= 0.5)
```



(e) (6 points) The function `cor(x,y)` calculates the correlation coefficient between two vectors x and y. First calculate the correlation coefficient between GDP growth and the debt ratio over the whole data set (all countries, all years). Your answer should be −0.1995. Second, compute the correlation coefficient separately for each year, and plot a histogram of these coefficients. The mean of these correlations should be −0.1906. Do not use a loop.

```
cor(debt$growth, debt$ratio)

## [1] -0.199468

cor.f <- function(df){
  return(cor(df$growth, df$ratio))
}

cor.split <- split(debt, debt$Year)
cor.year <- sapply(cor.split, cor.f)

mean(cor.year)
```

```
## [1] -0.1905526
```

As expected the mean of these correlations is -0.1906.

(f) (3 points) Some economists claim that high levels of government debt cause slower growth. Other economists claim that low economic growth leads to higher levels of government debt. The data file, as given, lets us relate this year's debt to this year's growth rate; to check these claims, we need to relate current debt to future growth. Create a new dataframe that contains all the rows of `debt` for France. It should have 54 rows and 4 columns. Note that some years are missing from the middle of this data set.

```
debt.france <- subset(debt, Country == "France")
dim(debt.france)

## [1] 54  4
```

As expected, it has 54 rows and 4 columns.

(g) Create a new column in your dataframe for France created in 2.f, labeled `next.growth`, which gives next year's growth *if* the next year is in the data frame, or `NA` if the next year is missing. Do this in two steps.

1. (7 points) First write a function `n.growth()` that takes in two arguments, a year and a dataframe (that has the same columns as `debt` but rows only corresponding to a single country), and outputs the proper next growth value for that year and that dataframe (i.e. it gives next year's growth if the next year is in the input dataframe, or `NA` if the next year is missing).

```
n.growth <- function(year, df){

  this.year <- which(df$Year == year)
  next.year <- this.year+1

  if(df$Year[length(df$Year)] == year){ # Puts NA for the last year
    return(NA)
  }

  if (df$Year[next.year] != year+1){
    return(NA)
  }

  else{
    return(df$growth[next.year])
  }
}
```

2. (5 points) Next use `n.growth()` and one of the functions from the apply family to create the `next.growth` column in the dataframe. (`next.growth` for 1971 should be 5.886, but for 1972 it should be `NA`.)

```
n.growth(1971, debt.france) # Verification

## [1] 5.885827
```

```r
n.growth(1972, debt.france) # Verification

## [1] NA

debt.france$next.growth <- lapply(debt.france$Year, n.growth, df=debt.france)
```

As expected, `next.growth` for 1971 is 5.886, but for 1972 it is `NA`.

(h) (8 points) Add a `next.growth` column, as in 2.g, to the whole of the `debt` data frame. Make sure that you do not accidentally put the first growth value for one country as the `next.growth` value for another. (The `next.growth` for France in 2009 should be `NA`, not 9.167 .) Hints: Write a function to encapsulate what you did in 2.f, and apply it using `ddply()`.

```r
encaps <- function(this.country) {
one.country <- subset(debt, Country == this.country)

return(sapply(one.country$Year, n.growth, df=one.country))
}

head(sapply(debt$Country, encaps)[1])

## $Australia
##  [1]  2.459474567  6.437534097  6.611993849  6.920201242  4.272611548
##  [6]  0.904651600  3.119280285  6.216813827  5.462089671  3.441608143
## [11]  2.003011657  4.802932156  6.156573691  4.191212638  0.689465884
## [16]  6.248841495  6.125569558  6.841499618  5.107341974  2.783063954
## [21]  6.751673467  5.878890615  6.081697094  6.359787029  4.448824070
## [26]  2.805801550  5.421130233  2.479194958  2.707462653  4.031053058
## [31]  1.046756909  2.901082071  5.265974799  1.998402053  4.155751638
## [36] -0.009740007 -0.526011598  6.414499812  5.716196264  2.110579592
## [41]  4.396280959  4.030412731  4.574949066  1.604439522 -1.250116968
## [46]  2.112287375  3.897211812  5.013538670  3.488831888  4.281607041
## [51]  3.996996860  5.051379311  4.401723734  3.426129648  2.089340216
## [56]  4.247348493  2.965939765  3.842278649  2.759194655  2.840372034
## [61]  4.048780298  2.354223964  0.732249739           NA

#This gives a list of each country with each element being the growth rate
column vector
```