# Homework 6

Gavin Gunawardena

7/9/21

## Instructions

There are only 3 exercises for this homework; these will be challenging enough that you don't need four. You will get 10 point bonus for completing the exercises.

*Warning* I will continue restricting the use of external libraries in R, particularly `tidyverse` libraries. You may choose to use `ggplot2`, but take care that the plots you produce are at least as readable as the equivalent plots in base R. You will be allowed to use whatever libraries tickle your fancy in the final project.

### Reuse

For many of these exercises, you may be able to reuse functions written in prior homework. Define those functions here. I'm also including data vectors that can be used in some exercises.

```
Year <- c(1936, 1946, 1951, 1963, 1975, 1997, 2006)
CaloriesPerServingMean <- c(268.1, 271.1, 280.9, 294.7, 285.6, 288.6, 384.4)
CaloriesPerServingSD <- c(124.8, 124.2, 116.2, 117.7, 118.3, 122.0, 168.3)

#Data Frame
CookingTooMuch.dat <- data.frame(
  Year <- Year,
  CaloriesPerRecipeMean <- CaloriesPerServingMean,
  CaloriesPerServingSD <- CaloriesPerServingSD
)

#Functions
norm.pdf <- function(x, mu=0, sigma=1) {
  pi2 <- pi*2
  var_1 <- sigma^2
  part1 <- (1/((sigma)*(sqrt(pi2))))
  part2 <- (exp((((-1*(x-mu)^2)/(2*var_1)))))
  return(part1 * part2)

}
```

## Exercise 1

### Part a.

Write a function or macro to compute mean, standard deviation, skewness and kurtosis from a single vector of numeric values. You can use the built-in mean function, but must use one (and only one) for loop to compute the rest. Be sure to include a check for missing values. Note that computationally efficient implementations of moments take advantage of $(Y_i - \bar{Y})^4 = (Y_i - \bar{Y}) \times (Y_i - \bar{Y})^3$, etc.

See https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm for formula for skewness and kurtosis. This reference gives several definitions for both skewness and kurtosis, you only need to implement one formula for each. Note that for computing skewness and kurtosis, standard deviation is computed using $N$ as a divisor, not $N - 1$.

Your function should return a list with `Mean`, `SD`, `Skewness` and `Kurtosis`. If you use IML, you will need to implement this as a subroutine and use call by reference; include these variables in parameter list.

```
# mean sd skewness kurtosis combo function
MeanStdevSkewnessKurtosis <- function(arrayOfValues){
  mean.x <- 0
  stDev.x <- 0
  sum.x <- 0
  skewness.x <- 0
  kurtosis.x <- 0
  runningCalcTotal1.x <- 0 #standard deviation
  runningCalcTotal2.x <- 0 #skewness
  runningCalcTotal3.x <- 0 #kurtosis
  n <- 0
  #Mean calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      sum.x <- sum.x + arrayOfValues[i]
      n <- n+1
    }
  }
  mean.x <- sum.x/n

  #Standard Deviation Calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal1.x <- runningCalcTotal1.x + ((arrayOfValues[i]-mean.x)
* (arrayOfValues[i]-mean.x))
    }
  }
  stDev.x <- sqrt(runningCalcTotal1.x/n)

  #Fisher-Pearson coefficient of skewness Calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal2.x <- (runningCalcTotal2.x + ((arrayOfValues[i]-
mean.x) * (arrayOfValues[i]-mean.x)^2))
     }
  }
  skewness.x <- ((runningCalcTotal2.x/n) / stDev.x^3)

  #kurtosis
    for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal3.x <- (runningCalcTotal3.x + ((arrayOfValues[i]-
mean.x) * (arrayOfValues[i]-mean.x)^3))
     }
  }
  kurtosis.x <- ((runningCalcTotal3.x/n) / stDev.x^4)
  c(mean.x,stDev.x,skewness.x,kurtosis.x)
}
print("Mean - Standard Deviation - Skewness - Kurtosis")

## [1] "Mean - Standard Deviation - Skewness - Kurtosis"

print(MeanStdevSkewnessKurtosis(c(1,5,6,NA,9)))

## [1]  5.25000000  2.86138079 -0.25210697  1.95542218
```

### Part b.

Test your function by computing moments for `Mean55` from `Khan.csv`, for `ELO` from `elo.csv` or the combine observations from `SiRstvt`.

```
KhanData = "Khan.csv"
Khan.dat <- read.csv(KhanData,header=TRUE)
#check mean
#mean(Khan.dat[ , c(4)])
#Checked the others in Excel
MeanStdevSkewnessKurtosis(Khan.dat[ , c(4)])

## [1] 1.5377777778 0.2319416167 0.0058249279 3.7860045794
```

If you wish, compare your function results with the `skewness` and `kurtosis` in the `moments` package.

```
library(moments)
skewness(Khan.dat[ , c(4)],TRUE)
kurtosis(Khan.dat[ , c(4)],TRUE)
```

## Exercise 2

Consider Newton's method to find a minimum or maximum value attained by a function over an interval. Given a function $f$, we wish to find

$$\max_{x\in[a,b]} f(x)$$

Start with an initial guess, $x_0$, then generate a sequence of guesses using the formula

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

where $f'$ and $f''$ are first and second derivatives. We won't be finding derivatives analytically, instead, we will be using numerical approximations (*central finite differences*), given by

$$\begin{aligned} f' &\approx \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} \\ f'' &\approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \end{aligned}$$

where $h$ is some arbitrary small value.

We will work with the normal pdf, $f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Let $\mu = m_{1936}$ be the mean Calories per Serving from 1936, and let $\sigma = s_{1936}$ be the corresponding standard deviation. We will wish to find the $x_*$ that maximizes

$$\max pdf(x; m_{1936}, s_{1936}^2)$$

Let the initial guess be $x_0 = 180$ and let $h = 0.1$. Calculate 10 successive $x_k$, saving each value in a vector. Print the final $x_k$. Why does this value maximize the likelihood function?

```r
#declare variables
guessesArray1.x <- rep(0,10)
mean.1936 <- CookingTooMuch.dat[1,2]
sd.1936 <- CookingTooMuch.dat[1,3]
guess.first <- 180

#Create function for Newton's Method
Newtons.Method.Next <- function(x, h=0.1) {
  num.var <- ((x+(h/2))-(x-(h/2)))/h
  den.var <- ((x+h)-(2*x)+(x-h))/h^2
  return(x - (num.var/den.var))
}
Newtons.Method.Next <- function(x, h=0.1) {
  num.var <- (norm.pdf(((x+h/2)),mean.1936,sd.1936)-norm.pdf((x-
h/2),mean.1936,sd.1936))/h
  den.var <- (norm.pdf(x+h,mean.1936,sd.1936)-
2*norm.pdf(x,mean.1936,sd.1936)+norm.pdf(x-h,mean.1936,sd.1936))/h^2
  return(x - (num.var/den.var))
}
#populate guesses array
i <- 1
  while (i<=length(guessesArray1.x)) {
    if(i == 1){
      guessesArray1.x[i] = 180
      i<-i+1
    }
    else{
      guessesArray1.x[i] <- Newtons.Method.Next(guessesArray1.x[i-1])
      i <- i+1

    }
  }
# print("Obtained Guesses")
# guessesArray1.x[]
print("Final Guess")

## [1] "Final Guess"

guessesArray1.x[10]

## [1] 268.1
```

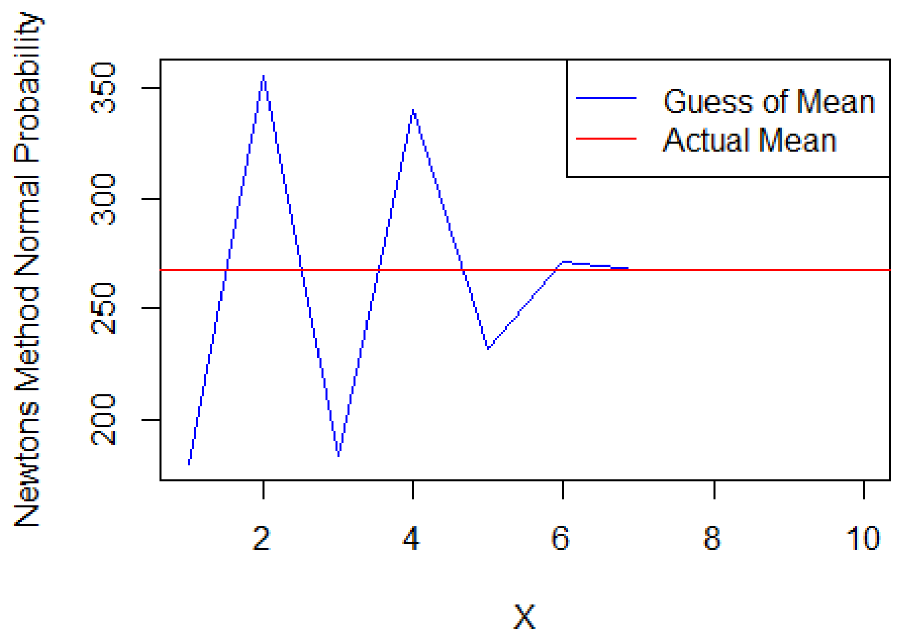
*After doing some research on Newton's Method, it works by finding local extremes via finding a root of a function. This above obtained value would maximize the likelihood function as, being equal to the mean, it is literally the zenith of the normal distribution.*

### Part b.

Plot the sequence of $x$ versus iteration number $(k)$ as the independent variable. Add a horizontal line corresponding to $m_{1936}$. How many iterations are required until $|x_{k+1} - x_k| < 10^{-6}$?

*It would take 8 iterations.*

```
x <- seq(1,10,1)
plot(x,guessesArray1.x,type='l',ylab='Newtons Method Normal
Probability',xlab='X',col='blue');
abline(a = mean.1936, b=0, col='red');
legend('topright', legend = c("Guess of Mean", "Actual Mean"),
       pch = c(NA,NA), lty = c(1, 1),
       col = c('blue','red'))
```



```
# lines(x,norm.pdf(x,mean.1936,sd.1936),col='blue',lty=2);
```

## Exercise 3

Consider the Trapezoidal Rule for integration. From "Analysis by Its History" (https://books.google.com/books/about/Analysis_by_Its_History.html?id=E2IhMXPZMNIC)

> On the interval $[x_i, x_{i+1}]$ the function $f(x)$ is replaced by a straight line passing through $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$. The integral between $x_i$ and $x_{i+1}$ is then approximated by the trapezoidal area $h \cdot (f(x_i) + f(x_{i+1}))/2$ and we obtain
>
> $$\int_a^b f(x)dx = F(x) \approx \sum_{i=1}^{N-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

We will calculate the integral for the normal pdf

$$\int_{-3}^{3} L(x; \mu, \sigma^2)dx = \int_{-3}^{3} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

with $\mu = 0$ and $sigma = 1$, using your `norm.pdf` function. We will do this by creating a sequence of approximations, each more precise than the preceding approximation.

### Part a.

Calculate a first approximation of step size $h_0 = 1$, using the sequence of $x_i = \{-2.0, -1.0, 0.0, 1.0, 2.0\}$. Let this approximation be $F_0$. Print the first approximation.

```
#Declare variables
x_i <- seq(-2,2,1)
h_0 <- 1
mu <- 0
sigma <- 1
f_0 <- rep(0,4)

#Create function for Integral
Trap.Int <- function(x, x_next, h) {
  return((h/2)*(dnorm(x,mu,sigma)+dnorm(x_next,mu,sigma)))}


#Run the function through a for loop that's going from 1 through the length
of the sequence between -2 and 2 but minus 1 to get a 4 iterations
  for (i in 1:(length(x_i)-1)){
    if(!is.na(x_i[i])){
      f_0[i] <- Trap.Int(x_i[i],x_i[i+1],1)
    }
  }
# f_0
print('First Approximation"')

## [1] "First Approximation\""

  sum(f_0)

## [1] 0.9368747
```

### Part b.

Continue to calculate a series of approximations $F_0, F_1, F_2, \ldots$ such that $F_{k+1}$ improves on $F_k$ by increasing $N$. Do this by decreasing the step size by 2, $h_{k+1} = h_k/2$. Thus, the sequence used to calculate $F_1$ will be of the form $x_i = \{-2.0, -1.5, -1.0, \ldots, 1.5, 2.0\}$

Calculate the first 10 approximations in the series and print the final approximation.

```
#Declare variables

x_i.df <- data.frame(matrix(ncol = 4, nrow = 10))
h_0 <- 1
mu <- 0
sigma <- 1
f_0 <- rep(0,5)
Final_Approx <- 0
Final_Approx_Array <- rep(0,10)
Final_Approx_Array <- rep(0,10)
```

```r
# 1:(x_i.df[1,]-1)

#Set up dynamic sequence
dyn_step <- 1
x_i <- seq(-2,2,dyn_step)

#Populate data frame with 0s
for (i in 1:4)
  {
      x_i.df[,i]<-rep(0,1)
}
#Notes to keep track of for loop
# x_i.df[1,] #5 columns
# length(x_i.df[1,]) #5
# x_i.df[,1] #10 rows
# length(x_i.df[,1]) #10

#Create function for Integral
Trap.Int <- function(x, x_next, h) {
  return((h/2)*(norm.pdf(x)+norm.pdf(x_next)))}
h <- h_0
#set up for loop to do the calculation
#First loop
  for (i in 1:length(x_i.df[,1])){

      #Second loop
      for (c in 1:(length(x_i)-1)){
      x_i.df[i,c] <- Trap.Int(x_i[c],x_i[c+1],h)

      }
      ###
  #Divide step size by 2
  h <- h/2
  #sum the row to get final result
  Final_Approx_Array[i] <- sum(x_i.df[i,])
  #Update dynamic sequence by dividing the step value
  dyn_step <- dyn_step/2
  x_i <- seq(-2,2,dyn_step)
  }

# x_i.df
# print("Final 10 Approximations")
# Final_Approx_Array
print("Final Approximation")
```

```
## [1] "Final Approximation"
```

```r
Final_Approx_Array[10]
```

```
## [1] 0.95449967
```

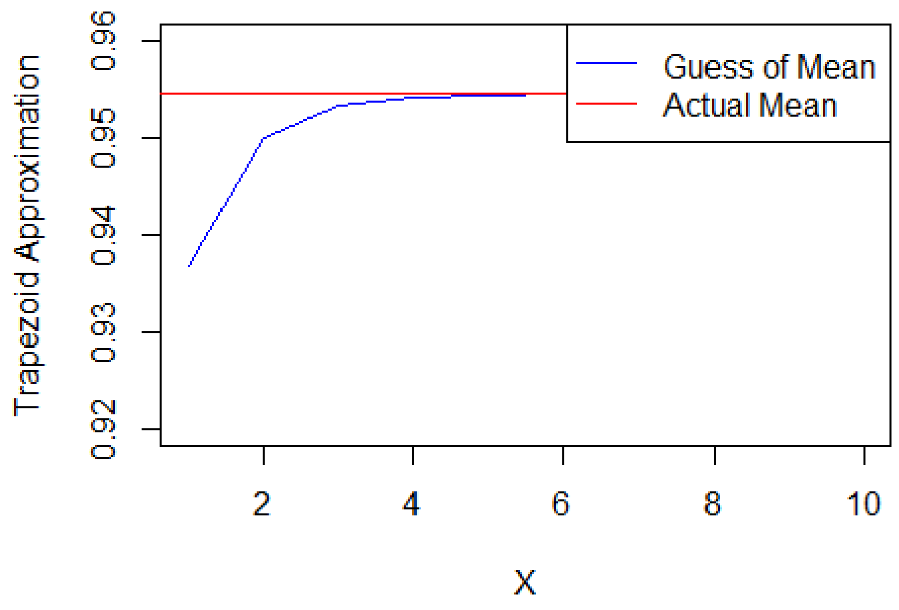
## Part c.

Plot the successive approximations $F_i$ against iteration number (you will need to define an array to store each approximation). Add a horizontal line for the expected value (`pnorm(2, lower.tail = TRUE)-pnorm(-2, lower.tail = TRUE)`). Set y-axis limits for this plot to be [0.92,0.96] to best view the progression of approximations.

It is common practice to terminate a sequence of approximations when the difference between successive approximations is less than some small value. What is the difference between your final two approximations (It should be less than $10^{-6}$)? *The difference is .00000021 which is less than 10^-6.*

```
x <- seq(1,10,1)

plot(x,Final_Approx_Array,type='l',ylab='Trapezoid
Approximation',xlab='X',col='blue',ylim=c(.92,.96));
abline(a=pnorm(2, lower.tail = TRUE)-pnorm(-2,lower.tail = TRUE),b=0,
col='red');
legend('topright', legend = c("Guess of Mean", "Actual Mean"),
       pch = c(NA,NA), lty = c(1, 1),
       col = c('blue','red'))
```


Guess of Mean
Actual Mean
Trapezoid Approximation
X