Homework 3

Ankur Patel

9/30/2020

Problem 3:

I am going to make sure that I align the curly braces when there is an if statement or while/for loop. I will also make sure that I give meaningful names and that I use BigCamelCase if I have long variable names without underscores or dots. For functions, I will use the return statement so that it is explicitly clear what is being returned.

Problem 5:

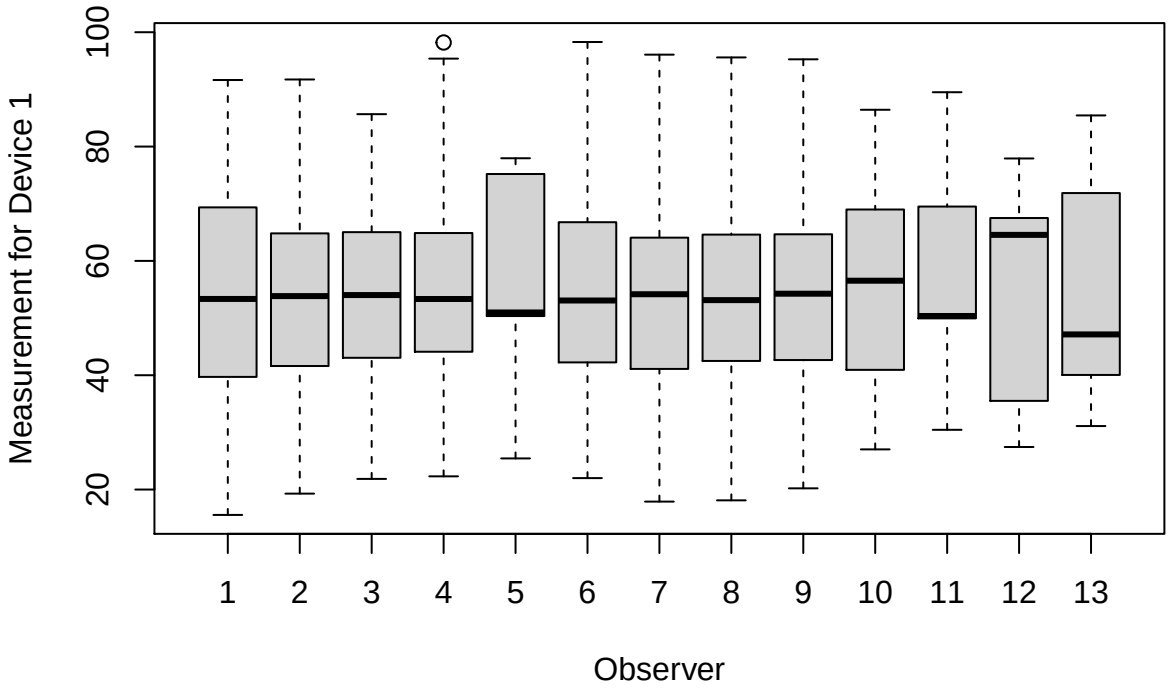
```
#summarize_data returns the mean and standard deviation of columns 1 and 2 and #the correlation between
summarize_data <- function(X)
  {
    #take the first and second column of the dataframe
    first_column <- X[,1]
    second_column <- X[,2]

    #define variables for the mean of the first and second columns
    mean_first_column <- mean(first_column)
    mean_second_column <- mean(second_column)

    #define variables for the standard deviations of the first and second         #columns
    SD_first_column <- sd(first_column)
    SD_second_column <- sd(second_column)

    #define a variable for the correlation between the two columns
    correlation <- cor(first_column,second_column)

    #return the means, standard deviations, and correlation
    return(ls = list(mean1 = mean_first_column, mean2 = mean_second_column, SD1 = SD_first_column, SD2 =
  }
#the data for Observers is denoted by Observers_data. I uploaded the RDS file
#into RStudio cloud.
Observers_data <- readRDS("HW3_data.rds")
#convert Observer to factor
Observers_data$Observer <- as.factor(Observers_data$Observer)
#Observers_summary is a data frame which has 6 columns. For a given Observer,
#it will store the means of devices 1 and 2, the standard deviations of device
#1 and 2, and the correlation between the device 1 and 2 columns. For right #now the dataframe has 5 co
Observers_summary <- matrix(data = NA, ncol = 5)
Observers_summary <- as.data.frame(Observers_summary)
```

```r
#this for loop will calculate the summary by observer using summarize_data
for (i in 1:13)
  {
    #Observers_subset is the subset of the Observers data for a given Observer
    Observers_subset <- subset(Observers_data, Observer == i)
    #Observers_subset_summary is a summary of Observers_subset. The summary is
    #performed by calling on the summarize_data function I wrote on the device
    #columns. The list is then unlisted and transformed into a vector.
    Observers_subset_summary <- summarize_data(Observers_subset[,2:3])
    Observers_subset_summary <- as.vector(unlist(Observers_subset_summary))
    #Observers_summary is the summary for the devices for each observer.
    Observers_summary <- rbind(Observers_summary, Observers_subset_summary)
  }
#the first row of Observers_summary is empty, so we reassign Observers_summary
#to the 2nd through 14th rows
Observers_summary <- Observers_summary[2:14,]
#here we append the Observer column and give column names
Observers_summary <- cbind(seq(1,13),Observers_summary)
names(Observers_summary) <- c("Observer","Mean of Device 1", "Mean of Device 2", "Standard Deviation of

#remove the rownames to get a cleaner table
rownames(Observers_summary) <- NULL
#use kable to make a table and round numeric columns to 4 digits
knitr::kable(Observers_summary, digits = 4)
```

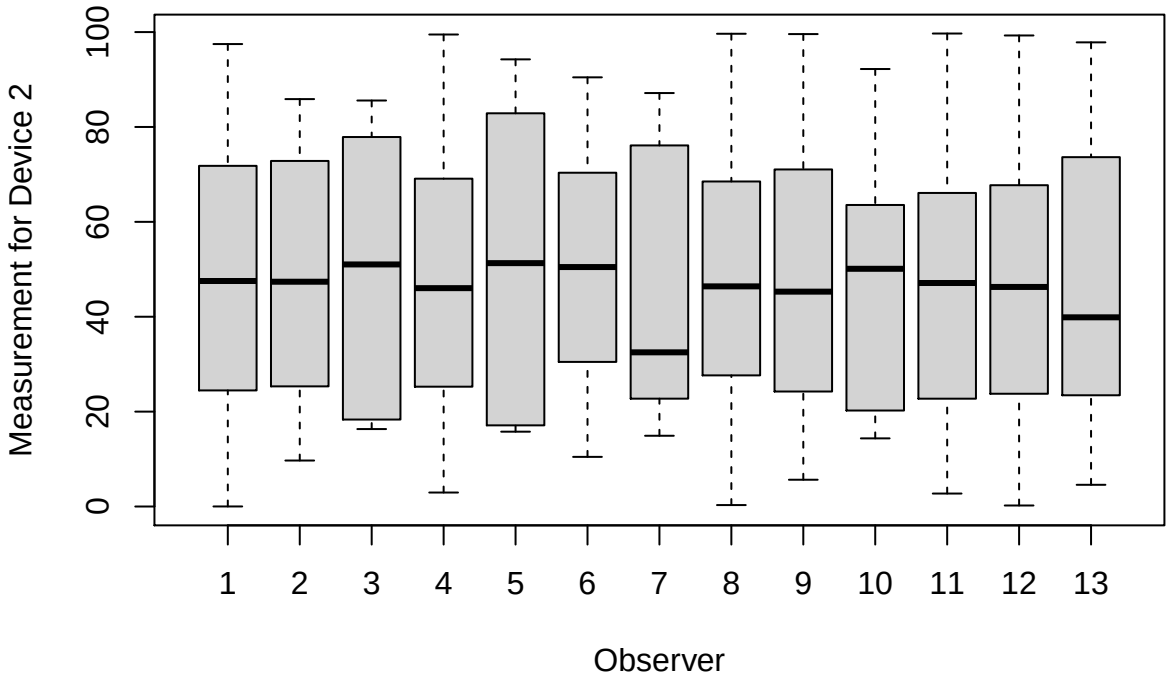
Observer	Mean of Device 1	Mean of Device 2	Standard Deviation of Device 1	Standard Deviation of Device 2	Correlation
1	54.2661	47.8347	16.7698	26.9397	-0.0641
2	54.2687	47.8308	16.7692	26.9357	-0.0686
3	54.2673	47.8377	16.7600	26.9300	-0.0683
4	54.2633	47.8323	16.7651	26.9354	-0.0645
5	54.2603	47.8398	16.7677	26.9302	-0.0603
6	54.2614	47.8303	16.7659	26.9399	-0.0617
7	54.2688	47.8355	16.7667	26.9400	-0.0685
8	54.2678	47.8359	16.7668	26.9361	-0.0690
9	54.2659	47.8315	16.7689	26.9386	-0.0686
10	54.2673	47.8395	16.7690	26.9303	-0.0630
11	54.2699	47.8370	16.7700	26.9377	-0.0694
12	54.2669	47.8316	16.7700	26.9379	-0.0666
13	54.2602	47.8397	16.7700	26.9300	-0.0656

From the table, we can see that given a device, the measures of center and spread (mean and standard deviation) are very similar across Observers. This suggests that the data may be identically distributed across Observers given a device but further analysis is needed. The small correlation suggests that the linear association between the devices for a given Observer is virtually non-existent; however it is important to keep in mind that there may still exist a non-linear association.

```r
#make some boxplots of device by Observer
boxplot(Observers_data$dev1~Observers_data$Observer, xlab = "Observer", ylab = "Measurement for Device
```

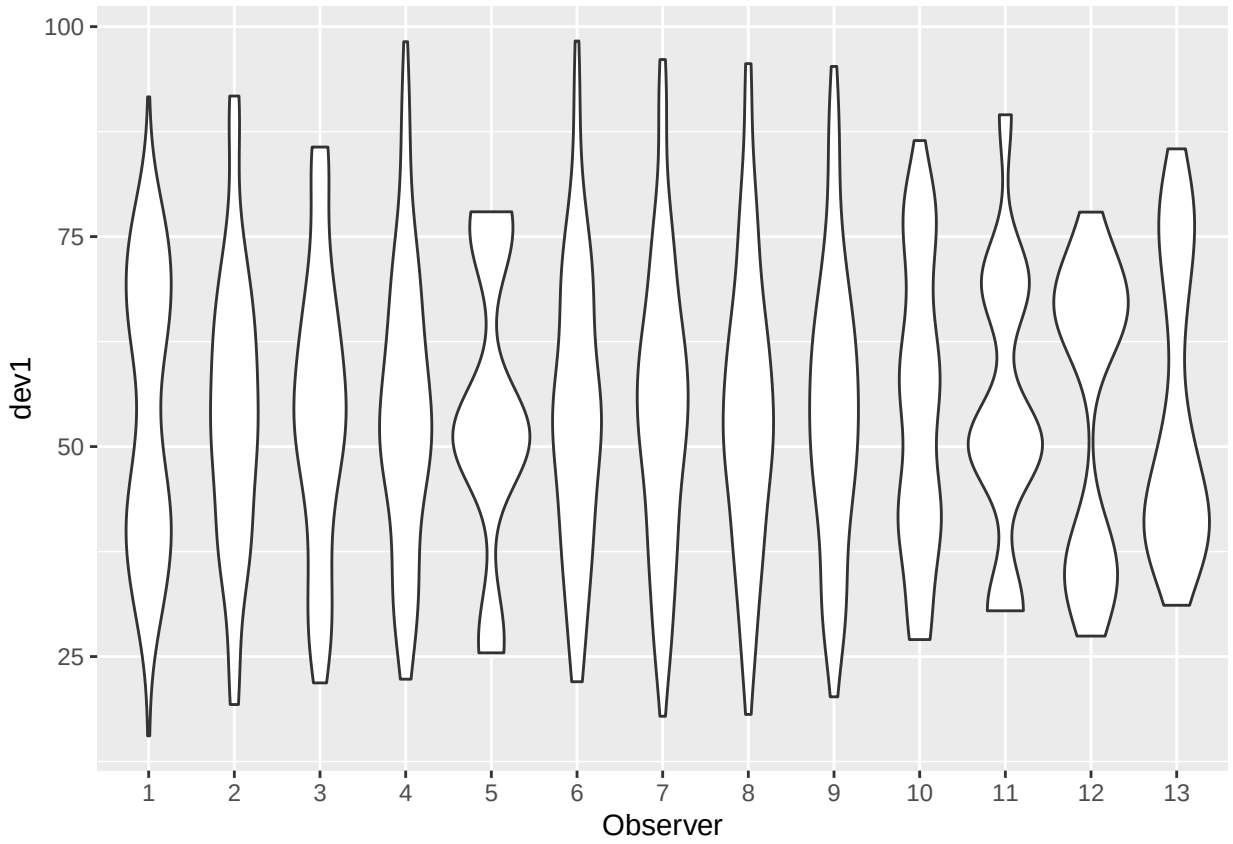



```
boxplot(Observers_data$dev2~Observers_data$Observer, xlab = "Observer", ylab = "Measurement for Device 1
```

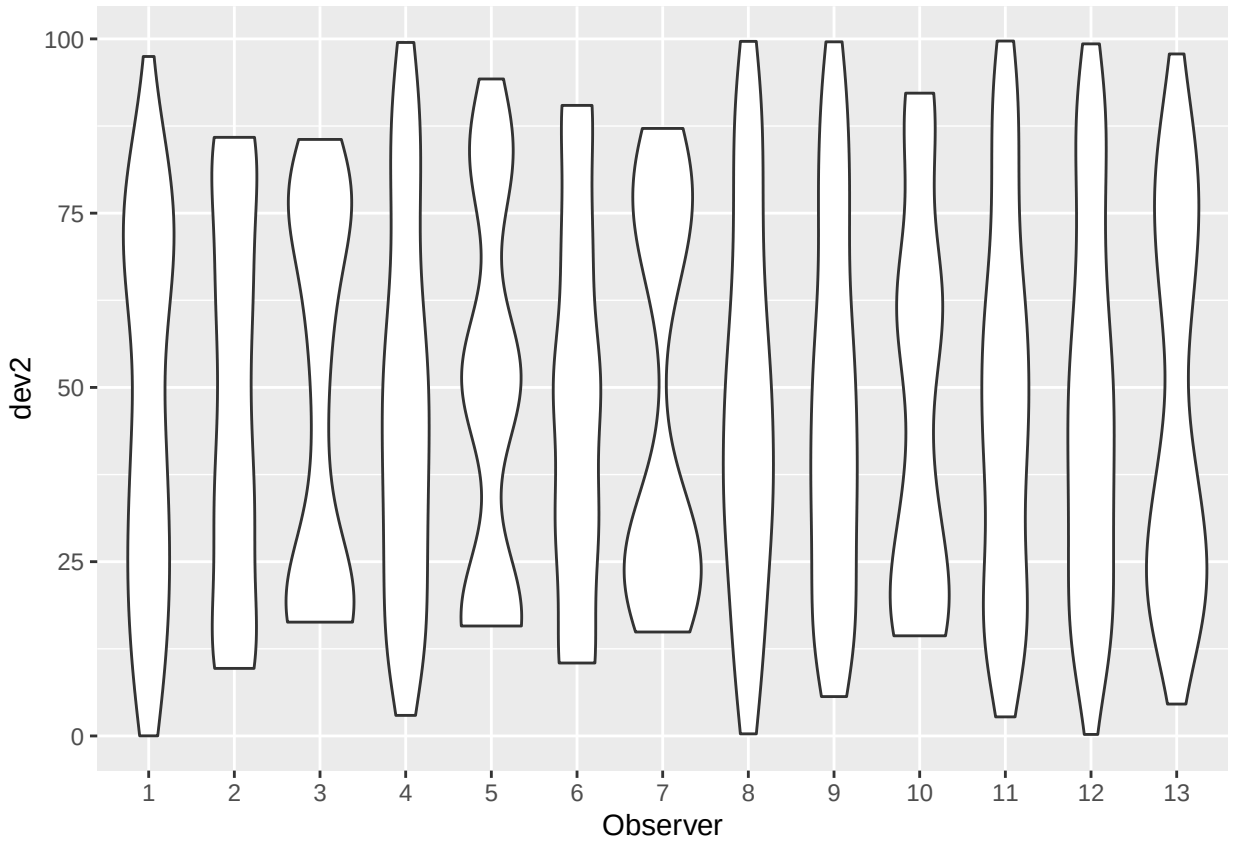


From the boxplot, it appears the spread for device 2 is less across Observers than the spread for device 1 across Observers. Device 1 appears to have an outlier for Observer 4. This suggests less uniformity in the data than our analysis using the table above.

```
#make violin plots by observer for each device
library(ggplot2)
ggplot(data = Observers_data) + geom_violin(mapping = aes(x = Observer,y=dev1))
```

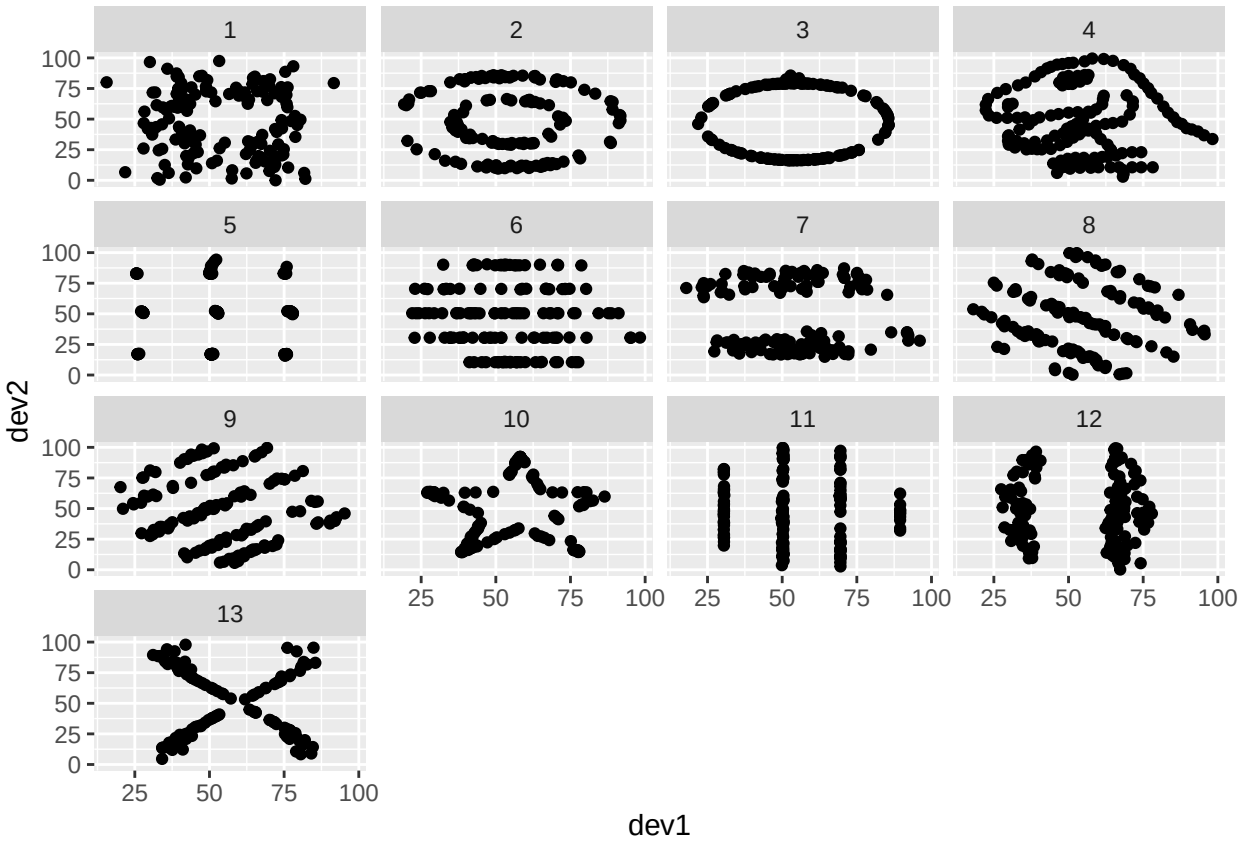



```
ggplot(data = Observers_data) + geom_violin(mapping = aes(x = Observer,y=dev2))
```



The violin plots suggest that device 1 is more homogeneous than device 2 based on the shapes of the plots which is different from what I concluded based on the boxplots. It continues to support the idea that for a given device, there is more heterogeneity among Observers than was suggested by the table.

```
#make a scatterplot of the data
ggplot(Observers_data, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)
```



Although there is little linear association between device 1 and device 2, there is definitely some strong non-linear association between the two devices based on the scatterplot.The lesson here is that just looking at the summary statistics will not give an accurate impression of the data. In addition, a boxplot and violin plot can enhance our understanding of the spread and center of the data and a scatterplot can tell us what sort of non-linear relationship there is between variables if the summary statistics suggested a weak linear relationship. In the EDA portion of a project, we should not rush to conclusions about the data from the summary statistics. Rather, we should construct appropriate plots to assess the spread, center, and possible non-linear association between variables.

Problem 6:

```
#integrand_to_use is the integrand for problem 6
integrand_to_use <- function(x)
  {
    integrand <- exp((-x^2)/2)
    return(integrand)
  }

#calculate_riemann is the function which will calculate the Riemann sums. It
#takes in the width, the integrand, and the start and end of the integral. It
#will return both the left and right Riemann sums.
calculate_riemann <- function(width,integrand,start,end)
  {
    #grid is the grid which we will use to calculate the Riemann sums, it has
    #increments of width
    grid <- seq(start,end,by = width)
```

```
    #assign the left and right sums to variables
    left_sum <- 0
    right_sum <- 0

    #for the left sum, we use the leftmost points on the grid which is why
    #the loop terminates at length(grid)-1
    for (i in 1:(length(grid)-1))
      {
        left_sum <- left_sum+width*integrand(grid[i])
      }

    #for the right sum, we use the rightmost points on the grid which is why
    #the loop starts at 2 and terminates at length(grid)
    for (i in 2:length(grid))
      {
        right_sum <- right_sum+width*integrand(grid[i])
      }

    #return the left sum and right sum as a 2 x 1 vector
    return(c(left_sum,right_sum))
  }

#the tolerance is 10^-6
tolerance <- 1e-6

#solution is the analytical solution which we are trying to estimate
solution <- integrate(f = integrand_to_use, lower = 0, upper = 1)
solution <- solution$value

#possible_widths is a vector of possible widths we will search through.
possible_widths <- c(5e-1,5e-2,5e-3,5e-4,5e-5,5e-6,5e-07)
#the number of possible widths we are searching through is stores in
#length_possible_width
length_possible_width <- length(possible_widths)
#the width needed to get a solution within 1e-6 of the analytical solution
#is width_needed. Both the right and left sums must converge at width_needed.
width_needed <- NA

for (i in 1:length_possible_width)
  {
    #use calculate_riemann to get the left and right sums
    riemann_output <- calculate_riemann(width = possible_widths[i],integrand =     integrand_to_use,sta

    #the left and right sums are stored in left_output and right_output           #respectively
    left_output <- riemann_output[1]
    right_output <- riemann_output[2]

    #we stop and return the width needed when both the left and right sums are
    #less than 1e-6 from the analytical solution
    if ((abs(left_output-solution) < tolerance) && (abs(right_output-solution)     < tolerance))
      {
        #the width needed to converge
        width_needed <- possible_widths[i]
```

```
        #the final values of the left and right sums that were within the
        #tolerance
        final_left  <- left_output
        final_right <- right_output
        break
      }
  }
```

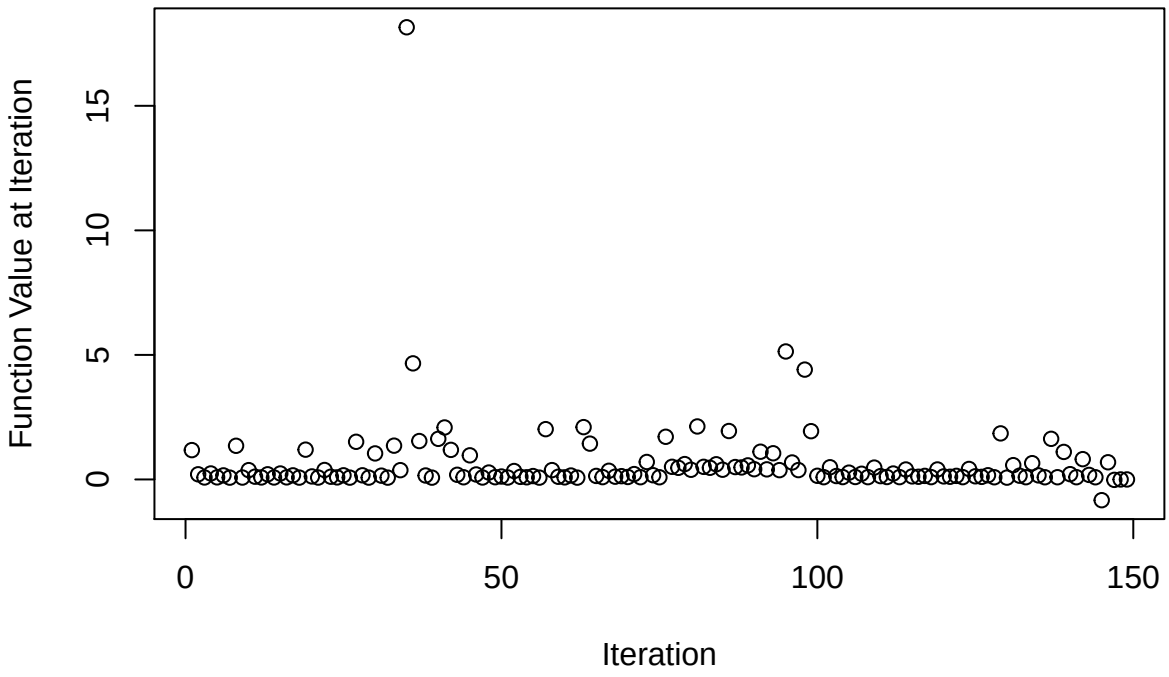
The various widths that were used were 5e-1,5e-2,5e-3,5e-4,5e-5,5e-6,5e-7.The final left sum was 0.8556254, the final right sum was 0.8556234 and the width necessary to get within 1e-6 of the solution was 5e-6. ## Problem 7:

```
#orig_function is the original function
orig_function <- function(x)
  {
    orig <- 3^x-sin(x)+cos(5*x)
    return(orig)
  }

#deriv_function is the derivative of deriv_function
deriv_function <- function(x)
  {
    deriv <- 3^x*log(3) - cos(x) - 5*sin(5*x)
    return(deriv)
  }

#The tolerance is 10^-6 and start_x is the initial value that we will use for
#the newton method
tolerance <- 1e-6
start_x <- -pi/2

#newton_method applies the newton method using the original function, the #derivative of the original f
newton_method <- function(tolerance,orig_function,deriv_function,start_x)
  {
    #before the while loop, we set x_vector which stores x_n for iteration n
    x_vector <- start_x
    #curr_x is the current approximation to the root
    curr_x <- start_x
    #iter is the current iteration and iter_vector stores how many iterations     #there have been
    iter <- 1
    iter_vector <- 1

    #since we are looking for the root, we run a while loop as long as the        #original function ev
    while(abs(orig_function(curr_x)) > tolerance)
      {
        #set the new current x using newton's method
        curr_x <- curr_x-orig_function(curr_x)/deriv_function(curr_x)
        #if the original function at the current approximation is less than           #the tolerance, w
        if (abs(orig_function(curr_x)) <= tolerance)
          {
            x_vector <- append(x_vector,curr_x)
            iter <- iter + 1
            iter_vector <- append(iter_vector,iter)
            solution <- curr_x
```

```
            break
          }
        x_vector <- append(x_vector,curr_x)
        iter <- iter + 1
        iter_vector <- append(iter_vector,iter)
  }

    #return the vector of iterations, approximations to the root, the original     #function evaluated
    return(ls = list(iter_vector = iter_vector,x_vector =                          x_vector,orig_functio
  }

#prob7_solution is the list returned by newton_method for our particular #function
prob7_solution <- newton_method(tolerance,orig_function = orig_function,deriv_function = deriv_function

#print the solution obtained via Newton's method
print(prob7_solution$solution)
```

```
## [1] -2.887058
```

```
#make a plot showing the values of the original function by iteration
plot(prob7_solution$iter_vector, prob7_solution$orig_function, xlab = "Iteration", ylab = "Function Valu
```



The tolerance was 1e-6. The interval I searched over was $(-\pi, 0)$. The solution was -2.887058.

Problem 8:

```
#set a seed for reproducibility
set.seed(2020)
#generate the data
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
```

```r
#calculate the mean of y and some vectors and matrices that will be needed
#later on
mean_y <- mean(y)
ones_vector <- rep(1,100)
J <- ones_vector %*% t(ones_vector)
J_over_n <- J * (1/100)

#load the microbenchmark library
library(microbenchmark)
#SST_loop and SST_matrix are the SST calculated using a for loop and matrix
#operations respectively
SST_loop <- 0
SST_matrix <- 0

#calculate SST using a for loop and matrix operations and time the #calculations
timing <- microbenchmark(SST_matrix <- 100*t(y) %*% (diag(100)-J_over_n) %*% y, for (i in 1:100)
    {
      SST_loop <- SST_loop + (y[i]-mean_y)^2
    })

#print the SST using the loop and the matrix calculations
print(SST_loop)
```

```
## [1] 2055420
```

```r
print(SST_matrix)
```

```
##          [,1]
## [1,] 2055420
```

The SST was 2055420. The mean time using the matrix calculations was 50.77028 and for the for loop it was 3283.83551.