Homework 3

Tessa Anwyll

September 6, 2018

Problem 4

My takeaway from looking through the style guides is that each organization may have different style preferences, but in general, style guidelines are predominantly to make code legible for other collaborators to easily edit or for readers to whome you are presenting code to be able to easily figure out what your code does. My code needs to have a more systematic naming convention (style-wise and names need to be more descriptive) and I need to utilize indentation more. I also need to comment more descriptively and open and close my braces so that it is clear what kinds of things are inside the braces and what is not. My code is generally just a disorganized series of lines so maybe if I indent and comment more it will be more legible. I also use short and apparently meaningless variable and function names (to make typing faster for me and with a meaning that I understand) but this is not good for communicating what I'm doing to other people and makes it hard to remember what I did on each line.

Problem 5

```
# Run homework 2 through lint. I commented out the code I ran so that it didn't print the output
#lint(filename = "C:/Users/Tessa/Git/STAT_5014_2018/homework/STAT_5014_homework/
#                        02_data_munging_summarizing_R_git/HW2_Anwyll_Tessa.Rmd")
```

The primary issues that lint noted were things like "put spaces around all infix operators" (and before left parentheses), "Variable function names should be all lowercase" (which I will not change, I like having capital letters at the start of each word in the name for readability), and keeping lines shorter than 80 characters. I already knew the spacing was totally inconsistent but I didn't realize just HOW inconsistent I was.

Problem 6

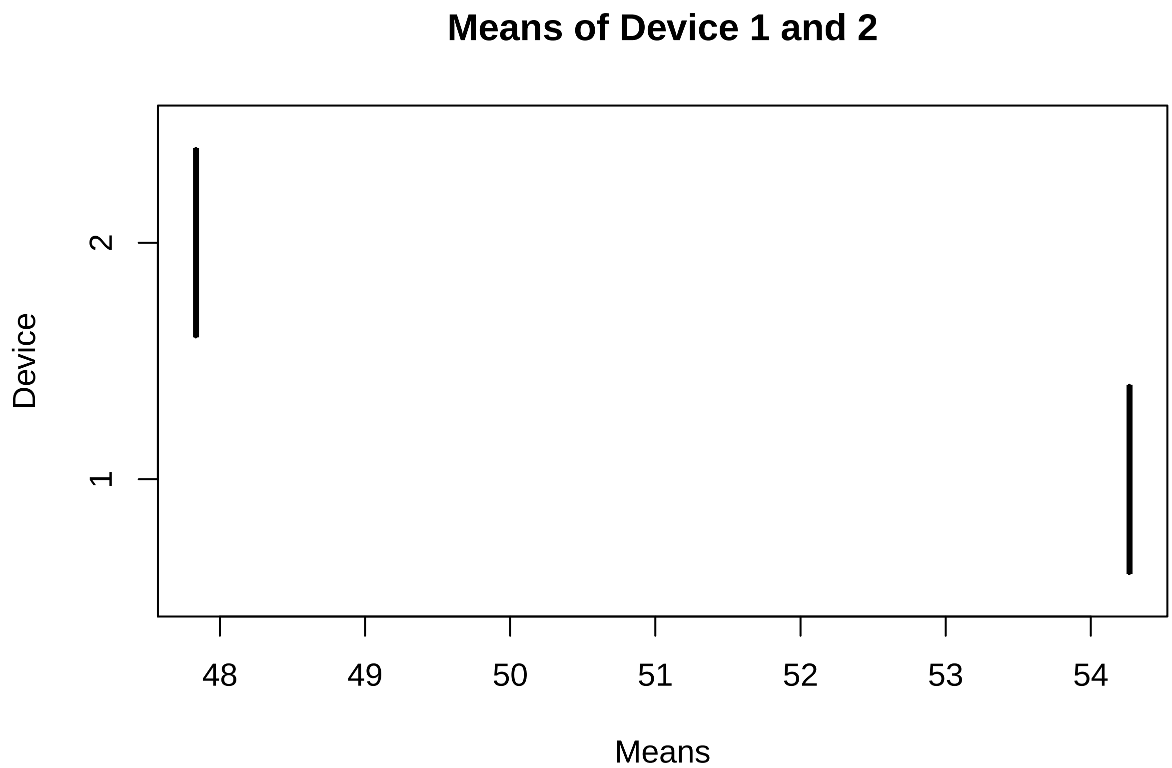
```
# Build function with arguments dev1 and dev2
repeatedStats <- function(dev1, dev2){
  dev1 <- as.vector(dev1)
  dev2 <- as.vector(dev2)
  # Calculate means of dev1 and dev2 and store in a new variable for each
  mean1 <- mean(dev1)
  mean2 <- mean(dev2)

  # Calculate standard deviations for both variables and store in new variables
  sd1 <- sd(dev1)
  sd2 <- sd(dev2)

  # Calculate the correlation between dev1 and dev2 and store in a new variable
  rValueDev1Dev2 <- cor(dev1, dev2, method = "pearson")

  # Store all data in a data frame
  results <- as.data.frame(t(c(mean1, mean2, sd1, sd2, rValueDev1Dev2)))
```

```
  # Name the columns to make it look nicer
  colnames(results) <- c("Device_1_Mean", "Device_2_Mean",
                         "Device_1_SD", "Device_2_SD", "Correlation")

  return(results)
}

input <- "C:/Users/Tessa/Git/STAT_5014_2018/homework/STAT_5014_homework/03_good_programming_R_functions/
# Read in data
measurements <- readRDS(input)

# Make a variable to use as a counter in the for loop
countObservers <- c(table(measurements$Observer))
forLoopCounter <- length(countObservers)

# Create a table to fill with results from my function and two vectors to collect the
# column names and observer numbers for labeling the final table
loopTable<- data.frame()
observerName <- vector()
finalColNames <- vector()

# Make a loop to calculate the set of statistics for each observer
for( i in 1:forLoopCounter){
  # For each i, put the all rows with that observer number in them from the
  # dev1 and dev2 columns into their own vectors
  device1 <- measurements[measurements[, "Observer"] == i, "dev1"]
  device2 <- measurements[measurements[, "Observer"] == i, "dev2"]

  # Collect observer names for the final table
  observerName <- c(observerName, i)

  # For each run of the loop, add the results to a new row at the bottom of the table
  loopTable <- rbind(loopTable, repeatedStats(device1, device2))

  # Collect the column names to use in the final table
  finalColNames <- colnames(loopTable)
}

# Add the observer names to the rows of the table
finalTable <- cbind(observerName, loopTable)

# Add "Observer" as a column name
colnames(finalTable) <- c("Observer", finalColNames)

# Gather columns so mean and sd have their own column and device is a new variable
deviceMeanTable <- gather(finalTable, key = "Device", value = "Mean",
                          Device_1_Mean, Device_2_Mean)
deviceSDTable <- gather(finalTable, key = "Device", value = "Standard_Deviation",
                        Device_1_SD, Device_2_SD)
finalTable <- cbind(finalTable$Observer, select(deviceMeanTable, Device, Mean),
                    select(deviceSDTable, Standard_Deviation, Correlation))

# Make device numbers look nicer
```

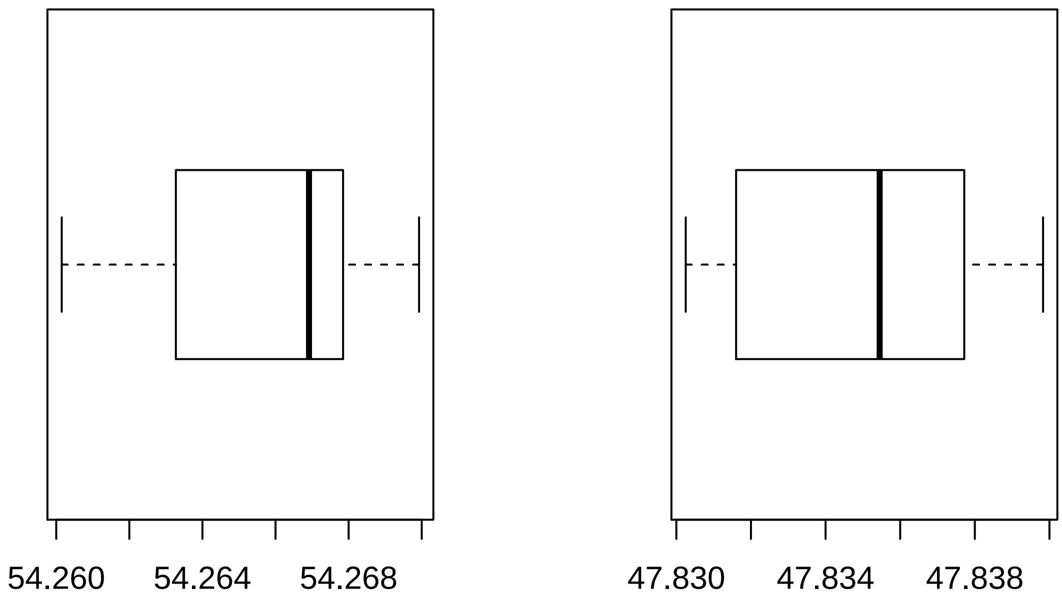
```r
for(i in 1:length(finalTable$Device)){
  if(finalTable[i, "Device"] == "Device_1_Mean"){
    finalTable[i, "Device"] <- "1"
  }
    else{
      if(finalTable[i, "Device"] == "Device_2_Mean"){
      finalTable[i, "Device"] <- "2"
      }
    }
}

# Put final column names on table
colnames(finalTable) <- c("Observer", "Device", "Mean", "Standard_Deviation", "Correlation")

# Print final table
finalTable
```

```
##    Observer Device     Mean Standard_Deviation Correlation
## 1         1      1 54.26610           16.76982 -0.06412835
## 2         2      1 54.26873           16.76924 -0.06858639
## 3         3      1 54.26732           16.76001 -0.06834336
## 4         4      1 54.26327           16.76514 -0.06447185
## 5         5      1 54.26030           16.76774 -0.06034144
## 6         6      1 54.26144           16.76590 -0.06171484
## 7         7      1 54.26881           16.76670 -0.06850422
## 8         8      1 54.26785           16.76676 -0.06897974
## 9         9      1 54.26588           16.76885 -0.06860921
## 10       10      1 54.26734           16.76896 -0.06296110
## 11       11      1 54.26993           16.76996 -0.06944557
## 12       12      1 54.26692           16.77000 -0.06657523
## 13       13      1 54.26015           16.76996 -0.06558334
## 14        1      2 47.83472           26.93974 -0.06412835
## 15        2      2 47.83082           26.93573 -0.06858639
## 16        3      2 47.83772           26.93004 -0.06834336
## 17        4      2 47.83225           26.93540 -0.06447185
## 18        5      2 47.83983           26.93019 -0.06034144
## 19        6      2 47.83025           26.93988 -0.06171484
## 20        7      2 47.83545           26.94000 -0.06850422
## 21        8      2 47.83590           26.93610 -0.06897974
## 22        9      2 47.83150           26.93861 -0.06860921
## 23       10      2 47.83955           26.93027 -0.06296110
## 24       11      2 47.83699           26.93768 -0.06944557
## 25       12      2 47.83160           26.93790 -0.06657523
## 26       13      2 47.83972           26.93000 -0.06558334
```

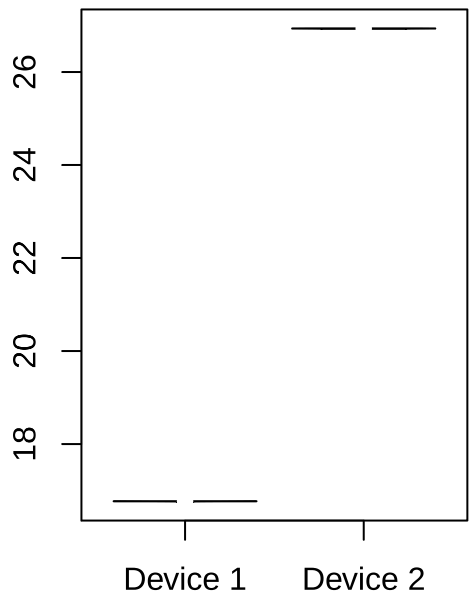
```r
# Make Boxplot of means of each device type
boxplot(Mean~Device, data = finalTable, main="Means of Device 1 and 2",
        ylab="Device", xlab="Means", horizontal = TRUE)
```



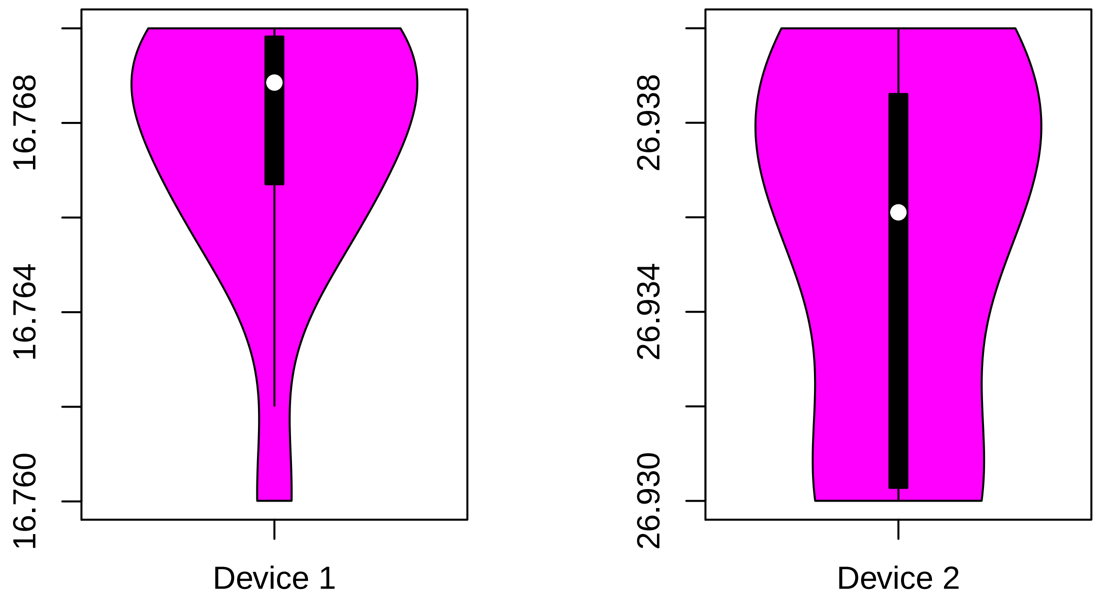

```
par(mfrow = c(1,2))
boxplot(finalTable$Mean[finalTable$Device == 1], horizontal = TRUE)
boxplot(finalTable$Mean[finalTable$Device == 2], horizontal = TRUE)
```




```r
# Make Violin Plot of Standard Deviations
x1 <- finalTable$Standard_Deviation[finalTable$Device == 1]
x2 <- finalTable$Standard_Deviation[finalTable$Device == 2]
vioplot(x1, x2, names = c("Device 1", "Device 2"))
par(mfrow = c(1, 2))
```




```
vioplot(x1, names = "Device 1")
vioplot(x2, names = "Device 2")
```



Problem 7

```r
# Import Data
bloodPressureData <- fread("http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/BloodPressure.dat")

# Change column names so Day is not replicated
bpdNames <- colnames(bloodPressureData)
bpdNames[5]<- "Day Copy"
colnames(bloodPressureData) <- bpdNames

# Remove the second "Day" column since it is a repeat of the first
bloodPressureData <- select(bloodPressureData, Day:Dev3, Doc1:Doc3)

# Gather doctor and device columns so reader (which device or doctor took the reading)
# is a variable and all readings are in one column
bpd <- gather(bloodPressureData, key = "Reader", value = "Blood_Pressure", Dev1:Doc3)

# Print cleaned data
bpd
```

```
##    Day Reader Blood_Pressure
## 1    1   Dev1         133.34
## 2    2   Dev1         110.94
## 3    3   Dev1         118.54
## 4    4   Dev1         137.94
```

```
## 5    5   Dev1          139.52
## 6    6   Dev1          139.23
## 7    7   Dev1          117.96
## 8    8   Dev1          119.59
## 9    9   Dev1          116.12
## 10  10   Dev1          128.38
## 11  11   Dev1          125.17
## 12  12   Dev1          134.62
## 13  13   Dev1          136.14
## 14  14   Dev1          131.21
## 15  15   Dev1          132.51
## 16   1   Dev2          133.36
## 17   2   Dev2          110.85
## 18   3   Dev2          118.56
## 19   4   Dev2          137.80
## 20   5   Dev2          139.62
## 21   6   Dev2          139.11
## 22   7   Dev2          117.81
## 23   8   Dev2          119.42
## 24   9   Dev2          116.00
## 25  10   Dev2          128.48
## 26  11   Dev2          125.25
## 27  12   Dev2          134.41
## 28  13   Dev2          136.07
## 29  14   Dev2          131.03
## 30  15   Dev2          132.86
## 31   1   Dev3          133.45
## 32   2   Dev3          110.92
## 33   3   Dev3          118.67
## 34   4   Dev3          137.77
## 35   5   Dev3          139.59
## 36   6   Dev3          139.36
## 37   7   Dev3          117.85
## 38   8   Dev3          119.48
## 39   9   Dev3          115.93
## 40  10   Dev3          128.41
## 41  11   Dev3          125.34
## 42  12   Dev3          134.55
## 43  13   Dev3          136.22
## 44  14   Dev3          130.96
## 45  15   Dev3          132.65
## 46   1   Doc1          126.54
## 47   2   Doc1          124.69
## 48   3   Doc1          125.46
## 49   4   Doc1          125.95
## 50   5   Doc1          125.90
## 51   6   Doc1          127.85
## 52   7   Doc1          125.55
## 53   8   Doc1          125.80
## 54   9   Doc1          125.11
## 55  10   Doc1          125.75
## 56  11   Doc1          128.77
## 57  12   Doc1          125.26
## 58  13   Doc1          126.26
```

```
## 59  14   Doc1          125.68
## 60  15   Doc1          124.47
## 61   1   Doc2          127.36
## 62   2   Doc2          128.86
## 63   3   Doc2          129.43
## 64   4   Doc2          130.72
## 65   5   Doc2          130.13
## 66   6   Doc2          132.03
## 67   7   Doc2          132.05
## 68   8   Doc2          129.87
## 69   9   Doc2          128.09
## 70  10   Doc2          131.94
## 71  11   Doc2          130.05
## 72  12   Doc2          131.13
## 73  13   Doc2          130.91
## 74  14   Doc2          128.83
## 75  15   Doc2          129.46
## 76   1   Doc3          131.88
## 77   2   Doc3          132.39
## 78   3   Doc3          134.43
## 79   4   Doc3          134.28
## 80   5   Doc3          134.44
## 81   6   Doc3          137.37
## 82   7   Doc3          132.17
## 83   8   Doc3          134.97
## 84   9   Doc3          133.97
## 85  10   Doc3          132.68
## 86  11   Doc3          134.75
## 87  12   Doc3          134.29
## 88  13   Doc3          133.38
## 89  14   Doc3          135.67
## 90  15   Doc3          134.39
```

Problem 8

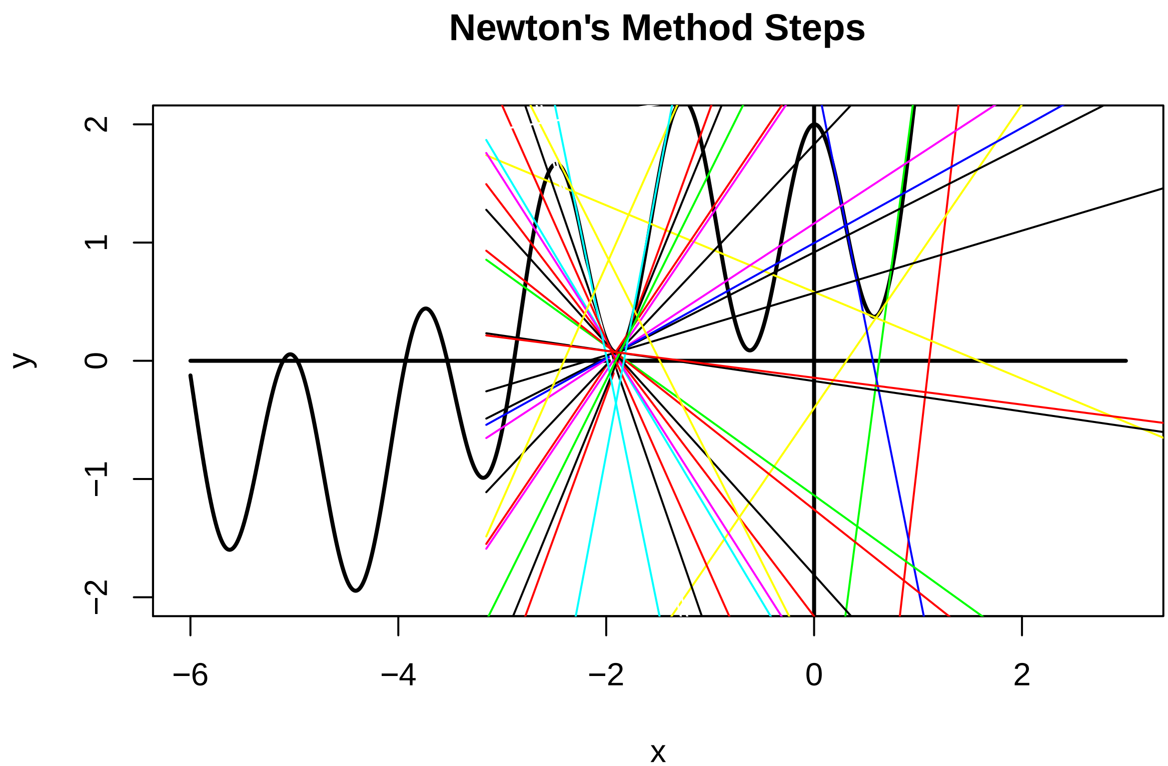
```
# Make a function to use that evaluates a function given a value
evaluate <- function(func, val){
  func(val)
}
# Create a function that stores the function we want to feed into Newton's method
myfunc <- function(x){
  3^x-sin(x)+cos(5*x)
}

# Create new function, accepts an interval where the initial guess is, the function, the threshold at
# which the function stops and how many digits it should round values contained in the vector storing a
newtonsMethod <- function(inta, intb, func, finish = .0001, check = FALSE, roundvals = 4){

  # Take derivative of function to use in the loop below
  derivative <- (Deriv(func, "x"))
 # Sample a random start point on the interval specified in the function call
  xn <- runif(1, inta, intb)
  # Initialize vectors that are going to have to be used outside of the loop later in the function,
```

```r
# these store each value generated in the while loop for the table and graph.
functionVals <- vector()
derivVals <- vector()
xvals <- vector()
iterations <- vector()
estimates <- vector()
tolerance <- vector()
# Values that will be x values in the graph later
z <- seq(xn-5, xn+5, .01)
# Initialize a counter to use to count how many steps it requires
# to reach an estimate and induce a break point if necessary
counter <- 1
while (check == FALSE){
  # Vector containing which number step we are on
  iterations <- c(iterations, counter)

  # Create numerator and store it in a vector of numerator values
  num <- func(xn)
  functionVals <- c(functionVals, round(num, roundvals))

  # Create denominator and store it in a vector of denominator values
  denom <- evaluate(derivative, xn)
  derivVals <- c(derivVals, round(denom, roundvals))

  # Store current value of xn
  xvals <- c(xvals, round(xn, roundvals))

  # Create xn+1 and store it in a vector
  xnext <- (xn - (num/denom))
  estimates <- c(estimates, round(xnext, roundvals))

  # Check to see if xn and xn+1 are close enough together to conclude that
  # we have reached a sufficient estimatio of the zero
  check <- (abs(xnext-xn)) <= finish

  # xn+1 becomes xn
  xn <- xnext
  counter <- counter + 1
  tolerance <- c(tolerance, finish)

  # Induce break point if the loop has run 100 times without reaching an
  # estimate with the correct amount of precision
  if(counter == 100){
    check = TRUE
  }

}

# Get a measure of how good our estimation is to display in the output
error <- func(xnext)

# Create table with all the vectors created in the loop and label accordingly
outTable <- cbind(iterations, xvals, functionVals, derivVals, estimates, tolerance)
```

```r
  colnames(outTable) <- c("Count", "Xn", "f(x)", "f'(x)", "Xn+1", "Tolerance")

  # Create rows that will summarize the answer, tolerance used and error and bind them to the table
  ansrow <- c("Answer:", round(xnext, 7), " ", " ", " ", " ")
  tolrow <- c("Tolerance", finish, " ", " ", " ", " ")
  errrow <- c("Error:", round(error, 7),  " ", " ", " ", " ")
  outTable <- rbind(outTable, ansrow, tolrow, errrow)

  # Remove row names and make table nicer
  rownames(outTable) <- NULL
  outTable<-kable(outTable, "markdown", digits = getOption("4"), align = "c")

  # Create a data frame to hold the values used to graph the tangent line at each estimation
  y <- data.frame(matrix(nrow = length(z), ncol = counter))
  #y[,1] <- functionVals[1]+derivVals[1]*z-derivVals[1]*xvals[1]

  # Create a set of x values to use to make "smooth" curves of the actual function
  xplot <- seq(-6, 3, .001)
  myFuncVals <- func(xplot)
  # Plot the function and make the axes a little easier to see
  plot(xplot, rep.int(0, length(xplot)), type = "l", lwd = 2, col = "black", ylim = c(-2,2),
       xlab = "x", ylab = "y", main = "Newton's Method Steps")
  lines(rep.int(0, length(xplot)), xplot , type = "l", lwd = 2)
  lines(xplot, myFuncVals, lwd = 2)

  # For each row entry in the data matrix, find the corresponding tangent line and graph it
  for(i in 1:counter){
    y[,i] <- functionVals[i]+derivVals[i]*z-derivVals[i]*xvals[i]
    lines(z, y[,i], col = rgb(sample(0:1, 1), sample(0:1, 1), sample(0:1,1)))
  }

  # Print the table
  print(outTable)

}
```

```r
# Run the newtons method function
newtonsMethod(-2,2, myfunc, finish = .01)
```

```
## Warning in rep(digits, length.out = m): 'x' is NULL so the result will be
## NULL
```



Count	Xn	f(x)	f'(x)	Xn+1	Tolerance
1	1.846	5.6556	7.6518	1.1069	0.01
2	1.1069	3.2119	6.6629	0.6248	0.01
3	0.6248	0.4018	1.2837	0.3118	0.01
4	0.3118	1.1135	-4.404	0.5647	0.01
5	0.5647	0.3747	-0.3668	1.5861	0.01
6	1.5861	4.6353	1.3047	-1.9666	0.01
7	-1.9666	0.1201	-1.4724	-1.885	0.01
8	-1.885	0.0771	0.446	-2.0579	0.01
9	-2.0579	0.3391	-3.2221	-1.9527	0.01
10	-1.9527	0.1018	-1.1591	-1.8649	0.01
11	-1.8649	0.091	0.9329	-1.9624	0.01
12	-1.9624	0.1142	-1.3796	-1.8797	0.01
13	-1.8797	0.0798	0.5755	-2.0184	0.01
14	-2.0184	0.2249	-2.5418	-1.9299	0.01
15	-1.9299	0.0814	-0.632	-1.8012	0.01
16	-1.8012	0.1983	2.4143	-1.8833	0.01
17	-1.8833	0.0779	0.4876	-2.0431	0.01
18	-2.0431	0.2932	-2.983	-1.9448	0.01
19	-1.9448	0.0934	-0.9793	-1.8495	0.01
20	-1.8495	0.1082	1.3017	-1.9326	0.01
21	-1.9326	0.0832	-0.6945	-1.8129	0.01
22	-1.8129	0.1716	2.1531	-1.8926	0.01

```
## |    23     |   -1.8926   | 0.0744  | 0.2637  | -2.1749  |    0.01    |
## |    24     |   -2.1749   | 0.7938  | -4.2946 |  -1.99   |    0.01    |
## |    25     |    -1.99    | 0.1606  | -1.9773 | -1.9088  |    0.01    |
## |    26     |   -1.9088   | 0.0733  | -0.1283 | -1.3374  |    0.01    |
## |    27     |   -1.3374   | 2.1225  | 1.9864  | -2.4059  |    0.01    |
## |    28     |   -2.4059   | 1.6016  | -1.737  | -1.4839  |    0.01    |
## |    29     |   -1.4839   | 1.6131  | 4.6636  | -1.8298  |    0.01    |
## |    30     |   -1.8298   | 0.1384  | 1.7653  | -1.9082  |    0.01    |
## |    31     |   -1.9082   | 0.0733  | -0.1136 | -1.2633  |    0.01    |
## |    32     |   -1.2633   | 2.2021  | 0.1391  | -17.0991 |    0.01    |
## |    33     |  -17.0991   | -1.7661 | -2.9362 | -17.7006 |    0.01    |
## |    34     |  -17.7006   | -0.0538 | 2.1547  | -17.6756 |    0.01    |
## |    35     |  -17.6756   | -0.0066 | 1.6227  | -17.6716 |    0.01    |
## |  Answer:  | -17.6715766 |         |         |          |            |
## | Tolerance |    0.01     |         |         |          |            |
## |  Error:   | -0.0001807  |         |         |          |            |
```