STAT 5014 HW 4

David Edwards

9/24/2019

Problem 4

Most of these suggestions I was already familiar with and will strive to use. I know in the past I haven't always been the best about remembering to use spaces or line up arguments that spread over multiple lines. I'll working on these things when scripting, specifically when it will be read by someone else.

Problem 5

Here is my code trying out the lint function.

```r
library(lintr)
lint(filename = "./HW3_Edwards.Rmd")
```

```
## ./HW3_Edwards.Rmd:35:1: style: lines should not be more than 80 characters.
## data <- fread(url, fill = TRUE, sep = " ",  skip = 2, col.names = c("Item", "Op1", "Op2",
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:36:1: style: lines should not be more than 80 characters.
##                                                                     "Op3","Op4","Op5"))
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:36:75: style: Commas should always have a space after.
##                                                                     "Op3","Op4","Op5"))
##                                                                           ^
## ./HW3_Edwards.Rmd:36:81: style: Commas should always have a space after.
##                                                                     "Op3","Op4","Op5"))
##                                                                                 ^
## ./HW3_Edwards.Rmd:37:3: style: Put spaces around all infix operators.
## j <-1
##   ^~~
## ./HW3_Edwards.Rmd:38:17: style: Commas should always have a space after.
## for (i in seq(1,dim(data)[1], 3)){
##                 ^
## ./HW3_Edwards.Rmd:39:9: style: Put spaces around all infix operators.
##   data[i+1] = c(j, data[i+1,1:5])
##        ~^~
## ./HW3_Edwards.Rmd:39:13: style: Use <-, not =, for assignment.
##   data[i+1] = c(j, data[i+1,1:5])
##             ^
## ./HW3_Edwards.Rmd:39:26: style: Put spaces around all infix operators.
##   data[i+1] = c(j, data[i+1,1:5])
##                         ~^~
## ./HW3_Edwards.Rmd:39:29: style: Commas should always have a space after.
##   data[i+1] = c(j, data[i+1,1:5])
##                             ^
## ./HW3_Edwards.Rmd:40:9: style: Put spaces around all infix operators.
##   data[i+2] = c(j, data[i+2,1:5])
##        ~^~
## ./HW3_Edwards.Rmd:40:13: style: Use <-, not =, for assignment.
##   data[i+2] = c(j, data[i+2,1:5])
```

```
##             ^
## ./HW3_Edwards.Rmd:40:26: style: Put spaces around all infix operators.
##   data[i+2] = c(j, data[i+2,1:5])
##                         ~^~
## ./HW3_Edwards.Rmd:40:29: style: Commas should always have a space after.
##   data[i+2] = c(j, data[i+2,1:5])
##                             ^
## ./HW3_Edwards.Rmd:41:4: style: Put spaces around all infix operators.
##   j<- j+1
##   ~^
## ./HW3_Edwards.Rmd:41:8: style: Put spaces around all infix operators.
##   j<- j+1
##       ~^~
## ./HW3_Edwards.Rmd:54:26: style: Put spaces around all infix operators.
## mydf <- data.frame("Year"=c(data$V1,data$V3,data$V5,data$V7[!is.na(data$V7)]),
##                         ~^~
## ./HW3_Edwards.Rmd:54:37: style: Commas should always have a space after.
## mydf <- data.frame("Year"=c(data$V1,data$V3,data$V5,data$V7[!is.na(data$V7)]),
##                                     ^
## ./HW3_Edwards.Rmd:54:45: style: Commas should always have a space after.
## mydf <- data.frame("Year"=c(data$V1,data$V3,data$V5,data$V7[!is.na(data$V7)]),
##                                             ^
## ./HW3_Edwards.Rmd:54:53: style: Commas should always have a space after.
## mydf <- data.frame("Year"=c(data$V1,data$V3,data$V5,data$V7[!is.na(data$V7)]),
##                                                     ^
## ./HW3_Edwards.Rmd:55:1: style: lines should not be more than 80 characters.
##                    "LongJump"=c(data$V2, data$V4, data$V6, data$V8[!is.na(data$V8)]))
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:55:30: style: Put spaces around all infix operators.
##                    "LongJump"=c(data$V2, data$V4, data$V6, data$V8[!is.na(data$V8)]))
##                             ~^~
## ./HW3_Edwards.Rmd:65:1: style: lines should not be more than 80 characters.
## url <- "http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/BrainandBodyWeight.dat"
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:67:28: style: Put spaces around all infix operators.
## mydf <- data.frame("BodyWt"=c(data$V1,data$V3,data$V5[!is.na(data$V5)]),
##                           ~^~
## ./HW3_Edwards.Rmd:67:39: style: Commas should always have a space after.
## mydf <- data.frame("BodyWt"=c(data$V1,data$V3,data$V5[!is.na(data$V5)]),
##                                       ^
## ./HW3_Edwards.Rmd:67:47: style: Commas should always have a space after.
## mydf <- data.frame("BodyWt"=c(data$V1,data$V3,data$V5[!is.na(data$V5)]),
##                                               ^
## ./HW3_Edwards.Rmd:68:29: style: Put spaces around all infix operators.
##                    "BrainWt"=c(data$V2, data$V4, data$V6[!is.na(data$V6)]))
##                            ~^~
## ./HW3_Edwards.Rmd:79:1: style: lines should not be more than 80 characters.
## data <- fread(url, fill = TRUE, header = T, skip = 1, col.names = c("Treatment", "10000","20000","30
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:79:90: style: Commas should always have a space after.
## data <- fread(url, fill = TRUE, header = T, skip = 1, col.names = c("Treatment", "10000","20000","30
##                                                                                          ^
## ./HW3_Edwards.Rmd:79:98: style: Commas should always have a space after.
## data <- fread(url, fill = TRUE, header = T, skip = 1, col.names = c("Treatment", "10000","20000","30
```

```
##                                                                                                  ^
## ./HW3_Edwards.Rmd:80:26: style: Commas should always have a space after.
## mydf <- bind_rows(data[1,],data[2,])
##                           ^
## ./HW3_Edwards.Rmd:80:28: style: Commas should always have a space after.
## mydf <- bind_rows(data[1,],data[2,])
##                             ^
## ./HW3_Edwards.Rmd:80:35: style: Commas should always have a space after.
## mydf <- bind_rows(data[1,],data[2,])
##                                    ^
## ./HW3_Edwards.Rmd:81:1: style: lines should not be more than 80 characters.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
## ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## ./HW3_Edwards.Rmd:81:32: style: Commas should always have a space after.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                 ^
## ./HW3_Edwards.Rmd:81:39: style: Commas should always have a space after.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                        ^
## ./HW3_Edwards.Rmd:81:50: style: Put spaces around all infix operators.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                                 ~^
## ./HW3_Edwards.Rmd:81:70: style: Commas should always have a space after.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                                                      ^
## ./HW3_Edwards.Rmd:81:77: style: Commas should always have a space after.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                                                             ^
## ./HW3_Edwards.Rmd:81:82: style: Commas should always have a space after.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                                                                  ^
## ./HW3_Edwards.Rmd:81:100: style: Put spaces around all infix operators.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##                                                                                                    ^
## ./HW3_Edwards.Rmd:81:129: style: Place a space before left parenthesis, except in a function call.
## mydf <- mydf %>% gather(Levels,Values,-Treatment)%>% separate(Values,c("V1","V2","V3"), sep = ",") %
##
## ./HW3_Edwards.Rmd:83:1: style: Trailing blank lines are superfluous.
##
## ^
```

I ended up with several "errors" most of whcih deal with spacing.

Problem 6

Below is my script for the funciton to compute the means, SDs and corrilation between the two devices. I have also include the code for the boxplot and the violin plot that the problem asks for.

```r
#Function that takes in a data frame with data on two different types of devices and returns
#summary data on the data for each device
SummaryOfDevs <- function(data){
  Summary <- data.frame(dev1Mean = mean(data$dev1),
                        dev2Mean = mean(data$dev2),
                        dev1Sd   = sd(data$dev1),
                        dev2Sd   = sd(data$dev2),
```

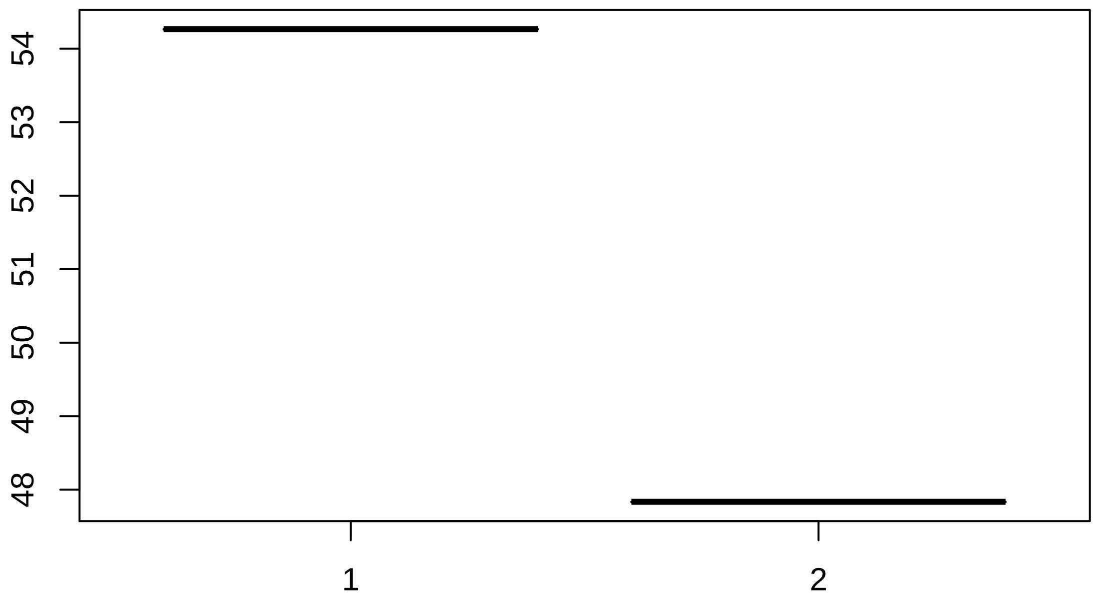
```
                        dev1Cordev2 = cor(data$dev1,data$dev2))
  return(Summary)

}
#Read in the data from the homework file
HW4_data <- readRDS("~/STAT_5014_Fall_2019/homework/HW4_data.rds")


#Initalize the table with the first row from Observer 1
tableOfObservers <- SummaryOfDevs(HW4_data[HW4_data$Observer == 1, ])

#Add in the rows for the rest of the Observers, 2-13
for(i in 2:13){
  tableOfObservers <- rbind(tableOfObservers, SummaryOfDevs(HW4_data[HW4_data$Observer == i, ]))
}

#Display the results
tableOfObservers
```

```
##    dev1Mean dev2Mean   dev1Sd   dev2Sd dev1Cordev2
## 1  54.26610 47.83472 16.76982 26.93974 -0.06412835
## 2  54.26873 47.83082 16.76924 26.93573 -0.06858639
## 3  54.26732 47.83772 16.76001 26.93004 -0.06834336
## 4  54.26327 47.83225 16.76514 26.93540 -0.06447185
## 5  54.26030 47.83983 16.76774 26.93019 -0.06034144
## 6  54.26144 47.83025 16.76590 26.93988 -0.06171484
## 7  54.26881 47.83545 16.76670 26.94000 -0.06850422
## 8  54.26785 47.83590 16.76676 26.93610 -0.06897974
## 9  54.26588 47.83150 16.76885 26.93861 -0.06860921
## 10 54.26734 47.83955 16.76896 26.93027 -0.06296110
## 11 54.26993 47.83699 16.76996 26.93768 -0.06944557
## 12 54.26692 47.83160 16.77000 26.93790 -0.06657523
## 13 54.26015 47.83972 16.76996 26.93000 -0.06558334
```

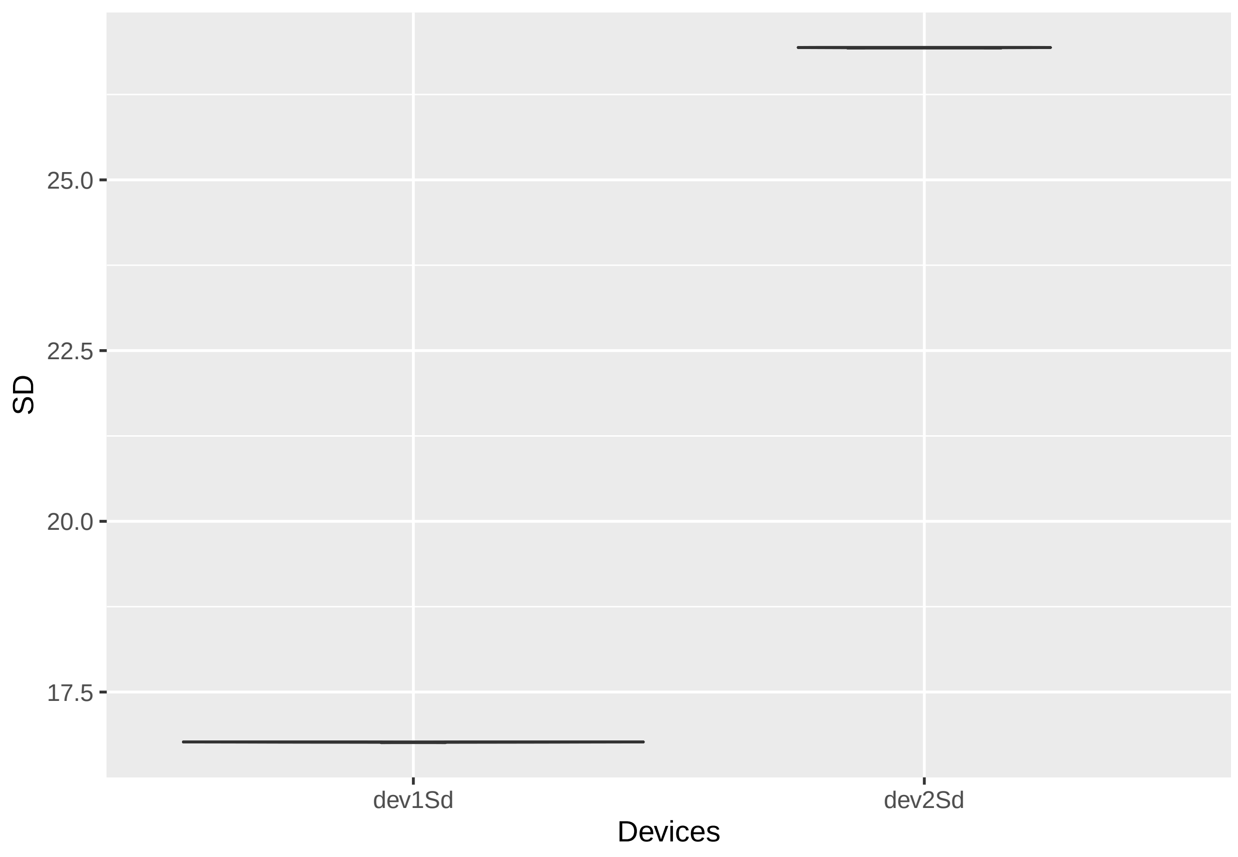
```
boxplot(tableOfObservers$dev1Mean, tableOfObservers$dev2Mean)

library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```



```
library(tidyr)

tableOfSD <- tableOfObservers %>%
             select(dev1Sd, dev2Sd) %>%
             gather("Devices", "SD")
ggplot(tableOfSD, aes(x = Devices, y = SD)) + geom_violin()
```



The boxplot and the violin plot show almost no variation of the mean and SD for each device when we compare these values among observers.

Problem 7

Below is my code for determining the integral using the left Reimann sum, which in this case will be an over estimate as the function is decreasing on this interval. The frist function is called and can take in vector of different widths. The second function is my helper function that then computes the estimate of the integral given the widths. I think this makes the code easier to read and follow.

```r
ReimannSum_widths <- function(widths = c(0.1, .001)){
  sums <- rep(NA, length(widths))
  for(i in 1:length(widths)){
    sums[i] <- ReimannSum_e(widths[i])
  }
  return(data.frame("Widths" = widths, "Integral Approx." = sums))
}

ReimannSum_e <- function(width){
  x <- seq(from = 0, to = 1-width, length.out = ceiling(1/width))
  exponent <- (-1/2)*x*x
  expx <- exp(exponent)
  return(sum(expx)*width)
}

ReimannSum_widths(c(.1, .01, .001, .0001, .00001, .000001, .0000001))
```

```
##   Widths Integral.Approx.
## 1  1e-01        0.8747922
## 2  1e-02        0.8575867
## 3  1e-03        0.8558211
## 4  1e-04        0.8556441
## 5  1e-05        0.8556264
## 6  1e-06        0.8556246
## 7  1e-07        0.8556244
```

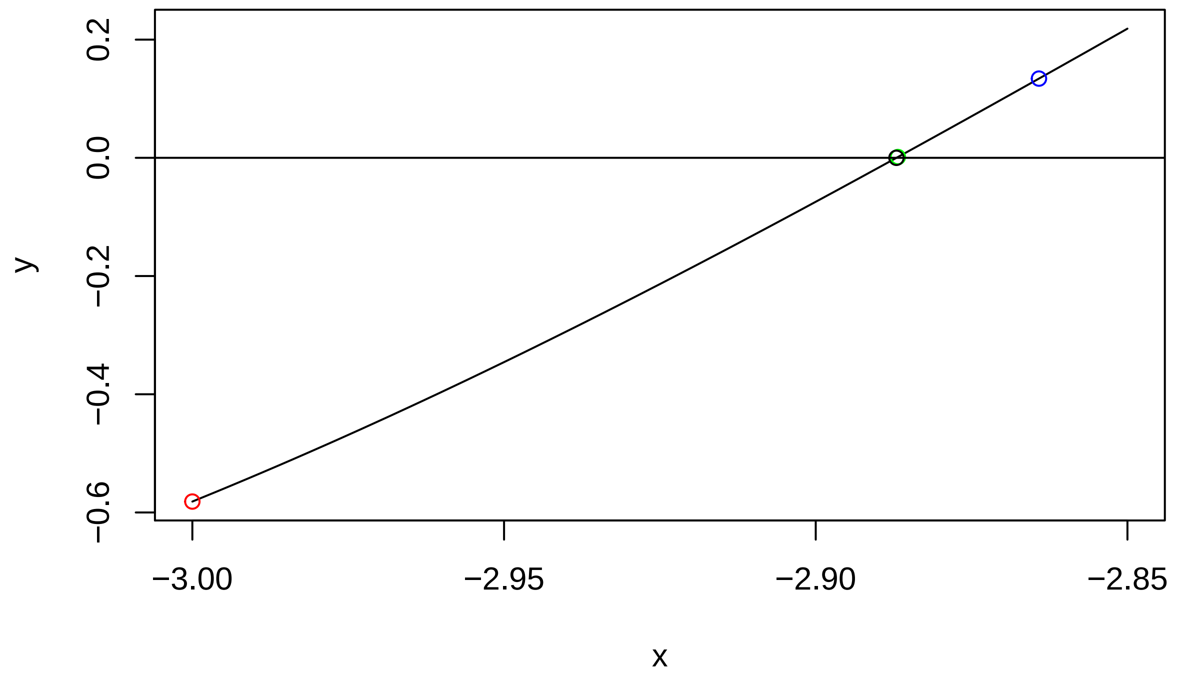
According to Wolframalpha.com the actual value of the integral is 0.855624 to six decimal points. To gain this level of precision we need to have our width be around 0.000001 or so which is also six significant figures.

Problem 8

Below is my code for Newton's Method to find the zeros of the function $f(x) = 3^x - sin(x) + cos(5x)$. The assignment said "The answer should include the solutions with tolerance used to terminate the loop, the interval used, and a plot showing the iterations on the path to the solution" but the way I set up the function it takes as an argument the amount of error that the final y-value can be from zero. Because of this I didn't have the function return the error. This seems like a more generic function to me and would allow the user to specify the amount of error. I also created two helper functions that will calculate the function and the derivative of the function to help make the code easier to read. Finally, I had the function return the sequence of x-values and y-values found on the way to the final soluiton which is given as the the last x-value.

```r
Newton <- function(InitialValue, error = 1e-5){
  yList <- y <- fOfX(InitialValue)
  xList <- InitialValue
  i <- 1
  while (abs(y) > error) {
    xOld <- xList[i]
    xNew <- xOld - fOfX(xOld)/fPrimeOfX(xOld)
    i <- i + 1
    xList[i] <- xNew
    yList[i] <- y <- fOfX(xNew)
  }

  return (data.frame("X Values" = xList, "Y Values" = yList))
}

fOfX <- function(x){
  return (3^x - sin(x) + cos(5*x))
}

fPrimeOfX <- function(x){
  return (3^x*log(3) - 5*sin(5*x) - cos(x))
}

x <- seq(from = -3, to = -2.85, by = 0.001)
y <- fOfX(x)

plot(x,y, type = 'l')
df <- Newton(-3)
points(df$X.Values[1], df$Y.Values[1], col = "red")
points(df$X.Values[2], df$Y.Values[2], col = "blue")
points(df$X.Values[3], df$Y.Values[3], col = "green")
points(df$X.Values[4], df$Y.Values[4], col = "black")
```

```r
abline(0,0)
```



As a plot I have created the function along with the points used to create the approimate answer. In the case above only 4 iterations were needed to get to a value that was within the error term. The points have been marked with different colors with black being the color of the solution point.