HW3

Tanner Glass

9/29/2020

Problem 3

In the lecture, there were two links to programming style guides. What is your takeaway from this and what specifically are *you* going to do to improve your coding style?

What i find for myself is that i typically throw statements all onto one line and don't separate my code via different lines. The programming style guides seem to help format the usage of different code styles by utilizing code on multiple lines where possible. For example, denoting for loops and function braces by separating the braces to their own line is a great way to identify the code chunk in total and make the function or for loop more understandable. It also help readers understand scope of variables as its easy to identify the code chunks.

Problem 4

Good programming practices start with this homework. In the last homework, you imported, munged, cleaned and summarized datasets from Wu and Hamada's *Experiments: Planning, Design and Analysis.*

Problem 5

A situation you may encounter is a data set where you need to create a summary statistic for each observation type. Sometimes, this type of redundancy is perfect for a function. Here, we need to create a single function which takes as input a two column dataframe and returns a vector containing

1. mean of column 1
2. mean of column 2
3. standard dev of column 1
4. standard dev of column 2
5. correlation between column 1 and 2

I will look at the code and comment on it, so make it NICE!!

We will use this function to summarize a dataset which has multiple repeated measurements from two devices (dev1 and dev2) by thirteen Observers. This file is preformatted as an R object, so it will read in nicely. "url <- https://github.com/rsettlage/STAT_5014_Fall_2020/blob/master/homework/HW3_data.rds". Please load the file (?readRDS – really nice format for storing data objects), loop through the Observers collecting the summary statistics via your function for each Observer separately and store in a single dataframe.

The output of this problem should be:

a. A single table of the means, sd, and correlation for each of the 13 Observers (*?kable*). From this table, what would you conclude? You can easily check your result using dplyr's group_by and summarize.

b. A box plot of dev, by Observer (*?boxplot*). From these plots, what would you conclude?

c. A violin plot of dev by Observer (*??violin* two "?" will search through installed packages). From these plots, what would you conclude? Compared to the boxplot and summary statistics, thoughts?

Now that you have made some conclusions and decided what your analysis may look like, you decide to make one more plot:

d. a scatter plot of the data using ggplot, geom_points, and add facet_wrap on Observer. For instance: `ggplot(df, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)`

What do you see? Combining the scatter plot with the summary statistics, what is the lesson here? As you approach data analysis, what things should you do in the "Exploratory Data Analysis" portion of a project to avoid embarrassment from making erroneos conclusions?

a.

We will begin by shaping the data into separated data frames by observers. And creating the requested function.

```
#Because we are using RStudio cloud, we upload the data into our environment directly by using the 'upl

#Use tidyverse to filter data in for loop to make for each observer
library(tidyr)
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
```

```
HW3_data<-readRDS("/cloud/project/HW3_data.rds")

for (i in 1:13){
  name <- paste("Observer", i, sep = "") #names each dataframa
  interm<-filter(HW3_data,Observer==i)
  interm<-select(interm,2:3)#Get the data frame in requested form
  assign(name,interm)

}
```

We will now create our function.

```
prob3fun<-function(df){
  mean1<- mean(df[,1])#find internal variables for each requested component
  mean2<- mean(df[,2])
  sd1<- sd(df[,1])
  sd2<- sd(df[,2])
  cor1<-cor(df[,1],df[,2])
  return(c(mean1,mean2,sd1,sd2,cor1)) #returns as vector
}
prob3_df<-rbind.data.frame(prob3fun(Observer1),prob3fun(Observer2),prob3fun(Observer3),prob3fun(Observe
prob3_df<-data.frame(1:13,prob3_df)
```

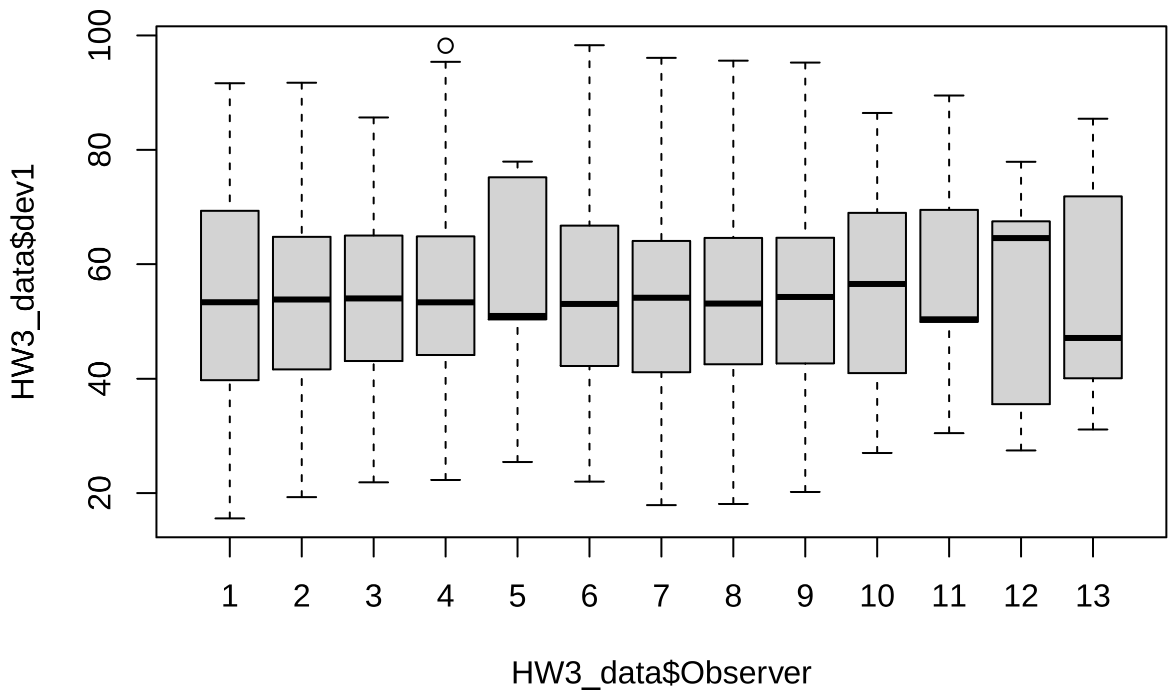
```r
names(prob3_df)<-c("Observer","mean1","mean2","sd1","sd2","correlation")
knitr::kable(prob3_df)
```

Observer	mean1	mean2	sd1	sd2	correlation
1	54.26610	47.83472	16.76983	26.93974	-0.0641284
2	54.26873	47.83082	16.76924	26.93573	-0.0685864
3	54.26732	47.83772	16.76001	26.93004	-0.0683434
4	54.26327	47.83225	16.76514	26.93540	-0.0644719
5	54.26030	47.83983	16.76774	26.93019	-0.0603414
6	54.26144	47.83025	16.76590	26.93988	-0.0617148
7	54.26881	47.83545	16.76670	26.94000	-0.0685042
8	54.26785	47.83590	16.76676	26.93610	-0.0689797
9	54.26588	47.83150	16.76885	26.93861	-0.0686092
10	54.26734	47.83955	16.76896	26.93027	-0.0629611
11	54.26993	47.83699	16.76996	26.93768	-0.0694456
12	54.26692	47.83160	16.77000	26.93790	-0.0665752
13	54.26015	47.83972	16.76996	26.93000	-0.0655833

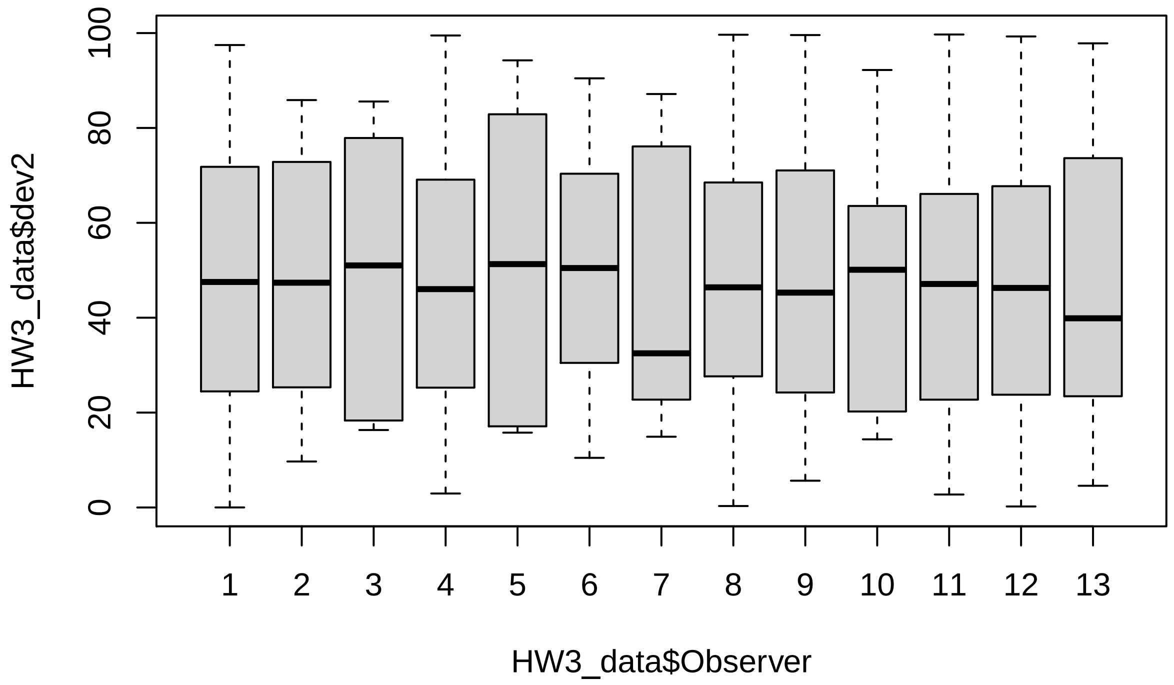
I can conclude that the data seems to have a similar mean and sd for each of the respective devices and they do not seem to correlate with each other based on the collective results from the observers.

b.

```r
boxplot(HW3_data$dev1~HW3_data$Observer)
```



```r
boxplot(HW3_data$dev2~HW3_data$Observer)
```

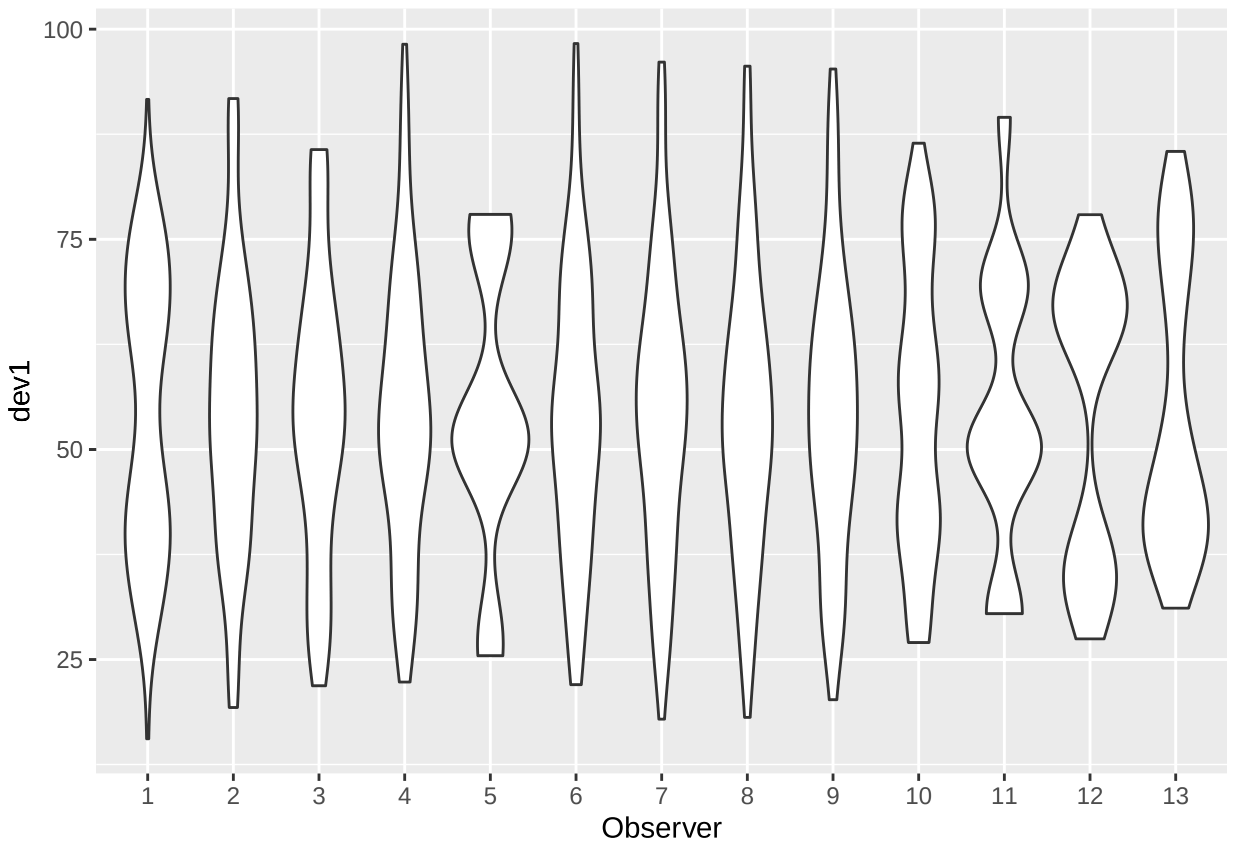


We see that the observers for each of the respective devices.

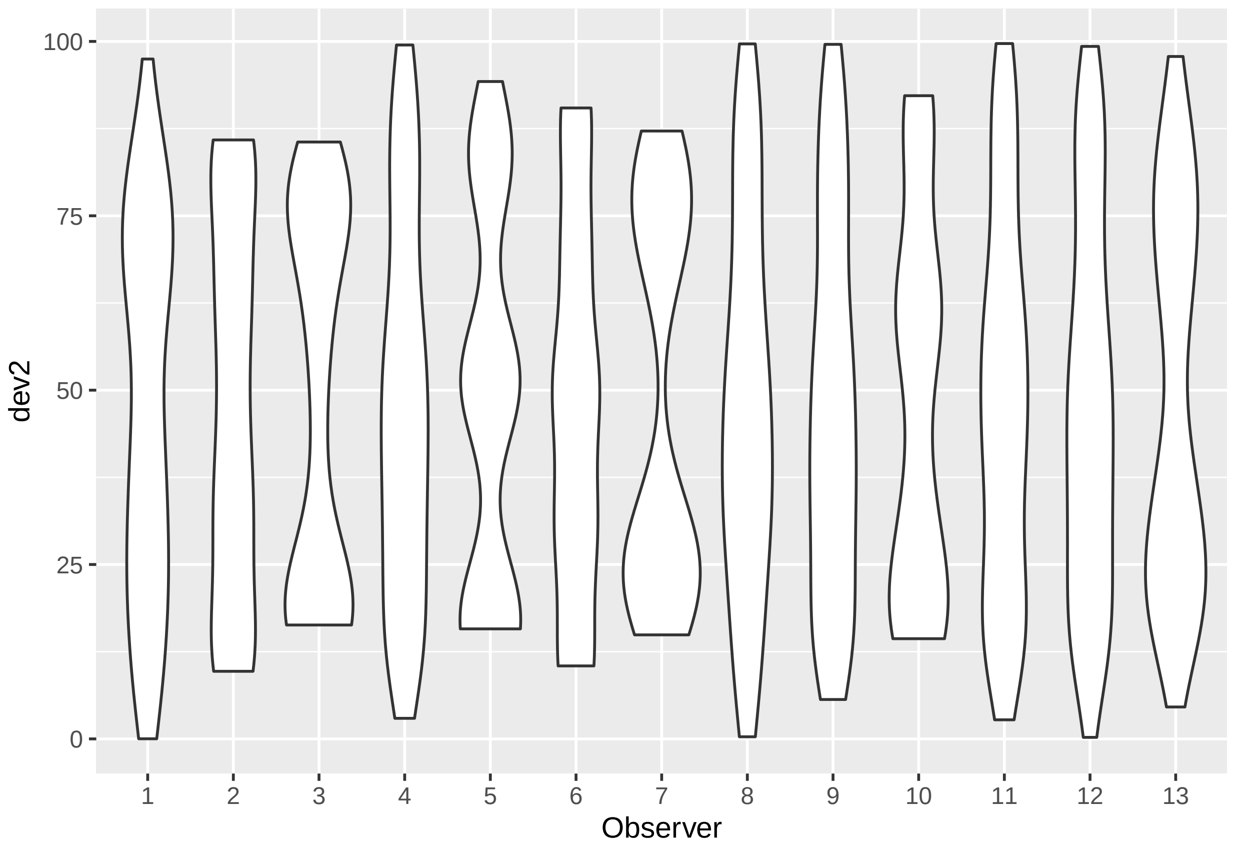
device 1 by observer seems to be consistent but now we see select observers are off on their mean and range. similarly for device 2.

c.

```
#install.packages("ggplot2")
library("ggplot2")
HW3_data$Observer<-as.factor(HW3_data$Observer)
ggplot(HW3_data, aes(x=Observer, y=dev1 )) +
  geom_violin()
```



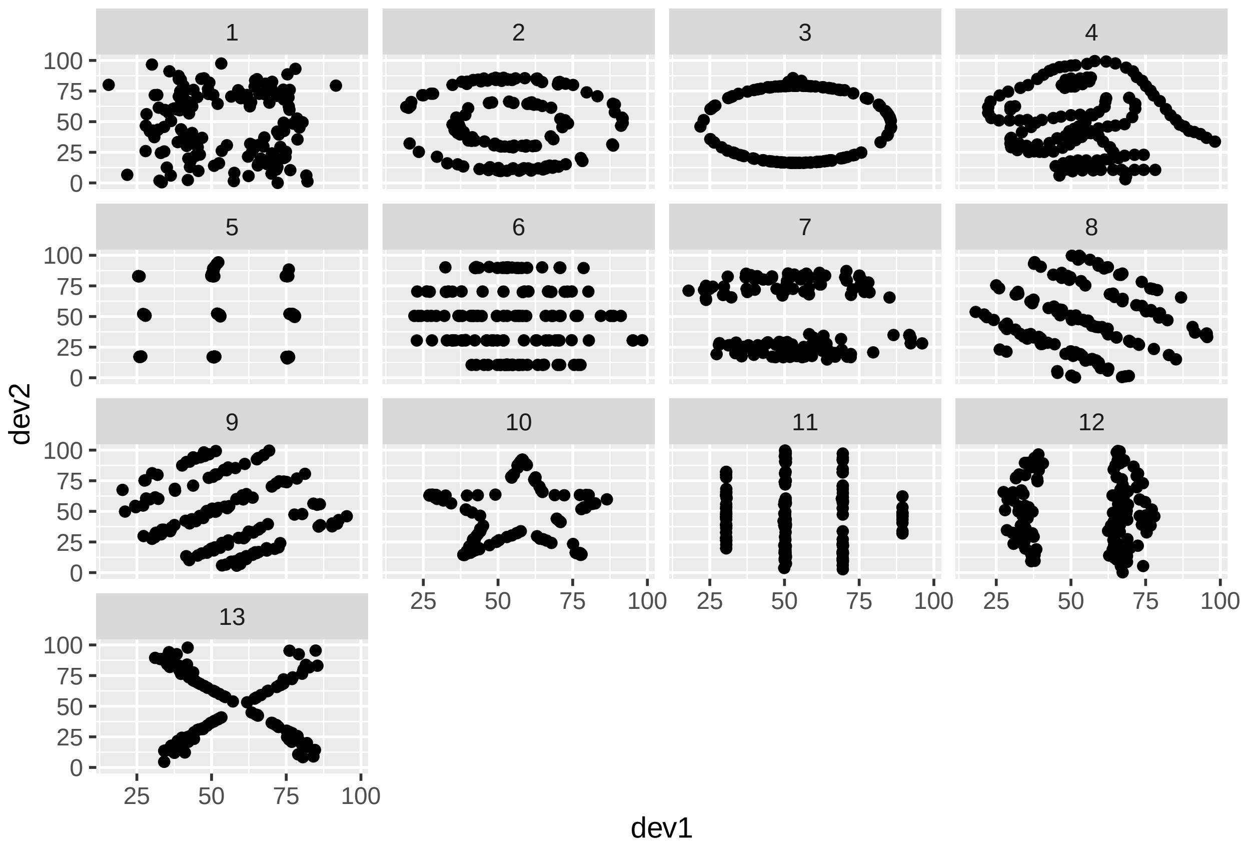
```r
ggplot(HW3_data, aes(x=Observer, y=dev2 )) +
  geom_violin()
```



From these plots we are able to shifts in variability and densities among all the observers for both device 1 and device 2 respectively.

d.

```
ggplot(HW3_data, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)
```



What I see is none of the data points observed are alike at all. The lesson here is data might convincing on one type of plot, but will not show the full picture.

Problem 6

Some numerical methods are perfect candidates for funtions. Create a function that uses Reimann sums to approximate the integral:

$$f(x) = \int_0^1 e^{-\frac{x^2}{2}}$$

The function should include as an argument the width of the slices used. Now use a looping construct (for or while) to loop through possible slice widths. Report the various slice widths used, the sum calculated, and the slice width necessary to obtain an answer within $1e^{-6}$ of the analytical solution.

```r
#Going with left riemann sum
riemann_fun<-function(width,f=exp(-(x^2/2))){
  x<-seq(-10,10,by= width) #it suffices to go from 10 since that is most of the mass.
  n<-length(x)
  if (x[n]==10){
    x<-x[-n]#removing the 'right' final endpoint for left R sum
  }
  ans<-(20/length(x))*sum(f)
  return(ans)

}
#We find the answer numerically to be approx 2.50662827463 could have been done in R
```

```
#but used separate integration computation.
width=10
while(abs(2.50662827463 - riemann_fun(width)) > 0.000001){

  width<-width-.01
}

width
```

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

We find that a width of 1 will satisfy as the width.

Problem 7

Create a function to find solutions to (1) using Newton's method. 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.

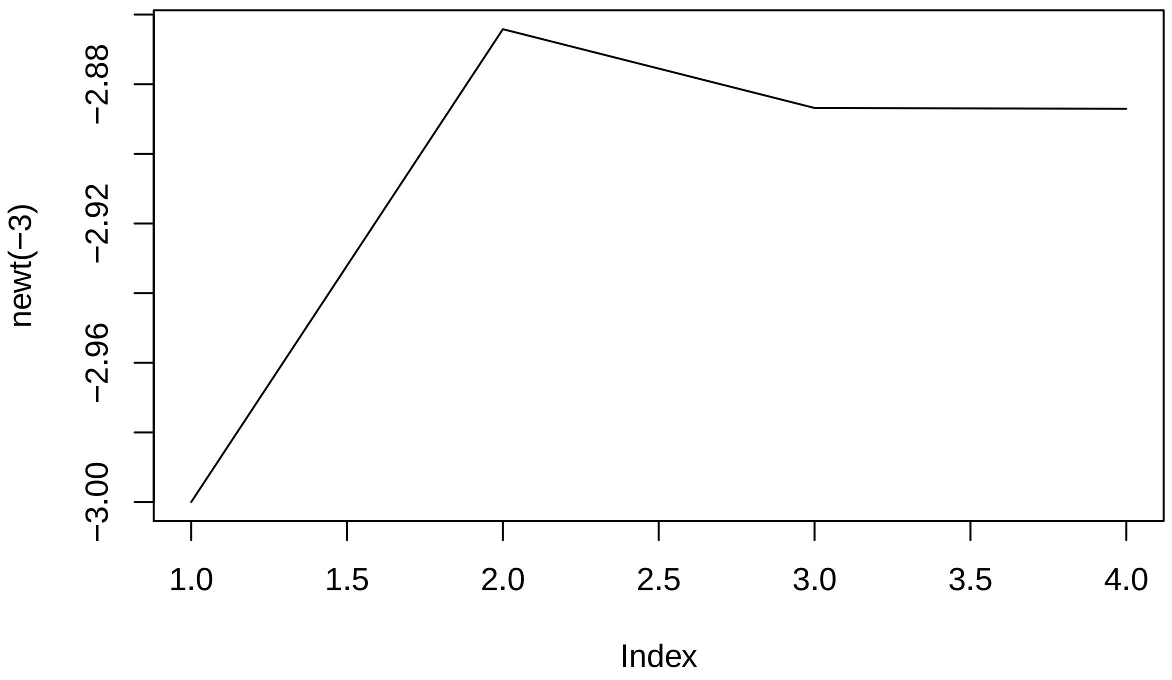
$$f(x) = 3^x - sin(x) + cos(5x) \tag{1}$$

The function is going to have infinite roots. we will devise a function that takes a 'guess' on which root you want to be close to and the computes the root. We will use .0001 as our error tolerance.

```
f<-function(x){
  ans2<-3^x-sin(x)+cos(5*x) #identifies our function
  return(ans2)
}
deriv_f<-function(x){#takes derivative value
  ans3<-3^x*log(3)-cos(x)-5*sin(5*x)
  return(ans3)
}
iteration<-function(guess){#The iteration step
  ans4<-guess-(f(guess)/deriv_f(guess))
  return(ans4)
}

newt<-function(guess){
  iterations<-c()#creates method with set bound of 0.0001
  while (abs(f(guess))>0.001){
    iterations<-c(iterations,guess)
    guess<-iteration(guess)
  }
  iterations<-c(iterations,guess)
  return(iterations)

}
plot(newt(-3),type='l')#gives us the plot of iterations
```



```r
newt(-3)[length(newt(-3))]#The last one will be our guess.
```

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

It works! as our intial guess, we guess 3 as we find the zero is in the interval (-3.5,-2.5) and we find that the value -2.887058 satifies as the root with a tolerance of .0001.

Problem 8

In most of your classes, you will be concerned with "sums of squares" of various flavors. SST = SSR + SSE for instance. Sums of square total (SST) = sums of square regression (SSR) + sums of square error (SSE). In this problem, we want to compare use of a for loop to using matrix operations in calculating these sums of squares. We will use data simulated using:

```r
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
```

Without going too far into the Linear Regression material, we want to calculate SST =

$$\Sigma_{i=1}^{100}(y_i - \bar{y})^2$$

Please calculate this using:

a. accumulating values in a for loop

b. matrix operations only

Note, you can precalculate mean(y) and create any vectors you need, ie perhaps a vector of 1 (ones). In both cases, wrap the calculation in the microbenchmark function. Report the final number and timings.

```r
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
```

```r
meany<-mean(y)#requested to calc these outside of time
J<-matrix(1,nrow=100,ncol=100)

start<-Sys.time()
accum<-0
for(i in 1:100){
  term<-(y[i]-meany)^2
  accum<-accum+term

}
accum #for loop method
```

```
## [1] 20582.27
```

```r
end<-Sys.time()



start1<-Sys.time()
t(y)%*%y-(1/100)*t(y)%*%J%*%y #matrix form
```

```
##          [,1]
## [1,] 20582.27
```

```r
end1<-Sys.time()
end-start #for loop timing
```

```
## Time difference of 0.008373976 secs
```

```r
end1-start1 #Matrix timing
```

```
## Time difference of 0.001973867 secs
```

We have found our desired SST above and both methods match. We were asked to look at the time for each of the methods and we find that the matrix method is faster than the for loop method.