Homework 3

Youngjin Cho

2020-09-29

For each assignment, turn in by the due date/time. Late assignments must be arranged prior to submission. In every case, assignments are to be typed neatly using proper English in Markdown.

The last couple of weeks, we spoke about R, version control and Reproducible Research, munging and 'tidying' data, good programming practice, some basic programming building blocs, and finally matrix/vector operations. In this homework, we will put this all together and actually analyze some data. Remember to adhere to both Reproducible Research and Good Programming Practices, ie describe what you are doing and comment/indent code where necessary.

Problem 1

In the "Getting and Cleaning Data" lesson set, you should be comfortable with lessons 1-3. Work through the "R Programming E" lesson as you see fit. Lessons 1-9 and 15 are ones you should consider paying special attention to. If you prefer the Rstudio.cloud Primers, the Primer on "Write Functions" is well done.

From the R command prompt:

```
library(swirl)
install_course("R_Programming_E")
install_course("Getting_and_Cleaning_Data")
install_course("Exploratory_Data_Analysis")
swirl()
```

I took these classes.

Problem 2

Create a new R Markdown file (file–>new–>R Markdown–>save as.

The filename should be: HW3_pid, i.e. for me it would be HW3_rsettlag

You will use this new R Markdown file to solve the following problems:

I made a file.

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?

I prefer Hadley Wickham's style guide. Using "_" in object is useful since it allows me to write a sentence as an object name. He also noted that file name should be meaningful. I agree to it since vague file names are hard to recognize when I look them few months after making them. Rules for spacing is kind of confusing

but it can be helpful for programming. I currently could not perfectly follow his style guide, but I will try to change my programming habit like that of Hadley Wickham.

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*.

Got it.

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?

```
hw3_data <- readRDS("/cloud/project/HW3_data.rds")
kable(head(hw3_data,10), caption="orignial data")
```

Table 1: orignial data

Observer	dev1	dev2
4	55.3846	97.1795

Observer	dev1	dev2
4	51.5385	96.0256
4	46.1538	94.4872
4	42.8205	91.4103
4	40.7692	88.3333
4	38.7179	84.8718
4	35.6410	79.8718
4	33.0769	77.5641
4	28.9744	74.4872
4	26.1538	71.4103

Imported original data.

(a)

```
problem_5_a_func <- function(input_data){
  colnames(input_data)=c("Observer", "dev1", "dev2")
  output_data=input_data %>%
    group_by(Observer) %>%
    summarize(dev_1_mean=mean(dev1),
              dev_2_mean=mean(dev2),
              dev_1_sd=sd(dev1),
              dev_2_sd=sd(dev2),
              dev_1_2_corr=cor(dev1,dev2))
  return(output_data)
}

problem_5_a <- problem_5_a_func(hw3_data)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

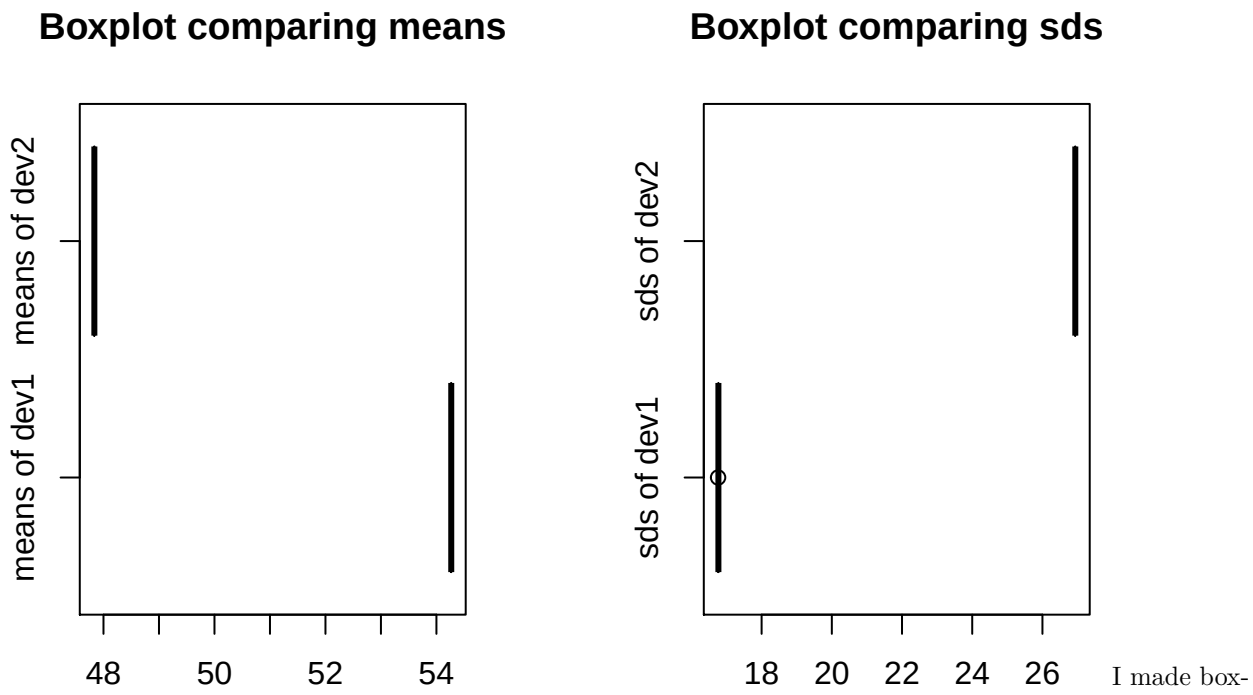
```
kable(problem_5_a)
```

Observer	dev_1_mean	dev_2_mean	dev_1_sd	dev_2_sd	dev_1_2_corr
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 made a function to make summary data frame from orignial data. It works very well. In summary, dev 1 and dev 2 observed by each observers look very similar.

(b)

```
par(mfrow=c(1,2))
boxplot(problem_5_a$dev_1_mean, problem_5_a$dev_2_mean,
        main="Boxplot comparing means",
        names=c("means of dev1", "means of dev2"),
        horizontal = TRUE
        )

boxplot(problem_5_a$dev_1_sd, problem_5_a$dev_2_sd,
        main="Boxplot comparing sds",
        names=c("sds of dev1", "sds of dev2"),
        horizontal = TRUE
        )
```



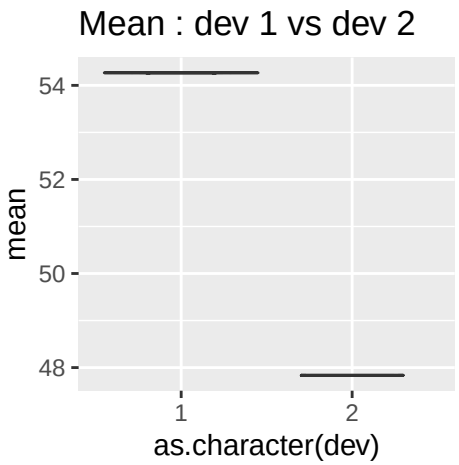
I made boxplots. It is clear that the means of dev2 are much smaller than those of dev1. Also, sds of dev2 are much larger than those of dev1. It means that the values of dev2 are usually smaller than those of dev1 and they have more higher variability than those of dev1.

(c)

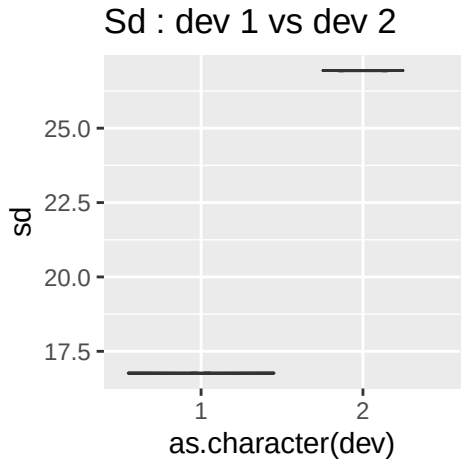
```
problem_5_c_mean <- gather(problem_5_a[,2:3],key="dev",value="mean",'dev_1_mean','dev_2_mean')
problem_5_c_mean <- problem_5_c_mean %>%
  mutate(dev = ifelse(dev == "dev_1_mean", 1, 2))

problem_5_c_sd <- gather(problem_5_a[,4:5],key="dev",value="sd",'dev_1_sd','dev_2_sd')
problem_5_c_sd <- problem_5_c_sd %>%
  mutate(dev = ifelse(dev == "dev_1_sd", 1, 2))
```

First, I prepared data for violin plot.

```
ggplot(problem_5_c_mean, aes(x=as.character(dev), y=mean)) +
  geom_violin() +
  ggtitle("Mean : dev 1 vs dev 2")
```



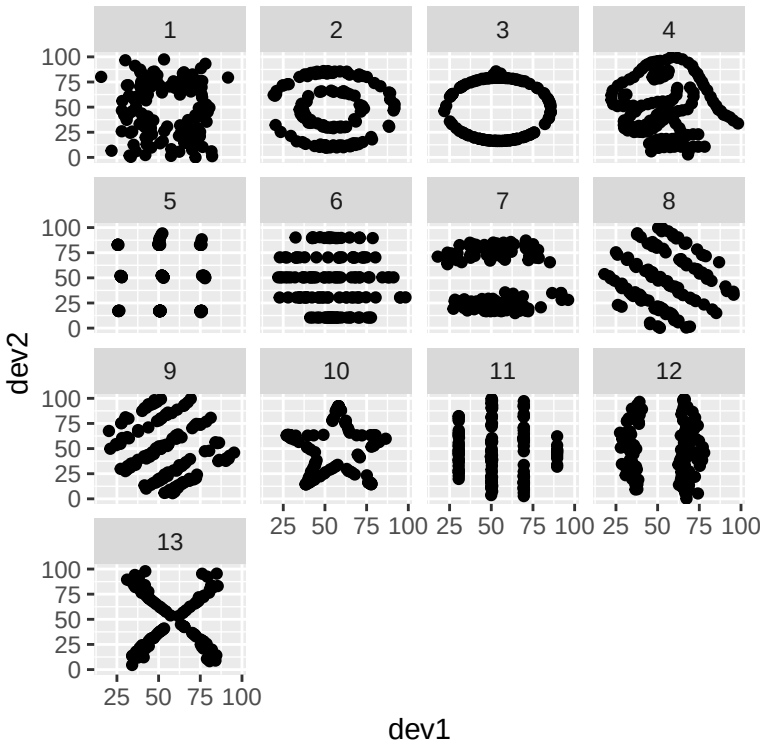

```
ggplot(problem_5_c_sd, aes(x=as.character(dev), y=sd)) +
  geom_violin() +
  ggtitle("Sd : dev 1 vs dev 2")
```



I made violin plots. the conclusion of (c) is same as that of (b). But the violin plot looks not very good since the values of dev1 and dev2 are so much different.

(d)

```
ggplot(hw3_data, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)
```



Before making scatter plots. I thought that data is weird since summary statistics are similar for each observers. But after I made scatter plots, I found that it is a point (x,y) data for some drawings! When doing explanatory data analysis, we should draw a scatter plot for orignal data before making summary statistics.

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.

Note: use good programming practices. For help on Reimann sums:
https://www.khanacademy.org/math/ap-calculus-ab/ab-integration-new/ab-6-2/a/left-and-right-riemann-sums

```
given_func_int <- function(x){
  exp(-x^2/2)
}
```

I made function for the given function. It is standard normal pdf without multiplying $\frac{1}{\sqrt{2\pi}}$Now I will integrate it from 0 to 1 by using my own Riemann integration function.

```
riemann_integral <- function(lb,ub,slice,fun){

  # Sliced the given range
  support <- seq(lb,ub,length=slice)
```

```
  # Made midpoint vector from range
  midpoint_vector <- (support[-length(support)]+support[-1])/2

  # Made width vector from range
  width_vector <- support[-1]-support[-length(support)]

  # Loop
  result <- 0
  for (i in 1:length(midpoint_vector)){
    result <- result+width_vector[i]*fun(midpoint_vector[i])
    i <- i+1
  }

  # I used loop but I think following vector equation is better
  # result <- t(width_vector)%*%fun(midpoint_vector)
  return(result)
}
```

I made the function.

```
# Integral results for my function in different width levels.

# 10 slices
riemann_integral(0,1,10,given_func_int)
```

```
## [1] 0.8559366
```

```
# 100 slices
riemann_integral(0,1,100,given_func_int)
```

```
## [1] 0.855627
```

```
# 1000 slices
riemann_integral(0,1,1000,given_func_int)
```

```
## [1] 0.8556244
```

```
# Exact result using pnorm.
(pnorm(1)-pnorm(0))*sqrt(2*pi)
```

```
## [1] 0.8556244
```

We can see that my function with 1000 slices gives answer within $1e^{-6}$ of the analytical solution.

Problem 7

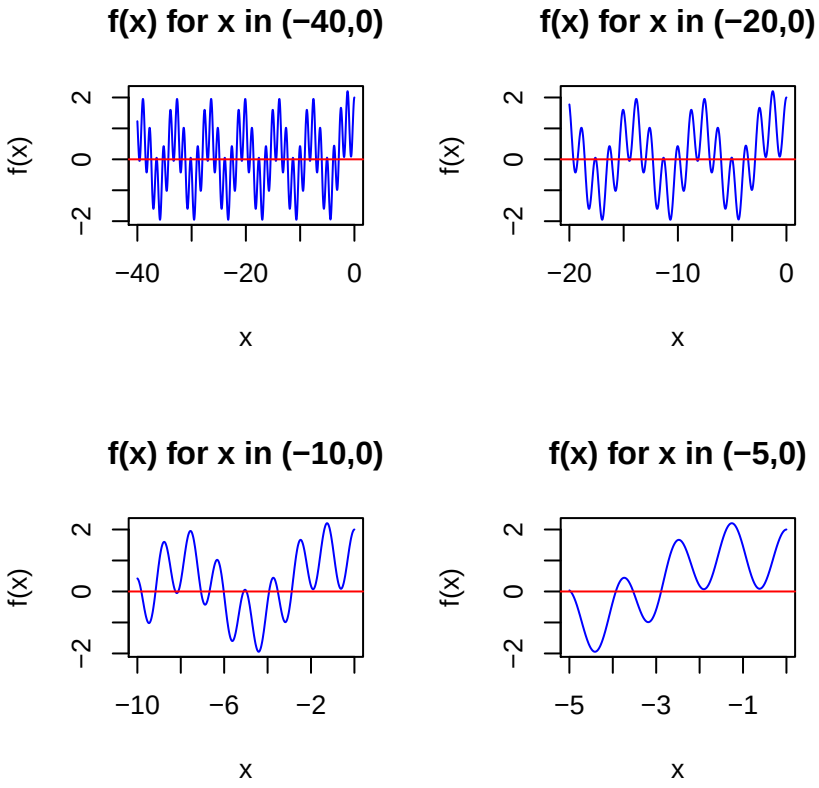
Create a function to find solutions to (1) using Newton's method. The answer should include the solutions with tolerance used to terminate the loop, the interval used, and a plot showing the iterations on the path to the solution.

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

For a refresher on Newton's method:
https://en.wikibooks.org/wiki/Calculus/Newton%27s_Method

```r
given_func <- function(x){3^x-sin(x)+cos(5*x)}
given_func_derivative <- function(x){log(3)*3^x-cos(x)-5*sin(5*x)}
```

I made a function for the given function and its derivative function. Now we have to find solutions that makes $f(x) = 0$ by Newton's Method.



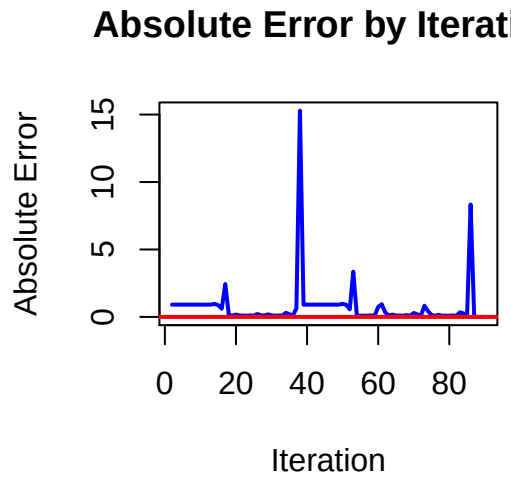
First, I drew plots for the function. There are multiple points that $f(x) = 0$.

```r
newton_for_given_func <- function(initial,max_iter,tolerance,lower_bound,upper_bound){

  # "initial" is initial value for the algorithm.
  # "max_iter" is maximum value for interation.
  # "tolerance" is minimum tolerance for absolute error in the solution.
  # "lower_bound" and "upper_bound" are lower and upper bound for the values.
  # If initial value is out of the boundary, the algorithm will give null values with warning message.
  # Also, If the solution of the algorithm is out of the boundary, there will be warning message.

  # Warning message and null output for initial value out of the boundary.

  if(initial<lower_bound | initial>upper_bound)
  {warning("Initial value should be included in (lower_bound, upper_bound)")
    return(NA)}

  # Set iteration value and values vector. outputs will be stored in values vector.
  iter <- 1
  values <- numeric(max_iter)
  values[1] <- initial

  # first iteration should be completed.
  values[2] <- values[1]-given_func(values[1])/given_func_derivative(values[1])
```

```
  iter <- 2

  # the other iterations.
  # If the iteration reaches max iteration or absolute error is less than tolerance,
  # the algorithm stops.
  while ( (iter<max_iter) && (abs(values[iter]-values[iter-1])>=tolerance) ){
    values[iter+1] <- values[iter]-given_func(values[iter])/given_func_derivative(values[iter])
    iter <- iter+1
  }

  # Plotting absolute errors for iterations.
  # The algorithm stops when the error reaches the tolerance or iteration reaches the max iteration.
  plot(2:iter,abs(values[2:iter]-values[1:iter-1]),type="l",lwd=2, col="blue",
       main="Absolute Error by Iteration", xlab="Iteration", ylab="Absolute Error")
  abline(h=tolerance,lwd=2,col="red")

  # Algorithm gives below list as a result.
  result <- list(values[iter],iter,abs(values[iter]-values[iter-1]), given_func(values[iter]))
  names(result) <- c("Solution", "The Number of Iterations", "Absolute Error", "f(x)")

  # Warning message for iteration reaches max iteration.
  if(iter==max_iter)
    {warning("The algorithm did not converged during iterations.")}

  # Warning message for the solution outside of the boundary.
  if(values[iter]<lower_bound | values[iter]>upper_bound)
    {warning("The solution is out of the boundary.")}
  return(result)
}
```

I made function to get solution from $f(x)$ using Newton's Method.

```
newton_for_given_func(initial=15,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```




```
## $Solution
## [1] -10.21018
##
## $`The Number of Iterations`
## [1] 90
```

```
##
## $`Absolute Error`
## [1] 1.680434e-11
##
## $`f(x)`
## [1] -4.218847e-15
```

Test for initial value 15. The algorithm converged and we can see that it is solution for f(x)=0.

```
newton_for_given_func(initial=15,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=10)
```

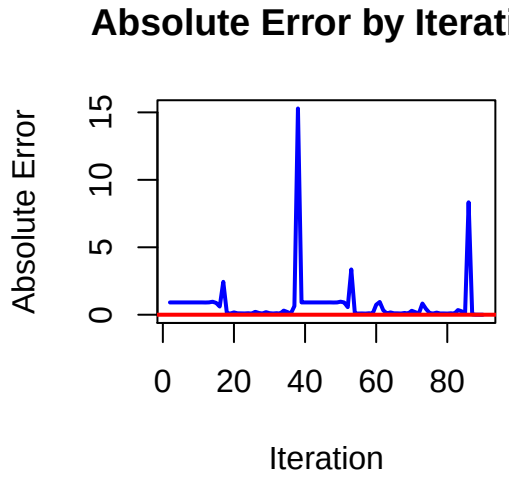
```
## Warning in newton_for_given_func(initial = 15, max_iter = 1000, tolerance =
## 0.1^10, : Initial value should be included in (lower_bound, upper_bound)
```

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

Test for initial value 15 and upper bound 10. We can see that there is warning message and the output is NA.

```
newton_for_given_func(initial=15,max_iter=1000,tolerance=0.1^10,lower_bound=-10,upper_bound=Inf)
```

```
## Warning in newton_for_given_func(initial = 15, max_iter = 1000, tolerance =
## 0.1^10, : The solution is out of the boundary.
```

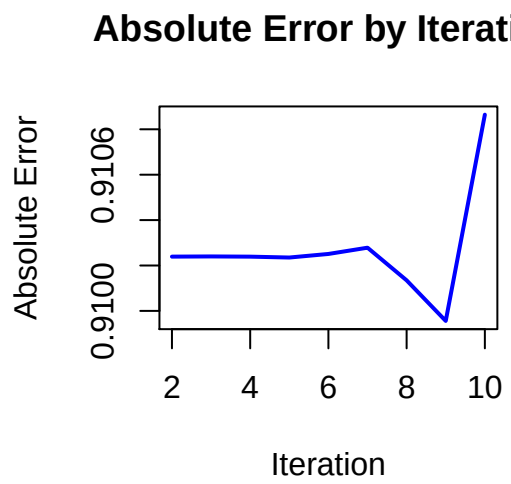



```
## $Solution
## [1] -10.21018
##
## $`The Number of Iterations`
## [1] 90
##
## $`Absolute Error`
## [1] 1.680434e-11
##
## $`f(x)`
## [1] -4.218847e-15
```

Test for initial value 15 and lower bound is -10. The algorithm converged and we can see that it is solution for f(x)=0. But as we set the lower bound is -10, our solution is out of the boundaries and it result in giving warning message.

```
newton_for_given_func(initial=15,max_iter=10,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```

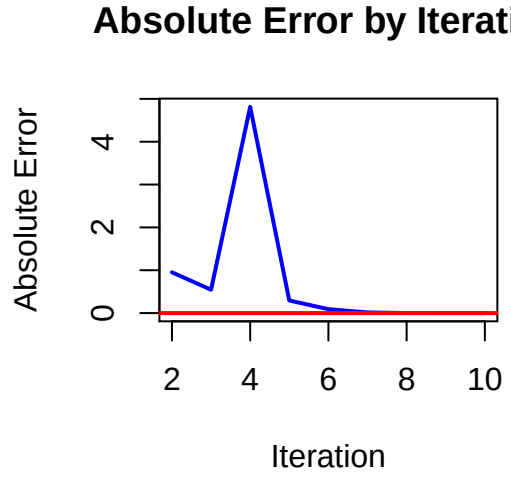
```
## Warning in newton_for_given_func(initial = 15, max_iter = 10, tolerance =
## 0.1^10, : The algorithm did not converged during iterations.
```




```
## $Solution
## [1] 6.807563
##
## $`The Number of Iterations`
## [1] 10
##
## $`Absolute Error`
## [1] 0.9108644
##
## $`f(x)`
## [1] 1768.875
```

Test for initial value 15 and maximum iteration as 10. The algorithm did not converged during the iteration. There is a warning message and we can see that it is not solution for f(x)=0.

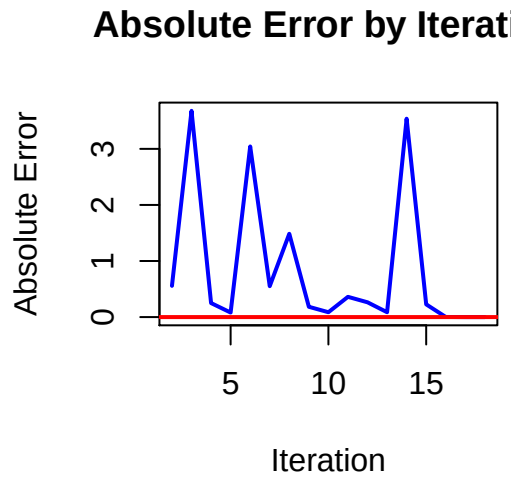
```
newton_for_given_func(initial=3,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```




```
## $Solution
## [1] -3.528723
##
## $`The Number of Iterations`
## [1] 10
##
## $`Absolute Error`
## [1] 3.108624e-15
```

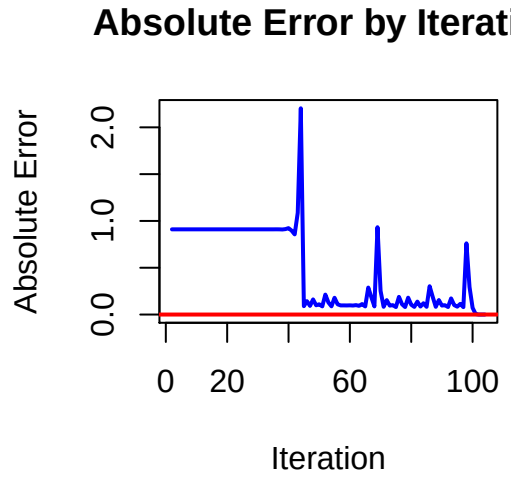
```
##
## $`f(x)`
## [1] -1.498801e-15
```

```
newton_for_given_func(initial=2,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```




```
## $Solution
## [1] -3.930114
##
## $`The Number of Iterations`
## [1] 18
##
## $`Absolute Error`
## [1] 7.861667e-11
##
## $`f(x)`
## [1] -1.44329e-15
```

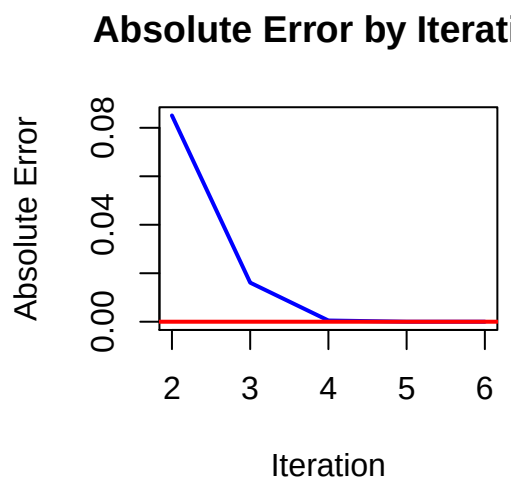
```
newton_for_given_func(initial=40,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```




```
## $Solution
## [1] -2.887058
##
## $`The Number of Iterations`
## [1] 104
```

```
##
## $`Absolute Error`
## [1] 0
##
## $`f(x)`
## [1] 9.436896e-16
```

```
newton_for_given_func(initial=-7,max_iter=1000,tolerance=0.1^10,lower_bound=-Inf,upper_bound=Inf)
```




```
## $Solution
## [1] -7.068483
##
## $`The Number of Iterations`
## [1] 6
##
## $`Absolute Error`
## [1] 4.494183e-13
##
## $`f(x)`
## [1] 4.440892e-16
```

Tests for the Newton's Algorithm function for other initial values. We can see that we got solution for $f(x) = 0$ for all cases.

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:

```
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.

```
set.seed(7.7)
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
```

Generated the data.

(a)

```
SST_a=function(y){
  mean_y <- mean(y)
  sum <- 0
  for(i in 1:length(y)){
    sum <- sum+(y[i]-mean_y)^2
    i <- i+1
  }
  return(sum)
}

SST_a(y)
```

```
## [1] 20575.59
```

```
abs(SST_a(y)-sum((y-mean(y))^2))
```

```
## [1] 3.637979e-12
```

I made SST function with for loop. The result of the function is the same as true SST.

(b)

```
SST_b <- function(y){
  SST <- t(y)%*%(diag(length(y))-matrix(1/length(y),length(y),length(y)))%*%y
  return(SST)
}

SST_b(y)
```

```
##          [,1]
## [1,] 20575.59
```

```
abs(SST_b(y)-sum((y-mean(y))^2))
```

```
##              [,1]
## [1,] 1.455192e-11
```

I made SST function with matrix equation. The result of the function is the same as true SST.

```r
library(microbenchmark)
times<-microbenchmark(result1<-SST_a(y),result2<-SST_b(y),times = 100, unit = "ms")
times
```

```
## Unit: milliseconds
##                 expr      min       lq       mean    median        uq      max
##  result1 <- SST_a(y) 0.008977 0.009270 0.01012092 0.0097105 0.0101800 0.021095
##  result2 <- SST_b(y) 0.035132 0.102216 0.09821934 0.1061820 0.1093005 0.127472
##  neval
##    100
##    100
```

SST_a used for loop and SST_b used matrix equation. SST_a is much faster than SST_b.

Problem 9

Finish this homework by pushing your changes to your repo.

Only submit the .Rmd and .pdf solution files. Names should be formatted HW3_pid.Rmd and HW3_pid.pdf