HW4_Kim_Jaiyool

Due Wednesday 9am September 25, 2019

2019-09-25

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.

This week, we spoke about R and version control, munging and 'tidying' data, good programming practice and finally some basic programming building blocs. To begin the homework, we will for the rest of the course, start by loading data and then creating tidy data sets.

Problem 1

Work through the "R Programming E" lesson parts 8 and 9.

From the R command prompt:

```
library(swirl)
swirl()
```

I have finished taking the classes you assigned.

Problem 2

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

The filename should be: HW4_lastname_firstname, i.e. for me it would be HW4_Settlage_Bob

You will use this new R Markdown file to solve problems 4-10.

I made the names of this R Markdown file as you ordered.

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 generally strongly agree with the ideas and cautions written in two webpages, especially Pipes, indentation, and explicit returns. In terms of these things, I will try to make an effort to be accustomed to use pipe when it comes to confront the situation that I have to reduce the redundant code firstly. I also will take care about the time I should use the indentation and return for viewers who will read my R code.

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*. In this problem, please using *lintr* to lint your last homework (if you didn't do it, perhaps the time is now ;)). In my case, the command looks like this (takes a few moments to run):

```
    lint(filename = "./02_data_munging_summarizing_R_git/HW3_Settlage_Bob.Rmd")
```

Can you clean up your code to pass the major issues?? <— just a challenge, not part of the problem!!

Note that really all we have done is syntax checking and received a few stylistic suggestions. We COULD create a custom linter to check for indenting, etc. For now, I think it is enough to know there are a lot of

opinions on what code should look like and working in teams requires you to code nicely!! So, clean up your code!!

From the messages, what are some things you need to change in your code?

Yes, I received a lot of messages from the above code, and I will soon reflect those messages into my previous homework R codes.

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** to:

1. calculate the mean for dev1
2. calculate the mean for dev2
3. calculate the sd for dev1
4. calculate the sd for dev2
5. calculate the correlation between dev1 and dev2
6. return the above as a single data.frame

We will use this function to summarize a dataset **which has multiple repeated measurements from two devices by thirteen Observers**. In the current lecture directory, you will see a file "HW4_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.

The output of this problem should be:

a. A single table of the means, sd, and correlation for each of the 13 Observers

b. A box plot of all the means to compare the spread of means from dev1 to dev2

c. A violin plot of all the sd to compare the spread of sd from dev1 to dev2

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

```
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
```

```
library(ggplot2)
library(knitr)


Prob.5_data <- readRDS("HW4_data.rds", refhook = NULL)

## Raw data (to 50th rows)
kable(Prob.5_data[1:50,], caption="Raw data")
```

Table 1: Raw data

Observer	dev1	dev2
4	55.3846	97.1795
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
4	23.0769	66.4103
4	22.3077	61.7949
4	22.3077	57.1795
4	23.3333	52.9487
4	25.8974	51.0256
4	29.4872	51.0256
4	32.8205	51.0256
4	35.3846	51.4103
4	40.2564	51.4103
4	44.1026	52.9487
4	46.6667	54.1026
4	50.0000	55.2564
4	53.0769	55.6410
4	56.6667	56.0256
4	59.2308	57.9487
4	61.2821	62.1795
4	61.5385	66.4103
4	61.7949	69.1026
4	57.4359	55.2564
4	54.8718	49.8718
4	52.5641	46.0256
4	48.2051	38.3333
4	49.4872	42.1795
4	51.0256	44.1026
4	45.3846	36.4103
4	42.8205	32.5641
4	38.7179	31.4103
4	35.1282	30.2564
4	32.5641	32.1795
4	30.0000	36.7949
4	33.5897	41.4103
4	36.6667	45.6410
4	38.2051	49.1026
4	29.7436	36.0256
4	29.7436	32.1795
4	30.0000	29.1026
4	32.0513	26.7949
4	35.8974	25.2564
4	41.0256	25.2564
4	44.1026	25.6410

```r
summary_tab<- Prob.5_data %>%
          group_by(Observer) %>%
          summarize(mean_dev.1 = mean(dev1),

                    mean_dev.2 = mean(dev2),

                    sd_dev.1 = sd(dev1),

                    sd_dev.2 = sd(dev2),

                    cor_dev.1_between_dev.2 = cor(dev1,dev2))

kable(summary_tab, caption="Tidy data")
```

Table 2: Tidy data

Observer	mean_dev.1	mean_dev.2	sd_dev.1	sd_dev.2	cor_dev.1_between_dev.2
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

```r
## Define the new dataframe (denoted by mean.data_frame by_dev) arranged by dev.

mean.dev1 <- summary_tab$mean_dev.1
mean.dev2 <- summary_tab$mean_dev.2

mean.matrix.by.dev <- as.matrix(cbind(mean.dev1,mean.dev2))

mean.data_frame_by_dev <- as.data.frame(mean.matrix.by.dev)

kable(mean.data_frame_by_dev, caption="Mean data sorted by Dev")
```

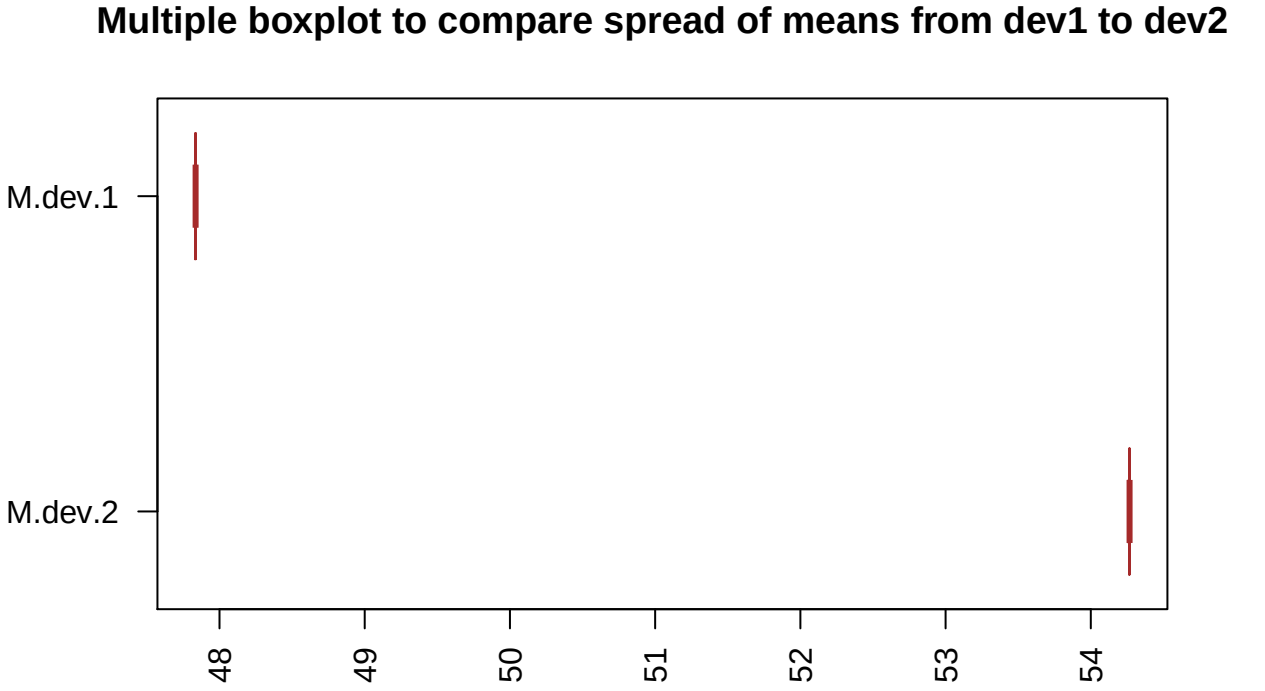
Table 3: Mean data sorted by Dev

mean.dev1	mean.dev2
54.26610	47.83472
54.26873	47.83082
54.26732	47.83772
54.26327	47.83225
54.26030	47.83983
54.26144	47.83025
54.26881	47.83545

mean.dev1	mean.dev2
54.26785	47.83590
54.26588	47.83150
54.26734	47.83955
54.26993	47.83699
54.26692	47.83160
54.26015	47.83972

```
## A box plot of all the means to compare the spread of means from dev1 to dev2

boxplot(mean.data_frame_by_dev$mean.dev1, mean.data_frame_by_dev$mean.dev2,
main = "Multiple boxplot to compare spread of means from dev1 to dev2",
at = c(1,3),
names = c("M.dev.2", "M.dev.1"),
las = 2,
col = c("blue","red"),
border = "brown",
horizontal = TRUE,
notch = TRUE
)
```

```
## Warning in bxp(list(stats = structure(c(54.260150334038,
## 54.2632732394366, : some notches went outside hinges ('box'): maybe set
## notch=FALSE
```



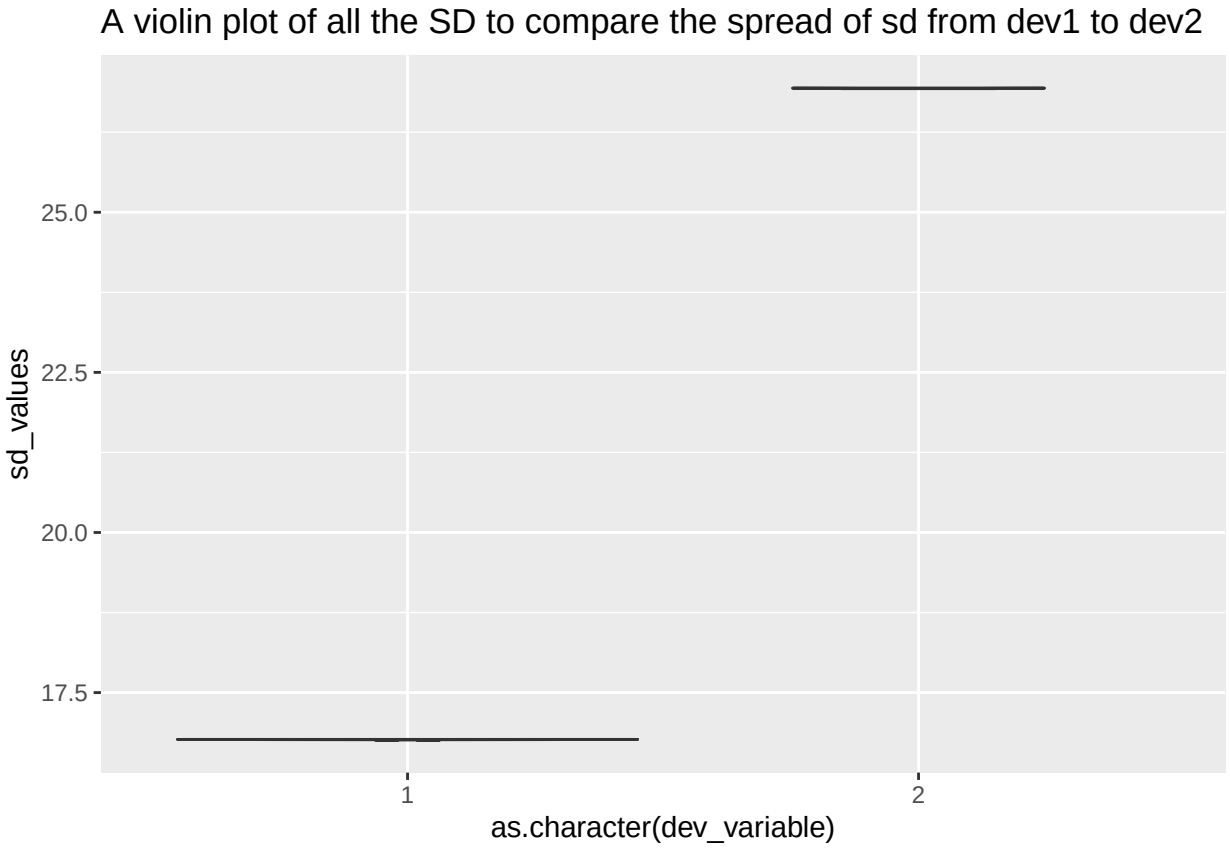

```
## Define the new dataframe (denoted by sd.data_frame by_dev) arranged by dev.

sd.dev1 <- summary_tab$sd_dev.1

sd.dev2 <- summary_tab$sd_dev.2

sd_values <- c(sd.dev1,sd.dev2)
```

```
## Define the factor variable (denoted by dev_variable)
dev_variable<-factor(rep(c('dev_1','dev_2'),each=length(sd.dev1)))

dev_variable <- dev_variable

## cbind the dev_variable and sd_values

sd.data_frame_by_dev <- as.data.frame(cbind(dev_variable,sd_values))

kable(sd.data_frame_by_dev, caption="Standard Deviation data sorted by Dev")
```

Table 4: Standard Deviation data sorted by Dev

dev_variable	sd_values
1	16.76983
1	16.76924
1	16.76001
1	16.76514
1	16.76774
1	16.76590
1	16.76670
1	16.76676
1	16.76885
1	16.76896
1	16.76996
1	16.77000
1	16.76996
2	26.93974
2	26.93573
2	26.93004
2	26.93540
2	26.93019
2	26.93988
2	26.94000
2	26.93610
2	26.93861
2	26.93027
2	26.93768
2	26.93790
2	26.93000

```
## A violin plot of all the sd to compare the spread of sd from dev1 to dev2
ggplot(sd.data_frame_by_dev,aes(x=as.character(dev_variable), y=sd_values)) +
geom_violin() +
ggtitle("A violin plot of all the SD to compare the spread of sd from dev1 to dev2")
```



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.

```
## Define dexp function first.

dexp <- function(quantiles){
  return(exp(-quantiles^2/2))
}


## This is the main function to obtain the Riemann sums !

reimann.sums <- function(x, f){

  #obtain length of variable of integration and integrand
```

```
  n.points = length(x)

  # midpoints

  midpoints = 0.5*(x[2:n.points]+x[1:(n.points-1)])

  # function evaluated at midpoints

  f.midpoints = f(midpoints)

  # calculate the widths of the intervals between adjacent pairs of
  # points along the variable of integration
  interval.widths = x[2:n.points] - x[1:(n.points-1)]

  # implement rectangular integration
  # calculate the sum of all areas of rectangles to
  # approximate the integral

   rectangular.integral = sum(interval.widths * f.midpoints)


   return(rectangular.integral)
}

# points along support set for exp(-x^2/2)

exp.support = seq(0, 1, by = 0.001)


# calculate integral of exp

reimann.sums(exp.support, dexp)
```

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

The final result of Reimann.sums function is nearly similar to the integrated value of given function (0.8555).

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.

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

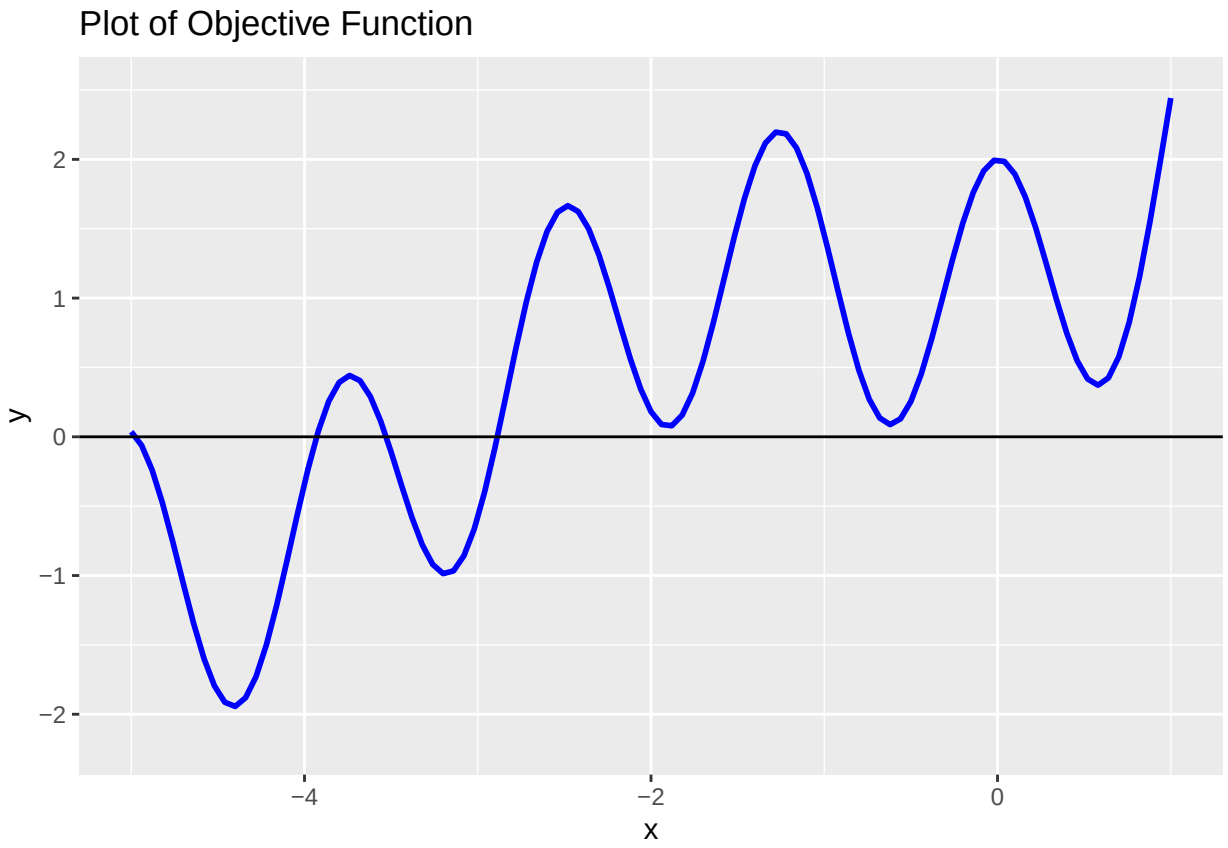
```
library(ggplot2)
library(tidyr)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------- t
## v tibble  2.1.3     v purrr   0.3.2
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------- tidyvers
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
## Let us define the objective function.

obj.function <- function(x){

  3^x - sin(x) + cos(5*x)

}



## Define the first derivative of the objective function.

dx.obj.function <- function(x) { 3^x*log(3) - cos(x) -5*sin(5*x) }



## Set the sequence denoted by x.

x <- seq(-5:1)


#### R code to find the root(s) of the given objective function.
## stop_criteria can be changed by users
## (left_b, right_b) : the interval where we want to find the optimal solution.
## x_0 : original initial value

Newton_Method <- function(f, df, x_0, stop_criteria = 0.0001,
                          max_iter = 10^3, left_b = -Inf, right_b = Inf) {


  # criteria for starting point x_0
  if (left_b > x_0 | right_b < x_0) { warning("your starting point is out of bound,
                                              please try point in the bounds")
    return(NA) }

  # variable initialization
  iters <- 0

  x_1 <- -4 ## Actual Initial value : very important
  ## if you wanna see the long progress of this algorithm.

  distance <- Inf

  # do while until

  # 1) x does not move

  # 2) exceed the maximun iteration

  # 3) the function value become close to zero
```

```
  while ((distance > stop_criteria) &
         (iters < max_iter) &
         (abs(f(x_0)) > stop_criteria)){

     # newton method

    x_1 <- x_0 - ( f(x_0) / df(x_0) )

    # out of bound case

    if (x_1 <= left_b) {

      { x_1 <- left_b }

    }

    if (right_b <= x_1) {

      { x_1 <- right_b }

    }


    distance <- abs(x_0 - x_1)

    iters <- iters + 1

    x_0 <- x_1 }

  # return the final point

  if (x_0 == left_b | x_0 == right_b){ warning("\n", "Final point is on the boundary", "\n") }

  if (iters > max_iter){ warning("\n", "Cannot find the root during the iteration.", "\n") }

  cat("Final value of function:", f(x_0), "\n")

  x_0

  }

## Drawing the shape of the objective function.

p <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))

p <- p + stat_function(fun= obj.function, color="blue", size=1.0) +
  geom_hline(yintercept=0) + xlim(-5, 1) + ylim(-2.2, 2.5)

p + ggtitle("Plot of Objective Function")
```




```
## Do simulate the following code to find the solution of objective function
## in the set : (-5,4) (This is arbitrarily set.)

## c.f) Actually, this code is really influenced by the initial value, so it is important
## to set a variety of initial values to find the global minimum of the objective function.

Newton_Method(f = obj.function, df = dx.obj.function, x_0 = -4, left_b = -5, right_b = 1)
```

```
## Final value of function: -2.80877e-08
```

```
## [1] -3.930114
```

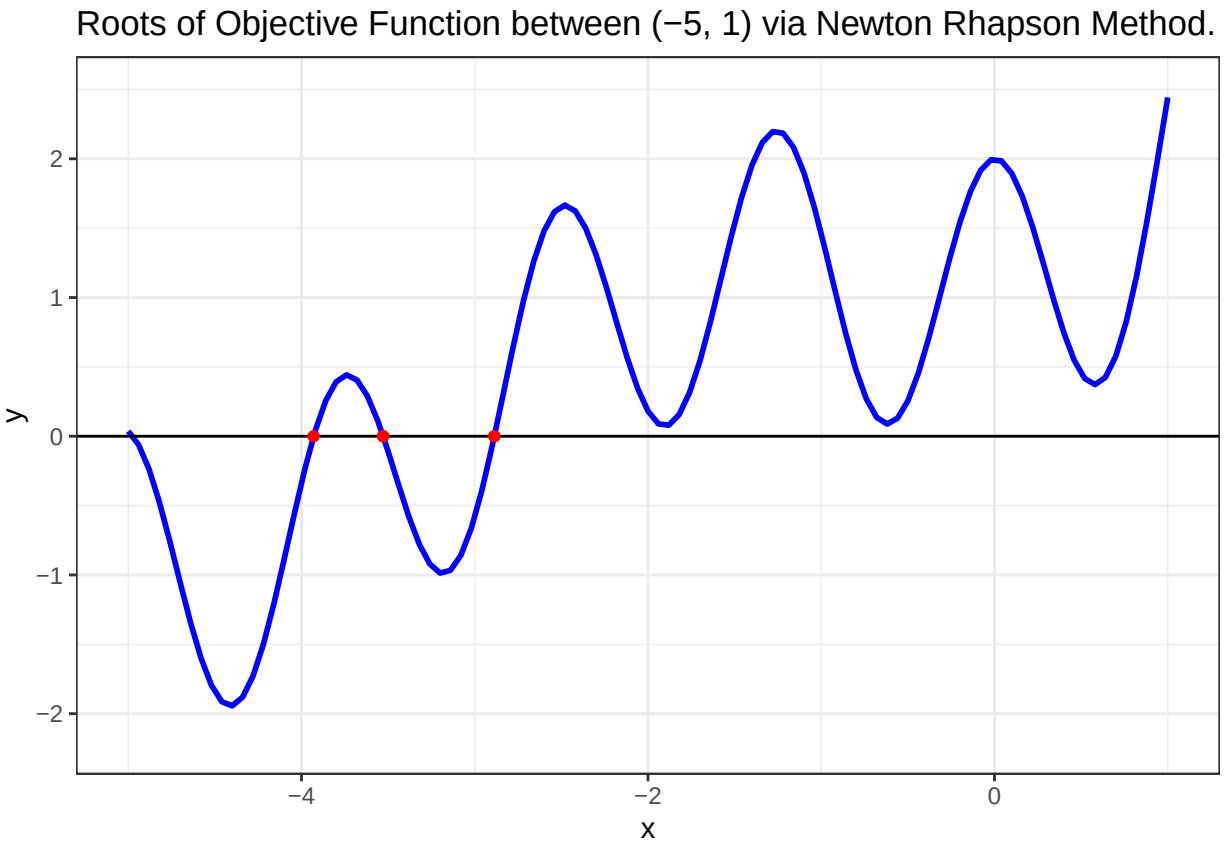
```
## We can apply functions simultaneously to find the optimal solutions via NR.

roots <- sapply(x, Newton_Method, f = obj.function, df = dx.obj.function)
```

```
## Final value of function: -7.83973e-06
## Final value of function: -2.747667e-05
## Final value of function: -1.803155e-07
## Final value of function: 2.767141e-07
## Final value of function: 4.369841e-06
## Final value of function: -1.666584e-06
## Final value of function: 3.35735e-05
```

```
## plotting the points on the objective function's graph.

roots_p <- data.frame(x_star = roots, y_star = obj.function(roots))

p <- p + geom_point(data = roots_p, aes(x = x_star, y = y_star),
```

```
                    color = "red", size = 1.5) + theme_bw()

p + ggtitle("Roots of Objective Function between (-5, 1) via Newton Rhapson Method.")
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



As you can see the above R code and results, we can find the optimal solution(s) for given equation. For reference, if you want to find the global minimun of this kind of functions that have many local minimum values, you should set a variety of initial values to keep track of the real global minimum.

Problem 8

Finish this homework by pushing your changes to your repo.

Only submit the .Rmd and .pdf solution files. Names should be formatted HW4_lastname_firstname.Rmd and HW4_lastname_firstname.pdf

Optional preperation for next class:

Next week we will talk about Exploratory Data Analysis and graphing. Swirl will be a bit part of this. Check out "Exploratory_Data_Analysis" Swirl lessons 1-10.