Homework 4 solutions

Due Wednesday 9am September 25, 2019

2019-10-04

Problem 4

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, personally, try to better comment my code, absolutely always indent, and try to keep a consistent naming strategy for objects and functions.

Problem 5

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

Mostly, what I have already stated. This isn't the best linter as configured out of the box.

Problem 6

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

```
### function to calculate summary statistics of 2 column data frame by column
summary_stats <- function(input_df = data.frame(matrix(rnorm(100),ncol=2))){
  # calc mean, sd for columns, calc correlation between columns, return as df
  temp_df <- as.data.frame(c(colMeans(input_df),
                   sd(input_df[,1]),sd(input_df[,2]),
                   cor(input_df[,1],input_df[,2]),cor(input_df[,2],input_df[,1])))
  rownames(temp_df) <- c("mean1","sd1","corr1","mean2","sd2","corr2")
  return(temp_df)
}
```

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

```r
HW4_data <- readRDS("HW4_data.rds")
# get summary stats by observer using a loop
result_df <- data.frame(matrix(rep(0,3*13*2),nrow=6))
for(i in 1:13){
  result_df[,i] <- summary_stats(HW4_data[HW4_data$Observer==i,2:3])
}
colnames(result_df) <- paste0("Obs",1:13)
rownames(result_df) <- c("mean1","sd1","corr1","mean2","sd2","corr2")

kable(result_df[,1:6], caption="Summary statistics by observer")
```

Table 1: Summary statistics by observer

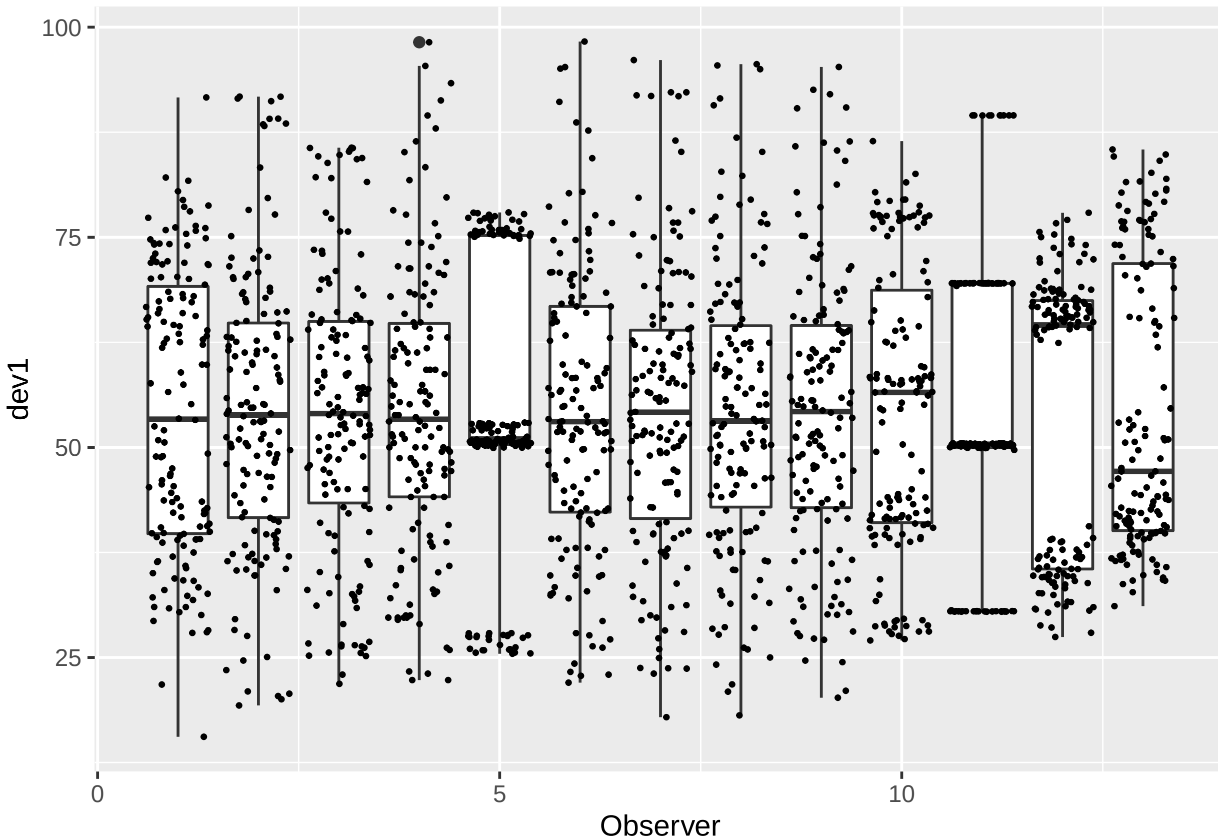
	Obs1	Obs2	Obs3	Obs4	Obs5	Obs6
mean1	54.2660998	54.2687300	54.2673197	54.2632732	54.2603035	54.2614418
sd1	47.8347206	47.8308232	47.8377173	47.8322528	47.8398292	47.8302519
corr1	16.7698250	16.7692395	16.7600127	16.7651420	16.7677355	16.7658979
mean2	26.9397434	26.9357267	26.9300361	26.9354035	26.9301915	26.9398762
sd2	-0.0641284	-0.0685864	-0.0683434	-0.0644719	-0.0603414	-0.0617148
corr2	-0.0641284	-0.0685864	-0.0683434	-0.0644719	-0.0603414	-0.0617148

```r
kable(result_df[,7:13], caption="Summary statistics by observer")
```

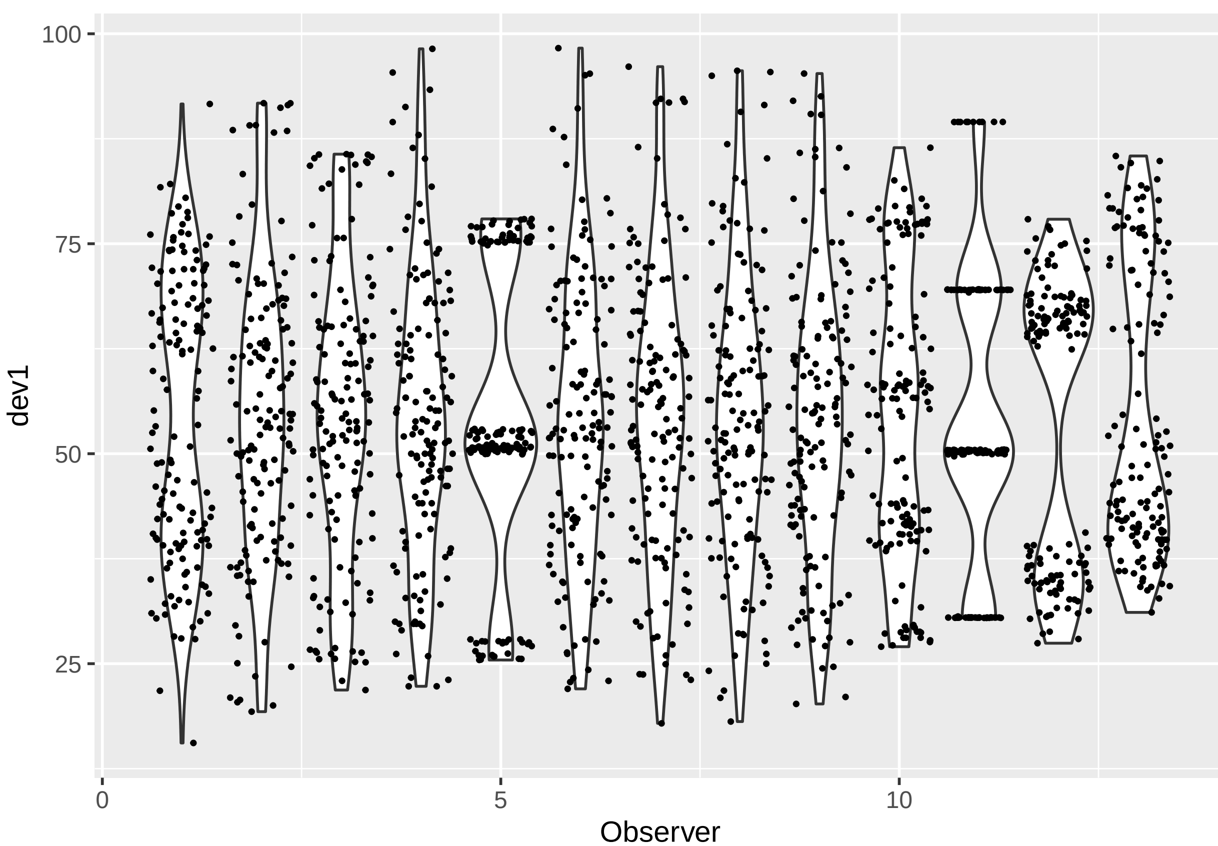
Table 2: Summary statistics by observer

	Obs7	Obs8	Obs9	Obs10	Obs11	Obs12	Obs13
mean1	54.2688053	54.2678488	54.2658818	54.2673411	54.2699272	54.2669163	54.2601503
sd1	47.8354502	47.8358963	47.8314957	47.8395452	47.8369880	47.8316020	47.8397173
corr1	16.7667040	16.7667589	16.7688527	16.7689592	16.7699586	16.7699996	16.7699577
mean2	26.9399980	26.9361049	26.9386081	26.9302747	26.9376838	26.9379019	26.9300017
sd2	-0.0685042	-0.0689797	-0.0686092	-0.0629611	-0.0694456	-0.0665752	-0.0655833
corr2	-0.0685042	-0.0689797	-0.0686092	-0.0629611	-0.0694456	-0.0665752	-0.0655833

This isn't very interesting as we will discuss next time. When combined with the following two plots, some questions may surface.



100
75
50
25
dev1
0
5
10
Observer



100
75
50
25
dev1
0
5
10
Observer

Problem 7

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.

We could solve this analytically to check out answer, in fact, as staticians, we could recognize the kernel of the standard normal distribution and take a big shortcut ($\sqrt{2\pi}*$(pnorm(1)-pnorm(0)) but, we will go ahead and do it the hard way:

```
  # quick function to compute the area in a rectangle, for ease, just hard coding the function
  area <- function(x_left, x_right, functn, keep="left"){
    #this function calculates the area under the curve as a rectangle approximation
    #keep can be one of "left", "right", "middle" depending on what you want for height
    if(keep=="left"){
      rect_area <- abs(x_left - x_right)*exp(-(x_left^2)/2)
    }else if(keep=="right"){
      rect_area <- abs(x_left - x_right)*exp(-(x_right^2)/2)
    }else{
      rect_area <- abs(x_left - x_right)*exp(-((x_left+(x_left-x_right)/2)^2)/2)
    }
    return(rect_area)
  }

slices <- 10000
bounds <- seq(from=0,to=1,length.out = slices)
total_area <- 0
for(i in 2:slices){
  total_area <- total_area + area(bounds[i-1],bounds[i],keep = "mid")
}
```

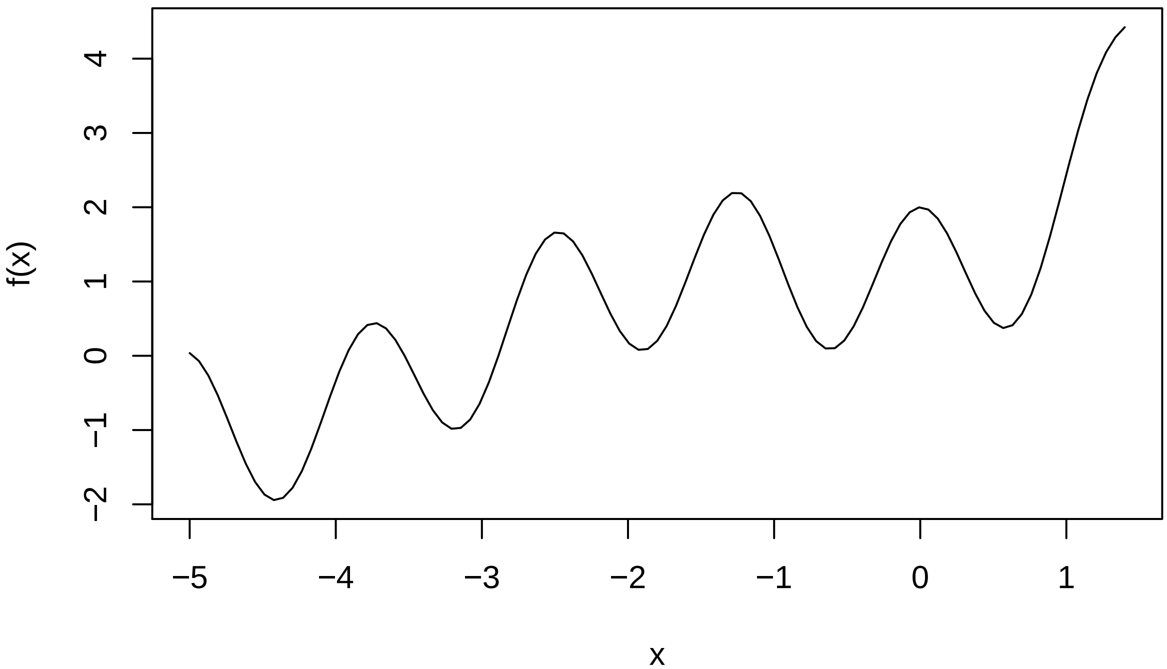
Checking the analytical (and shortcut) answer with our answer of 0.8556637, we see agreement to the 4th decimal.

Problem 8

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.

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

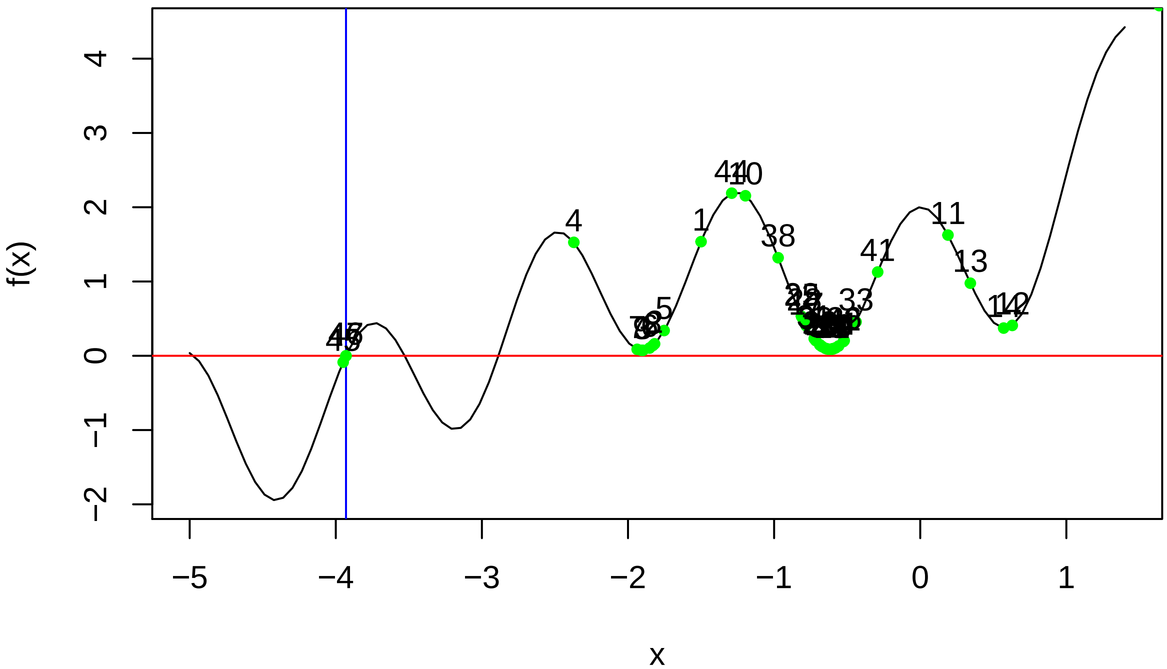
Well, ok, first let's plot it to see where we should look for roots. Starting with wide start and end because don't know much about the equation.




```r
# function will use Newtons method given in class notes
# for simplicity, plugging in the derivative directly
newton <- function(initGuess){
  fx <- 3^initGuess - sin(initGuess) + cos(5*initGuess)
  fxprime <- log(3)*3^(initGuess) - cos(initGuess) - 5*sin(5*initGuess)
  f <- initGuess - fx/fxprime
}

roots <- c(-1.5,rep(0,1000))
i<-2
tolerance <- 0.01
move <- 2
while(move>tolerance && i < 1002){
  roots[i] <- newton(roots[i-1])
  move <- abs(roots[i-1]-roots[i])
  i <- i+1
}
```

OK, so decided to start at -1.5 and see what we get with a 0.01 tolerance. Not too bad as shown below. Although, kinda surprised it missed the one closer to the start. It is kinda cool how it dithered in the bottoms where the curve gets close to 0.



f(x)
x
−5
−4
−3
−2
−1
0
1
−2
−1
0
1
2
3
4