HW4_linal20

Lina Lee

9/30/2020

Problem 3

good coding is making coding easier to read. 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? To improve my coding style, I will name variable or function with meaningful meaning, I will use nouns for the variable names and verbs for function names. I will name function with BigCamelCase to distinguish it from variable names. In addition, I will place proper spaces around all infix operators.

Problem 5 —-

create a single function which returns a vector containing mean and standard deviation

```
summaryStat <- function(x,y){
  mean1 <- mean(x)
  mean2 <- mean(y)
  sdv1 <- sd(x)
  sdv2 <- sd(y)
  corr <- cor(x,y)
  result <- c(mean1,mean2,sdv1,sdv2,corr)
return(result)
  }
```

read data into R

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

```
# loop through the Observers collecting the summary statistics via your function
# for each Observer separately and store in a single dataframe.

# create the empty array and list
sumDat <- array(NA,c(13,6))
sumDatList <- list()


for(i in 1:13){
```

Table 1: The table of the means, sd, and correlation for each of the 13 Observers

Observer	mean1	mean2	sdv1	sdv2	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

```
  # make subset data for each observer
subDat <- df %>%
  filter(Observer==i)
  # caclulate summary statistics and save into the empty array
sumDat[i,1:5] <- summaryStat(subDat$dev1,subDat$dev2)
sumDat[i,6] <- i

}
# transfrom array into dataframe
sumDat <- as.data.frame(sumDat)
# change the column names
names(sumDat) <- c("mean1","mean2","sdv1","sdv2","corr","Observer")
# change the column position
sumDat <- sumDat[,c(6,1,2,3,4,5)]
```

a. A single table of the means, sd, and correlation for each of the 13 Observers (?kable). From this

table, what would you conclude?

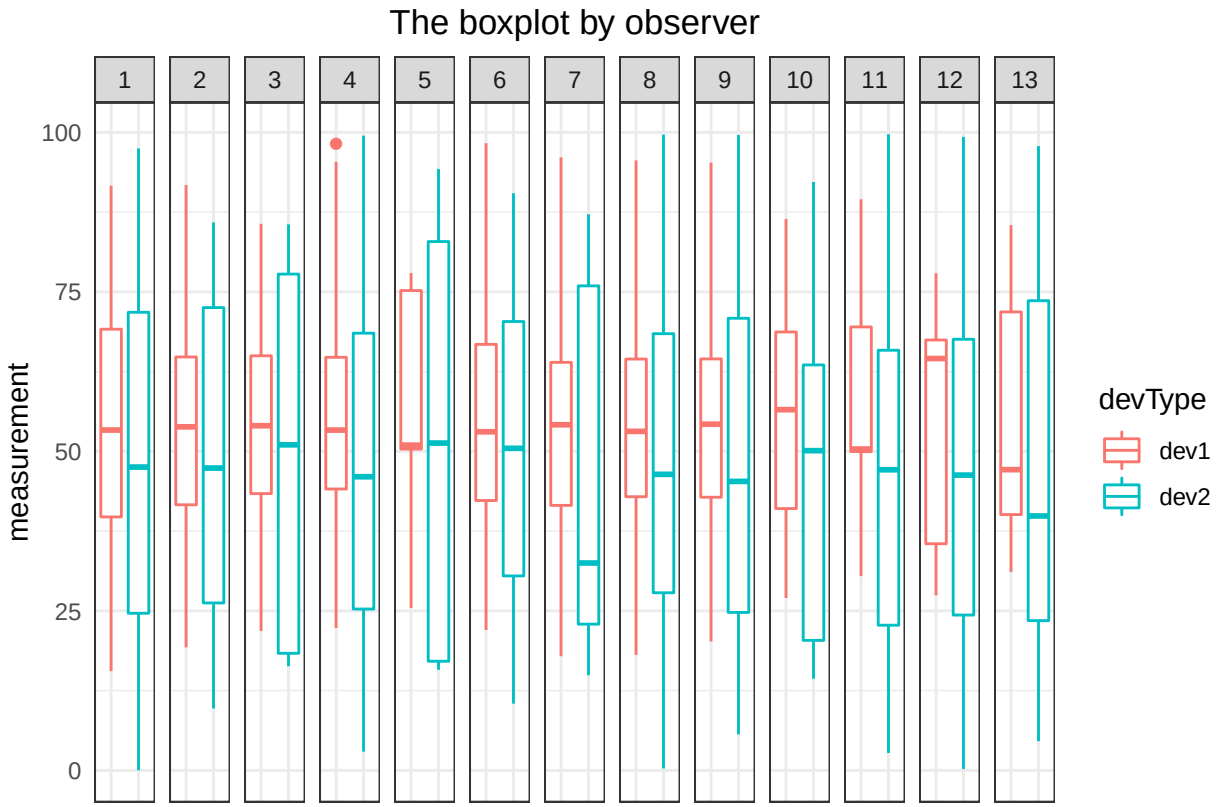
```
knitr::opts_chunk$set(fig.align = 'center')

kable(sumDat,caption = "The table of the means, sd, and correlation for each of the 13 Observers")
```

```
##%>% kable_styling(full_width = T)
```

For every observer, the mean measurement of device 1 is larger than the mean measurement of device 2. However, the standard deviation of device 2 is larger than the standard deviation of device 1. data points for device2 spread more widely than those of device1.

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

```
dfBox <- df %>% gather(key = "devType", value = "measurement",2:3)

ggplot()+
  geom_boxplot(aes(x=devType,y=measurement,
                   group=devType,
                   color=devType),
               data=dfBox)+
  theme_bw()+
  theme(axis.ticks = element_blank(),axis.text.x = element_blank())+
  facet_grid(~Observer)+
  labs(title="The boxplot by observer",x="")+
  theme(plot.title = element_text(hjust = 0.5))
```



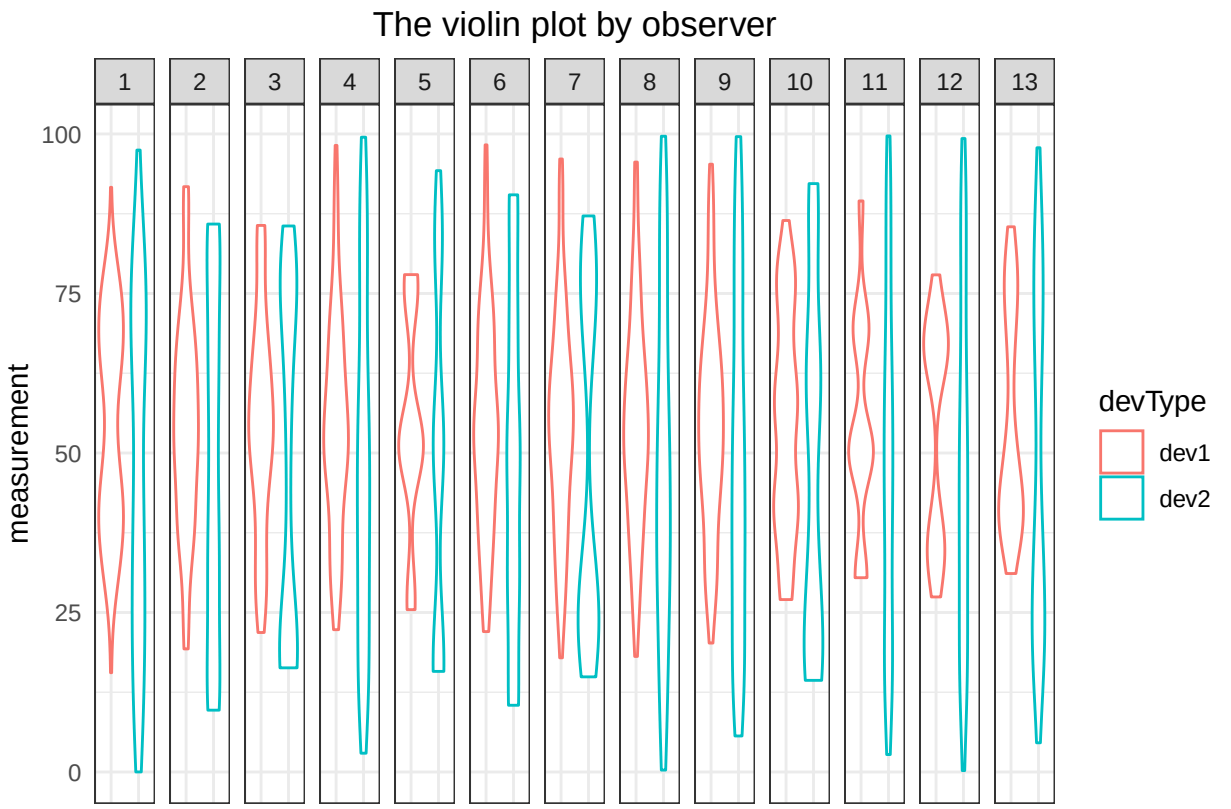
It shows the similar results of the conclusion of a. The median of device 1 is larger than that of device 2. The IQR of device 2 is larger than that of device 1, which means the measurements for device2 spread more widely than that of device1.

c. A violin plot of dev by Observer.

From these plots, what would you conclude? Compared to the boxplot and summary statistics, thoughts?

```
ggplot()+
  geom_violin(aes(x=devType,y=measurement,
                  group=devType,
```

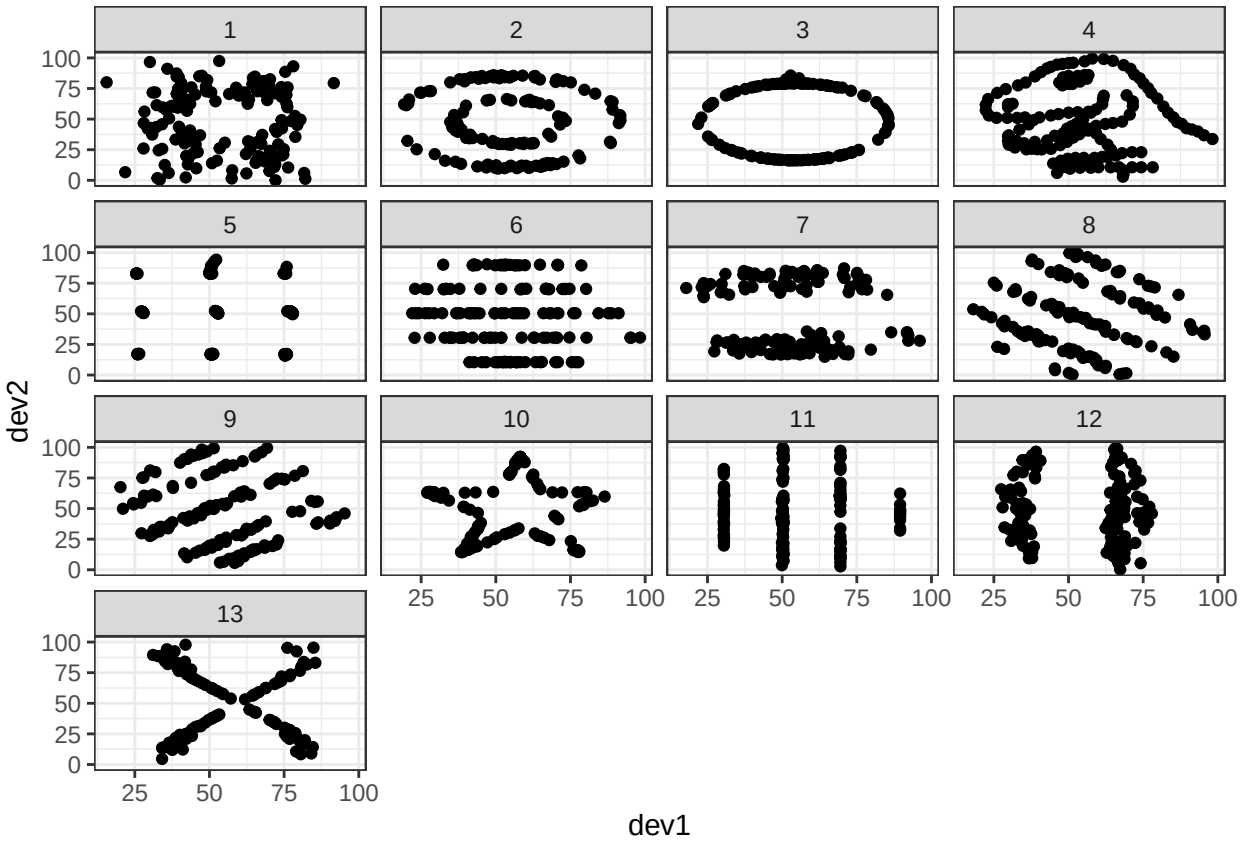
```
                color=devType),
            data=dfBox)+
  theme_bw()+
  theme(axis.ticks = element_blank(),axis.text.x = element_blank())+
  facet_grid(~Observer)+
  labs(title="The violin plot by observer",x="")+
  theme(plot.title = element_text(hjust = 0.5))
```



The violin plot matches the pattern of the boxplot. However, this shows the density of device measurement for each observation although the boxplot shows median and IQR. The measurement of device 2 spread more widely than the measurement of device1. the most medians of device 1 are higher than that of device2. However, the density of device 1 and device2 are different by each observer.

d.a scatter plot of the data using ggplot, geom_points, and add facet_wrap on Observer. 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?

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



The summary statistics do now shows all the aspect of data points. Each statistics or plot display certain aspect of information. Therefore, we need to look at all the possible information (summary statistics or graph) and incorporate those into a conclusion.

Problem 6 —-

```
# create a function for the integrand

integrand <- function(x) {exp((-x**2)/2)}

# see integral of exp((-x**2)/2)
integral(integrand,0,1)
```

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

```
# create a function of ReimannSum
ReimannSum<-function(dx){
x<-seq(0,1,by=dx)
integral<-sum(exp(-x**2/2)*dx)
return(integral)
}


# create an empty vector
rSumVec<-rep(NA,5)
# create a vector of different dx
```

```r
dxVec<-c(0.01,0.001,0.0001,0.00001,0.000001)

# loop ReimannSum through different dx
for(i in 1:5 ){
  rSumVec[i]<-ReimannSum(dxVec[i])
}

# show the ReimannSum result
rSumVec
```

```
## [1] 0.8636520 0.8564276 0.8557047 0.8556324 0.8556252
```

```r
# dxVec<-rev(seq(0.000001,0.00001,by=0.000001))
# length(dxVec)

# for(i in 1:length(dxVec)){
#   rSumVec[i]<-ReimannSum(dxVec[i])
# diff[i]=abs(rSumVec[i]-0.8556244)
# if(diff<0.000001){
#   print(dxVec[i])
# }
# }
#

# check whether the sum is within $1e^{-6}$ of the analytical solution

diff=abs(rSumVec[i]-0.8556244)
diff<0.000001
```

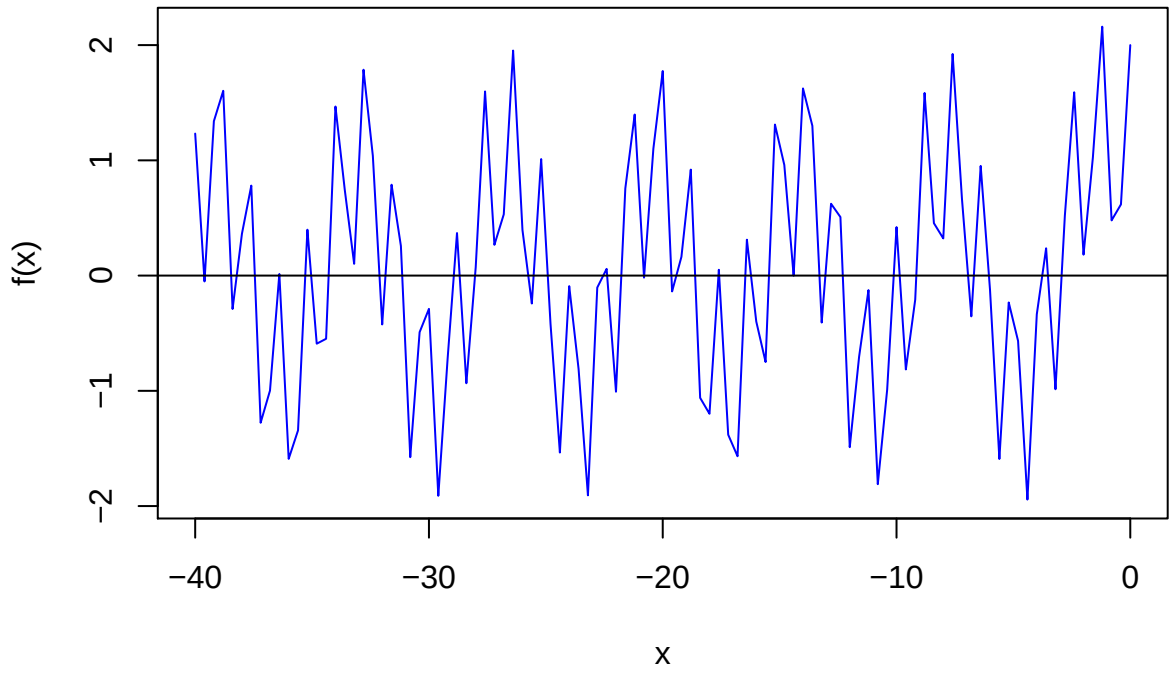
```
## [1] TRUE
```

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.

The various slice widths I used are 0.01,0.001,0.0001,0.00001, and 0.000001 and the sum calculated are 0.86365, 0.85642, 0.85570, 0.85563,and 0.85562. When bandwidh are 0.000001, the sum is within $1e^{-6}$ of the analytical solution

problem7 —-

```r
# create a function of given formula
f <- function(x){return(3^x-sin(x)+cos(5*x))}

# plot the function
curve(expr=f,from=-40,to=0,col="blue")
abline(h=0)
```

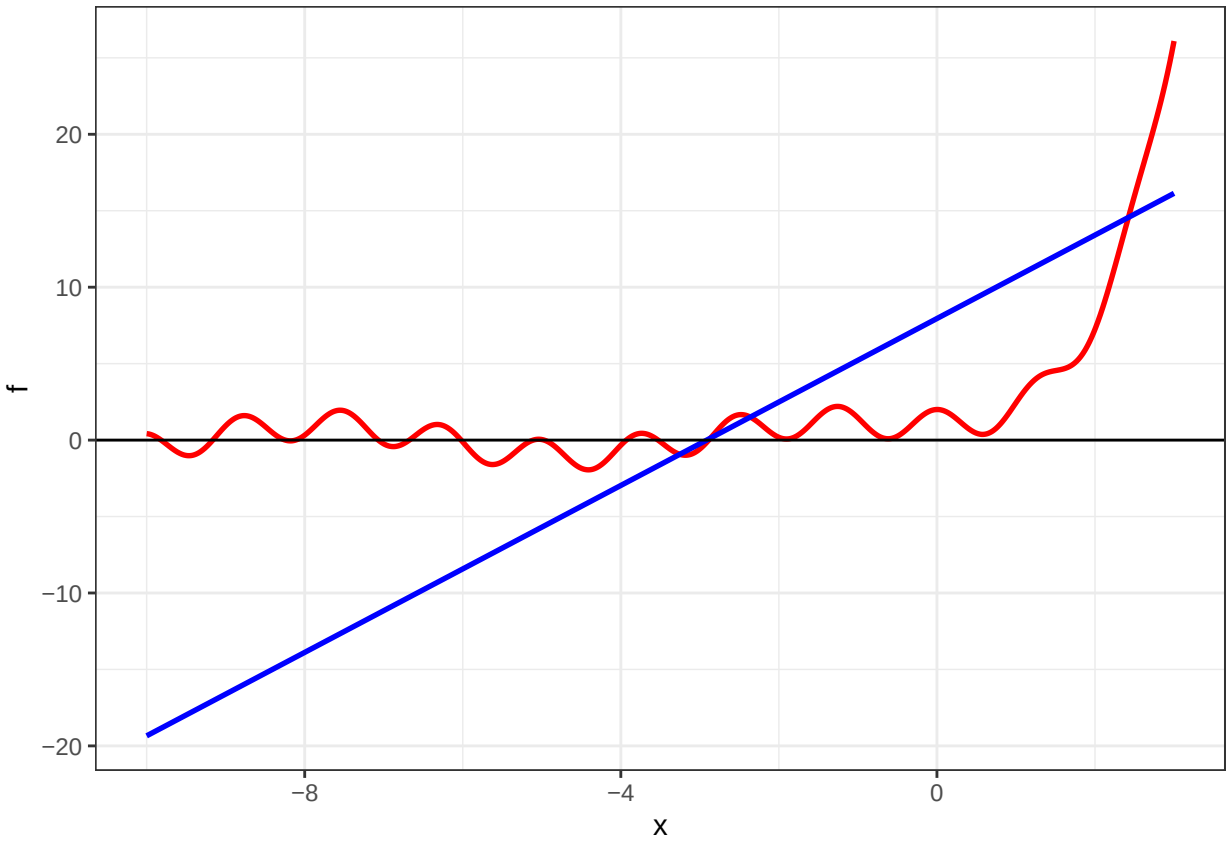


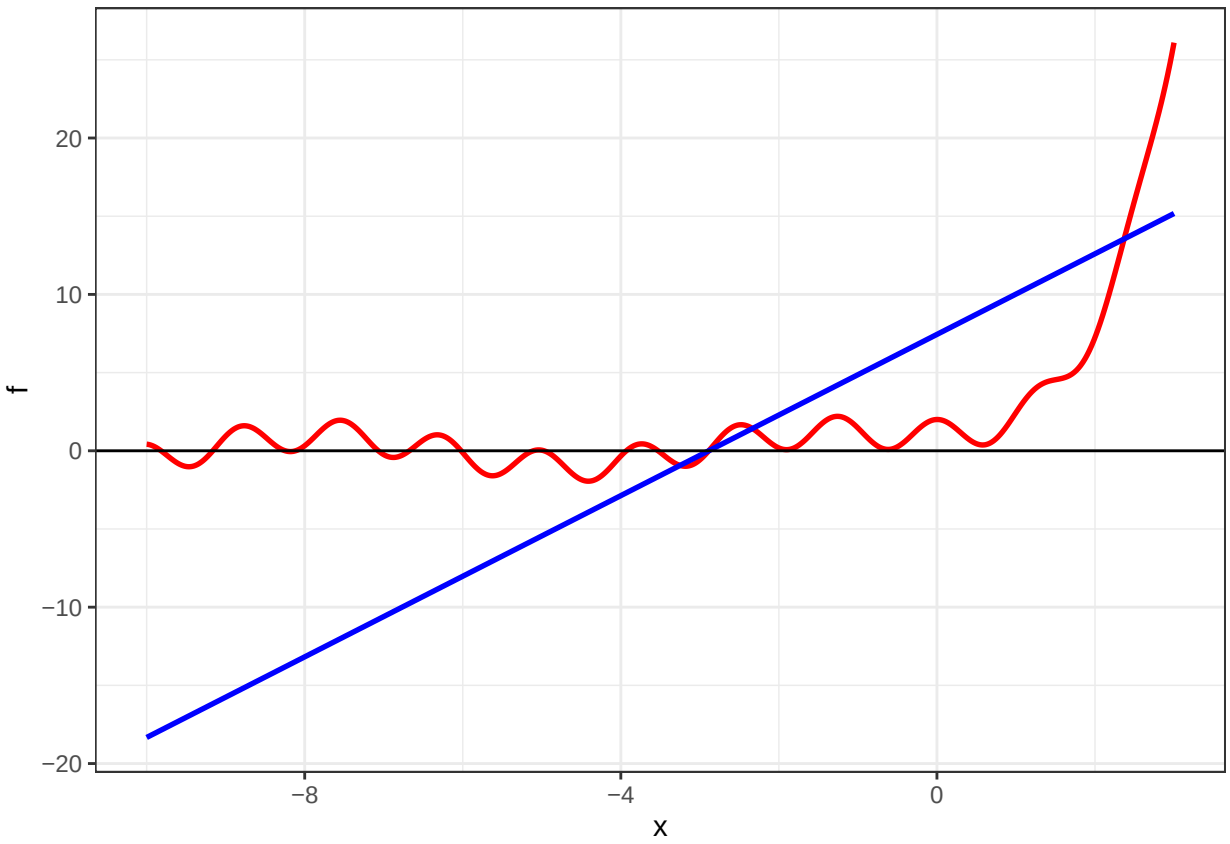

```r
# create a function of newton algorithm
newton <- function(f, tol=1E-12,x0=-3,N=50) {
  h <- 0.001
  i <- 1; x1 <- x0
  p <- numeric(N)

  while (i<=N) {
    df.dx <- (f(x0+h)-f(x0))/h
    x1 <- (x0 - (f(x0)/df.dx))
    p[i] <- x1
    i <- i + 1
  if (abs(x1-x0) < tol) break
    x0 <- x1
  }
  return(p[1:(i-1)])
}

p<-newton(f)
```

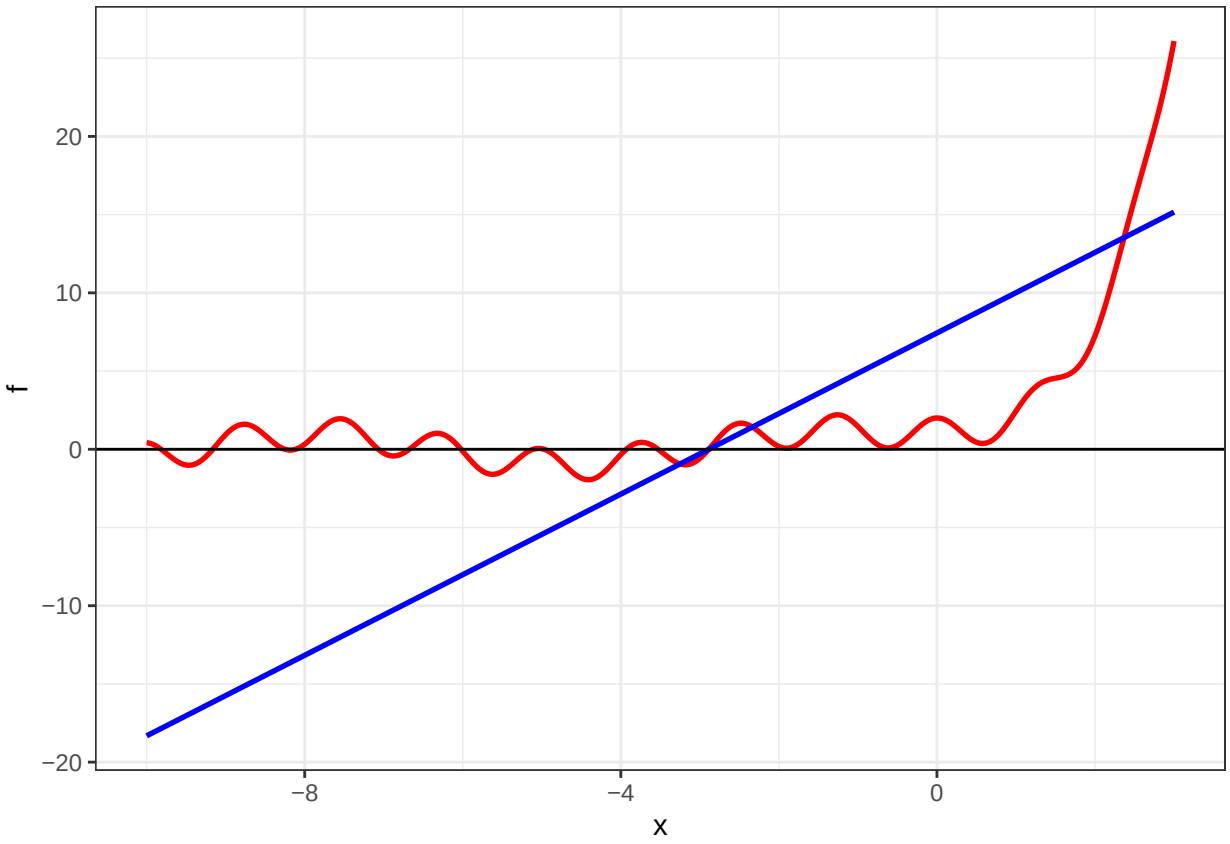
```r
# create a input (x) data
plot_data<-as.data.frame(x=seq(-10,3,by=0.01))
names(plot_data)<-c("x")

# calculate f(x) and add the column of f(X) into data
plot_data$f <- sapply(plot_data$x, function(x) 3**x-sin(x)+cos(5*x))

# create a name list for derivative function "drevFunc1, drevFunc2,...
nameList<-c()
for(i in 1:length(p)){
nameList[i]<-paste("drevFunc", i, sep = "")
}

# create a tangent line function at some point a
```

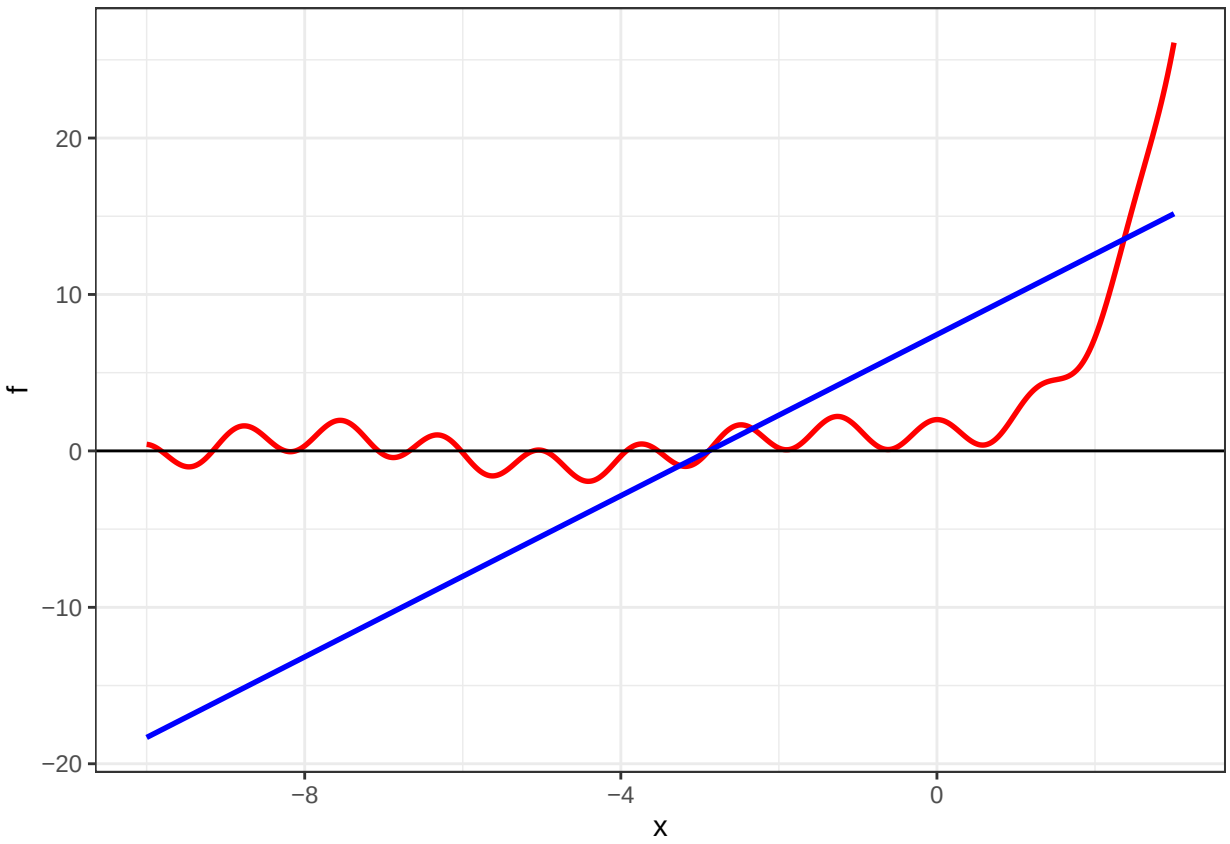
```
f_tangent <- function(x, f, df, a){
  # We need the function and its derivative df
  # a is the point where we create the tangent to the function
  # returns the tangent line
  df(a)*(x - a) + f(a)
}

drevFunc<-list()
for(i in 1:length(p)){

  # calculate values of a tangent line function for each iteration of Newton Method and save it into li
  drevFunc[[i]] <- sapply(plot_data$x, f_tangent, f = function(x) 3**x-sin(x)+cos(5*x),
                          df = function(x) x*log(3)-cos(x)-5*sin(5*x) , a = p[i])
  assign(paste("drevFunc", i, sep = ""), unlist(drevFunc[[i]]))

  # add the column of derivative function values for each loop into plot_data
  # data will consist of x,f(x),and f'(x) for each loop
  plot_data<-cbind(plot_data,eval(parse(text =paste0("drevFunc",i))))

  # change the column names for the columns of derivative function values.
  names(plot_data)[i+2]<-nameList[i]

}
```

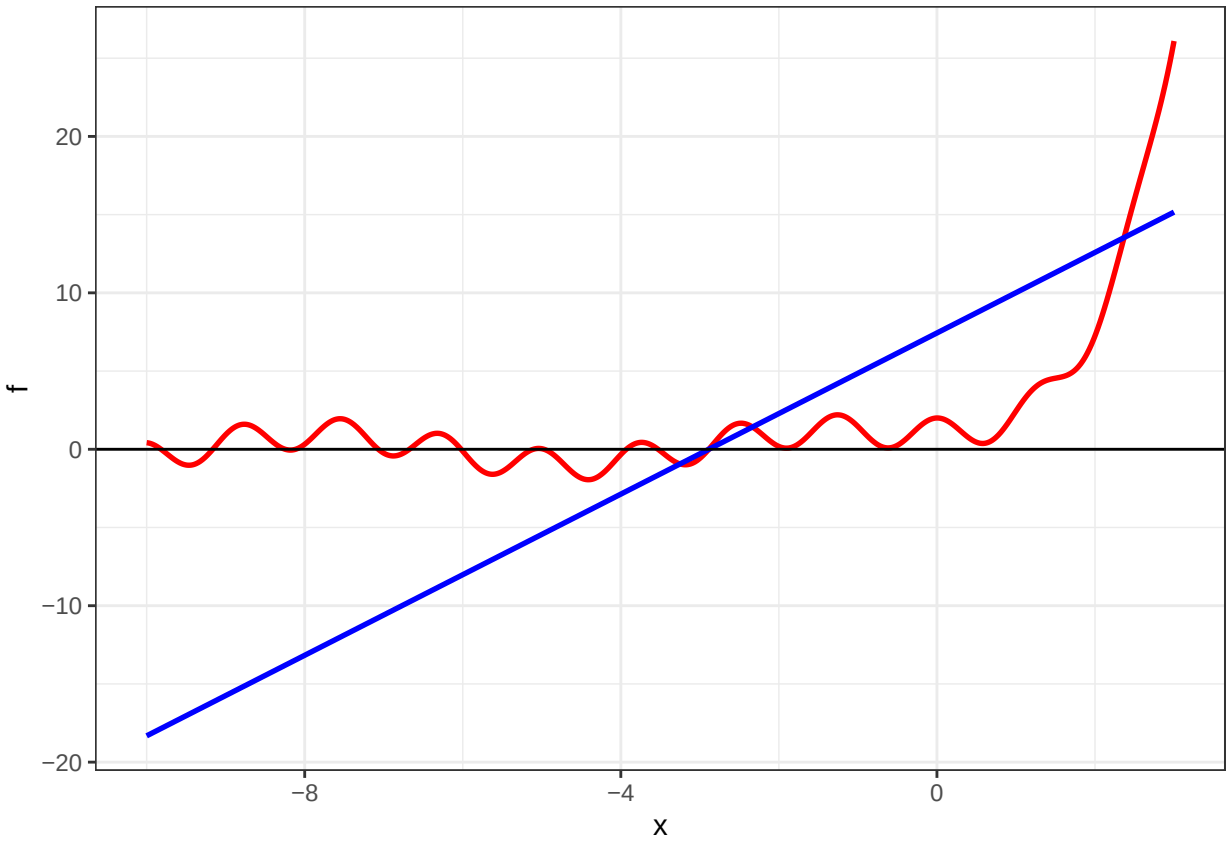
```
# loop through ggplot function. create ggplot for each iteration of Newton Method.
NMplot<-function(i){
pl<-ggplot(plot_data, aes(x = x)) +
  geom_line(aes(y = f), color = "red", size = 0.95) +
  geom_line(aes(y = eval(parse(text = paste0("drevFunc",i)))), color = "blue", size = 0.95) +
   # coord_cartesian(xlim = c(0,2), ylim = c(-5, 10)) +
  geom_hline(yintercept=0)+
  theme_bw()
  # labs(x = "x", y = "f(x) and tangent line")

return(pl)
}


for(i in 1:length(p)){
NMpl<-NMplot(i)
print(NMpl)
}
```

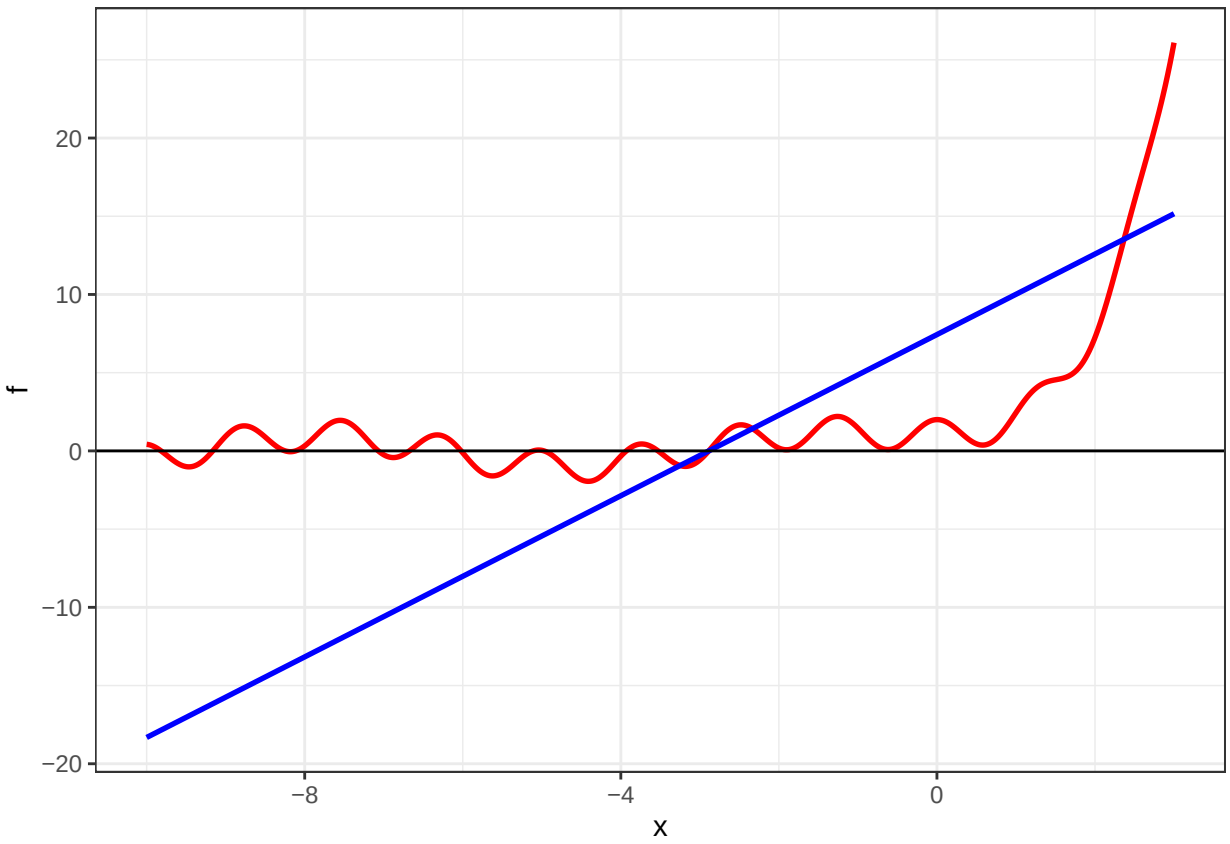

20
10
0
−10
−20
f
−8
−4
0
x


20
10
0
−10
−20
f
−8
−4
0
x


20
10
0
−10
−20
f
−8
−4
0
x


20
10
0
−10
−20
f
−8
−4
0
x


20
10
0
−10
−20
f
−8
−4
0
x


20
10
0
−10
−20
f
−8
−4
0
x

Problem 8 —-

```
set.seed(123)
ss <- rep(NA,100)
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
ybar <- mean(y)
mat1 <- rep(1,100)
```

accumulating value in a for loop

```
loopOp <- function(y){
for(i in 1:100){
  ss[i] <- (y[i] - ybar)**2
}
  sst <- sum(ss)
return(sst)
}
```

matrix operation only

```
matOp <- function(y){
sst <- mat1%*%(y - ybar)**2
return(sst)
}
```

```
# calculate SST using loop
loopOp(y)
```

```
## [1] 20832.22
```

```
# calculate SST using matrix
matOp(y)
```

```
##          [,1]
## [1,] 20832.22
```

```
# timing comparison between loop and matrix operation
microbenchmark(loopOp(y),matOp(y),times=100)
```

```
## Unit: microseconds
##       expr    min     lq     mean  median      uq     max neval
##  loopOp(y) 11.128 11.481 11.78113 11.6655 11.9260   18.03   100
##   matOp(y)  1.476  1.654 28.39141  1.7915  1.9075 2660.51   100
```

The final value for SST is 20832.22. The time it took for the loop procedure is about 17.5 seconds. The time it took for the matrix procedure is about 2.4 seconds.