Homework3_zhiyuan

Zhiyuan Du

9/21/2020

Problem 3

The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. There are a few suggestions about this:

1. Identifying functions with BigCamelCase to clearly distinguish them from other objects.

2. Naming objects with dot.case. do not support using right-hand assignment.

3. It is better to be clear about your intent to return() an object.

Problem 5

First, let's import the data into R.

```
data5 = readRDS("/cloud/project/data5.RDS")
saveRDS(data5,"data5.RDS")
data5=readRDS("data5.RDS")
```

a. After loading the data into R, Let's write the function.

```
Summary_table=function(data5, table_show=TRUE){
  # Define the final return table
  table_p5=matrix(NA,nrow=13,ncol=6)
  for (i in 1:13) {
     table_p5[i,1]=i
     # Calculate the mean of col1
     table_p5[i,2]=mean(data5[data5$Observer==i,2])
     # Calculate the mean of col2
     table_p5[i,3]=mean(data5[data5$Observer==i,3])
     # Calculate the sd of col1
     table_p5[i,4]=sd(data5[data5$Observer==i,2])
     # Calculate the sd of col2
     table_p5[i,5]=sd(data5[data5$Observer==i,3])
     # Calculate the correlaton of col1 and col2
     table_p5[i,6]=cor(data5[data5$Observer==i,2],data5[data5$Observer==i,3])
  }

  # Create the names for the variables
  colnames(table_p5)=c("Observer","mean_dev1","mean_dev2","sd_dev1","sd_dev2","correlation")

  # Show the table
```

```
  if(table_show==TRUE){
   kable(table_p5)
 }
}

Summary_table(data5,table_show=TRUE)
```

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	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

Now, let's do a check about the data we created by the dplyr's package.

```
check=data5 %>%
  group_by(Observer) %>%
  summarise(mean_dev1=mean(dev1),mean_dev2=mean(dev2),
            sd_dev1=sd(dev1),sd_dev2=sd(dev2),correlation=cor(dev1,dev2))
kable(check)
```

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	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

After the check by the dylyr's package, the table we created by the loop is correct. From the table, we can easily conclude that for each device, the means and variance among 13 observers' measurements are so close. By the correlation we got, they are all close to 0, which means there is no linear correlation between them.

b. Let's make the boxplots for the data.

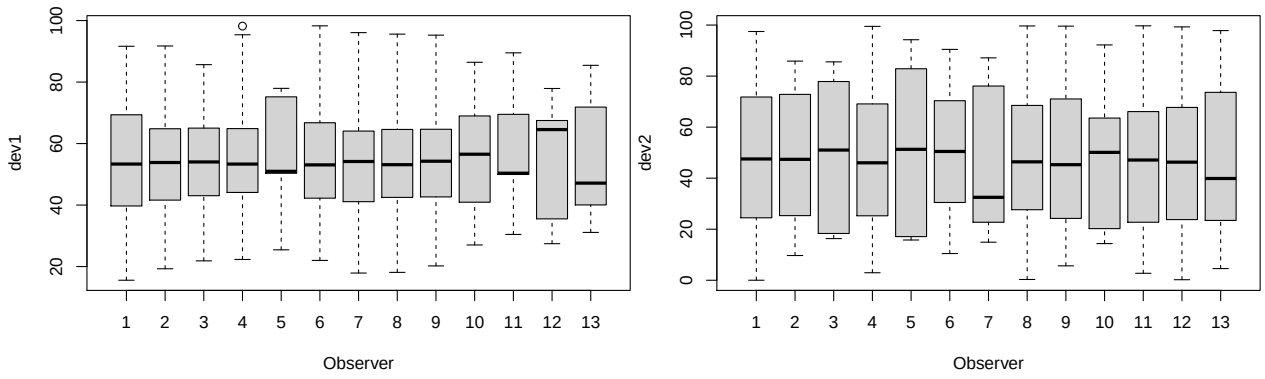


Figure 1: The boxplots for dev1 and dev2

From the plots for device1 and device2, the ranges of each device by the observer are kind of similar, but the meadian and the quantiles are kind of different among the observers. Some are skewed and some are not. What their distributions are is needed to be explored.

c. Let's make the violin plots for the data.

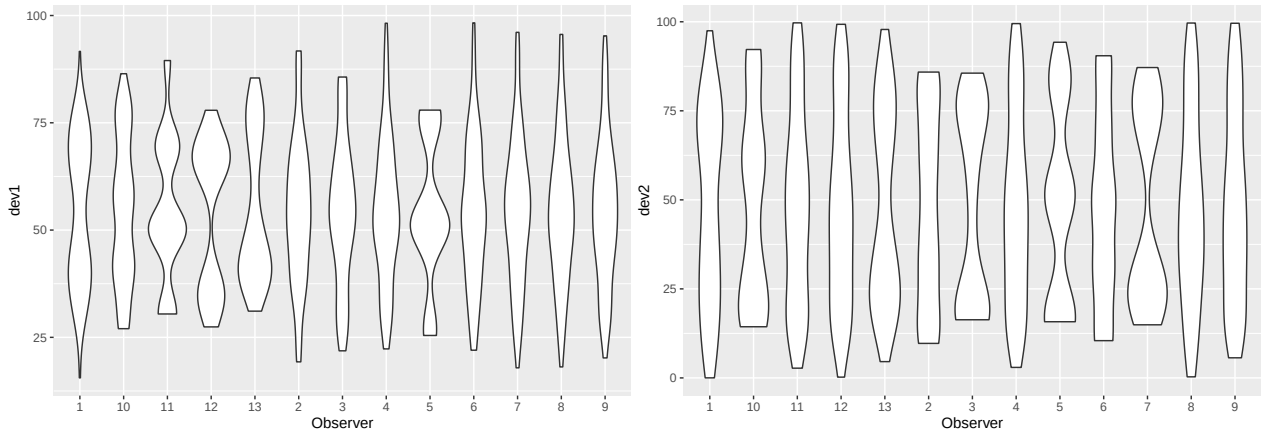
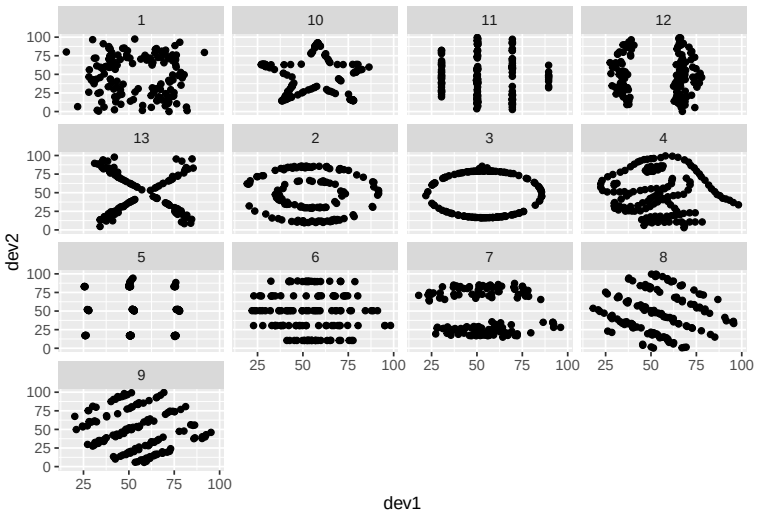


Figure 2: The violinplots for dev1 and dev2

By the violin plots above, we can vividly see the differences of their distributions among the observers, which is hard to achieved by the summary statistics we got from the part(a) and the plots we got in part(b).

Thus, when we are trying to summarize a data, it had better for us to analyze not only by the summary statistics, but also make the data into graphics to explore the detailed imformation, like its distribution.

d. Let make a scatter plot.



The scatter plots we got are kind of surprising with some shapes like dinosaurs and some like star. After operating all the parts of this question. There are a few suggestions when we are trying to approach a data and do the analysis.

- Check the data by making the scatter plot or histogram to have a approximate impression of the distribution.
- After getting the summary statistics, try to make graphics like boxplots and violin plots to make the summary statistics easier to tell by eyes.

- Do not make conclusions in rush without doing the above steps.

Problem 6

From the question, we are using the Reimann sums to approximate the integral:

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

Let's try different slice width from large to small.

1. When the slice width is 0.01, there would be 100 slices.

```
# Write a function on calculating the Reimann sums to approximate
# and calculate the gap between the real one.
# x and y are the left and right region to do the integral
integral=function(s=0.01,x=0,y=1){

  # integral1 is the area we are about to sum, starting from 0.
  integral1=0

  # Loop by the Reimann sums
  for(i in 1:((y-x)/s)){
 integral1=s*exp(-(x+s*i)^2/2)+integral1
  }
```

```
  # Calculate the real integral
  fun1=function(x) exp(-x^2/2)

  # The gap between the real and the one we got by Reimann sums
  gap=area(fun1,x,y)-integral1
  result=c(integral1,gap)
  return(result)
}

# Get the integral with slice as 0.01 and input the x-axis from 0 to 1.
integral(0.01, 0, 1)
```

```
## [1] 0.853651991 0.001972401
```

From the reaults, the value we approximated is 0.853652, which is a little "far" away of the real integral value. Now let's shrink the slice to 0.0001 to approximate the integral.

```
# Get the integral with slice as 0.0001 and input the x-axis from 0 to 1.
integral(0.0001, 0, 1)
```

```
## [1] 8.556047e-01 1.967373e-05
```

From the reaults, the value we approximated is 0.8556047, which is closer the real integral value but the gap is still larger than $1e^{-6}$. Now let's shrink the slice to 0.000001 to approximate the integral.

```
# Get the integral with slice as 0.000001 and input the x-axis from 0 to 1.
integral(0.000001, 0, 1)
```

```
## [1] 8.556242e-01 1.964971e-07
```

From the reaults, the value we approximated is 0.8556242, which is extremely closer to the real integral value and the gap is still smaller than $1e^{-6}$, which is $1.96e^{-7}$.

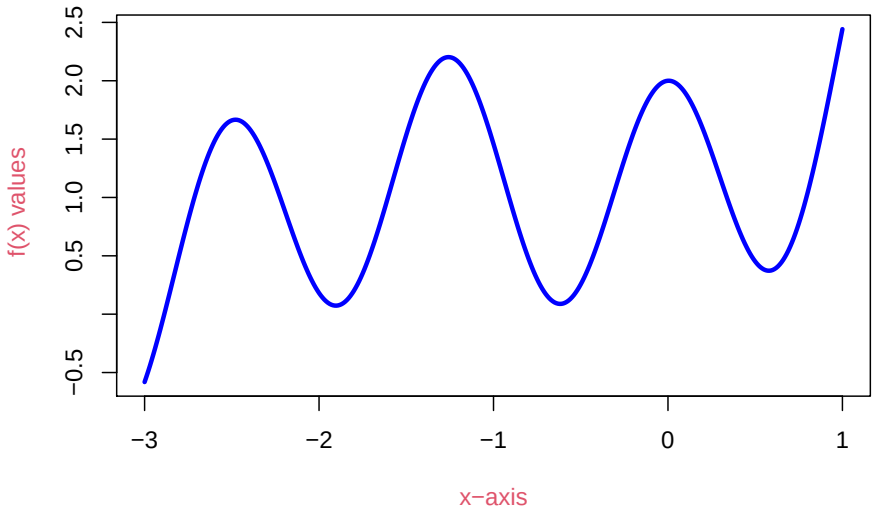
Problem 7

Newton's method for solving equations is another numerical method for solving an equation f(x)=0. We seek a solution of f(x)=0, starting from an initial estimate $X = x_1$. At the n'th step, given x_n, compute the next approximation x_{n+1} by:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Note that if $f(x_n) = 0$, so that x_n is an exact solution of $f(x) = 0$. From the question, $f(x) = 3^x - sin(x) + cos(5x)$, then $f'(x) = 3^x ln3 - cos(x) - 5sin(5x)$. Let's write the function to get the solution.

```
# Define the f(x)
fun1=function(x) 3^x-sin(x)+cos(5*x)

# The derivative of f(X)
fun_de=function(x) 3^x*log(3)-cos(x)-5*sin(5*x)

# Have a see about the plot of f(X)
plot(seq(-3,1,0.01),fun1(seq(-3,1,0.01)),type = "l",xlab = "x-axis",
     ylab=" f(x) values", lty=1, lwd=3, col="blue",col.lab=2)
```

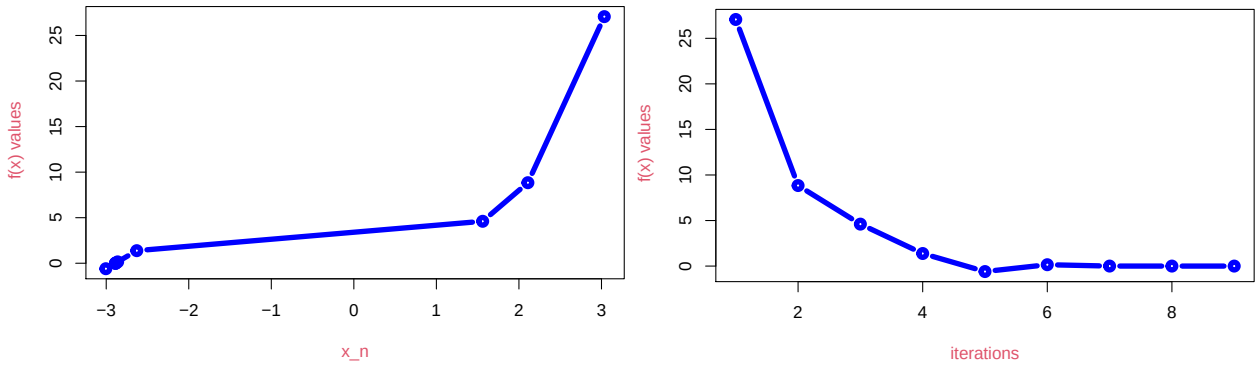



```
# Define the solution
solution= function(x1=4, draw=TRUE, report=TRUE){

  # Define the matrix with NA values firstly
  n=2
  x=matrix(NA,nrow=25, ncol=4)
  x0=matrix(c(1,2,3,4,1,2,4,3),nrow=2,byrow = TRUE)
  x=rbind(x0,x)
  x_n=pi

  # When there is few difference between the X_n and X_n-1,
  # it means x converges, x_n is the solution.
  while(abs(x[n,3]-x[n-1,3])>0.000001){
   x_n=x1-fun1(x1)/fun_de(x1)
   n=n+1
   x[n,]=c(n,x1,x_n,fun1(x_n))
   x1=x_n
   }
  x_solutions=as.data.frame(cbind(seq(1,23,1),x[3:25,]))[1:(n-2), -2]
  colnames(x_solutions)=c("iterations","x_n-1","x_n","f(x)")

  # draw the plots for x_n and iterations as x-axis.
  if(draw==TRUE){
    plot(x_solutions$x_n, x_solutions$'f(x)',type = "b",xlab = "x_n",
         ylab=" f(x) values", lty=1, lwd=5, col="blue",col.lab=2)
    plot(x_solutions$iterations, x_solutions$`f(x)`,xlim=c(1,9),type = "b",
         xlab = "iterations", ylab=" f(x) values", lty=1, lwd=5, col="blue",col.lab=2)

  # Report the results
  if(report==TRUE){
   cat("Solution is x_n =",x_solutions[n-2,3],"iterations are", x_solutions[n-2,1])
  }
  }
return(x_solutions)

}
```

```r
solution(4,draw = TRUE,report = TRUE)
```

```
## Solution is x_n = -2.887058 iterations are 9
```

```
##   iterations     x_n-1       x_n          f(x)
## 1          1  4.000000  3.034224  2.706800e+01
## 2          2  3.034224  2.108383  8.840846e+00
## 3          3  2.108383  1.560762  4.605056e+00
## 4          4  1.560762 -2.629876  1.380105e+00
## 5          5 -2.629876 -3.004324 -5.998704e-01
## 6          6 -3.004324 -2.861486  1.501408e-01
## 7          7 -2.861486 -2.886779  1.615767e-03
## 8          8 -2.886779 -2.887058  2.767141e-07
## 9          9 -2.887058 -2.887058  7.327472e-15
```



Problem 8

To do the sum of squares, le's do the calculation by the loop firstly.

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

# Write the loop to calculate the sum of squares.
SumSquares = function(y){
  results=matrix(NA,nrow = 100)

  # Use the loop
  for(i in 1:100){
    results[i,] = (y[i,]-colMeans(y))^2
  }
  results = sum(results)
  return(results)

}

cat("The Sum of Squares is", SumSquares(y))
```

```
## The Sum of Squares is 20407.87
```

Now let's do the calculation by matrix operations.

```
SosMatrix = function(y){

  # Transpose the matrix then times it to get its sums of squares.
  t(y-colMeans(y)) %*% (y-colMeans(y))

}

cat("The Sum of Squares is", SosMatrix(y))
```

```
## The Sum of Squares is 20407.87
```

Now use the microbenmark to get the timmings of the functions.

```
microbenchmark(result1<-{SumSquares(y)},
               result2<-{SosMatrix(y)},times = 100, unit = "ms")
```

```
## Unit: milliseconds
##                              expr      min        lq      mean    median
##  result1 <- {     SumSquares(y) } 0.444913 0.4561180 0.4899261 0.4689405
##   result2 <- {     SosMatrix(y) } 0.011890 0.0132985 0.0442089 0.0141960
##         uq      max neval
##  0.5074815 0.826315   100
##  0.0179520 2.853739   100
```

```
 max(result1 - result2)
```

```
## [1] 7.275958e-12
```

From the report, the timming of doing by loop is longer than just doing matrix operations.