HW3_tdolkar

Due Wednesday Sep 30

Tsering Dolkar

9/24/2020

Problem 1

I did primer on Rstudio cloud.

Problem 2

Created the Rmd file

Problem 3

Problem 4

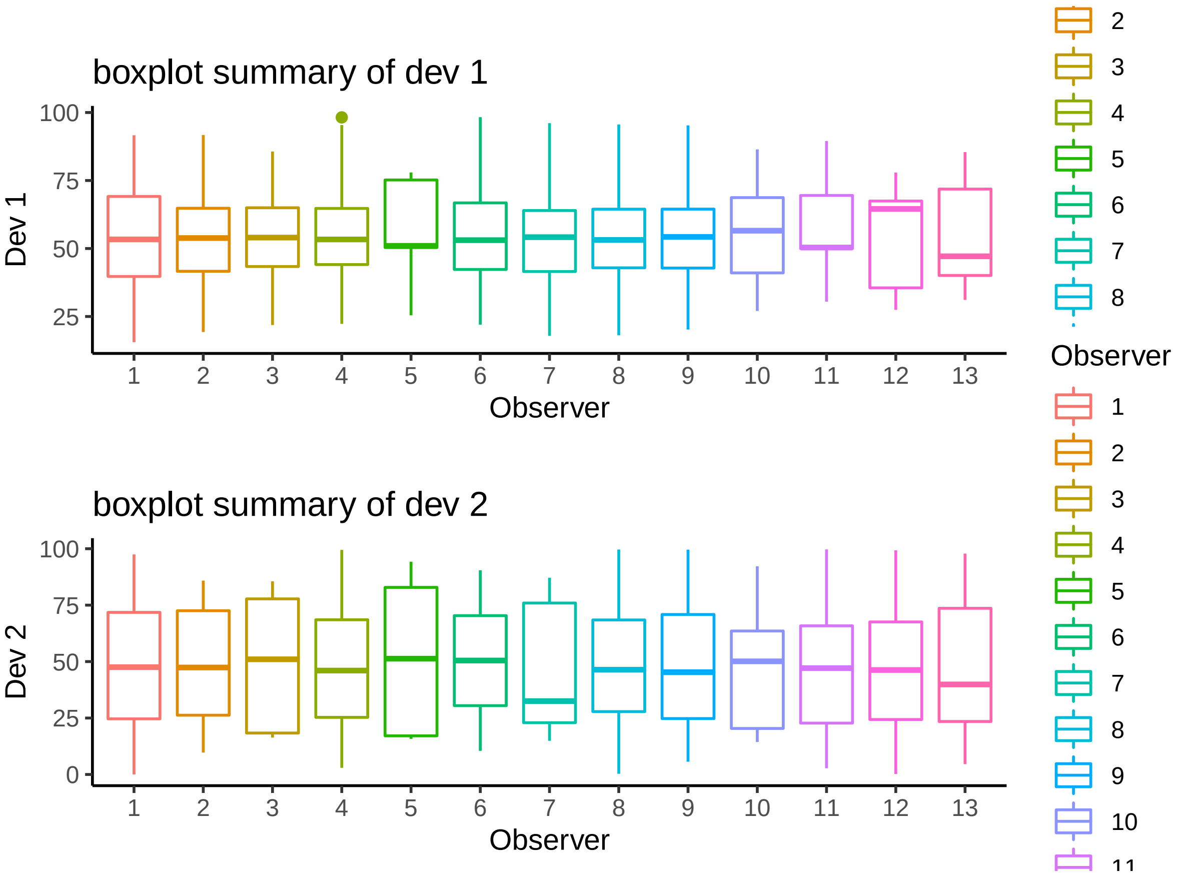
Problem 5

#dplyr + summarize + group_by

```
summarise_data <- function(our_data){
  #function to calculate summary of a dataframe, returns a vector of the summary
  observer_summary <- double(length = 5)
  for(i in 1:5){
    if(i == 1)
      observer_summary[i] <- mean(our_data[,1])
    if(i == 2)
      observer_summary[i] <- mean(our_data[,2])
    if(i == 3)
      observer_summary[i] <- sd(our_data[,1])
    if(i == 4)
      observer_summary[i] <- sd(our_data[,2])
    if(i == 5)
      observer_summary[i] <- cor(our_data[,1], our_data[,2])
  }
  return(observer_summary)
}
```

```
# We'll find the summary for each observer:
# We will be returned a vector of length 5 for each of 13 observers from the function
# summarise_data such that
# 1. mean of dev 1
# 2. mean of dev 2
# 3. standard dev of dev 1
# 4. standard dev of dev 2
# 5. correlation between dev 1 and 2
# are the values respectively in a row.

#separate the data into vectors to make it less confusing to work with first
observer <- observations$Observer
dev1 <- observations$dev1
dev2 <- observations$dev2

#initialize vectors we will need:
dev1_by_observer <- double(0)
dev2_by_observer <- double(0)
summary_statistics <- data.frame()
colnames(summary_statistics) <-

#we have a nested for loop here. The outer for loop keeps track of observer 1 to 13 and
#the inside for loop looks for all the data by the said observer in the dataset from top to bottom
#once.
for(i in 1:13){
   track_observer <- i
   for(j in 1:length(observer)){
     if(track_observer == observer[j]){
     dev1_by_observer <- c(dev1_by_observer, dev1[j])
     dev2_by_observer <- c(dev2_by_observer, dev2[j])
     }
   }
   raw_data <- data.frame(dev1_by_observer, dev2_by_observer)
   colnames(raw_data) <- c("dev1", "dev2")
   summary_statistics_each_observer <- cbind(rep(i, 5), summarise_data(raw_data))
   summary_statistics <- rbind(summary_statistics, summary_statistics_each_observer)
}
summary_statistics <- data.frame(rep(c("Mean_dev1", "Mean_dev2", "SD_dev1", "SD_dev2",
                                       "Cor_dev1_dev2"), 13), summary_statistics)
colnames(summary_statistics) <- c(’V1’,’Observer’,’V3’)
summary_statistics <- summary_statistics %>%
                          spread(key = V1, value = V3)
# reorder by column name
summary_statistics <- summary_statistics[c("Observer", "Mean_dev1", "Mean_dev2",
                                             "SD_dev1", "SD_dev2", "Cor_dev1_dev2")]
summary_statistics <- kable(summary_statistics)
```

```
observations$Observer <- as.factor(observations$Observer)

dev1 <- observations %>%
  ggplot(aes(x = Observer, y = dev1, colour = Observer)) +
  geom_boxplot() +
  labs(title = "boxplot summary of dev 1", x="Observer", y = "Dev 1") +
  theme_classic()

dev2 <- observations %>%
  ggplot(aes(x = Observer, y = dev2, colour = Observer)) +
  geom_boxplot() +
  labs(title = "boxplot summary of dev 2", x="Observer", y = "Dev 2") +
  theme_classic()

figure1 <- multi_panel_figure(columns = 1, rows = 2, panel_label_type = "none")
figure1 %<>%
  fill_panel(dev1, column = 1, row = 1) %<>%
  fill_panel(dev2, column = 1, row = 2)
figure1
```



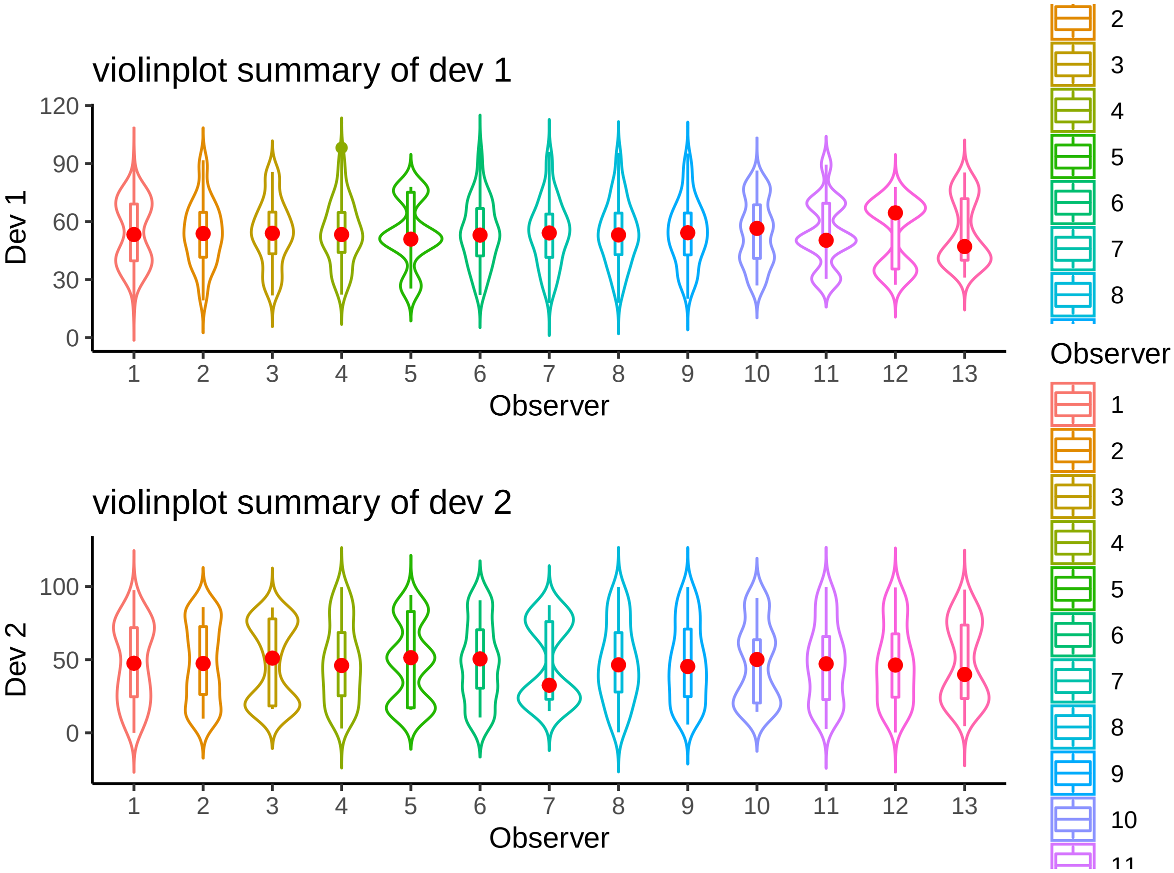
```
dev1 <- observations %>%
  ggplot(aes(x = Observer, y = dev1, colour = Observer)) +
  geom_violin(trim = F) +
  geom_boxplot(width=0.1) +
  stat_summary(fun.y=median, geom="point", size=2, color="red") +
  labs(title = "violinplot summary of dev 1", x="Observer", y = "Dev 1") +
  theme_classic()
```

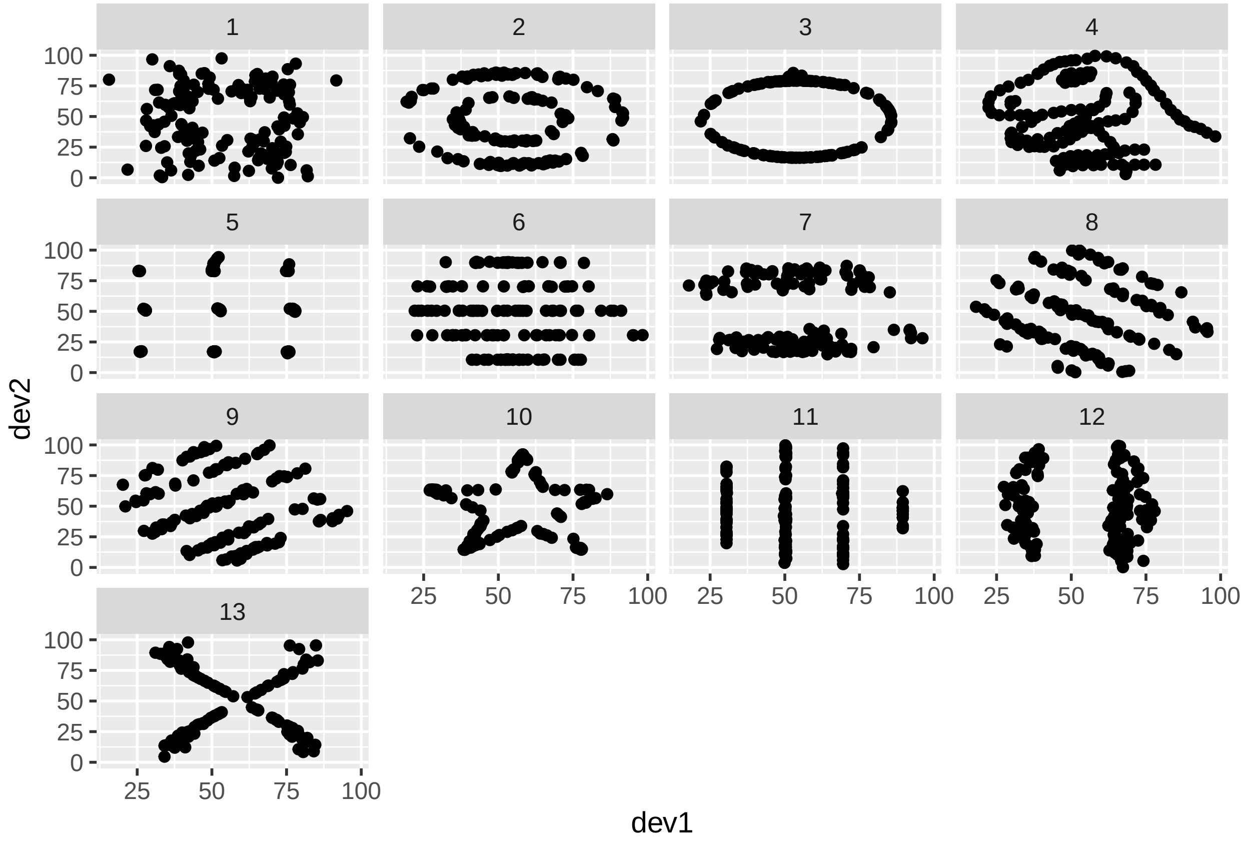
```
## Warning: ‘fun.y‘ is deprecated. Use ‘fun‘ instead.
```

```
dev2 <- observations %>%
  ggplot(aes(x = Observer, y = dev2, colour = Observer)) +
  geom_violin(trim = F) +
  geom_boxplot(width=0.1) +
  stat_summary(fun.y=median, geom="point", size=2, color="red") +
  labs(title = "violinplot summary of dev 2", x="Observer", y = "Dev 2") +
  theme_classic()
```

```
## Warning: 'fun.y' is deprecated. Use 'fun' instead.
```

```
figure1 <- multi_panel_figure(columns = 1, rows = 2, panel_label_type = "none")
figure1 %<>%
  fill_panel(dev1, column = 1, row = 1) %<>%
  fill_panel(dev2, column = 1, row = 2)
figure1
```





Problem 6

```
R_sum <- function(a, b, n){
  x <- double(length = n)
  dx <- (b - a)/n
  for(i in 1:n){
    x[i] <- a + dx*i
  }
  return(sum(dx * exp(-x^2/2)))
}
```

```
# how_many_parts <- seq(from = 1, to = 1000000, by = 1)
# a <- 0
# b <- 1
# track_RS <- data.frame()
# report <- data.frame()
#
# integrand <- function(x) {exp((-x^2)/2)}
# integral_value <- integrate(integrand, lower = 0, upper = 1)
# integral_value <- integral_value$value
#
# # put length of seq instead of 5
# for(i in 1:length(how_many_parts)){
#   RS <- R_sum(a, b, how_many_parts[i])
#   slice_width <- ((b-a)/how_many_parts[i])
#   track_RS <- rbind(track_RS, c(slice_width, RS))
#   # if riemann sum is within e^(-6) points from the integral value, store it in
#   # dataframe report.
#   if(abs(RS - integral_value) <= 1e-06){
#   report <- rbind(report, c(slice_width, RS))
#   }
# }
# colnames(track_RS) <- c("slice_width", "Riemann_Sum")
# if(length(report) > 0){
#   colnames(report) <- c("slice_width", "Riemann_Sum")
# }
#
#
# #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.
#
# summary(report)
```

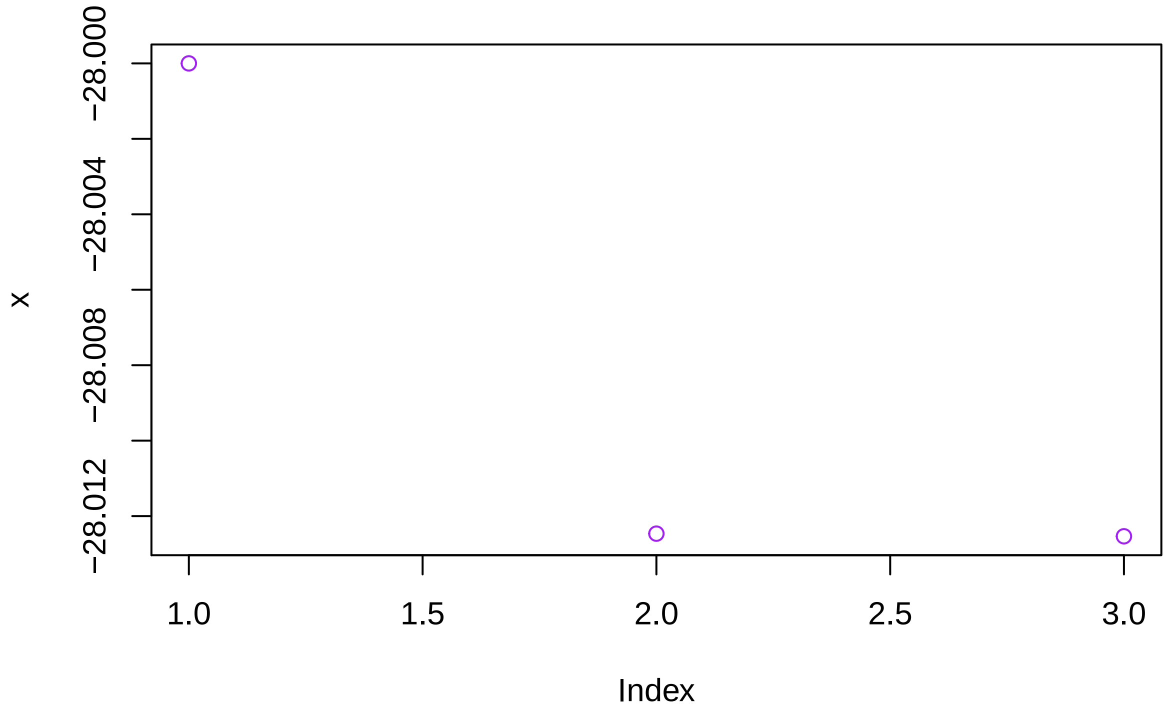
The width necessary to obtain an answer within $1e^{-6}$ of the analytical solution for me was the min value of slice_width. At that slice_width, I get the min value of Riemann_Sum. I could make the answer more accurate by increasing the to value in seq(how_many_parts).

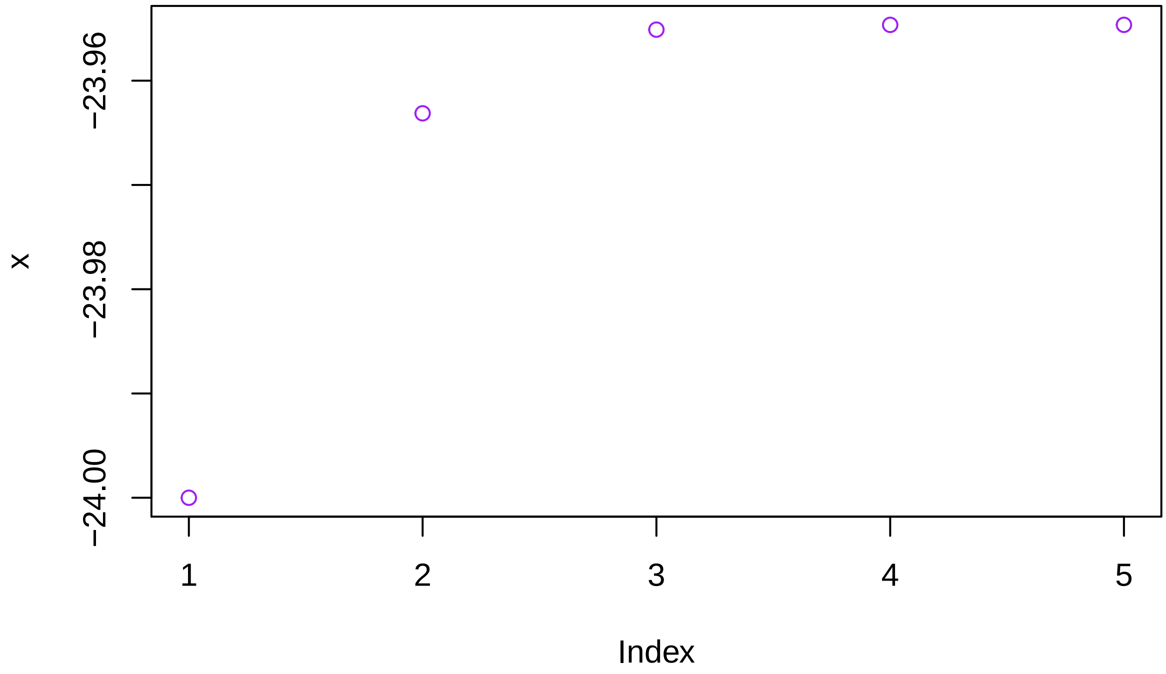
Problem 7

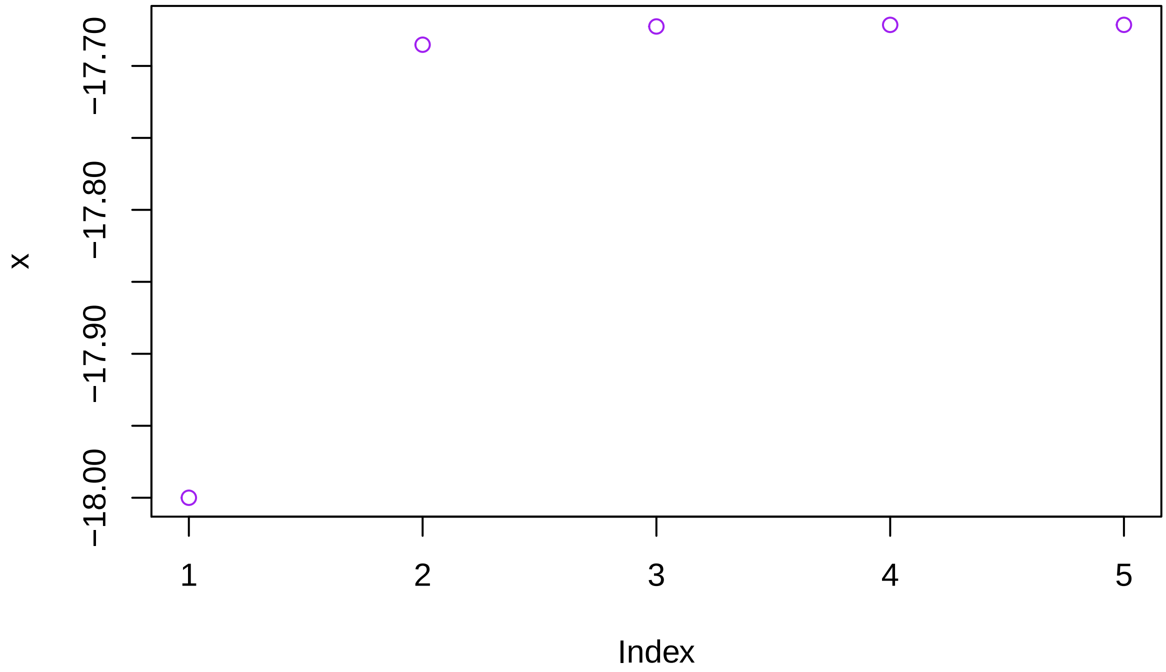
```
N_method <- function(x1 = -5){
  #function takes in a first approximation of the root x0 of the function 3^x-sin(x)+cos(5*x)
  #function uses Newton's method of approximate the root of the function
  #function terminates when successive estimates are within 1e-04 of each other
  seq <- 100
  x <- double(0)
  x[1] <-  x1
  # f <- 3^x- sin(x) + cos(5*x)
  # f_diff <- 3^x * log(3) - cos(x) - sin(5 * x) * 5
  for(i in 2:seq){
    if(3^x[i-1] * log(3) - cos(x[i-1]) - sin(5 * x[i-1]) * 5 == 0){
    stop("choose a new starting place")
    }
    x[i] <- x[i-1] - ((3^x[i-1]- sin(x[i-1]) + cos(5*x[i-1]))/
                          (3^x[i-1] * log(3) - cos(x[i-1]) - sin(5 * x[i-1]) * 5))
    if (abs(x[i]-x[i-1])<1e-04){
      break
    }
  }
  paste0("For the initial estimate of ", x[1])
  paste0("Xn approaches 0 at ", tail(x,1))
  plot(x, type = 'b', col = 'purple')
}
```

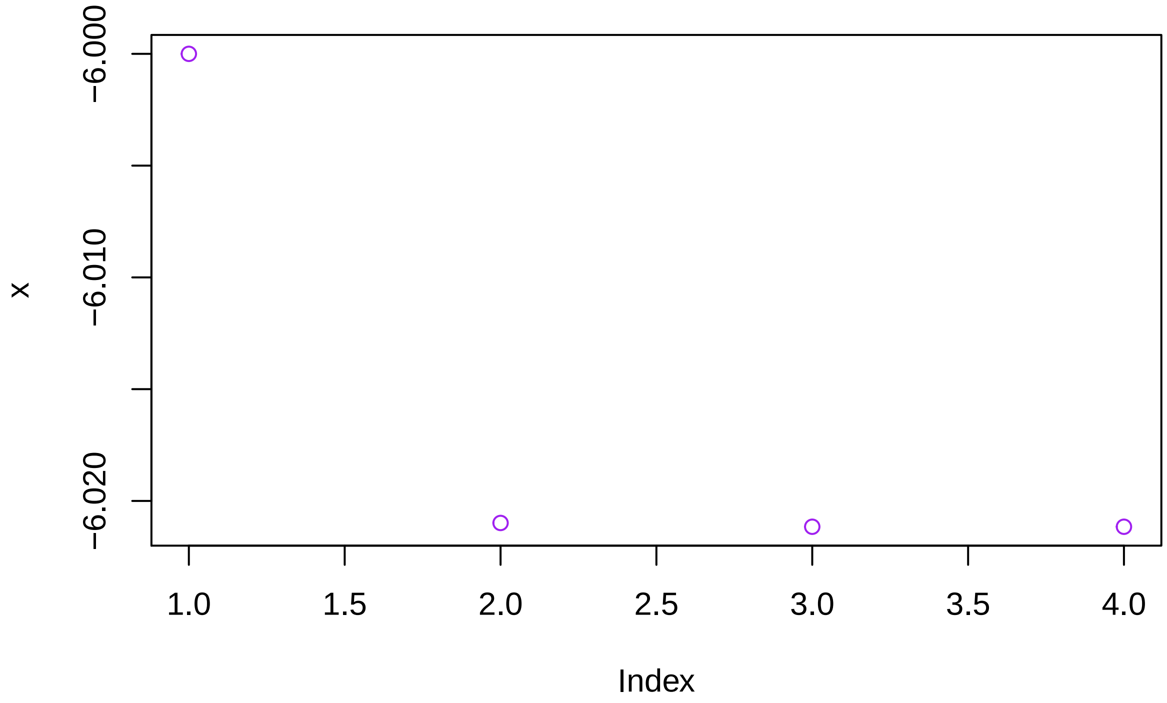
```
# we have:
#f <- expression(3^x- sin(x) + cos(5*x))
```

```
init_x <- c(-28, -24, -18, -6)
for(i in 1:length(init_x)){
  N_method(init_x[i])
}
```




x
−23.96
−23.98
−24.00
1
2
3
4
5
Index


−17.70
−17.80
−17.90
−18.00
x
1
2
3
4
5
Index



Problem 8

```
#Given simulated data from question
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)

# precalculation and vectors for SST using for_loop:
y_bar <- mean(y)
SST <- double(0)

# required values and matrices for SST using matrix operation:
n <- 100
J <- matrix(1, nrow = n, ncol = n)
I <- diag(x = 1, nrow = n, ncol = n)

times <- microbenchmark(SST1 <- {for(i in 1:length(y)){ SST[i] <- (y[i] - y_bar)^2};sum(SST)},
               SST2 <- {t(y)%*%(I - (1/n)*J)%*%y}, times = 100, unit = "ms")
print(times)
```

```
## Unit: milliseconds
##                                                                                              expr
##  SST1 <- {     for (i in 1:length(y)) {         SST[i] <- (y[i] - y_bar)^2      }     sum(SST) }
##                                                   SST2 <- {      t(y) %*% (I - (1/n) * J) %*% y }
##       min        lq       mean    median        uq       max neval
##  2.545715 2.7593890 3.37196252 2.8960875 3.2057695 16.500412   100
##  0.024941 0.0277965 0.04361442 0.0409165 0.0495545  0.105255   100
```

```
print(SST1)
```

```
## [1] 20429.84
```

```
print(SST2)
```

```
##          [,1]
## [1,] 20429.84
```