Homework 3

Steven Barnett

9/28/2020

Problem 3

Programming style guides are extremely helpful when collaborating and reading new code. In fact, one of the reasons I like the Golang programming language is because style guidelines are already decided and enforced by the compiler. Many of the style guidelines for programming in R are conventions that I follow. However, two guidelines I want to focus on are spacing (e.g. between assignment operators, function arguments, etc) and focusing my comments on why and not what.

Problem 5

First, we will define a function to read in a two column data frame and return summary statistics of and between the two columns.

```r
## Calculates summary statistics for each column in a two column dataset
## dataset: two column dataframe
## returns a vector containing:
##    - mean of column 1
##    - mean of column 2
##    - standard dev of column 1
##    - standard dev of column 2
##    - correlation between column 1 and 2
get_summary_stats <- function(dataset) {
  mean_col1 <- sum(dataset[[1]]) / length(dataset[[1]])
  mean_col2 <- sum(dataset[[2]]) / length(dataset[[2]])
  std_dev_col1 <- sd(dataset[[1]])
  std_dev_col2 <- sd(dataset[[2]])
  correlation_col1_col2 <- cor(dataset[1], dataset[2])
  return(c("Mean_Col1" = mean_col1, "Mean_Col2" = mean_col2,
          "Std_Dev_Col1" = std_dev_col1, "Std_Dev_Col2" = std_dev_col2,
           "Correlation_Cols1_2" = correlation_col1_col2))
}
```

Now, we will read in data containing repeated measurements from two devices by thirteen observers

```r
## url <- "https://github.com/rsettlage/STAT_5014_Fall_2020/blob/master
##  /homework/HW3_data.rds"
## device_data <- fread(url)
device_data <- readRDS("dwnldd_data/HW3_data.rds")
summary_data <- data.frame()
for (i in 1:13) {
  observer_rows <- select(filter(device_data, Observer == i), !Observer)
  observer_stats <- get_summary_stats(observer_rows)
  summary_data <- rbind(summary_data, c(i, observer_stats))
}
```

```r
colnames(summary_data) <- c("observer", "mean_device1", "mean_device2",
                            "std_dev_1", "std_dev_2", "cor_1_2")
```

After reading in the data, we will get a first look at the data by looking at key summary statistics.

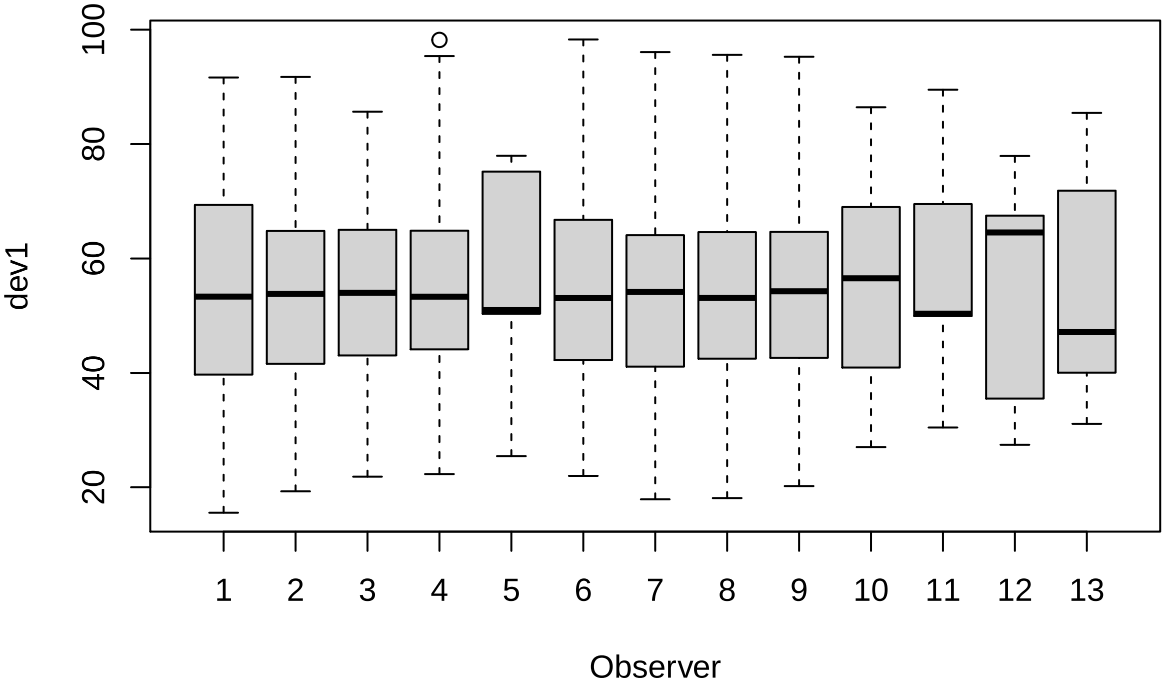
```r
knitr::kable(summary_data)
```

observer	mean_device1	mean_device2	std_dev_1	std_dev_2	cor_1_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

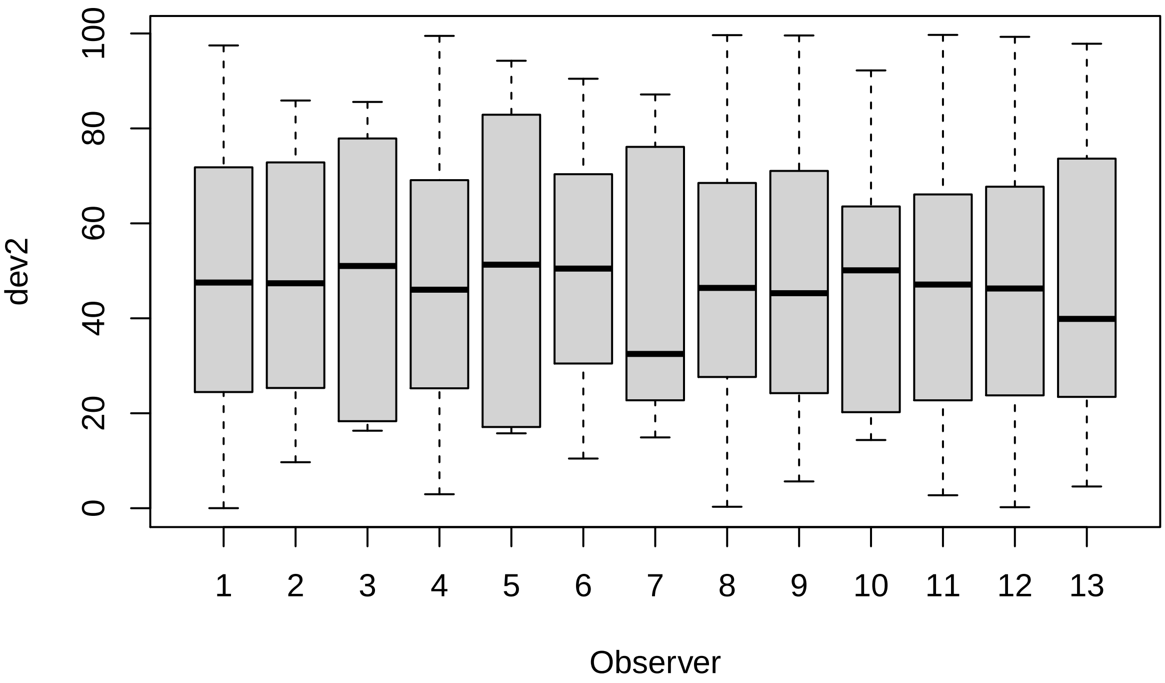
As this table shows, the data from each observer seems to be identically distributed. The means and standard deviations from each observer are virtually identical. After thinking about this, my conclusion is to be somewhat suspicious. I find it hard to believe that thirteen different observers would make measurements resulting in identical summary statistics.

Another way to view the data is through the use of box plots. This will show some key sample statistics as well as the general spread of the data.

```r
boxplot(dev1 ~ Observer, device_data)
```



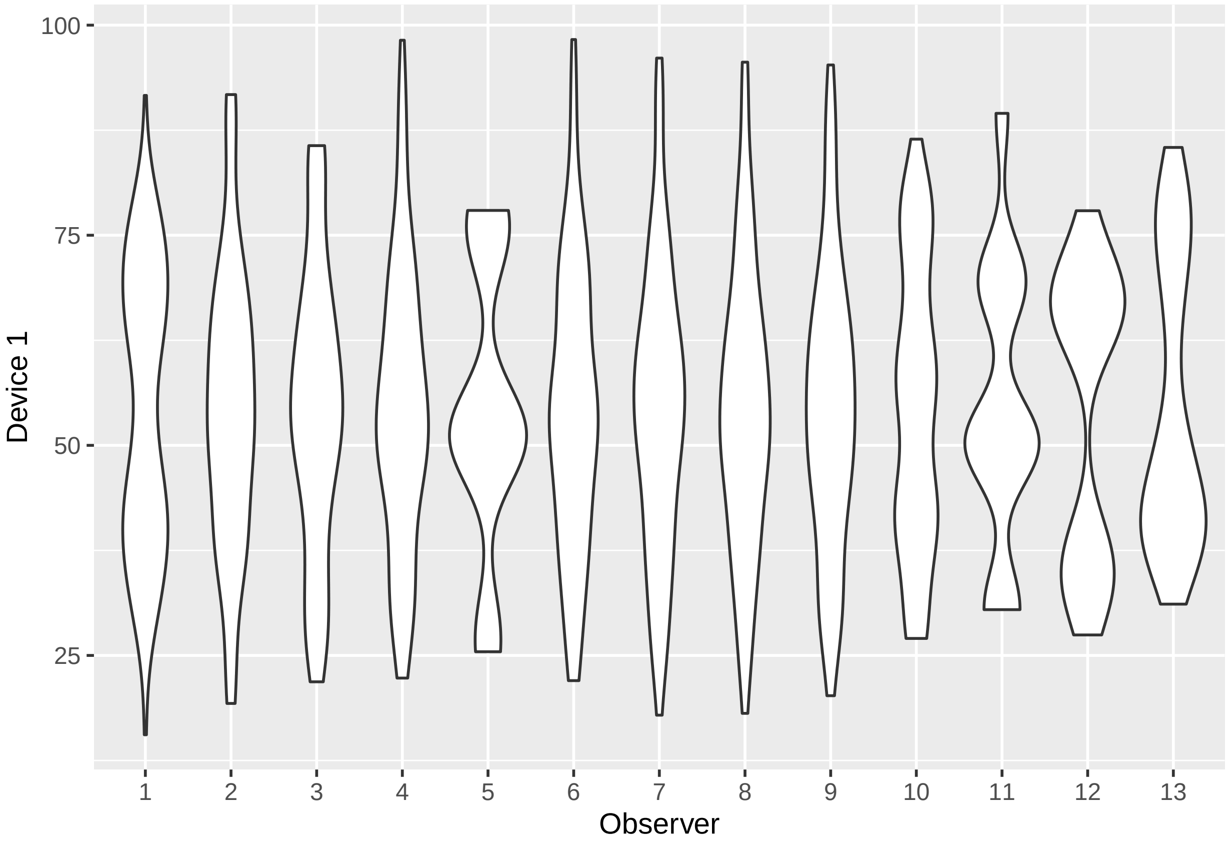
```
boxplot(dev2 ~ Observer, device_data)
```



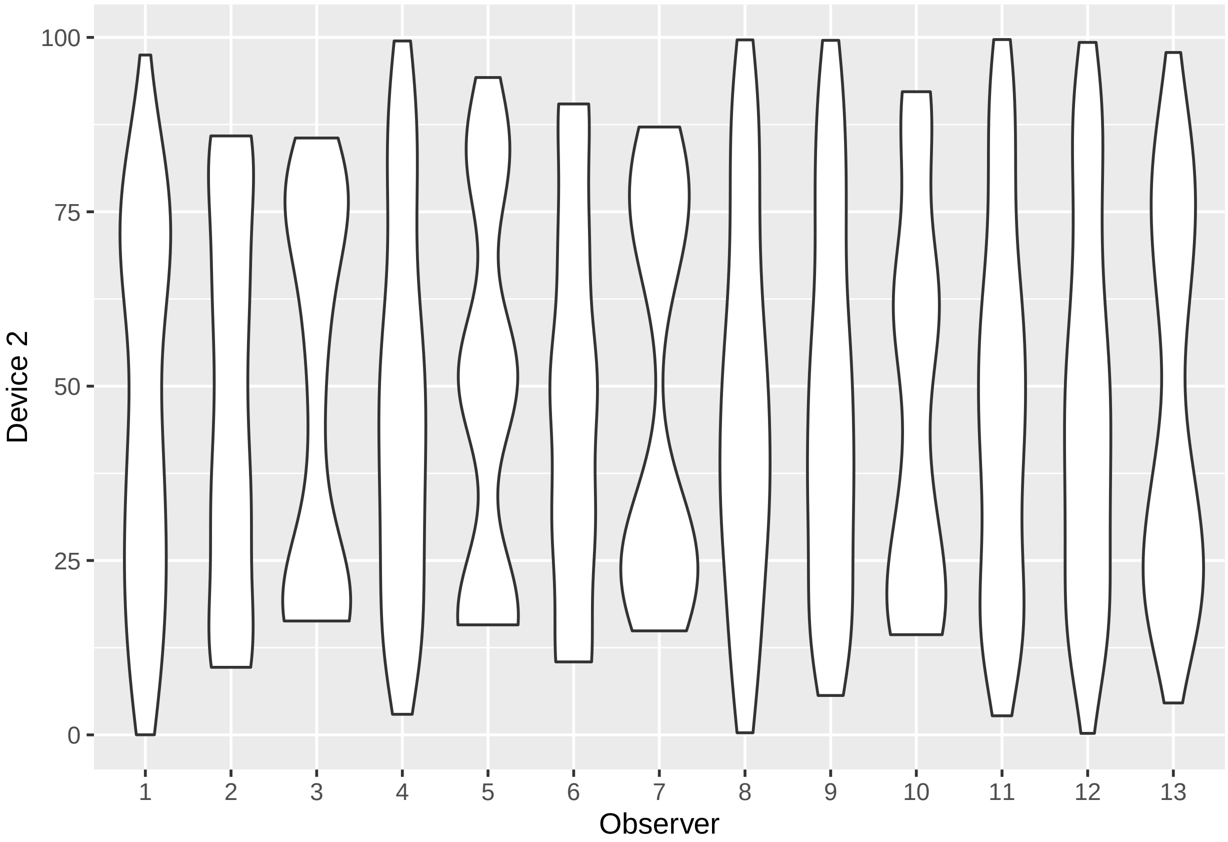
Although the summary statistics of these data indicate identical distributions, the box plots show that there is more variability in shapes. The medians of Device 2 observations vary widely across observer. Some of the observers have extremely skewed distributions, with a few containing the median equal to the first quartile. This adds to my suspicions from when the summary statistics lined up perfectly over all observers.

Another way to look at the spread or distribution of measurements is by viewing a violin plot. The windows where each observer listed more measurments than others will be shown more clearly.

```
ggplot(device_data, aes(x = factor(Observer), y = dev1)) + geom_violin() +
  labs(x = "Observer", y = "Device 1")
```

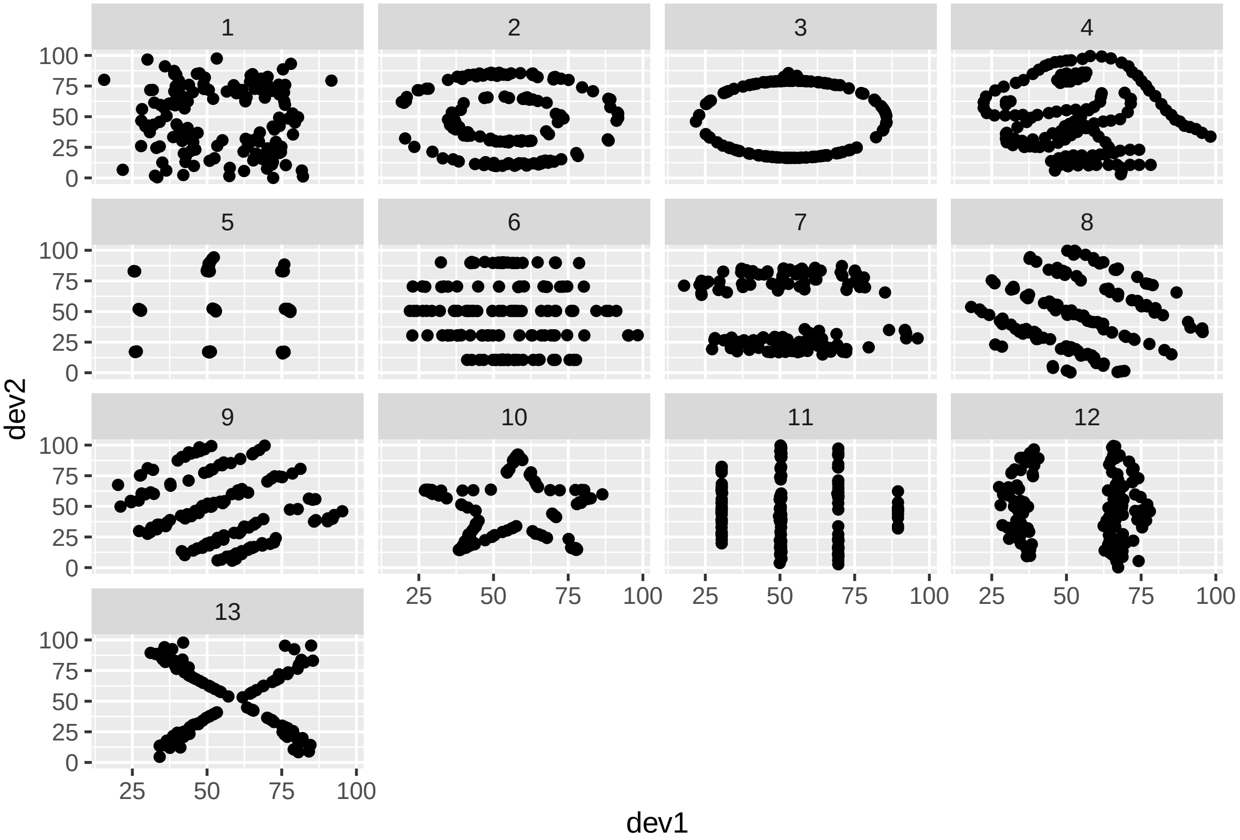


```
ggplot(device_data, aes(x = factor(Observer), y = dev2)) + geom_violin() +
  labs(x = "Observer", y = "Device 2")
```



These violin plots confirm some of the differences between the summary statistics and the boxplot. The measurements across each observe seem to be centered around a common point. However, the individual observers contain wildly different distributions of measurements.

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



The lesson here is to view your data in many different ways before jumping to conclusions. The summary statistics tell a very different story than the plots viewed here. Additionally, I think a valuable lesson is to be skeptical of perfect data. If the data are lining up quite conveniently, I would try to view the data from a different perspective and double check my work. my work is lining up

Problem 6

Now we are going to use the method of Riemann sums to approximate an integral. In this implementation, we will use a left Riemann sum. The rectangles created will touch the curve at their top-left corner.

```
## Calculate a left Riemann sum
## slice_width: width of rectangles when calculating sum
## start, finish: endpoints of apporximated integral
## returns a scalar of the left Riemann sum
get_riemann_sum <- function(slice_width, start = 0, finish = 1) {
  sum <- 0
  ## Rectangles created from the left
  rectangles <- seq(start, finish - slice_width, by = slice_width)
  for (index in rectangles) {
    sum <- sum + slice_width * exp(-(index)^2 / 2)
  }
  return(sum)
}

## Approximate the integral using varying slice widths
## actual sum: 0.855624
exact_integral <- 0.855624
riemann_results <- data.frame()
```

```r
slice_width <- 1
attempt <- 1
result <- .Machine$integer.max
while (result - exact_integral > 1e-6) {
  result <- get_riemann_sum(slice_width)
  riemann_results <- rbind(riemann_results,
                           c(attempt, format(slice_width, scientific = F),
                            format(result, digits = 6), exact_integral))
  slice_width <- slice_width * .1
  attempt <- attempt + 1
}
colnames(riemann_results) <- c("Attempt", "Slice_Width", "Approximation",
                               "Exact Integral")
```

Problem 7

Now we are going to use Newton's method to find the roots of an equation.

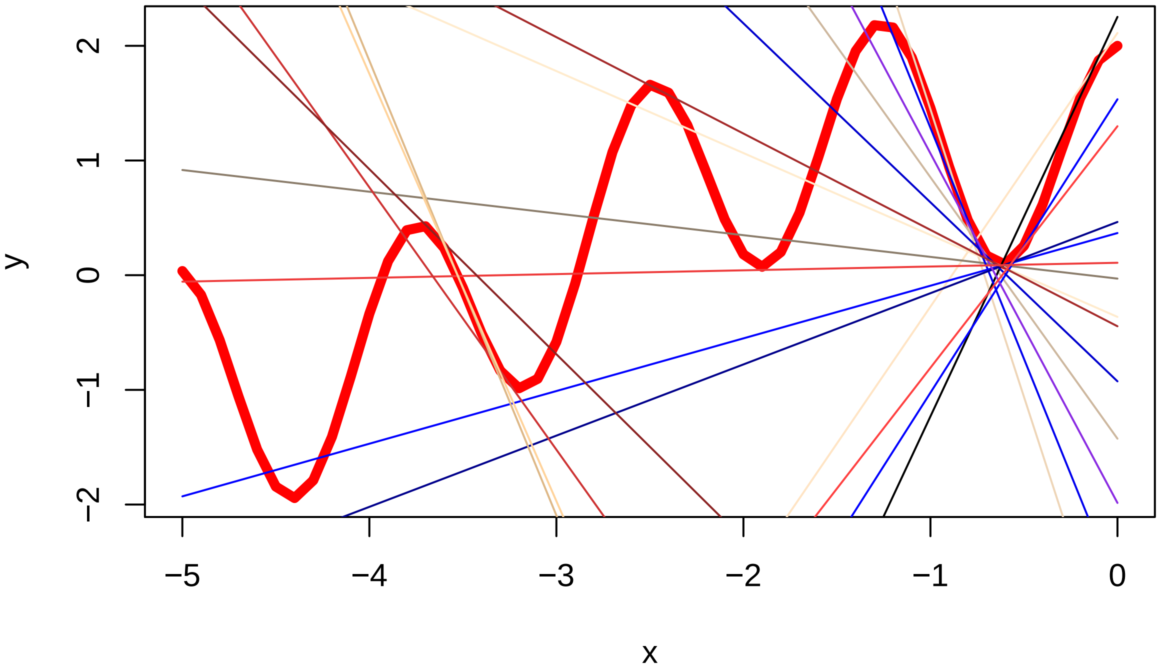
```r
## Get a root of an equation using Newton's method
## x_0: starting approximation
## tolerance: different in successive approximations that will
## indicate completion
## returns a scalar of the approximated root
get_newton_method <- function(x_0, tolerance, plot = FALSE) {
  attempt <- 0
  new_estimate <- x_0
  ## Generate data to plot the function from -5 to 0
  x_values <- seq(-5, 0, by = 0.1)
  y_values_orig <- 3^x_values - sin(x_values) + cos(5 * x_values)
  if (plot) {
    plot(x = x_values, y = y_values_orig, type = "l", col = "red", lwd = 5,
       xlab = "x", ylab = "y")
  }
  should_continue <- TRUE
  while (should_continue) {
    prev_estimate <- new_estimate
    func_result <- 3^prev_estimate - sin(prev_estimate) + cos(5 * prev_estimate)
    derivative_result <- 3^prev_estimate * log(3) - cos(prev_estimate) -
                          5 * sin(5 * prev_estimate)
    new_estimate <- prev_estimate - (func_result / derivative_result)
    ## Calculate the line tangent to the curve
    slope <- derivative_result
    intercept <- 0 - slope * new_estimate
    y_values <- slope * x_values + intercept
    if (plot) {
      lines(x = x_values, y = y_values, col = colors()[attempt + 20])
    }
    attempt <- attempt + 1
    ## Evaluating if we have reached the tolerance level to be satisfied with
    ## our result
    should_continue <- abs(prev_estimate - new_estimate) > tolerance
  }
  return(c("num_attempts" = attempt, "final_result" = new_estimate))
}
```

```r
## Construct table for varying starting values and tolerance levels
newton_results <- data.frame()
starting_values <- c(-0.1, -1.9, -3.75, -6.8, -9.0)
tolerance_levels <- c(0.1, 0.01, 0.001, 0.0001)
for (value in starting_values) {
  for (level in tolerance_levels) {
    results <- get_newton_method(value, level)
    newton_results <- rbind(newton_results, c(value, level,
                        results["num_attempts"], results["final_result"]))
  }
}

colnames(newton_results) <- c("Starting_Value", "Tolerance_Level",
                       "Num_Attempts", "Final_Result")
knitr::kable(newton_results)
```

Starting_Value	Tolerance_Level	Num_Attempts	Final_Result
-0.10	1e-01	3	-0.625571
-0.10	1e-02	19	-3.528723
-0.10	1e-03	19	-3.528723
-0.10	1e-04	20	-3.528723
-1.90	1e-01	3	-2.887053
-1.90	1e-02	3	-2.887053
-1.90	1e-03	4	-2.887058
-1.90	1e-04	4	-2.887058
-3.75	1e-01	4	-3.528658
-3.75	1e-02	4	-3.528658
-3.75	1e-03	5	-3.528723
-3.75	1e-04	5	-3.528723
-6.80	1e-01	2	-6.673820
-6.80	1e-02	3	-6.676055
-6.80	1e-03	4	-6.676061
-6.80	1e-04	4	-6.676061
-9.00	1e-01	2	-9.162753
-9.00	1e-02	3	-9.162986
-9.00	1e-03	3	-9.162986
-9.00	1e-04	4	-9.162986

```r
## Plot the tangent lines as we approach the result
store <- get_newton_method(-0.1, 0.001, plot = TRUE)
```



Problem 8

We would like to compare the performance of for loops to matrix operations. In order to do this, we will calculate SST for data from a normal distribution

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

## Calculate SST using for loop to measure performance
sst_for_loop <- function(y, mean_y) {
  sum_squares_for_loop <- 0
  for (y_val in y) {
    sum_squares_for_loop <- sum_squares_for_loop + (y_val - mean_y)^2
  }
  return(sum_squares_for_loop)
}

sst_matrix <- function(y, mean_y) {
  error <- y - mean_y
  sum_squares_matrix <- t(error) %*% (error)
  return(sum_squares_matrix)
}

mean_y <- mean(y)
results <- microbenchmark::microbenchmark(sst_for_loop(y, mean_y),
                                          sst_matrix(y, mean_y), times = 100)
results
```

```
## Unit: microseconds
##                       expr   min     lq     mean median     uq      max neval
##    sst_for_loop(y, mean_y) 4.854 4.9780 71.27219 5.1545 5.7585 6520.079   100
##      sst_matrix(y, mean_y) 4.466 4.6635 61.20298 4.9470 6.6040 5306.100   100
```