Authors: Sean Saito, John Reid, Hrishi Olickel, Chong Woon Han

1.a) We use the following code at the beginning of the script:

```
#imports
require(ggplot2)

# Automatically sets working directory to source file location
this.dir <- dirname(parent.frame(2)$ofile)
setwd(this.dir)
```

The next few lines imports florida2000.txt as a table and we use the data to generate the plot:

```
# Read the text file
florida <- read.table("florida2000.txt", header=TRUE)

# Get the votes for Buchanan and Nader
buchanan <- florida$Buchanan
nader <- florida$Nader

# Plot
plot(nader, buchanan, xlab="Nader", ylab="Buchanan", pch=20, main =
     "Nader vs. Buchanan")
```

The scatterplot is shown in the plot below:

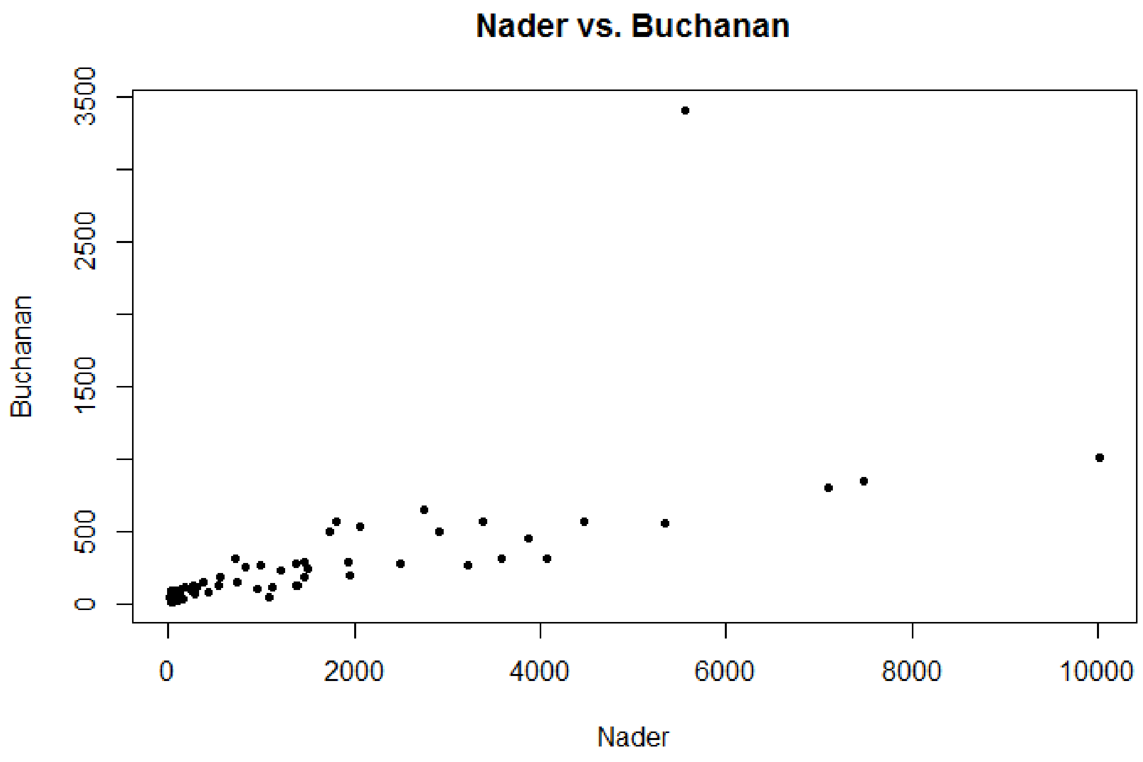


Figure 1: Scatterplot

1.b) The next line of code fits the data to a linear model and plots the line of regression onto our previous scatterplot. Figure 2 shows the resulting plot.

```
# Linear regression
fit <- lm(buchanan ~ nader)
abline(fit)
```

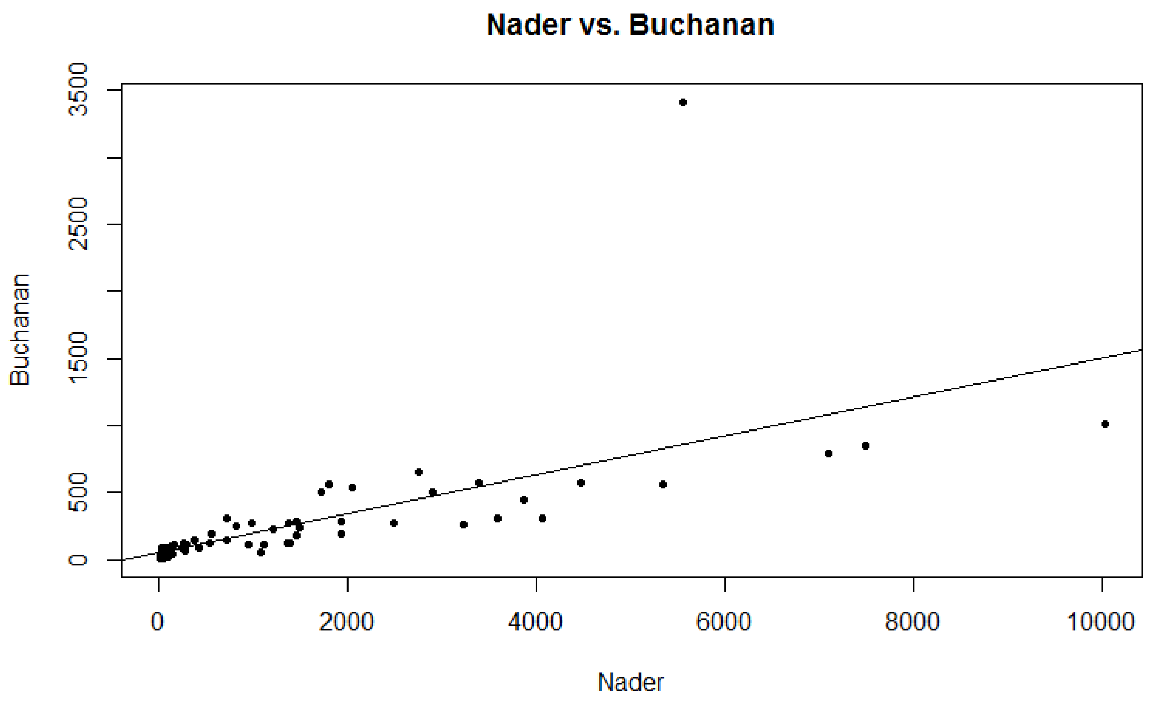


Figure 2: Scatterpolot with regression line

1.c) To test whether the regression line is a reasonable fit, we plot the residuals using the following code. The plot is shown in Figure 3.

```
res <- residuals(fit)
plot(jitter(res)~jitter(buchanan), ylab="Residuals", xlab="buchanan",
data=florida, pch=20)
abline(0, 0)
```

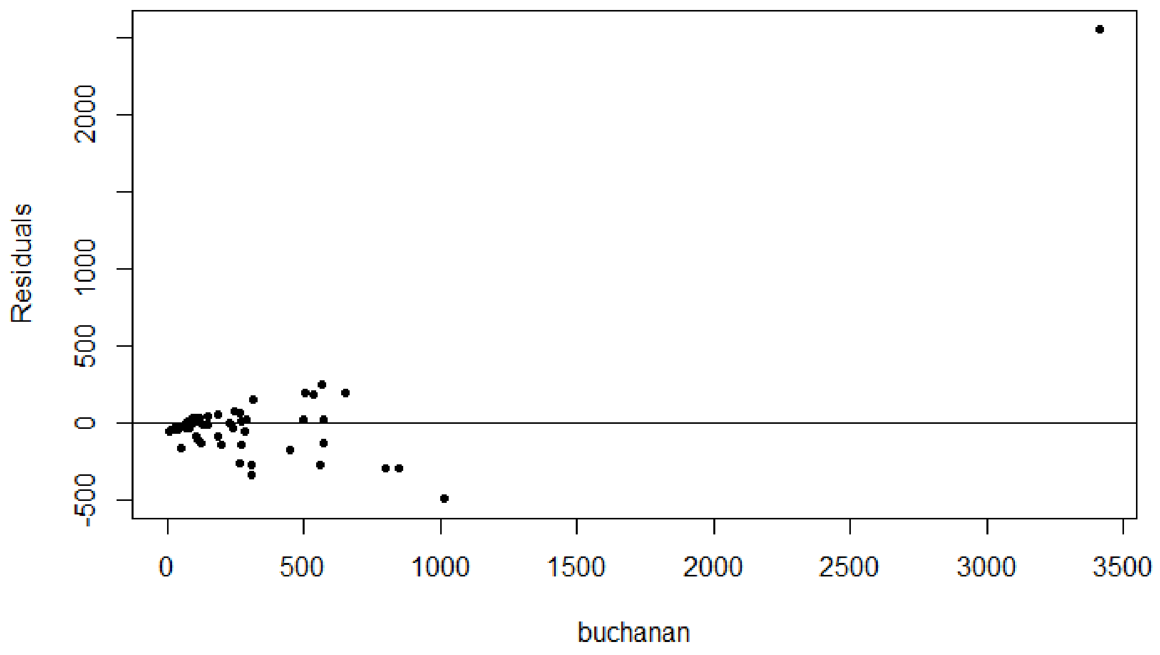


Figure 3: Plot of residuals

The plot of residuals in Figure 3 suggests increasing vertical spread across the x-axis, with suggests that a linear model may not be particularly suitable. We perform a further check by examining the R^2-value with the following code:

```
print(summary(fit)$r.squared)
```

The code returns a value of 0.42799778, suggesting that the linear model may not be a reasonable fit.

1.d) First we create a function to calculate the residual sum of squares:

```
# Residual Sum of Squares function
rss = function(row) {
  z <- sum((buchanan - row[1] - row[2] * nader)^2)
}
```

Next, we prepare vectors of intercepts and slopes and compute RSS to prepare for plotting.

```
# Generate of vector of intercepts and slopes
b0 <- seq(1, 120,3)
b1 <- seq(0.11, 0.18, length=40)

# Compute rss for all (b0, b1)
prod = merge(b0, b1) # Cartesian product of b0 and b1
z <- matrix(apply(prod, 1, rss), nrow=length(b0), ncol=length(b1))
```

We generate a perspective plot using the following code, with the plot shown in Figure 4:

```
# Prepare a matrix of colors scaled to the RSS value, to use in the plot
alphas <- (z-min(z))/(max(z)-min(z))
colors <- rgb(matrix(1,40,40),matrix(0,40,40),matrix(0,40,40),alphas)

# Plot persp
persp(b0, b1, z, theta=65, phi=10, xlab="Intercept",
      ylab="Slope", zlab="RSS", ticktype="detailed",
      expand=0.5,col=colors)
```

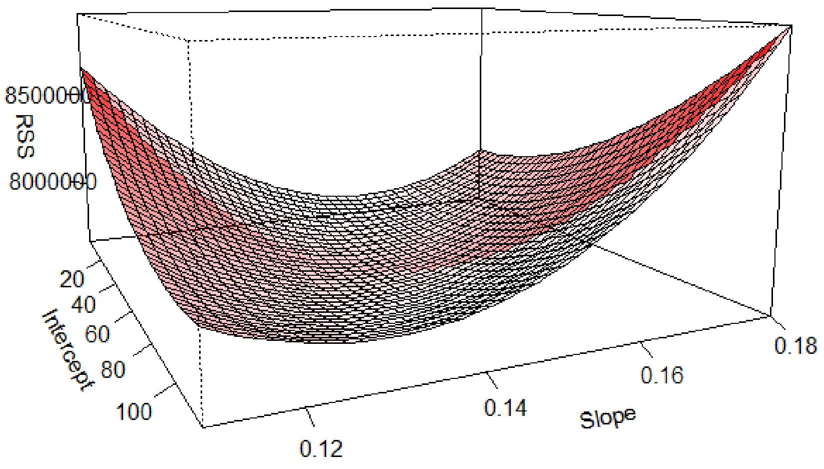


Figure 4: Perspective plot. Points with a higher RSS value have a deeper colour of red.

1.e) We generate the contour plot using the following code:

```
# Contour plots
contour(b0, b1, z, xlab="Intercept", ylab="Slope", main="Contour of
Intercept, Slope, and RSS")
```

We then plot a point indicating the position of the regression slope and intercept obtained earlier, using the following code:

```
# Plot the best combination
points(fit$coefficients[1], fit$coefficients[2], pch=20, col="red")
```

The resultant plot is shown in Figure 5 below:

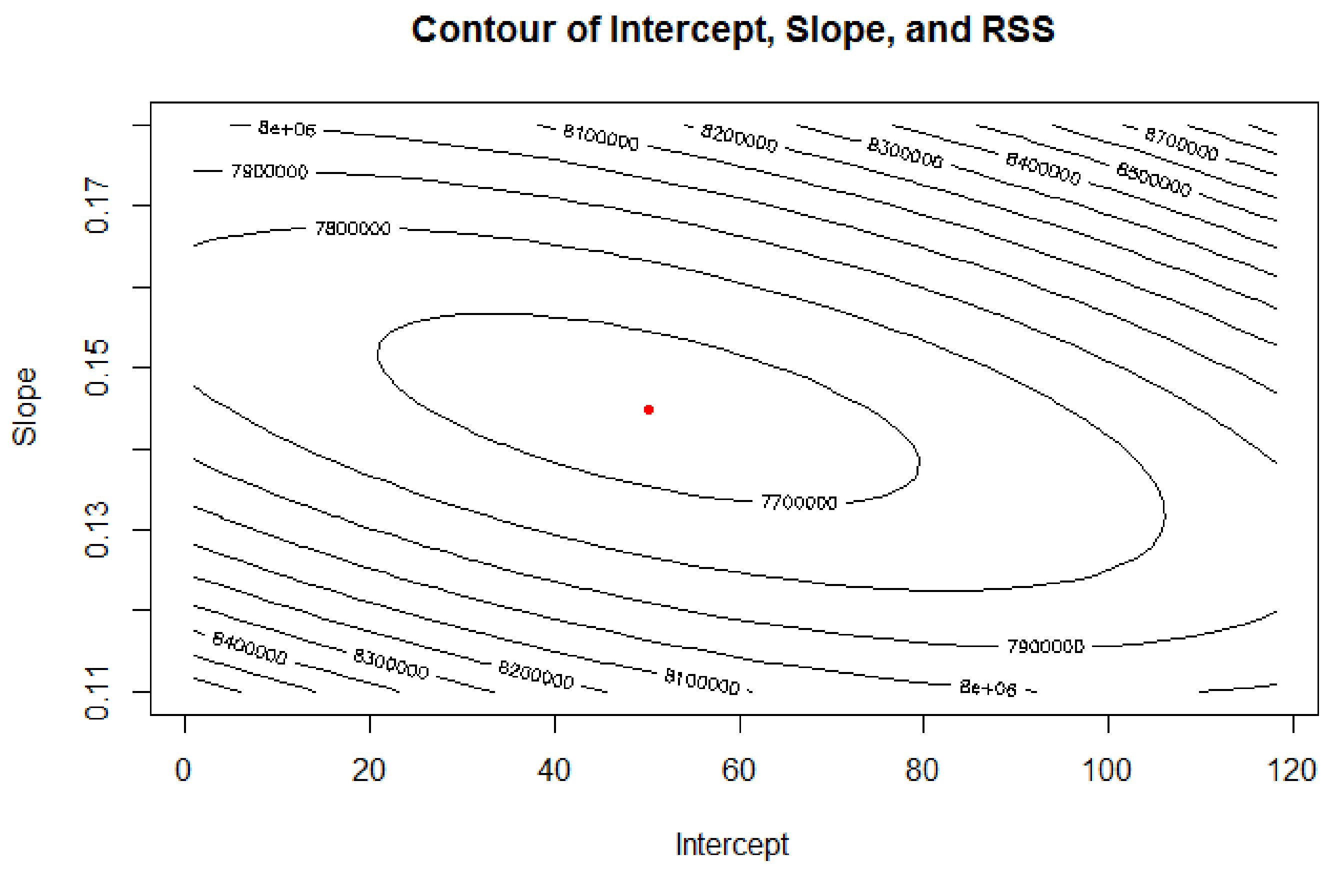


Figure 5: Contour plot. The red point represents the position given by the linear model.

2.a) We implement the Taylor approximation as a function using the following code:

```
taylorLoop <- function (n, x) {
  if (n < 1 || length(x) == 0 ){
    print("Either n or x is an invalid parameter")
    break
  }
  output <- numeric(0)  # Setting up the output vector
  for (a in x){   # Outer for loop iterating over x

    result <- 0   # Setting up the summation
    for (j in 0:n ){
      result <- result + (a^j / factorial(j))
    }
    output <- c(output, result)    #Concatenate the result to the output
vector
  }
  return(output)
}
```

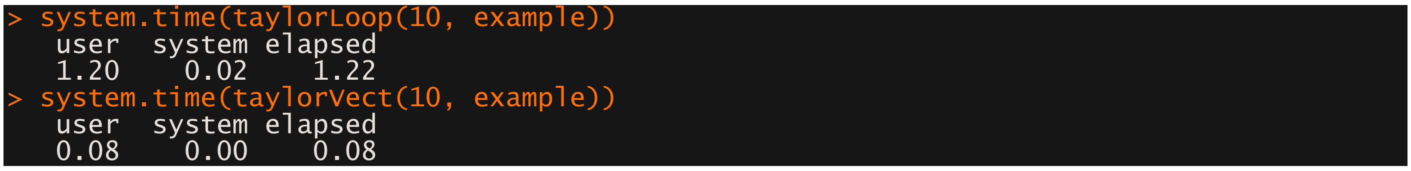
2.b) We implement a vectorised function for the Taylor approximation in the following code:

```
taylorVect <- function (n, x){
  if (n < 1 || length(x) == 0 ){
    print("Either n or x is an invalid parameter")
    break
  }
  return(unlist(lapply(x, function(a) sum((a ^ (0:n)) / factorial(0:n)))))
}
```

2.c) The following code was used to compare the run times of the two functions:

```
example <- seq(-10, 10, 0.001)
system.time(taylorLoop(10, example))
system.time(taylorVect(10, example))
```

We have the following results:



```
> system.time(taylorLoop(10, example))
   user  system elapsed
   1.20    0.02    1.22
> system.time(taylorVect(10, example))
   user  system elapsed
   0.08    0.00    0.08
```

2.d) The following code implements a vectorised function for the Pade approximation:

```
# Compute S_nx
snx_vect <- function(n, x)
  (sum(unlist(lapply(c(0:n), function(j) ((factorial((2*n)-
j)*factorial(n))*(x^j))/(factorial(2*n)*factorial(j)*factorial(n-j))))))

# Apply over x
pade <- function(n, x) {
  return(unlist(lapply(x, function(xval) (snx_vect(n, xval)/snx_vect(n,
0-xval)))))
}
```

2.e) The following code is used to plot a comparison of values produced by the three functions:

```
taylor10 <- function(x) taylorVect(10, x)
pade10 <- function(x) pade(10, x)

ggplot(data.frame(x = c(-10, 10)), aes(x)) +
  stat_function(fun = exp, aes(colour="exp"),size=1.5) +
  stat_function(fun = taylor10, aes(colour="taylor"),geom="point") +
  stat_function(fun=pade10, aes(colour="pade"),geom="point") +
  scale_y_log10()
```

The resultant plot is shown below in Figure 6.

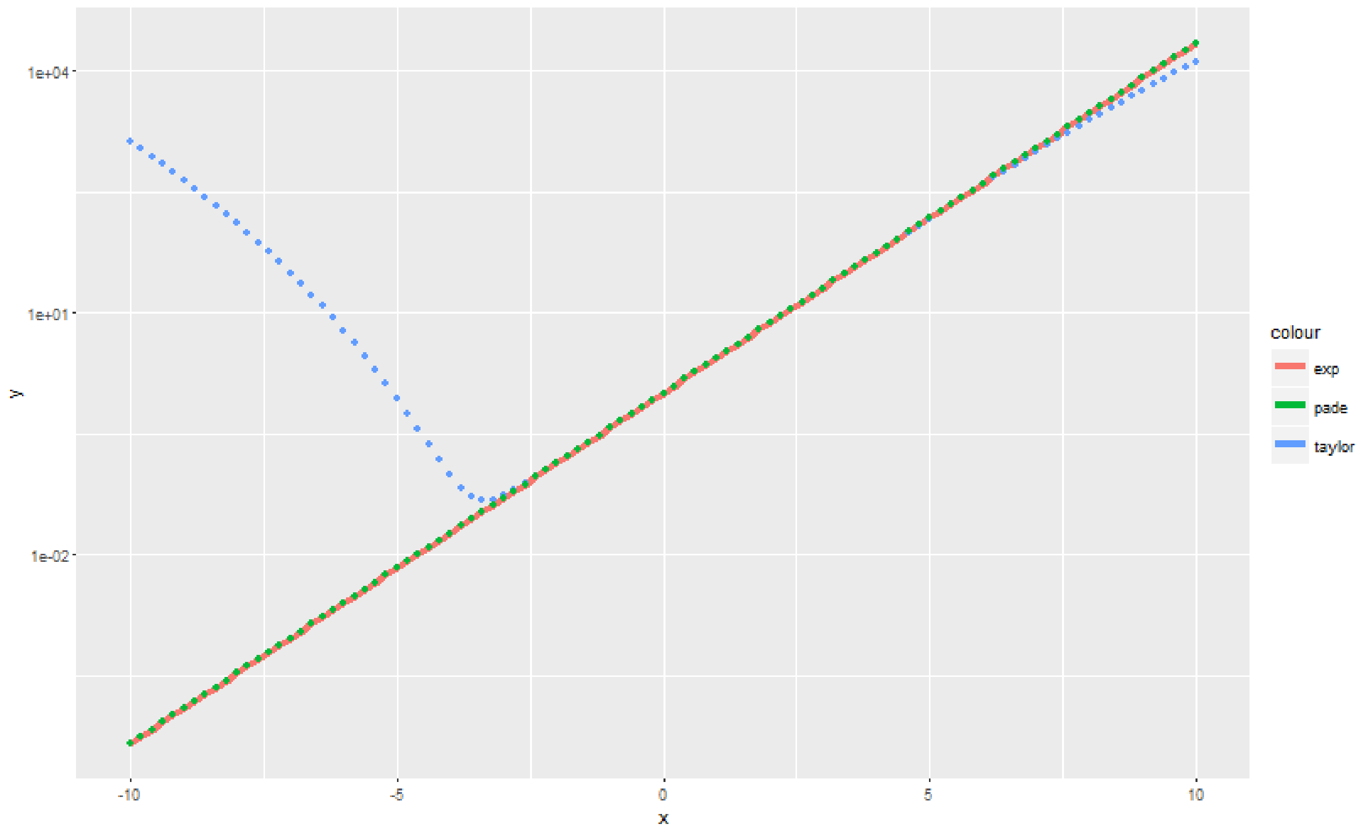


Figure 6: Comparison of values evaluated by exp(x), T_{10}(x) and P_{10}(x).

2.f) The following code compares the run time of the three functions:

```
system.time(taylorLoop(10, example))
system.time(taylorVect(10, example))
system.time(pade(10, example))
```

The results are as follows:

```
> system.time(taylorLoop(10, example))
   user  system elapsed
   1.25    0.01    1.27
> system.time(taylorVect(10, example))
   user  system elapsed
   0.08    0.00    0.08
> system.time(pade(10, example))
   user  system elapsed
   2.75    0.00    2.75
```

2.g) In short, Pade is the most accurate representation, but the vectorised Taylor function is much faster. We will demonstrate this through a further comparison of runtimes and accuracy.

The following code generates a graph comparing the runtimes of Pade and (vectorised) Taylor:

```
xval <- seq(1,100,1)
padeY <- unlist(lapply(xval, function(x) as.double(system.time(pade(x,
c(10:100)))[3])))
taylorY <- unlist(lapply(xval, function(x)
as.double(system.time(taylorVect(x, c(10:100)))[3])))
ggplot(data.frame(x=xval, tY=taylorY, pY=padeY),
aes(x))+geom_line(aes(y=tY, color='taylor')) +
 geom_line(aes(y=pY, color='pade')) + ggtitle('Runtime against n - pade
and taylor')
```

The resultant graph is shown below in Figure 7:

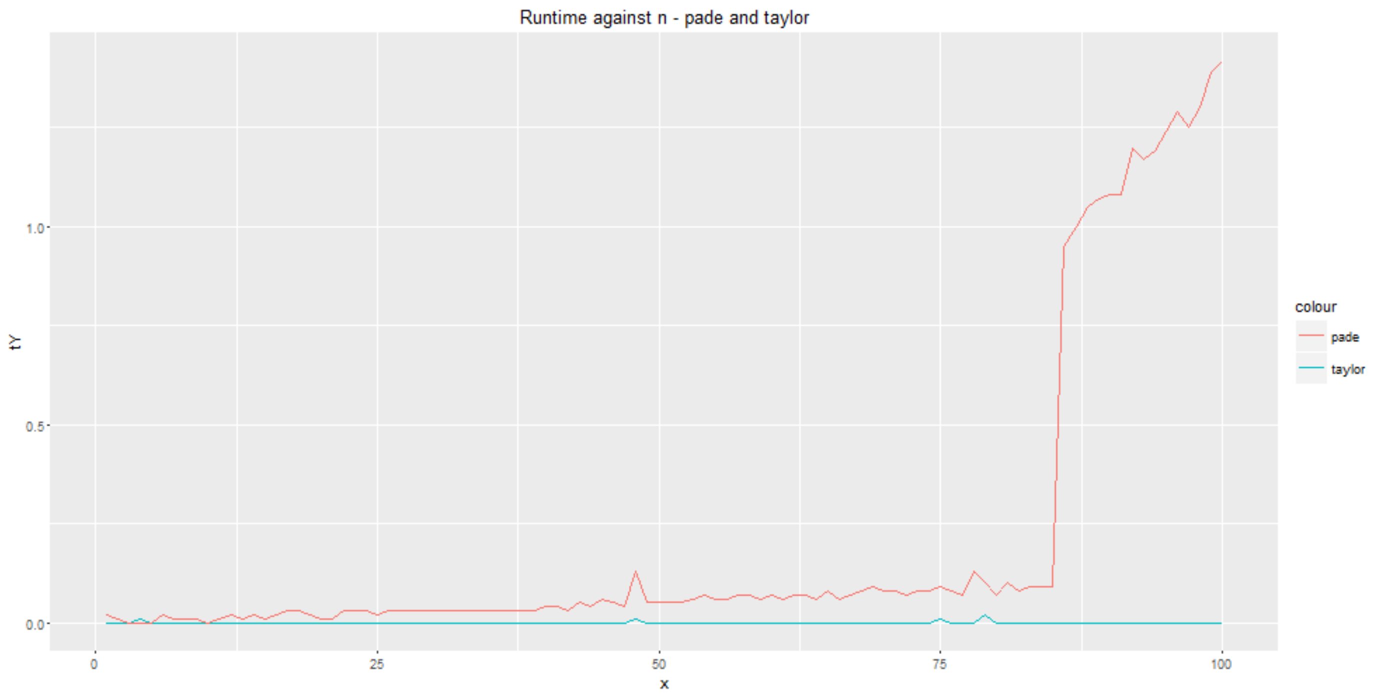


Figure 7: Comparison of runtimes for pade() and taylorVect().

From Figure 7 we can see that the taylorVect() is much faster than pade(), even for high values of n. Next we will use the following code to compare the accuracy of these functions for different values of x:

```
ggplot(data.frame(x = c(-10, 10)), aes(x)) + stat_function(fun = exp,
aes(colour="exp"))+
 stat_function(fun = function (x) taylorVect(10, x), aes(color="10")) +
 stat_function(fun = function (x) taylorVect(20, x), aes(color="20")) +
 stat_function(fun = function (x) taylorVect(30, x), aes(color="30")) +
 stat_function(fun = function (x) taylorVect(40, x), aes(color="40")) +
 stat_function(fun = function (x) taylorVect(50, x), aes(color="50")) +
 stat_function(fun = function (x) taylorVect(60, x), aes(color="60")) +
 scale_y_log10() + ggtitle('Values of exp and taylor(n=10-60)')
```

The resultant graph is shown in Figure 8 below:

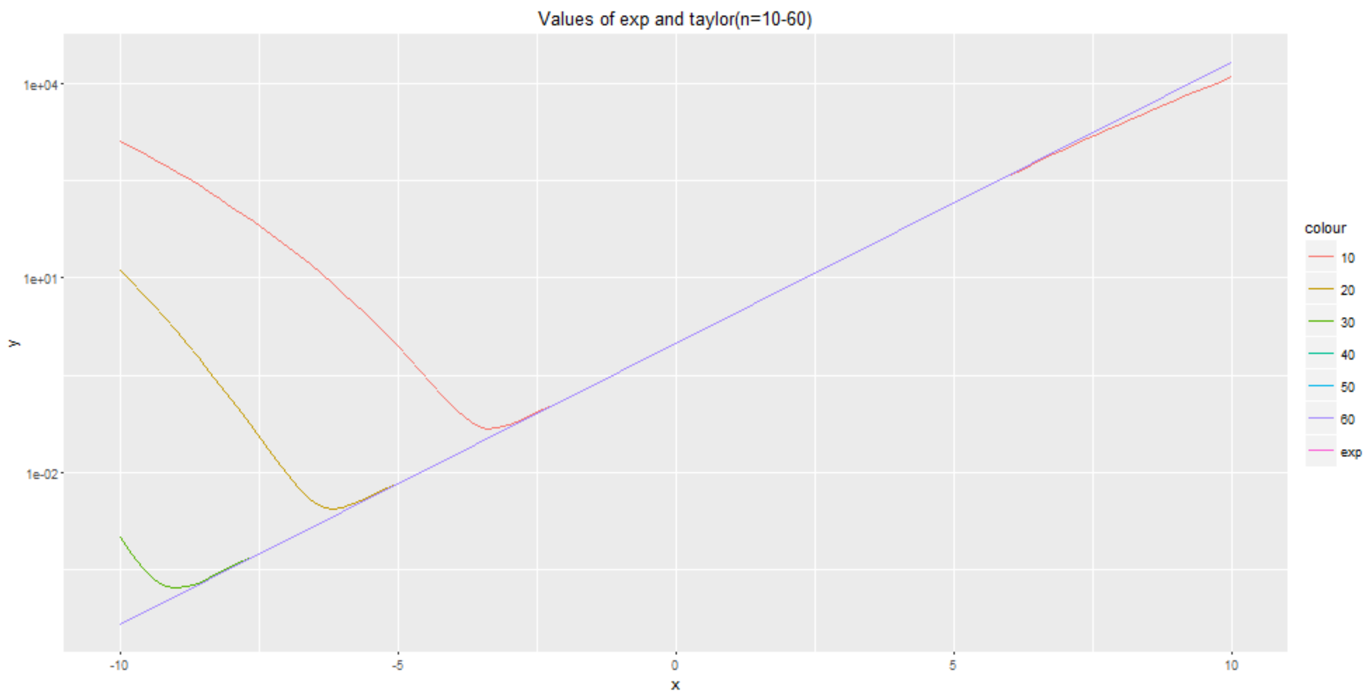


Figure 8: Accuracy of taylorVect() over a range of n values.

Based on the results shown in Figure 7 and Figure 8, we conclude that taylorVect() is the better choice. By using higher values of n depending on the value of x, we can provide results faster than using pade() without sacrificing accuracy. Thus, taylorVect() is a better choice for industry, but pade() is a better solution for smaller datasets.