# HUDM6026 Homework_05

Chenguang Pan & Seng Lei

Feb 24, 2023

## Q1:

*Determine the first derivative of f and encode it in a function called f_prime.*

**MY SOLUTION:**
Based on chain rule, the first derivative of $f(x)$ is

$$f(x)' = \frac{-2x}{x^2+1} + \frac{1}{3}x^{-\frac{2}{3}}$$

. Based on this equation, I write the code below. Certainly, we can use the R-built-in function to get the derivative quickly.

```
> f_prime <- function(x) {
+   out_ <- (-2*x)/(x^2 + 1) + (1/3)*(x^(-2/3))
+   return(out_)
+ }
> # since this question requires the maximum, I use -1 times the function.
> f_prime_neg <- function(x){
+   out_ <- (-2*x)/(x^2 + 1) + (1/3)*(x^(-2/3))
+   out_ <- out_*(-1)
+   return(out_)
+ }
```

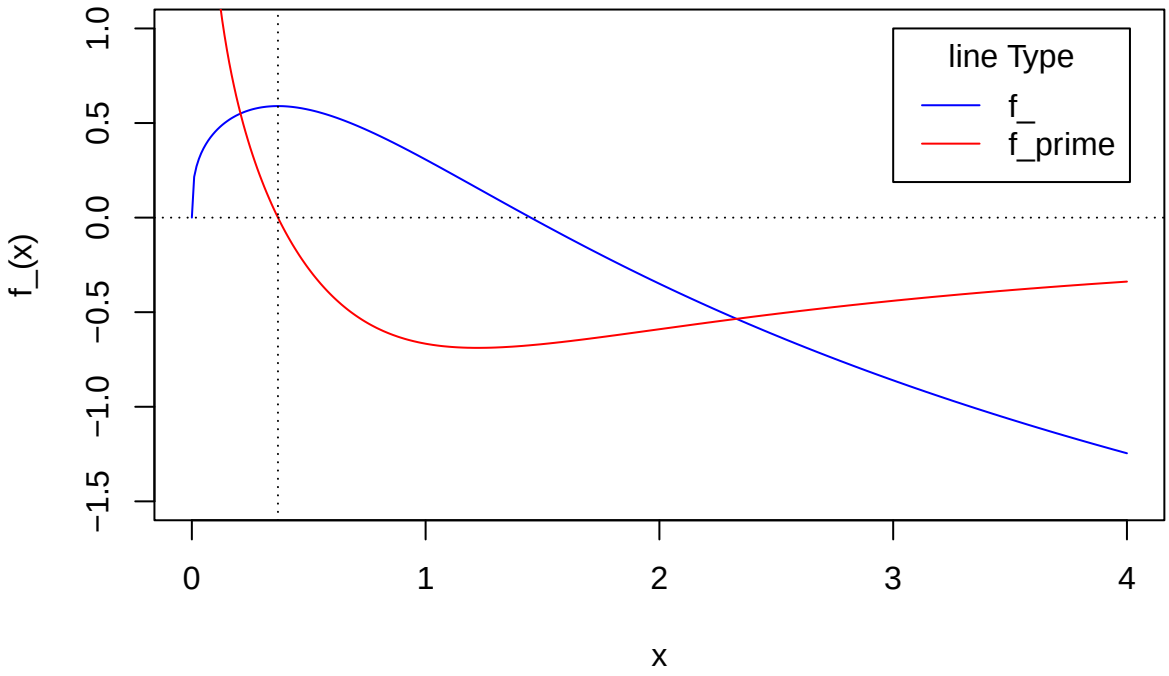
## Q2:

*Create a plot of f and f on [0,4] in different colors and line types and add a legend.* **MY SOLUTION:**

```
> # write the original function with the name of f_
> f_ <- function(x){
+   out_ <- (-1)*log(x^2 + 1) + x^(1/3)
+   return(out_)}
> # since this question requires the maximum, I use -1 times the function.
> f_neg <- function(x){
+   out_ <- (-1)*log(x^2 + 1) + x^(1/3)
+   return(out_*(-1))}
>
> # first plot the original function
> x <- seq(0,4,0.01)
> # plot the original function with blue line
> plot(x, f_(x), col="blue", type = "l", ylim = c(-1.5,1))
> # plot the first derivative with red line
```

```
> lines(x, f_prime(x), col="red", type = "l")
> # add the legend
> legend(3,1, inset = 0.1, c("f_","f_prime"), lty = 1,
+        col = c("blue","red"), title="line Type")
> # add a horizontal line to indicate the y=0
> abline(h=0,lty=3)
> abline(v=0.3683525,lty=3)
> mtext("Figure 1. The Lines of the Given Function and it's First Derivative",
+       side = 3,
+       line = -2,
+       outer = T)
```

Figure 1. The Lines of the Given Function and it's First Derivative




**Q3:**

*Finish the functions that I started in the R code notes for univariate optimization for the golden section search, the bisection method, and Newton's method.*

**MY SOLUTION:**

### 3.1 The Golden Section Search

```
> golden <- function(f, int, precision = 1e-6)
+ {
+   rho <- (3-sqrt(5))/2 # ::: Golden ratio
+   # ::: Work out first iteration here
+   f_a <- f(int[1] + rho*(diff(int)))
```

```
+   f_b <- f(int[2] - rho*(diff(int)))
+   ### How many iterations will we need to reach the desired precision?
+   N <- ceiling(log(precision/(diff(int)))/log(1-rho))
+   for (i in 1:(N))                    # index the number of iterations
+   {
+     if (f_a < f_b)
+     {
+       int[2] <- int[2] - rho*(diff(int))
+       f_b <- f(int[2])
+     } else{
+       if (f_a >= f_b)
+       {
+         int[1] <- int[1] + rho*(diff(int))
+         f_a <- f(int[1])
+       } }
+   }
+   int
+   print(paste0("Iteration for ",N," times;",
+                " The location is at ", round(int[1],6)))
+ }
```

### 3.2 The Bisection Method

More information about this method can be found on *Page 116* of Chong and Zak (2013).

```
> bisection <- function(f_prime, int, precision = 1e-7)
+ {
+   # ::: f_prime is the function for the first derivative
+   # ::: of f, int is an interval such as c(0,1) which
+   # ::: denotes the domain
+
+   N <- ceiling(log(precision/(diff(int)))/log(.5))
+   # find the midpoint of the initial uncertainty range
+   midpoint <- (int[1]+int[2]) /2
+   # evaluate the f_prime on the midpoint
+   f_prime_a <- f_prime(midpoint)
+   i <- 1
+   for (i in 1:N)
+   {
+     i <- i + 1
+     if(f_prime_a < 0)
+     {
+       # if the f_prime on the midpoint is less than 0,
+       # the minimizer must be on the right side of midpoint.
+       int[1] <- midpoint
+       # update the uncertainty range
+       midpoint <- (int[1]+int[2]) /2
+     } else
+       if(f_prime_a > 0)
+       {
+         # if the f_prime on the midpoint is less than 0,
+         # the minimizer must be on the left side of midpoint.
+         int[2] <- midpoint
```

```
+         midpoint <- (int[1]+int[2]) /2
+       } else
+         if(f_prime_a == 0)
+         {
+           break
+         }
+     # ::: FILL IN CODE HERE (UPDATE)
+     f_prime_a <- f_prime(midpoint)
+   }
+   int
+   print(paste0("Iteration for ",N," times;",
+                " The location is at ", round(int[1],6)))
+ }
```

### 3.3 The Newton's Method

More information about this method can be found on *Page 116* of Chong and Zak (2013).

```
> newton <- function(f_prime, f_dbl, precision = 1e-6, start)
+ {
+   x_old <- start
+   x_new <- x_old - f_prime(x_old)/f_dbl(x_old)
+
+   i <- 1
+   print(paste0("Iteration ", i, "; Estimate = ", x_new) )
+   while (abs(x_new-x_old) > precision){
+     x_old <- x_new
+     x_new <- x_old - f_prime(x_old)/f_dbl(x_old)
+     # ::: redefine variables and calculate new estimate
+     # ::: keep track of iteration history
+     print(paste0("Iteration ", i+1, "; Estimate = ", x_new) )
+     i <- i + 1
+   }
+   x_new
+ }
```

## Q4:

*Apply each of the three functions to this example to discover the minimum. Keep track of and report the number of iterations required for each method. Report the coordinates of the minimum discovered by each of the three functions as well as the number of iterations required*

**MY SOLUTION:**

### 4.1 Apply the Golden Section Search

```
> golden(f_neg, c(0,4))
[1] "Iteration for 32 times;The location is at 0.368352"
```

### 4.2 Apply the Bisection Method

```
> bisection(f_prime_neg, c(0,4))
[1] "Iteration for 26 times;The location is at 0.368352"
```

### 4.3 Apply the Newton's Method

First, I need to get the second derivative of the original function. Based on the Quotient Rule and Chain Rule, one can easily have

$$f(x)'' = -6x^2 - 2 - \frac{2}{9}x^{-\frac{5}{3}}$$

```
> # write the second derivative function
> f_dbl <- function(x){
+   out_ <- -6*x^2-2-(2/9)*x^(-5/3)
+   return(out_)
+ }
> # since this question requires the maximum, I use -1 times the function.
> f_dbl_neg <- function(x){
+   out_ <- -6*x^2-2-(2/9)*x^(-5/3)
+   return(out_*(-1))
+ }
```

Plug the first and the second derivatives into the Newton's Method function.

```
> newton(f_prime_neg, f_dbl_neg,start = 1)
[1] "Iteration 1; Estimate = 0.918918918918919"
[1] "Iteration 2; Estimate = 0.830999851550479"
[1] "Iteration 3; Estimate = 0.736988742573808"
[1] "Iteration 4; Estimate = 0.639870150729756"
[1] "Iteration 5; Estimate = 0.546642635261104"
[1] "Iteration 6; Estimate = 0.468667930513546"
[1] "Iteration 7; Estimate = 0.416016821629135"
[1] "Iteration 8; Estimate = 0.388211559563474"
[1] "Iteration 9; Estimate = 0.376060038291039"
[1] "Iteration 10; Estimate = 0.371254711645403"
[1] "Iteration 11; Estimate = 0.369432581306114"
[1] "Iteration 12; Estimate = 0.368752709077139"
[1] "Iteration 13; Estimate = 0.368500562445562"
[1] "Iteration 14; Estimate = 0.368407257199536"
[1] "Iteration 15; Estimate = 0.368372758807196"
[1] "Iteration 16; Estimate = 0.36836000738855"
[1] "Iteration 17; Estimate = 0.368355294697857"
[1] "Iteration 18; Estimate = 0.36835355304667"
[1] "Iteration 19; Estimate = 0.368352909401224"
[1] 0.3683529
```

All three methods have the identical results, that is the maximum point is at $x = .368352$. As for the iteration times at the x interval $[0, 4]$, the Golden Section Search iterated 32 times; the Bisection Method iterated 26 times. However, Newton's method iteration time largely depends on the start numbers. That is, the more guessing number close to the target, the less times needed.