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

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

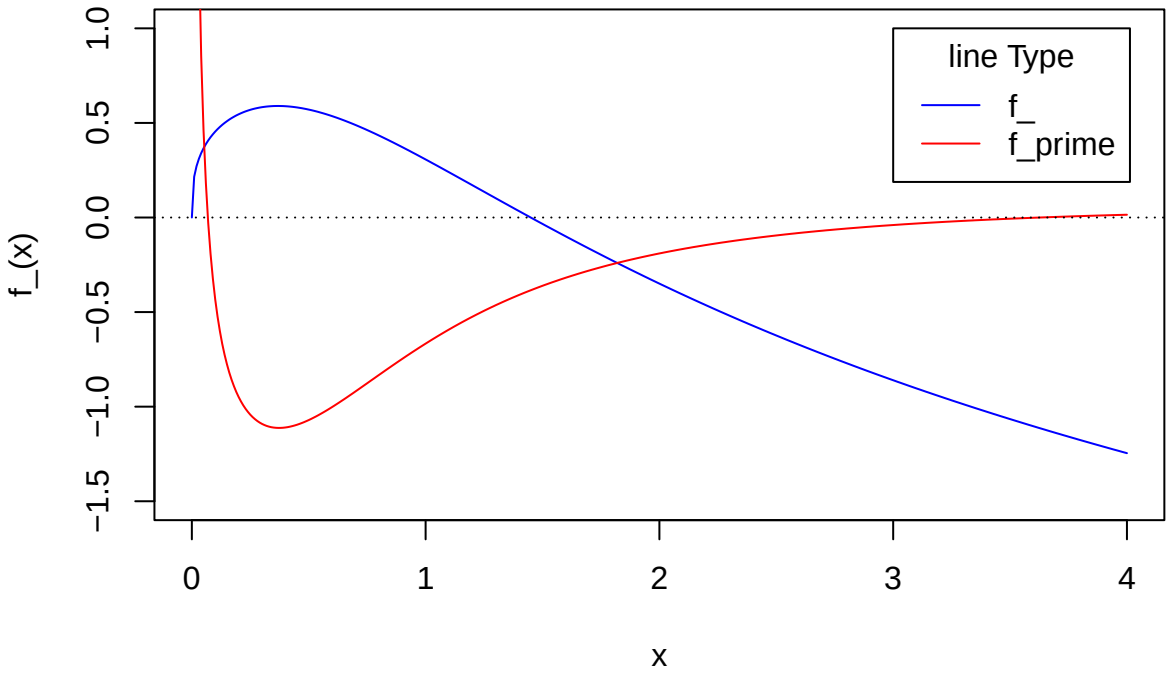
. Based on this equation, I write the code below. Certainly, we can use the R-built-in function to get the derivative quickly.

```
> f_prime <- function(x) {
+   out_ <- (-2*x)*((x^2 + 1)^(-1)) + (1/3)*(x^(-2/3))
+   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_)}
> f_(1)
[1] 0.3068528
> # 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)
```



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

```
> ############################################################
> ####### FUNCTION TO DO THE GOLDEN SECTION SEARCH #########
> ############################################################
> golden <- function(f, int, precision = 1e-6)
+ {
+   # ::: This function implements the golden section search for a
+   # ::: *minimum* for the function 'f' on the range [int]
+   # ::: with precision no greater than 'precision'.
+   # ::: Note: 'int' is an interval such as c(2,3).
+   # ::: If you want to *maximize*, multiply your function by -1.
+
+   rho <- (3-sqrt(5))/2 # ::: Golden ratio
+   # ::: Work out first iteration here
+   # select two test points, that is, move the x=a_0 to the right
+   # move the x=b_0 to the left, to shrink the range.
+   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] - 0.382*(int[2]-int[1])
+       f_b <- f(int[2] - rho*(diff(int)))
+     } else{
+       if (f_a >= f_b) # already have "else" above, is this if expression redundant?
+       {
+         int[1] <- int[1] + 0.382*(int[2]-int[1])
+         f_a <- f(int[1] + rho*(diff(int)))
+       } }
+   }
+   int
+ }
```

3.2 The Bisection Method

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

```
> ############################################################
> ####### FUNCTION FOR BISECTION METHOD ######################
> ############################################################
> 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)
+   for (i in 1:N)
+   {
+     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
+     } 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
+       } else
+         if(f_prime_a == 0)
+         {
```

```
+           break
+         }
+     # ::: FILL IN CODE HERE (UPDATE)
+     midpoint <- (int[1]+int[2]) /2
+     f_prime_a <- f_prime(midpoint)
+   }
+   int
+ }
```

3.3 The Newton's Method

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

```
> ##############################################################
> ####### FUNCTION FOR NEWTON'S METHOD #####################
> ##############################################################
> newton <- function(f_prime, f_dbl, precision = 1e-6, start)
+ {
+   # ::: f_prime is first derivative function
+   # ::: f_dbl is second derivative function
+   # ::: start is starting 'guess'
+
+   x_old <- start
+   x_new <- x_old - f_prime(x)/f_dbl(x)
+
+   i <- 1 # ::: use 'i' to print iteration number
+   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)/f_dbl(x)
+     # ::: 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: