Class 1 Intro to R

Yubo Rasmussen

# Introduction

This notebook demonstrates how to define functions in R, plot them, and find solutions using graphical and numerical methods. Examples include calculating the maximum likelihood estimate (MLE) and solving equations.

## Exiting R

To exit R, use the following command:

# Quit R  
#q()

## Example 1: Partial Log-Likelihood Function

We start by defining and plotting the partial log-likelihood function, as used in the Cox model tutorial.

### Define the Function

The following code defines a function PartialL with argument b. The {} braces are necessary because the function spans multiple lines. The function’s value is the last line without an assignment.

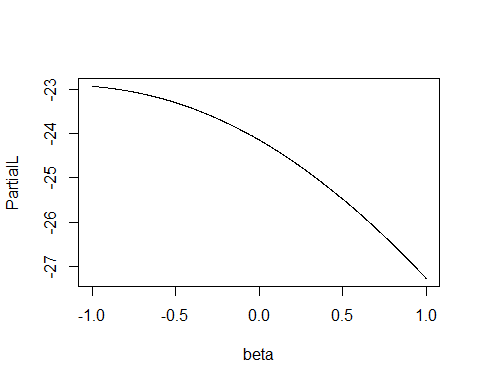
PartialL = function(b){  
 x = exp(b)  
 2 \* b - log(14 + 15 \* x) - log(12 + 14 \* x) - log(12 + 13 \* x) -   
 log(12 + 11 \* x) - log(10 + 11 \* x) - log(8 + 8 \* x) -   
 log(7 + 8 \* x) - log(6 + 8 \* x)  
}  
  
# Example usage: Calculate the partial log-likelihood at b = 1  
PartialL(1)

[1] -27.26969

### Plot the Function

We can plot PartialL over a range of b values using the following code:

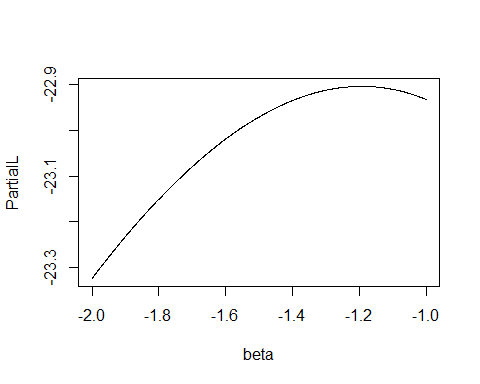
x = seq(-1, 1, length = 1000)  
plot(x, PartialL(x), type = "l", xlab = "beta", ylab = "PartialL")



### Refine the Plot for MLE

To refine the plot and locate the MLE of beta, we can narrow the range of b values:

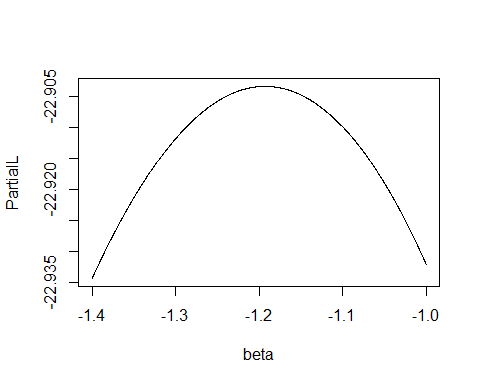
x = seq(-2.0, -1.0, length = 1000)  
plot(x, PartialL(x), type = "l", xlab = "beta", ylab = "PartialL")



### Create a Function to Refine the Plot

We can create a function to refine the plot further. This function takes the left and right bounds as arguments:

Plot.PartialL = function(Left, Right){  
 x = seq(Left, Right, length = 1000)  
 plot(x, PartialL(x), type = "l", xlab = "beta", ylab = "PartialL")  
}  
  
# Example usage  
Plot.PartialL(-1.4, -1.0)



## Example 2: Solving an Equation Graphically

### Define the Function

The following defines a function My.function. Note that X is a dummy argument in the definition:

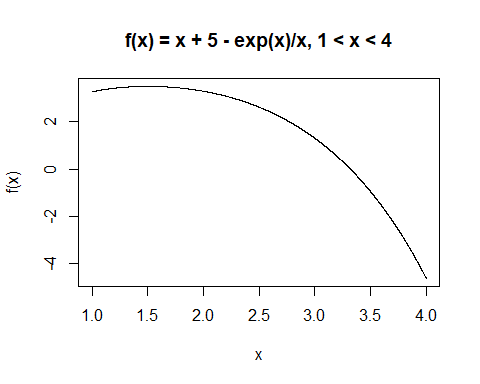
My.function <- function(X) X + 5 - exp(X) / X  
  
# Example usage: Calculate the value of the function at X = 3  
My.function(3)

[1] 1.304821

### Plot the Function

We can set up a sequence of x values and plot the function:

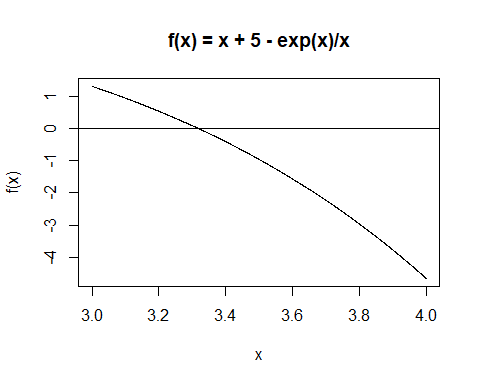
x <- seq(1, 4, by = 0.001)  
plot(x, My.function(x), type = "l", ylab = "f(x)",  
 main = "f(x) = x + 5 - exp(x)/x, 1 < x < 4")



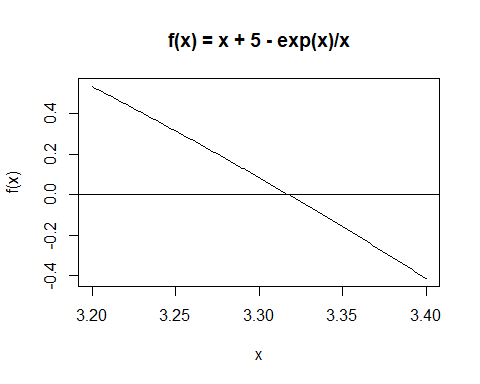
### Solve the Equation Graphically

To solve the equation graphically, we can define another function to add a horizontal line at y = 0 and refine the range:

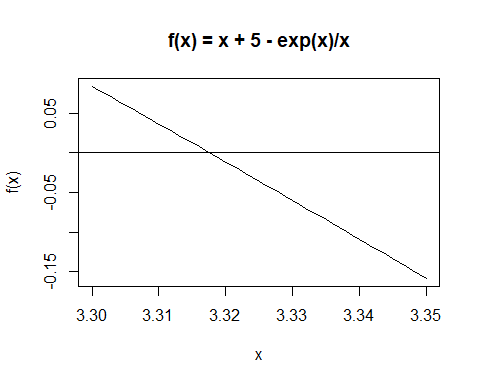
Solve.function <- function(Left, Right, By){  
 x <- seq(Left, Right, by = By)  
 plot(x, My.function(x), type = "l", ylab = "f(x)",  
 main = "f(x) = x + 5 - exp(x)/x")  
 abline(h = 0)  
}  
  
# Refine the solution range  
Solve.function(3, 4, 0.001)



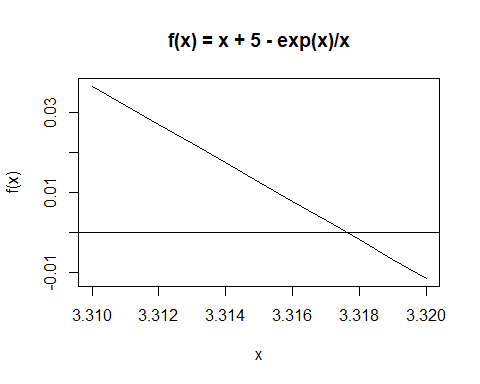
Solve.function(3.2, 3.4, 0.001)



Solve.function(3.3, 3.35, 0.001)



Solve.function(3.31, 3.32, 0.001)



## Example 3: Newton-Raphson Method

### Define the Derivative and Algorithm

We first define the derivative of the function:

Deriv <- function(x) 1 - exp(x) / x + exp(x) / x^2

Next, we define the Newton-Raphson algorithm using both My.function and Deriv:

Newton <- function(x) x - My.function(x) / Deriv(x)  
  
# Example usage: Perform iterations  
Newton(3)

[1] 3.37674

Newton(3.37674)

[1] 3.31933

Newton(3.31933)

[1] 3.317628

# Perform four iterations in one go  
Newton(Newton(Newton(Newton(3))))

[1] 3.317627

### Visualize Convergence

We can visualize the function with a plot to see how the solution converges:

plot(x, My.function(x), type = "l", ylab = "f(x)",  
 main = "f(x) = x + 5 - exp(x)/x, 1 < x < 4")

