Math 6019 Assignment 1.

Problem 1:

Implement Univariate scan method which would output an interval with the minimum bracketed.

The function is defined in the python file named *UnivariateMethod.py.* This script contains two functions viz ‘univariatescan’ and ‘univariate\_plot’ to scan an interval that contains a local minimum of an objective function given an initial guess and to plot the objective function along with the bracketed values respectively.

The function ‘univariatescan’ takes the following input values:

1. ‘F’: It is the objective function whose local minimum is being calculated. The function is defined as any mathematical equation using ‘Anonymous (lambda) function’ in python.
2. ‘x0’: It is the initial guess that user inputs and has a default value of ‘0.0’.
3. ‘tol’: It is the tolerance allowed in the error calculation and has a default value of 10-8.
4. ‘alp’: It is the step size used in this calculation and has a default value of 10-2.
5. ‘maxiter’: It is the maximum # of iterations defined for stopping criteria and has a default value of 103.

The ‘univariatescan’ function returns the values ‘a’ (Lower bracketed value), ‘b’ (Upper bracketed value) and ‘k’ (# of iterations the function took to find the bracketed values).

The function ‘univariate\_plot’ takes the following inputs:

1. ‘a’: It is the lower bracketed value.
2. ‘b’: It is the upper bracketed value.
3. ‘f’: It is the objective function defined by the user.

The ‘univariate\_plot’ returns a figure in a new window showing the plot of the objective function and the scanned values for the local minimum for the given objective function.

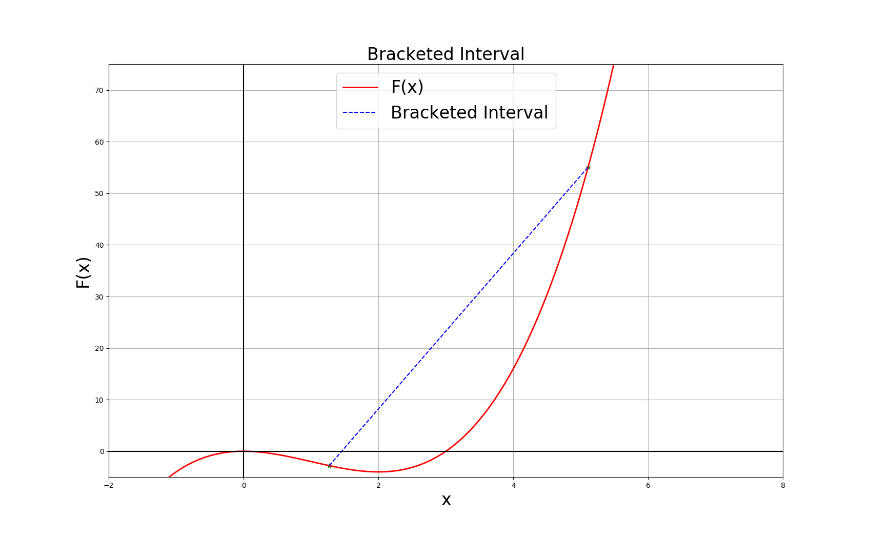
Running the script:

1. At the end of the script, I am defining an equation in ‘x’ using python’s anonymous function and storing it in the variable ‘Func’.
2. Inputting the value for initial guess and assigning it to the variable ‘X0’ (**The input value is a float value**).
3. Calling the function ‘univariatescan’ and passing the values, ‘Func’ and ‘X0’ to scan the values containing the local minimum of the given objective function. Assigning the returned values of ‘a’, ‘b’, and ‘k’ to ‘A’, ‘B’ and ‘K’ respectively.
4. Calling the function ‘univariate\_plot’ and passing the values ‘A’, ‘B’ and ‘Func’ to show the figure containing objective function plot and the scanned values containing the local minimum of the objective function.
5. Displaying the scanned values and the number of iterations ‘k’ to the console.

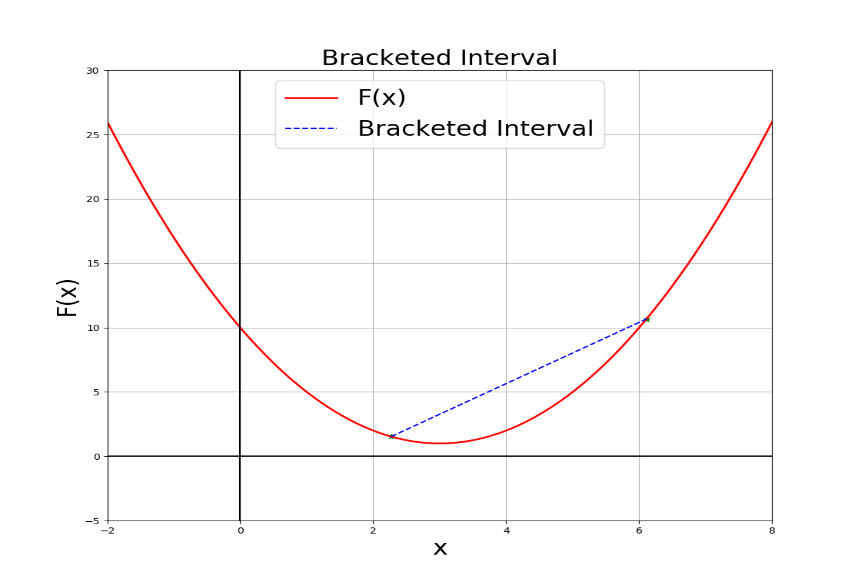
The python code is appropriately commented defining the process.

Some of the compiled results:

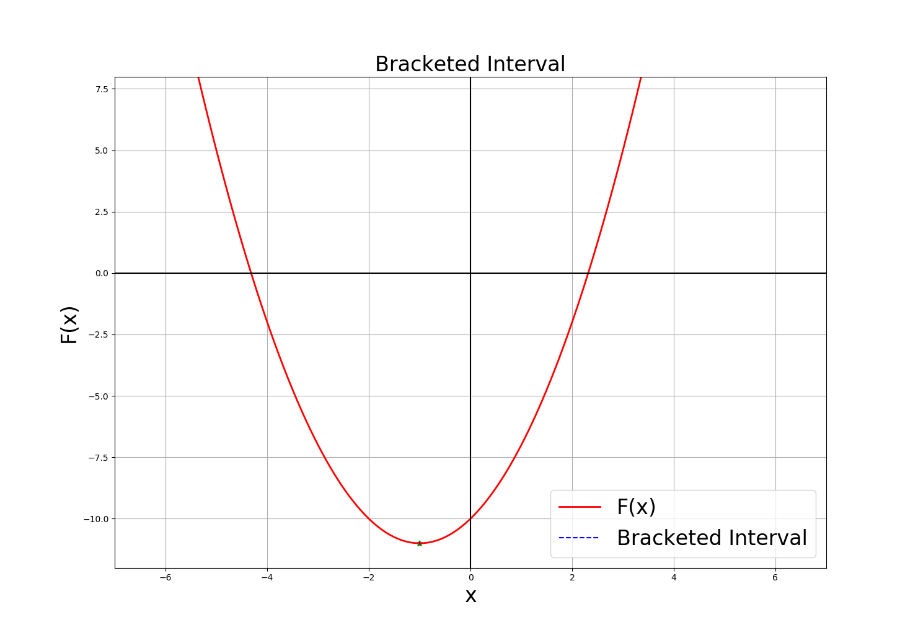
1. F(x) = x3-3x2

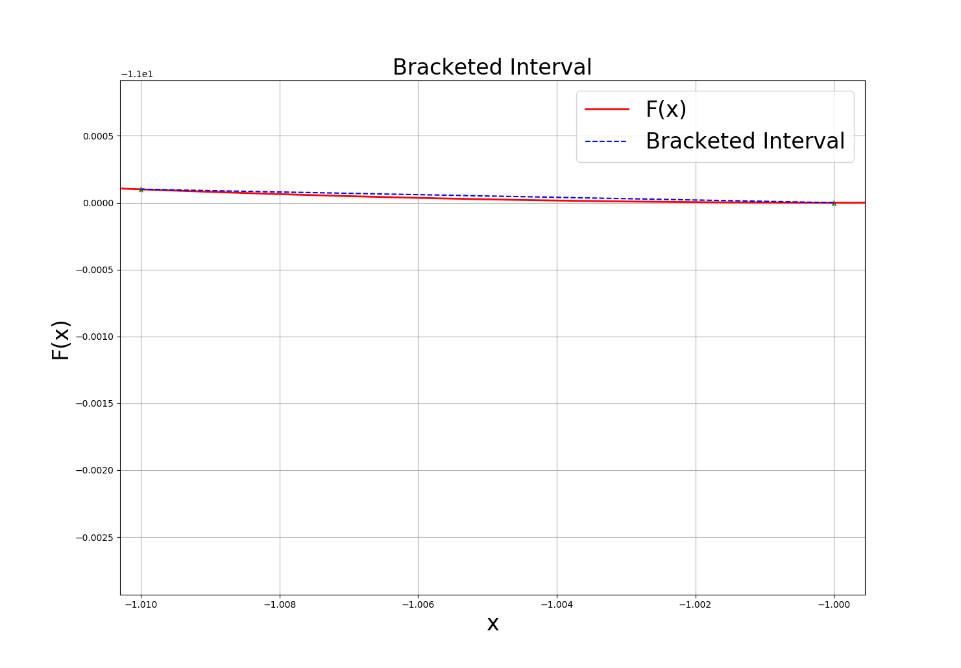


1. F(x) = x2- 6x + 10



1. F(x) = x2 + 3x-10





Problem 2:

Implement Bisection method which would output a minimum value of the objective function given two guesses that contain the minimum.

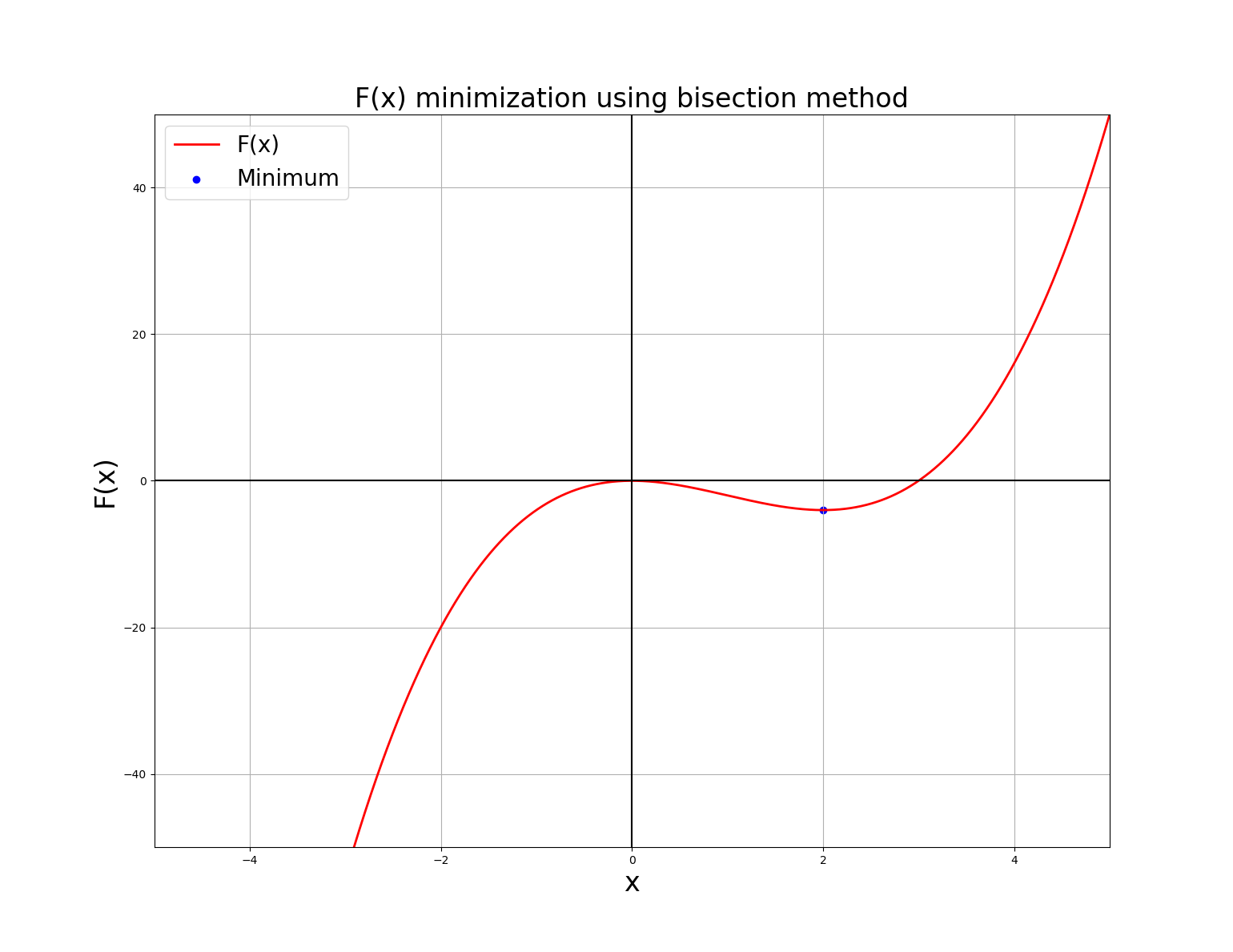
The function is defined in the python file named *bisection.py.* This is the structure of function ‘bisection(f, a, b, tol, maxiter)’ where ‘f’ is any mathematical equation defined using ‘lambda function’ in python and doesn’t have a default equation, ‘a’ is the lower interval guess point for the minimum and ‘b’ is the upper interval guess point for the minimum. ‘tol’ is the tolerance allowed in the error calculation and has a default value of 10-8, ‘maxiter’ is the maximum iterations defined for stopping criteria and has a default value of 103.

Once the function is called with above mentioned inputs, the function outputs, plots showing objective function defined by the user and the minimum. The function also returns minimum value of the objective function ‘c’ and the total no. of iterations ‘k’ it took the function to find the minimum.

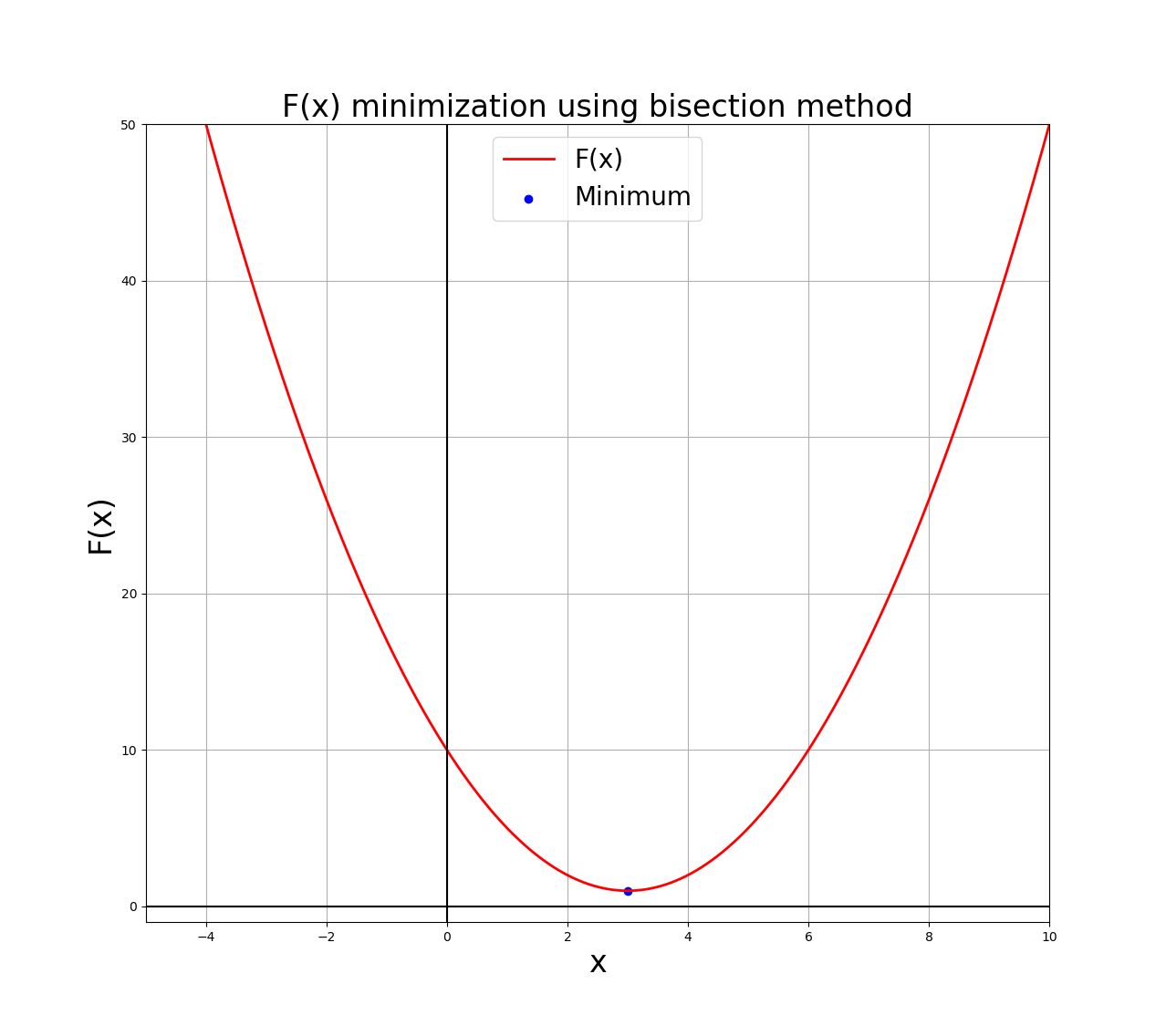
The python code is appropriately commented defining the process.

Some of the compiled results:

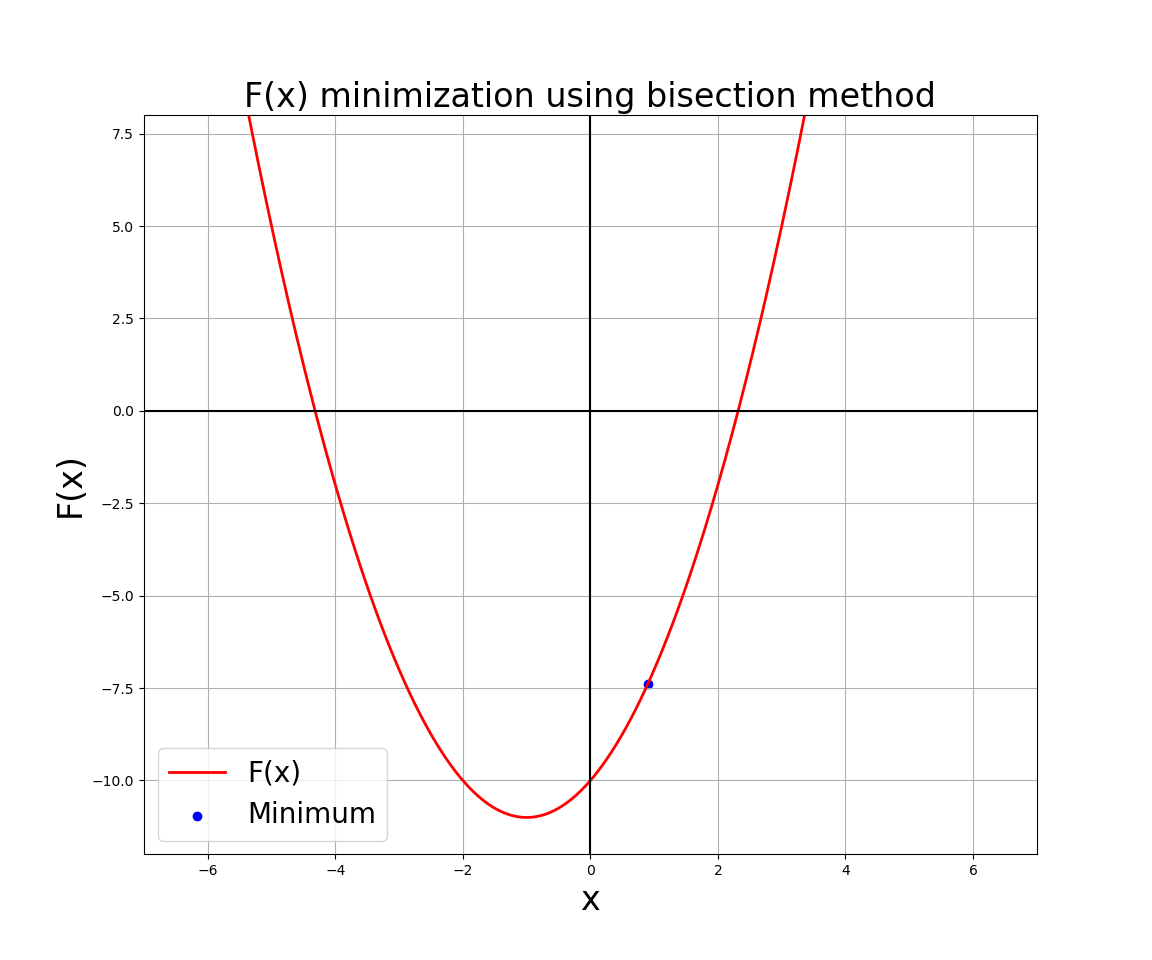
1. F(x) = x3-3x2



1. F(x) = x2- 6x + 10



1. F(x) = x2 + 3x-10



Problem 3:

Implement Newton – Raphson method which would output a minimum value of the objective function given an initial guess.

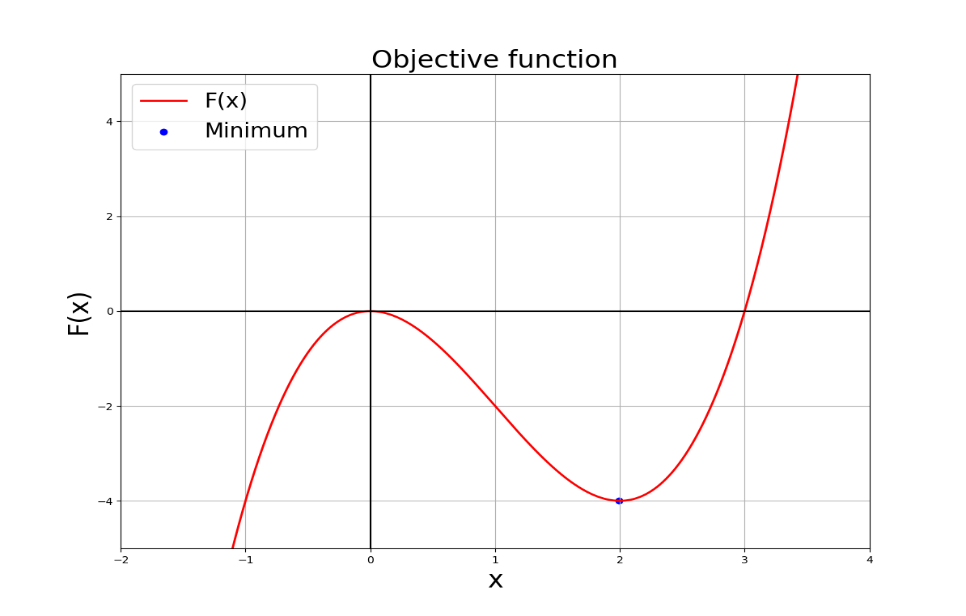
The function is defined in the python file named *Newton.py.* This is the structure of function ‘newton\_raphson(f, x0, tol, alp, maxiter)’ where ‘f’ is any mathematical equation defined using ‘lambda function’ in python and doesn’t have a default equation, ‘x0’ is the initial guess the user has to input and doesn’t have a default value. ‘tol’ is the tolerance allowed in the error calculation and has a default value of 10-8, ‘alp’ is the step size and has a default value of 10-2, ‘maxiter’ is the maximum iterations defined for stopping criteria and has a default value of 103.

Once the function is called with above mentioned inputs, the function outputs, plots showing objective function defined by the user and the minimum. The function also returns minimum value of the objective function ‘c’ and the total no. of iterations ‘k’ it took the function to find the minimum.

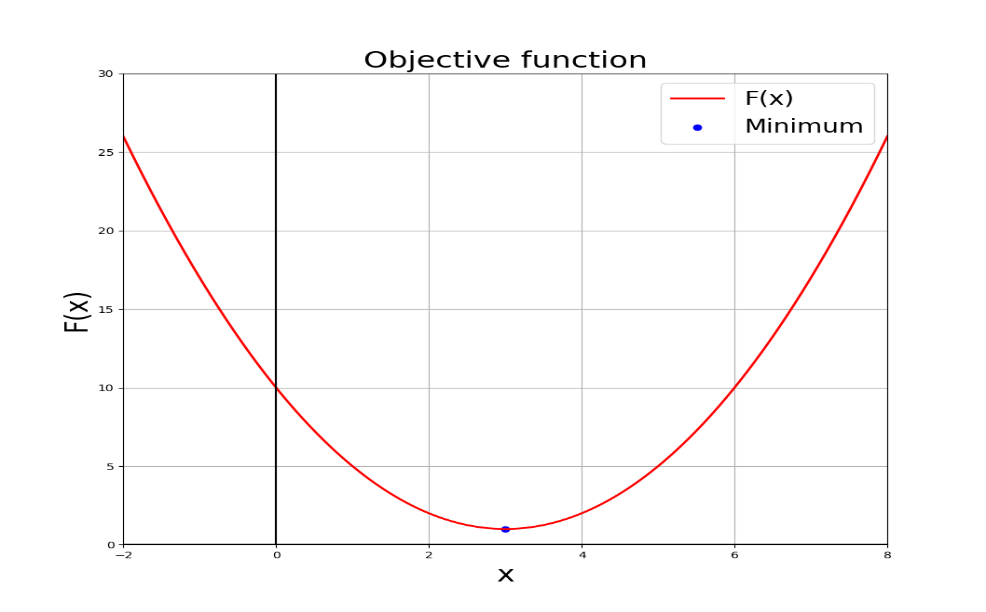
The python code is appropriately commented defining the process.

Some of the compiled results:

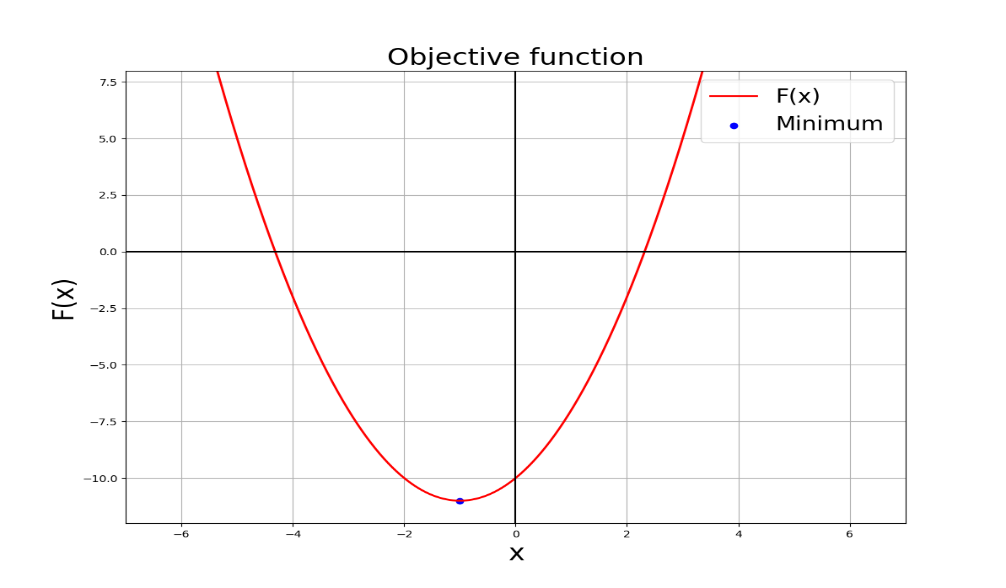
1. F(x) = x3-3x2



1. F(x) = x2- 6x + 10



1. F(x) = x2 + 3x-10



Problem 4:

The total\_least\_squares (x0=0.1, m=0.5, n=1000, d\_range=(0,5), b=0.0, eps=1e-2) function takes the following inputs

INPUT:

x0 - It takes a float value, which acts as an initial guess to Newton-Raphson Method

m - It takes a float value and it represents the slope of the input line

n - It is the no. of points that are being mapped. It takes an integer value >0

d\_range - It is the range defined for the graph

b - is the y-intercept and takes a float values

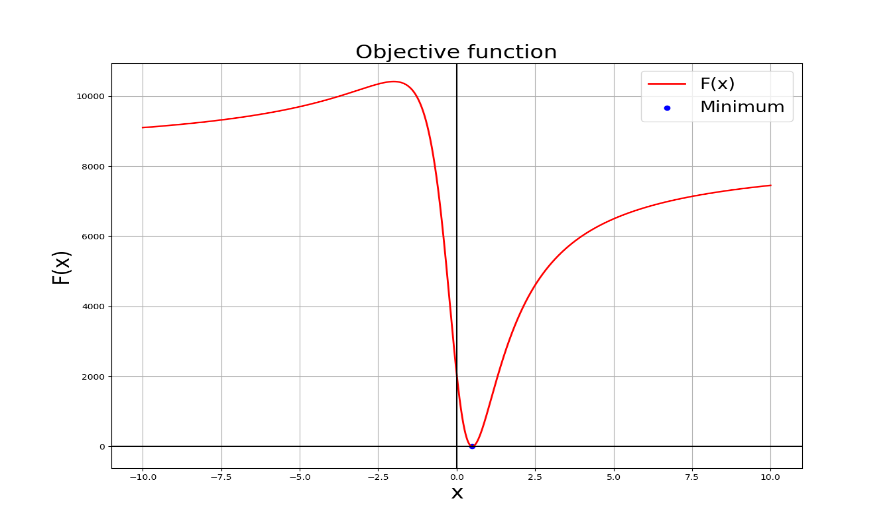
eps - represents the magnitude of the noise in the input data

OUTPUT:

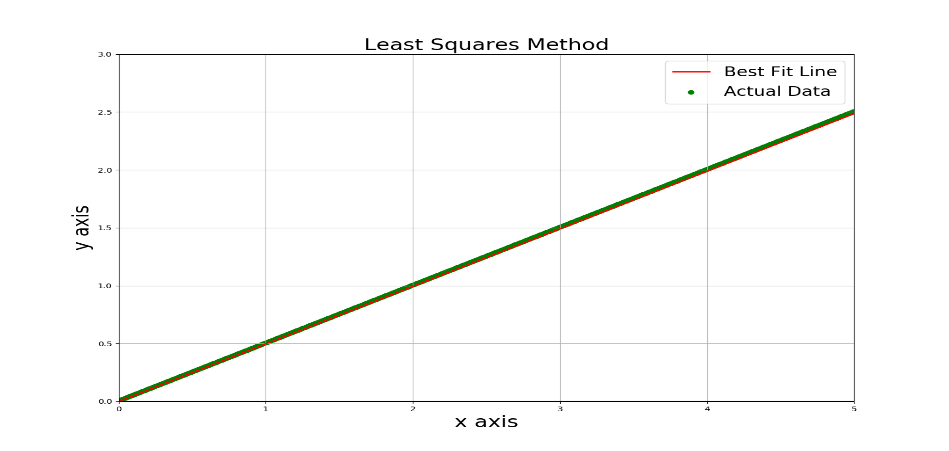
Plots showing objective function with the minimum point and the best fit line along with original data

Also prints the values of minimum x, minimum F(x) to the console.

Here are the compiled results for the default values.



Best Line Fit plot:



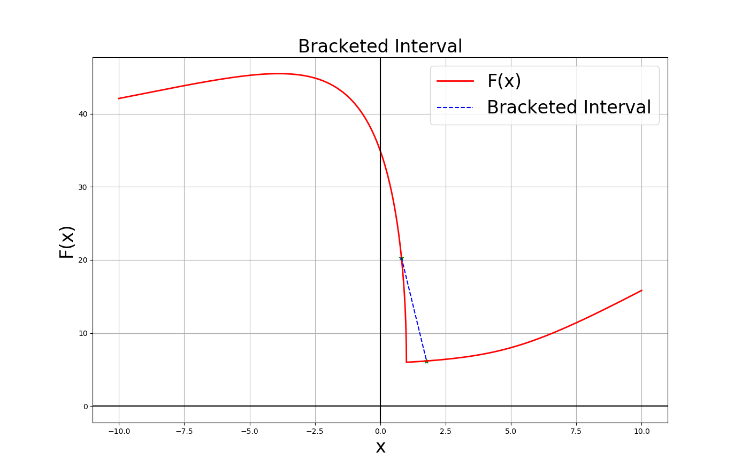
Problem 5:

The collective compiled results for a, b, c for all 3 cases are shown in following figures.

1. P = (5, 3), C = (2, 2), r = 1

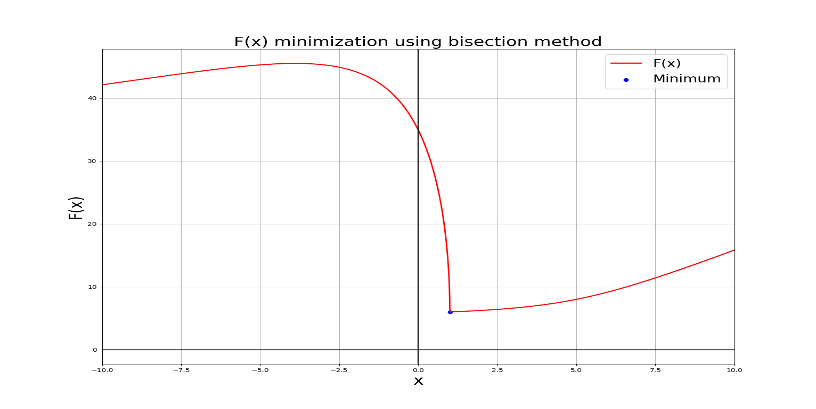
Interval containing the minimum. For initial guess value x0 = 0.5

Output a = 0.81 , b = 1.77

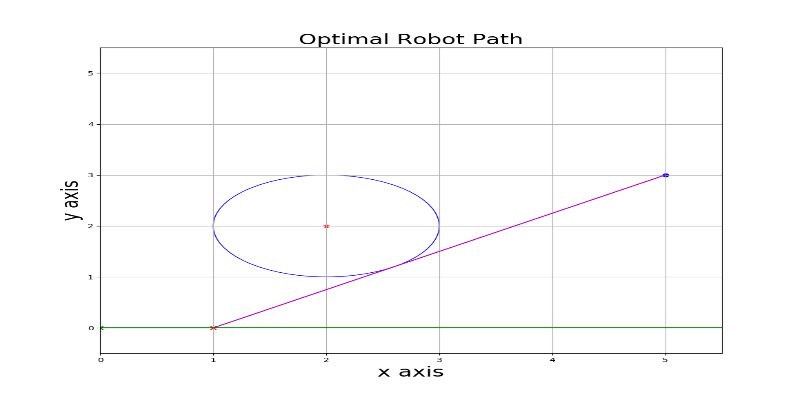


Objective function with minimum point.

Output minimum = 1.0 and F(xmin) = 6.0

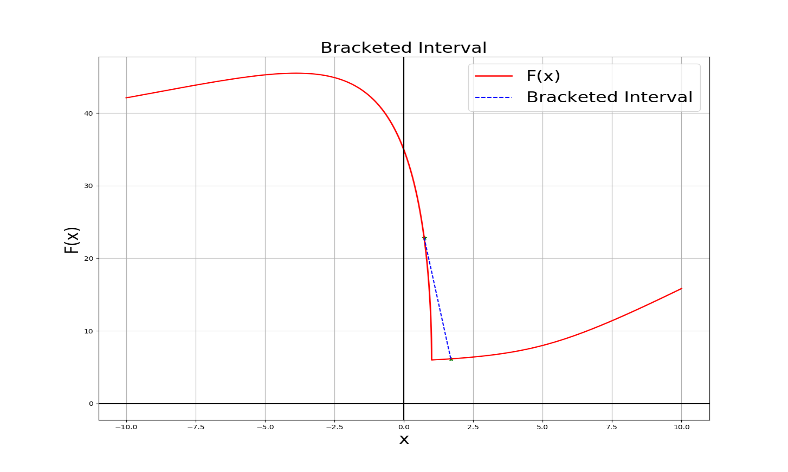


Optimal Path for Robot. The robot has to travel until x = 1.0 along x-axis to avoid the circle while reaching the point P.



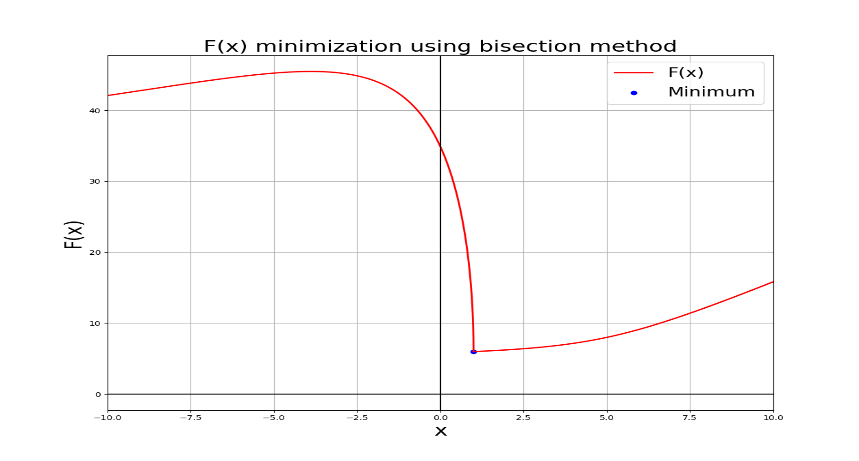
Interval containing the minimum. For initial guess value x0 = 3.0

Output a = 0.73 , b = 1.69

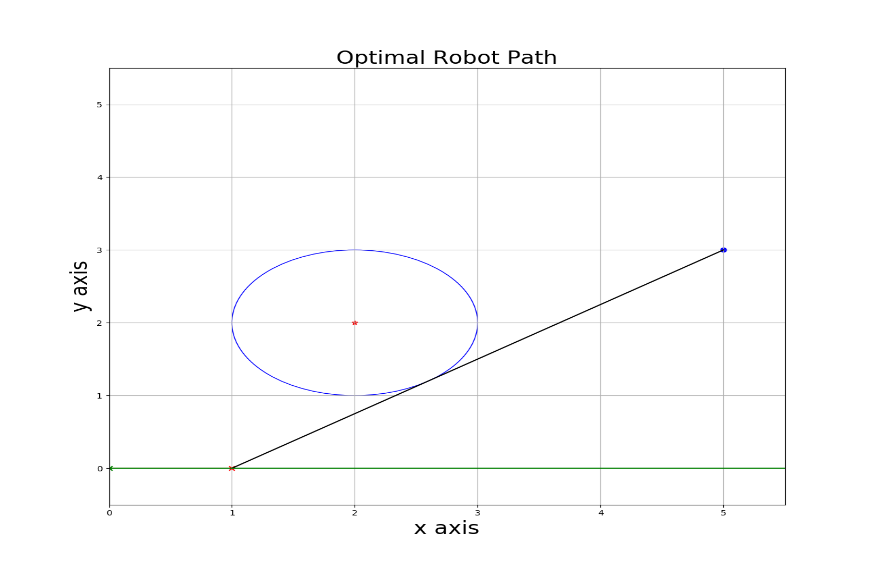


Objective function with minimum point.

Output minimum = 1.0 and F(xmin) = 6.0



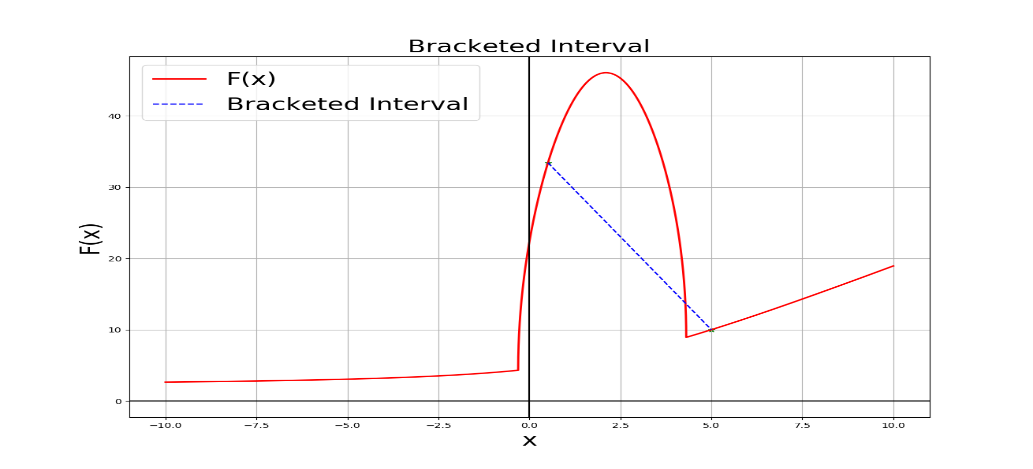
Optimal Path for Robot. The robot must travel until x = 1.0 along x-axis to avoid the circle while reaching the point P.



1. P = (2, 4), C = (2, 2), r = 1

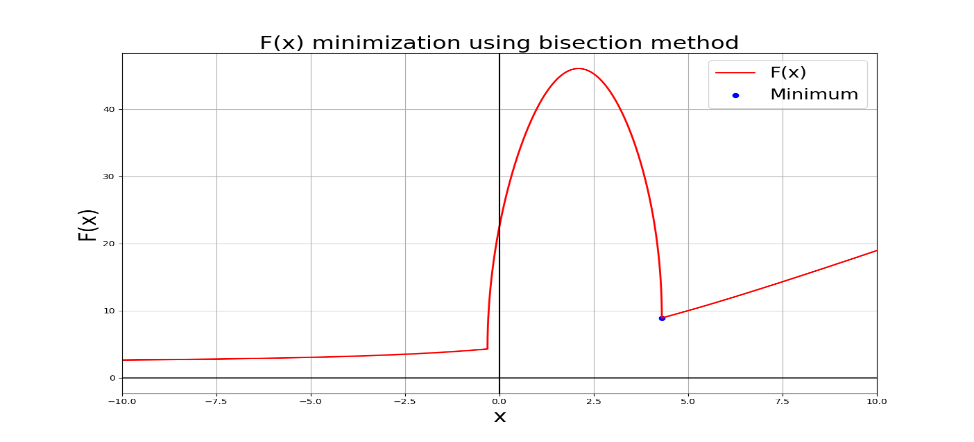
Interval containing the minimum. For initial guess value x0 = 0.5

Output: a = 0.5, b = 5.0

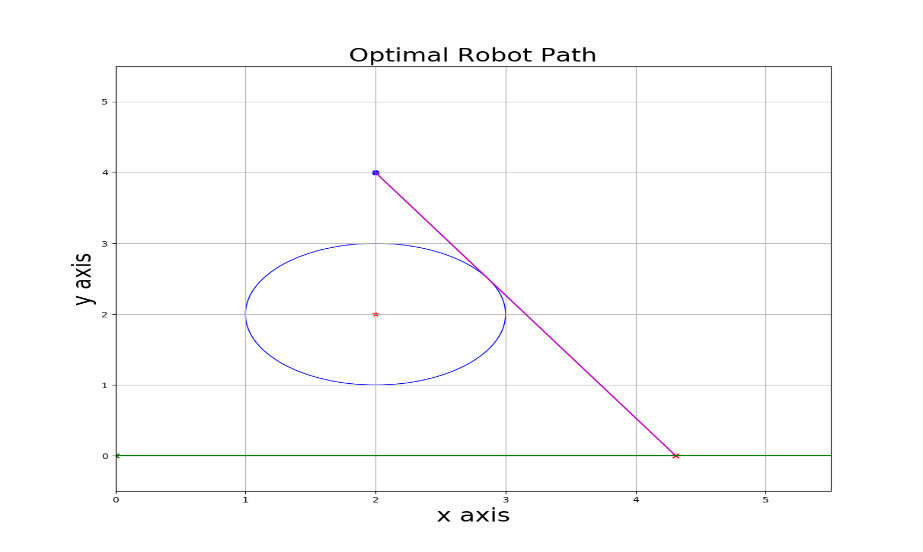


Objective function with minimum point.

Output: minimum = 4.3094 and F(xmin) = 8.9282

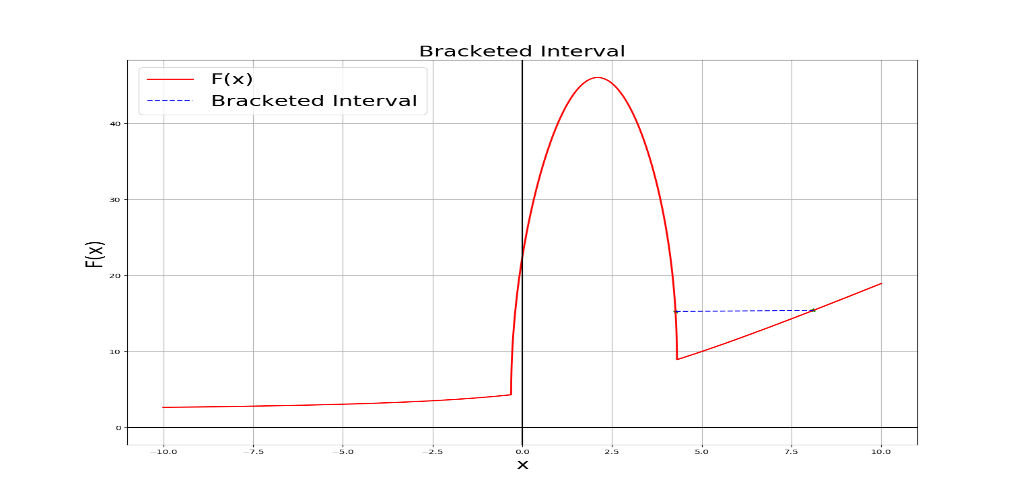


Optimal Path for Robot: The robot must travel until x = 4.3094 along x-axis to avoid the circle while reaching the point P.



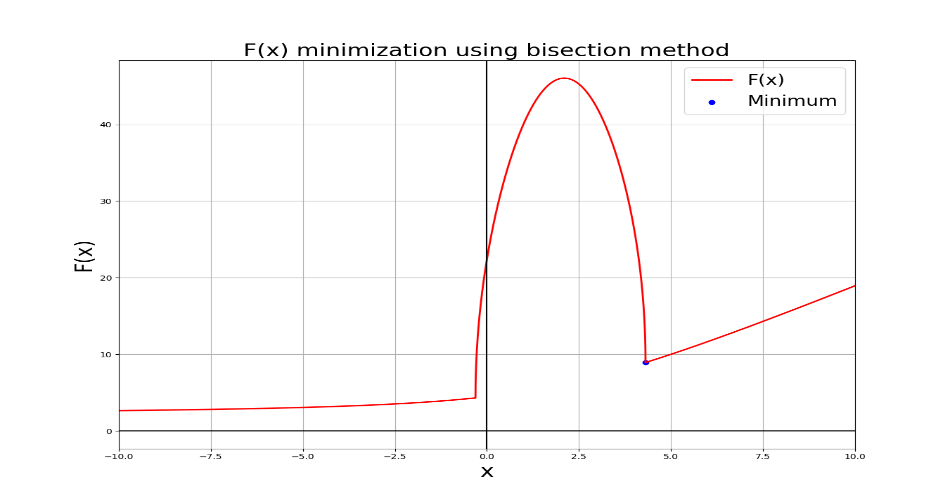
Interval containing the minimum. For initial guess value x0 = 3.0

Output: a = 4.27, b = 8.11

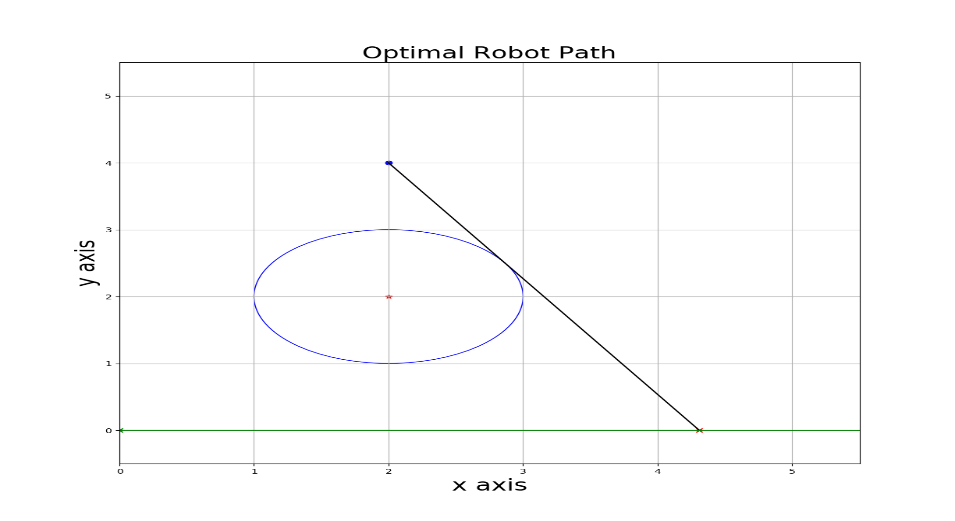


Objective function with minimum point.

Output: minimum = 4.3094 and F(xmin) = 8.9282



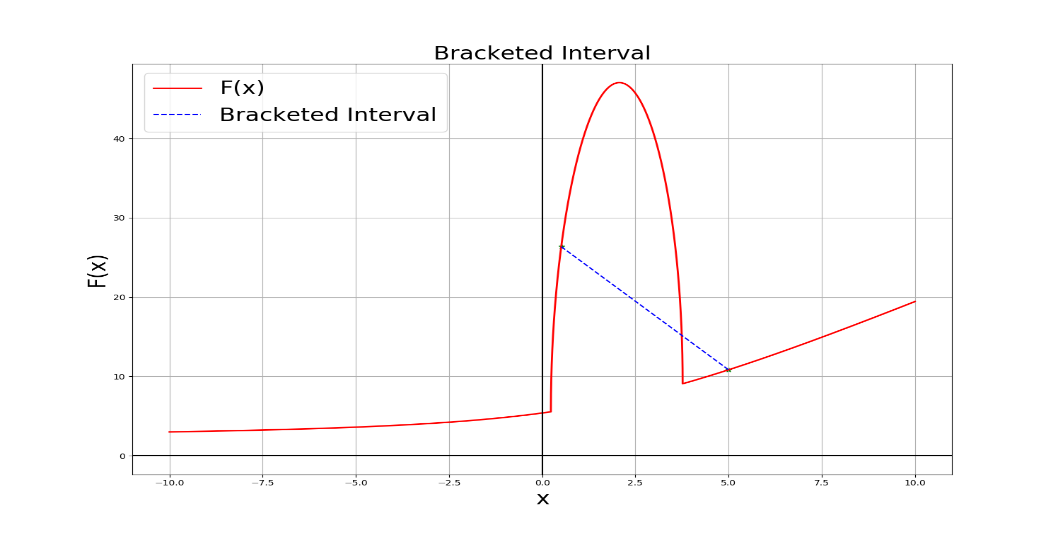
Optimal Path for Robot: The robot must travel until x = 4.3094 along x-axis to avoid the circle while reaching the point P.



1. P = (2, 5), C = (2, 2), r = 1

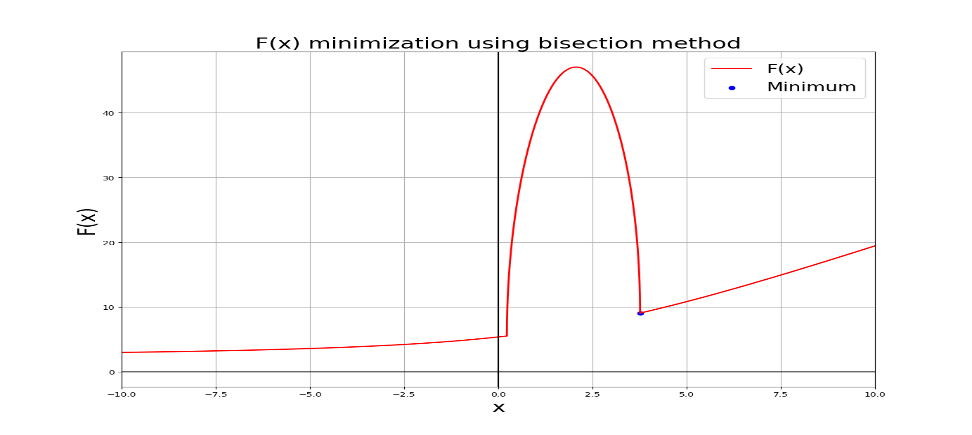
Interval containing the minimum. For initial guess value x0 = 0.5

Output: a = 0.5, b = 5.0

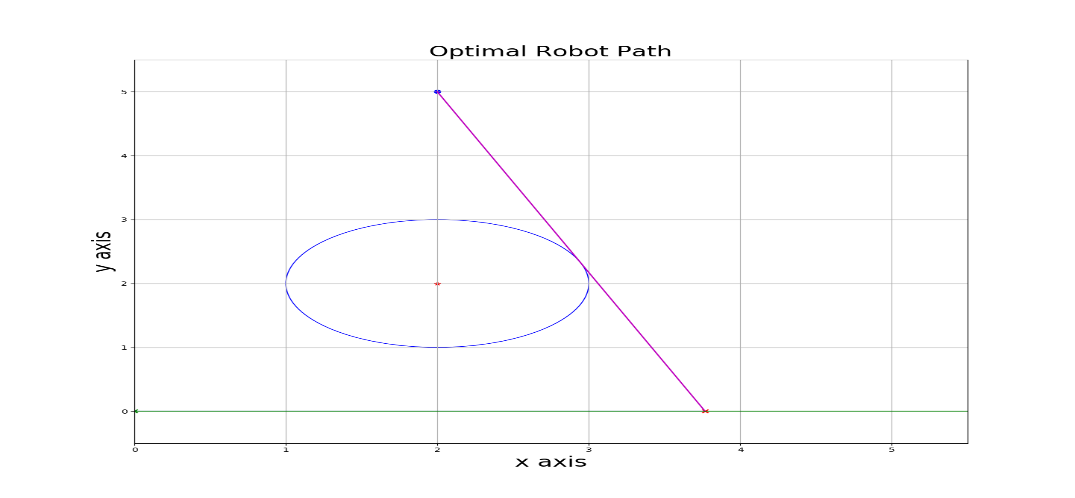


Objective function with minimum point.

Output: minimum = 3.7678 and F(xmin) = 9.0711

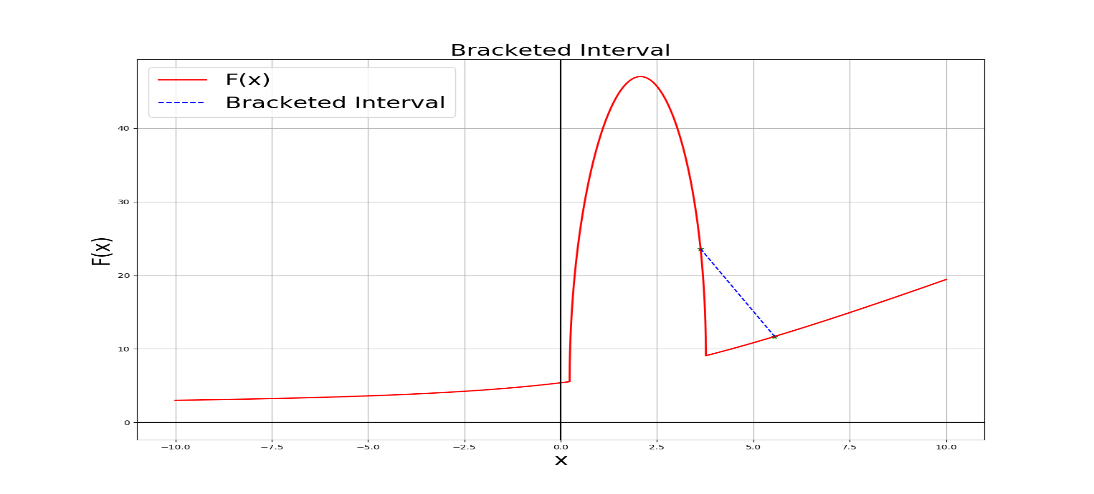


Optimal Path for Robot: The robot must travel until x = 3.7678 along x-axis to avoid the circle while reaching the point P.



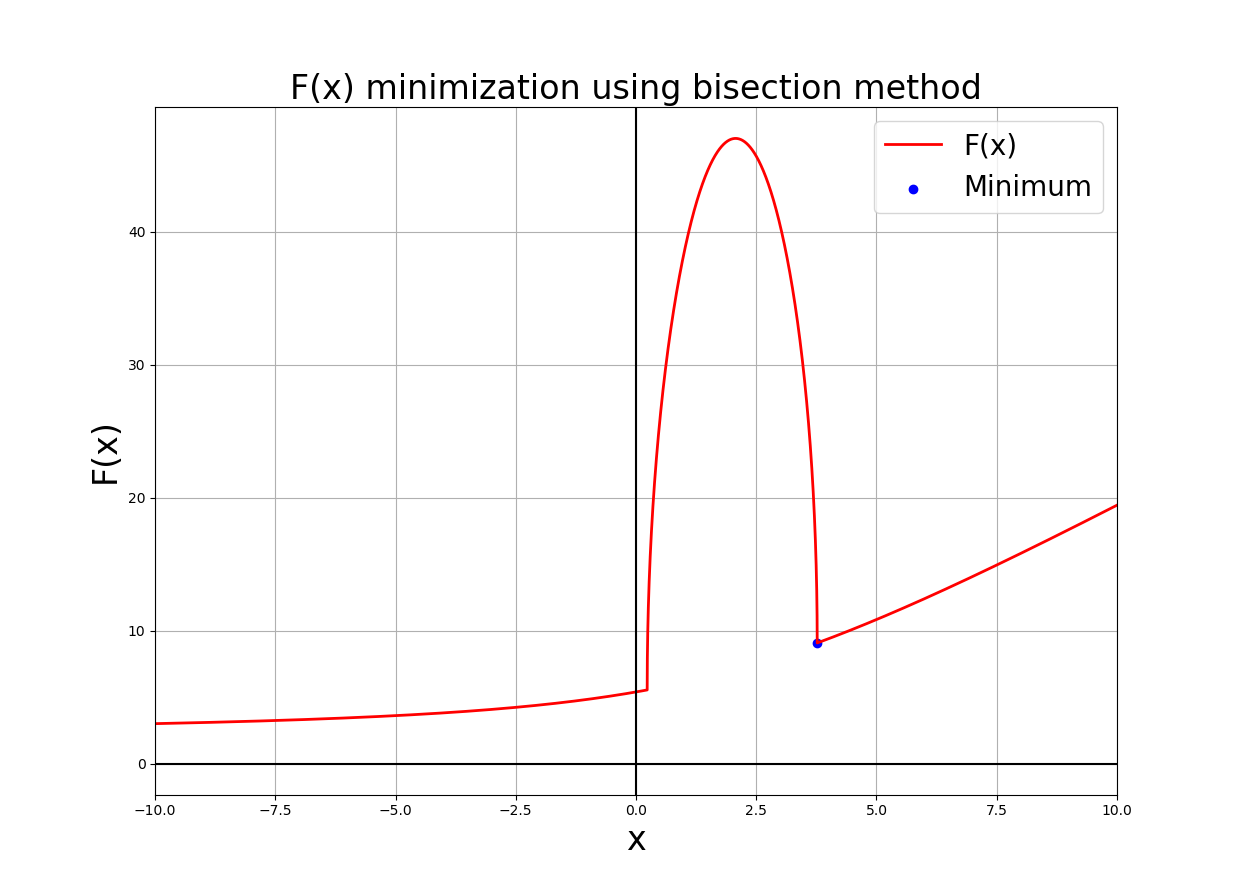
Interval containing the minimum. For initial guess value x0 = 3.0

Output: a = 3.63, b = 5.55



Objective function with minimum point.

Output: minimum = 3.7678 and F(xmin) = 9.0711



Optimal Path for Robot. The robot must travel until x = 3.7678 along x-axis to avoid the circle while reaching the point P.

