Lab Problem: 06.

Course code: CCE-312.
Course Title: Numerical Methods sessional.

Name of the Lab Report: Solve Real world problem using False-position Method after that implement it by Python.

Remarks & Signature:

Submitted To

Professor Dr. Md. Samsuzzaman.
Professor,
Department of Computer and Communication Engineering.
Faculty of Computer Science & Engineering.

Submitted By

HASAN AHAMMAD
ID No: 1902073
Reg No: 08779
Level- 3, Semester- 1
Session: 2019-2020
Faculty of Computer Science & Engineering.

Patuakhali Science & Technology University.
Dumki, Patuakhali-8602.

1.

Problem Statement. Use the graphical approach to determine the drag coefficient c needed for a parachutist of mass $m = 68.1$ kg to have a velocity of 40 m/s after free-falling for time $t = 10$ s. *Note:* The acceleration due to gravity is 9.8 m/s^2.

Solution. This problem can be solved by determining the root of Eq. (PT2.4) using the parameters $t = 10$, $g = 9.8$, $v = 40$, and $m = 68.1$:

$$f(c) = \frac{9.8(68.1)}{c}\left(1 - e^{-(c/68.1)10}\right) - 40$$

or

$$f(c) = \frac{667.38}{c}\left(1 - e^{-0.146843c}\right) - 40 \tag{E5.1.1}$$

Solution. As in Example 5.3, initiate the computation with guesses of $x_l = 12$ and $x_u = 16$.

First iteration:

$$x_l = 12 \qquad f(x_l) = 6.0699$$
$$x_u = 16 \qquad f(x_u) = -2.2688$$
$$x_r = 16 - \frac{-2.2688(12 - 16)}{6.0669 - (-2.2688)} = 14.9113$$

which has a true relative error of 0.89 percent.

Second iteration:

$$f(x_l)\,f(x_r) = -1.5426$$

Therefore, the root lies in the first subinterval, and x_r becomes the upper limit for the next iteration, $x_u = 14.9113$:

$$x_l = 12 \qquad f(x_l) = 6.0699$$
$$x_u = 14.9113 \qquad f(x_u) = -0.2543$$
$$x_r = 14.9113 - \frac{-0.2543(12 - 14.9113)}{6.0669 - (-0.2543)} = 14.7942$$

which has true and approximate relative errors of 0.09 and 0.79 percent. Additional iterations can be performed to refine the estimate of the roots.

✪ Implement using python:

```
import math


def function_to_find_root(c):
    return (667.38 / c) * (1 - math.exp(-0.146843 * c)) - 40


def false_position_method(func, a, b, tol=1e-6, max_iter=100):

    if func(a) * func(b) > 0:
```

```python
        raise ValueError("The function must have different signs at the interval endpoints.")

    iterations = 0
    while iterations < max_iter:
        c = (a * func(b) - b * func(a)) / (func(b) - func(a))

        if abs(func(c)) < tol:
            return c, iterations

        if func(c) * func(a) < 0:
            b = c
        else:
            a = c

        iterations += 1

    raise ValueError("False-position method did not converge within the maximum number of iterations.")

x1 = 12
x2 = 16
tolerance = 1e-6

root, iterations = false_position_method(function_to_find_root, x1, x2, tol=tolerance)
print(f"Approximated root: {root:.6f}")
print(f"Iterations: {iterations}")
```

2.

Problem Statement. Use bisection and false position to locate the root of

$$f(x) = x^{10} - 1$$

between $x = 0$ and 1.3.

Solution. Using bisection, the results can be summarized as

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.65	100.0	35
2	0.65	1.3	0.975	33.3	2.5
3	0.975	1.3	1.1375	14.3	13.8
4	0.975	1.1375	1.05625	7.7	5.6
5	0.975	1.05625	1.015625	4.0	1.6

Thus, after five iterations, the true error is reduced to less than 2 percent. For false position, a very different outcome is obtained:

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.09430		90.6
2	0.09430	1.3	0.18176	48.1	81.8
3	0.18176	1.3	0.26287	30.9	73.7
4	0.26287	1.3	0.33811	22.3	66.2
5	0.33811	1.3	0.40788	17.1	59.2

✪ **Implement using Python:**

```
✪ def false_position_method(func, a, b, tol=1e-6, max_iter=100):
    if func(a) * func(b) > 0:
        raise ValueError("The function must have different signs at the interval endpoints.")

    iterations = 0
    while iterations < max_iter:
        c = (a * func(b) - b * func(a)) / (func(b) - func(a))

        if abs(func(c)) < tol:
            return c, iterations

        if func(c) * func(a) < 0:
            b = c
        else:
            a = c

        iterations += 1

    raise ValueError("False-position method did not converge within the maximum number of iterations.")


# Example usage:
def example_function(x):
    return x ** 10 - 1


a = 0
b = 1.3
tolerance = 1e-6

root, iterations = false_position_method(example_function, a, b, tol=tolerance)
print(f"Approximated root: {root:.6f}")
print(f"Iterations: {iterations}")
```