University of Dhaka

Department of Computer Science and Engineering

CSE3212: Numerical Methods Lab

3rd Year 2nd Semester
Session: 2021-22

Implementation and Comparative Analysis of the Bissection, False Position, Newton–Raphson, and Secant Methods for finding roots of nonlinear equations.

Submitted by:

Jannatul Ferdousi (08)

Submitted To:

Mr. Palash Roy, Lecturer, Dept. of CSE, University of Dhaka
Dr. Muhammad Ibrahim, Associate Professor, Dept. of CSE, University of Dhaka

Submission Date: October 11, 2025

Contents

1 Introduction

Root-finding is a fundamental problem in numerical analysis, where the objective is to find the values of a variable for which a given function equals zero. These values are known as roots, or zeros, of the function. In many engineering and computational problems, analytical solutions are often difficult or impossible to obtain, so iterative numerical approaches are employed instead.

In this lab, the objective is to find the height h in a draining tank that satisfies the flow balance equation:

$$f(h) = h^3 - 10h + 5e^{-h/2} - 2 = 0$$

The following four root-finding algorithms are applied:

- **Bisection Method:** A bracketing approach based on the Intermediate Value Theorem.
- **False Position (Regula Falsi):** A bracketing method that uses linear interpolation.
- **Newton–Raphson Method:** An open method that relies on tangents and derivatives.
- **Secant Method:** An open, derivative-free approach similar to Newton–Raphson.

2 Objectives

The primary objectives of this lab report are:

1. To implement the Bisection and False-Position methods to find the root of the given nonlinear equation within a suitable interval, ensuring an approximate relative error (ϵ_a) of less than or equal to 0.001%.
2. To implement the Newton-Raphson and Secant methods using specified initial guesses to determine the root, also targeting an approximate relative error (ϵ_a) of less than or equal to 0.001%.
3. To tabulate all necessary data, including iteration numbers, approximate root values, function values, and approximate relative errors, for each method.
4. To compare the convergence rates of all implemented methods by plotting the approximate relative error (ϵ_a) versus the number of iterations and providing a detailed analysis.

3 Algorithms

3.1 Bisection Method

1. Choose an interval $[x_l, x_u]$ such that $f(x_l) \cdot f(x_u) < 0$, guaranteeing a root exists within the interval.
2. Calculate the midpoint $x_r = (x_l + x_u)/2$.
3. If $f(x_l) \cdot f(x_r) < 0$, the root is in $[x_l, x_r]$, so set $x_u = x_r$.
4. If $f(x_l) \cdot f(x_r) > 0$, the root is in $[x_r, x_u]$, so set $x_l = x_r$.
5. If $f(x_l) \cdot f(x_r) = 0$, then x_r is the root.
6. Repeat steps 2-5 until the approximate relative error ϵ_a is below the threshold or the maximum number of iterations is reached.

3.2 False Position Method

1. Choose an interval $[x_l, x_u]$ such that $f(x_l) \cdot f(x_u) < 0$.
2. Estimate the new root $x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$.
3. If $f(x_l) \cdot f(x_r) < 0$, the root is in $[x_l, x_r]$, so set $x_u = x_r$.
4. If $f(x_l) \cdot f(x_r) > 0$, the root is in $[x_r, x_u]$, so set $x_l = x_r$.
5. If $f(x_l) \cdot f(x_r) = 0$, then x_r is the root.
6. Repeat steps 2-5 until the approximate relative error ϵ_a is below the threshold or the maximum number of iterations is reached.

3.3 Newton-Raphson Method

1. Choose an initial guess x_0.
2. Calculate the next approximation $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.
3. Set $x_0 = x_1$.
4. Repeat steps 2-3 until the approximate relative error ϵ_a is below the threshold or the maximum number of iterations is reached.

3.4 Secant Method

1. Choose two initial guesses x_0 and x_1.
2. Calculate the next approximation $x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$.
3. Set $x_0 = x_1$ and $x_1 = x_2$.
4. Repeat steps 2-3 until the approximate relative error ϵ_a is below the threshold or the maximum number of iterations is reached.

4 Implementation

4.1 Function Definitions

```python
import math
import matplotlib.pyplot as plt

def f(h):
    return h**3 - 10*h + 5*math.exp(-h/2) - 2

def df(h):
    return 3*h**2 - 10 - (5/2)*math.exp(-h/2)
```

4.2 Bisection Method Implementation

```python


def bisection(f, xl, xu, max_it=100, rel_er_thr=None):
    fl = f(xl)
    fu = f(xu)
    if fl * fu > 0:
        return ValueError("The interval is not valid")

    results = []
    it = 0
    old_xr = None
    while it < max_it:
        xr = (xl + xu)/2
        fr = f(xr)
        ea = None if old_xr is None else abs((xr - old_xr)/xr)*100

        results.append({
            'iteration': it+1,
            'xl': xl,
            'xu': xu,
            'xr': xr,
            'f(xl)' : fl,
            'f(xu)' : fu,
            'f(xr)': fr,
            'ea(%)': ea
        })

        if ea is not None and rel_er_thr is not None and ea <
           rel_er_thr:
            print(f"Bisection approximate root found at x =
               {xr:.10f} (iter {it+1})")
            break

        if fl * fr < 0:
            xu = xr
            fu = fr
        elif fl * fr > 0:
            xl = xr
```

```
            fl = fr
        else:
            print(f"The real root is: {xr:.10f}")
            break
        old_xr = xr
        it += 1

    return results
```

4.3 False Position Method Implementation

```

def false_position(f, xl, xu, max_it=100, rel_er_thr=None):
    fl = f(xl)
    fu = f(xu)
    if fl * fu > 0:
        return ValueError("The interval is not valid")

    results = []
    it = 0
    old_xr = None

    while it < max_it:
        xr = (xl*fu - xu*fl)/(fu - fl)
        fr = f(xr)
        ea = None if old_xr is None else abs((xr - old_xr)/xr)*100

        results.append({
            'iteration': it+1,
            'xl': xl,
            'xu': xu,
            'xr': xr,
            'f(xl)' : fl,
            'f(xu)' : fu,
            'f(xr)': fr,
            'ea(%)': ea
        })

        if ea is not None and rel_er_thr is not None and ea <
           rel_er_thr:
            print(f"False Position approximate root found at x =
               {xr:.10f} (iter {it+1})")
            break

        if fl * fr < 0:
            xu = xr
            fu = fr
        elif fl * fr > 0:
            xl = xr
            fl = fr
        else :
            print(f"The real root is: {xr:.10f}")
            break

```

```
        old_xr = xr
        it += 1

    return results
```

4.4 Newton Raphson Method Implementation

```

def newton_raphson(f, df, x0, max_it=100, rel_er_thr=None):
    results = []
    it = 0
    while it < max_it:
        fx0 = f(x0)
        dfx0 = df(x0)
        if dfx0 == 0:
            print("Division by zero encountered")
            break
        x1 = x0 - fx0/dfx0
        ea = abs((x1 - x0)/x1)*100
        results.append({
            'iteration': it+1,
            'm_k': x0,
            'm_r': x1,
            'f(m_k)': fx0,
            "f'(m_k)": dfx0,
            'ea(%)': ea
        })
        if ea is not None and rel_er_thr is not None and ea <
           rel_er_thr:
            print(f"Approximate Newton Raphson root found at x =
               {x1:.10f} on iteration {it+1} (relative error =
               {ea:.2e} <= threshold {rel_er_thr})")
            break
        if abs(f(x1)) < 1e-12:
            print(f"Approximate Newton Raphson root found at x =
               {x1:.10f} on iteration {iter+1} (|f(m_r)| <
               1e-12)")
            break
        x0 = x1
        it += 1
    return results
```

4.5 Secant Method Implementation

```

def secant(f, x0, x1, max_iteration=100, rel_er_thr=None):
    f0=f(x0)
    f1=f(x1)
    results = []
    iteration = 0
    while iteration < max_iteration:
```

```
        if abs(f1 - f0) < 1e-12:
            print("Division by Zero or Subtract Cancellation Encountered.")
            break
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
        f2 = f(x2)
        ea = abs((x2 - x1) / x2) * 100
        results.append({
            'iteration': iteration+1,
            'x_i-1': x0,
            'x_i': x1,
            'x_i+1': x2,
            'f(x_i-1)': f0,
            'f(x_i)': f1,
            'f(x_i+1)': f2,
            'ea(%)': ea
        })

        if abs(f2) < 1e-12:
            print(f"Approximate Secant root found at x = {x2:.10f} on iteration {iteration+1} (|f(x)| < 1e-12)")
            break
        if ea is not None and rel_er_thr is not None and ea <= rel_er_thr:
            print(f"Approximate Secant root found at x = {x2:.10f} on iteration {iteration+1} (relative error = {ea:.2e} <= threshold {rel_er_thr})")
            break
        x0 = x1
        f0 = f1
        x1 = x2
        f1 = f2
        iteration += 1

    return results
```

4.6 Generic printing function and rest of the code

```

def print_table(results, method_name):
    """Generic function to print tables for any root-finding method"""
    if not results:
        print(f"No results to display for {method_name}")
        return

    columns = [key for key in results[0].keys() if key != 'iteration']

    width = 6 + len(columns) * 12 + (len(columns) - 1)

    print("\n" + "="*width)
    print(method_name.upper())
    print("="*width)
```

```
15
16      header = f"{'Iter':<6} "
17      for col in columns:
18          header += f"{col:<12} "
19      print(header)
20      print("-"*width)
21
22      for r in results:
23          row = f"{r['iteration']:<6} "
24          for col in columns:
25              value = r[col]
26              if value is None or (col == 'ea(%)' and value is None):
27                  row += f"{'N/A':<12} "
28              elif isinstance(value, (int, float)):
29                  row += f"{value:<12.5f} "
30              else:
31                  row += f"{str(value):<12} "
32          print(row)
33
34      print("="*width)
35
36
37
38  rel_err = 0.001
39
40  bisection_results = bisection(f, 0.1, 0.4, rel_er_thr=rel_err)
41  print_table(bisection_results, "Bisection Method")
42  falsepos_results = false_position(f, 0.1, 0.4, rel_er_thr=rel_err)
43  print_table(falsepos_results, "False Position Method")
44  newton_results = newton_raphson(f, df, 1.5, rel_er_thr=rel_err)
45  print_table(newton_results, "Newton-Raphson Method")
46  secant_results = secant(f, 1.5, 2.0, rel_er_thr=rel_err)
47  print_table(secant_results, "Secant Method")
48
49  plt.figure(figsize=(10, 6))
50  def plot_errors(results, label):
51      iterations = [r['iteration'] for r in results if r['ea(%)'] is
        not None]
52      errors = [r['ea(%)'] for r in results if r['ea(%)'] is not
        None]
53      plt.plot(iterations, errors, marker='o', label=label)
54
55  plot_errors(bisection_results, 'Bisection')
56  plot_errors(falsepos_results, 'False Position')
57  plot_errors(newton_results, 'Newton-Raphson')
58  plot_errors(secant_results, 'Secant')
59
60  plt.xlabel('Iteration')
61  plt.ylabel('Approximate Error (%)')
62  plt.title('Convergence of Root-Finding Methods')
63  plt.grid(True, alpha=0.3)
64  plt.legend()
65  plt.show()
```

5 Output

5.1 Bisection Method Output

The Bisection method applied with an initial interval of $[0.1, 0.4]$ and a relative error threshold of 0.001% converged to $h = 0.2440238953$ in 17 iterations. The error decreases steadily by approximately half with each iteration.

```
Bisection approximate root found at x = 0.2440238953 (iter 17)

================================================================================================
BISECTION METHOD
================================================================================================
Iter    xl          xu          xr          f(xl)       f(xu)       f(xr)       ea(%)
------------------------------------------------------------------------------------------------
1       0.10000     0.40000     0.25000     1.75715     -1.84235    -0.07189    N/A
2       0.10000     0.25000     0.17500     1.75715     -0.07189    0.83645     42.85714
3       0.17500     0.25000     0.21250     0.83645     -0.07189    0.38059     17.64706
4       0.21250     0.25000     0.23125     0.38059     -0.07189    0.15391     8.10811
5       0.23125     0.25000     0.24063     0.15391     -0.07189    0.04090     3.89610
6       0.24063     0.25000     0.24531     0.04090     -0.07189    -0.01552    1.91083
7       0.24063     0.24531     0.24297     0.04090     -0.01552    0.01268     0.96463
8       0.24297     0.24531     0.24414     0.01268     -0.01552    -0.00142    0.48000
9       0.24297     0.24414     0.24355     0.01268     -0.00142    0.00563     0.24058
10      0.24355     0.24414     0.24385     0.00563     -0.00142    0.00210     0.12014
11      0.24385     0.24414     0.24399     0.00210     -0.00142    0.00034     0.06004
12      0.24399     0.24414     0.24407     0.00034     -0.00142    -0.00054    0.03001
13      0.24399     0.24407     0.24403     0.00034     -0.00054    -0.00010    0.01501
14      0.24399     0.24403     0.24401     0.00034     -0.00010    0.00012     0.00750
15      0.24401     0.24403     0.24402     0.00012     -0.00010    0.00001     0.00375
16      0.24402     0.24403     0.24403     0.00001     -0.00010    -0.00005    0.00188
17      0.24402     0.24403     0.24402     0.00001     -0.00005    -0.00002    0.00094
================================================================================================
```

Bisection Method Output Table

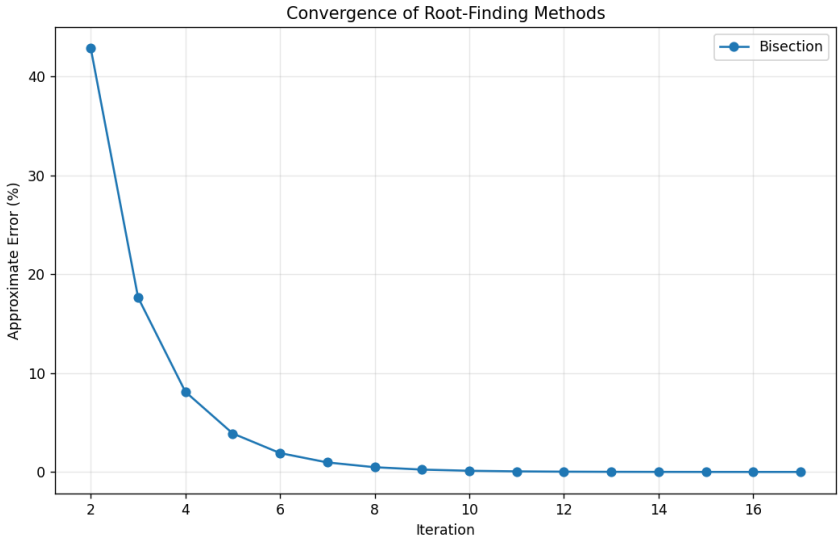

Figure 1: Error vs Iterations (Bisection Method)

5.2 False Position Method Output

The False Position method applied with the same initial interval of $[0.1, 0.4]$ and a relative error threshold of 0.001% converged to $h = 0.2440223280$ in just 4 iterations. This is due to its use of the secant line to make a better estimate of the root, resulting in superlinear convergence, which is typically faster than the linear convergence of Bisection.

False Position Method Output Table

```
False Position approximate root found at x = 0.2440223280 (iter 4)

================================================================================================
FALSE POSITION METHOD
================================================================================================
Iter    xl          xu          xr          f(xl)       f(xu)       f(xr)       ea(%)
------------------------------------------------------------------------------------------------
1       0.10000     0.40000     0.24645     1.75715     -1.84235    -0.02920    N/A
2       0.10000     0.24645     0.24406     1.75715     -0.02920    -0.00040    0.98095
3       0.10000     0.24406     0.24402     1.75715     -0.00040    -0.00001    0.01340
4       0.10000     0.24402     0.24402     1.75715     -0.00001    -0.00000    0.00018
================================================================================================
```

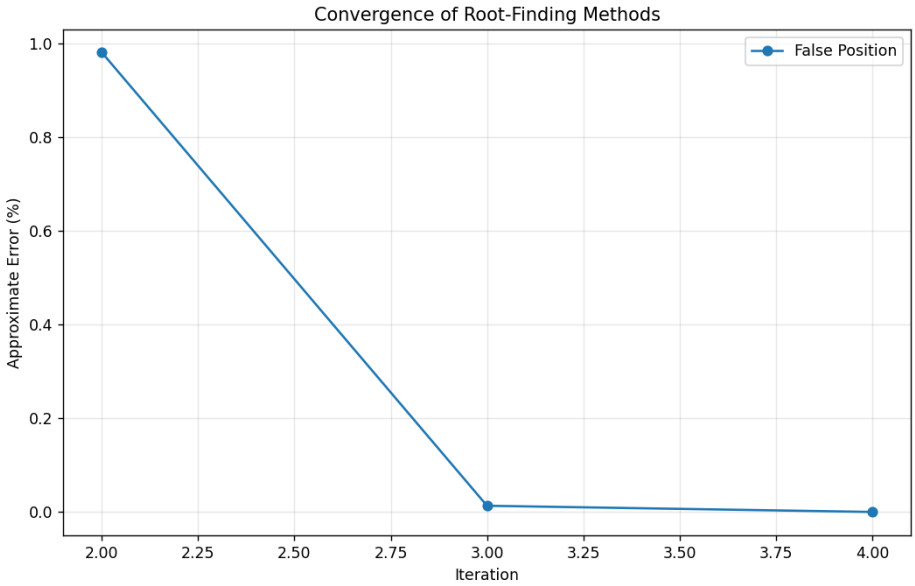

Figure 2: Error vs Iterations (False Position Method)

5.3 Newton-Raphson Method Output

The Newton-Raphson method applied with an initial guess $h_0 = 1.5$ and a relative error threshold of 0.001% converged to $h = 0.2440223219$ in 5 iterations. Despite an initial guess that was not very close to the actual root, the method quickly adjusted. Its quadratic convergence rate is evident after the initial few iterations, where the error decreases drastically in each step. The high initial error is due to the first guess being relatively far from the root compared to the interval-based method.

Newton-Raphson Method Output Table

```
Approximate Newton Raphson root found at x = 0.2440223219 on iteration 5 (relative error = 3.77e-04 <= threshold 0.001)

======================================================================
NEWTON-RAPHSON METHOD
======================================================================
Iter    m_k         m_r         f(m_k)      f'(m_k)     ea(%)
----------------------------------------------------------------------
1       1.50000     -1.04195    -11.26317   -4.43092    243.96086
2       -1.04195    0.39216     15.70664    -10.95219   365.69568
3       0.39216     0.24108     -1.75155    -11.59350   62.66830
4       0.24108     0.24402     0.03543     -12.04175   1.20579
5       0.24402     0.24402     0.00001     -12.03421   0.00038
======================================================================
```

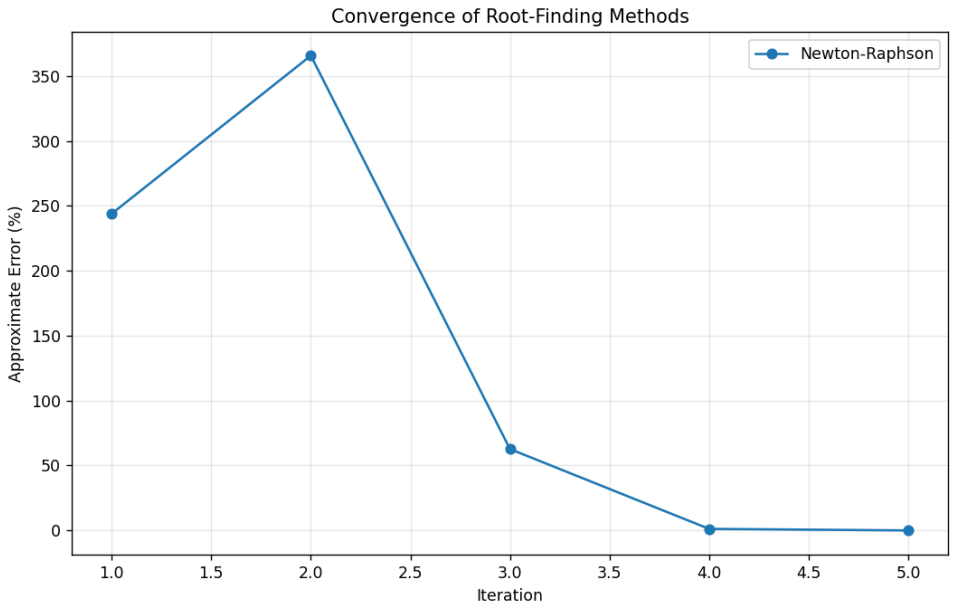

Figure 3: Error vs Iterations (Newton–Raphson Method)

5.4 Secant Method Output

The Secant method applied with initial guesses $h_0 = 1.5$ and $h_1 = 2.0$, and a relative error threshold of 0.001% converged to $h = -4.4744263967$ in 8 iterations (Note: The Secant method converged to a different root compared to the other methods, which is a common occurrence for open methods with different initial guesses or intervals). Its convergence rate is superlinear.

Secant Method Output Table

```
Approximate Secant root found at x = -4.4744263967 on iteration 8 (relative error = 2.40e-04 <= threshold 0.001)

================================================================================================
SECANT METHOD
================================================================================================
Iter   x_i-1        x_i          x_i+1        f(x_i-1)     f(x_i)       f(x_i+1)     ea(%)
------------------------------------------------------------------------------------------------
1      1.50000      2.00000      -4.77520     -11.26317    -12.16060    -8.69786     141.88310
2      2.00000      -4.77520     -21.79342    -12.16060    -8.69786     259857.48459 78.08882
3      -4.77520     -21.79342    -4.77577     -8.69786     259857.48459 -8.71563     356.33363
4      -21.79342    -4.77577     -4.77634     259857.48459 -8.71563     -8.73344     0.01195
5      -4.77577     -4.77634     -4.49644     -8.71563     -8.73344     -0.59009     6.22492
6      -4.77634     -4.49644     -4.47615     -8.73344     -0.59009     -0.04605     0.45312
7      -4.49644     -4.47615     -4.47444     -0.59009     -0.04605     -0.00029     0.03837
8      -4.47615     -4.47444     -4.47443     -0.04605     -0.00029     -0.00000     0.00024
================================================================================================
```

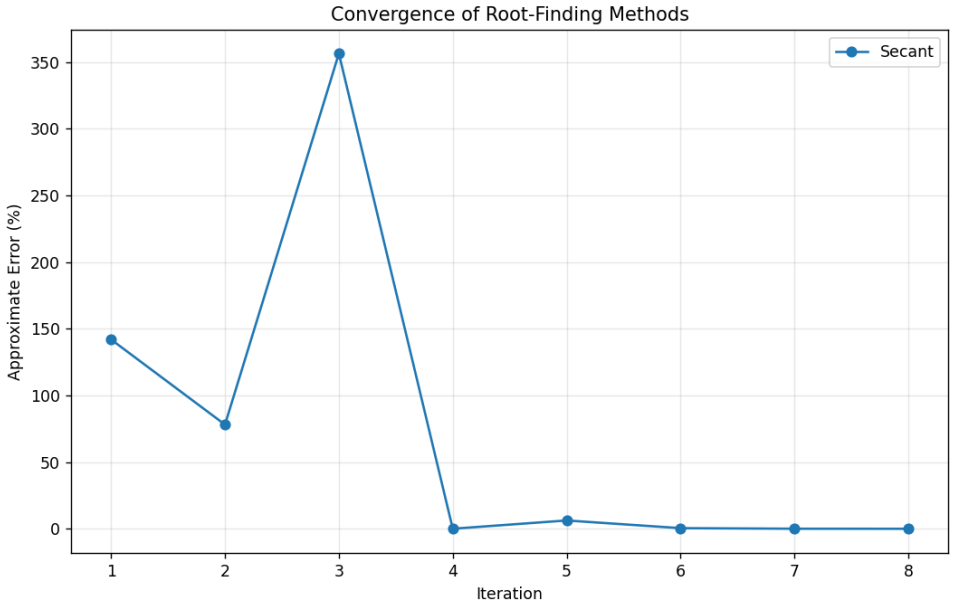

Figure 4: Error vs Iterations (Secant Method)

5.5 Comparative Visualization of Convergence

The following graph depicts the approximate relative error (ϵ_a) versus the number of iterations for all four methods, providing a visual comparison of their convergence rates.

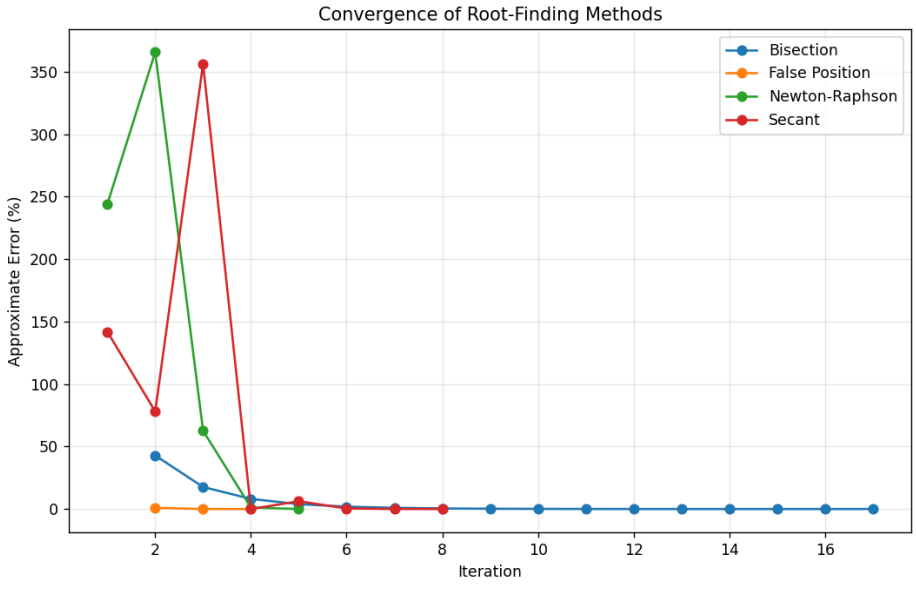

Figure 5: Convergence Comparison of Root Finding Methods

5.5.1 Rationale and Discussion of Convergence

The convergence plot clearly illustrates the efficiency of each method:

- **Bisection Method:** Shows the slowest convergence since its error reduction is linear, meaning the number of correct decimal places increases by a fixed amount with each iteration. We can also see that the error rate at each iteration is only halved,so it takes a high number of (17) iterations to reach the desired tolerance rate of 0.001 This method is guaranteed to converge but is not the most efficient.

- **False Position Method:** This bracketing method Demonstrates fastest convergence needing only four iterations to reach the result. The main reason for this is that it gets very close to the root in the first step. Because of that, in the next few steps, one side stays close to the root, which helps the method reach a small error quickly.

- **Newton-Raphson Method:** Exhibits the second best convergence. After the initial few iterations, its quadratic convergence is evident, meaning the number of correct decimal places roughly doubles with each iteration. This method is highly efficient but requires the derivative of the function and a good initial guess to ensure convergence.

- **Secant Method:** The Secant Method performs moderately, needing 8 iterations to converge. It avoids derivative calculations and works faster than the bisection method, but, like other open methods, it can diverge. In this case, it even finds a different root. Because it's not globally stable, poor starting points or flat slopes can cause large jumps - for example, the root suddenly shifts to -21.79342 in the second iteration, with a huge increase in error.

It's important to note that the performance of open methods (Newton-Raphson, Secant) heavily depends on the initial guess, and they may diverge if the guess is poor. Bracketing methods (Bisection, False Position) are more robust as they always converge if a root is present in the initial interval.

6 Summary

In this experiment, we compared four root-finding methods—Bisection, False Position, Newton-Raphson, and Secant—to solve $f(h) = h^3 - 10h + 5e^{-h/2} - 2 = 0$.

All methods successfully found a root of the function. Among them, the False Position method showed the fastest convergence, reaching the solution in just a few(4) iterations using linear interpolation. Newton-Raphson followed closely due to its quadratic convergence in just 5 iterations, performing efficiently but requiring a more complex derivative calculation. The Secant method showed moderate performance (required 8 iterations), with a noticeable jump in the second iteration that affected its stability but it found different root because of initial guess. The Bisection method was the slowest needing the most iterations(17) but it remained the simplest and most reliable in terms of convergence.

Overall, while the Newton-Raphson method is typically the most efficient for well-behaved functions, the False Position method performed best for this particular case. The experiment highlighted a key trade-off: bracketing methods are robust but slow, while open methods are fast but sensitive, may fail due to poor initial guess or irregular behavior of derivative. This highlights that the efficiency of a numerical method depends on the behavior of the function, and choosing the right method requires understanding the function's nature.