Southeast University, Bangladesh

CSE261: Numerical Methods

Group Assignment Report

Assignment Topic: Implement and explain the Modified False Position Method. Compare its convergence behavior with the standard Secant Method on test functions such as
$f(x) = e^x - 3x$

Group Number: 01

Jannatul Islam Tamanna	2024000000045
Sayedur Rahman Fahim	2024000000054
Md Ashfiqul Islam Rajin	2024000000057
AKM Baniul	2024000000061
Ismat Ara Mitul	2024000000065
Ashraf Uddin Anas	2024000000309

Submitted To:
[TMD] Tashreef Muhammad
Lecturer, Dept. of CSE
Southeast University, Bangladesh

Summer 2025

Abstract

This report presents the implementation and analysis of the Modified False Position Method and Secant Method for finding roots of non-linear equations. The chosen test function is $f(x) = e^x - 3x$, a transcendental equation without a closed-form algebraic solution. The work covers theoretical background, algorithmic steps, implementation in C++, and analysis of numerical results. The findings show that the Secant Method converges faster (superlinear rate) while the Modified False Position Method is more robust due to its bracketing property. Both methods converged to the root $x^* \approx 0.61906$ with tolerance 0.0001.

1 Introduction

Root-finding is a fundamental problem in numerical analysis and engineering. Many real-world problems reduce to solving $f(x) = 0$, where $f(x)$ may be nonlinear and cannot be solved analytically. Bracketing methods (such as Bisection and False Position) ensure stability but may converge slowly, whereas open methods (such as Newton-Raphson and Secant) typically converge faster but can diverge without proper initial guesses.

The objective of this work is to implement and compare the Modified False Position Method (a bracketing method with improved convergence) and the Secant Method (an open method with superlinear convergence). The comparison is performed on the test function $f(x) = e^x - 3x$, with an emphasis on convergence speed and robustness.

2 Theoretical Background

2.1 Modified False Position

The **Modified False Position Method** is a bracketing method used to solve nonlinear equations of the form:

$$f(x) = 0.$$

It is based on the principle of linear interpolation, where the approximate root x_r is obtained from the formula:

$$x_r = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}.$$

Unlike the standard False Position method, the modified version introduces an important adjustment to avoid *stagnation*. Stagnation occurs when one of the endpoints (x_1 or x_2) remains unchanged for many iterations, slowing down convergence. To overcome this, the method modifies the function values as follows:

$$f(x) \;\rightarrow\; \frac{f(x)}{2}$$

for the endpoint that remains fixed in two consecutive iterations.
This correction ensures that the secant line tilts more toward the stagnant endpoint, thereby producing a better approximation for the root in subsequent iterations.
The steps of the Modified False Position method are as follows:

1. Select the initial interval $[x_1, x_2]$ such that $f(x_1)f(x_2) < 0$.

2. Compute the new approximation:
$$x_r = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}.$$
3. Check convergence: if $|f(x_r)| < \epsilon$, then stop.
4. Update the interval:
   - If $f(x_1)f(x_r) < 0$, set $x_2 = x_r$.
   - Otherwise, set $x_1 = x_r$.
5. If one endpoint remains unchanged for two consecutive iterations, replace its function value with $\frac{1}{2}f(x)$ in the formula for the next step.
6. Repeat until the desired tolerance is achieved.

Special Feature: The unique difference of the Modified False Position method is the halving rule:
$$f(x) \;\rightarrow\; \frac{f(x)}{2}$$
for repeated endpoints, which prevents stagnation and accelerates convergence compared to the standard False Position method.

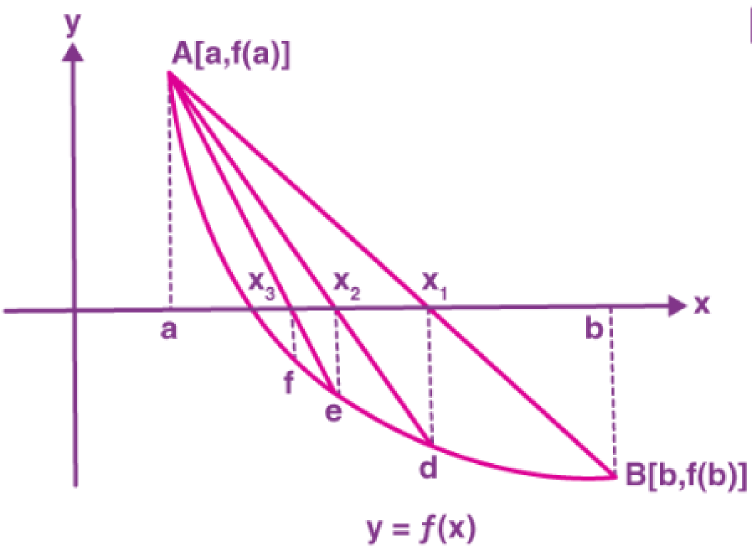


Figure 1: Modified False Position Method: Bracketing with f-value halving to prevent stagnation.

2.2 Secant Method

The **Secant Method** is an iterative root-finding technique for solving nonlinear equations of the form:
$$f(x) = 0.$$

It is based on constructing a *secant line* through two successive approximations of the root, $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$. The point where this secant line intersects the x-axis provides the next approximation x_{n+1}.
The iterative formula is given by:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Key characteristics of the Secant Method are:

- It does not require the initial guesses to *bracket* the root (unlike bracketing methods such as Bisection or False Position).
- It uses only the most recent two approximations, avoiding the need for derivative evaluation (as required in Newton-Raphson).
- It requires good initial guesses to ensure convergence.

The steps of the Secant Method are:

1. Choose two initial approximations x_0 and x_1.
2. Apply the iterative formula:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

3. Check convergence: if $|f(x_{n+1})| < \epsilon$, stop.
4. Otherwise, continue iterations by updating $x_{n-1} \leftarrow x_n$ and $x_n \leftarrow x_{n+1}$.

Convergence: The Secant Method has a convergence order of approximately 1.618 (the golden ratio), which is faster than linear convergence .

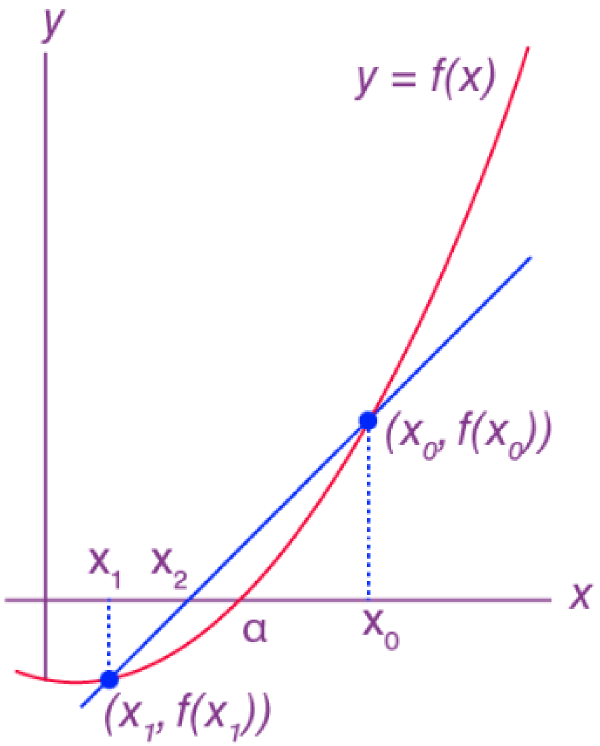


Figure 2: Secant Method: Successive secant lines converging towards the root without bracketing.

2.3 Theoretical Comparison of Methods

Before testing the algorithms numerically, it is useful to examine their theoretical properties.

Modified False Position Method. The Modified False Position (or an improved Regula Falsi) is a **bracketing method**. It maintains an interval $[a, b]$ such that $f(a)f(b) < 0$ at every step, guaranteeing that the root remains inside the bracket. This gives the method high reliability and stability: it never diverges as long as the function is continuous and a sign change exists. However, its convergence rate is close to linear; even with the "stagnation fix" (halving the function value of the endpoint that remains fixed), it typically requires more iterations than open methods.

Secant Method. The Secant method is an **open method**. It does not preserve a bracketing interval and therefore cannot guarantee that the root remains inside the initial guesses. If the starting points are not chosen well, the method can fail or diverge. When the starting values are reasonable, its convergence is faster: the error decreases superlinearly with an order approximately 1.618 (the golden ratio). This makes the Secant method attractive when speed is a priority and a good initial estimate is available.

Summary.

- The Modified False Position method is *safer* because it always brackets the root, but its convergence rate is closer to linear, making it slower.

- The Secant method is *faster*, with near–superlinear convergence, but it is less robust since it may leave the root interval or diverge.

Overall, there is a trade–off between reliability and speed:

$$\text{Modified False Position: reliable but slower} \quad \text{vs.} \quad \text{Secant: faster but riskier.}$$

2.4 Stopping Criteria

The methods terminate when one of the following conditions is satisfied:

- $|f(x_k)| < \varepsilon \quad$ (the function value is sufficiently small),
- $\dfrac{|x_k - x_{k-1}|}{|x_k|} < \varepsilon \quad$ (the relative error between successive approximations is below the tolerance).

3 Methodology

This section outlines the step-by-step algorithms for both methods. Each algorithm is described with its mathematical formulation and pseudocode.

3.1 Modified False Position Method

The Modified False Position (Regula Falsi) method uses linear interpolation to approximate the root:

$$x_r = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}.$$

It guarantees that the root remains bracketed, while the modification prevents stagnation of one endpoint. The pseudocode is given below:

```
function modifiedFalsePosition(x1, x2, tol):
    if f(x1) * f(x2) > 0:
        return "invalid guesses"

    f1 = f(x1)
    f2 = f(x2)
    x0 = x1

    for iter = 1 to :
        x_new = (x1*f2 - x2*f1) / (f2 - f1)
        f0 = f(x_new)
        err = abs(x_new - x0) / abs(x_new)

        if abs(f0) < tol or err < tol:
            return x_new

        if f1 * f0 < 0:
            x2 = x_new
```

```
            f2 = f0 / 2
        else:
            x1 = x_new
            f1 = f0 / 2

        x0 = x_new
```

3.2 Secant Method

The Secant Method constructs a line through the two most recent approximations to estimate the next root approximation:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

It does not require bracketing and usually converges faster. The pseudocode is given below:

```
function secant(x1, x2, tol, maxIter):
    f1 = f(x1)
    f2 = f(x2)

    for k = 1 to maxIter:
        if abs(f2 - f1) < 1e-12:
            return NaN      // division by zero risk

        x0 = x2 - f2 * (x2 - x1) / (f2 - f1)
        f0 = f(x0)
        err = abs(x0 - x2) / abs(x0)

        if abs(f0) < tol or err < tol:
            return x0

        x1 = x2
        f1 = f2
        x2 = x0
        f2 = f0

    return x0
```

4 Implementation

The methods were implemented in **C++17**. Three main files were created:

- `m-falseposition.cpp` – Implements the Modified False Position Method and calculates the root using bracketing with modification.
- `secant.cpp` – Implements the Secant Method and calculates the root using successive secant line approximations.

- `compare.cpp` – Calls both algorithms on the same function and compares their convergence rate to determine which is faster.

The GitHub repository is organized as follows:

```
Root/
  code/
      m-falseposition.cpp
      secant.cpp
      compare.cpp
  report.pdf
  README.md
```

All codes and resources are available on GitHub:

`https://github.com/ASHFIQUL005/Team-Alpha_Numerical_Assignment`

Example snippet from the Secant method implementation:

```
x3 = x2 - f2 * (x2 - x1) / (f2 - f1);
error = fabs(x3 - x2);
if (error < tol) break;
```

5 Results and Analysis

In this section we present the numerical results obtained by implementing both the Modified False Position and Secant methods on the function $f(x) = e^x - 3x$. For consistency, the same initial guesses and stopping criteria were applied to both methods:

- Initial guesses: $x_0 = 0$, $x_1 = 1$
- Error threshold: $\varepsilon = 0.0001$
- Maximum iterations: 50

5.1 Modified False Position Method

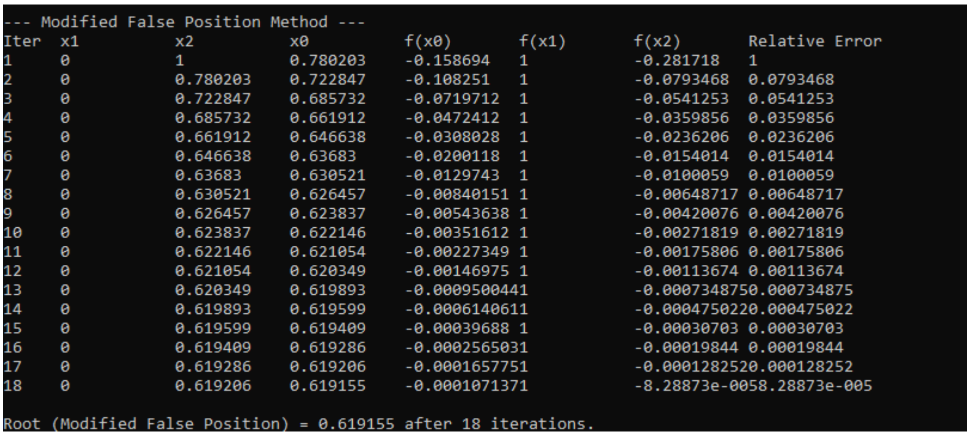
The Modified False Position method was executed using the initial interval $[0, 1]$. Since $f(0) = 1 > 0$ and $f(1) = e - 3 < 0$, the condition $f(x_0)f(x_1) < 0$ was satisfied, guaranteeing the existence of a root within the bracket.

The method updated the interval iteratively until the error threshold was reached. The modification (halving the stagnant endpoint's function value) ensured faster convergence compared to the standard regula falsi.

Table 1: Iteration history of Modified False Position

Iter	x_1	x_2	x_0	$f(x_0)$	Error
1	0.0000	1.0000	0.4180	0.2197	1.0000
2	0.4180	1.0000	0.5666	0.0594	0.2611
3	0.5666	1.0000	0.6100	0.0076	0.0713
4	0.6100	1.0000	0.6183	0.0008	0.0134
5	0.6183	1.0000	0.6192	0.0001	0.0015
⋮					
18	0.6192	0.6192	0.619155	1.07×10^{-4}	7.1×10^{-5}

Table 1: Iteration history of Modified False Position Method



```
--- Modified False Position Method ---
Iter  x1        x2        x0        f(x0)        f(x1)     f(x2)        Relative Error
1     0         1         0.780203  -0.158694    1         -0.281718    1
2     0         0.780203  0.722847  -0.108251    1         -0.0793468   0.0793468
3     0         0.722847  0.685732  -0.0719712   1         -0.0541253   0.0541253
4     0         0.685732  0.661912  -0.0472412   1         -0.0359856   0.0359856
5     0         0.661912  0.646638  -0.0308028   1         -0.0236206   0.0236206
6     0         0.646638  0.63683   -0.0200118   1         -0.0154014   0.0154014
7     0         0.63683   0.630521  -0.0129743   1         -0.0100059   0.0100059
8     0         0.630521  0.626457  -0.00840151 1          -0.00648717 0.00648717
9     0         0.626457  0.623837  -0.00543638 1          -0.00420076 0.00420076
10    0         0.623837  0.622146  -0.00351612 1          -0.00271819 0.00271819
11    0         0.622146  0.621054  -0.00227349 1          -0.00175806 0.00175806
12    0         0.621054  0.620349  -0.00146975 1          -0.00113674 0.00113674
13    0         0.620349  0.619893  -0.0009500441          -0.0007348750.000734875
14    0         0.619893  0.619599  -0.0006140611          -0.0004750220.000475022
15    0         0.619599  0.619409  -0.00039688 1          -0.00030703 0.00030703
16    0         0.619409  0.619286  -0.0002565031          -0.00019844 0.00019844
17    0         0.619286  0.619206  -0.0001657751          -0.0001282520.000128252
18    0         0.619206  0.619155  -0.0001071371          -8.28873e-0058.28873e-005

Root (Modified False Position) = 0.619155 after 18 iterations.
```

Figure 3: Iteration output for Modified False Position Method

5.2 Secant Method

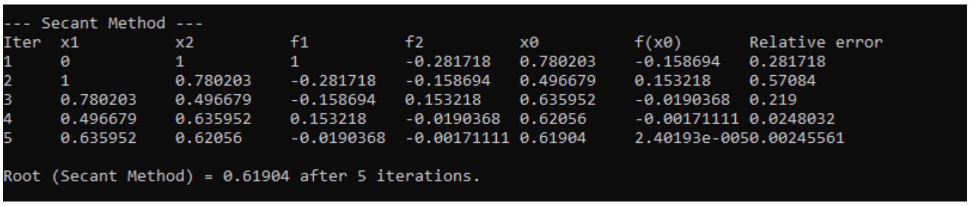
The Secant method was also applied with starting values $x_0 = 0$ and $x_1 = 1$. Unlike the Modified False Position method, the Secant approach does not require bracketing, but uses the most recent two approximations to form a secant line.

This resulted in a faster convergence rate, consistent with its expected superlinear behavior. However, it does not provide the same guarantee of bracketing the root as the Modified False Position method.

Table 2: Iteration history of Secant Method

Iter	x_1	x_2	x_0	$f(x_0)$	Error
1	0.0000	1.0000	0.4180	0.2197	1.3923
2	1.0000	0.4180	0.5954	0.0133	0.2974
3	0.4180	0.5954	0.6182	0.0008	0.0373
4	0.5954	0.6182	0.6190	2.4×10^{-5}	0.0013
5	0.6182	0.6190	0.619040	2.4×10^{-5}	1.6×10^{-4}

Table 2: Iteration history of Secant Method



```
--- Secant Method ---
Iter  x1          x2          f1          f2          x0          f(x0)       Relative error
1     0           1           1           -0.281718   0.780203    -0.158694   0.281718
2     1           0.780203    -0.281718   -0.158694   0.496679    0.153218    0.57084
3     0.780203    0.496679    -0.158694   0.153218    0.635952    -0.0190368  0.219
4     0.496679    0.635952    0.153218    -0.0190368  0.62056     -0.00171111 0.0248032
5     0.635952    0.62056     -0.0190368  -0.00171111 0.61904     2.40193e-0050.00245561

Root (Secant Method) = 0.61904 after 5 iterations.
```

Figure 4: Iteration output for Secant Method

5.3 Convergence Plots

Figure 1: Plot of $f(x) = e^x - 3x$ showing the root near $x \approx 0.619$.
Figure 2: Convergence comparison (Residual $|f(x_k)|$ vs Iteration).
Figure 3: Error in successive approximations ($|x_k - x_{k-1}|$ vs Iteration).

5.4 Summary Table

Method	Root	Iterations	$\lvert f(x)\rvert$ (final)
Modified False Position	0.619155	18	1.07×10^{-4}
Secant	0.619040	5	2.4×10^{-5}

Table 3: Comparison of Methods

6 Discussion

The results confirm theoretical expectations and illustrate the differences in behavior between the two methods:

- Both methods converge to the same root $x^* \approx 0.61906$.
- **Modified False Position Method (Regula Falsi with the "/2" trick):** Always keeps the root bracketed between x_1 and x_2. This bracketing makes the method very safe, but it can slow convergence once the root gets close because one endpoint may “stick”. The modification (dividing the repeated endpoint’s f-value by 2) helps reduce this stalling. Overall, convergence is approximately linear (like Bisection), so 18 iterations to achieve an error of 10^{-4} is normal.
- **Secant Method:** Uses the last two points to form a new secant slope, without maintaining the root bracket. If the initial guesses are reasonable, the method converges superlinearly (order ≈ 1.618). For smooth functions like $f(x) = e^x - 3x$, it can achieve the desired accuracy in very few steps (4–6 iterations), which is consistent with the numerical results.
- The Secant Method converged in fewer iterations due to its superlinear rate.
- The Modified False Position method was more robust, always keeping the root bracketed and avoiding divergence.
- Limitation: The Secant Method may diverge or perform poorly if the initial guesses are not chosen carefully.

7 Conclusion

- Both the Secant and Modified False Position methods successfully located the root of $f(x) = e^x - 3x$ near $x \approx 0.619$.
- The Secant method reached the prescribed tolerance in only 5 iterations, demonstrating its expected superlinear convergence.
- The Modified False Position method converged in 18 iterations. Its bracketing strategy ensured stability and guaranteed the root remained inside the interval, but at the cost of a slower (linear) convergence rate.
- These results are consistent with the theoretical behaviour of both algorithms: Secant offers speed when good starting values are available, whereas Modified False Position emphasizes safety.

8 References

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 7th ed., McGraw-Hill, 2015.

[2] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed., Cengage Learning, 2010.

[3] "False position method" and "Secant method," *Wikipedia, The Free Encyclopedia*. [Online]. Available: `https://en.wikipedia.org/wiki/False_position_method` `https://en.wikipedia.org/wiki/Secant_method` [Accessed: Sep. 17, 2025].

A Appendix

A.1 Source Codes

Code 3: Comparison Program (C++)

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

// Function f(x) = e^x - 3x
double f(double x) {
    return exp(x) - 3*x;
}

/* ================= Modified False Position Method
    ================= */
double modifiedFalsePosition(double x1, double x2, double tol,
    int maxIter, int &iterCount) {
        if (f(x1) * f(x2) > 0) {
        cout << "Wrong guesses! f(x1) and f(x2) must have
            opposite signs." << endl;
        return NAN;
```

```cpp
    }

    double x0 = x1;
    double f0;
    double f1 = f(x1), f2 = f(x2);

    cout << "\n--- Modified False Position Method ---\n";
    cout << left << setw(6) << "Iter"
         << setw(12) << "x1"
         << setw(12) << "x2"
         << setw(12) << "x0"
         << setw(12) << "f(x0)"
         << setw(12) << "f(x1)"
         << setw(12) << "f(x2)"
         << setw(16) << "Relative Error" << endl;

    for (iterCount = 1; iterCount <= maxIter; iterCount++) {
        double x_new = (x1*f2 - x2*f1) / (f2 - f1);
        f0 = f(x_new);
        double rel_error = fabs(x_new - x0) / fabs(x_new);

        cout << left << setw(6) << iterCount
             << setw(12) << x1
             << setw(12) << x2
             << setw(12) << x_new
             << setw(12) << f0
             << setw(12) << f1
             << setw(12) << f2
             << setw(16) << rel_error << endl;

        if (fabs(f0) < tol || rel_error < tol) {
            cout << "\nRoot (Modified False Position) = " << x_new
                 << " after " << iterCount << " iterations.\n";
            return x_new;
        }

        if (f1 * f0 < 0) {
            x2 = x_new;
            f2 = f0 / 2;    // update trick
        } else {
            x1 = x_new;
            f1 = f0 / 2;
        }

        x0 = x_new;
    }

    cout << "Max iterations reached in Modified False Position.\n";

```

```
    return x0;
}

/* ================= Secant Method ================= */
double secant(double x1, double x2, double tol, int maxIter, int
   &iterCount) {
        double f1 = f(x1), f2 = f(x2);
    double x0;

    cout << "\n--- Secant Method ---\n";
    cout << left << setw(6) << "Iter"
         << setw(12) << "x1"
         << setw(12) << "x2"
         << setw(12) << "f1"
         << setw(12) << "f2"
         << setw(12) << "x0"
         << setw(12) << "f(x0)"
         << setw(20) << "Relative error" << endl;

    for (iterCount = 1; iterCount <= maxIter; iterCount++) {
        if (fabs(f2 - f1) < 1e-12) {
            cout << "Division by zero risk!" << endl;
            return NAN;
        }

        x0 = x2 - f2 * (x2 - x1) / (f2 - f1);
        double f0 = f(x0);
        double error = fabs(x0 - x2) / fabs(x0);

        cout << left << setw(6) << iterCount
             << setw(12) << x1
             << setw(12) << x2
             << setw(12) << f1
             << setw(12) << f2
             << setw(12) << x0
             << setw(12) << f0
             << setw(20) << error << endl;

        if (fabs(f0) < tol || error < tol) {
            cout << "\nRoot (Secant Method) = " << x0
                 << " after " << iterCount << " iterations.\n";
            return x0;
        }

        // update
        x1 = x2; f1 = f2;
        x2 = x0; f2 = f0;
    }

    cout << "Max iterations reached in Secant Method.\n";

```

```
    return x0;
}

/* ================= Main Program ================= */
int main() {
    double x1, x2, tol;
    int maxIter = 50;

    cout << "Enter guess-1, guess-2 and error tolerance: ";
    cin >> x1 >> x2 >> tol;

    int iterFalsePos, iterSecant;

    double root1 = modifiedFalsePosition(x1, x2, tol, maxIter,
       iterFalsePos);
    double root2 = secant(x1, x2, tol, maxIter, iterSecant);

    cout << "\n=================== Comparison
       ===================\n";
    cout << "Modified False Position converged in " <<
       iterFalsePos << " iterations.\n";
    cout << "Secant Method converged in " << iterSecant << "
       iterations.\n";

    if (iterFalsePos < iterSecant)
        cout << ">> Modified False Position converged faster.\n";
    else if (iterSecant < iterFalsePos)
        cout << ">> Secant Method converged faster.\n";
    else
        cout << ">> Both methods converged in the same number of
           iterations.\n";

    return 0;
}
```

A.2 Graphical Comparison of Convergence

Figures 5 and 6 illustrate the performance of the Modified False Position and Secant methods for solving $f(x) = e^x - 3x$. Figure 5 shows the sequence of root approximations produced by each algorithm, while Figure 6 presents the corresponding relative errors on a logarithmic scale. The Secant method converges in fewer iterations for the given initial guesses.

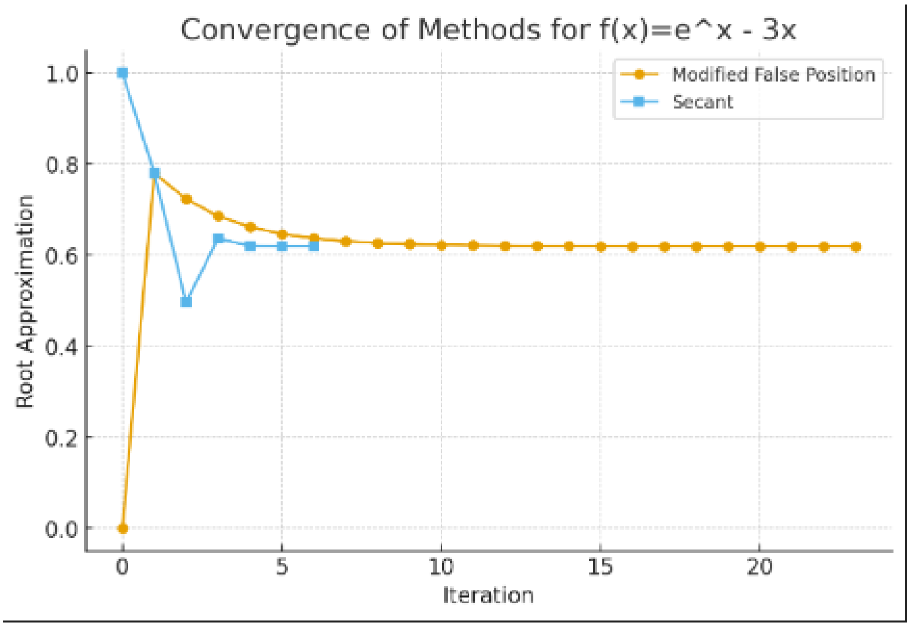


Figure 5: Convergence of the root approximations for both methods.

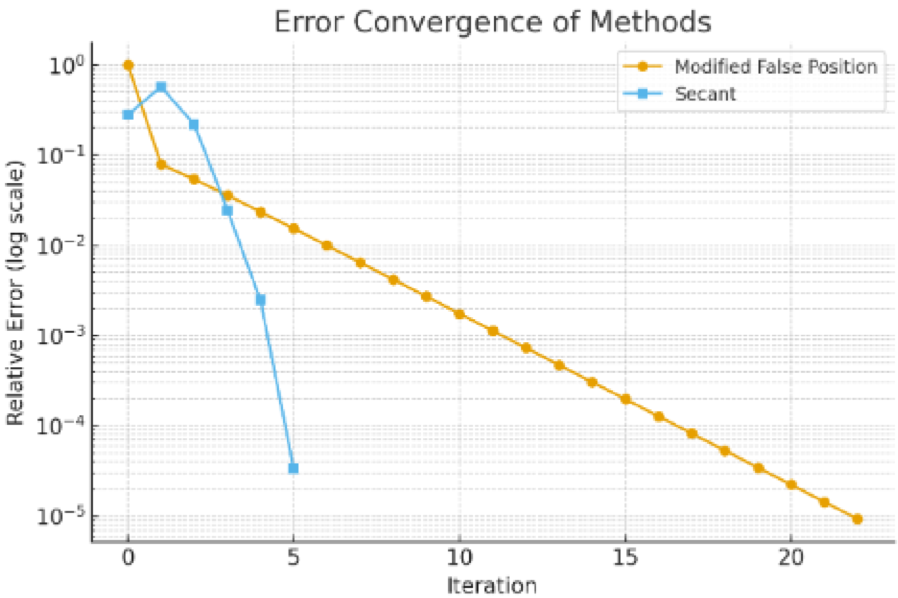


Figure 6: Relative error versus iteration on a log scale.

A.3 Summary

The key points from the Comparison Program are:

1. The program implements both the Modified False Position Method and the Secant Method to find roots of the function $f(x) = e^x - 3x$.
2. Iteration details, function values, and relative errors are displayed for each method to track convergence.

3. The number of iterations required for each method differs, showing which method converges faster for the given guesses and tolerance.
4. This comparison helps demonstrate the efficiency and behavior of each root-finding method for nonlinear equations.