جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY
كلية الهندسة

Computer & Systems Engineering Department

Project - Phase 2

Roots of Equations

Contributors:

	Name	ID
1	Omar Khaled Hussien EL-sayed	22010962
2	Yaseen Asaad Ahmed Farid	22011349
3	Abdulrahman Khaled Nour El-din	22010877
4	Ziad Islam Hosny	22010778
5	Mahmoud Magdy Mahmoud	22011195
6	Mohamed Amr Mohamed	22011153

1. Introduction

The aim of this project is to compare and analyze the behavior of the different numerical methods used for calculating the roots of equations:

1. Bisection
2. False-Position
3. Fixed point
4. Original Newton-Raphson
5. Modified Newton-Raphson
6. Secant Method.

---

2. Pseudo-codes

✓ Bisection Method:

```
Class BisectionMethod:
    Method __init__(equation, x_min, x_max, eps=1e-5,
max_iter=50, show_steps=False, precision=6):
        Initialize symbols, functions, and parameters

    Method format_significant_figures(num):
        Return formatted number with specified significant figures

    Method plot_function():
        Plot the function within the given range

    Method find_root(a, b):
        If f(a) * f(b) > 0:
            Print "Bisection method fails"
            Return None

        Set X_l to a, X_u to b, X_r_old to None

        For n from 1 to max_iter:
            Compute X_r as midpoint of X_l and X_u
            Compute f_m_n as f(X_r)

            If show_steps:
                Print iteration details

            If f_m_n == 0:
                Print "Root found"
                Return X_r

            If f(X_l) * f_m_n > 0:
                Set X_l to X_r
            Else:
                Set X_u to X_r

            If X_r_old is not None:
                Compute relative_error
```

```
If show_steps:
                Print relative error
            If relative_error < eps:
                Break

            Set X_r_old to X_r

        Return X_r

    Method solve():
        Set a to x_min, b to x_max
        Record start time
        Call find_root(a, b) and store result in root

        If root is not None:
            Print root, number of iterations, relative error, and
execution time

Main:
    Define equation
    Get user inputs for x_min, x_max, eps, max_iter, precision,
show_steps
    Create BisectionMethod instance with user inputs
    Call solve() method on the instance
```

✓ False position method:

```
Class FalsePosition:
  Method __init__(equation, x_min, x_max, eps=1e-5, max_iter=50, show_steps=False, precision=6):
    Initialize symbols, functions, and parameters

  Method format_significant_figures(num):
    Return formatted number with specified significant figures

  Method plot_function():
    Plot the function within the given range

  Method find_root(a, b):
    If f(a) * f(b) > 0:
      Print "False Position method fails"
      Return None

    Set X_l to a, X_u to b, X_r_old to None

    For n from 1 to max_iter:
      Compute X_r using False Position formula
      Compute f_m_n as f(X_r)

      If show_steps:
        Print iteration details

      If f_m_n == 0:
        Print "Root found"
        Return X_r

      If f(X_l) * f_m_n > 0:
        Set X_l to X_r
      Else:
        Set X_u to X_r

      If X_r_old is not None:
        Compute relative_error
        If show_steps:
          Print relative error
        If relative_error < eps:
          Break

      Set X_r_old to X_r

    Return X_r
```

```
Method solve():
    Set a to x_min, b to x_max
    Record start time
    Call find_root(a, b) and store result in root

    If root is not None:
      Print root, number of iterations, relative error, and execution time

Main:
  Define equation
  Get user inputs for x_min, x_max, eps, max_iter, precision, show_steps
  Create FalsePosition instance with user inputs
  Call solve() method on the instance
```

✓ Fixed Point method:

```
Class FixedPointMethod:
    Method __init__(equation, x_min, x_max, eps=1e-5, max_iter=50, show_steps=False, precision=6):
        Initialize symbols, functions, and parameters

    Method format_significant_figures(num):
        Return formatted number with specified significant figures

    Method plot_function():
        Plot the function g(x) and y = x within the given range

    Method find_root(x0):
        Try:
            Set x_old to x0
            For n from 1 to max_iter:
                Compute x_new as g(x_old)

                If g(x_old) == x_old:
                    Print "Root found"
                    Append "Root found" to answer string
                    Break

                If show_steps:
                    Print iteration details

                If abs(x_new) < eps:
                    Print "Root close to zero"
                    Append "Root close to zero" to answer string
                    Break

                Compute relative_error
                If show_steps:
                    Print relative error

                If relative_error < eps:
                    Break

                Set x_old to x_new

            Return x_new
        Except Exception as e:
            Print error message
            Append error message to answer string
            Return None
```

```
Method solve():
        Set x0 to x_min
        Record start time
        Call find_root(x0) and store result in root

        If root is not None:
            Print root, number of iterations, relative error, and execution time

Main:
    Define equation
    Get user inputs for x_min, x_max, eps, max_iter, precision, show_steps
    Create FixedPointMethod instance with user inputs
    Call solve() method on the instance
```

✓ Original Newton Raphson method :

```
Class NewtonRaphsonMethod:
   Method __init__(equation, x_min, x_max, eps=1e-5,
max_iter=50, show_steps=False, precision=6):
      Initialize symbols, functions, and parameters

   Method format_significant_figures(num):
      Return formatted number with specified significant
figures

   Method plot_function():
      Plot the function f(x) within the given range

   Method find_root(x0):
      Try:
         Set x_old to x0
         For n from 1 to max_iter:
            Compute f_value, f_prime_value,
f_double_prime_value

            If abs(f_value) == 0:
               Print "Root found"
               Append "Root found" to answer string
               Break

            If f_prime_value == 0:
               Print "Derivative is zero. No roots found."
               Append "Derivative is zero. No roots found." to
answer string
               Return None

            If f_double_prime_value == 0:
               Print "Second derivative is zero. Inflection point."
               Append "Second derivative is zero. Inflection
point." to answer string
               Return None

            Compute x_new using Newton-Raphson formula

            If show_steps:
               Print iteration details

            If abs(x_new) < eps:
               Print "Root close to zero"
               Append "Root close to zero" to answer string
               Break

            If abs(x_new) > 1000:
               Print "Method will diverge"
               Append "Method will diverge" to answer string
               Return None

            Compute relative_error
            If show_steps:
               Print relative error
```

```
If relative_error < eps:
               Break

            Set x_old to x_new

         Return x_new
      Except Exception as e:
         Print error message
         Append error message to answer string
         Return None

   Method solve():
      Set x0 to x_min
      Record start time
      Call find_root(x0) and store result in root

      If root is not None:
         Print root, number of iterations, relative error, and
execution time

Main:
   Define equation
   Get user inputs for x_min, x_max, eps, max_iter,
precision, show_steps
   Create NewtonRaphsonMethod instance with user inputs
   Call solve() method on the instance
```

✓ Modified Newton Raphson method :

```
Class ModNewtonRaphsonMethod:
    Method __init__(equation, x_min, x_max, eps=1e-5, max_iter=50, show_steps=False, precision=6):
        Initialize symbols, functions, and parameters

    Method format_significant_figures(num):
        Return formatted number with specified significant figures

    Method plot_function():
        Plot the function f(x) within the given range

    Method find_root(x0):
        Try:
            Set x_old to x0
            For n from 1 to max_iter:
                Compute f_value, f_prime_value, f_double_prime_value

                If abs(f_value) == 0:
                    Print "Root found"
                    Append "Root found" to answer string
                    Break

                If f_prime_value^2 - (f_value * f_double_prime_value) == 0:
                    Print "Division by zero. No roots found."
                    Append "Division by zero. No roots found." to answer string
                    Return None

                Compute x_new using Modified Newton-Raphson formula

                If show_steps:
                    Print iteration details

                If abs(x_new) < eps:
                    Print "Root close to zero"
                    Append "Root close to zero" to answer string
                    Break

                If abs(x_new) > 1000:
                    Print "Method will diverge"
                    Append "Method will diverge" to answer string
                    Return None

                Compute relative_error
                If show_steps:
                    Print relative error

                If relative_error < eps:
                    Break
```

```
Set x_old to x_new

            Return x_new
        Except Exception as e:
            Print error message
            Append error message to answer string
            Return None

    Method solve():
        Set x0 to x_min
        Record start time
        Call find_root(x0) and store result in root

        If root is not None:
            Print root, number of iterations, relative error, and execution time

Main:
    Define equation
    Get user inputs for x_min, x_max, eps, max_iter, precision, show_steps
    Create ModNewtonRaphsonMethod instance with user inputs
    Call solve() method on the instance
```

✓ Secant method :

```
Class SecantMethod:
    Method __init__(equation, x_min, x_max, eps=1e-5,
max_iter=50, show_steps=False, precision=6):
        Initialize symbols, functions, and parameters

    Method format_significant_figures(num):
        Return formatted number with specified significant
figures

    Method plot_function():
        Plot the function f(x) within the given range

    Method find_root(x0, x1):
        For n from 1 to max_iter:
            Compute f_x0 and f_x1

            If f_x0 - f_x1 == 0:
                Print "Division by zero. No roots found."
                Append "Division by zero. No roots found." to
answer string
                Return None

            Compute x_new using Secant formula

            If abs(f(x_new)) == 0:
                Print "Root found"
                Append "Root found" to answer string
                Break

            If show_steps:
                Print iteration details

            If abs(x_new) < eps:
                Print "Root close to zero"
                Append "Root close to zero" to answer string
                Break

            Compute relative_error
            If show_steps:
                Print relative error

            If relative_error < eps:
                Break

            If abs(x_new) > 1000:
                Print "Method will diverge"
                Append "Method will diverge" to answer string
                Return None

            If show_steps and n != max_iter:
                Print separator
                Append separator to answer string

            Set x0 to x1 and x1 to x_new
```

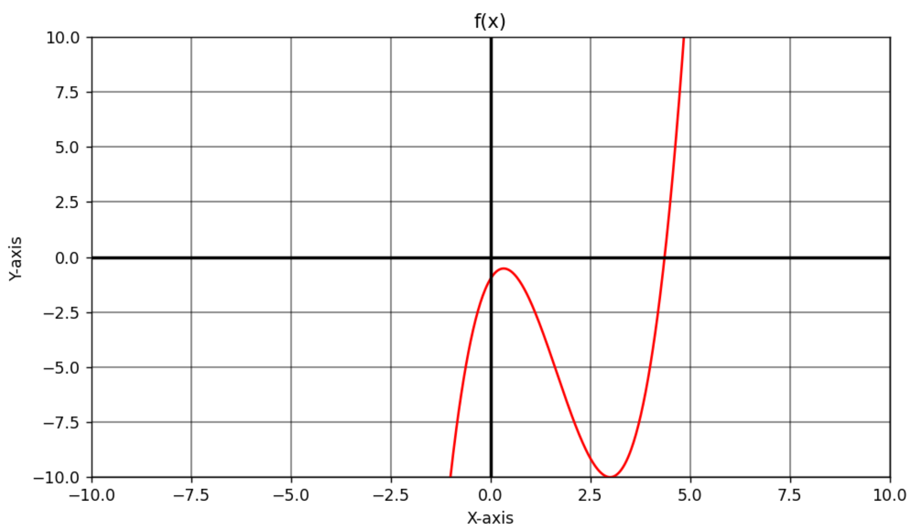
```
        Return x_new

    Method solve():
        Set x0 to x_min and x1 to x_max
        Record start time
        Call find_root(x0, x1) and store result in root

        If root is not None:
            Print root, number of iterations, relative
error, and execution time

Main:
    Define equation
    Get user inputs for x_min, x_max, eps, max_iter,
precision, show_steps
    Create SecantMethod instance with user inputs
    Call solve() method on the instance
```

3. Sample Runs

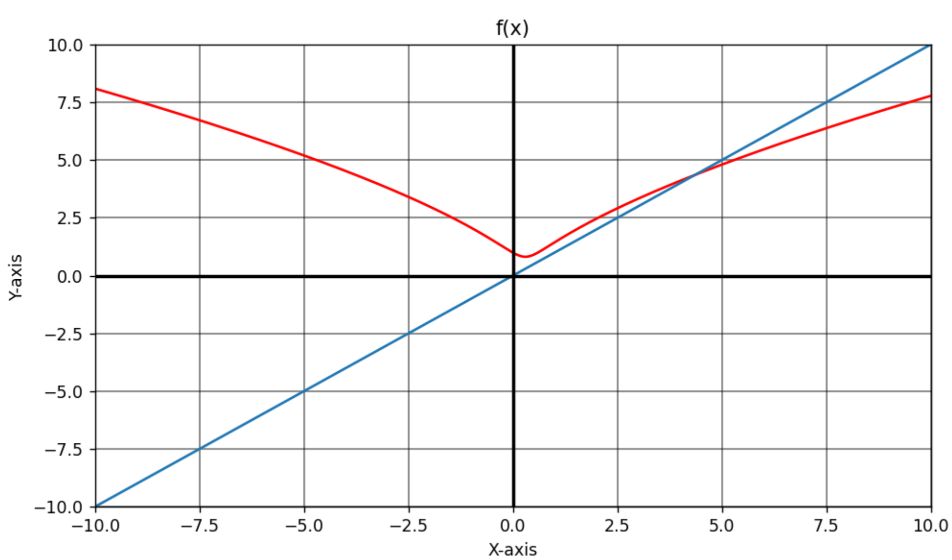
TC_1:

Plotting:

1. F(x) = **x^3-5x^2+3x-1**

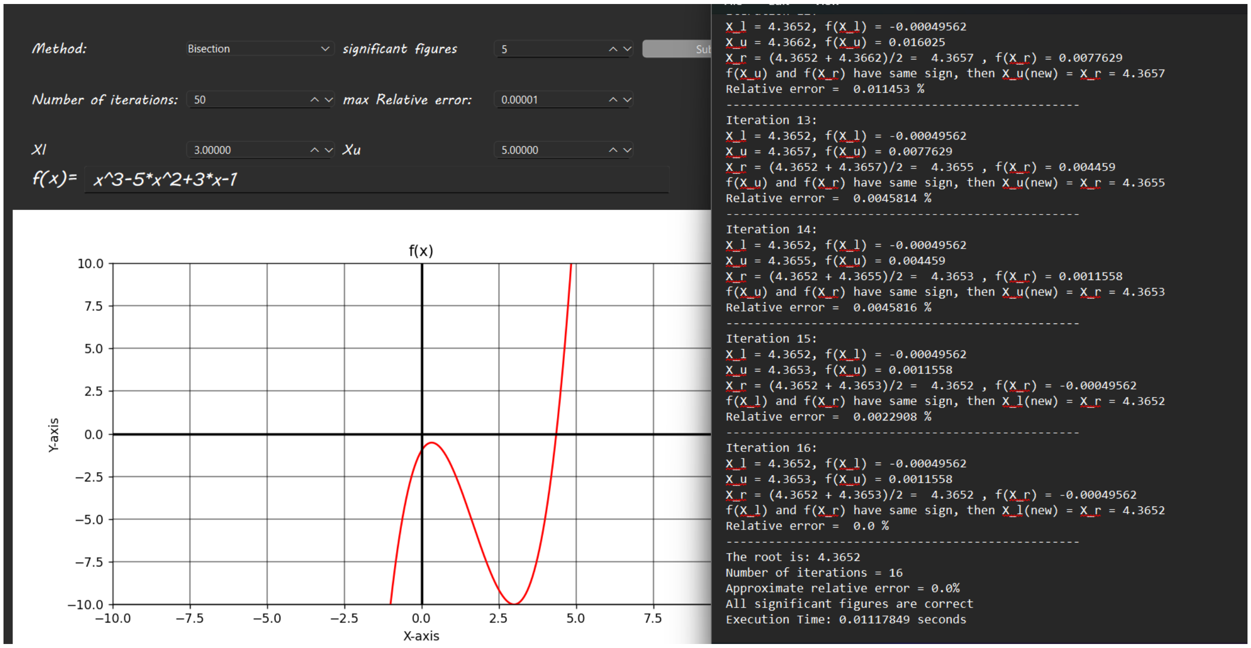


2. Will assume **g(x) = (5x^(2)-3x+1)^(1/3) :**

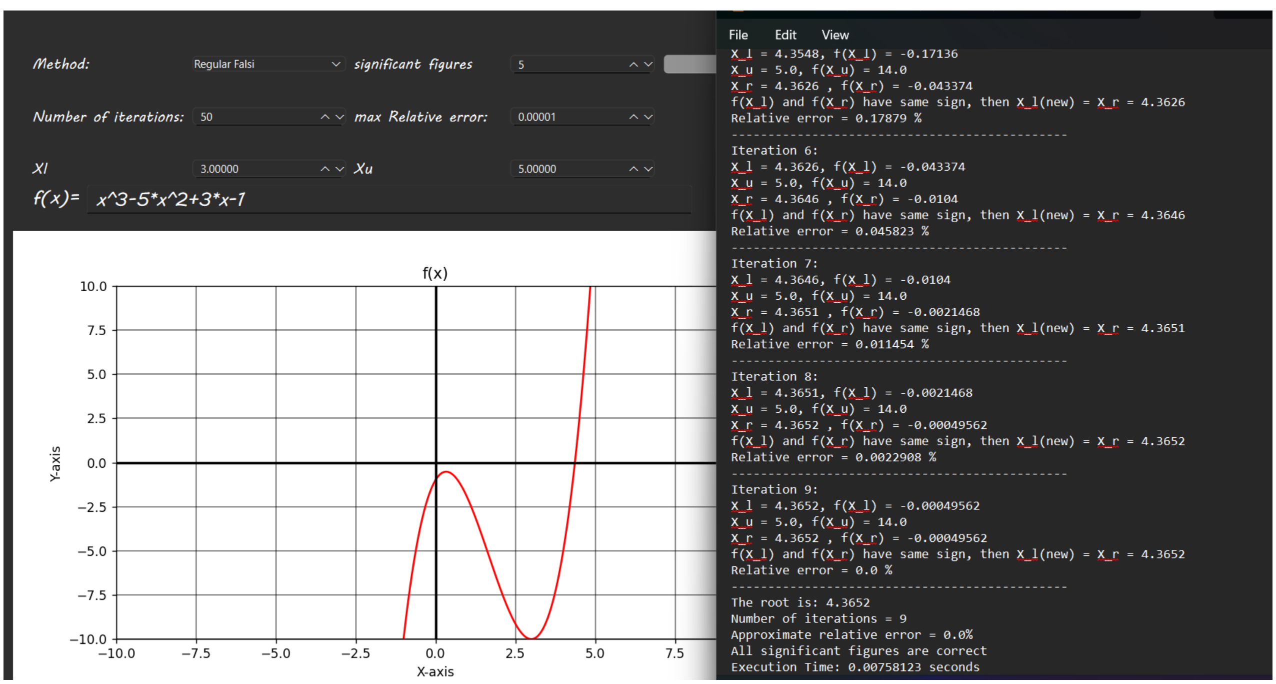


Methods:

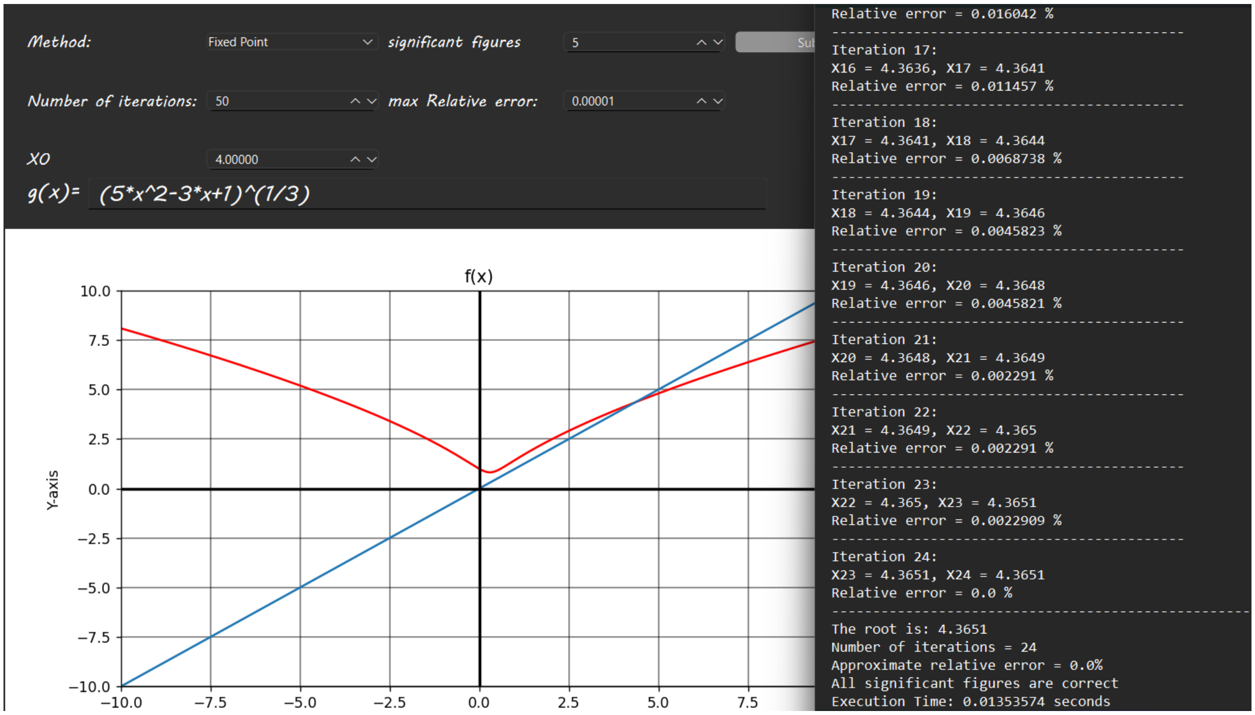
- Bisection



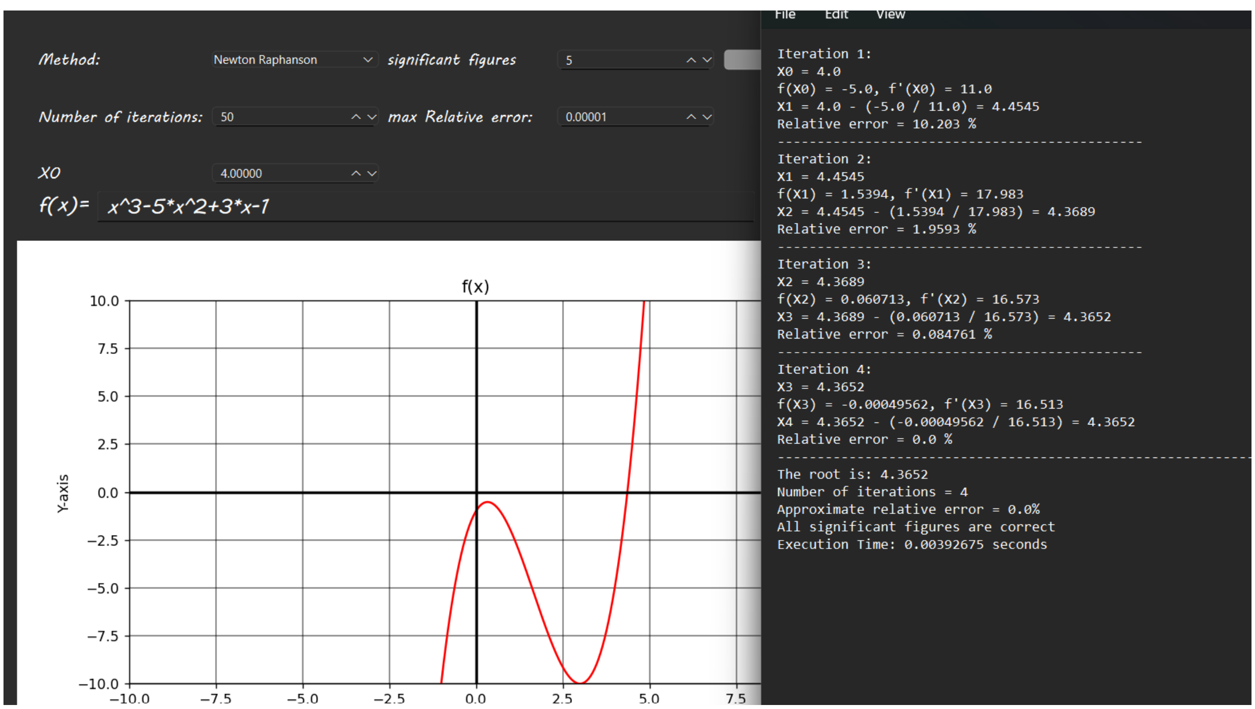
- False Position



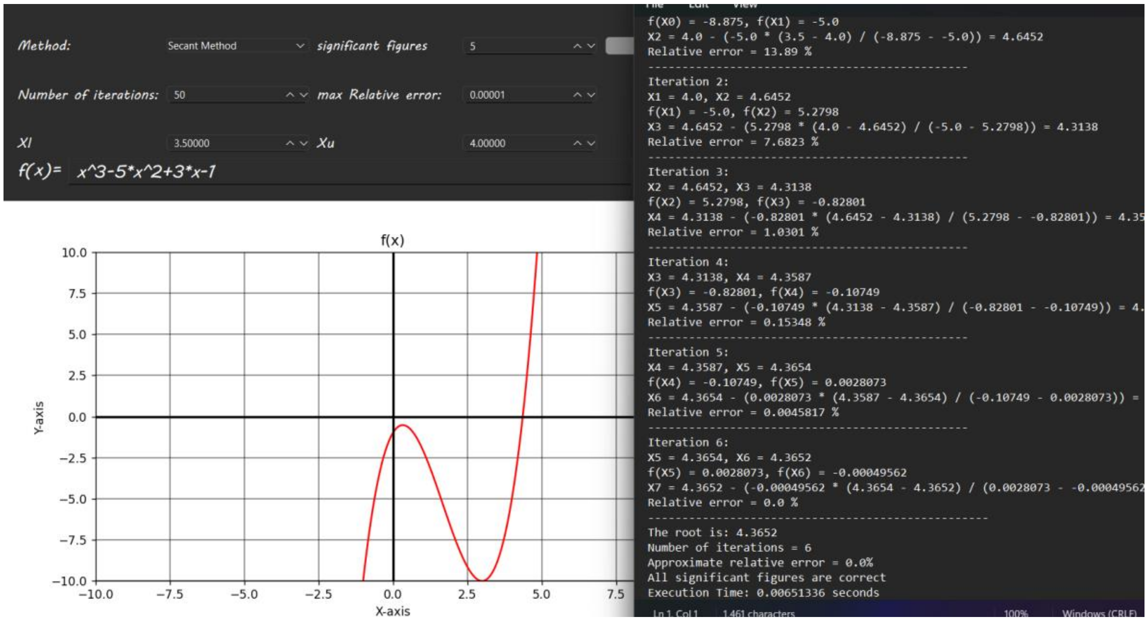
- Fixed Point



- Original Newton



- Secant



Comparable Table:

Method	Number of iterations	Run time	App. Root	initial guess	Relative Error
Bisection	16	0.01117849 sec	4.3652	Xl = 3 Xu = 5	0 %
False Position	9	0.00758123 sec	4.3652	Xl = 3 Xu = 5	0 %
Fixed Point	24	0.01353574 sec	4.3651	Xo = 4	0 %
Original Newton	4	0.00392675 sec	4.3652	Xo = 4	0 %
Secant	6	0.00651336 sec	4.3652	Xo = 3.5 X1 = 4	0 %

Comments:

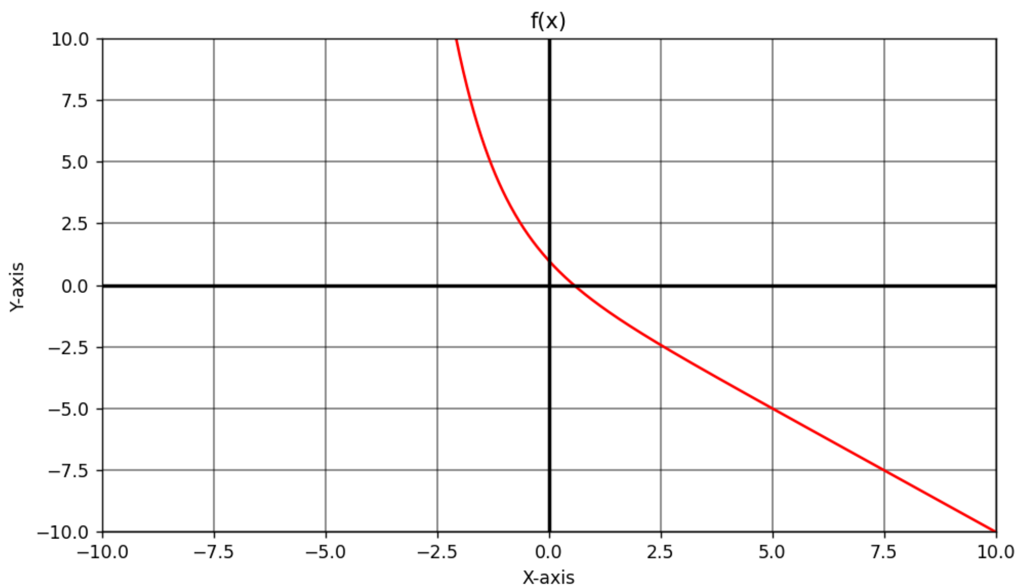
- **Efficiency**: The Original Newton method was the most efficient, followed by the Secant and False Position methods.
- **Accuracy**: All methods achieved high accuracy with zero relative error.
- **Iterations**: The Fixed Point method required the most iterations, indicating lower efficiency for this specific problem.

Overall, the Original Newton method stands out as the best performer in terms of both speed and accuracy, while the Fixed Point method, despite being accurate, was less efficient.

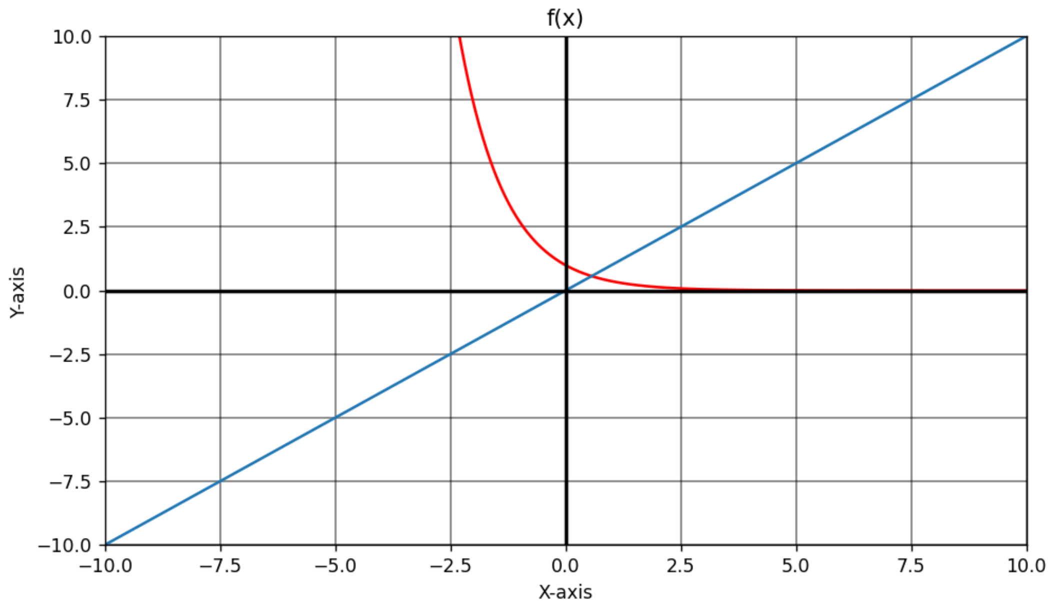
TC_2:

Plotting:

1. F(x) = **e^(-x)-x**

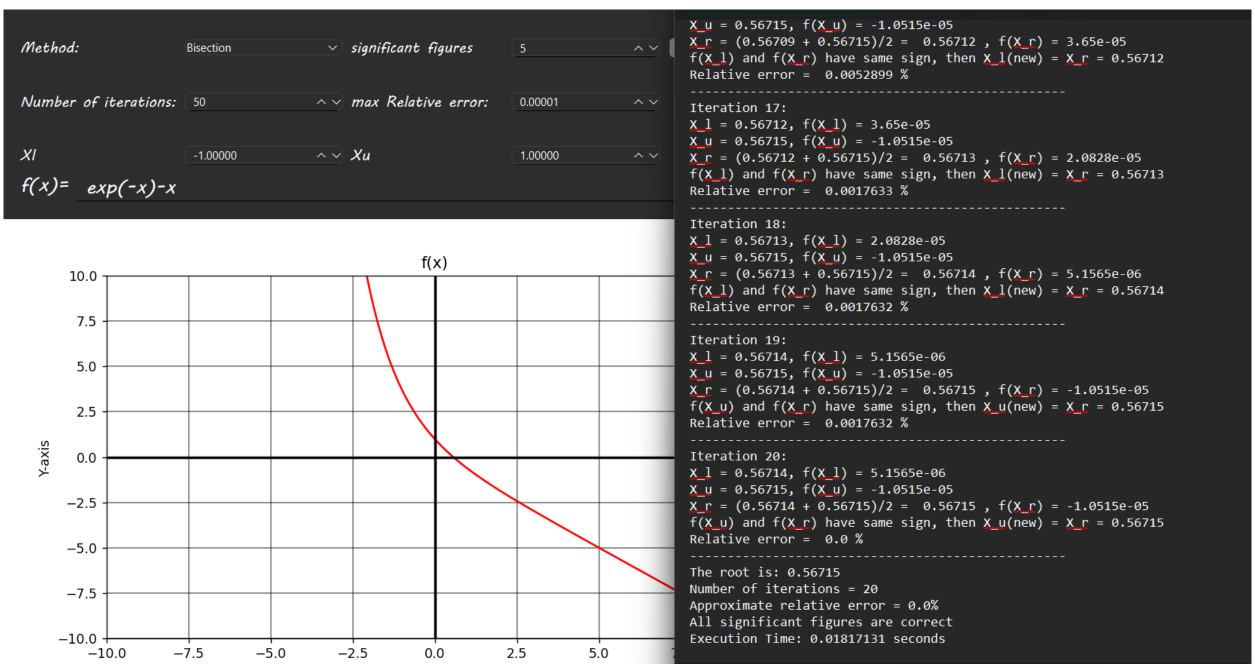


2. Will assume that **g(x) = e^(-x) :**

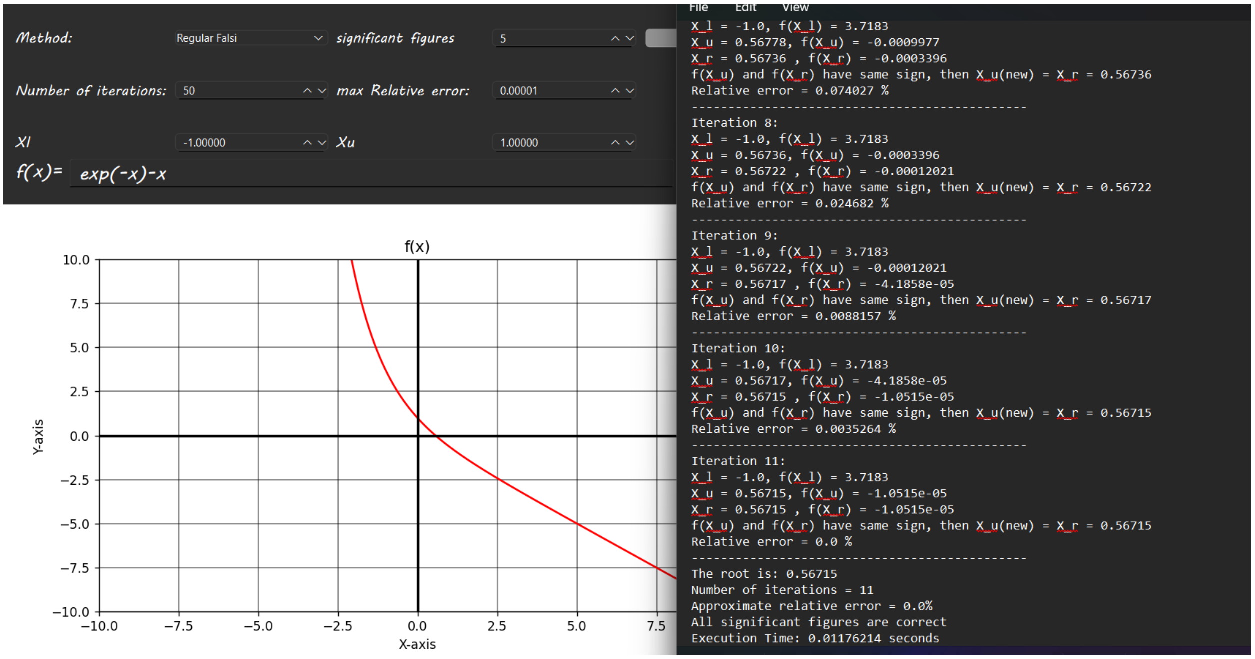


Methods:

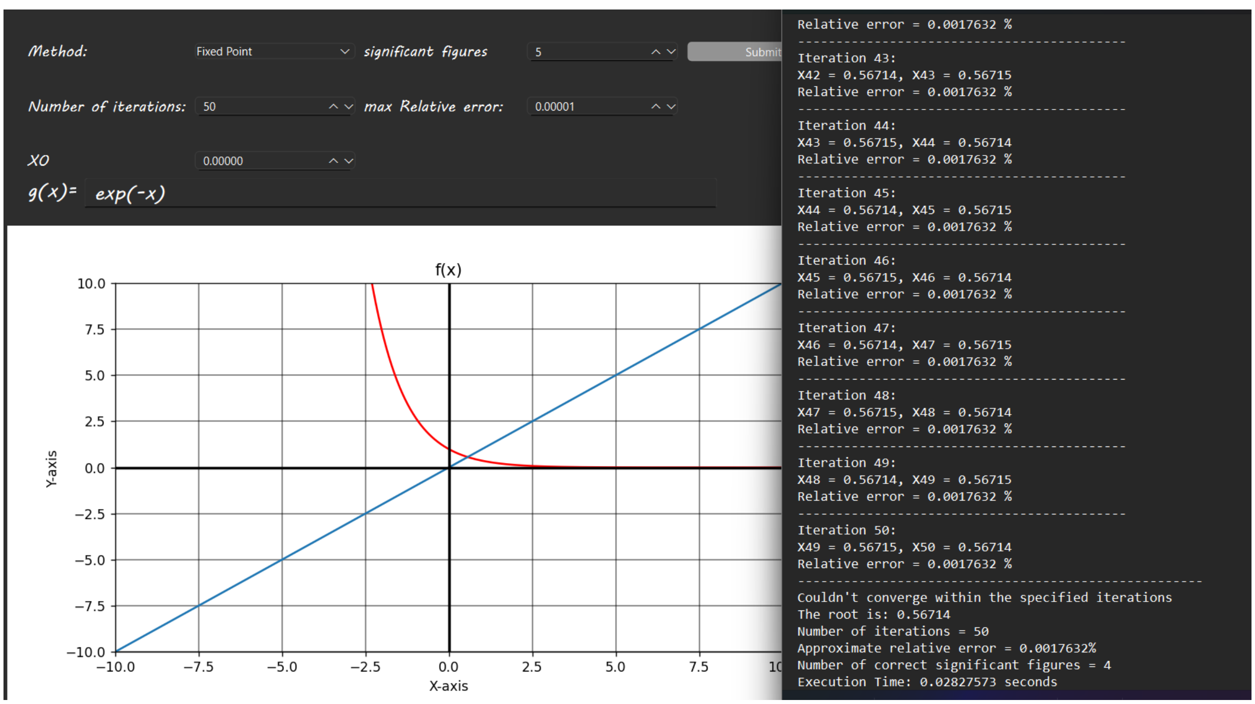
- Bisection



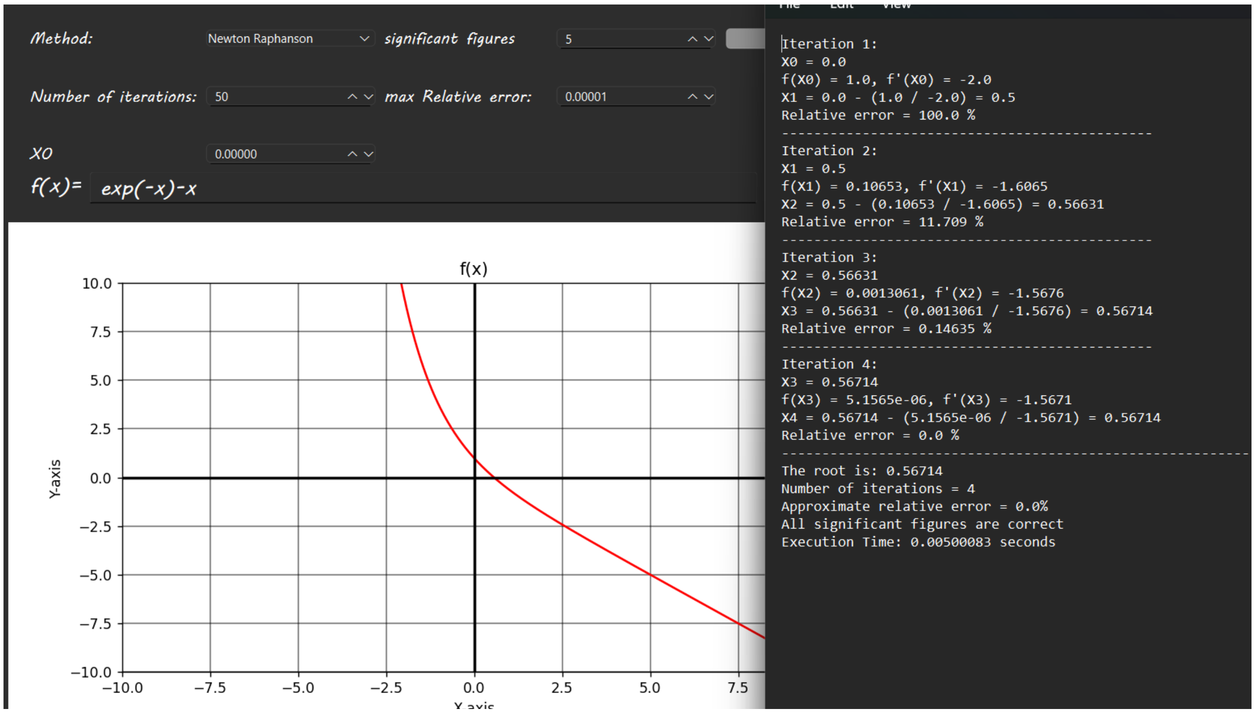
- False Position



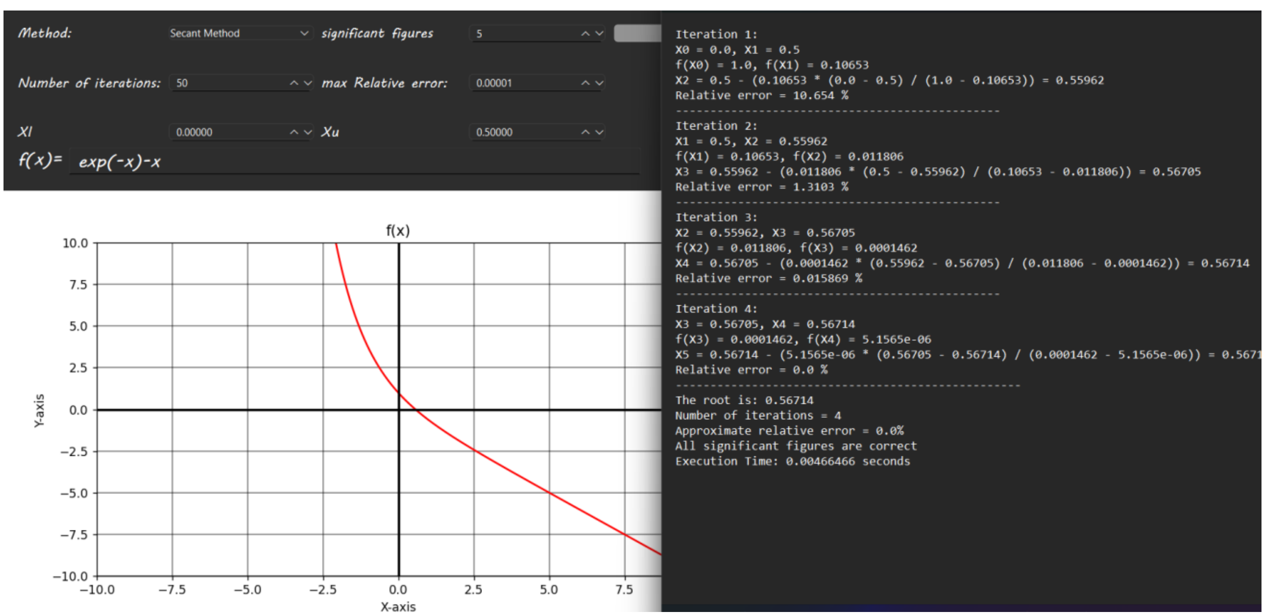
- Fixed Point



- Org Newton Raphson



- Secant



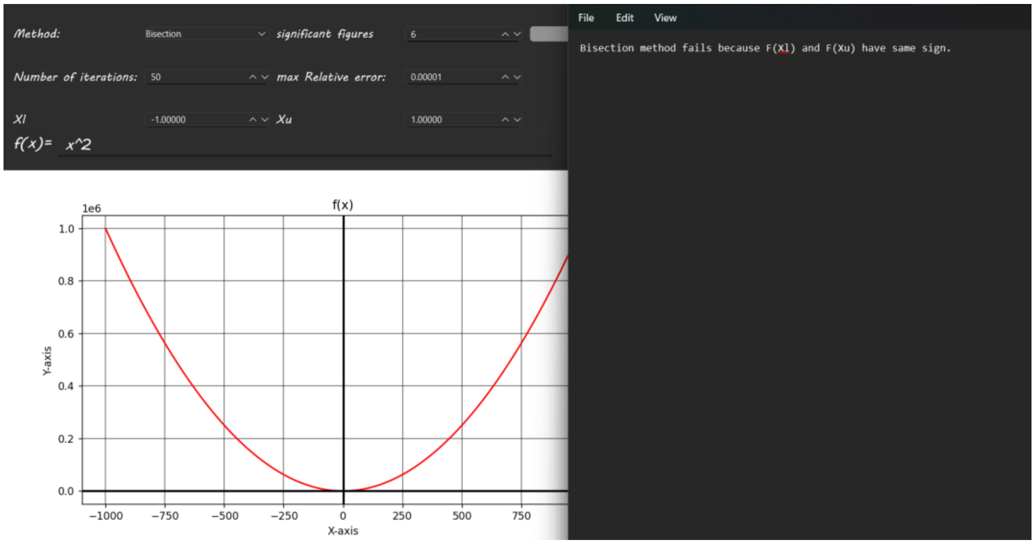
Comparable Table:

Method	Number of iterations	Run time	App. Root	initial guess	Relative Error
Bisection	20	0.01817131 sec	0.56715	Xl = -1 Xu = 1	0 %
False Position	11	0.01176214 sec	0.56715	Xl = -1 Xu = 1	0 %
Fixed Point	50	0.02827573 sec	0.56714	Xo = 0	0.0017632 %
Original Newton	4	0.005 sec	0.56714	Xo = 0	0 %
Secant	4	0.00466466 sec	0.56714	Xo = 0 X1 = 0.5	0 %

Comments:

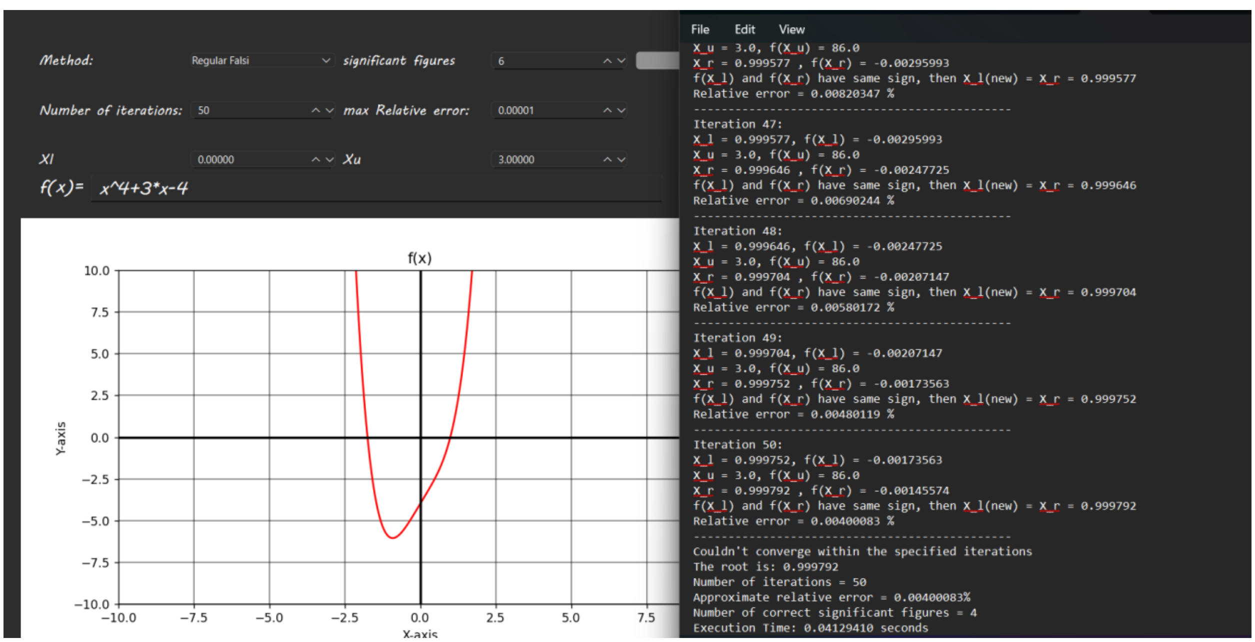
- **Efficiency**: The Original Newton method and Secant were the most efficient.
- **Accuracy**: All methods achieved high accuracy **except** the Fixed Point method having a very small relative error and doesn't reach the needed epsilon so the iterations stopped due to reaching max number iterations = 50
- **Iterations**: The Fixed Point method required the most iterations, indicating lower efficiency for this specific problem.

TC_3:



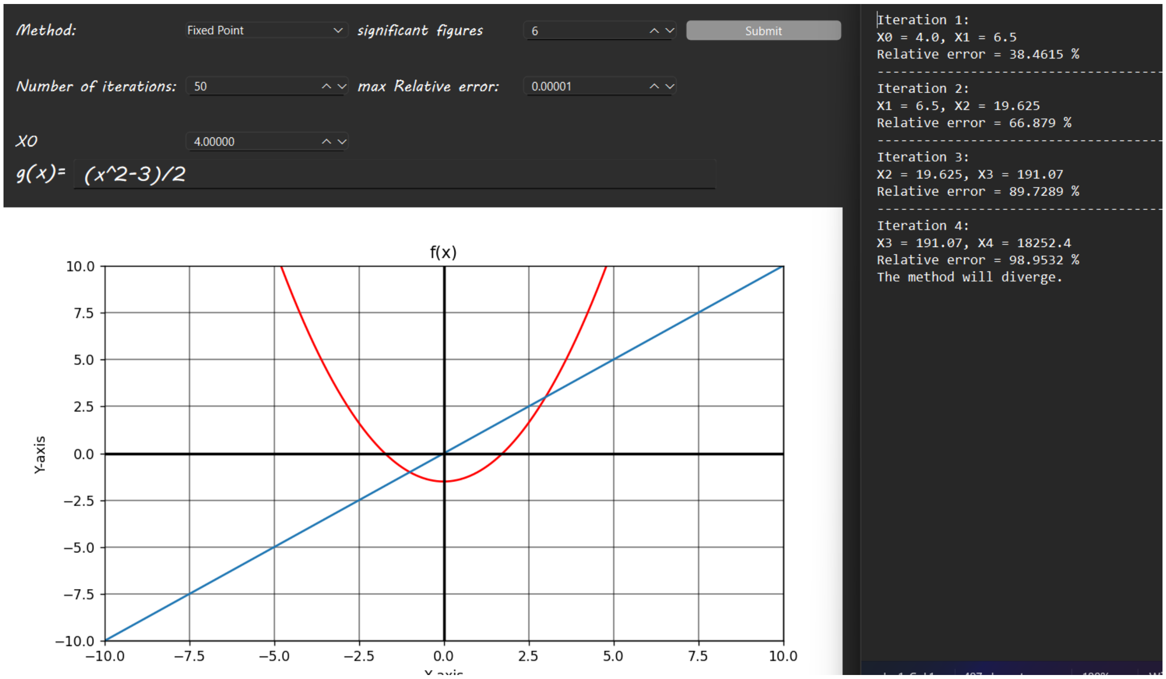
- As shown, in the graph F(-1) and F(1) are both positive and the only condition to apply Bisection method is that
  F(-1) * F(1) < 0 , so Bisection Method FAILS.

TC_4:



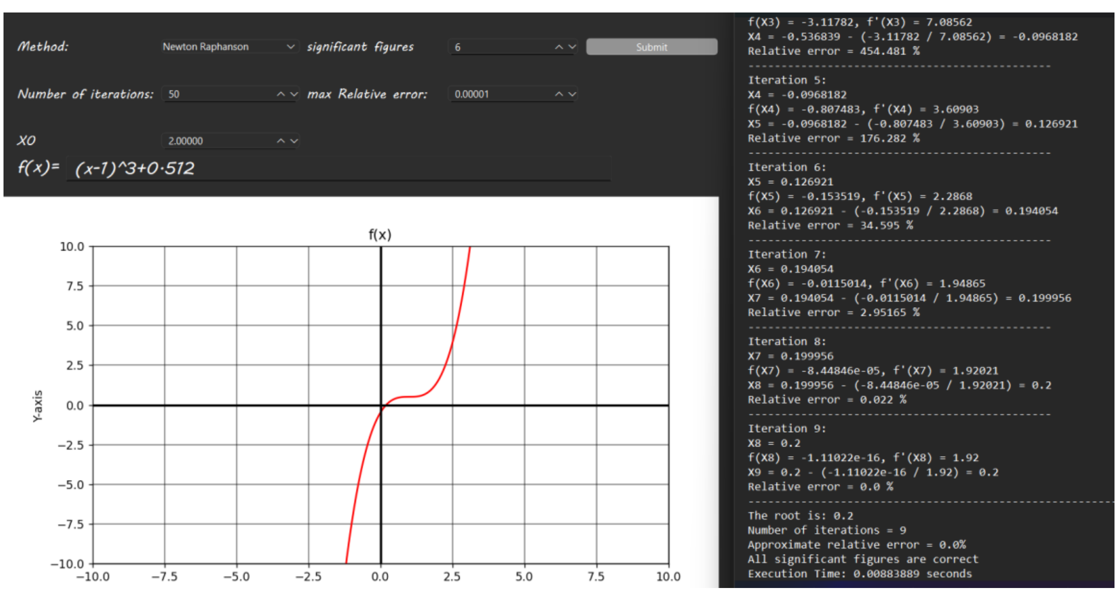
- As shown, it **couldn't converge** within the max number of iterations
  With error = 0.004% > epsilon (0.00001)

TC_5:



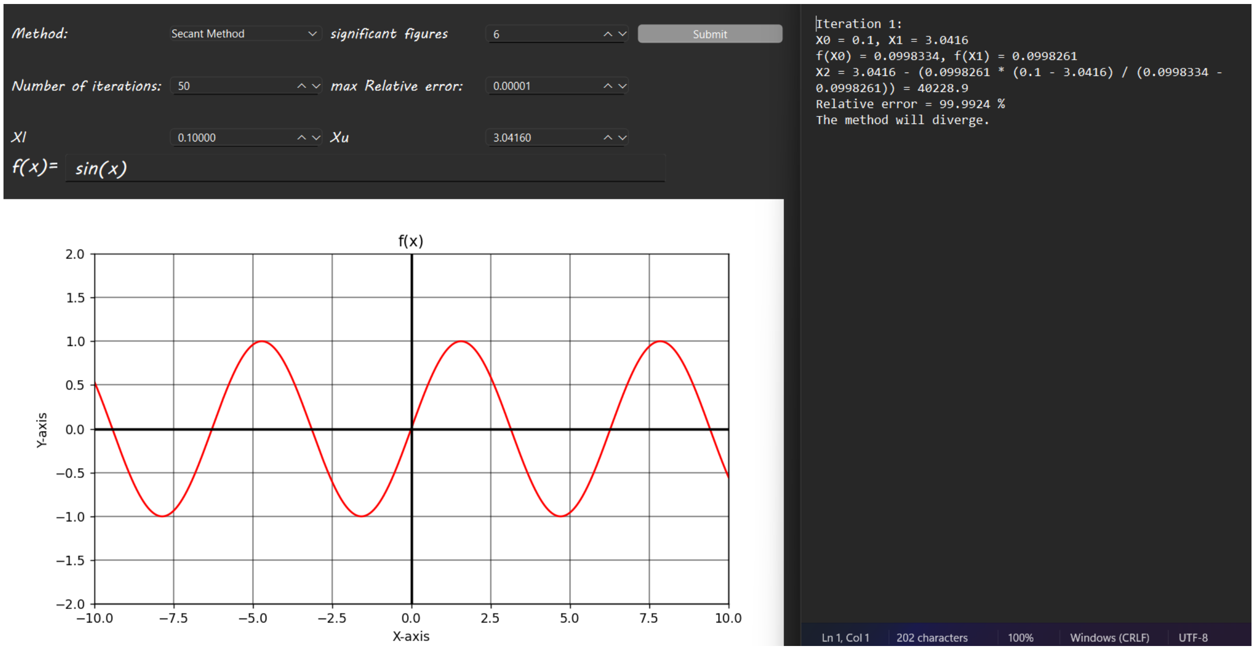
➢ As shown, in fourth iteration X increased enormously in one jump so It will Diverge.

TC_6:



➢ Will converge to exact root wish is 0.2 after 9 iterations with error = 0%

TC_7:



- As shown, it will diverge as x increased enormously and according to my assumption that if x increased enormously in one jump so it will diverge.

4. Comparison between methods

Method	Time	Convergence	Error
Bisection	Very Slow	Always Converge **Converge Linearly**	The error decreases linearly with each iteration.
False Position	Faster than Bisection but still slow	Always converge **Converge Linearly**	the error may decrease faster than Bisection.
Fixed Point	The slower method in the open methods	May converge and may diverge according to initial guess and g(x) **Converge quadratically**	The error decreases according to the convergence rate
Original Newton	Fast (if near the root)	May converge and may diverge according to initial guess **Converge quadratically**	The error typically decreases quadratically after each iteration.
Modified Newton	The fastest Method (if near the root)	May converge and may diverge according to initial guess **Converge quadratically**	The error typically decreases quadratically after each iteration, which makes this method very fast when it works well.
Secant	Fast (Slower than Newton)	May converge and may diverge according to initial guesses **Converge quadratically**	The error in each iteration can decrease roughly

---

5. Data Structures used

- **Arrays:** primarily used for numerical computations and evaluation Like **NumPy arrays**.
- **Lists:** Collects and formats output messages for display or logging.

Advantages:

- Arrays provide direct indexing for rapid access.
- Lists offer dynamic sizing for iterative methods.