Warsaw University of Technology
Faculty of Electronics and Information Technology
Course : ENUME-2020

#11 Assignment B: Solving Nonlinear Equation

I declare that this piece of work, which is the basis for recognition of achieving learning outcomes in the Numerical Methods course, was completed on my own.

Author: Keshav Dandeva [302333]
Advisor: Paweł Mazurek

Warsaw, 5th May 2020

Contents

1. Theoretical Introduction

Let us begin by first understanding the aim of this assignment. This assignment examines and analyses the solving of nonlinear equations with the aid of MATLAB software and implementation and systematic testing of iterative algorithms (IAs) for solving numerical problems. An iterative algorithm can be defined as the repetition of identical or similar sequences of operations. An iterative algorithm (IA) for solving a numerical problem $\mathbf{d} \rightarrow \mathbf{w}$ has the form:

$$\mathbf{w}^{(i+1)} = \phi_i \left(\mathbf{w}^{(i)}, \mathbf{w}^{(i-1)}, ...; \mathbf{d}\right) \quad \text{for } i = 0, 1, ...$$

where: ϕ_i is an operator defined by an algorithm of its implementation;
$\mathbf{w}^{(i)}$ is the i^{th} approximation of the solution to the problem

In this assignment, we are going to focus on **Root-Finding Problem**. We are given an equation $f(x) = 0$, where $f(x)$ is a function of single variable x and our goal is to find the value of x where the function changes sign i.e. root of the function.

Except for some very special functions, it is not possible to find an analytical expression for the root, from where the solution can be exactly determined. To find a root, an interval wherein it is located, should be first identified. This phase is called **root bracketing**.

In order to do the process of root bracketing allegorically, checking values of the function at both ends of the interval is a core part of the procedure. If a function $f(x)$ is continuous on a closed interval $[a, b]$ and $f(a) * f(b) < 0$, then at least one root of $f(x)$ is located within $[a, b]$. If an interval $[a = x1, b = x2]$ does not contain a root $f(x1) * f(x2) > 0$, then a reasonable way to proceed is to expand the interval.

After finding an interval containing a root we can initiate an iterative method. In this method, we start with one or more initial guesses (x_o, x_1) of the root(α) as starting values, then each iteration of the algorithm produces a successively more accurate approximation to the root: $x_n \rightarrow \alpha$ (where $n \rightarrow \infty$). Since the iteration must be stopped at some point these methods produce an approximation to the root, not an exact solution.

Two important aspects of an iterative method are: **convergence** and **stopping criterion**.

Convergence

Generally, order of convergence of an iterative method is defined as the largest number $p \geq 1$ such that:

$$\lim_{n\to\infty} \frac{|x_{n+1} - a|}{|x_n - a|} = k < \infty$$

where the coefficient k is the convergence factor. The larger the order of convergence, the better the convergence of the equation.

Stopping Criteria

The final approximation (x_n) of the root of the equation $f(x)$ is accepted if any one of the following criteria is satisfied:

1. $|x_n| \leq \epsilon$ (The functional value is less than or equal to the tolerance)
2. $\dfrac{|x_{n-1} - x_n|}{x_n} \leq \epsilon$ (The relative change is less than or equal to the tolerance)
3. The number of iterations are greater than or equal to a predetermined number.

The methods used in this assignment to evaluate nonlinear equations are described as follows:

1.1 Bisection Method

The Bisection Method, also called the interval halving method is based on the Bolzano's theorem for continuous functions. For $f(x)$ continuous in $[a_o, b_o]$ and satisfying the condition $f(a_o) \cdot f(b_o) < 0$, the bisection method is defined by the formula:

$$x_i = \frac{1}{2}(a_i + b_i) \qquad \text{for } i = 0, 1, 2...$$

$$[a_{i+1}, b_{i+1}] = \begin{cases} [a_i, x_i] & \text{if } f(a_i) \cdot f(x_i) < 0 \\ [x_i, b_i] & \text{if } f(a_i) \cdot f(x_i) > 0 \end{cases}$$

and the stopping rule for the iterations of this method is:

$$\frac{1}{2}(b_i - a_i) < \Delta \qquad \text{for } \Delta = 10^{-3}, 10^{-4}, ...10^{-16}$$

1.2 Regula-Falsi Method

The Regula-Falsi Method, also called the false position method is also based on the Bolzano's theorem for continuous functions. For $f(x)$ continuous in $[x_o, x_i]$ and satisfying the condition $f(x_o) \cdot f(x_i) < 0$. The regula-falsi method is defined by the formula:

$$x_{i+1} = x_i - \frac{x_i - x_o}{f(x_i) - f(x_o)} f(x_i) \qquad \text{for } i = 0, 1, 2...$$

where: x_o is a constant point
and the stopping rule for the iteration of this method is:

$$|x_i - x_{i-1}| < \Delta \qquad \text{for } \Delta = 10^{-3}, 10^{-4}, ...10^{-16}$$

1.3 Secant Method

The secant method is very similar to the bisection method except instead of dividing each interval by choosing the midpoint the secant method divides each interval by the secant line connecting the endpoints. The secant method is defined by the formula:

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i) \qquad \text{for } i = 0, 1, 2...$$

and the stopping rule for the iteration of this method is:

$$|x_i - x_{i-1}| < \Delta \qquad \text{for } \Delta = 10^{-3}, 10^{-4}, ...10^{-16}$$

1.4 Newton's Method

In Newton's method, we have to provide an approximation for the result and also find the derivative of the given function. The Newton's method is defined by the formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad \text{for } i = 0, 1, 2...$$

and the stopping rule for the iterations is:

$$|x_i - x_{i-1}| < \Delta \qquad \text{for } \Delta = 10^{-3}, 10^{-4}, ...10^{-16}$$

2. Task 1

In this task, we are provided with the equation $f(x) = 0$:

$$f(x) = \left(\frac{6}{5}\right)^x + x + sin(\sqrt{x}) - \frac{33}{4} \text{ for } x \in (0, 10]$$

The goal of the task is to find the root of the function using the MATLAB function *fzero*. The code for this task is attached in the appendix for reference.

2.1 Result

Using the *fzero* function, the root obtained of the function is:

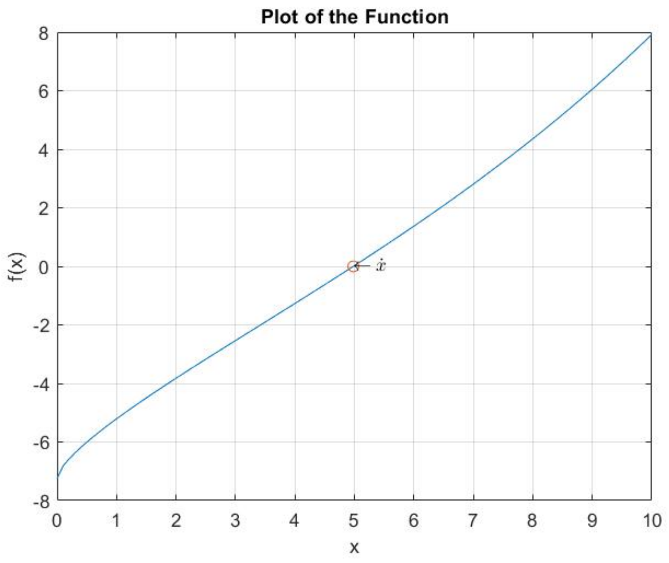
$$\dot{x} = 4.980937540361226$$

Hereafter, this will be referenced as exact solution and will be used in further tasks to evaluate absolute errors.

2.2 Graphical Representation

The graph of the function $f(x)$ is presented below. It is obtained with the help of *plot* function in MATLAB and the exact solution of the equation ($\dot{x}$) is marked on the graph with a red circle.

Figure 2.1: Plot of $f(x)$ with Exact Solution



3. Task 2

In this task, we have to find the root of the equation $f(x) = 0$ by using **Bisection Method**.

We are provided with the initial conditions:
$a_o = 10^{-6}$
$b_o = 10$

This will be used as the initial interval for calculating root for every Δ given. The values of Δ are changed with the help of *for* loop and stored in a vector called *Bisection_ epsilon.*The implementation of the bisection method is done inside the *for* loop.

First, the condition $f(x1) * f(x2) < 0$ is checked and proceeded if it is satisfied.
Then, the iterative algorithm for the bisection method is implemented inside the *while* loop with the terminating condition $\frac{1}{2}(b_i - a_i) < \Delta$.
The roots are calculated for every value of Δ and are stored in the vector *Bisection_ roots.* Actual absolute error is calculated by taking absolute value of the difference between exact solution (from Task 1) and roots from this task:

$$\hat{\Delta}x = |\hat{x} - \dot{x}|$$

The code for this task is attached in Appendix for reference.

3.1 Result

The following table represents the value of root obtained for each Δ and number of iterations required and the actual absolute error for each value:
As we can see from Table 3.1, although we get accurate answer when Δ is relatively lower this method takes a lot of iterations to converge to the root and has high absolute errors.

Table 3.1: Bisection Method

Δ	**Iterations**	**Roots**	**Absolute Error** [$10^{-3}*$]
10^{-3}	13	4.981079603454589	0.142063093362843
10^{-4}	16	4.981003309516906	0.065769155679796
10^{-5}	19	4.980936552321433	0.000988039793093
10^{-6}	23	4.980937148367821	0.000391993404847
10^{-7}	26	4.980937520896815	0.000019464411416
10^{-8}	29	4.980937548836489	0.000008475263158
10^{-9}	33	4.980937540105341	0.000000255885091
10^{-10}	36	4.980937540323620	0.000000037606362
10^{-11}	39	4.980937540369094	0.000000007868373
10^{-12}	43	4.980937540361705	0.000000000478728
10^{-13}	46	4.980937540361207	0.000000000018652
10^{-14}	49	4.980937540361234	0.000000000007994
10^{-15}	52	4.980937540361226	0
10^{-16}	54	4.980937540361227	0.000000000000888

3.2 Graphical Representation

A counter variable is used to count the number of iterations done for each value of Δ and then stored in a vector called *Bisection_Iterations*. This vector is used along with *Bisection_ epsilon* vector to plot a graph representing trend of Iteration vs Δ (Figure 3.1). The *semilogx* function is used to plot this graph.

Another graph(Figure 3.2) representing the relation between actual absolute error of the solution and criterion value is presented below. The *loglog* function is used to plot this graph.

Figure 3.1: Bisection Method: I vs Δ

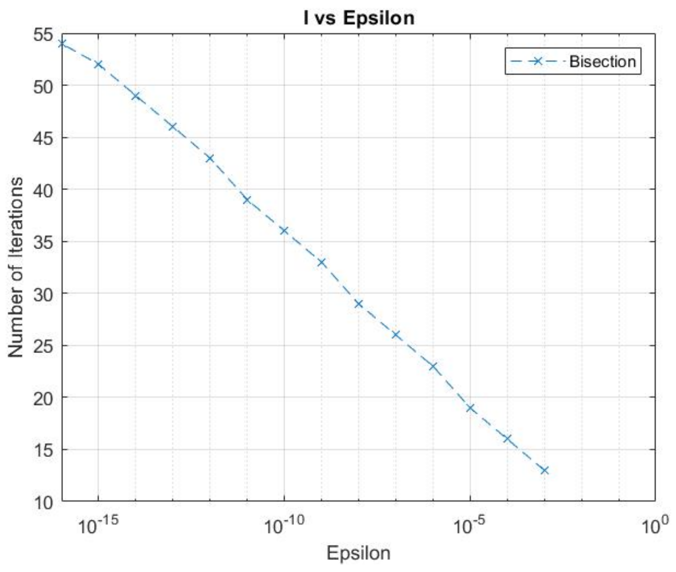
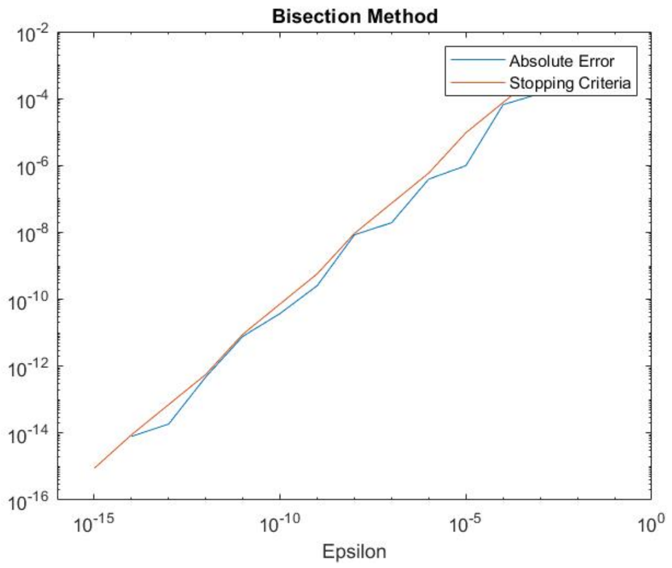


Figure 3.2: Absolute Error and Stopping Criteria vs Δ



4. Task 3

In this task, we have to find the root of the equation $f(x) = 0$ by using **Regula-Falsi Method**.

Here, we are asked to set the initial conditions as per our choice. A constant integer point x_o is to be used as initial value for this method and should be not closer to than 0.5 and a proper starting point x_1 is also set.
$x_o = 6$
$x_1 = 0$

In this procedure, the point x_o remains constant and x_1 changes its value dynamically with every iteration to get closer to the root. This is used as the converging interval for calculating root for every Δ given. The values of Δ are changed with the help of *for* loop and stored in a vector called *RFM_ epsilon.* The implementation of the Regula-falsi method is done inside the *for* loop.
The code for this task is attached in Appendix for reference.

The iterative algorithm for this method is implemented inside the *while* loop with the terminating condition of iterations < 100 or $|x_i - x_{i-1}| < \Delta$.
The roots are calculated for every value of Δ and are stored in the vector *RegulaFalsi_ roots.* Actual absolute error is calculated by taking absolute value of the difference between exact solution (from Task 1) and roots from this task:

$$\hat{\Delta}x = |\hat{x} - \dot{x}|$$

4.1 Result

The following table depicts the value of root obtained for each Δ and number of iterations required and the actual absolute error for each value:
As we can see from Table 4.1, although we get accurate answer when Δ is relatively lower this method is a slow convergent because the endpoint of interval is not changing during the whole iterations.

Table 4.1: Regula-Falsi Method

Δ	**Iterations**	**Roots**	**Absolute Error [$10^{-5}*$]**
10^{-3}	3	4.980938547527331	0.100716610518248
10^{-4}	3	4.980938547527331	0.100716610518248
10^{-5}	4	4.980937565661987	0.002530076148588
10^{-6}	4	4.980937565661987	0.002530076148588
10^{-7}	5	4.980937540996800	0.000063557425989
10^{-8}	6	4.980937540377192	0.000001596589527
10^{-9}	6	4.980937540377192	0.000001596589527
10^{-10}	7	4.980937540361627	0.000000040145665
10^{-11}	8	4.980937540361236	0.000000000976996
10^{-12}	8	4.980937540361236	0.000000000976996
10^{-13}	9	4.980937540361227	0.000000000088818
10^{-14}	9	4.980937540361227	0.000000000088818
10^{-15}	10	4.980937540361226	0
10^{-16}	11	4.980937540361226	0

4.2 Graphical Representation

A counter variable is used to count the number of iterations done for each value of Δ and then stored in a vector called *RFM_ Iterations*. This vector is used along with *RFM_ epsilon* vector to plot a graph representing trend of Iteration vs Δ (Figure 4.1). The *semilogx* function is used to plot this graph.

Another graph representing the relation between actual absolute error of the solution and criterion value is presented below. The *loglog* function is used to plot this graph.

Figure 4.1: Regula-Falsi Method: I vs Δ

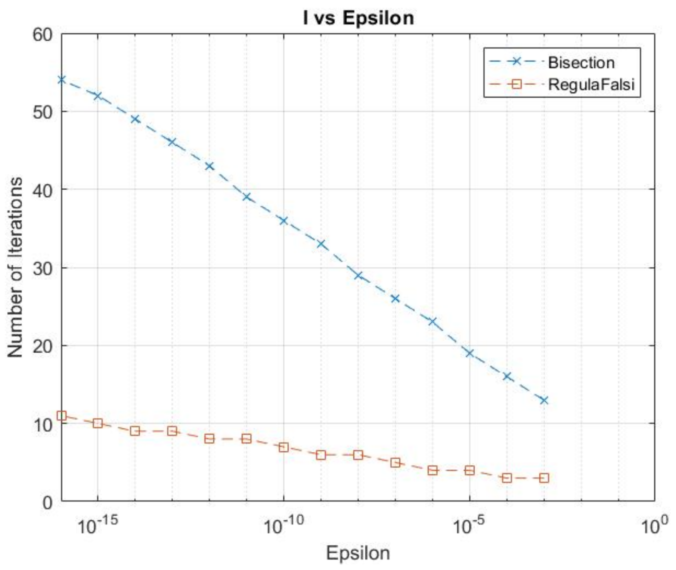
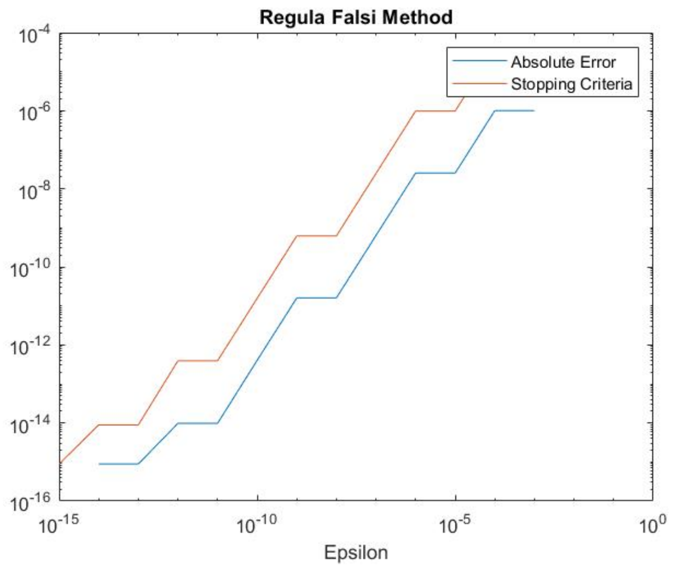


Figure 4.2: Absolute Error and Stopping Criteria vs Δ



5. Task 4

In this task, we have to find the root of the equation $f(x) = 0$ by using **Secant Method**.

Here, we are asked to set the initial conditions as per our choice.x_o and x_1 are chosen as proper starting points.This is the initial interval for the iterative method to converge.
$x_o = 0$
$x_1 = 10$

In this procedure, the initial chosen interval changes its value dynamically with every iteration converging towards the root. This is used for calculating root for every Δ given. The values of Δ are changed with the help of *for* loop and stored in a vector called *Secant_ epsilon*. The implementation of the Secant method is done inside the *for* loop.
The code for this task is attached in Appendix for reference.

The roots are calculated for every value of Δ and are stored in the vector *Secant_ roots*. The iterative algorithm for this method is implemented inside the *while* loop with the terminating condition of $|x_i - x_{i-1}| < \Delta$.
Actual absolute error is calculated by taking absolute value of the difference between exact solution (from Task 1) and roots from this task:

$$\hat{\Delta}x = |\hat{x} - \dot{x}|$$

5.1 Result

The following table depicts the value of root obtained for each Δ and number of iterations required and the actual absolute error for each value:
As we can see from Table 5.1, in Secant method we get the accurate results in less iterations than the previous methods and also the magnitude of absolute error decreases up to a great extent.

Table 5.1: Secant Method

Δ	**Iterations**	**Roots**	**Absolute Error** $[10^{-12}*]$
10^{-3}	4	4.980937540360924	0.301980662698043
10^{-4}	5	4.980937540361227	0.000888178419700
10^{-5}	5	4.980937540361227	0.000888178419700
10^{-6}	5	4.980937540361227	0.000888178419700
10^{-7}	5	4.980937540361227	0.000888178419700
10^{-8}	6	4.980937540361225	0.000888178419700
10^{-9}	6	4.980937540361225	0.000888178419700
10^{-10}	6	4.980937540361225	0.000888178419700
10^{-11}	6	4.980937540361225	0.000888178419700
10^{-12}	6	4.980937540361225	0.000888178419700
10^{-13}	7	4.980937540361226	0
10^{-14}	7	4.980937540361226	0
10^{-15}	8	4.980937540361226	0
10^{-16}	9	4.980937540361226	0

5.2 Graphical Representation

A counter variable is used to count the number of iterations done for each value of Δ and then stored in a vector called *Secant_ Iterations*. This vector is used along with *Secant_ epsilon* vector to plot a graph representing trend of Iteration vs Δ (Figure 5.1). The *semilogx* function is used to plot this graph.

Another graph representing the relation between actual absolute error of the solution and criterion value is presented below. The *loglog* function is used to plot this graph.

Figure 5.1: Secant Method: I vs Δ

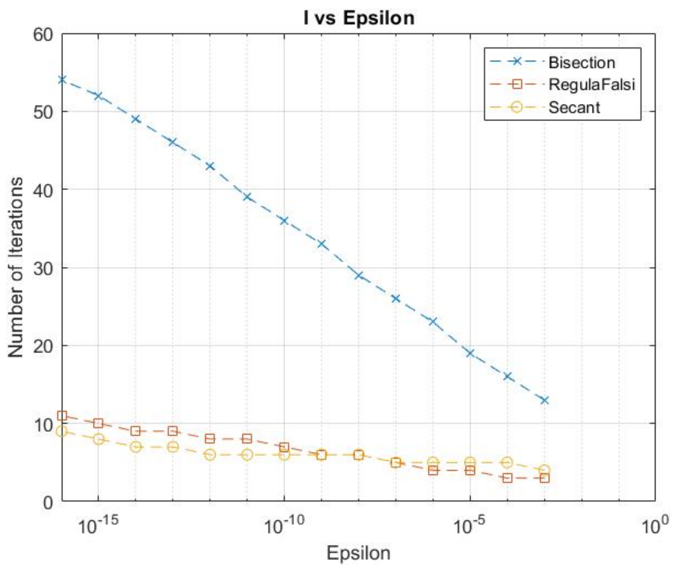
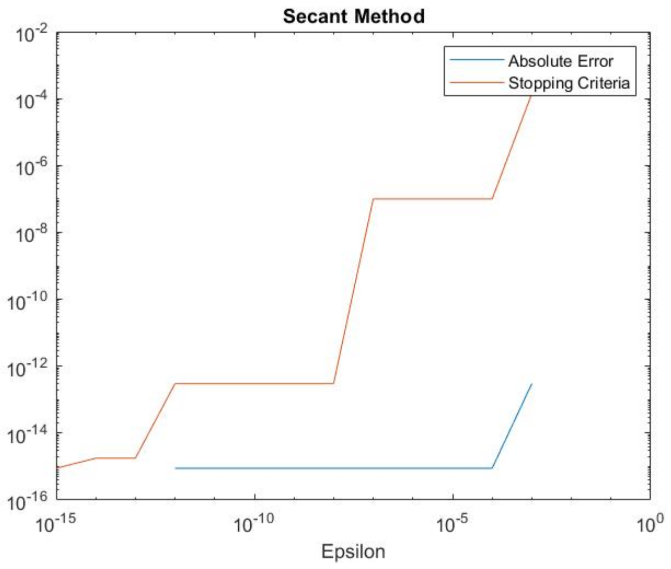


Figure 5.2: Absolute Error and Stopping Criteria vs Δ



6. Task 5

In this task, we have to find the root of the equation $f(x) = 0$ by using **Newton's Method**.

In this method, we are only required to set a starting point x_0.This point is used by the method as an estimation of the root and can be chosen by looking at the graph of the function.
$x_o = 4$
In this method, we also have to calculate the derivative of the given function $f(x)$ with respect to x analytically.
$f(x) = \left(\frac{6}{5}\right)^x + x + sin(\sqrt{x}) - \frac{33}{4}$
$f'(x) = \frac{\ln(6) \cdot 6^x}{5^x} - \frac{\ln(5) \cdot 6^x}{5^x} + \frac{\cos(\sqrt{x})}{2\sqrt{x}} + 1$
In this procedure, we start with the initial approximation x_o. This changes its value dynamically with every iteration to get closer to the root. This is repeated for every Δ given. The values of Δ are changed with the help of *for* loop and stored in a vector called *Newton_ epsilon.*The implementation of the Newton's method is done inside the *for* loop.
The code for this task is attached in Appendix for reference.

The roots are calculated for every value of Δ and are stored in the vector *Newton's_ roots.* The iterative algorithm for this method is implemented inside the *while* loop with the terminating condition of iterations < 50 or $|x_i - x_{i-1}| < \Delta$.
Actual absolute error is calculated by taking absolute value of the difference between exact solution (from Task 1) and roots from this task:

$$\hat{\Delta}x = |\hat{x} - \dot{x}|$$

6.1 Result

The following table depicts the value of root obtained for each Δ and number of iterations required and the actual absolute error for each value:
As we can see from Table 6.1,this method provides the most accurate results with the least iterations possible.

Table 6.1: Newton's Method

Δ	**Iterations**	**Roots**	**Absolute Error** $[10^{-12}*]$
10^{-3}	2	4.980937540361573	0.347277762102749
10^{-4}	2	4.980937540361573	0.347277762102749
10^{-5}	2	4.980937540361573	0.347277762102749
10^{-6}	3	4.980937540361226	0
10^{-7}	3	4.980937540361226	0
10^{-8}	3	4.980937540361226	0
10^{-9}	3	4.980937540361226	0
10^{-10}	3	4.980937540361226	0
10^{-11}	3	4.980937540361226	0
10^{-12}	3	4.980937540361226	0
10^{-13}	4	4.980937540361226	0
10^{-14}	4	4.980937540361226	0
10^{-15}	4	4.980937540361226	0
10^{-16}	4	4.980937540361226	0

6.2 Graphical Representation

A counter variable is used to count the number of iterations done for each value of Δ and then stored in a vector called *Newton_Iterations*. This vector is used along with *Newton_ epsilon* vector to plot a graph representing trend of Iteration vs Δ (Figure 4.1). The *semilogx* function is used to plot this graph.

Another graph representing the relation between actual absolute error of the solution and criterion value is presented below. The *loglog* function is used to plot this graph.

Figure 6.1: Newton's Method: I vs Δ

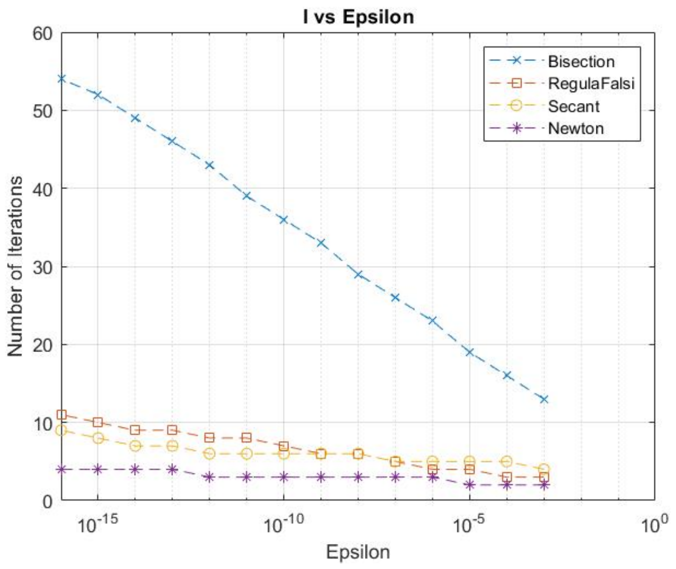
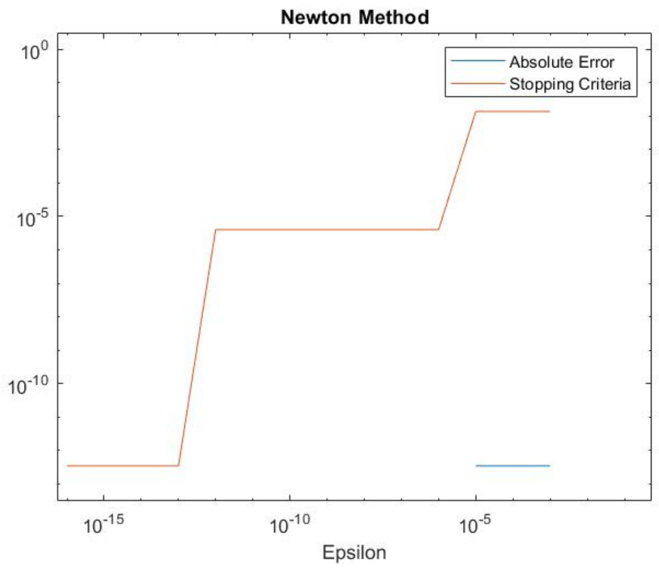


Figure 6.2: Absolute Error and Stopping Criteria vs Δ



7. Task 6

In this task, we have to examine the impact of the initial points provided to the methods on the relationships presented in the Figures 2-6.

In **Bisection Method**, the provided initial interval $[a_0, b_0]$ was changed and the difference between them was increased at a large scale. Due to this, there was a little increase in the number of iterations for all the values of Δ. As is depicted in the Figure 7.1. Due to the increase in the initial interval, the absolute error values and stopping criteria values were also a bit closer to each other. As is depicted in Figure 7.2.

In **Regula-Falsi Method**, the value of the starting point and the value of the fixed point was changed such that the initial interval was increased at a large scale. As a result of this, for higher Δ it takes a lot of iterations and for lower Δ it reaches to the maximum limit of iterations set. This is depicted in the Figure 7.3.Also, the absolute error decreases upto certain Δ and then stays constant for all the lower Δ. As is depicted in Figure 7.4.

In **Secant Method**, the value of the starting points: x_0, x_1 was changed such that the initial interval was present before the root of the equation. As a result of this, there is not much fluctuation in iterations but absolute error value decreases to a great extent. This is depicted in the Figure 7.5 and Figure 7.6.
Also, when the initial interval is increased to a great extent, the number of iterations increase but stay almost same for all different values of Δ and the absolute error increases to great magnitude and stays constant for all Δ. This is depicted in the Figure 7.7 and Figure 7.8.

In **Newton's Method**, the value of the starting point x_o is chosen arbitrarily. As a result of this,for all the values Δ it reaches to the maximum limit of iterations set. This is depicted in the Figure 7.9. Also, the absolute error and stopping criteria do not have any value available. As is depicted in Figure 7.10.

Figure 7.1: Task-6: Bisection Method

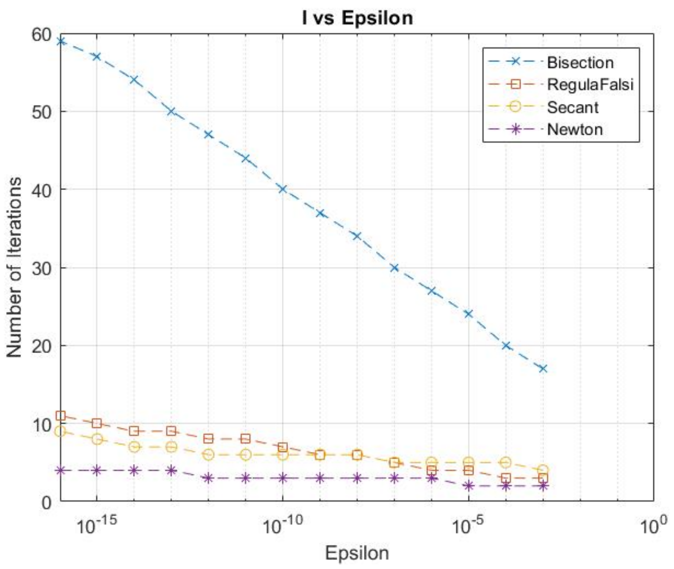


Figure 7.2: Task-6: Bisection Method

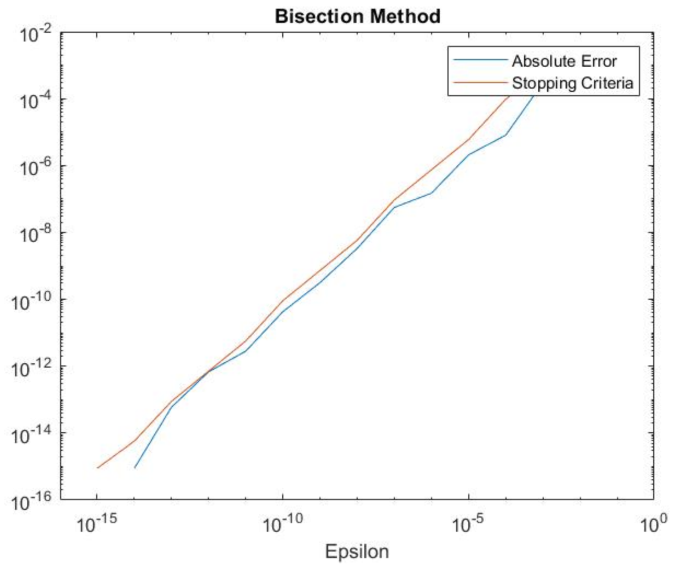


Figure 7.3: Task-6: Regula-Falsi Method

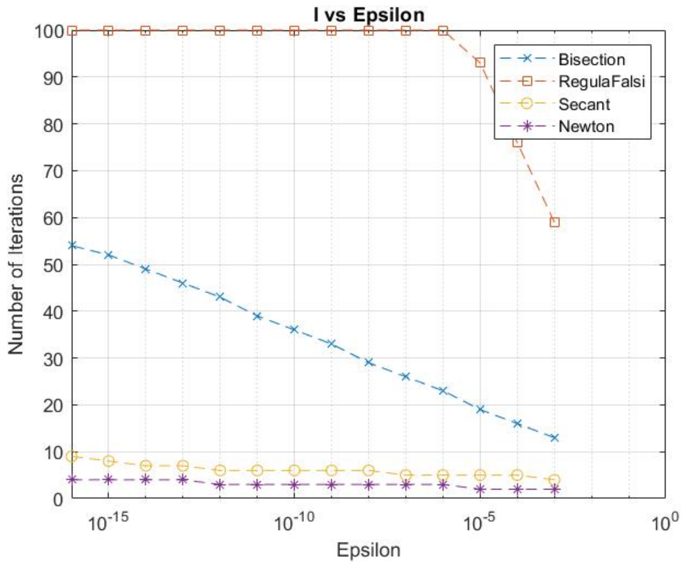


Figure 7.4: Task-6: Regula-Falsi Method

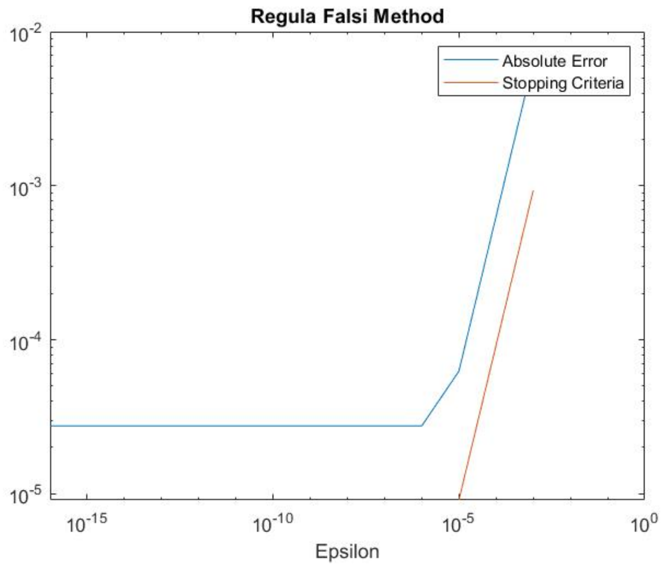


Figure 7.5: Task-6: Secant Method

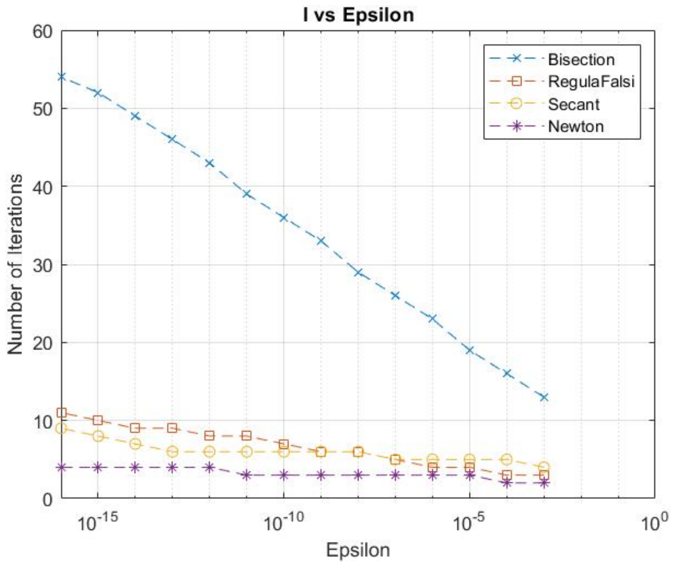


Figure 7.6: Task-6: Secant Method

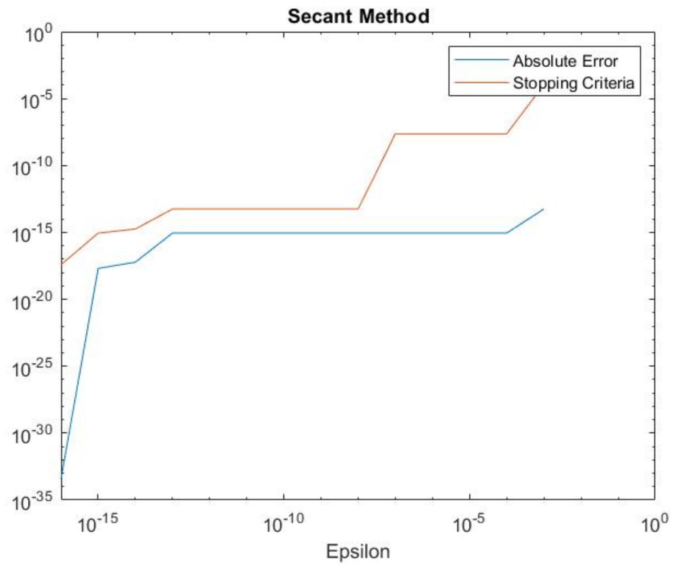


Figure 7.7: Task-6: Secant Method

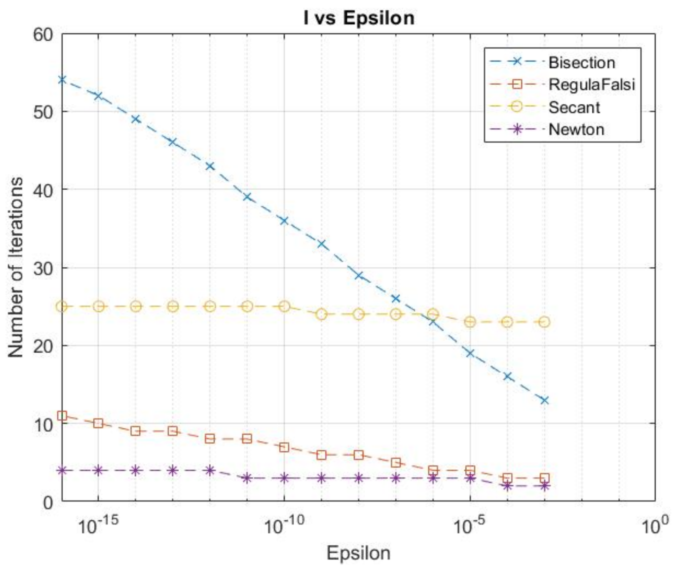


Figure 7.8: Task-6: Secant Method

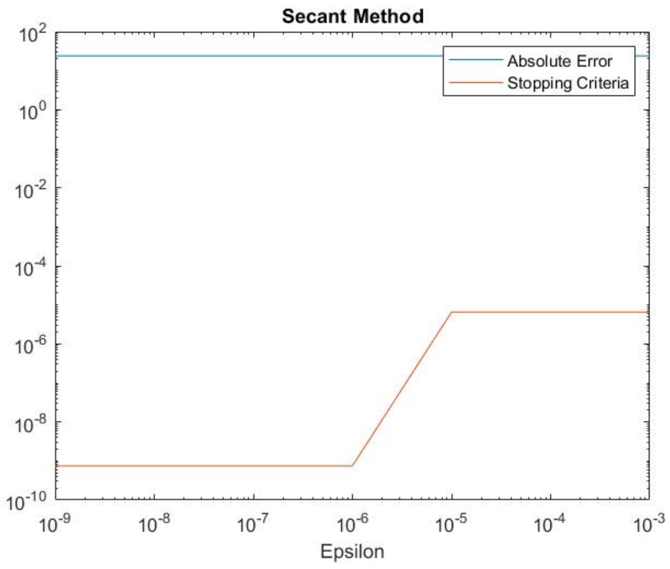


Figure 7.9: Task-6: Newton's Method

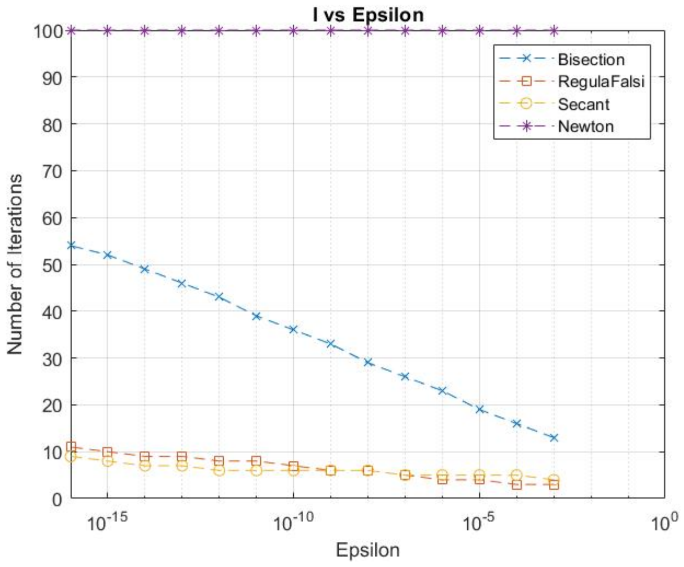
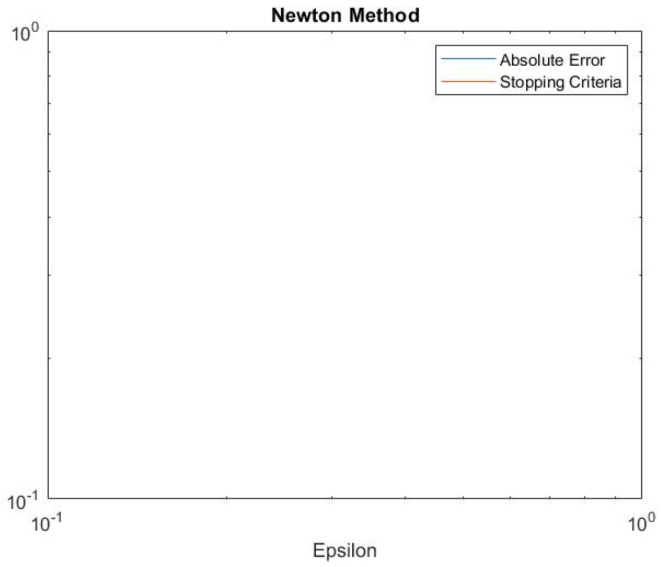


Figure 7.10: Task-6: Newton's Method



8. Conclusion

Bisection Method and **Regula-Falsi Method** always converges to the root of the equation. Regula-falsi converges faster than the bisection method as we can see from the number of iterations required by both to converge to root of the equation.

In **Regula-Falsi Method**, one of the approximation is constant for all the iterations and the step size does not go to zero when converged.Unlike the **Bisection Method**, the reduction in interval for regula-falsi is not consistent.

Secant Method is faster than the **Regula-Falsi Method**. It uses existing function evaluations to approximate the derivative of the function. It does converge slightly slower than **Newton's method**, however.

Newton's method is a quadratic convergence method that applies to initial guesses that are close to a root. For an arbitrary initial guess, Newton's method can result in divergence or convergence to a far-away root(if present).

In Conclusion, Newton's method and **Secant Method** are faster but initial guess is important for ensuing convergence. On the other hand, **Regula-Falsi Method** and **Bisection Method** guarantee convergence but take longer time and more iterations.

9. Appendix

9.1 Task 1

```
y =@(x) (6/5).^x + x + sin(x.^(1/2)) - 33/4 ;
x0 = [0 10];
Exact_Solution = fzero(y,x0);

figure(1)
x = linspace(0,10,100);
y = (6/5).^x + x + sin(x.^(1/2)) - 33/4 ;
plot(x,y);
grid on;
hold on;
plot(Exact_Solution,0,'-o');
text(Exact_Solution,0,'$\leftarrow \dot{x}$', 'Interpreter','latex');
xlabel('x');
ylabel('f(x)');
title('Plot of the Function');
```

9.2 Task 2

```
for i = (1:14)
    Δ = 10.^−(i+2);
    Bisection_epsilon(i) = Δ;

    f =@(x) (6/5).^x + x + sin(x.^(1/2)) − 33/4 ;
    a = 10^(−6);
    b = 10;
    xi = (a + b)./2;
    counter = 0;

    if f(a)*f(b)>0
        disp('Error! Values provided does not satisfy Bisection method')
    else
        while abs((b−a)./2) > Δ
            if f(a)*f(xi)<0
                b = xi;
            else
                a = xi;
            end
            xi = (a + b)/2;
            counter = counter + 1;

        end
        Bisection_StoppingValue(i) = abs((b−a)./2);
        Bisection_AbsoluteError(i) = abs(Exact_Solution − xi);
        Bisection_Roots(i) = xi;
        Bisection_Iterations(i) = counter;
    end
end

figure(2);
semilogx(Bisection_epsilon,Bisection_Iterations,'−−x');
grid on
legend('Bisection')
xlabel('Epsilon');
ylabel('Number of Iterations');
title('I vs Epsilon');

figure(3);
loglog(Bisection_epsilon,Bisection_AbsoluteError);
hold on
loglog(Bisection_epsilon,Bisection_StoppingValue);
hold off
legend('Absolute Error','Stopping Criteria')
xlabel('Epsilon');
title('Bisection Method');
```

9.3 Task 3

```
for i=1:14
    Δ2 = 10.^−(i+2);
    RFM_epsilon(i) = Δ2;
    f =@(x) (6/5).^x + x + sin(x.^(1/2)) − 33/4 ;
    x0 = 6;
    f0 = f(x0);
    x1 = 0;
    f1 = f(x1);
    count = 0;
    while count < 100
        x2 = x1 −((x1−x0)./(f1−f0)).*f1;

        if abs(x2−x1) < Δ2
            break;
        end

        x1 = x2;
        f1 = f(x1);
        count = count + 1;
    end
    RFM_StoppingValue(i) = abs(x2−x1);
    RFM_roots(i) = x2;
    RFM_AbsoluteError(i) = abs(Exact_Solution − x2);
    RFM_Iterations(i) = count;
end

figure(2);
semilogx(Bisection_epsilon,Bisection_Iterations,'−−x');
grid on
hold on
semilogx(RFM_epsilon,RFM_Iterations,'−−s');
hold off
legend('Bisection','RegulaFalsi')
xlabel('Epsilon');
ylabel('Number of Iterations');
title('I vs Epsilon');

figure(4);
loglog(RFM_epsilon,RFM_AbsoluteError);
hold on
loglog(RFM_epsilon,RFM_StoppingValue);
hold off
legend('Absolute Error','Stopping Criteria')
xlabel('Epsilon');
title('Regula Falsi Method');
```

9.4 Task 4

```
for i = 1:14
    Δ3 = 10^-(i+2);
    Secant_epsilon(i) = Δ3;

    f =@(x) (6/5).^x + x + sin(x.^(1/2)) - 33/4 ;
    x0 = 0;
    x1 = 10;

    count = 0;
    while true
        x2 = x1 - (((x1-x0).*f(x1))./(f(x1)-f(x0)));

        if abs(x1-x0) < Δ3
            break;
        end
        count = count + 1;
        x0 = x1;
        x1 = x2;

    end
    Secant_Iterations(i) = count;
    Secant_StoppingValue(i) = abs(x1-x0);
    Secant_roots(i) = x2;
    Secant_AbsoluteError(i) = abs(Exact_Solution - x2);
end

figure(2);
semilogx(Bisection_epsilon,Bisection_Iterations,'--x');
grid on
hold on
semilogx(RFM_epsilon,RFM_Iterations,'--s');
semilogx(Secant_epsilon,Secant_Iterations,'--o');
hold off
legend('Bisection','RegulaFalsi','Secant')
xlabel('Epsilon');
ylabel('Number of Iterations');
title('I vs Epsilon');

figure(5);
loglog(Secant_epsilon,Secant_AbsoluteError);
hold on
loglog(Secant_epsilon,Secant_StoppingValue);
hold off
legend('Absolute Error','Stopping Criteria')
xlabel('Epsilon');
title('Secant Method');
```

9.5 Task 5

```
f =@(x) (6/5).^x + x + sin(x.^(1/2)) - 33/4 ;
df =@(x) ((log(6).*6.^x)-(log(5).*6.^x))./5.^x + (cos(x.^(1/2)))./(2.*x.^(1/2)) + 1;


for i = 1:14
    ∆4 = 10^-(i+2);
    Newton_epsilon(i) = ∆4;
    x0 = 4;

    count = 0;
    while count < 50
        x1 = x0 - (f(x0)./df(x0));
        if abs(x1-x0) < ∆4
            break;
        end
        temp_x0 = x0;
        x0 = x1;

        count = count + 1;
    end
    Newton_Iterations(i) = count;
    Newton_roots(i) = x1;
    Newton_StoppingValue(i) = abs(x1-temp_x0);
    Newton_AbsoluteError(i) = abs(Exact_Solution - x1);
end

figure(6);
loglog(Newton_epsilon,Newton_AbsoluteError);
hold on
loglog(Newton_epsilon,Newton_StoppingValue);
hold off
legend('Absolute Error','Stopping Criteria')
xlabel('Epsilon');
title('Newton Method');

figure(2);
semilogx(Bisection_epsilon,Bisection_Iterations,'--x');
grid on
hold on
semilogx(RFM_epsilon,RFM_Iterations,'--s');
semilogx(Secant_epsilon,Secant_Iterations,'--o');
semilogx(Newton_epsilon,Newton_Iterations,'--*');
hold off
legend('Bisection','RegulaFalsi','Secant','Newton')
xlabel('Epsilon');
ylabel('Number of Iterations');
title('I vs Epsilon');
```

10. References

- ENUME Lecture Notes 2020 - Prof Roman Morawski
- ENUME 2020 Assignment B: Introduction Video - Prof Pawel Mazurek