**Numerical Analysis**

**Phase 1 – Root Finding**

**Algorithms**

**Dahlia Chehata Mahmoud 28**

**Moamen Raafat El-Baroudy 52**

**Mohamed Ayman Farrag 61**

**Mahmoud Mohamed Hussein 70**

**Moustafa Mahmoud Reda El-Deen 75**

1. **Algorithms & Pseudo Code:**
   1. **Bisection:**
      1. **Description:**

The Bisection method is an algorithm the repeatedly bisects an interval into two halves and then select a half in which a root must lie in for further processing.

* + 1. **Pros:**
       1. The Basic idea is easy and robust.
       2. It'll always converge to a root if the interval is valid.
       3. Number of the iterations required to attain an absolute error can be computed theoretically.
    2. **Cons:**
       1. The Method may appear to be fast due to its logarithmic time complexity but It's slow compared to other methods like Newton Raphson or Regular Falsi.
       2. It's required to find a valid interval which contains an odd number of roots to begin the algorithm.
       3. It doesn't use the fact that which side has the minimum absolute value is closer to the root.
    3. **Pseudo Code:**



* 1. **Regular Falsi:**
     1. **Description:**

Regula Falsi assumes that is linear even though these methods are needed only when is not linear and usually work well anyway.

The ratio of the change in , to the result change in is:

Then we try to interpolate the position of the root on the x-axis but since the function doesn't represent a linear function, in most cases it won't be the root so we choose a new interval with the new point and one of the previous two points which makes the interval valid.

* + 1. **Pros:**
       1. It'll always converge to a root if the interval is.
       2. The method converges quickly and fast compared to the Bisection in most cases.
    2. **Cons:**
       1. The Method may become slow or get stuck at all in case of convex and concave curves near the x-axis if they aren’t handled separately.
       2. It's required to find a valid interval which contains an odd number of roots to begin the algorithm.
    3. **Pseudo Code:**



* 1. **Fixed Point:** 
     1. **Description:**

a fixed point (sometimes shortened to fix point, also known as an invariant point) of a function is an element of the function's domain that is mapped to itself by the function. That is to say, c is a fixed point of the function f(x) if and only if f(c) = c.

Not all functions have fixed points: for example, if f is a function defined on the real numbers as f(x) = x + 1, then it has no fixed points, since x is never equal to x + 1 for any real number. In graphical terms, a fixed point means the point (x, f(x)) is on the line y = x, or in other words the graph of f has a point in common with that line.

* + 1. **Pros:**
       1. There can be multiple valid g(x) for a function f(x) where they can give multiple options and some converge faster.
    2. **Cons:**
       1. The Method doesn’t guarantee convergence where upon the g(x) it may sometimes diverge.
    3. **Pseudo Code:**



* 1. **Newton Raphson:** 
     1. **Description:**

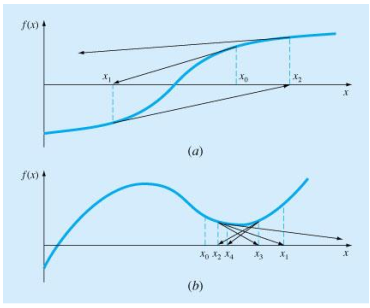
The Newton–Raphson method in one variable is implemented as follows:

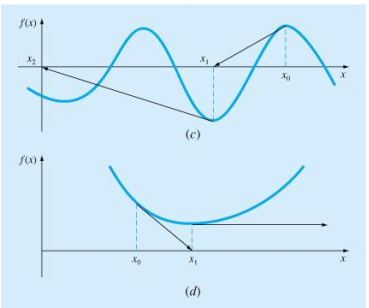
The method starts with a function defined over the real numbers x, the function's derivative , and an initial guess for a root of the function f. If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x1 is

Geometrically, (x1, 0) is the intersection of the x-axis and the tangent of the graph of at

* + 1. **Pros:**
       1. When the initial guess is near the root, the method usually converges.
       2. The Method converges Quadratically when it doesn’t diverge.
       3. Knowledge of the multiplicity of the roots can speed the convergence for a non-1 value multiplicity.
       4. A good algorithm can identify if the iterative process is diverging or not.
    2. **Cons:**
       1. The calculation of the derivative of the function isn’t always possible or straight forward.
       2. There’s no general convergence criteria, It depends on the nature and accuracy of the initial guess.
       3. If the first derivative isn’t well behaved near the initial guess the method may overshoot and diverge.
       4. The Method may misbehave by going into a cycle or jump to another loop.
       5. Slow convergence for roots of multiplicity other than 1.
       6. Zero slope causes division by zero because of the derivative which needs to be handled separately in the code.
    3. **Pseudo Code:**







* 1. **Secant:** 
     1. **Description:**

is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite difference approximation of Newton's method. However, the method was developed independently of Newton's method, and predates it by over 3,000 years.

* + 1. **Pros:**
       1. Knowledge of the multiplicity of the roots can speed the convergence for a non-1 value multiplicity.
       2. It’s very fast compared to bisection and other slow algorithms.
    2. **Cons:**
       1. There’s no general convergence criteria, It depends on the nature and accuracy of the initial guess.
       2. It requires two initial guesses instead of one in Newton Raphson.
       3. It’s slower than Newton Raphson.
    3. **Pseudo Code:**



* 1. **Birge Vieta:** 
     1. **Description:**

This is an iterative method to find a real root of the **n**th degree polynomial equation.

* + 1. **Pseudo Code:**



* 1. **General Algorithm(Illinois):** 
     1. **Introduction:**

In practical root-finding problems and the industry where the initial estimates of the roots are known or at least the range where you can find the initial guess there's a wide range of computationally efficient algorithms to use.

However, when using these algorithms there's two points to consider.

First Not every Method converges for example such methods as Secant, Newton Raphson will sometime fail to converge and may diverge, cycle itself, or get stuck. Second Although some Methods will always converge to a solution like Bisection but It's very slow compared to the number of the required iterations by other methods like Newton Raphson, Secant. That was the reason of using Regula Falsi.

We consider the equation

**F(x) = 0**

We'll start with two approximation of the root where their function values must have an opposite sign to be able to continue. then we compute  
 . Then for the next iteration we will use and either or whichever

gives us a valid interval containing an odd number of roots. The iterative process will continue until some criterion is satisfied.

For example, reaching a certain small value of the error or a certain previously specified number of iterations.

Although Regula Falsi may seem to be an excellent Method but it isn’t optimal due to one main draw back. In case of convex or concave curves near the x-axis Regular Falsi starts to get slow reducing its asymptotic convergence to Linear convergence and may in some extreme cases get stuck and stop changing its value at all.

We’ll introduce a modification to handle such case.

* + 1. **Modification:**

The first solution that comes to mind to handle the slow convergence might be using Bisection, when detecting that the change in the absolute approximate error reaches a certain limit so we can instead switch for the Bisection method for 3 or 4 Iterations until maybe the part covering the Convex or concave part may have passed.

Such solution is straight forward but It has a binding that when trying to escape slow convergence of Bisection, we still used it again and we will be bound by the logarithmic complexity.

1. **Detection:**

We will detect the convex or concave curve in the situations required as a non-changing side for the interval where if doesn’t change for two consecutive iteration then we may consider using the modification later specified in this section.

1. **Modification:**

Assuming that we have two initial Guesses a and b.

C: the new approximation of the root.



The above pseudo code describes the iterative process.

This method was used because Regula Falsi is just an interpolation and in case of convex or concave, we can identify the side which doesn’t change and by dividing it by two we balance the ratio of the cut in x-axis and enhance the converging speed.

In the first Picture is the normal Regula Falsi and in the second picture we can see that the change of the value of F(b) resulted in decreasing the slope of the line and speeding the algorithm.

It can be noted that the first step will be the same but in case of the second step, the Illinois got an approximation which was attained in the third iteration in the normal Regula Falsi.

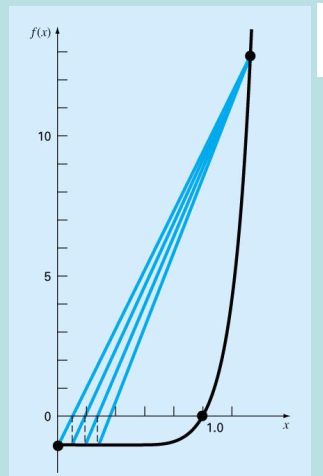
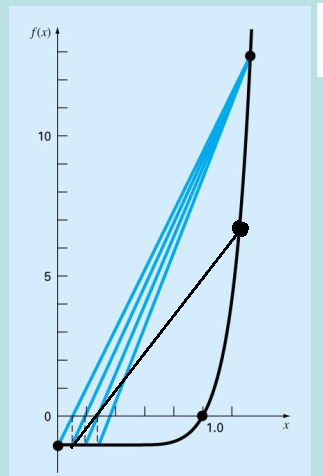
 

Figure 1(Regula Falsi) Figure 2(Illinois)

* + 1. **Pseudo code:**

There was no data structure required in the Algorithm other than normal data value storage represented in variables used to store the value



* + 1. **Difference:**

These functions have a sloppy curvature that slows down the Regular Falsi on the interval of [0.01,1] with a tolerance of 10-6

|  |  |  |  |
| --- | --- | --- | --- |
| **n\Method** | **Bisection** | **Regular Falsi** | **Illinois** |
| **2** | **1** | **1** | **1** |
| **5** | **20** | **16** | **6** |
| **10** | **20** | **32** | **9** |
| **15** | **20** | **47** | **10** |

|  |  |  |  |
| --- | --- | --- | --- |
| **n\Method** | **Bisection** | **Regular Falsi** | **Illinois** |
| **2** | **20** | **457** | **13** |
| **5** | **20** | **208** | **13** |
| **10** | **20** | **110** | **13** |
| **15** | **20** | **74** | **12** |

1. **Problematic Functions:**
   * + 1. **Regular Falsi:**

One of the biggest disadvantages of the Regular Falsi is that in the Convex and concave curves near the x-axis its convergence tends to become slower and also may even get stuck and the method won't converge at all.  
There're two solutions to this problem:

* + - * 1. We can try to detect when the convergence become slower and then use bisection for 3~5 iterations until the Regular Falsi method can converge again.
        2. We can use the Illinois modification where the value of the non-changing end of the interval can be divided by two to increase the convergence speed that it may even has faster convergence than the normal Regular Falsi in normal cases.
      1. **Fixed point:**

The main problems with the Fixed Point Method.

1. The function G(x) may converge very slow or fast or may even diverge depending on the nature of the initial guess and the function G(x).  
   The solution of this problem is to change the function G(x) or try to use different initial guesses and we can use the rules describing the rates of convergence for the algorithm for this G(x) to avoid using the method on a G(x) that may diverge.
   * + 1. **Newton Raphson:**
          1. One main problem of Newton Raphson is that it may diverge if the initial guess isn't near a root for this function or may converge slowly. By calculating the rate of convergence successively a good software can identify if the algorithm is diverging or not.
          2. It may go into a cycle iterating over the same set of points. To avoid this pitfall, we can save the previous points in an array or any data structure and in case of repeating a previous pattern we identify the current state as a loop.
          3. Local maximum or minimum can cause Oscillation of the Newton Raphson Method around the local maximum or minimum. To avoid this pitfall, we can try to pick the guess as far as possible from the local maximum or minimum and as near as possible from the expected root.
          4. A Local minimum point would have a derivative function equaling zero which would cause division by zero, so we should check the value of the derivative first before calculating the next approximation of the root to know when to stop before causing division by zero error.
          5. Newton Raphson doesn't converge Quadratically when it converges where the root is a multiple root. We can try to guess or specify the multiplicity of the root and use modified Newton Raphson which will speed the algorithm.
       2. **Secant:**
          1. It requires two guesses instead of one guess as in Newton Raphson. To overcome this problem, we can use the modified secant method which will require only one initial guess if the delta is selected properly because if it's too small it can cause subtractive cancelation in the denominator and if it's too big it can cause the method to slow or even diverge in the worst case scenario.
2. **Sample Runs and Analytics:**

## **Bisection**

* ***Test One***

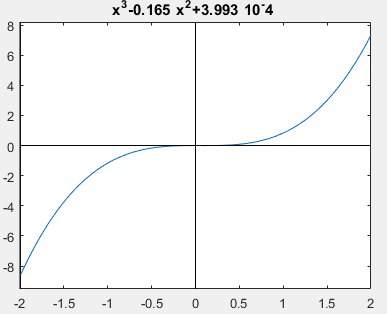
**Inputs:**

* Equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

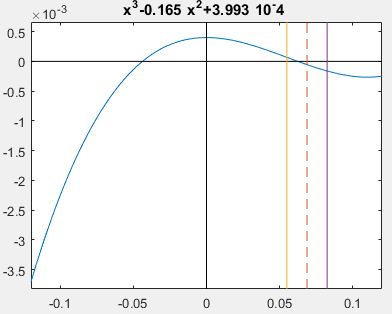
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

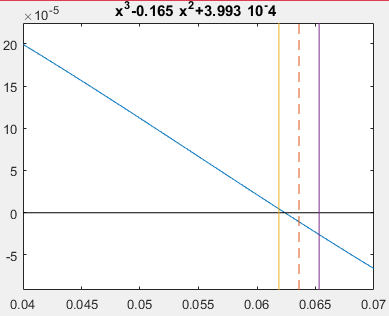
Function plotting & step simulation.



Function plotting



Bisection 3rd iteration



Bisection 6th iteration

* ***Test Two***

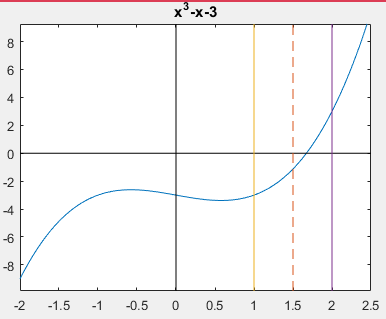
**Inputs:**

* Equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

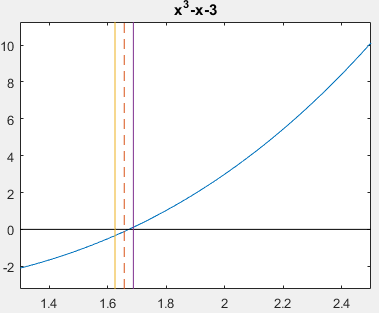
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

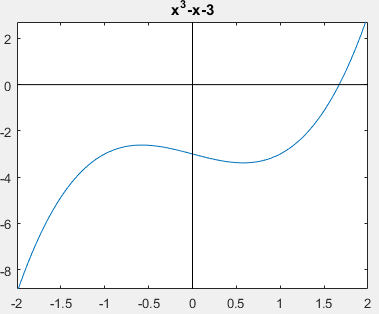
Function plotting & step simulation.



Bisection 3rd iteration



Bisection 6th iteration



Function plotting

* ***Test Three***

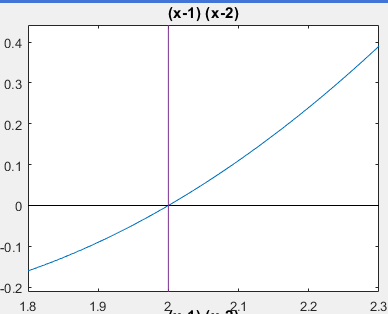
**Inputs:**

* Equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

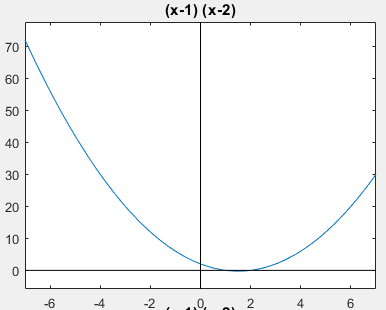
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

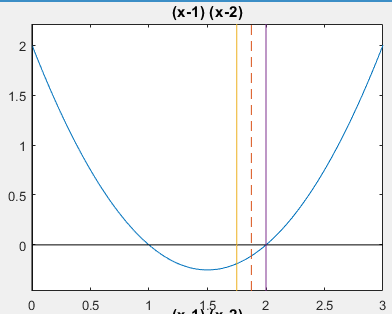
Function plotting & step simulation.



17th (last) iteration



Function plotting



3rd iteration

## **False-Position**

* **First Test Case**

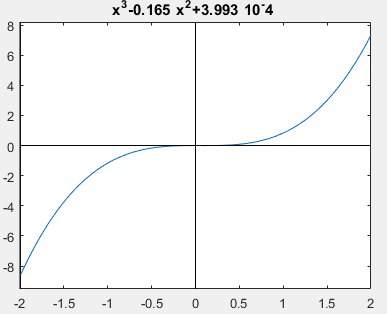
**Inputs:**

* Initial guesses:
* Precision: 0.00001
* Max iterations:

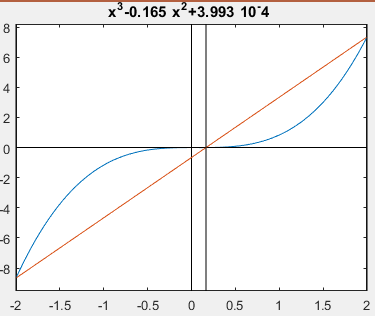
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

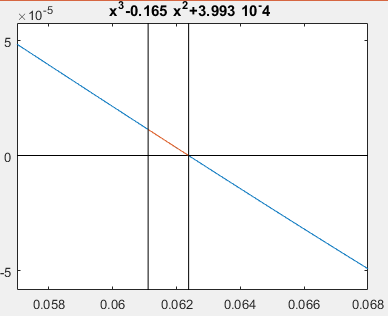
Function plotting & step simulation.



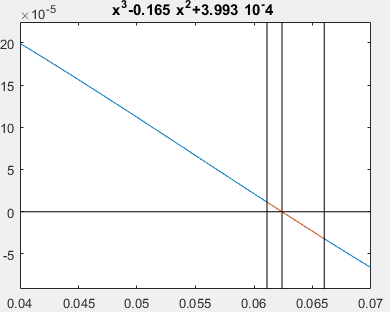
Function plotting



False-Position 1st iteration



False-Position 5th (Last) iteration



False-Position 3th iteration

* **Second Test Case**

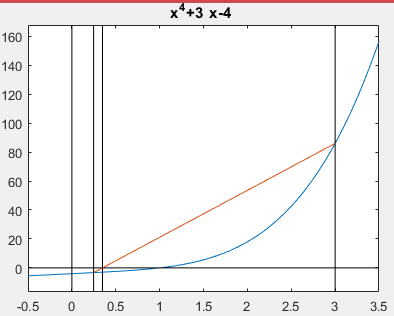
**Inputs:**

* Initial guesses:
* Precision: 0.00001
* Max iterations:

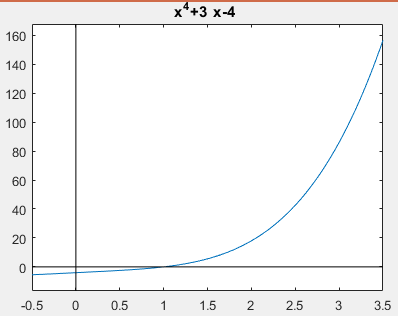
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

Function plotting & step simulation.

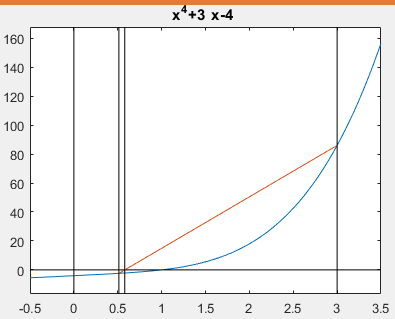


False-Position 3th iteration



Function plotting

**Node:** The difference between the plot for the 3rd and the 6th is very small because this method is very slow for this test case.



False-Position 6th iteration

* **Third Test Case**

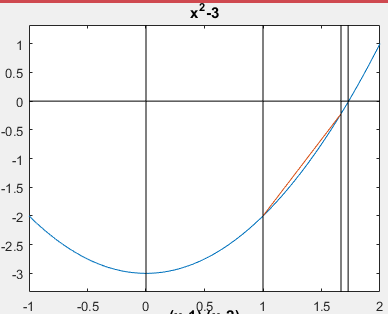
**Inputs:**

* Equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

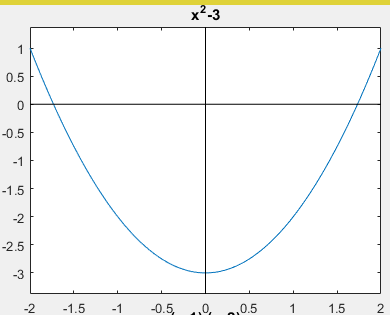
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

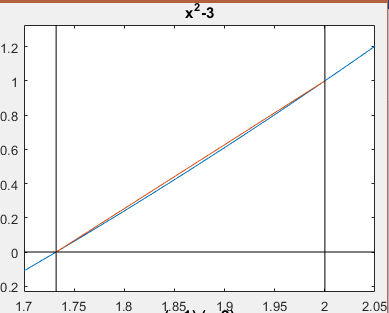
Function plotting & step simulation.



False-Position 1st iteration



Function plotting



False-Position 6th (last) iteration

## **Fixed Point**

* ***First Test***

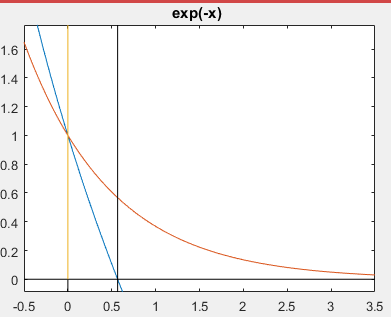
**Inputs:**

* Initial guesses:
* Precision: 0.00001
* Max iterations:

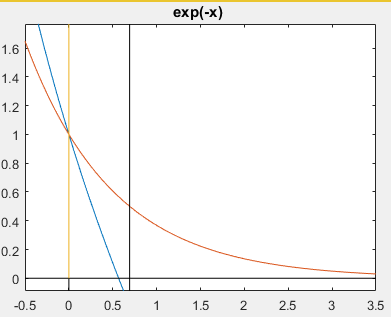
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

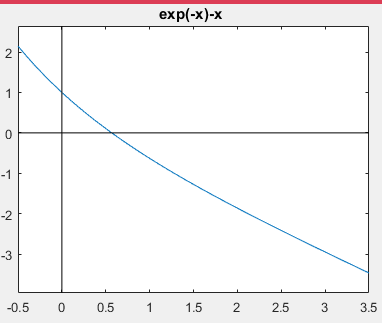
Function plotting & step simulation.



12th iteration plotting



3rd iteration plotting



Function Plotting

* ***Second Test***

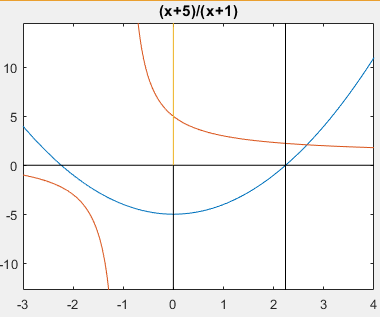
**Inputs:**

* Initial guesses:
* Precision: 0.00001
* Max iterations:

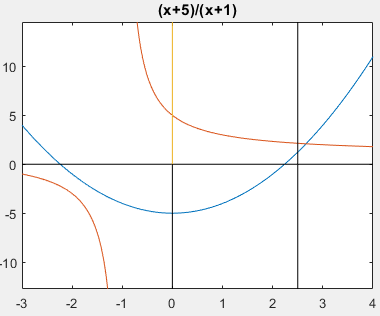
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

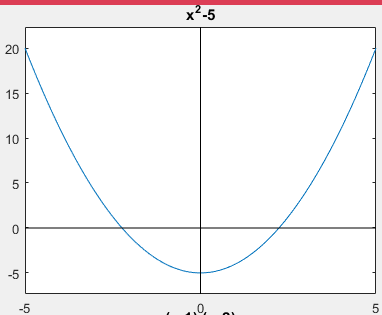
Function plotting & step simulation.



15th (last) iteration plotting



3rd iteration plotting



Function Plotting

* ***Third Test***

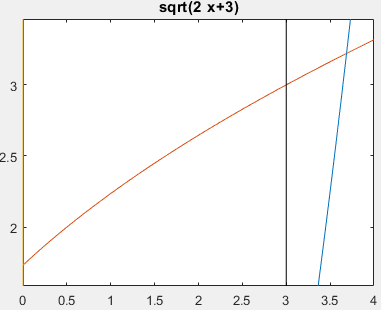
**Inputs:**

* Equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

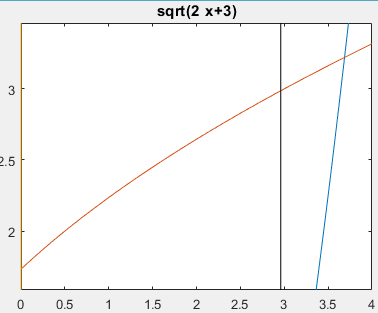
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

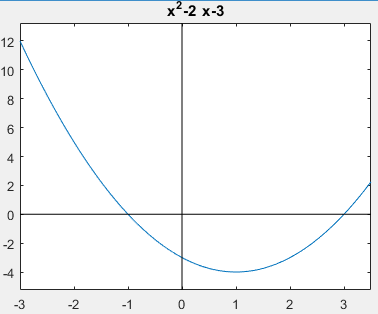
Function plotting & step simulation.



12th (last) iteration plotting



3rd iteration plotting



Function Plotting

## **Newton-Raphson**

* ***First Test***

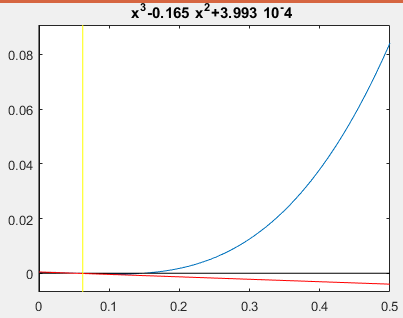
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

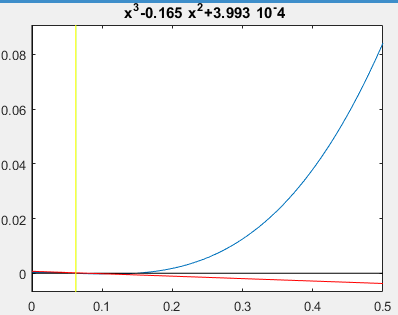
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

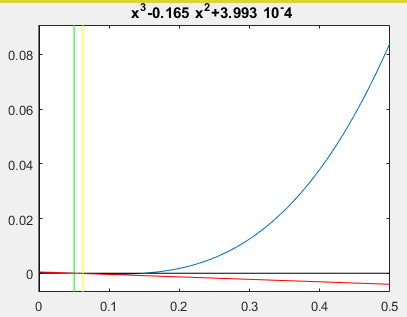
Function plotting & step simulation.



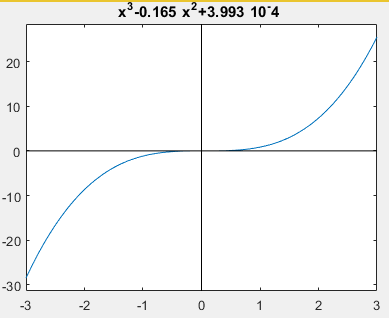
Third (last) iteration



Second iteration



First iteration



Function plotting

* ***Second Test***

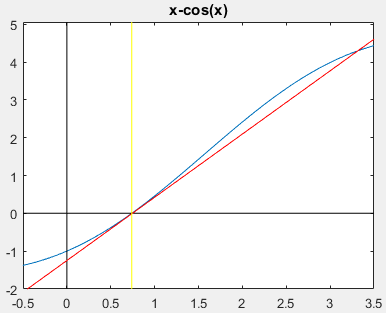
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

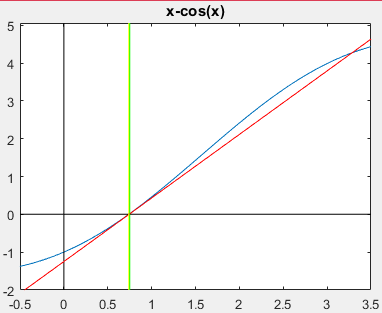
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

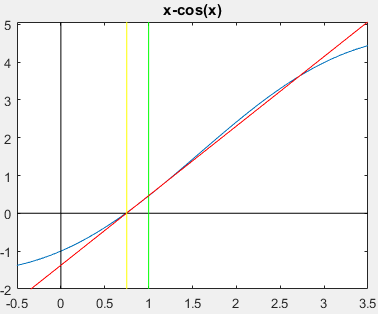
Function plotting & step simulation.



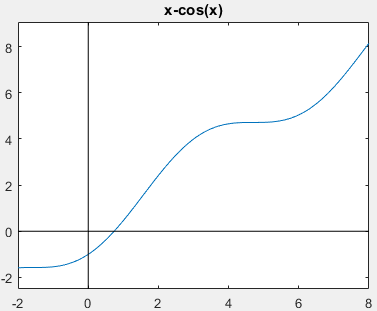
Fourth (last) iteration



Second iteration



First iteration



Function plotting

* ***Third Test***

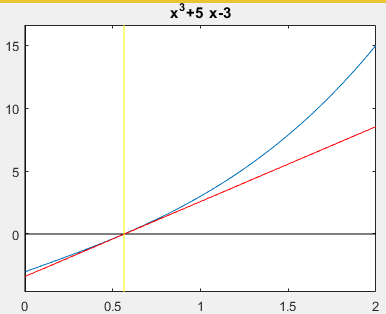
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

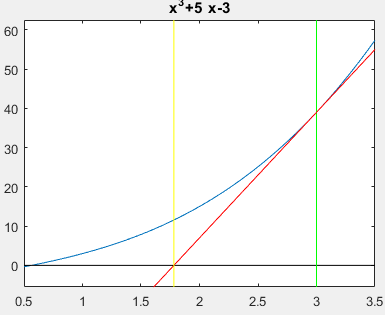
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

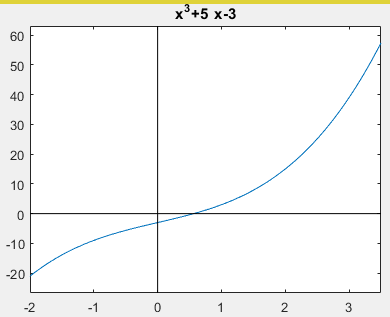
Function plotting & step simulation.



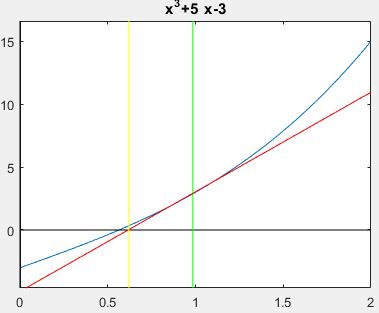
Sixth iteration



First iteration



Function plotting



Third iteration

## **Secant Method**

* ***First Test***

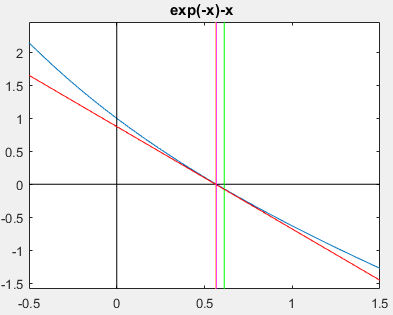
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

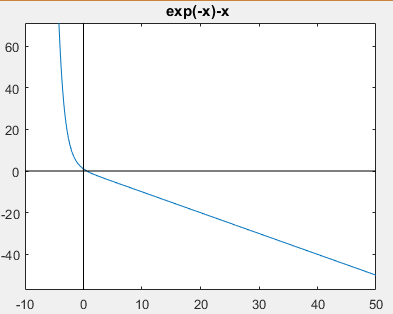
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

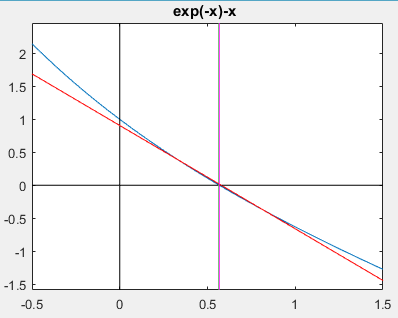
Function plotting & step simulation.



3rd iteration plotting



Function plotting



4th (Last) iteration plotting

* ***Second Test***

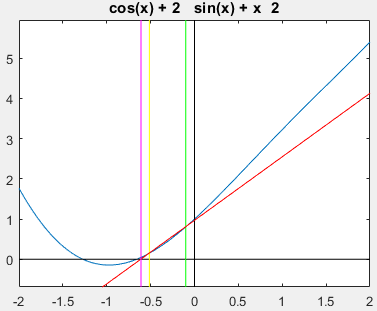
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

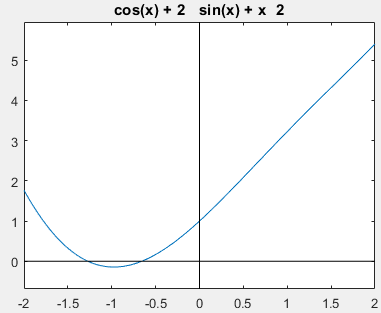
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

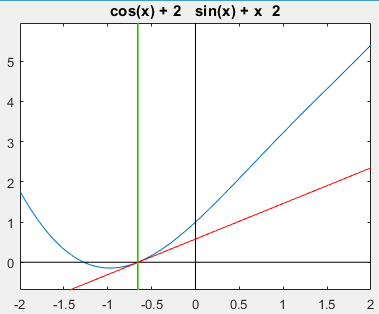
Function plotting & step simulation.



Second iteration plotting



Function plotting



Fifth (Last) iteration plotting

* ***Third Test***

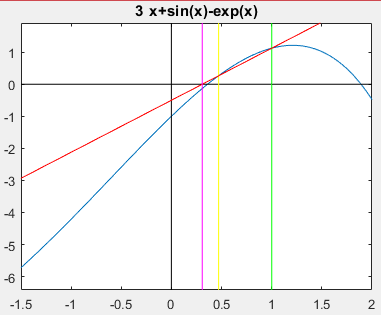
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

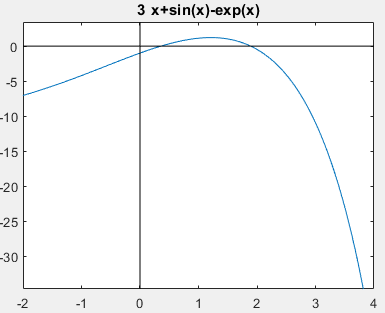
**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

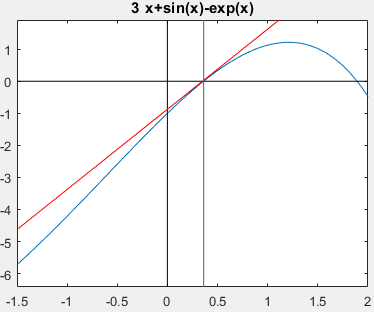
Function plotting & step simulation.



Second iteration plotting



Function plotting



Fifth (Last) iteration plotting

## **Birge Vieta:**

* ***First Test***

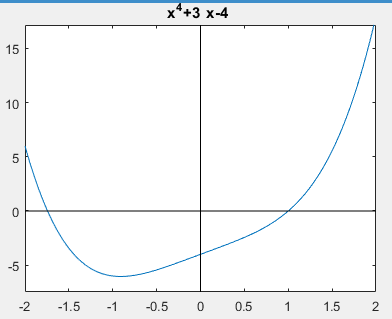
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

Function plotting.



Function plotting

* ***Second Test***

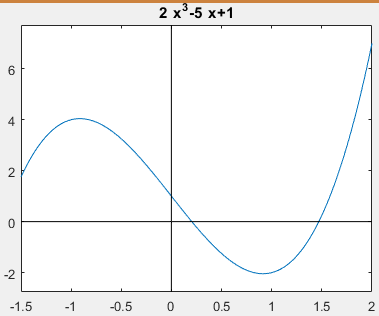
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

Function plotting.



Function plotting

* ***Third Test***

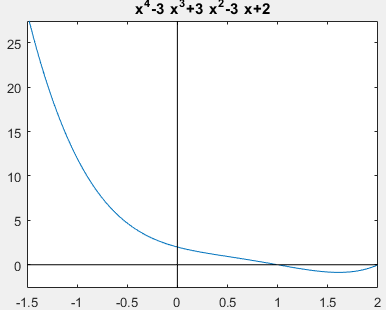
**Inputs:**

* Input equation:
* Initial guesses:
* Precision: 0.00001
* Max iterations:

**Outputs:**

* Root:
* Time:
* Iterations:
* Precision:

Function plotting.



Function plotting

## **All Methods:**

* **Test 1**

**Inputs:**

Equation:

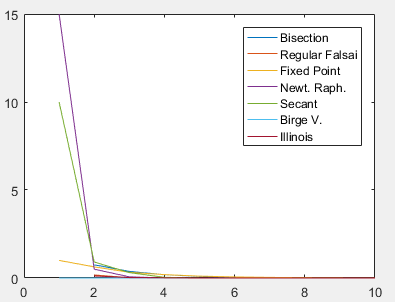
Bisection interval:

False-Position interval:

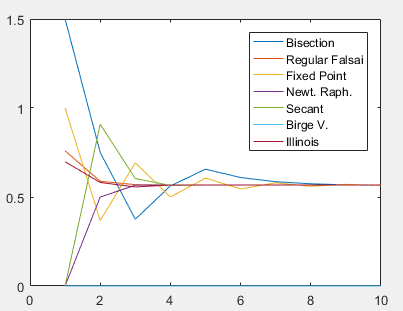
Fixed point initial guess = 0

Newton’s Raphson initial guess =

Secant initial guess =



Absolute error/iterations



Root/iterations

* **Test 2**

**Inputs:**

Equation:

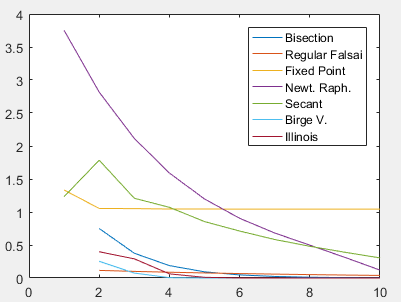
Bisection interval:

False-Position interval:

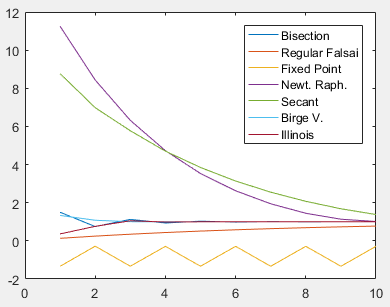
Fixed point initial guess = 0

Newton’s Raphson initial guess =

Secant initial guess =



Absolute error/iterations



Root/iterations

**Note:** the function is a bad because in this case it will diverge.