FACULTY OF ENGINEERING

Computer and systems engineering department

JUNE 25, 2014

Numerical Analysis

Assignment One

PART TWO

PARTS ONE, TWO

PRESENTED BY:

Ebtehal El_Aaraby (3).

Amany Ibrahim Hassan (11).

Remon Hanna Wadie Youssef (23).

Sarah Moustafa Mohamed (25).

Shaimaa Mousa (28).

Tarek Khaled Ragab (32).

PART ONE

Objectives:

The aim of this assignment is to compare and analyze the behavior of numerical methods studied in class {Bisection, False-position, Fixed point, Newton-Raphson, Secant}.

Specification:

The program must contain the following features:

- An interactive GUI that enables the user to enter equations containing different functions such as: {poly, exp, cos, sin}. Reading from files must be available as well.
- Differentiation and Parsing is your task.
- A way to choose a method to solve the given equation.
- A way to enter the precision and the max number of iterations otherwise default values are used, Default Max Iterations = 50, Default Epsilon = 0.00001;
- The answer for the chosen method indicating the number of iterations, execution time, all iterations, approximate root, and precision.
- Compute the theoretical bound of the error for the methods.

Analysis and Description

NUMERICAL METHODS

Bisection Method:

Description:

It's a method to get the root of the given function by iterating to get xr which the average of the upper and the lower value and substitute in function if the result is 0 then it's the exact root of the equation else we calculate f(Xl) and f(Xr) if the multiplication less than 0 we change the upper value with xr else we change the lower value with it.

Pseudo Code:

```
Get xlower and xupper
//Check whether the root is one of them
If(f(xlower)==0)
Root = xlower
Else if(f(xupper)==0)
Root = xupper
//Make a while loop by the given iteration or when the bound
//of error less than the given epsilon
While(iteration <maxIteration&&error >epsilon)
//get xr
Xr = xlower+xupper/2
```

//Check if xr is the exact solution

If(f(xr)==0)

Root = xr

//If it isn't check whether f(xlower)*f(xr)<0 then xupper = xr

Else if(f(xlower)*f(xr)<0)

xupper = xr

//if f(xlower)*f(xr)>0 then xlower = xr

Else if(f(xlower)*f(xr)>0)

xlower = xr

End while.

The theoretical bound of error can be calculated using this formula ess = L0/(2^k*xlower) , where k is the max number of iteration.

Fixed Point :

Description:

For the given function we get g(x) that is a function of x to get value of x that solve the equation so we iterate and substitute in g(x) by the previous value of x until we get the exact value or the approximating value of the root of the equation.

Pseudo code:

```
//get g(x) and the initial value of x (xo)
//for loop while number of iteration < max number of iteration and error is greater than epsilon
While(iteration <maxIteration&&error >epsilon)
//get xi
Xi= g(xi-1);
//Check if xi is the exact solution
If(f(xi)==0)
Root =xi
End while
```

Newton Raphson

Description:

This method calculate the root of the equation by iterating and substituting in the formula Xi= x(i-1) – f(x(i-1))/ f'(x(i-1)) until we get the exact or the approximating value of the root.

Pseudo code:

```
//get f(x) and the initial value of x (xo)
//for loop while number of iteration < max number of iteration and error is greater than epsilon
While(iteration <maxIteration&&error >epsilon)
//get xi
Xi= x(i-1) – f(x(i-1))/ f'(x(i-1));
//Check if xi is the exact solution
If(f(xi)==0)
Root =xi
End while
```

Newton Raphson Modified 1:

```
//get f(x) ,the initial value of x (xo)and m
//for loop while number of iteration < max number of iteration and error is greater than epsilon
While(iteration <maxIteration&&error >epsilon)
//get xi
Xi= x(i-1) – m*f(x(i-1))/ f'(x(i-1));
//Check if xi is the exact solution
If(f(xi)==0)
Root =xi
End while
```

Newton Raphson Modified 2:

```
//get f(x) and the initial value of x (xo)
//for loop while number of iteration < max number of iteration and error is greater than epsilon
While(iteration < maxIteration&&error >epsilon)
//get xi
Xi= x(i-1) – f(x(i-1))* f'(x(i-1))/ f'(x(i-1))^2- f(x(i-1))* f'(x(i-1));
//Check if xi is the exact solution
If(f(xi)==0)
Root =xi
End while
```

Secant

Description:

It is open method to find the root of given equation by evaluating Xi at every time and compare it by the previous value of it and compute relative error to get the root.
It is preferred to newton raphson method because newton raphson method needs to compute the derivatives.
But secant is used the newton raphson equation to find approximate value to the root regarding the following equation :

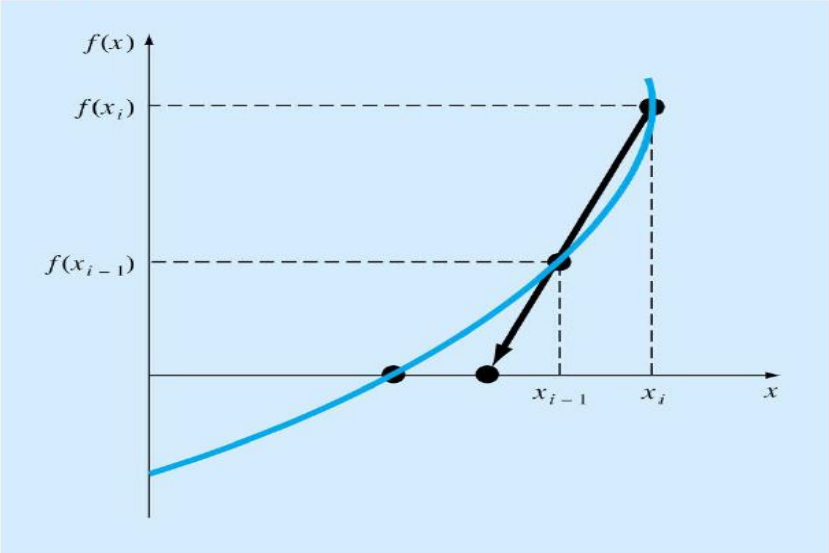
$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

Pseudo Code:

```
Secant( equation,MaxIter,epsilon,xl,xu )
output : arrayOfResult ,root
root =0;
curIter=0;
curEpsilon=100;
curtemp=100;
while(curEpsilon>epsilon )
 if(curIter<MaxIter)
 fl=compute(equation, xl);
 fu=compute(equation, xu);
 xDiff= xl - xu;
 root = xu -((fu*xDiff)/(fl-fu));
 curEpsilon=abs((root - xu)/ root);
  curtemp=curEpsilon*100;
  xl = xu;
```

```
xu = root;
curIter=curIter+1;
// add root & curEpsilon  to the output array
fx=compute(equation, root);
end
end
```

Secant Method



Advantages of secant method :

- Fast (slower than Newton)
- One function evaluation per iteration
- No knowledge of derivative is needed

Disadvantages of secant :

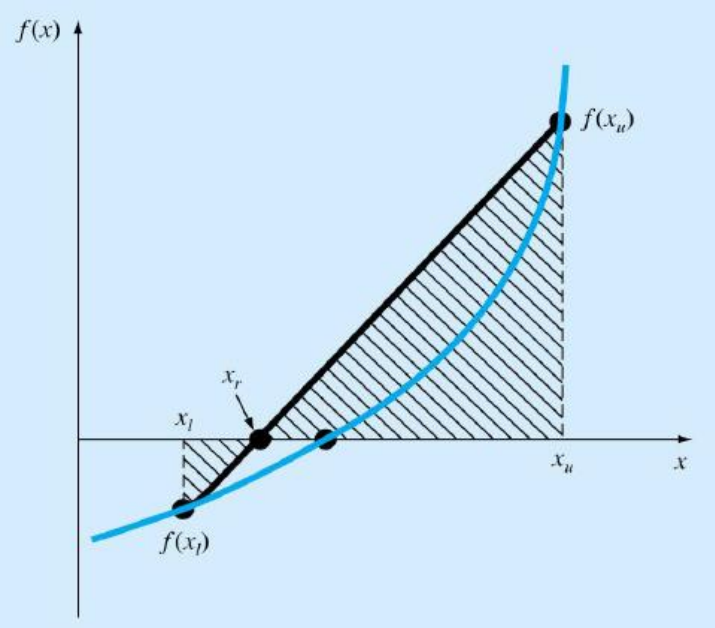
- May diverge
- Needs two initial points guess x0, x1 such that f(x0)- f(x1) is nonzero

False Position:

Description:

It is bracketing method to find the root of given equation by Approximation the solution by doing a linear interpolation between f(xu) and f(xl)

And find xr in every iteration by the following rule :



$$x_r = \frac{x_l f_u - x_u f_l}{f_u - f_l}$$

If f(xr)<0 then xl=xr && fl=f(xr)
If f(xr)>0 then xu=xr && fu=f(xr)
If f(xr)=0,get the exact root !

Pseudo Code:

```
function [ arr,xr,time ] =
FalseMethod( equation,MaxIter,epsilon ,low,high)
xr=low;
curIter=0;
curEpsilon=100;
fLowValue =compute(equation,low);
fUpperValue =compute(equation,high);
if  fUpperValue<0 &&fLowValue<0
   xr='Error in initial conditions';
return ;
else
while(curEpsilon>epsilon)
   if  curIter< MaxIter
   xTemp=xr;
   xr=(low*fUpperValue-high*fLowValue)/(fUpperValue-
fLowValue);
  fxrValue =compute(equation,xr);
   curEpsilon=abs((xr-xTemp)/xr);
   curTemp= curEpsilon*100;
   curIter=curIter+1;
// add xThird & error to the array of the output

   if fxrValue>0
      high=xr;
      fUpperValue=fxrValue;

   elseif fxrValue<0
      low=xr;
      fLowValue=fxrValue;
```

```
elseif fxrValue==0
   found the exact root
   return;
end
else
   return;
end
```

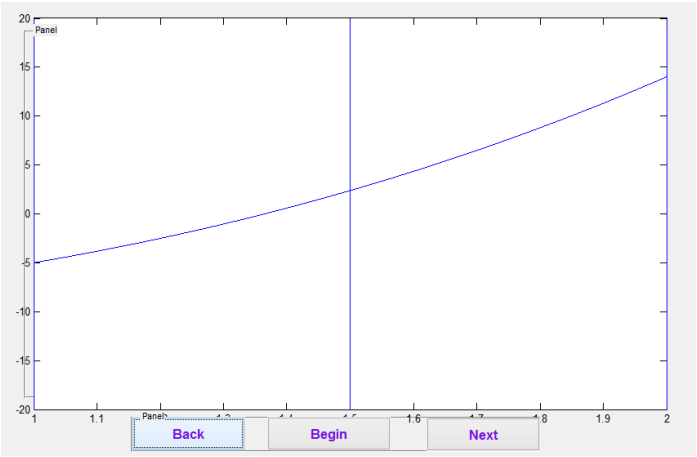
BONUS PART:

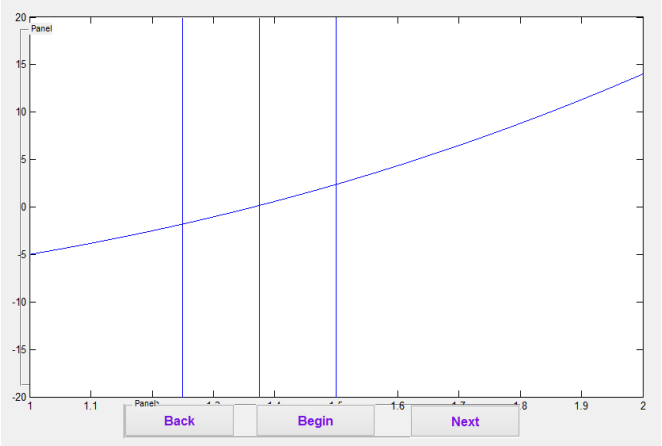
Description:

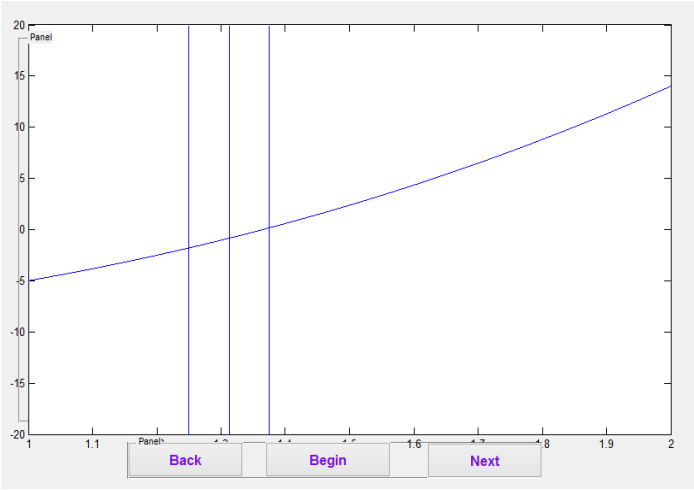
We plot the function for the Bisection method so when executing code of the Bisection method we save the result of each iteration for the values of x lower , x upper and xr then we plot every single step in that iteration and we give the user to undo or redo these steps.

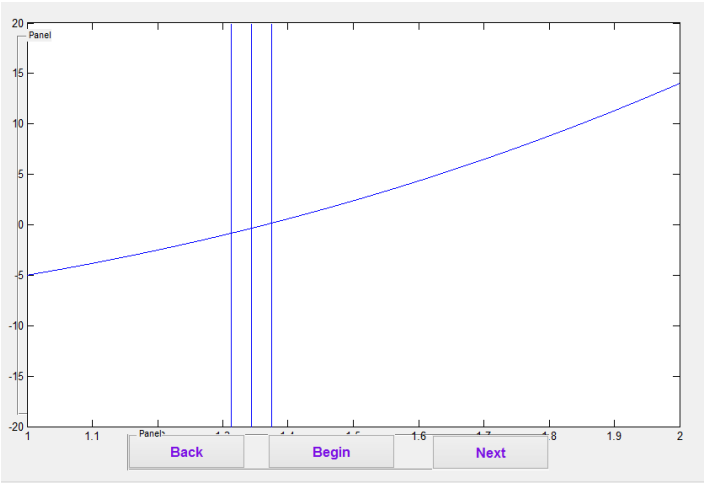
Sample Runs:

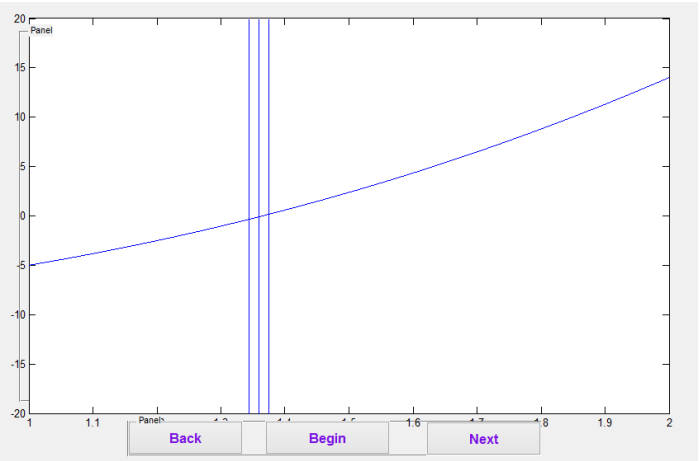
For the given equation((x)^3+4*(x)^2-10) , xl = 1 and xu = 2 the output is :

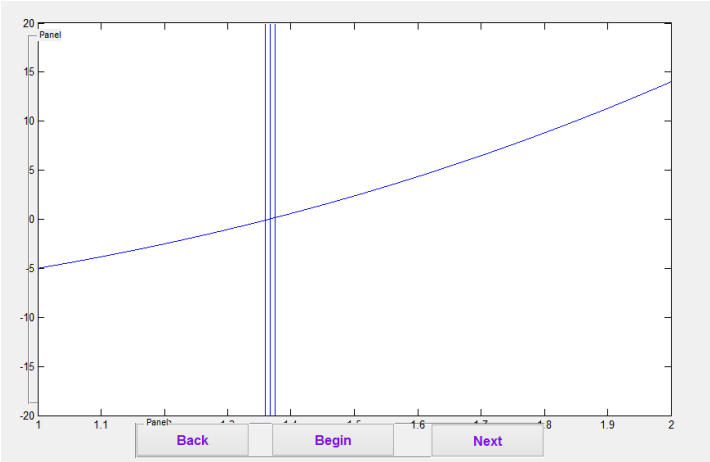


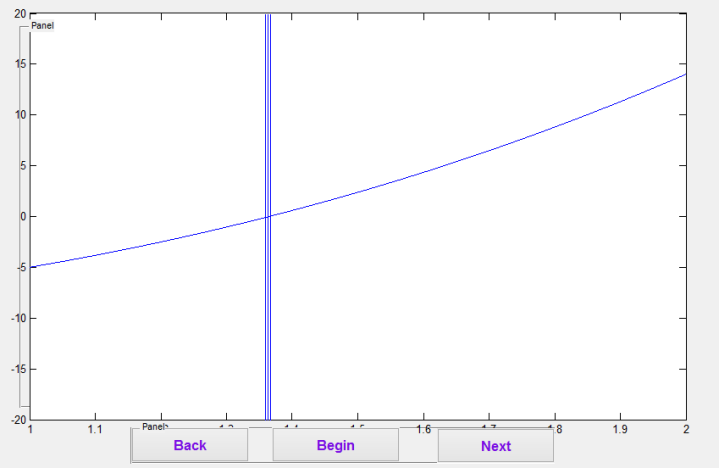


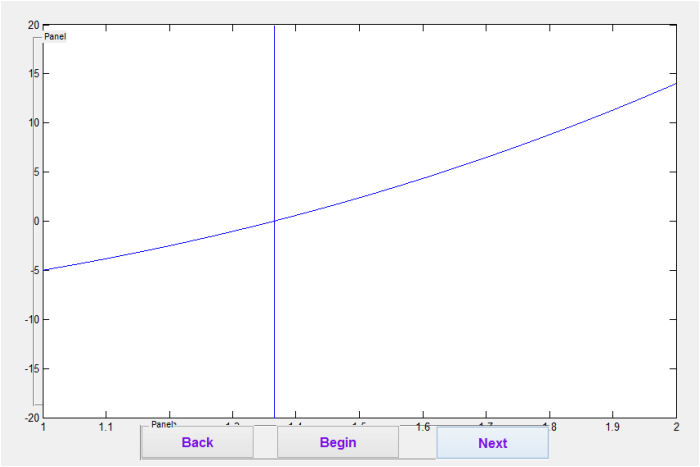

Panel
Back
Begin
Next


Panel
Back
Begin
Next


Panel
Back
Begin
Next


Panel
Back
Begin
Next

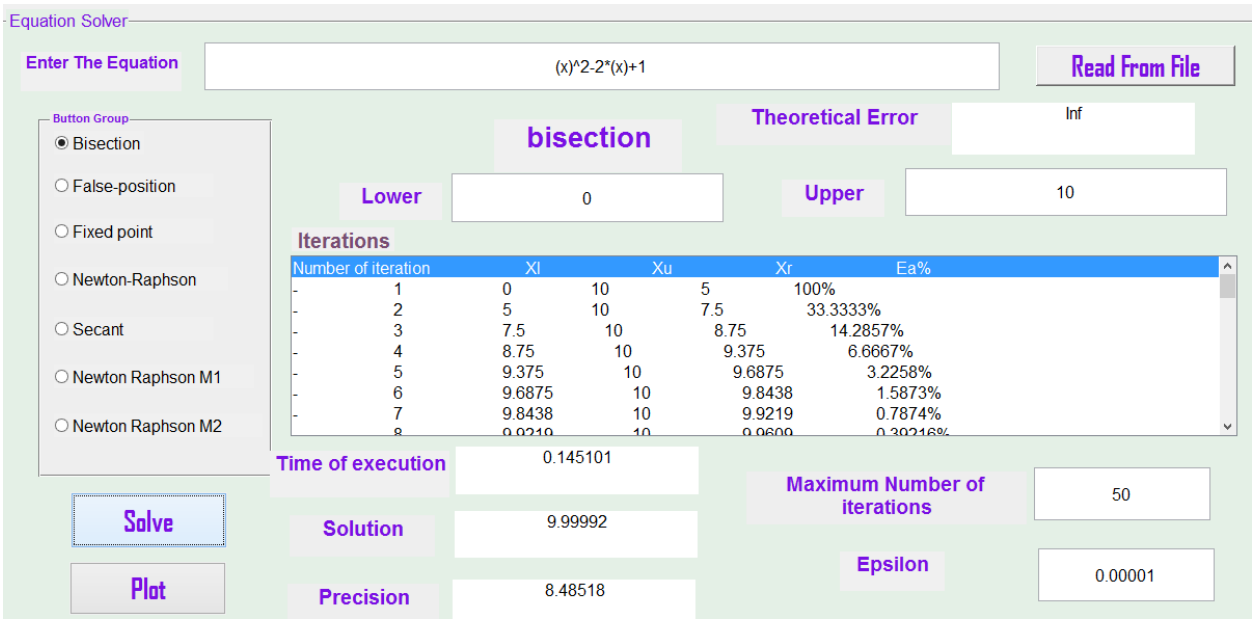

Panel
Back
Begin
Next

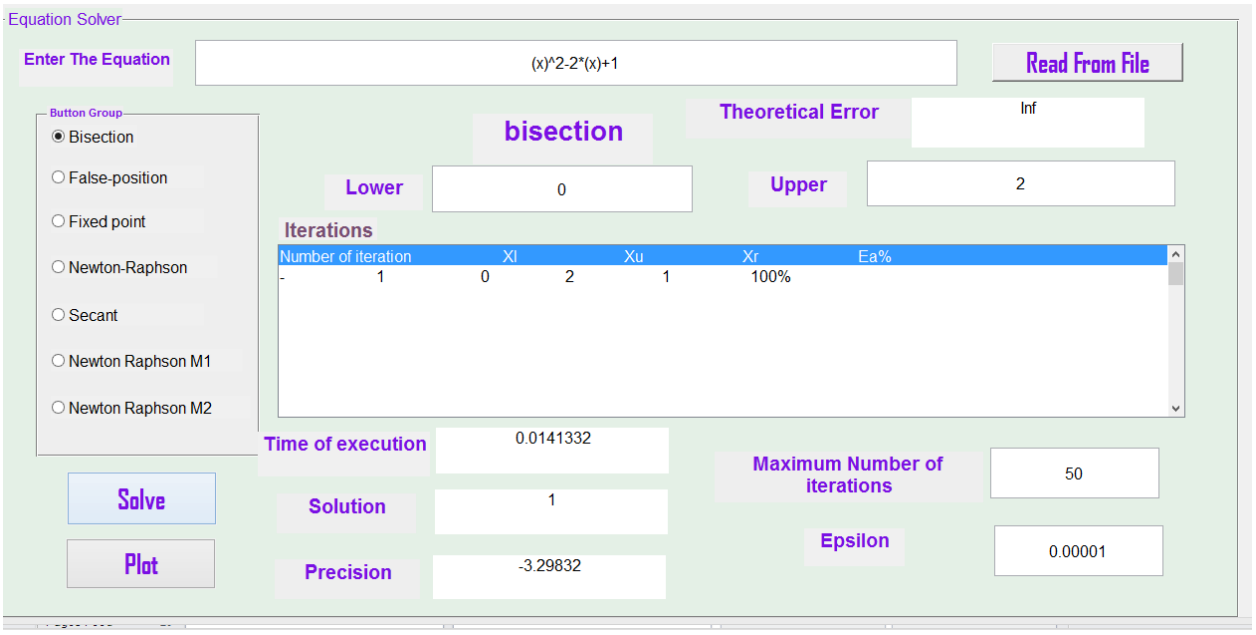

Panel
Back
Begin
Next

Problematic functions:

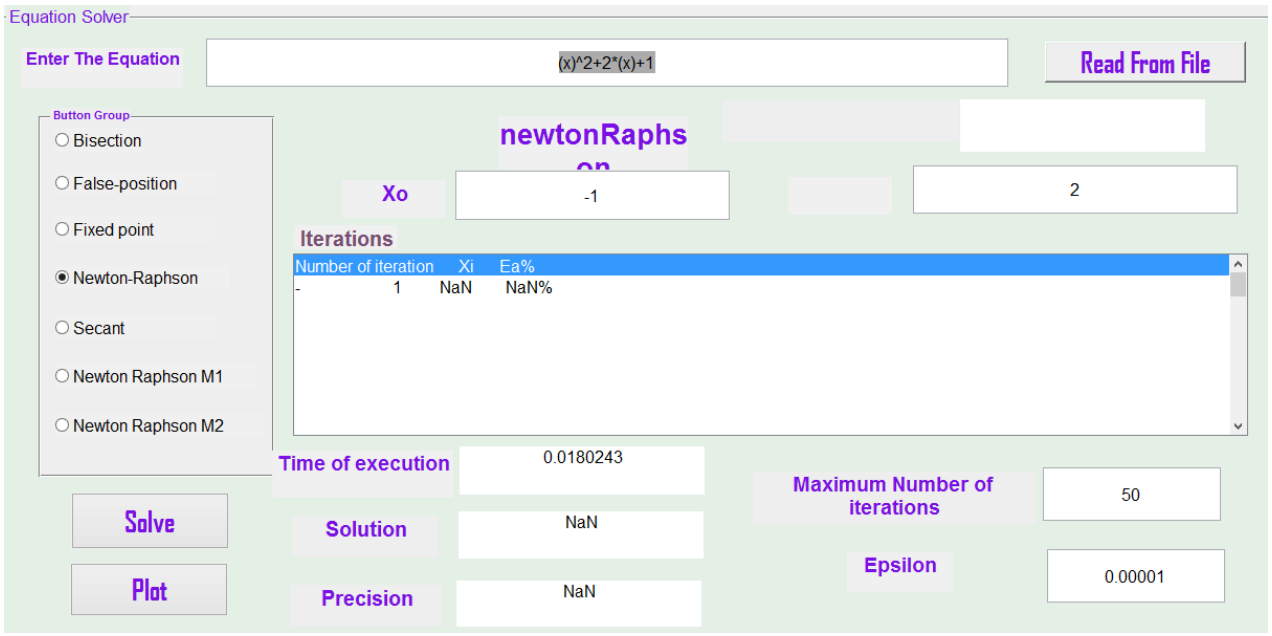
For each method there are some cases that make the method diverge and never reach the exact solution or even an approximation to it for example:

- For Bisection method: if the given interval is not suitable it gives a root but it's not necessary to be true as for the equation "(x)^2-2*(x)+1" the exact value is 1 it will get 1 if the interval from 0 to 2 but if we change it to any other interval it will diverge, the solution for it whether to choose a suitable interval or use the secant method which diverge less than the Bisection method.

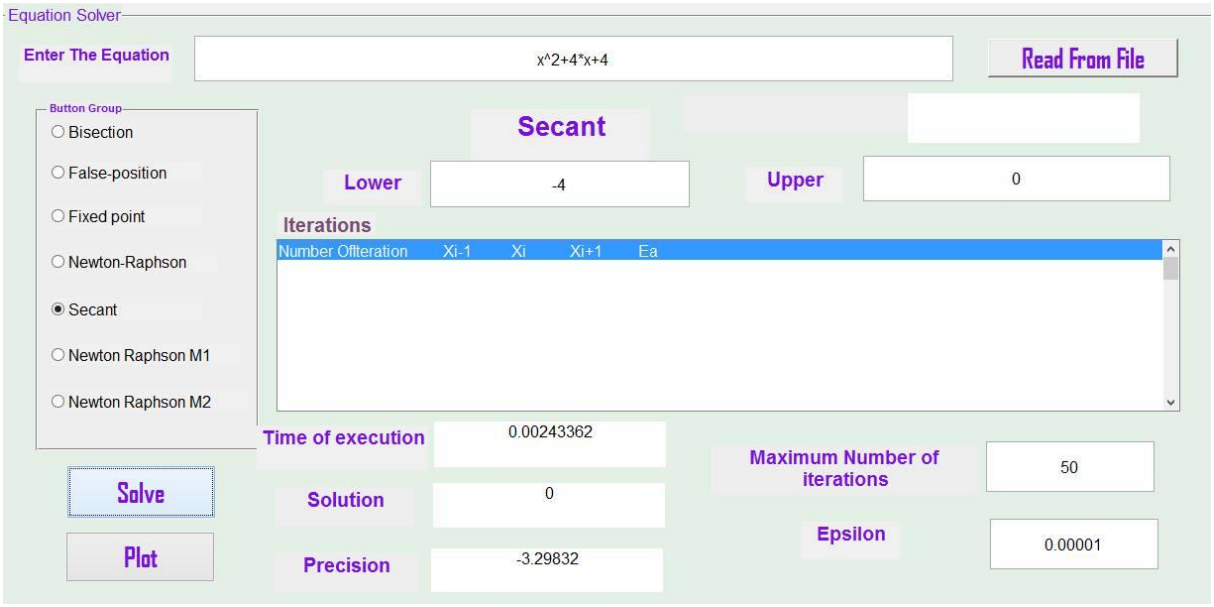


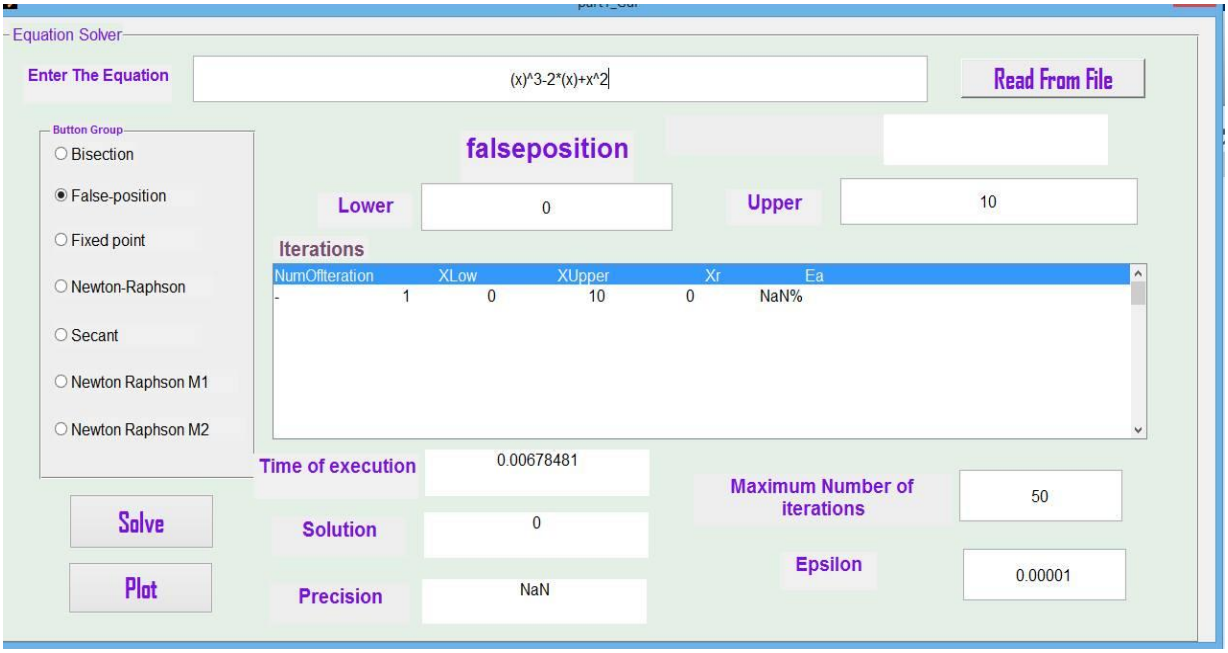


- For Newton Raphson: because the formula of it is divided by F'(x) if then it will lead to infinity and diverge as for" (x)^2+2*(x)+1" , the solution is to use either Newton Raphson Modified 1 or 2 they converge faster and have a little chance todiverge.



- For Secant and False Position: it will get NaN when the dominator becomes 0 as for the example “(x)^3-2*(x)+x^2” for false position and “(x)^2+4*(x)+4”.



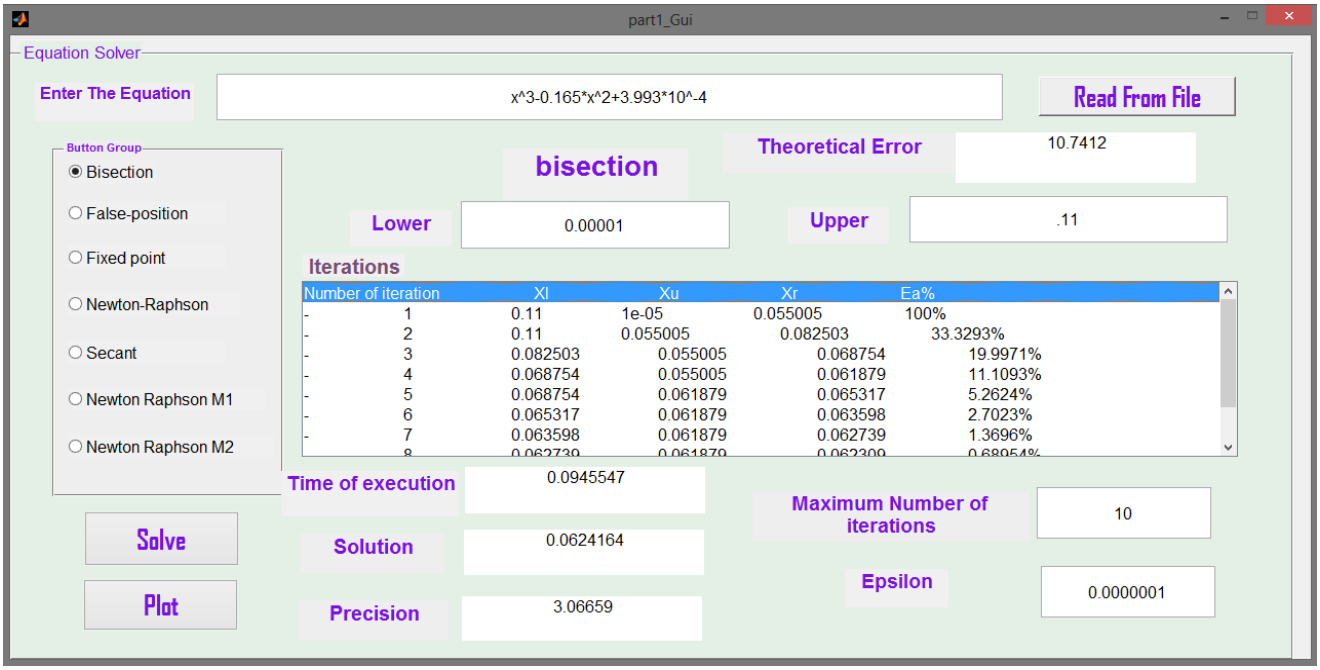


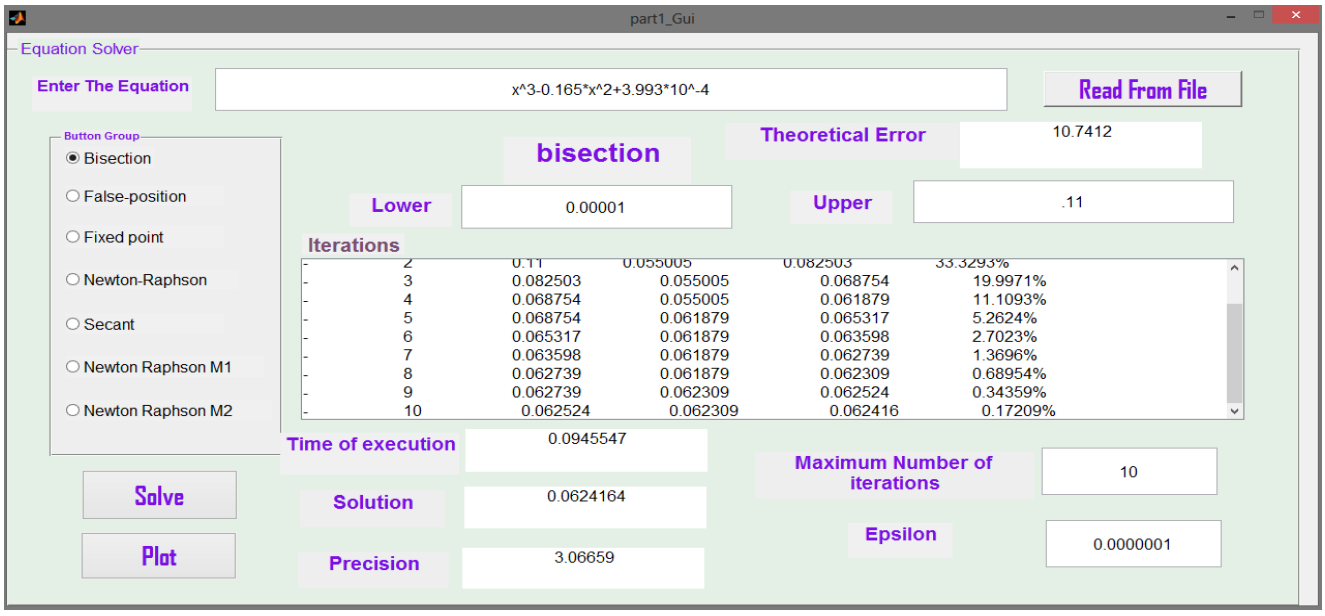
Analysis the behavior of all methods by different examples:

Example 1:

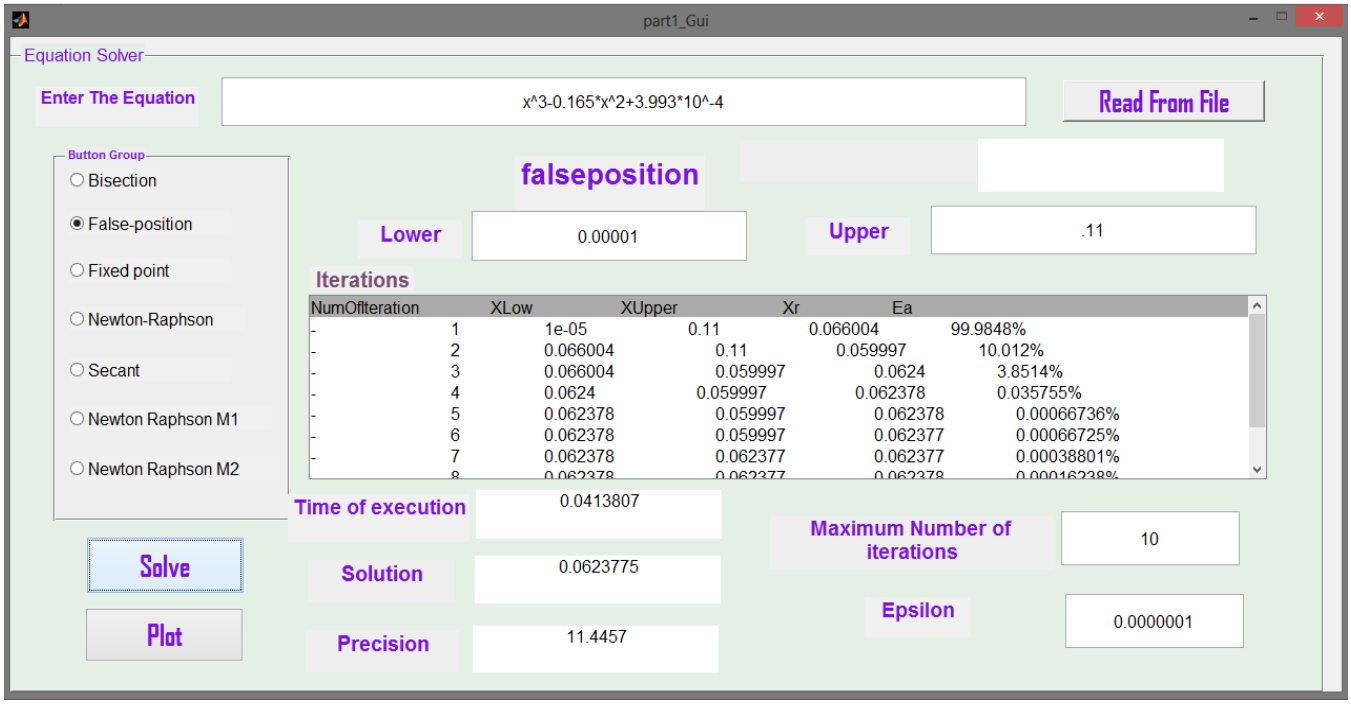
Equation x^3-0.165*x^2+3.993*10^-4

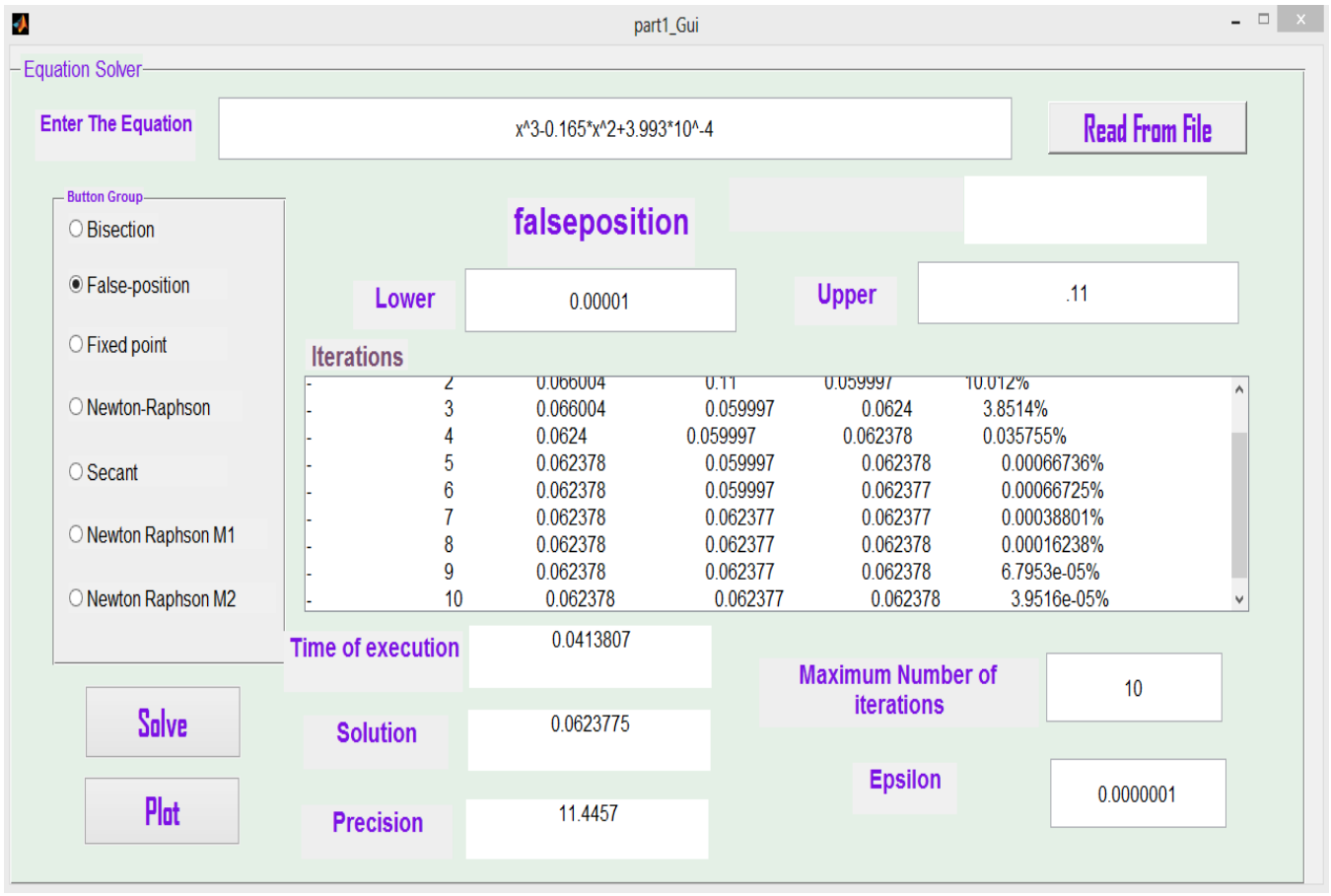
In bisectiom:



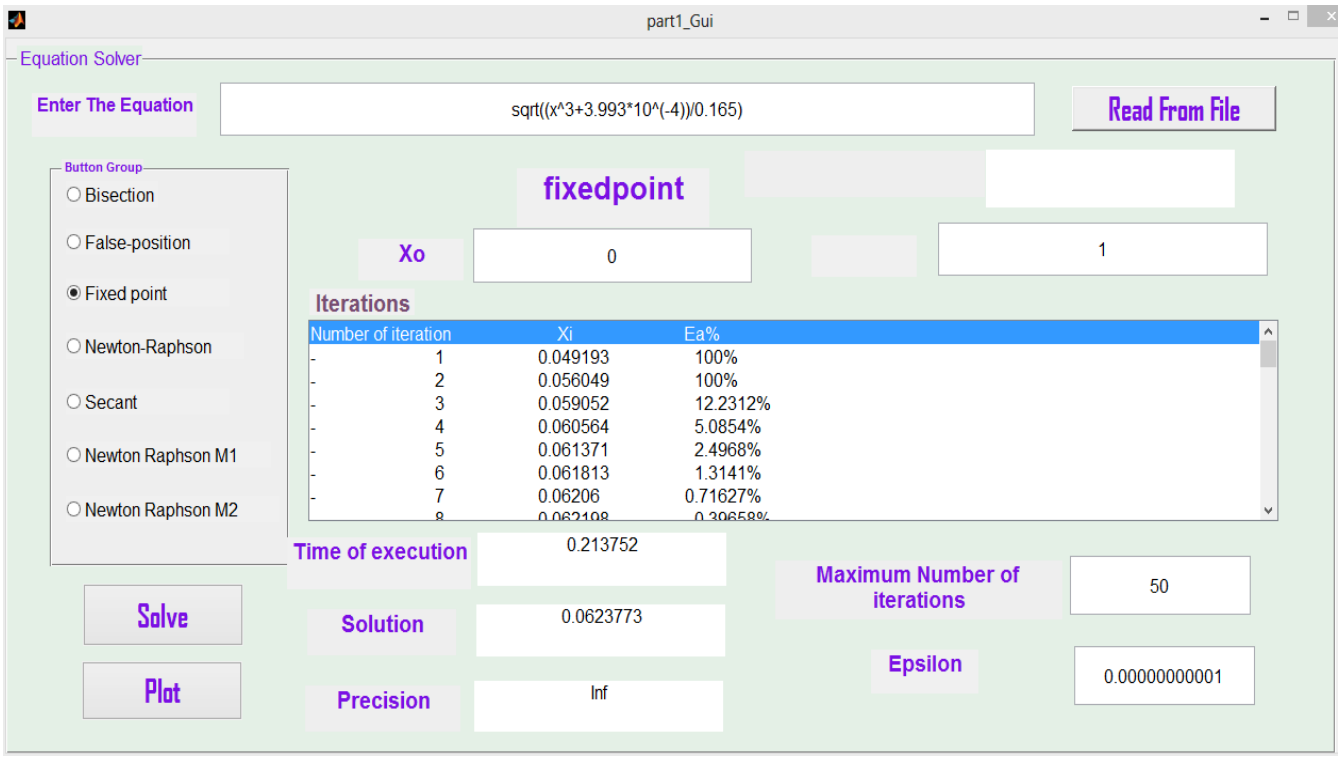


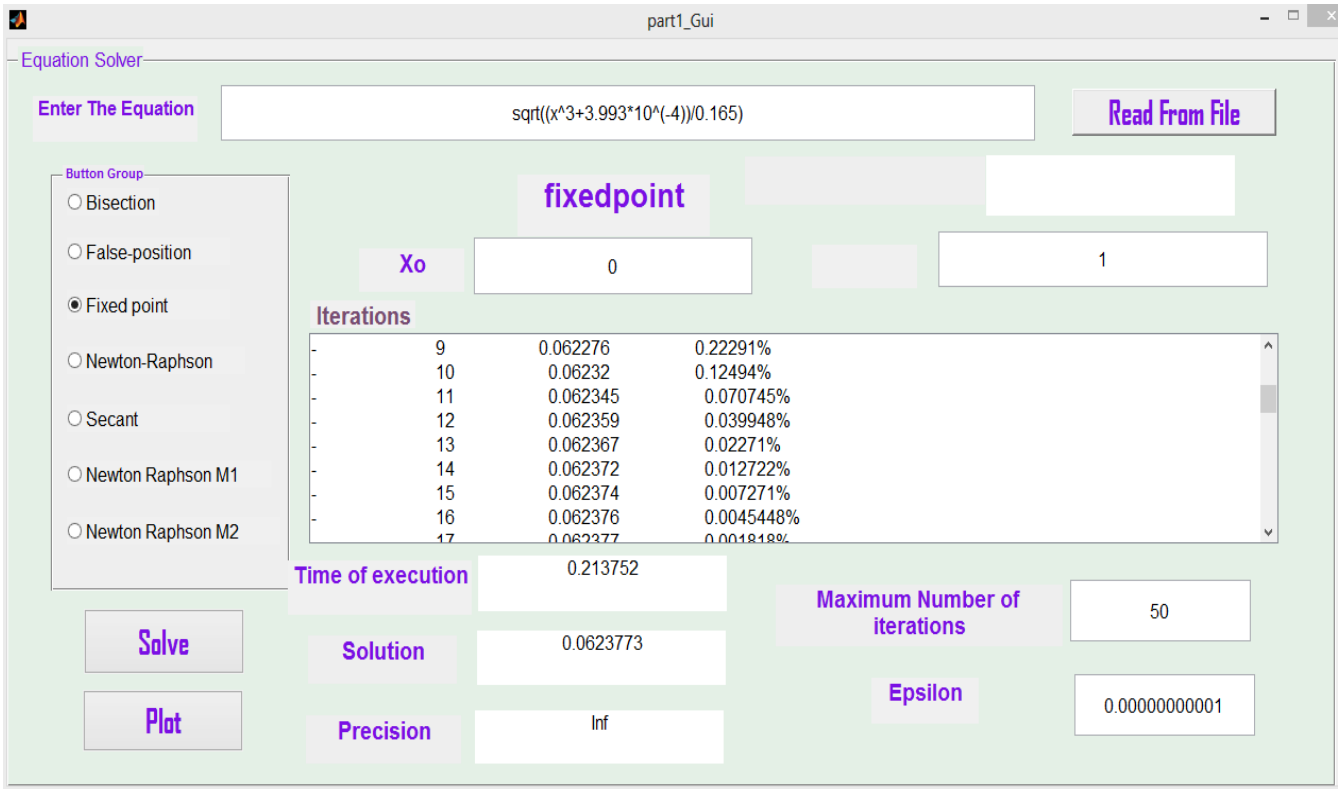
In False Position :

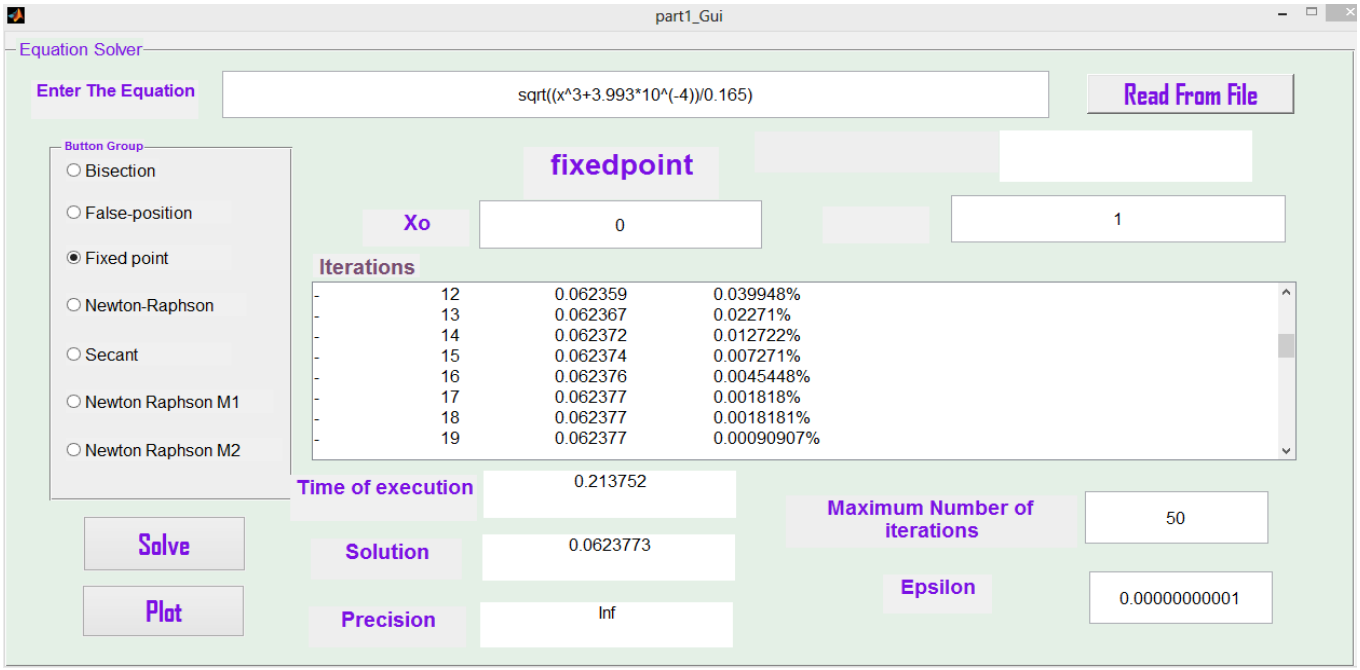




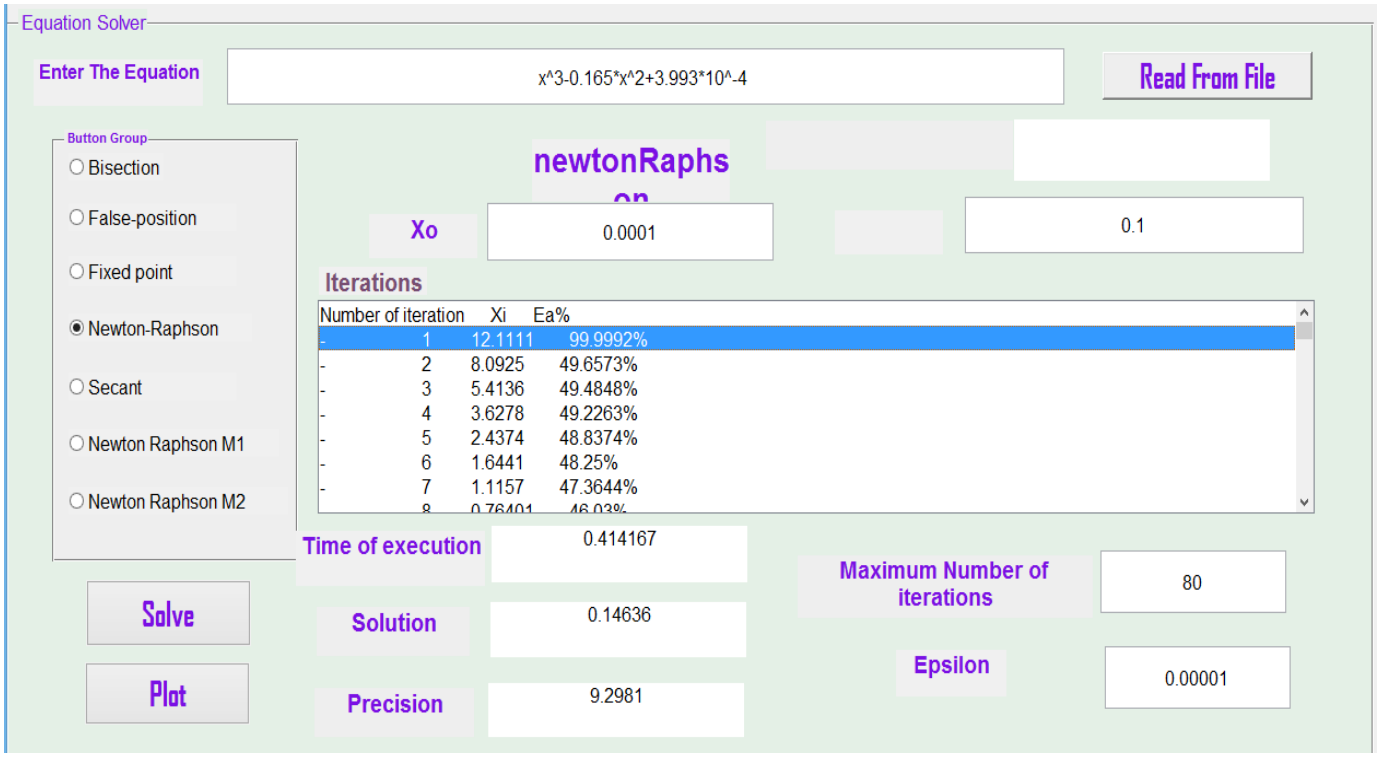
In Fixed Position:

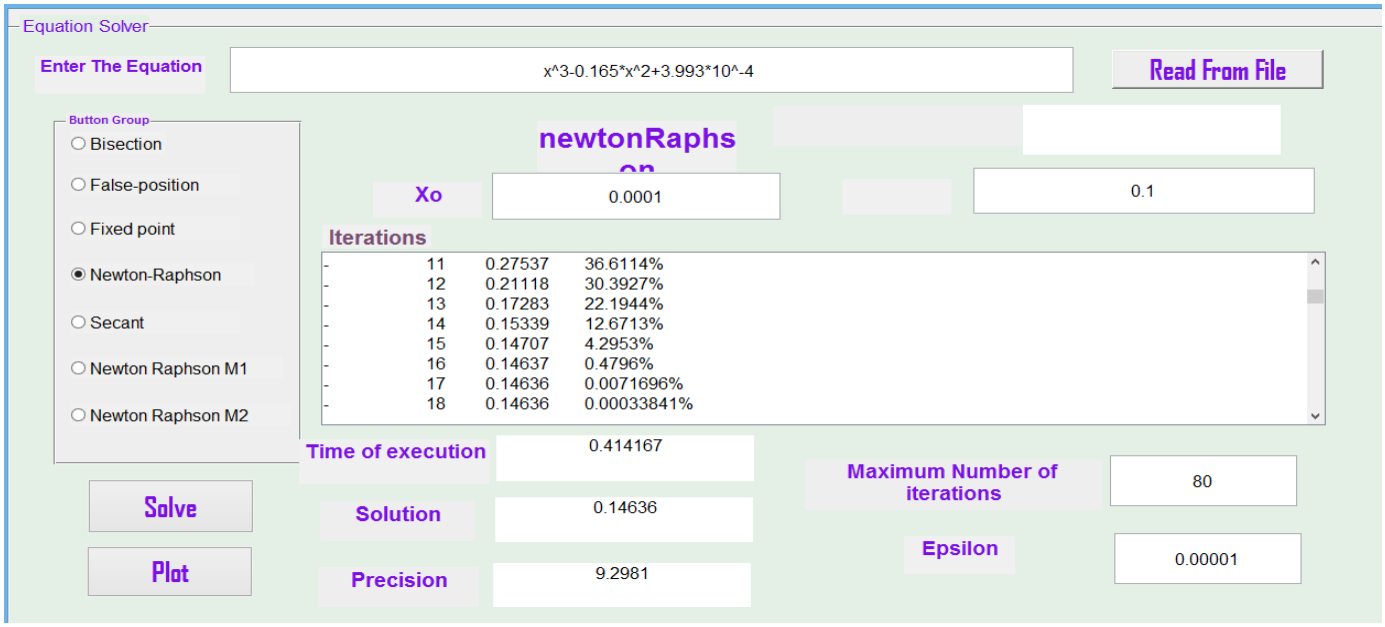




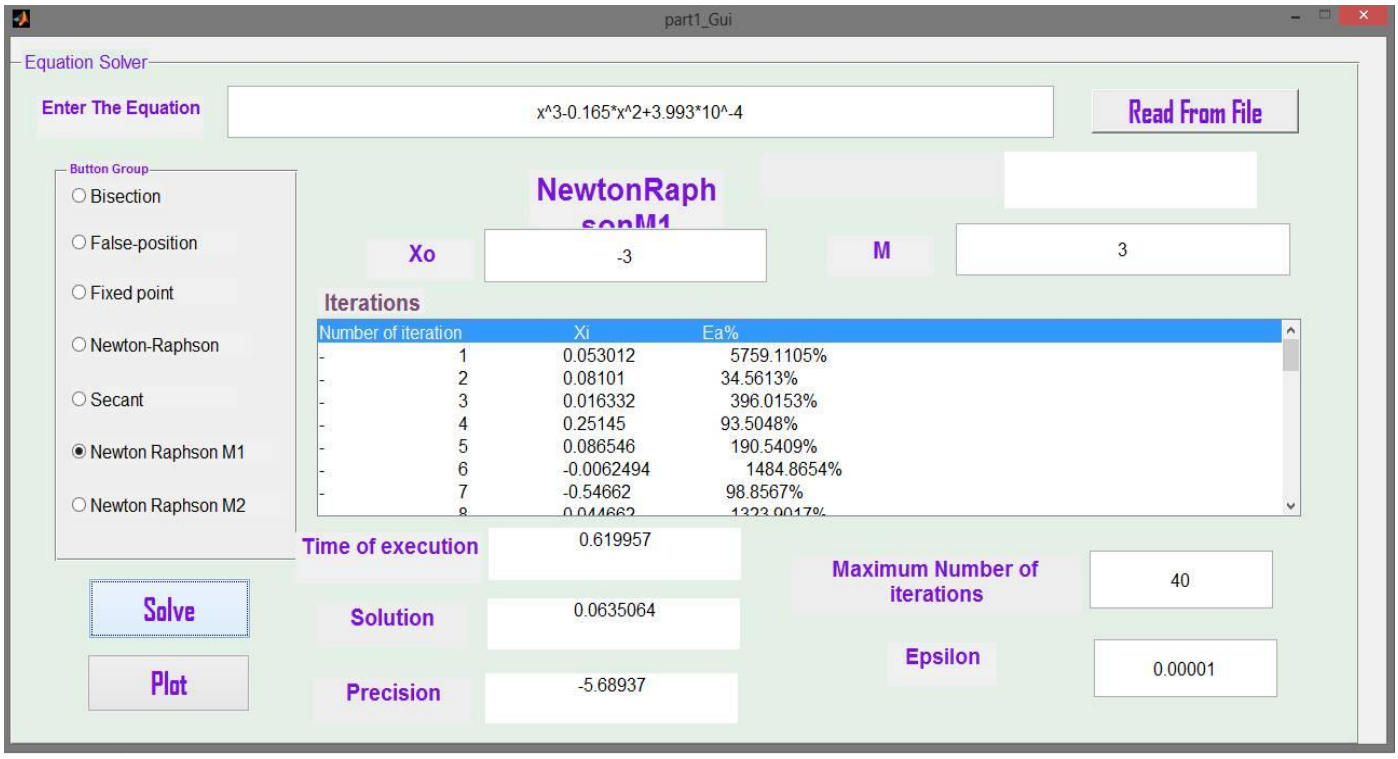


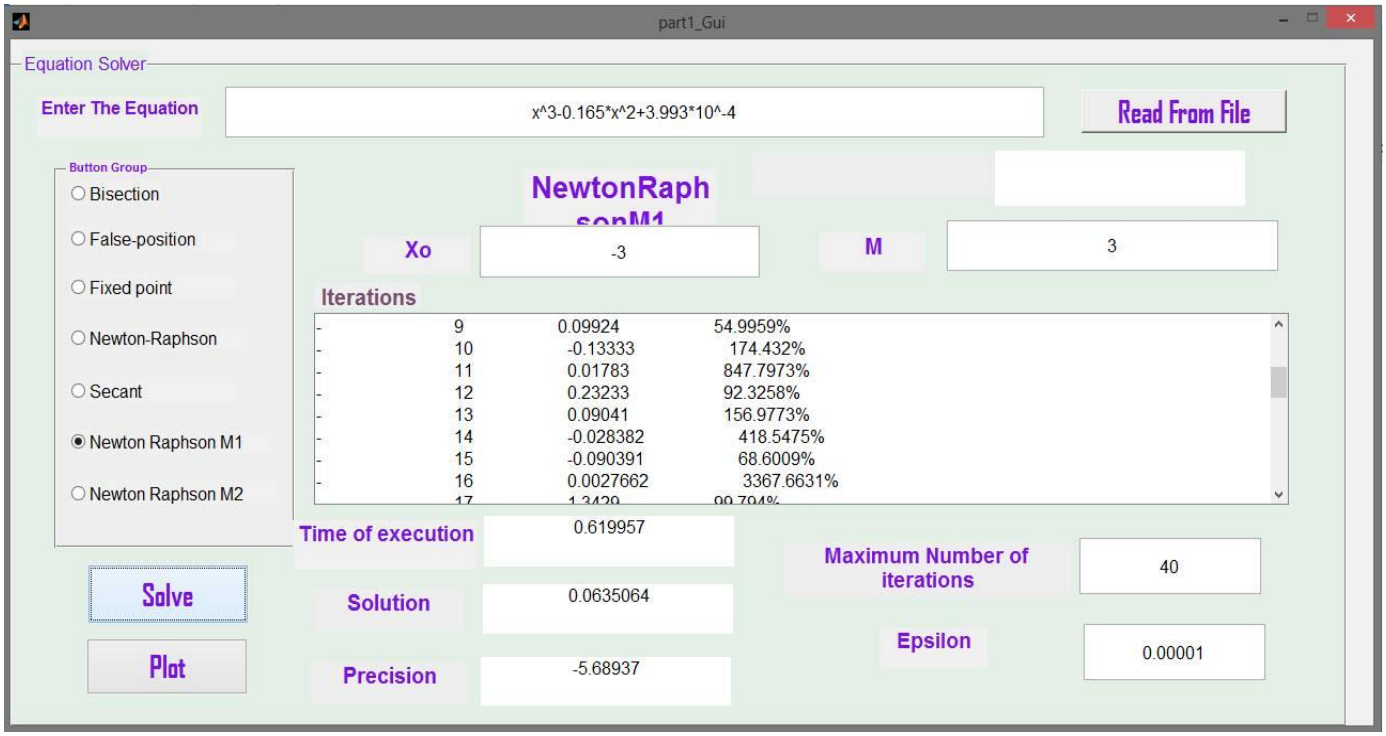
In Newton Raphson :

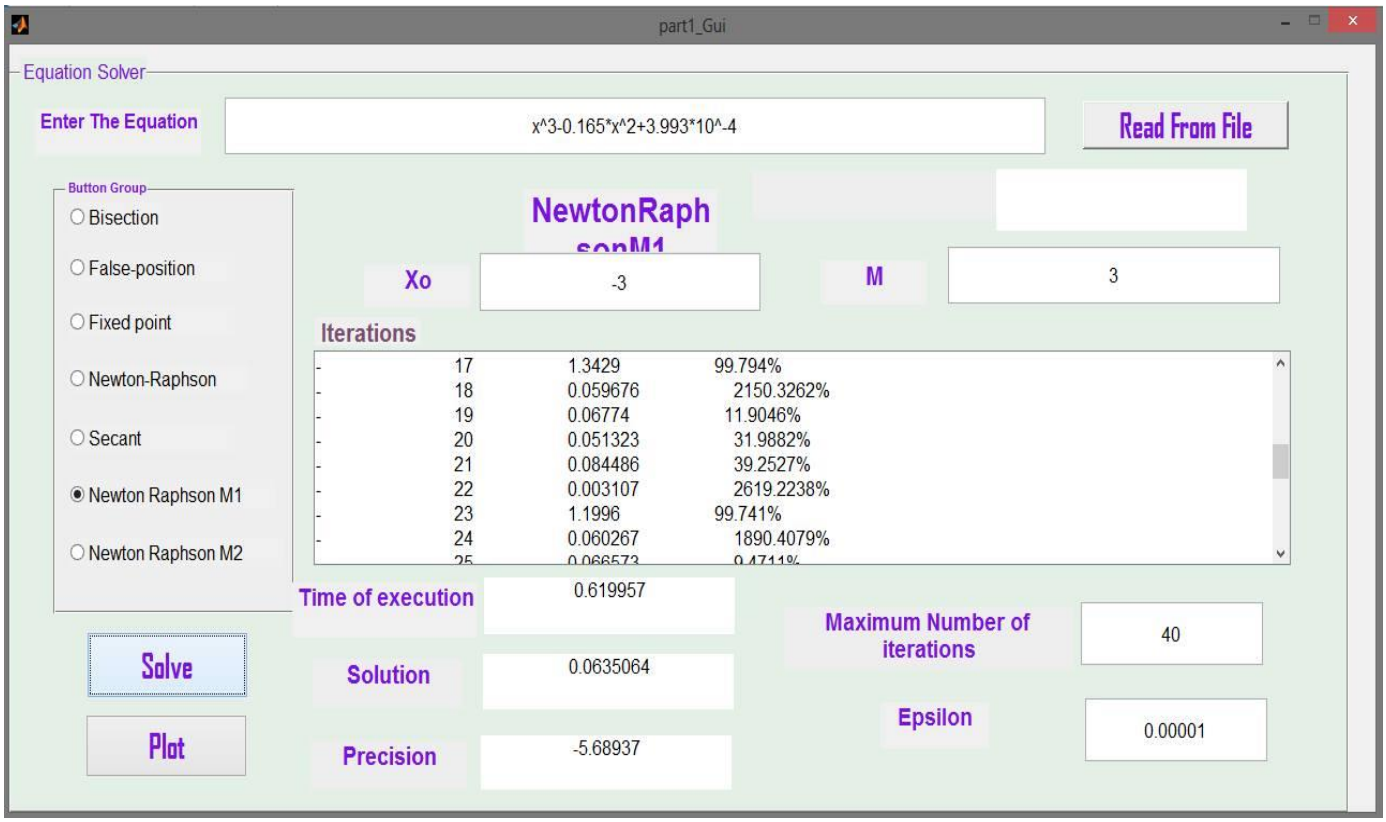


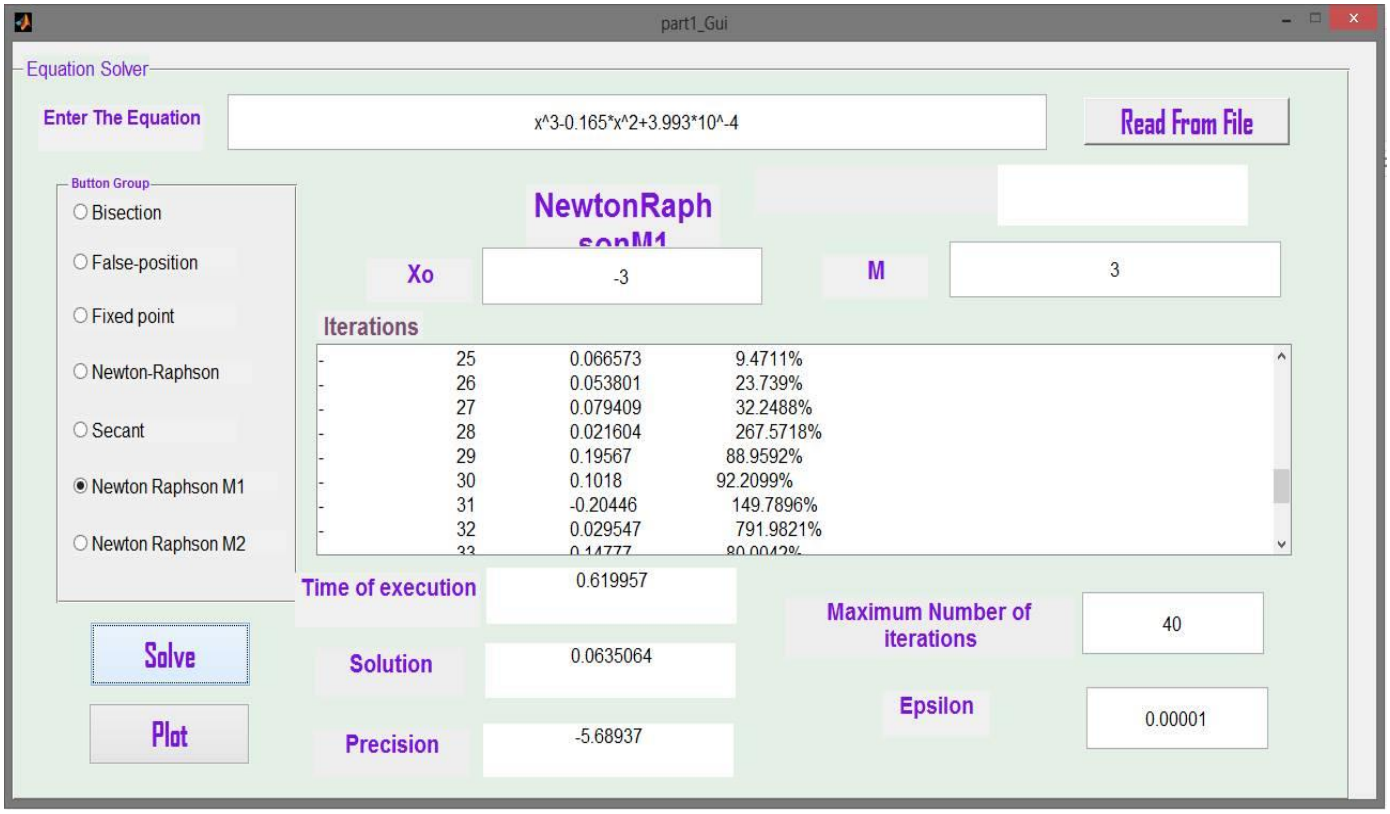


In Newton Raphson Modified 1 :

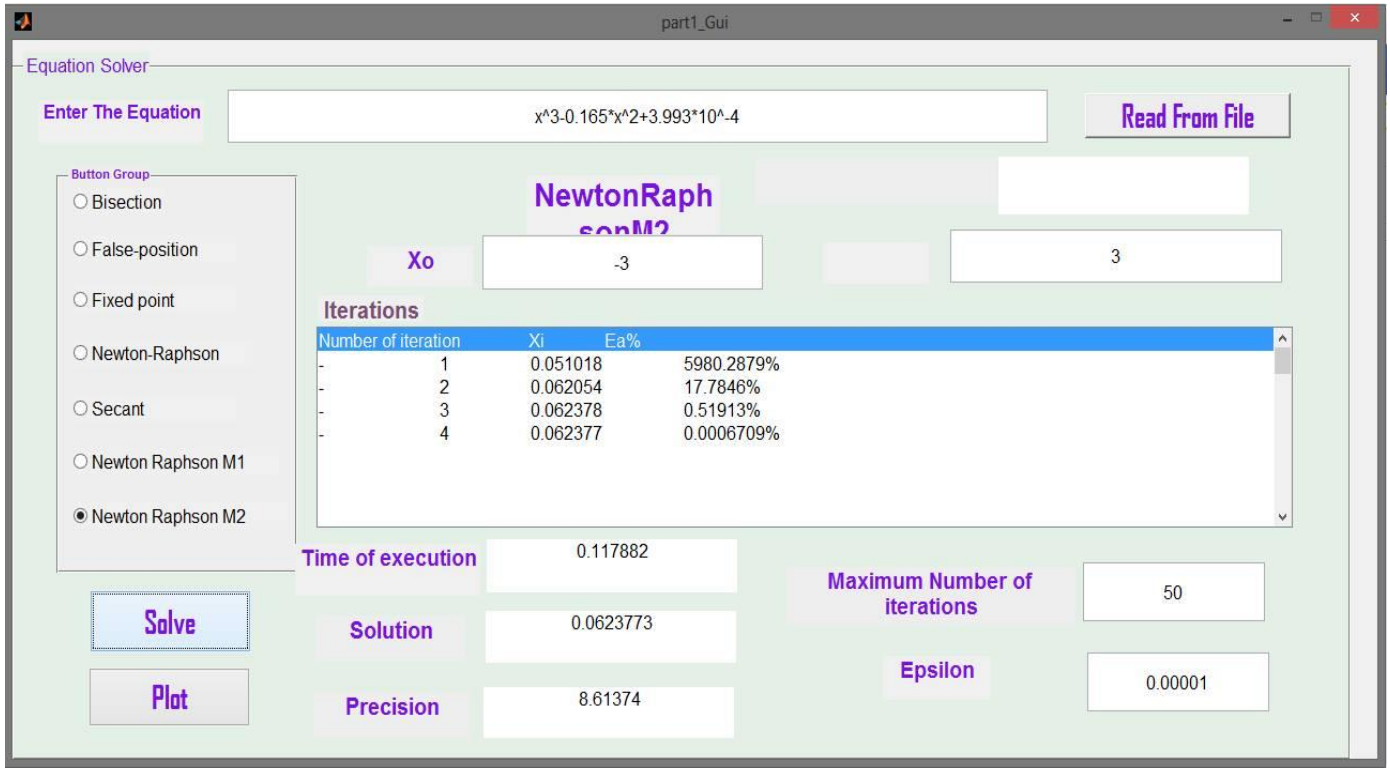



part1_Gui
Equation Solver
Enter The Equation
x^3-0.165*x^2+3.993*10^-4
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaph
Xo
-3
M
3
Iterations
- 9 0.09924 54.9959%
- 10 -0.13333 174.432%
- 11 0.01783 847.7973%
- 12 0.23233 92.3258%
- 13 0.09041 156.9773%
- 14 -0.028382 418.5475%
- 15 -0.090391 68.6009%
- 16 0.0027662 3367.6631%
Time of execution
0.619957
Maximum Number of iterations
40
Solve
Solution
0.0635064
Epsilon
0.00001
Plot
Precision
-5.68937

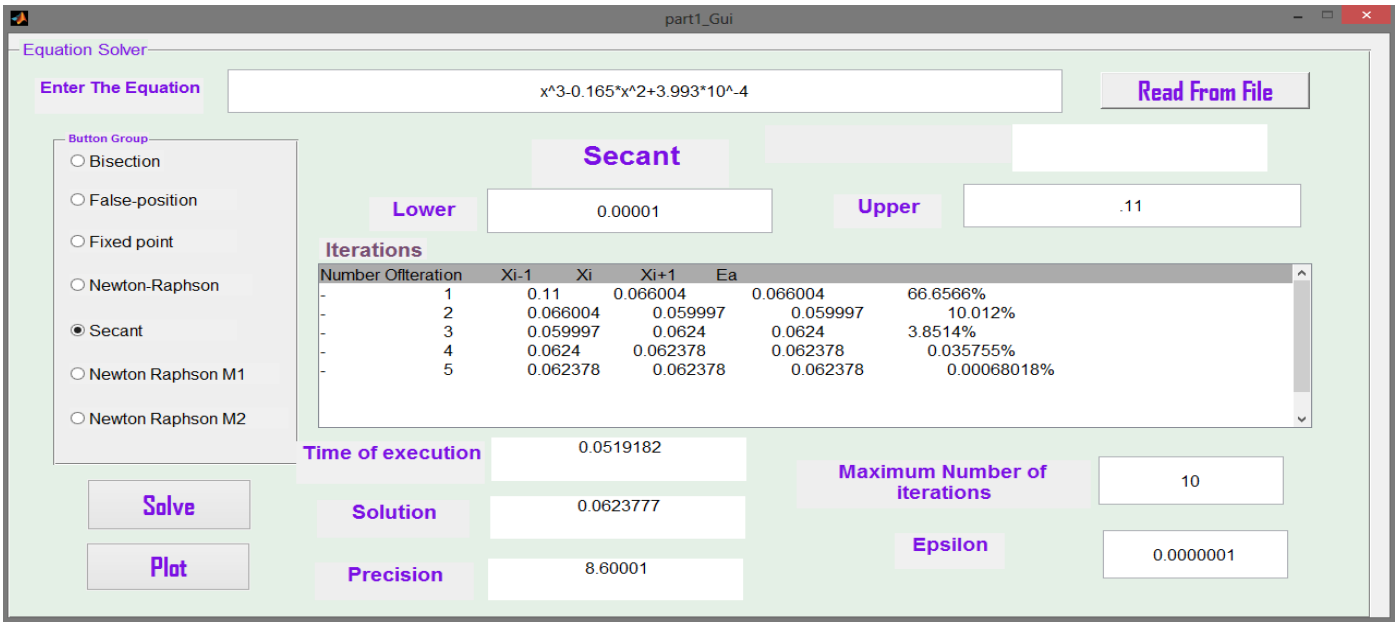

part1_Gui
Equation Solver
Enter The Equation
x^3-0.165*x^2+3.993*10^-4
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaph
Xo
-3
M
3
Iterations
- 17 1.3429 99.794%
- 18 0.059676 2150.3262%
- 19 0.06774 11.9046%
- 20 0.051323 31.9882%
- 21 0.084486 39.2527%
- 22 0.003107 2619.2238%
- 23 1.1996 99.741%
- 24 0.060267 1890.4079%
Time of execution
0.619957
Maximum Number of iterations
40
Solve
Solution
0.0635064
Epsilon
0.00001
Plot
Precision
-5.68937



In Newton Raphson Modified 2:



In secant method :



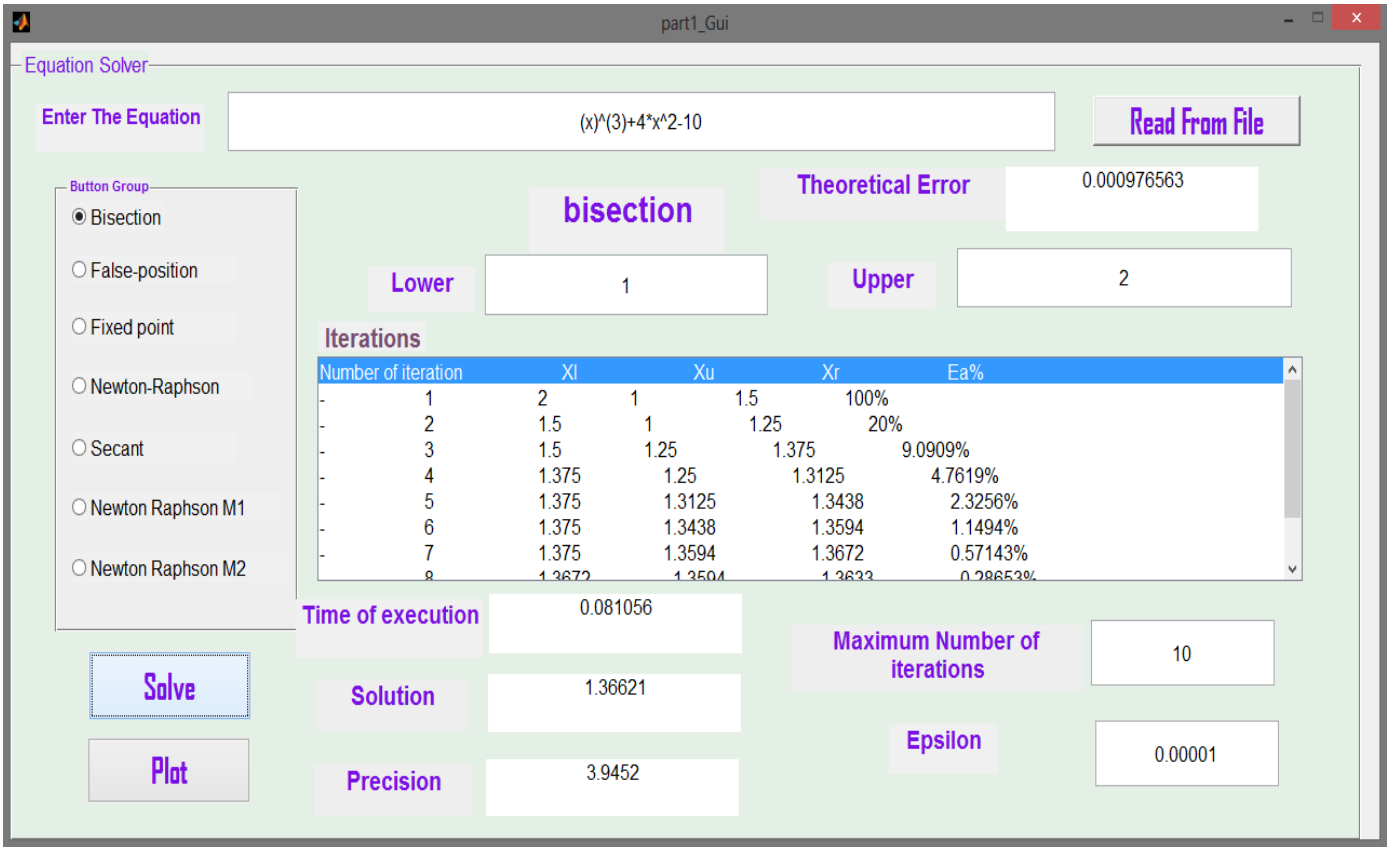
Method	Time	Solution	Precision	Iterations	relativeError
Bisection	0.094557	0.0624164	3.06659	10	0.17209%
Falseposition	0.0413807	0.0623775	11.4457	10	3.9516e-05%
Fixedpoint	0.213752	0.0623773	inf	19	0.0009006 %
Newton	0.189097	3.20212	-2.58	10	49.1208%
Newton 1	0.619957	0.0635064	-5.689	40	40.5%
Newton 2	0.117882	0.0623773	8.61374	4	0.000679%
Secant	0.0519182	0.062377	8.60001	5	0.000681%

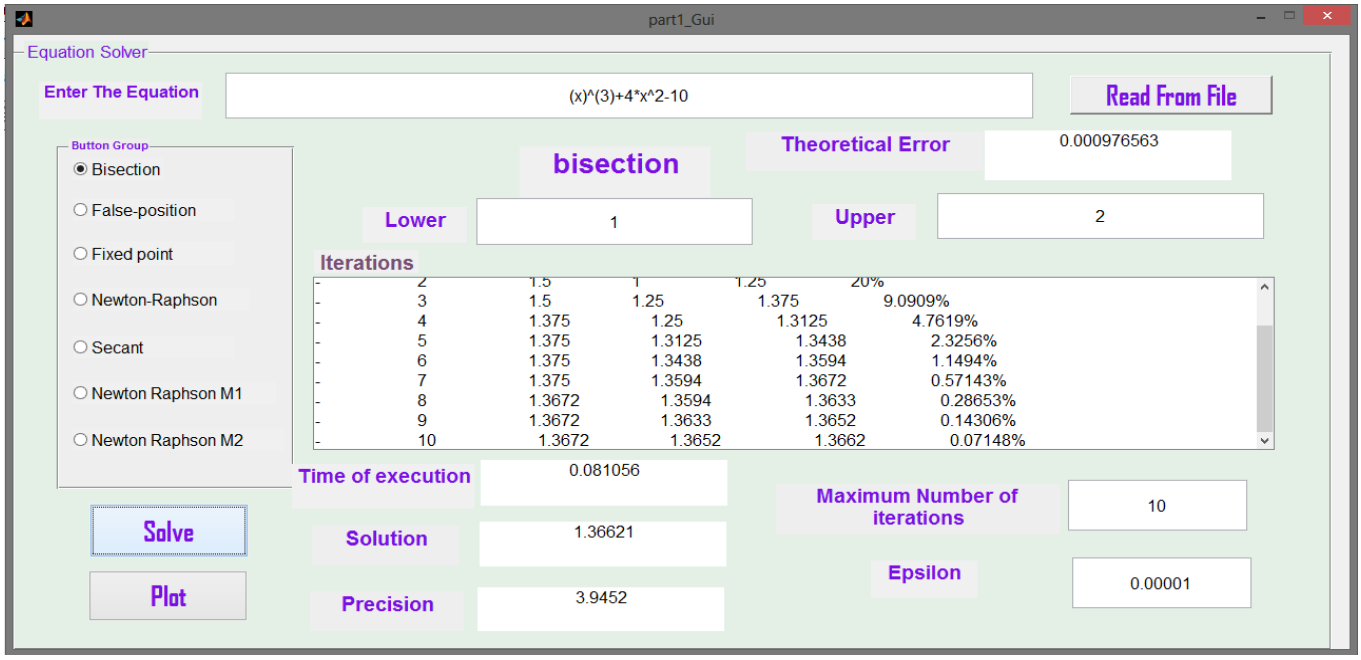
We can see that false Position take the least time of execution and it got the accurate root, Newton Raphson Modified 2 takes the least number of iteration and precisions are almost equal in every method so we can say that false position is the accurate and the fastest method in this example.

Example 2:

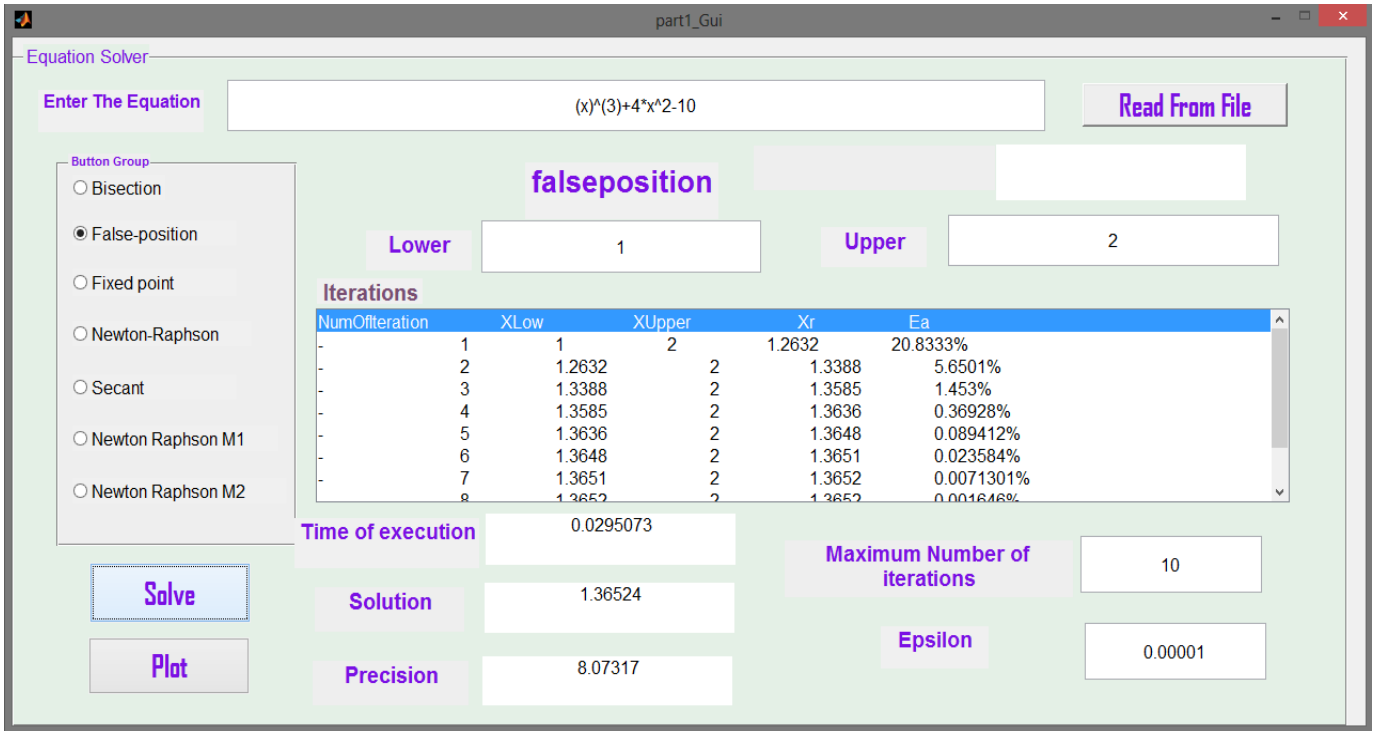
Equation : (x)^(3)+4*x^2-10

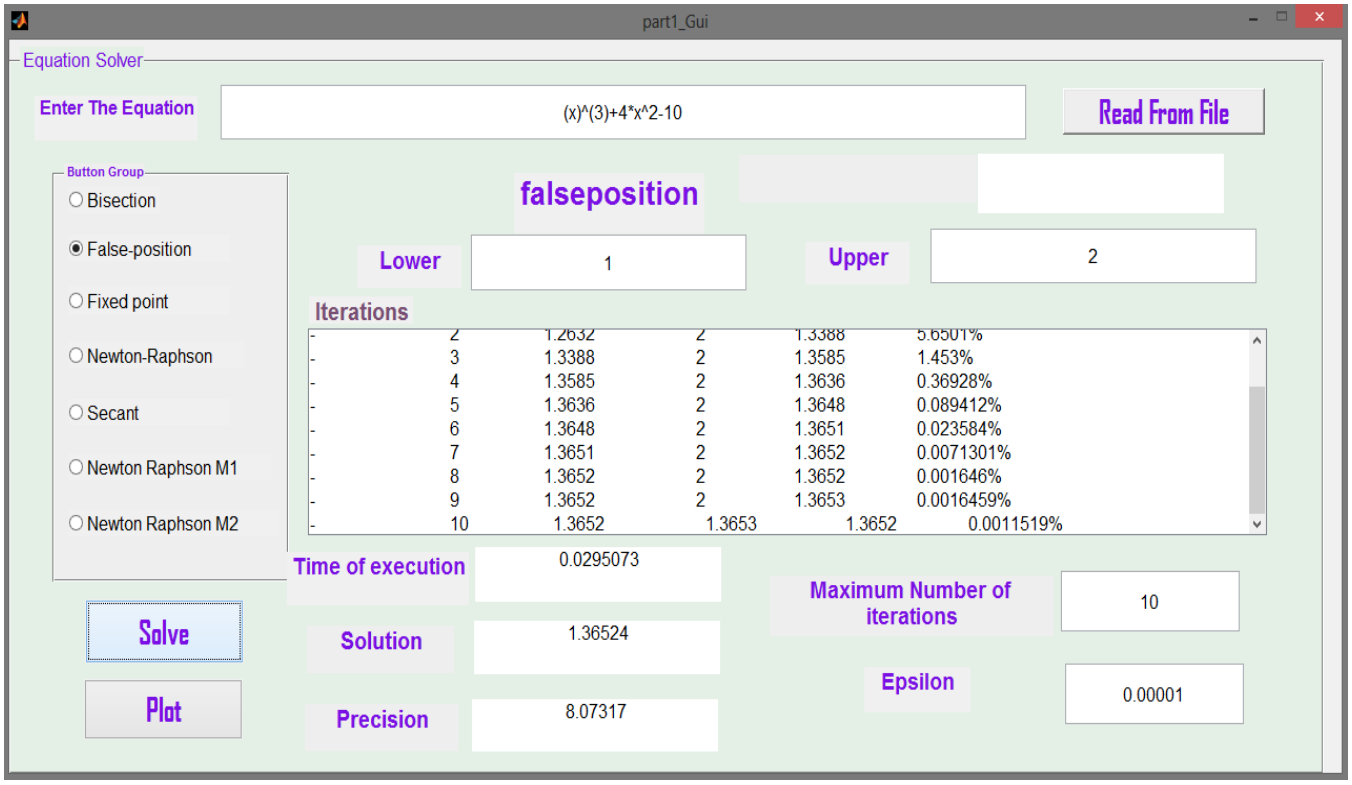
In bisection :



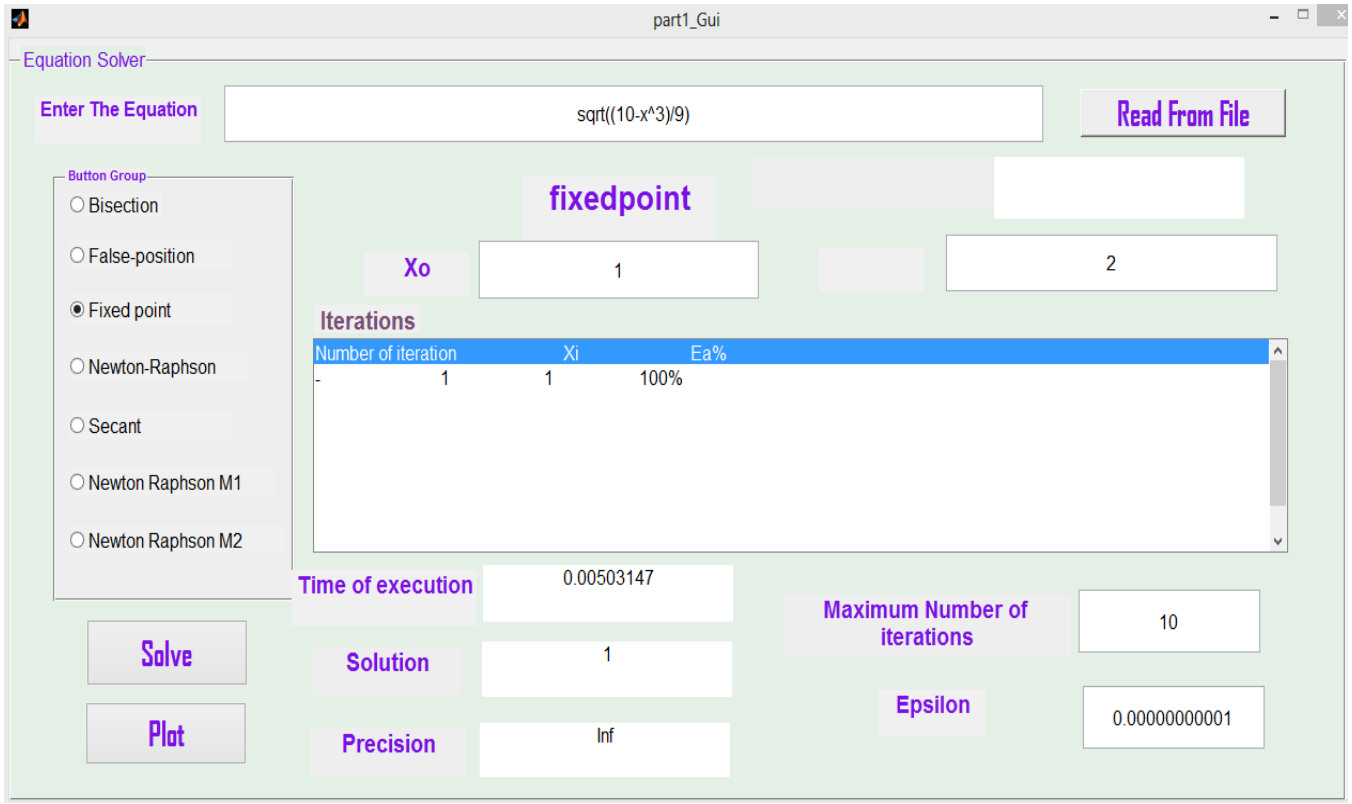


In falseposition :

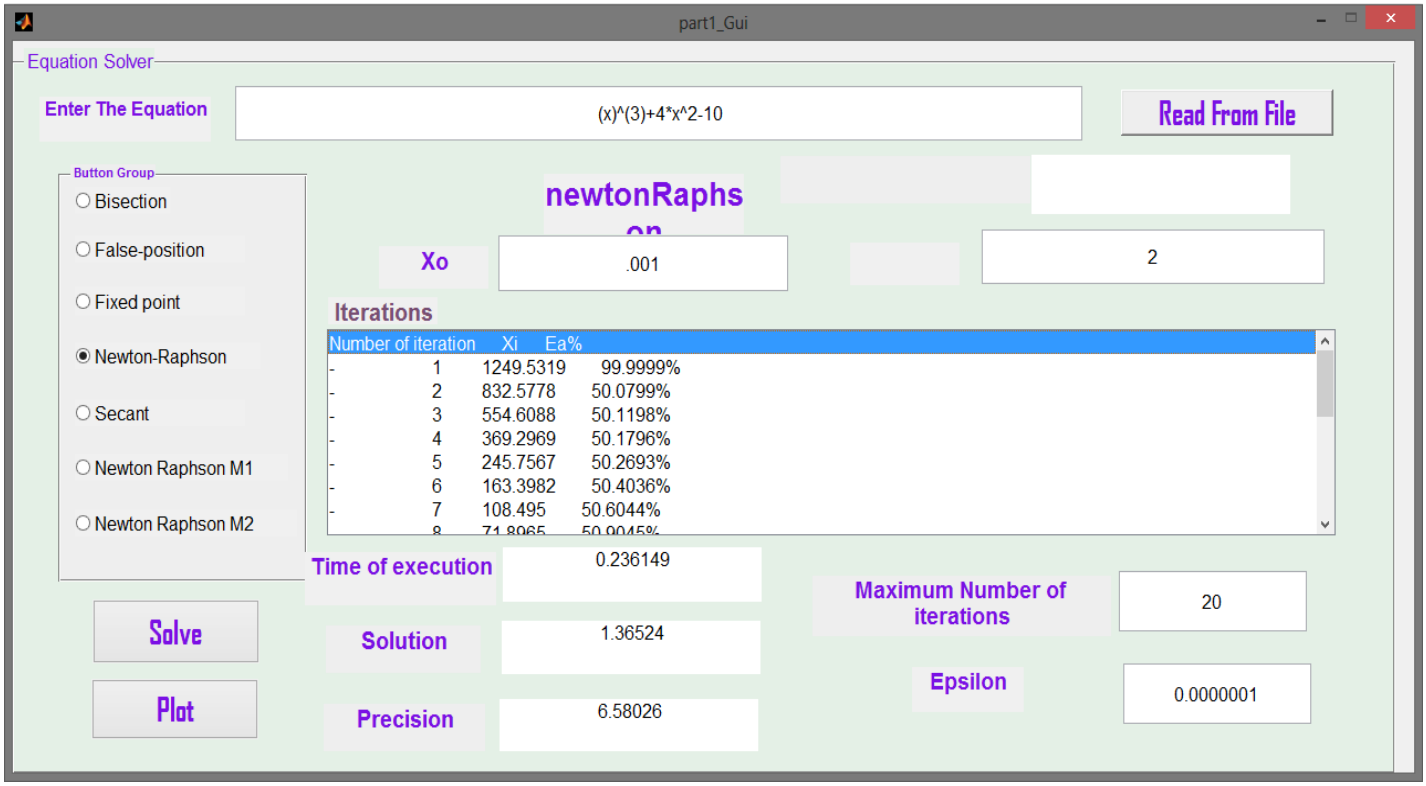


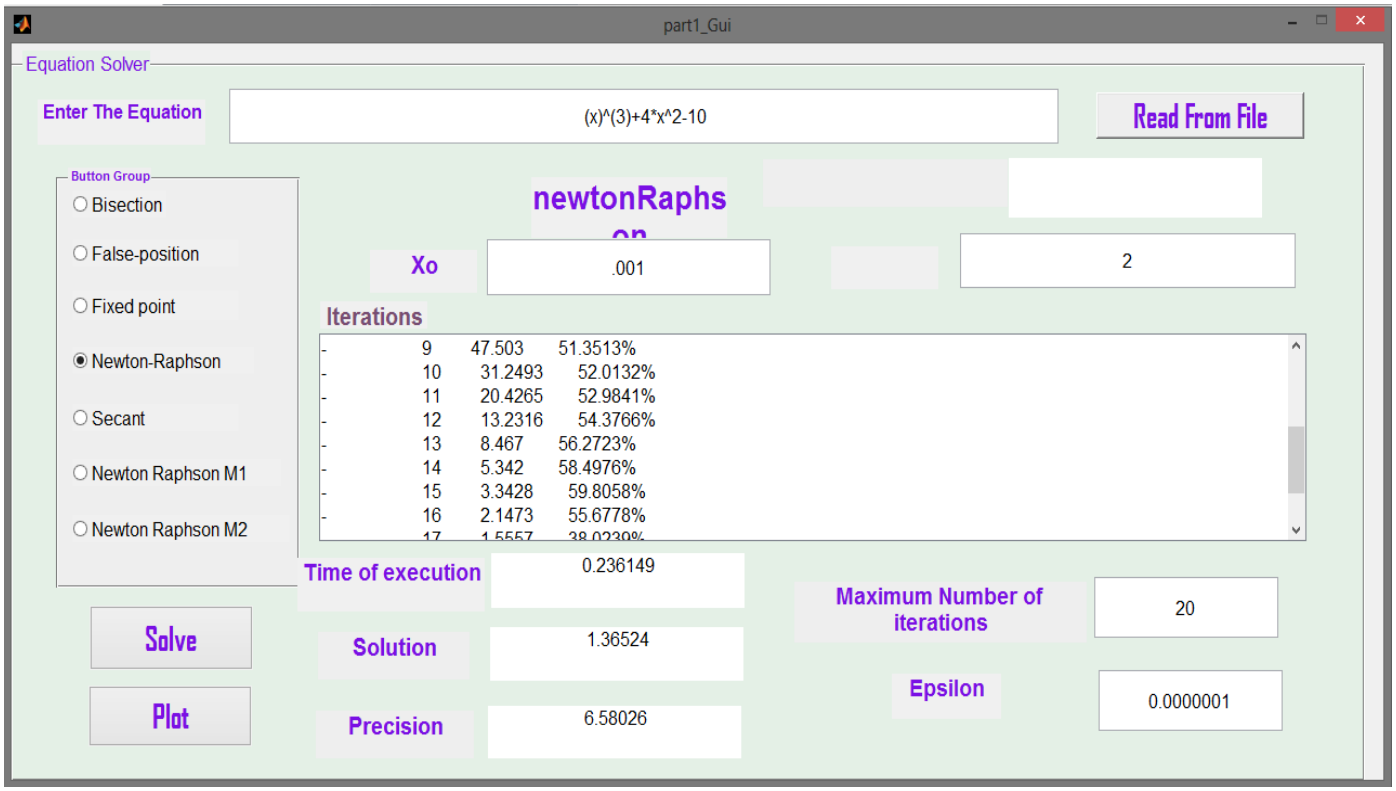


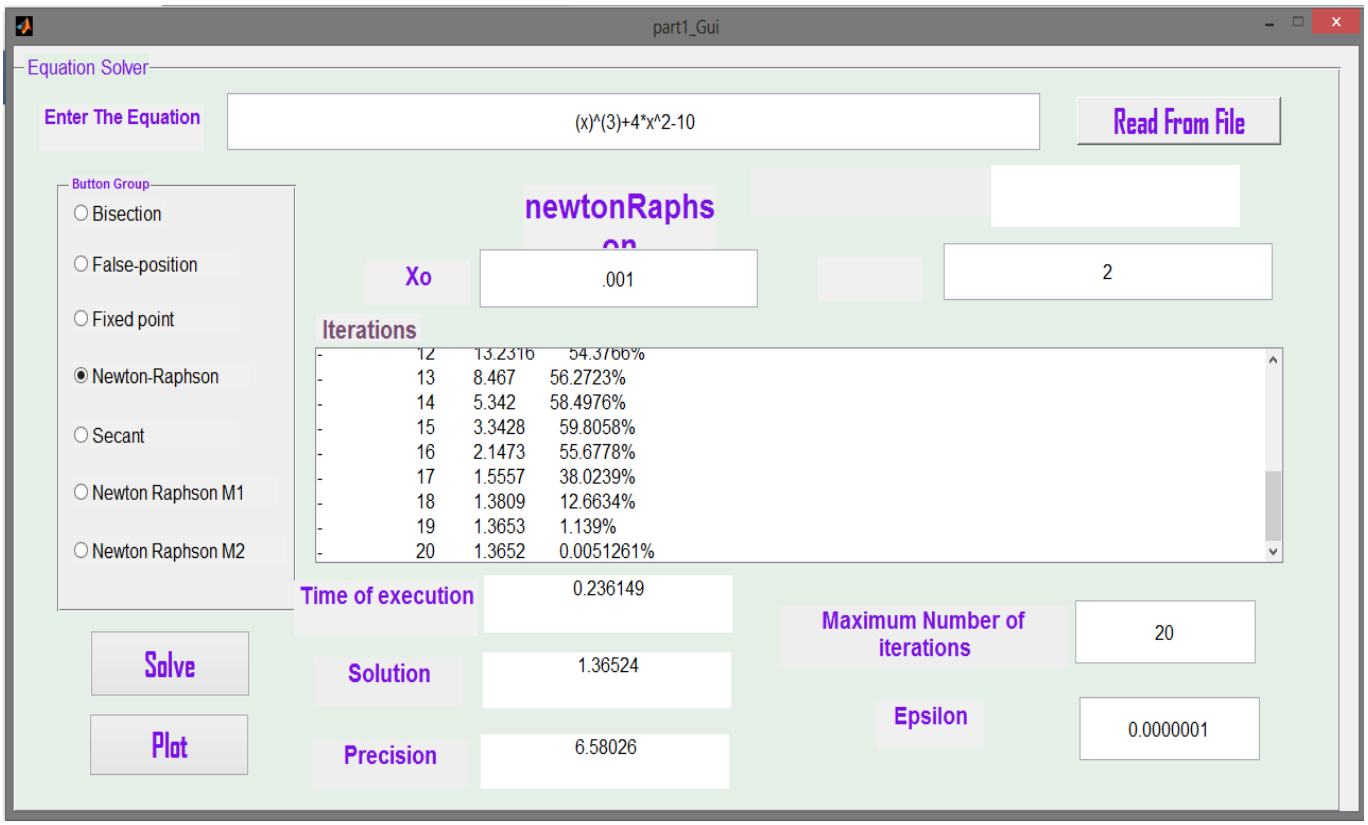
In fixedpoint :



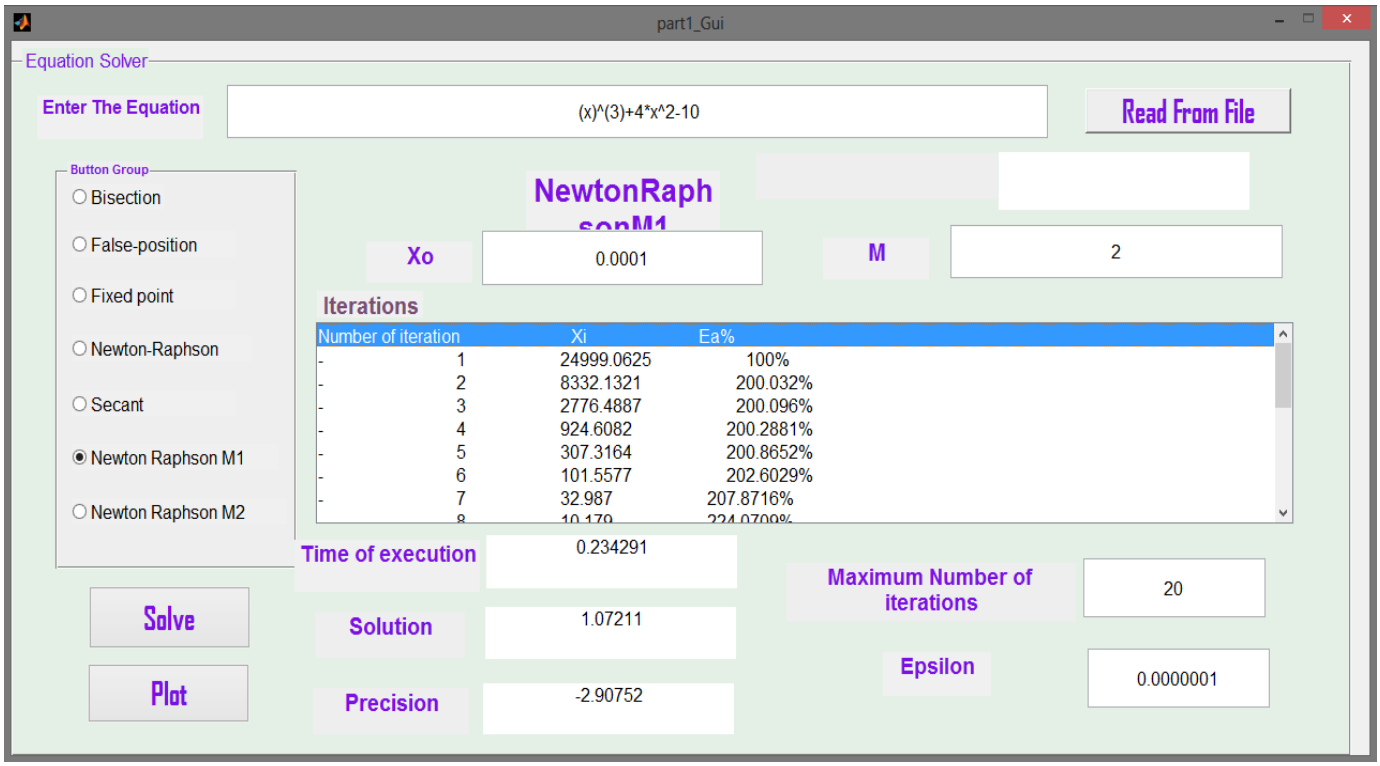
In newtonraphson :

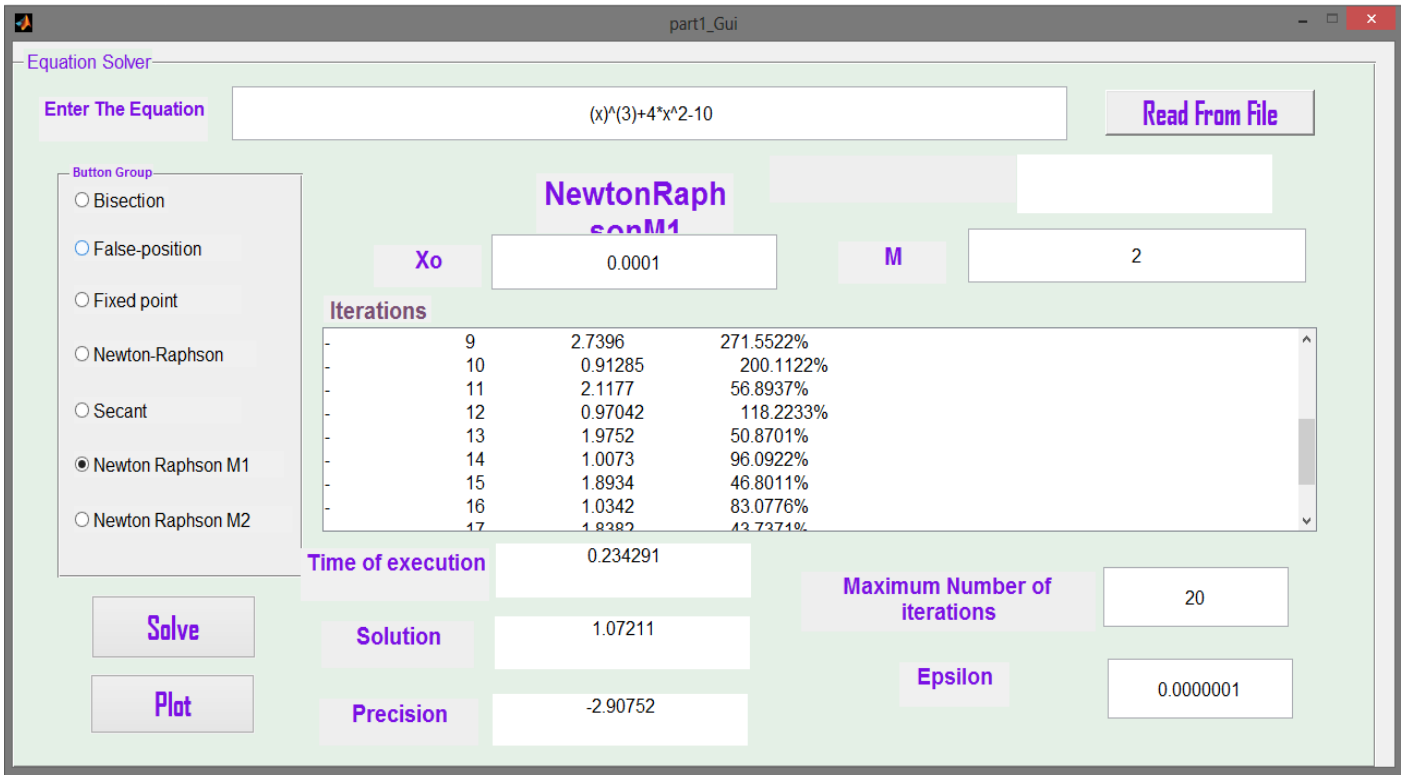


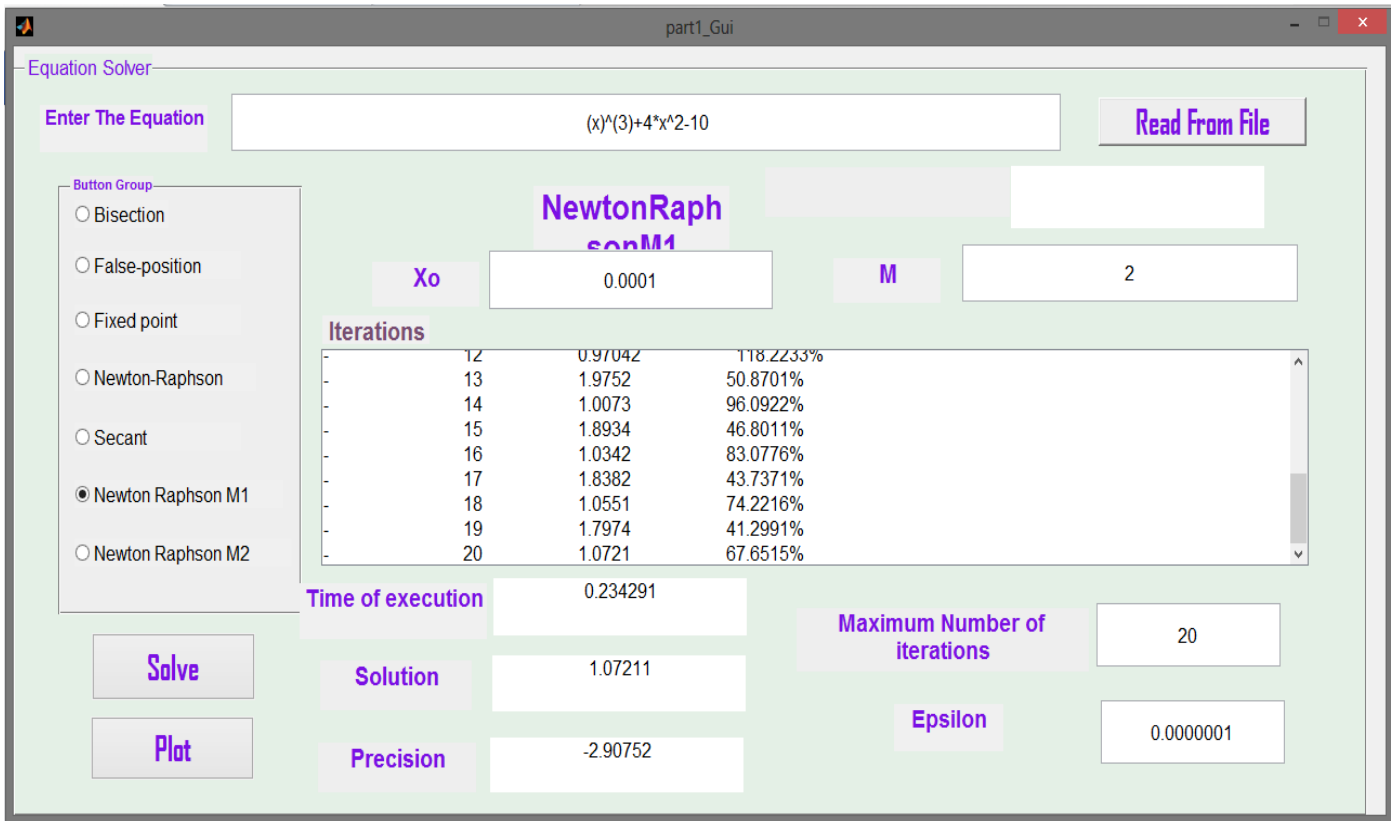




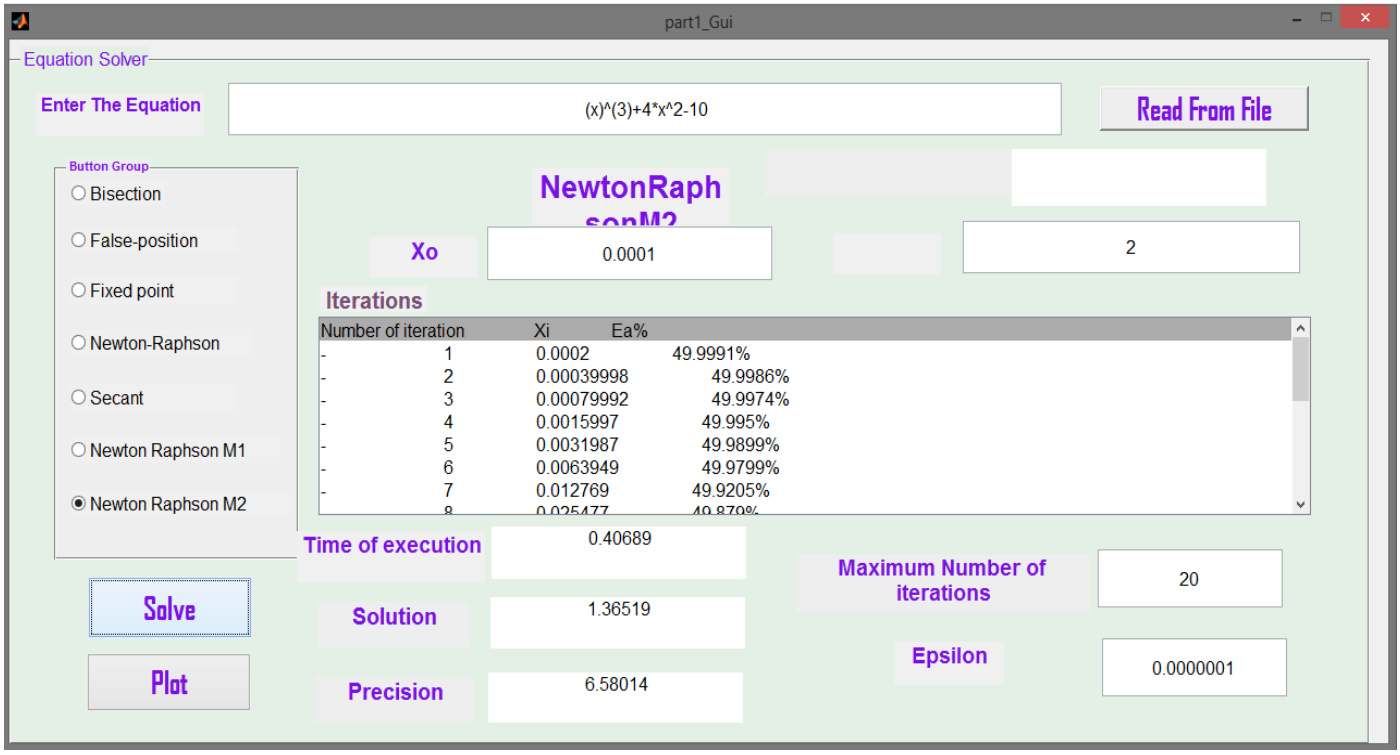
In newtonraphon 1:

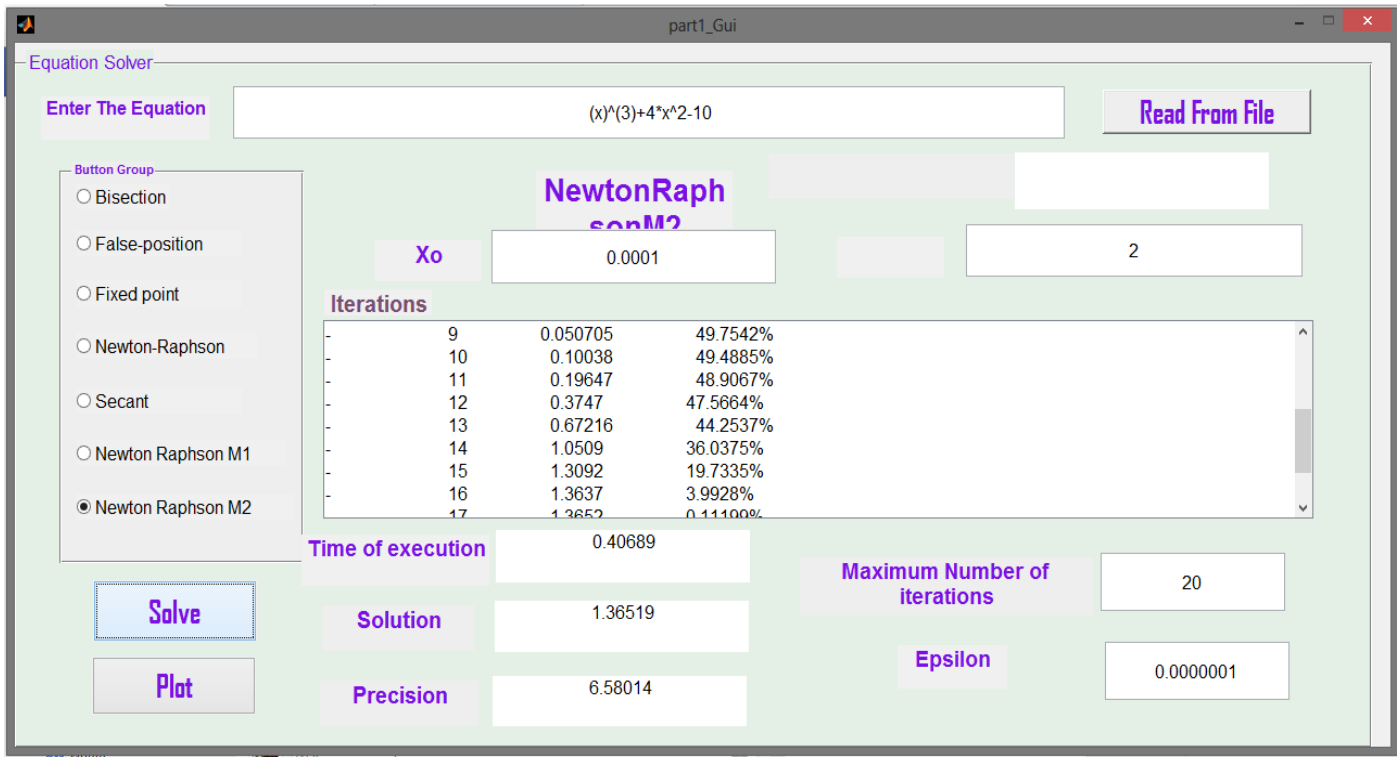


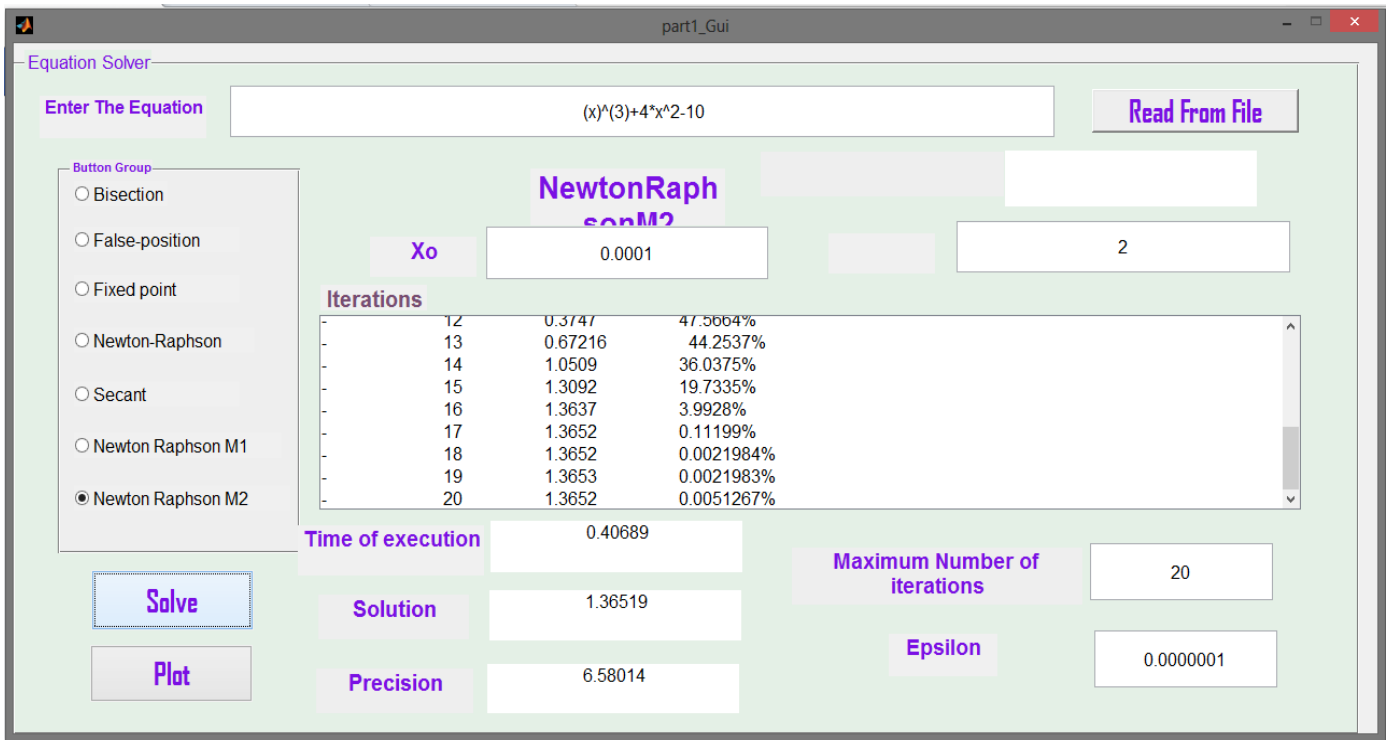

part1_Gui
Equation Solver
Enter The Equation
(x)^(3)+4*x^2-10
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaphsonM1
Xo
0.0001
M
2
Iterations
9 2.7396 271.5522%
10 0.91285 200.1122%
11 2.1177 56.8937%
12 0.97042 118.2233%
13 1.9752 50.8701%
14 1.0073 96.0922%
15 1.8934 46.8011%
16 1.0342 83.0776%
Time of execution
0.234291
Maximum Number of iterations
20
Solve
Solution
1.07211
Epsilon
0.0000001
Plot
Precision
-2.90752


part1_Gui
Equation Solver
Enter The Equation
(x)^(3)+4*x^2-10
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaphsonM1
Xo
0.0001
M
2
Iterations
12 0.97042 118.2233%
13 1.9752 50.8701%
14 1.0073 96.0922%
15 1.8934 46.8011%
16 1.0342 83.0776%
17 1.8382 43.7371%
18 1.0551 74.2216%
19 1.7974 41.2991%
20 1.0721 67.6515%
Time of execution
0.234291
Maximum Number of iterations
20
Solve
Solution
1.07211
Epsilon
0.0000001
Plot
Precision
-2.90752

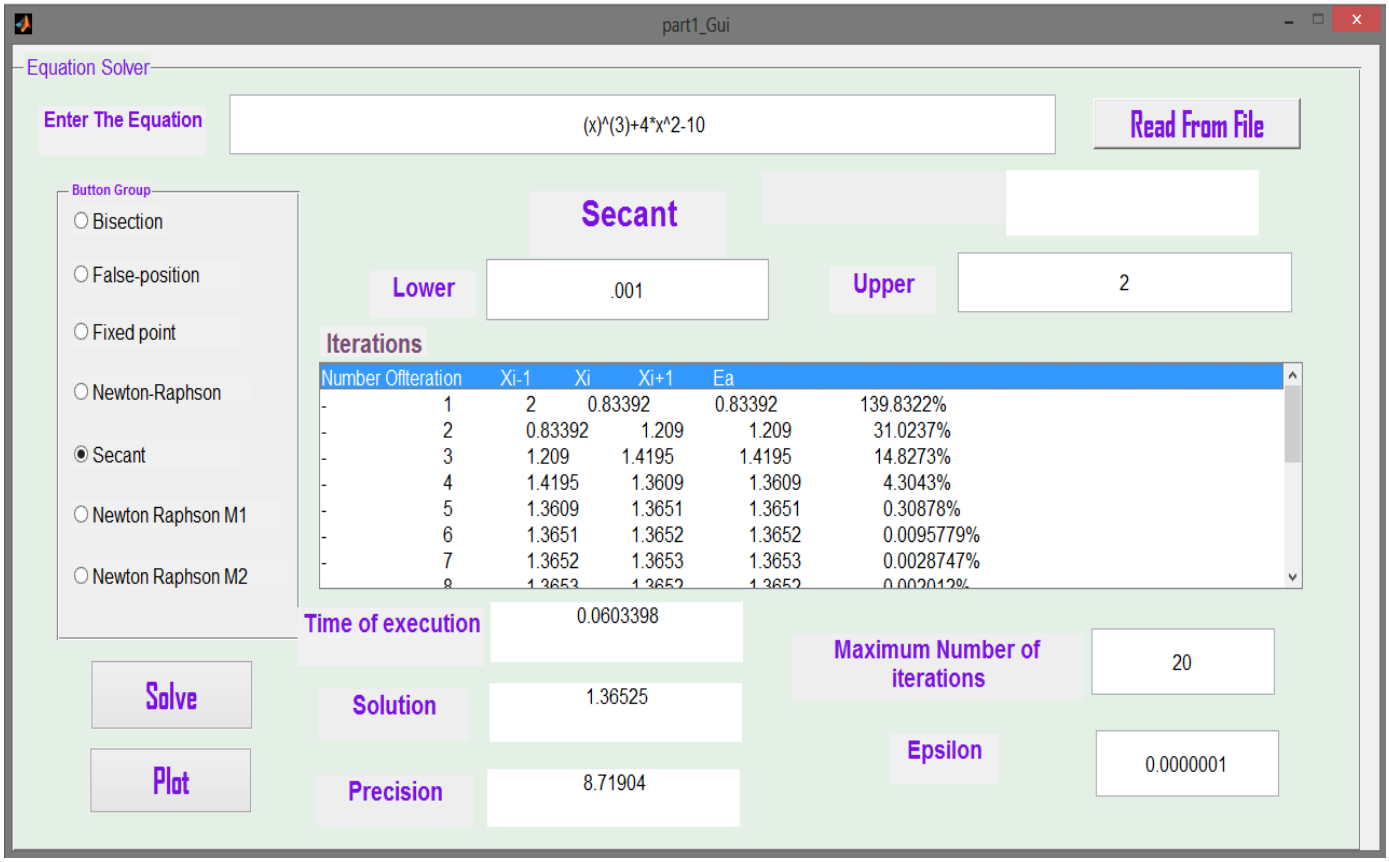
In newtonraphson 2:

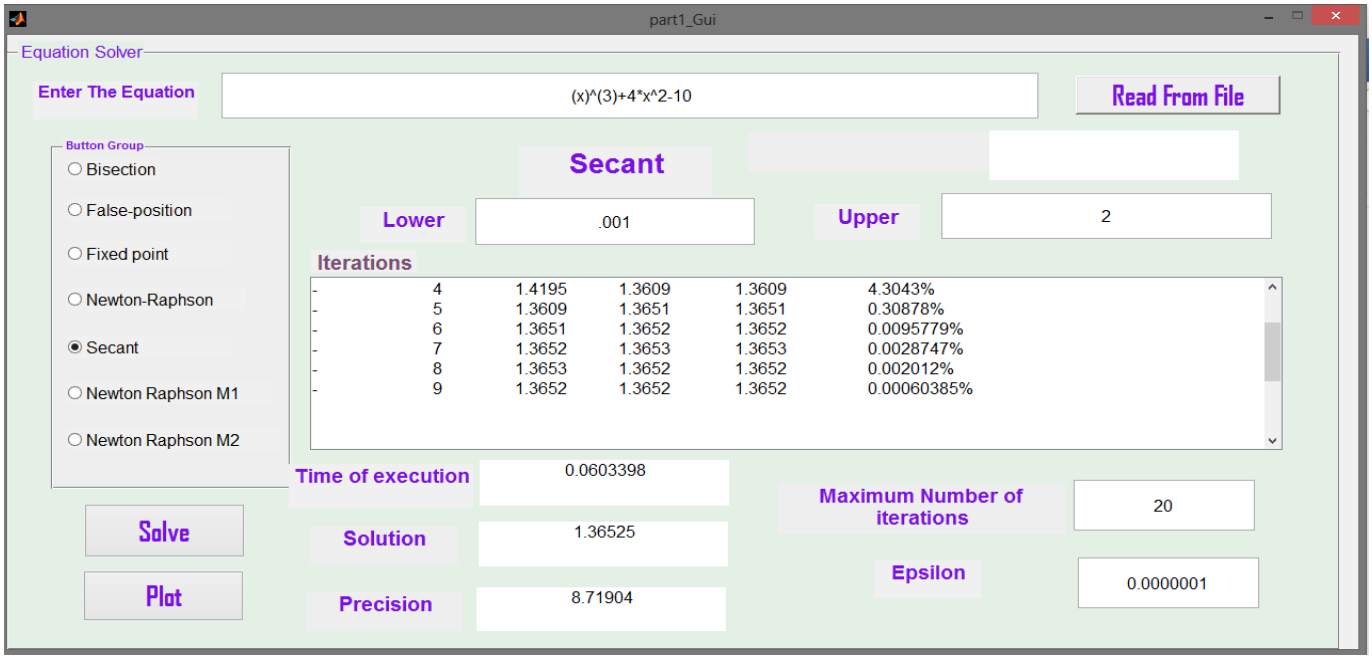





part1_Gui
Equation Solver
Enter The Equation
(x)^(3)+4*x^2-10
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaph
Xo
0.0001
2
Iterations
13 0.67216 44.2537%
14 1.0509 36.0375%
15 1.3092 19.7335%
16 1.3637 3.9928%
17 1.3652 0.11199%
18 1.3652 0.0021984%
19 1.3653 0.0021983%
20 1.3652 0.0051267%
Time of execution
0.40689
Maximum Number of iterations
20
Solve
Solution
1.36519
Epsilon
0.0000001
Plot
Precision
6.58014

In secant :


part1_Gui
Equation Solver
Enter The Equation
(x)^(3)+4*x^2-10
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
Secant
Lower
.001
Upper
2
Iterations
Number Ofiteration Xi-1 Xi Xi+1 Ea
1 2 0.83392 0.83392 139.8322%
2 0.83392 1.209 1.209 31.0237%
3 1.209 1.4195 1.4195 14.8273%
4 1.4195 1.3609 1.3609 4.3043%
5 1.3609 1.3651 1.3651 0.30878%
6 1.3651 1.3652 1.3652 0.0095779%
7 1.3652 1.3653 1.3653 0.0028747%
Time of execution
0.0603398
Maximum Number of iterations
20
Solve
Solution
1.36525
Epsilon
0.0000001
Plot
Precision
8.71904



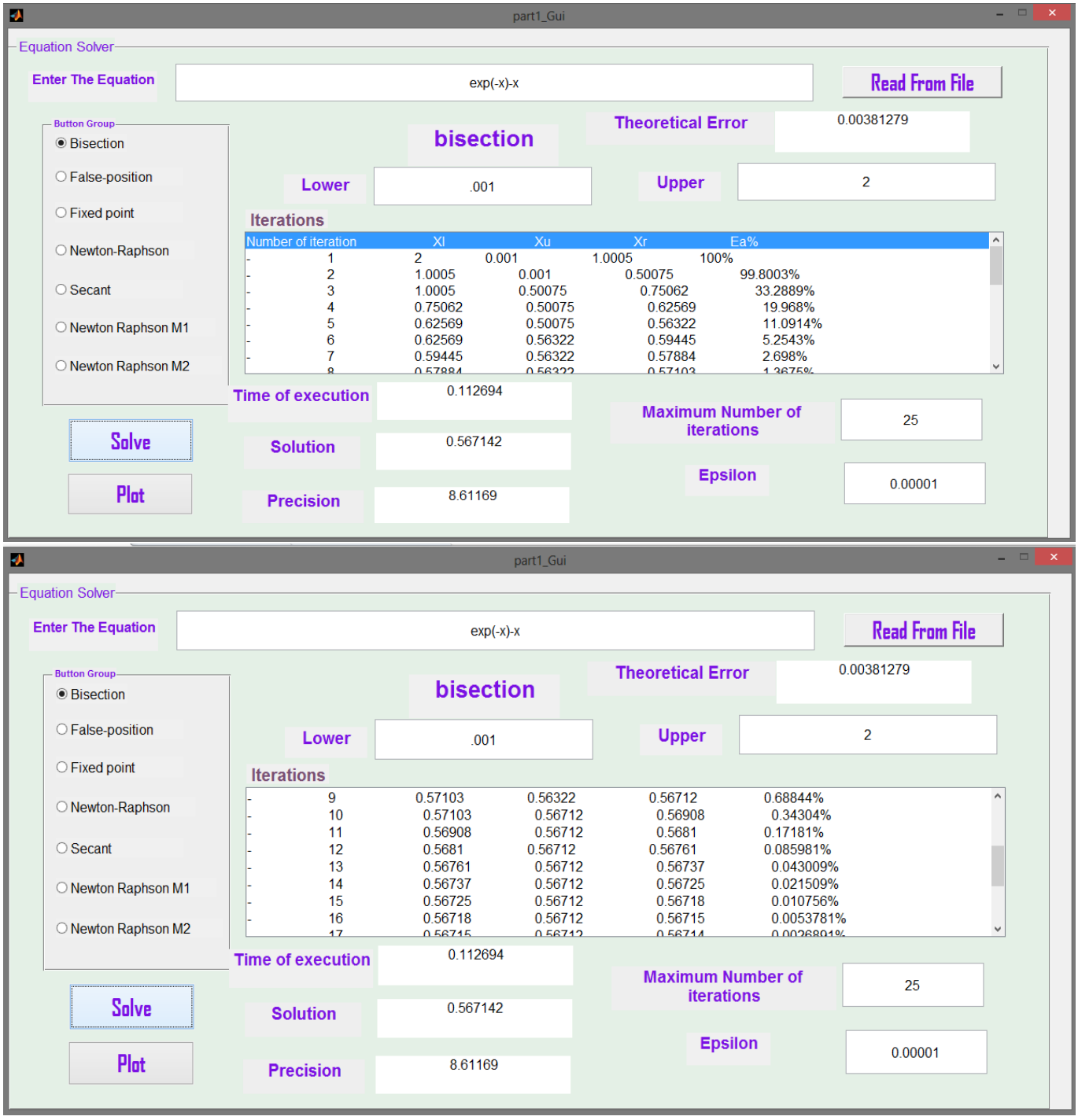
Method	Time	Solution	Precision	Iterations	Relativeerror
Bisection	0.081056	1.36621	3.9452	10	0.07148%
Falseposition	0.0295073	1.36524	8.07317	10	0.001519%
Fixedpoint	0.0050317	1	inf	1	100%
Newton	0.236149	1.36245	6.58026	20	0.0051216%
Newton 1	0.234291	1.07211	-2.9705	20	67.6511%
Newton2	0.40689	1.36159	6.58014	20	0.0051267%
Secant	0.0603398	1.36525	8.71904	9	0.00060358%

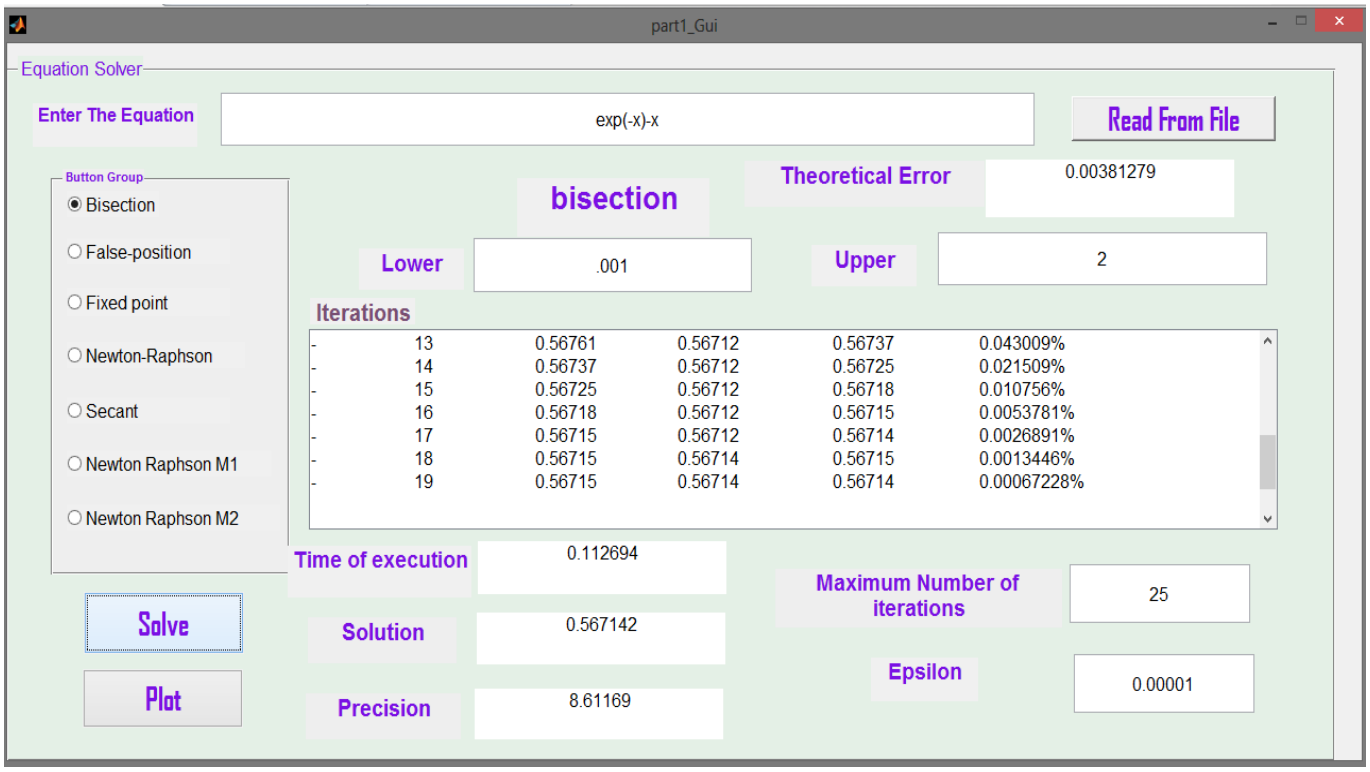
We can see that Secant take the least time of execution ,it got the accurate root and the least number of and the least precision is in bisection method so we can say that Secant is the accurate and the fastest method in this example.

Example 3:

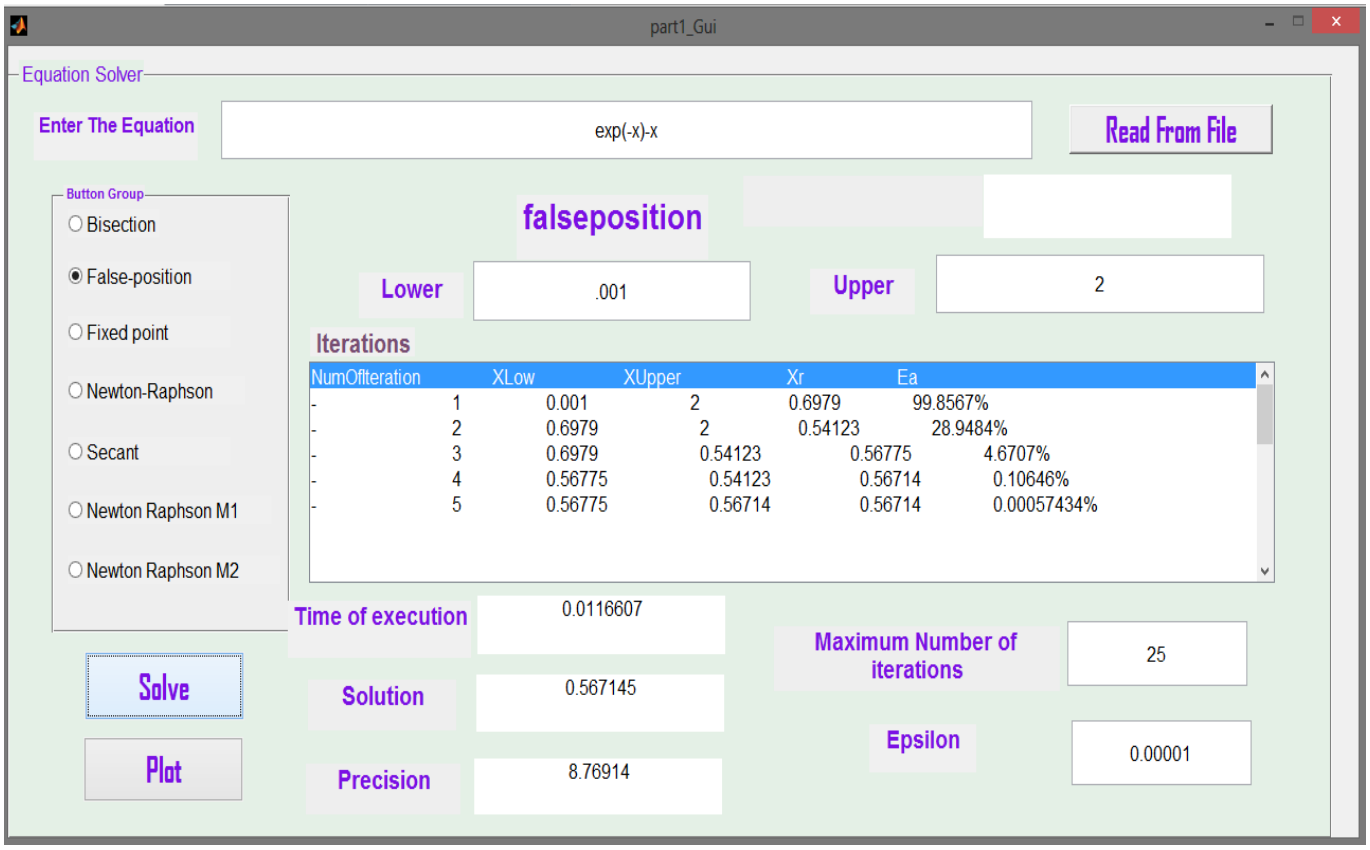
Equation = e^(-x) –x

- In Bisection

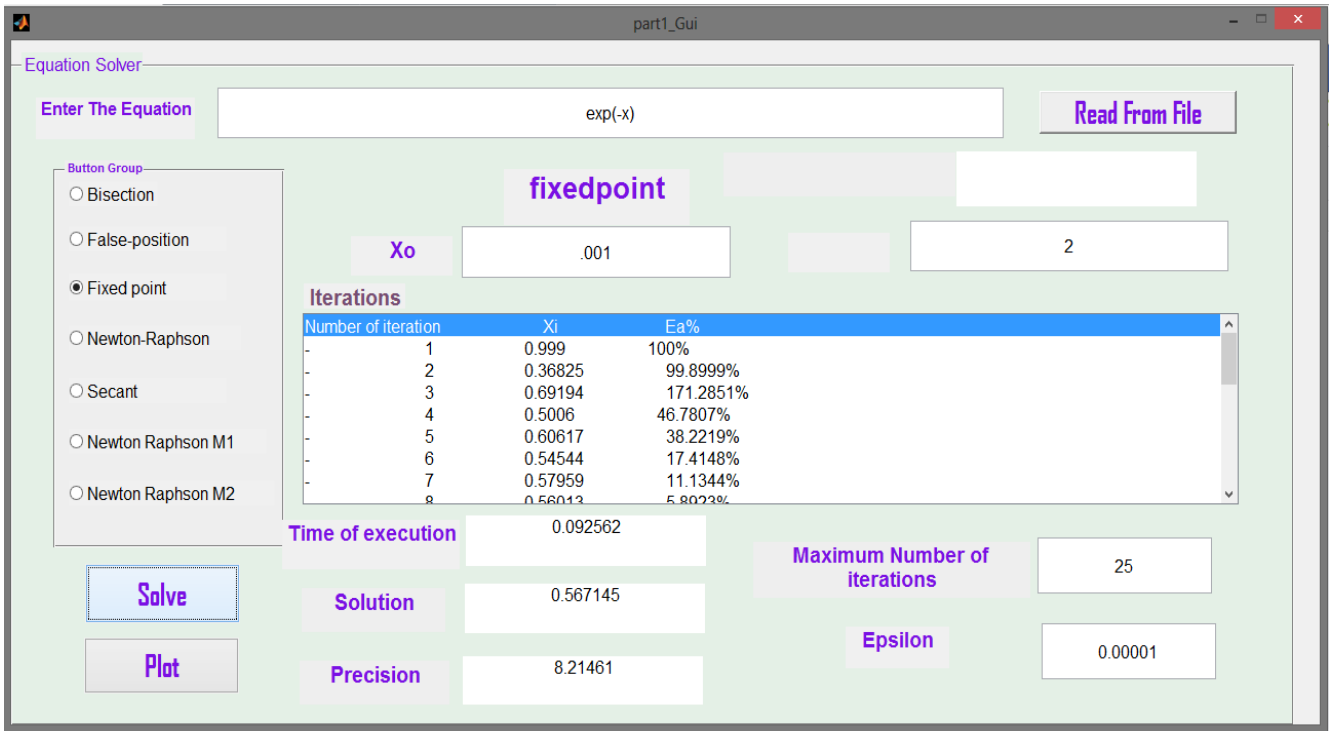


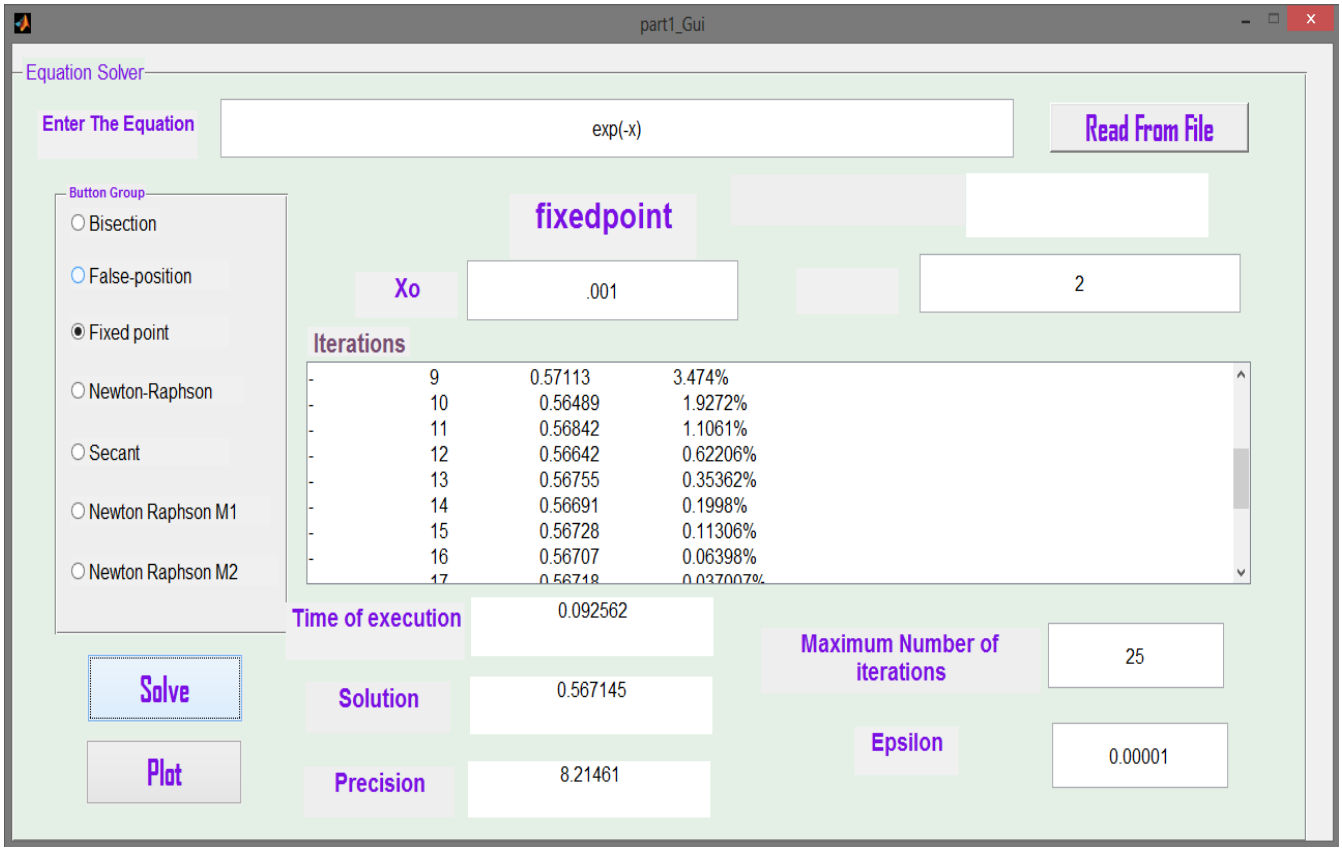


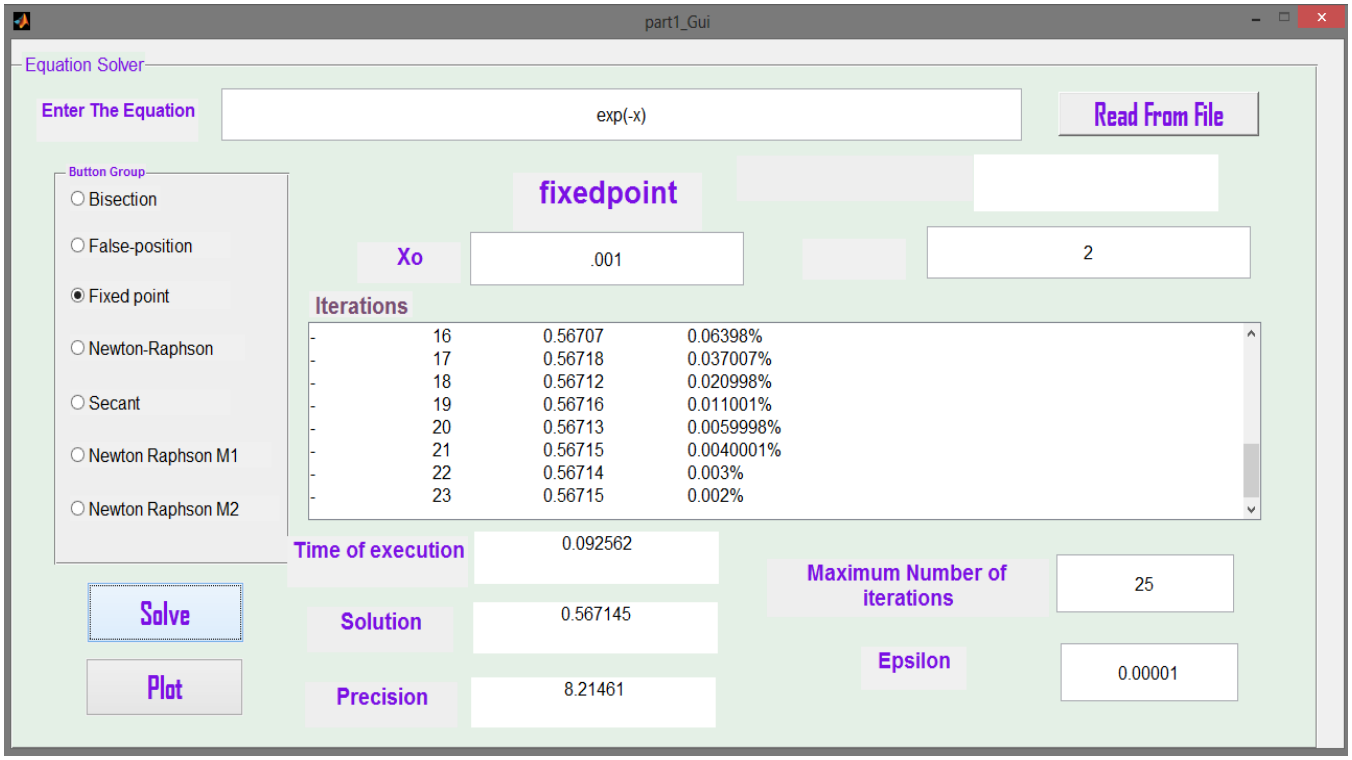
- In False Position



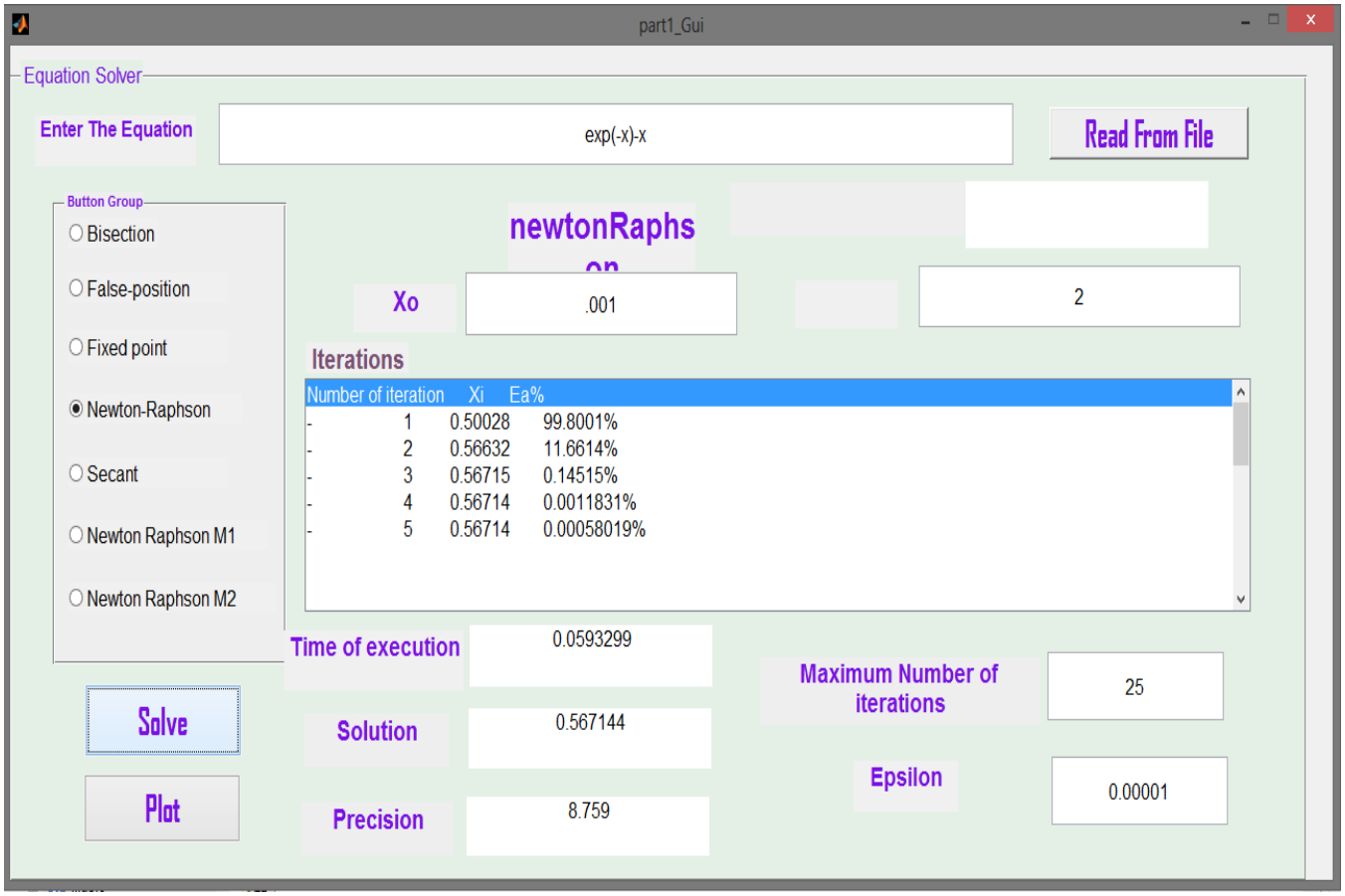
- In Fixed point



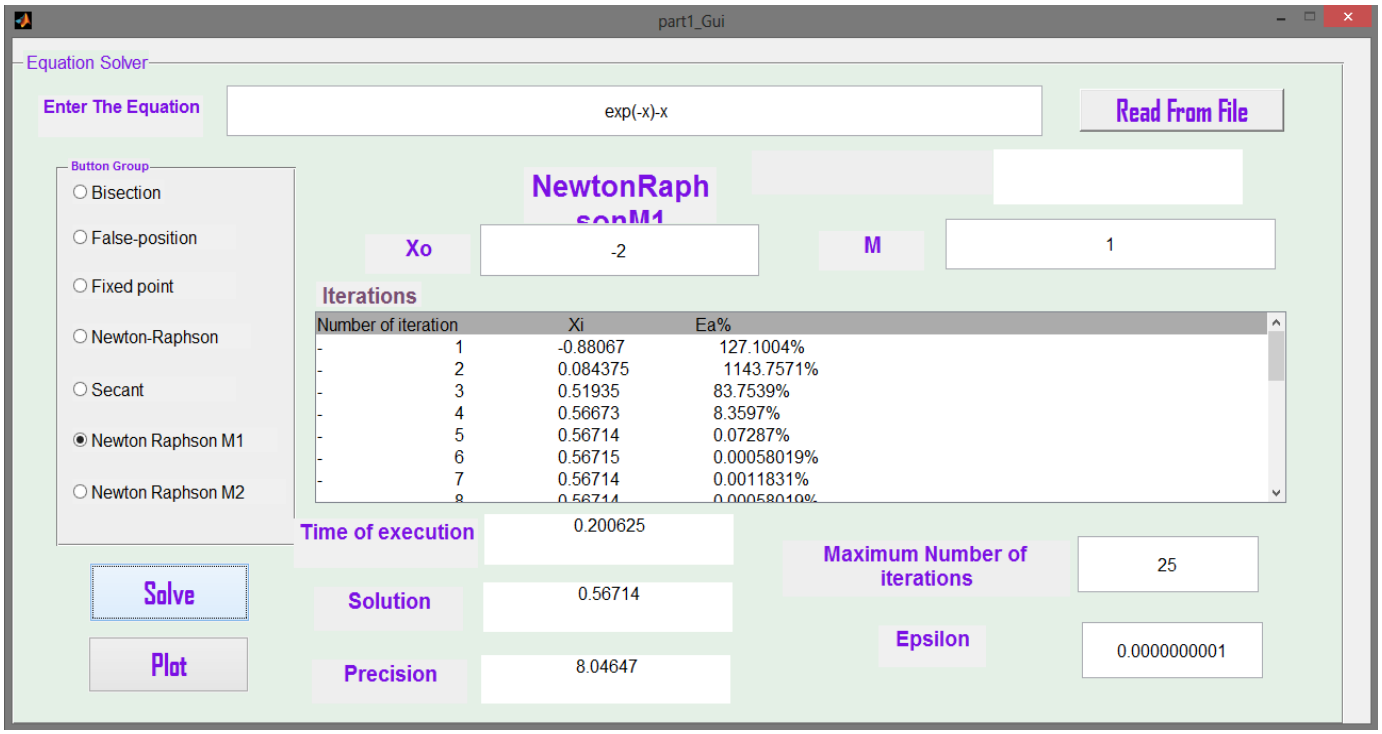


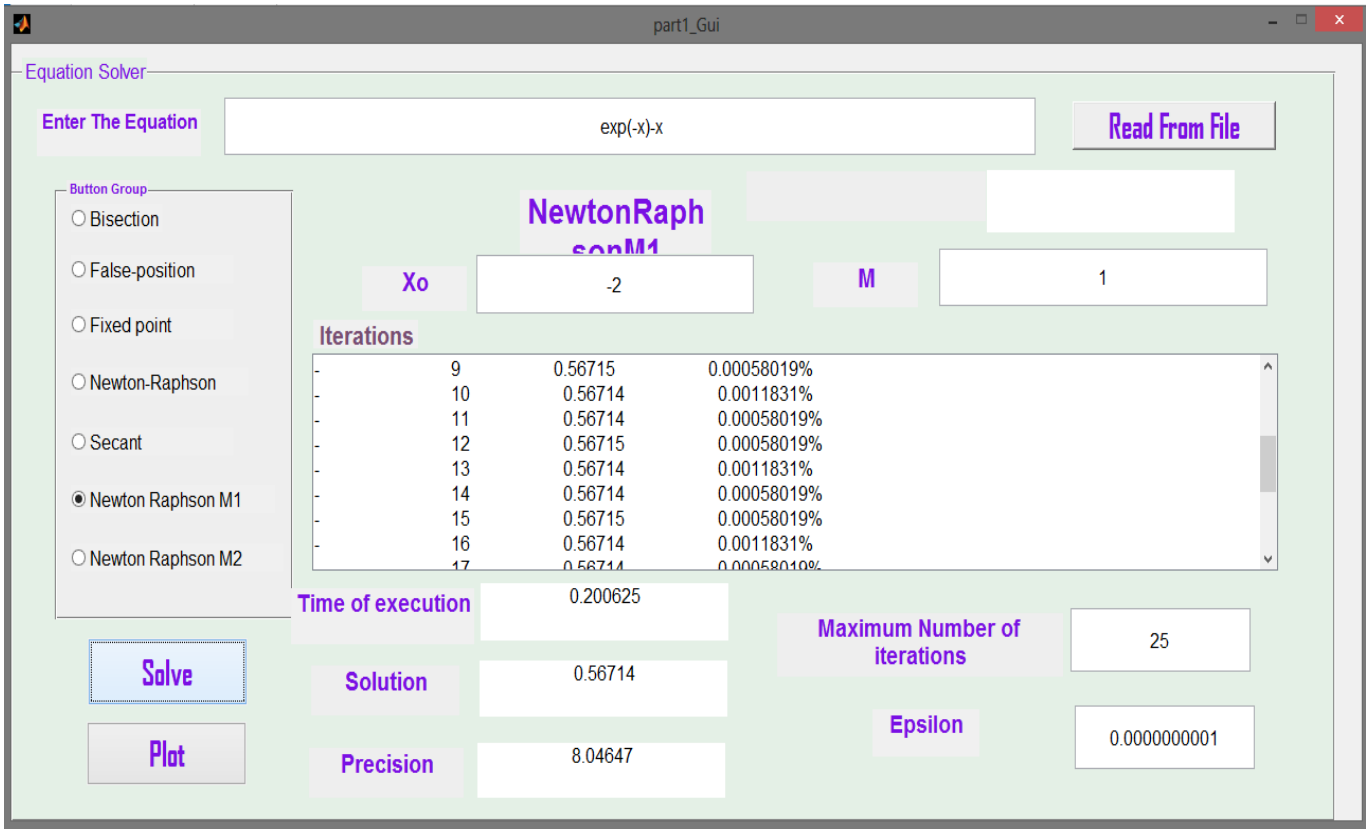


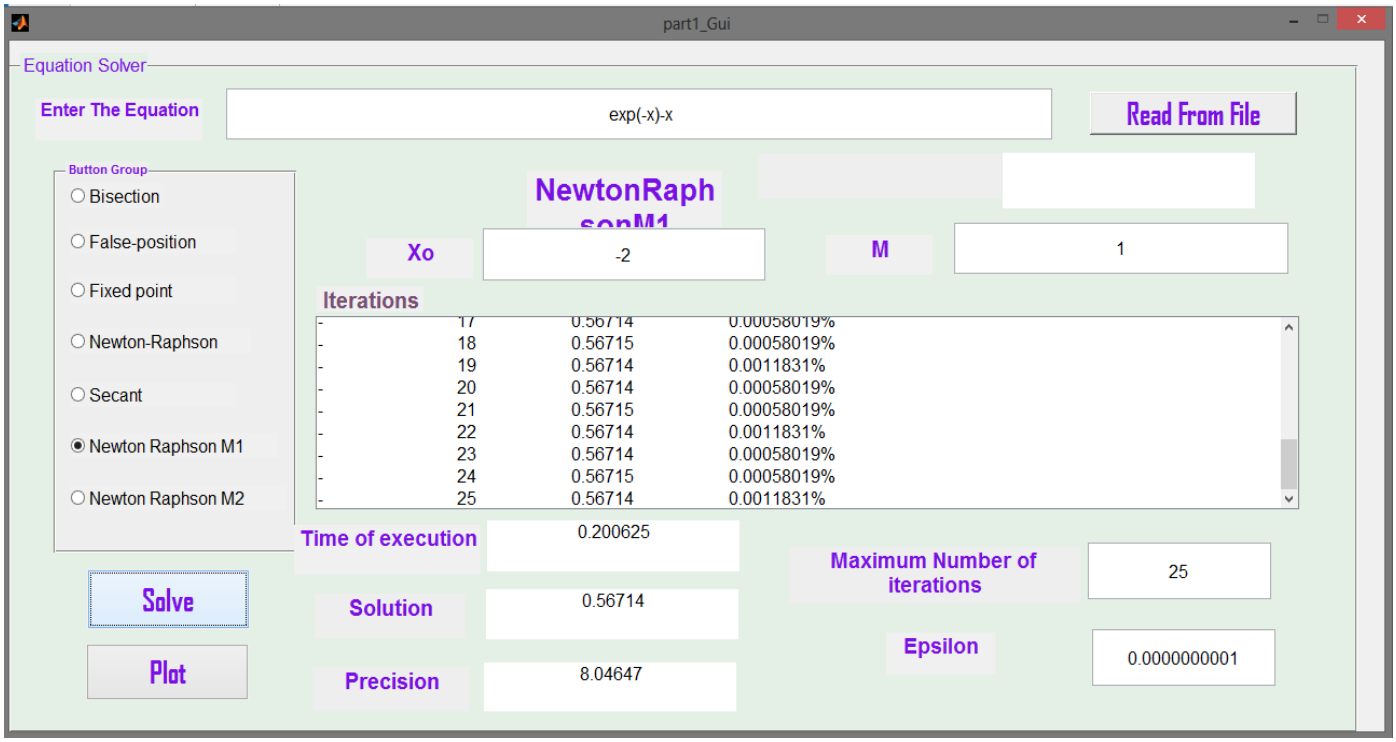
- In Newton Raphson



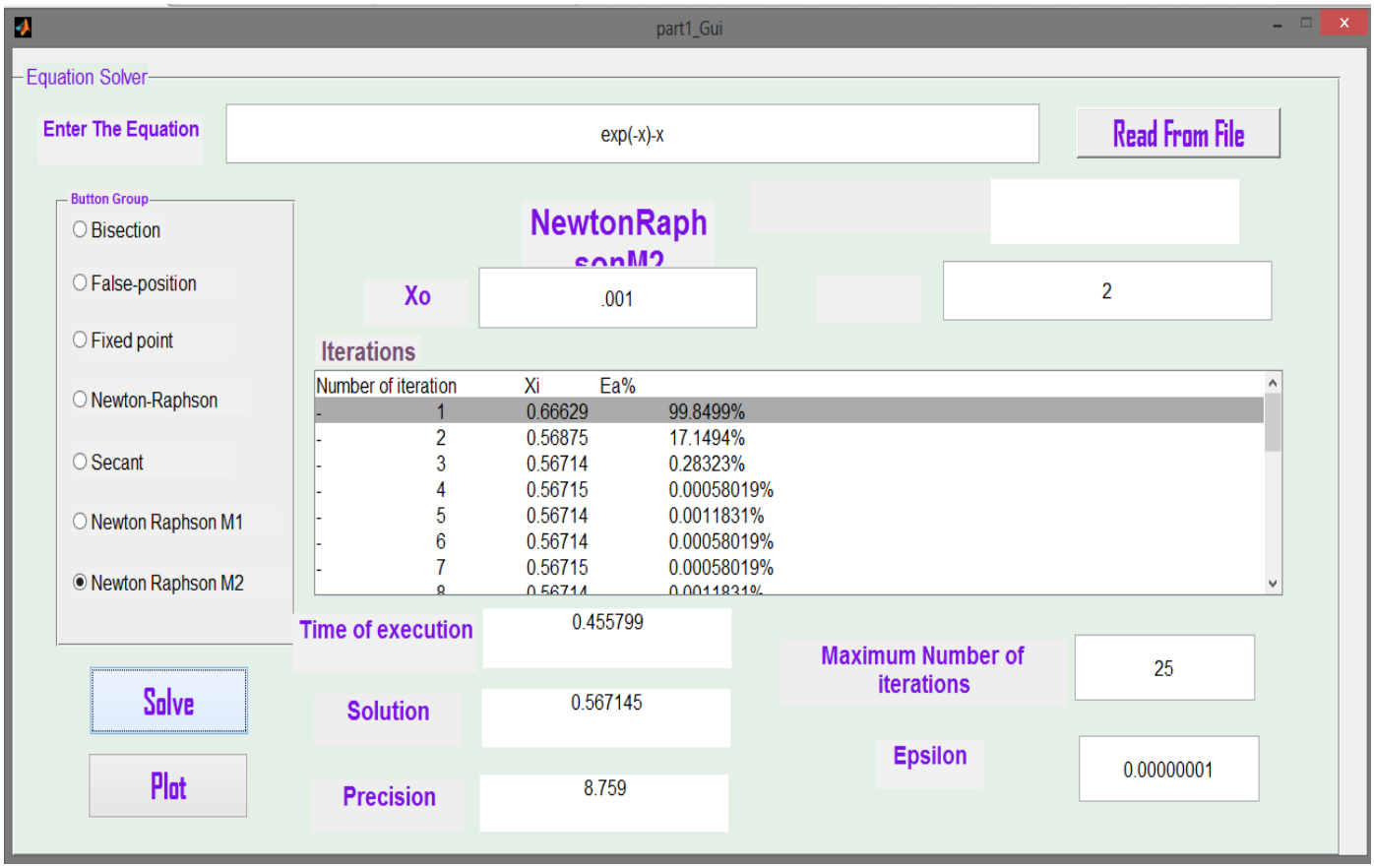
- In Newton Raphson Modified 1

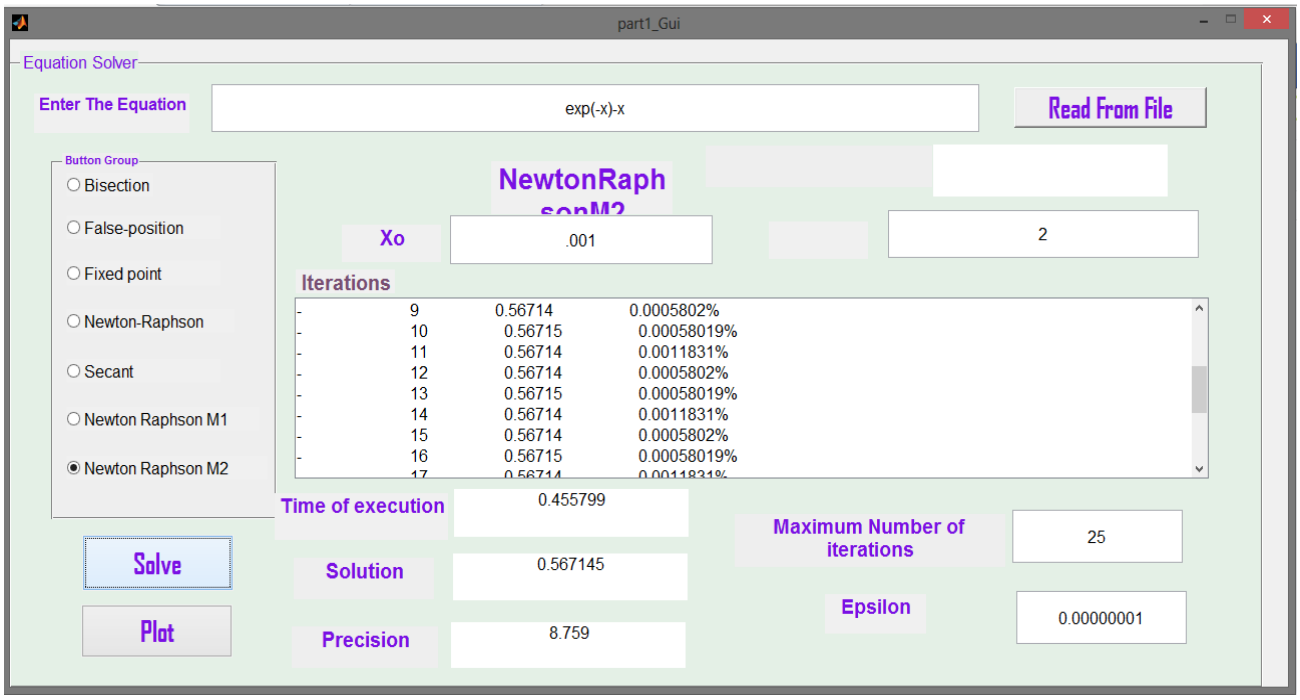


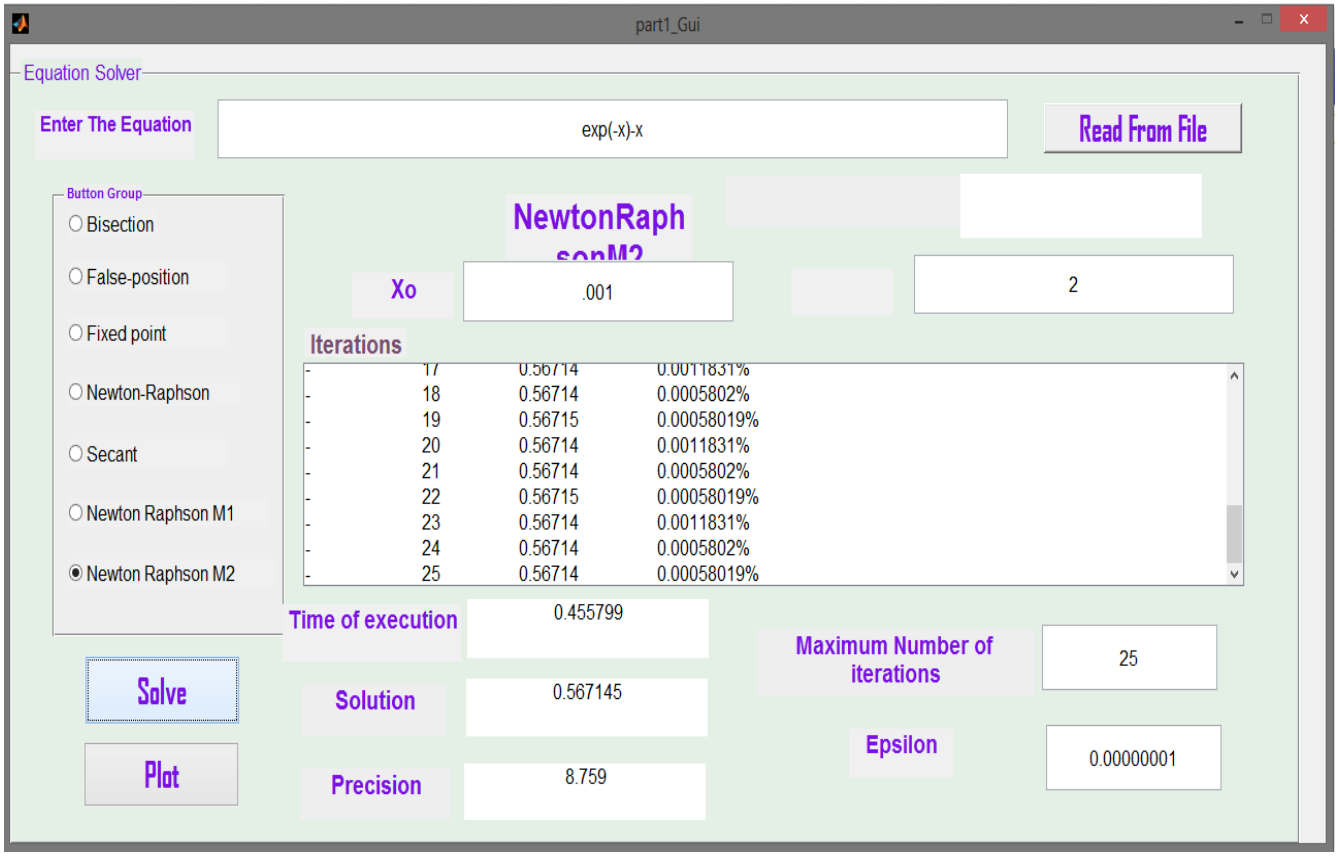




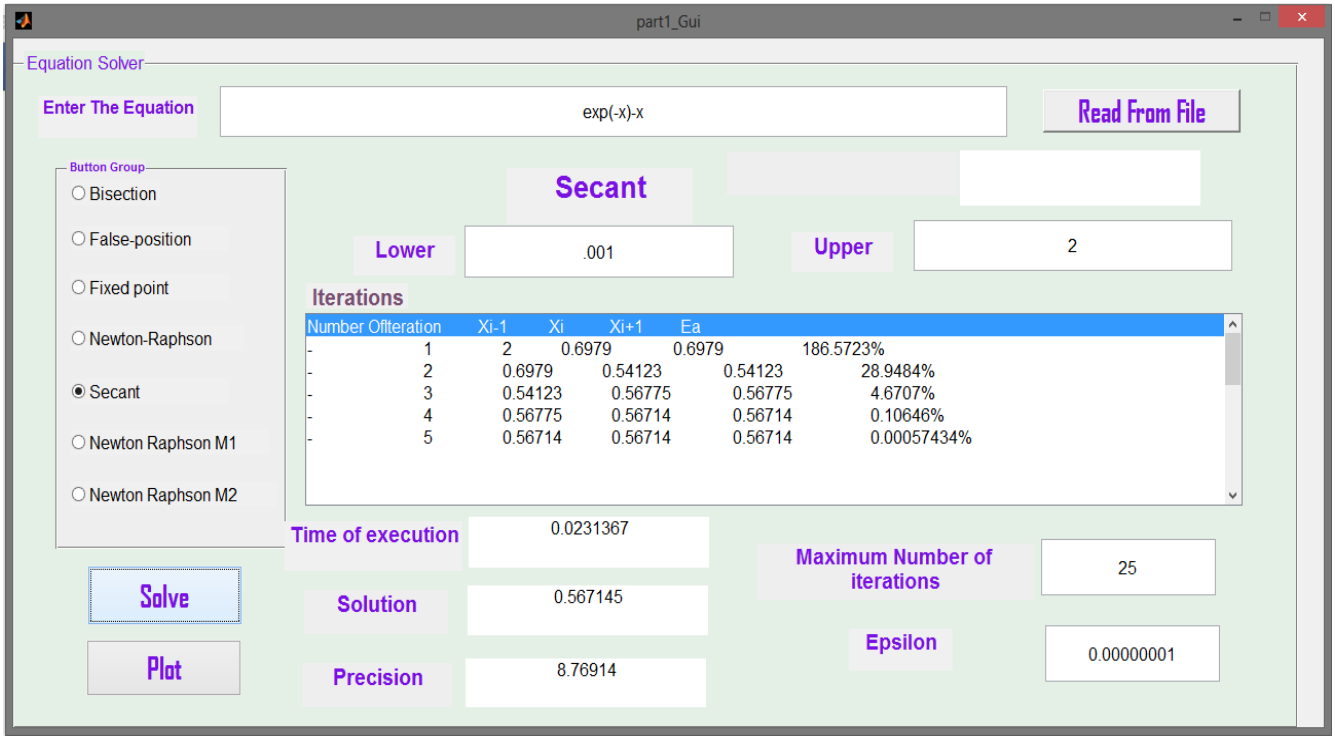
- In Newton Raphson Modified 2




part1_Gui
Equation Solver
Enter The Equation
exp(-x)-x
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaph
Xo
.001
2
Iterations
9 0.56714 0.0005802%
10 0.56715 0.00058019%
11 0.56714 0.0011831%
12 0.56714 0.0005802%
13 0.56715 0.00058019%
14 0.56714 0.0011831%
15 0.56714 0.0005802%
16 0.56715 0.00058019%
Time of execution
0.455799
Maximum Number of iterations
25
Solve
Solution
0.567145
Epsilon
0.00000001
Plot
Precision
8.759


part1_Gui
Equation Solver
Enter The Equation
exp(-x)-x
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
NewtonRaph
Xo
.001
2
Iterations
17 0.56714 0.0011831%
18 0.56714 0.0005802%
19 0.56715 0.00058019%
20 0.56714 0.0011831%
21 0.56714 0.0005802%
22 0.56715 0.00058019%
23 0.56714 0.0011831%
24 0.56714 0.0005802%
25 0.56714 0.00058019%
Time of execution
0.455799
Maximum Number of iterations
25
Solve
Solution
0.567145
Epsilon
0.00000001
Plot
Precision
8.759

- In Secant



Comparison and analysis of all methods:

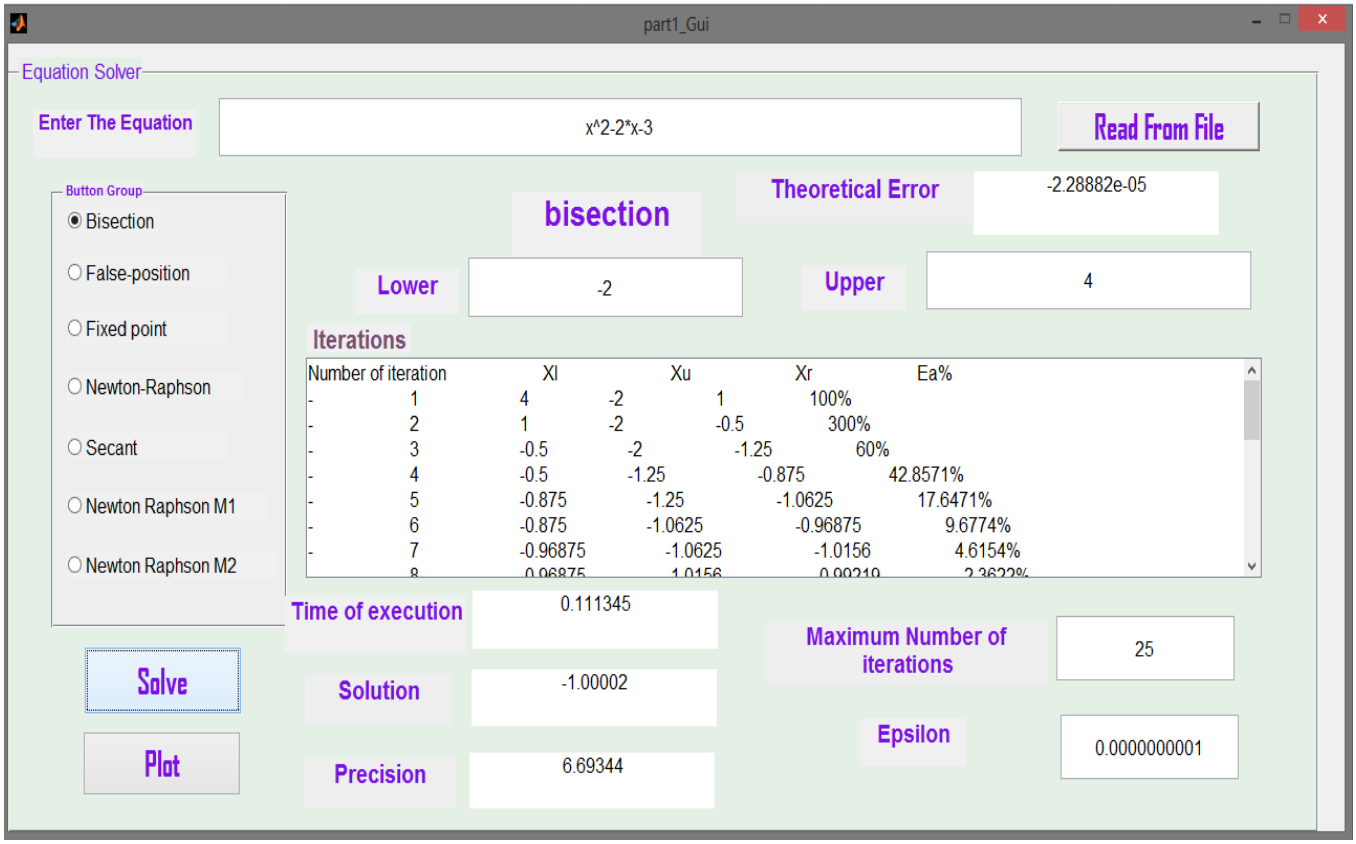
Method	Time	Root	Precision	Iteration	Relative Error
Bisection	0.112694	0.567142	8.61169	19	0.0006722%
False Position	0.0116607	0.567145	8.76914	5	0.00057434%
Fixed Point	0.092562	0.567145	8.21461	23	0.002%
Newton Raphson	0.0593299	0.567144	8.759	5	0.00058019%
Newton Raphson M 1	0.200625	0.56714	8.04647	25	0.0011831%
Newton Raphson M2	0.455799	0.567145	8.759	25	0.00058019%
Secant	0.0231367	0.567145	8.76914	5	0.00057434%

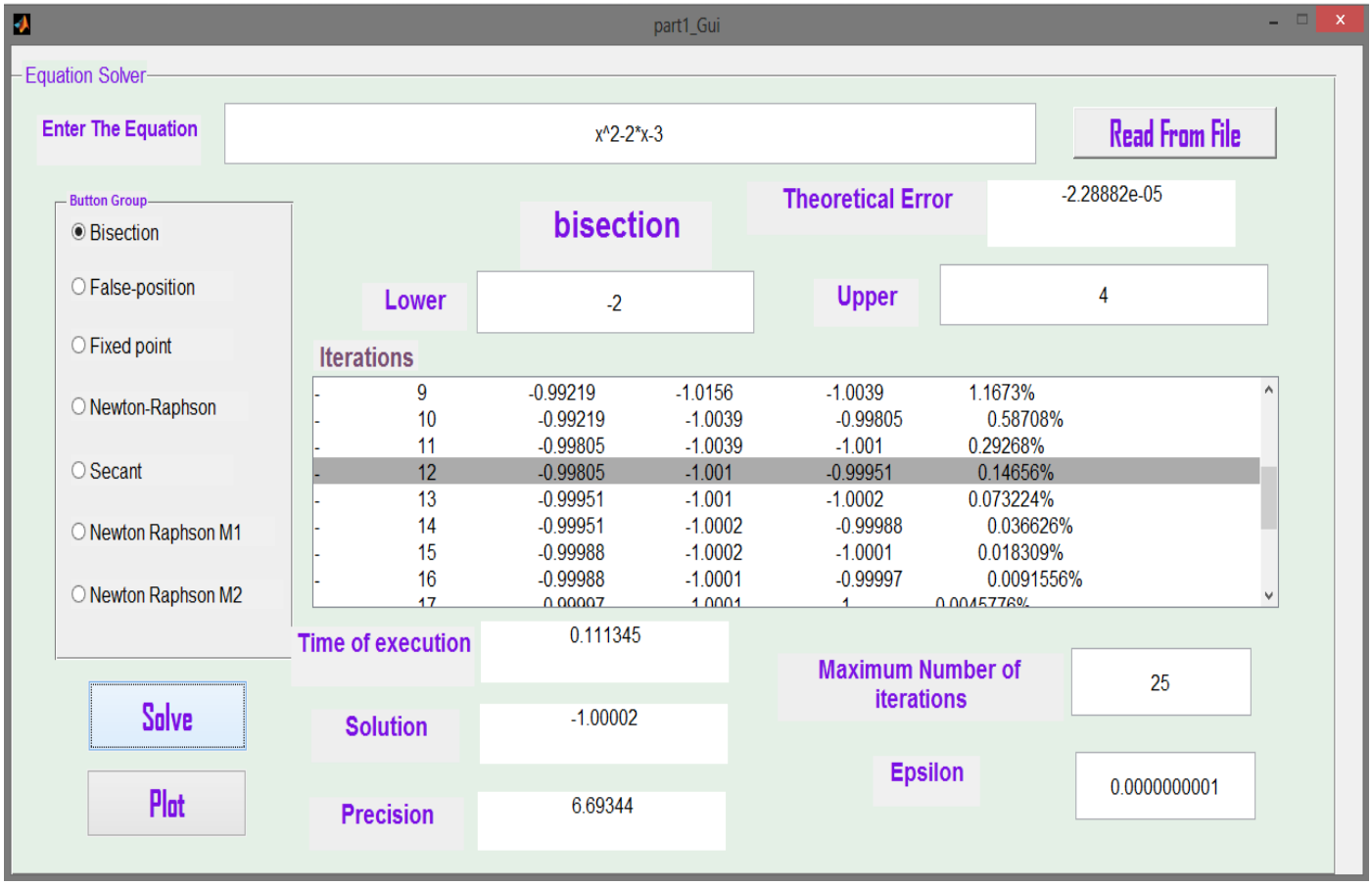
We can see that false Position take the least time of execution and it got the accurate root , false Position ,Secant and Newton Raphson take the least number of iteration and precisions are almost equal in every method so we can say that false position is the accurate and the fastest method in this example.

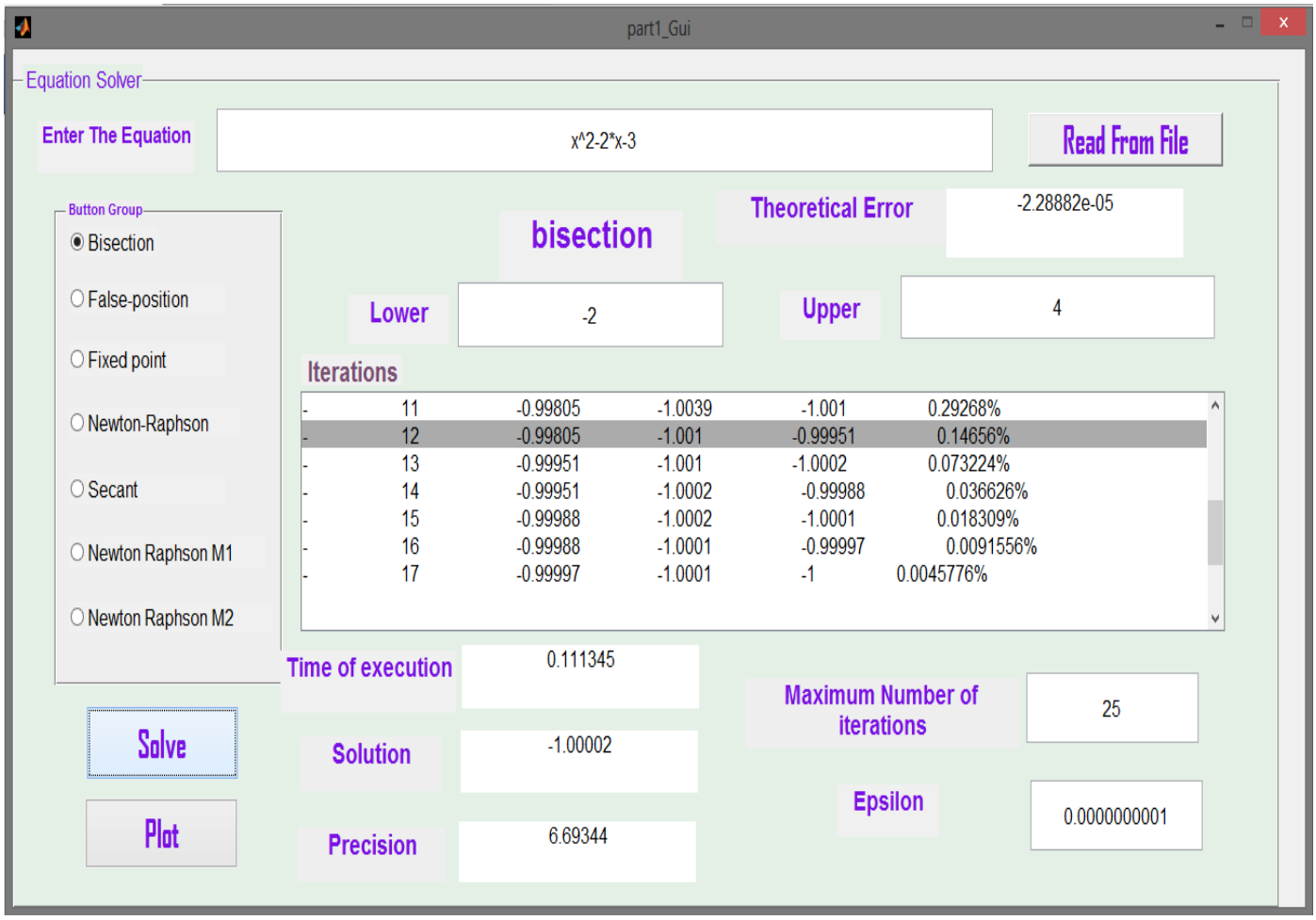
Example 4:

Equation = “x^2-2*x-3”:

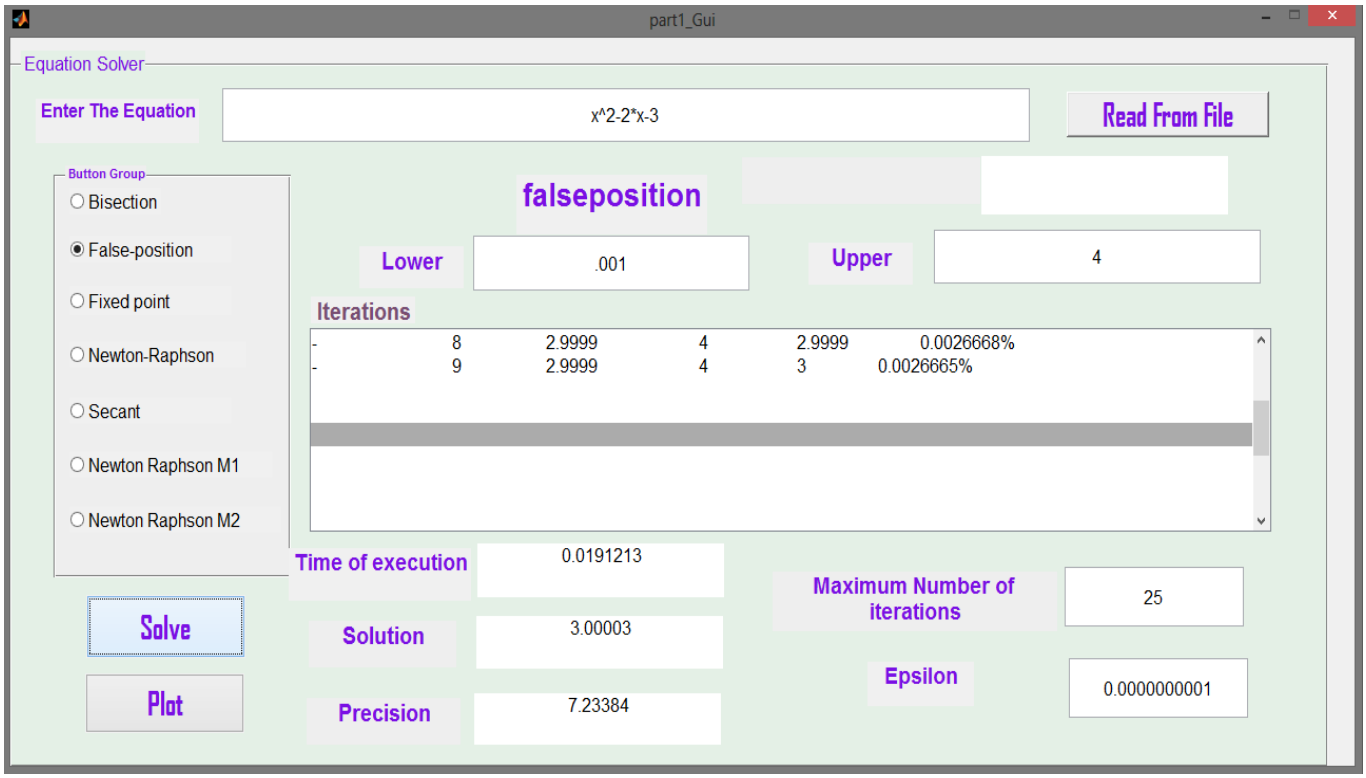
- In Bisection method



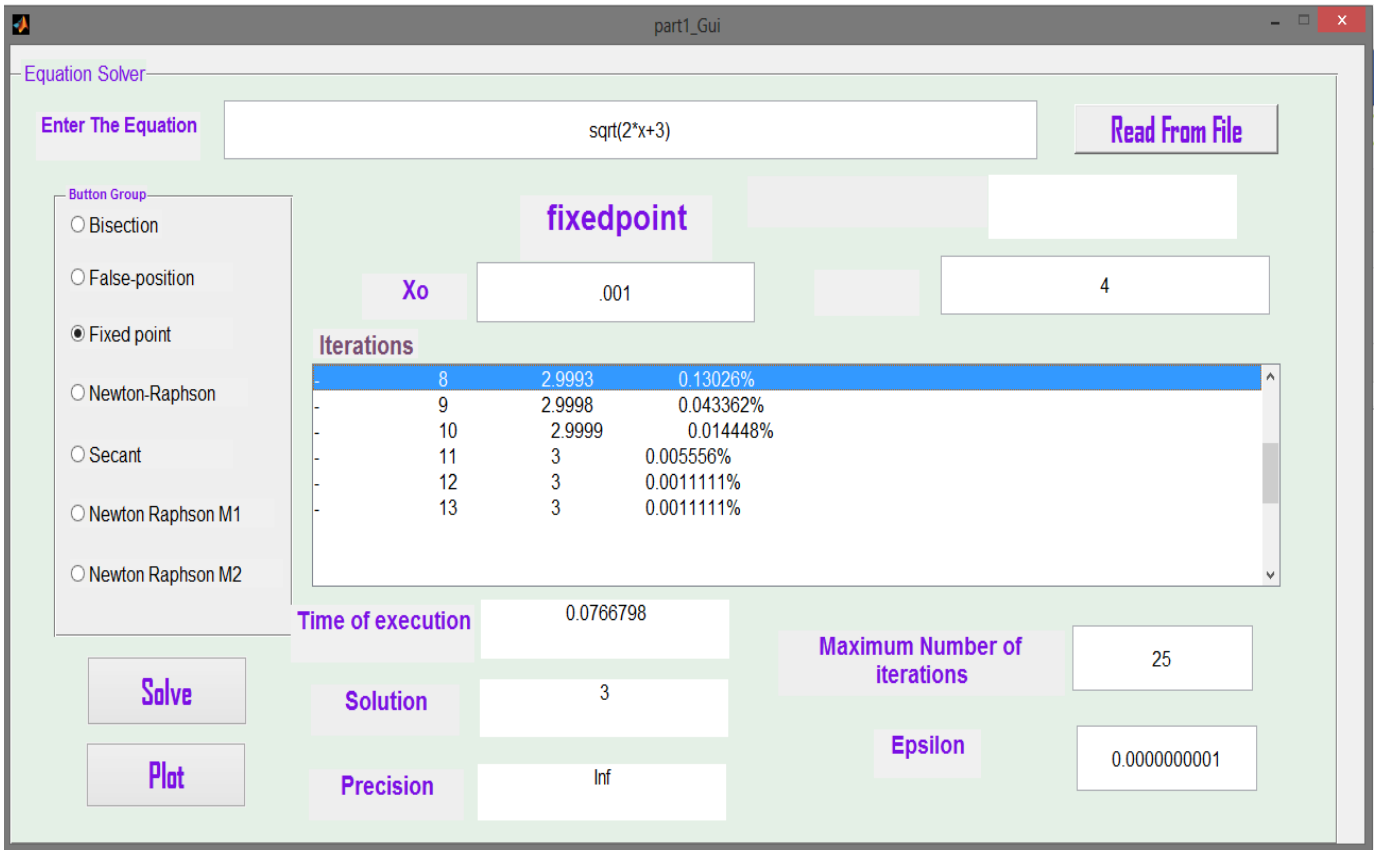

part1_Gui
Equation Solver
Enter The Equation
x^2-2*x-3
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
bisection
Theoretical Error
-2.28882e-05
Lower
-2
Upper
4
Iterations
9 -0.99219 -1.0156 -1.0039 1.1673%
10 -0.99219 -1.0039 -0.99805 0.58708%
11 -0.99805 -1.0039 -1.001 0.29268%
12 -0.99805 -1.001 -0.99951 0.14656%
13 -0.99951 -1.001 -1.0002 0.073224%
14 -0.99951 -1.0002 -0.99988 0.036626%
15 -0.99988 -1.0002 -1.0001 0.018309%
16 -0.99988 -1.0001 -0.99997 0.0091556%
Time of execution
0.111345
Maximum Number of iterations
25
Solve
Solution
-1.00002
Epsilon
0.0000000001
Plot
Precision
6.69344


part1_Gui
Equation Solver
Enter The Equation
x^2-2*x-3
Read From File
Button Group
Bisection
False-position
Fixed point
Newton-Raphson
Secant
Newton Raphson M1
Newton Raphson M2
bisection
Theoretical Error
-2.28882e-05
Lower
-2
Upper
4
Iterations
- 11 -0.99805 -1.0039 -1.001 0.29268%
- 12 -0.99805 -1.001 -0.99951 0.14656%
- 13 -0.99951 -1.001 -1.0002 0.073224%
- 14 -0.99951 -1.0002 -0.99988 0.036626%
- 15 -0.99988 -1.0002 -1.0001 0.018309%
- 16 -0.99988 -1.0001 -0.99997 0.0091556%
- 17 -0.99997 -1.0001 -1 0.0045776%
Time of execution
0.111345
Maximum Number of iterations
25
Solve
Solution
-1.00002
Epsilon
0.0000000001
Plot
Precision
6.69344

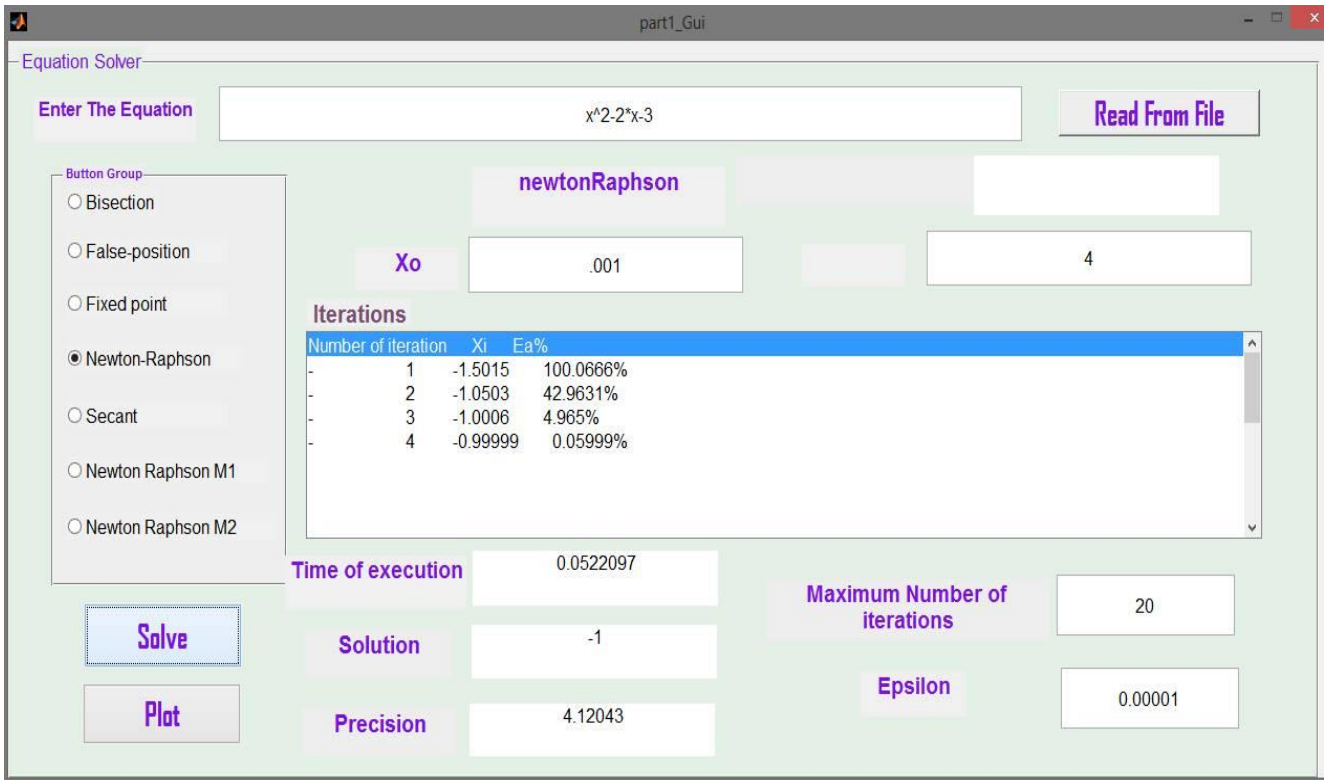
- False Position



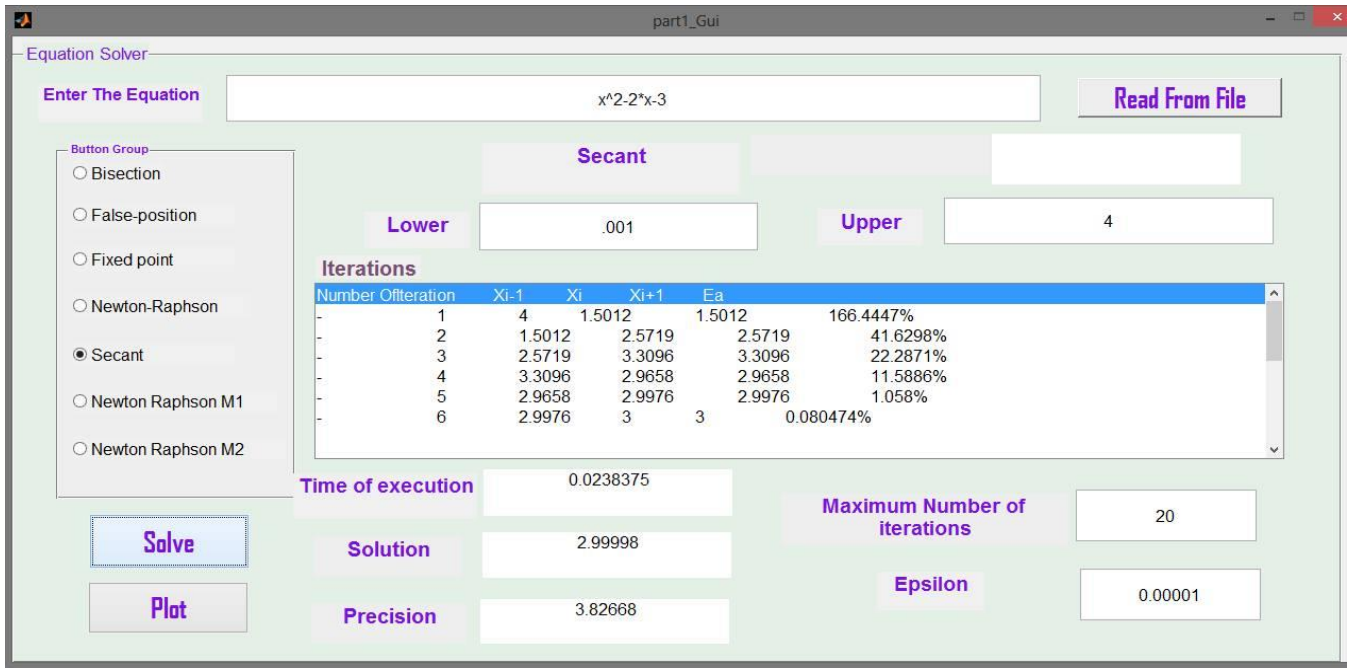
- In fixed point



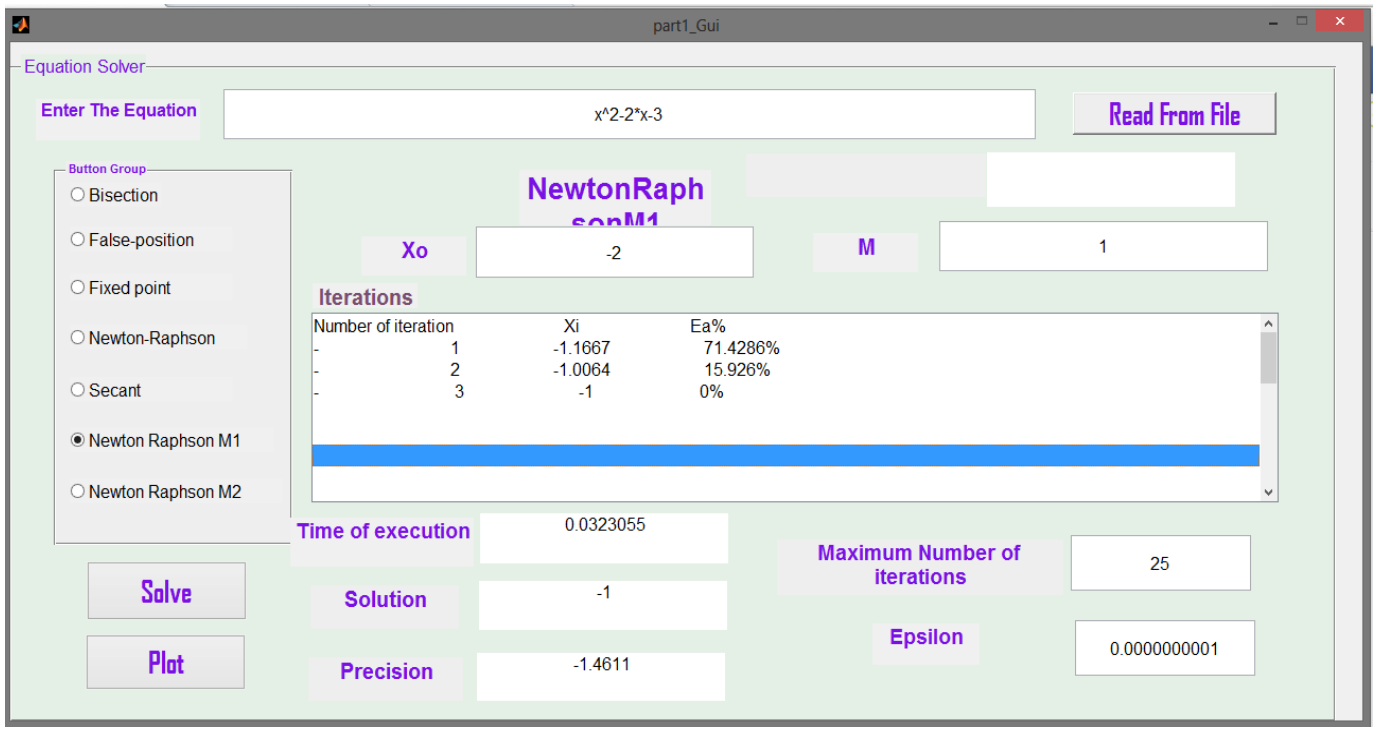
- In Newton Raphson



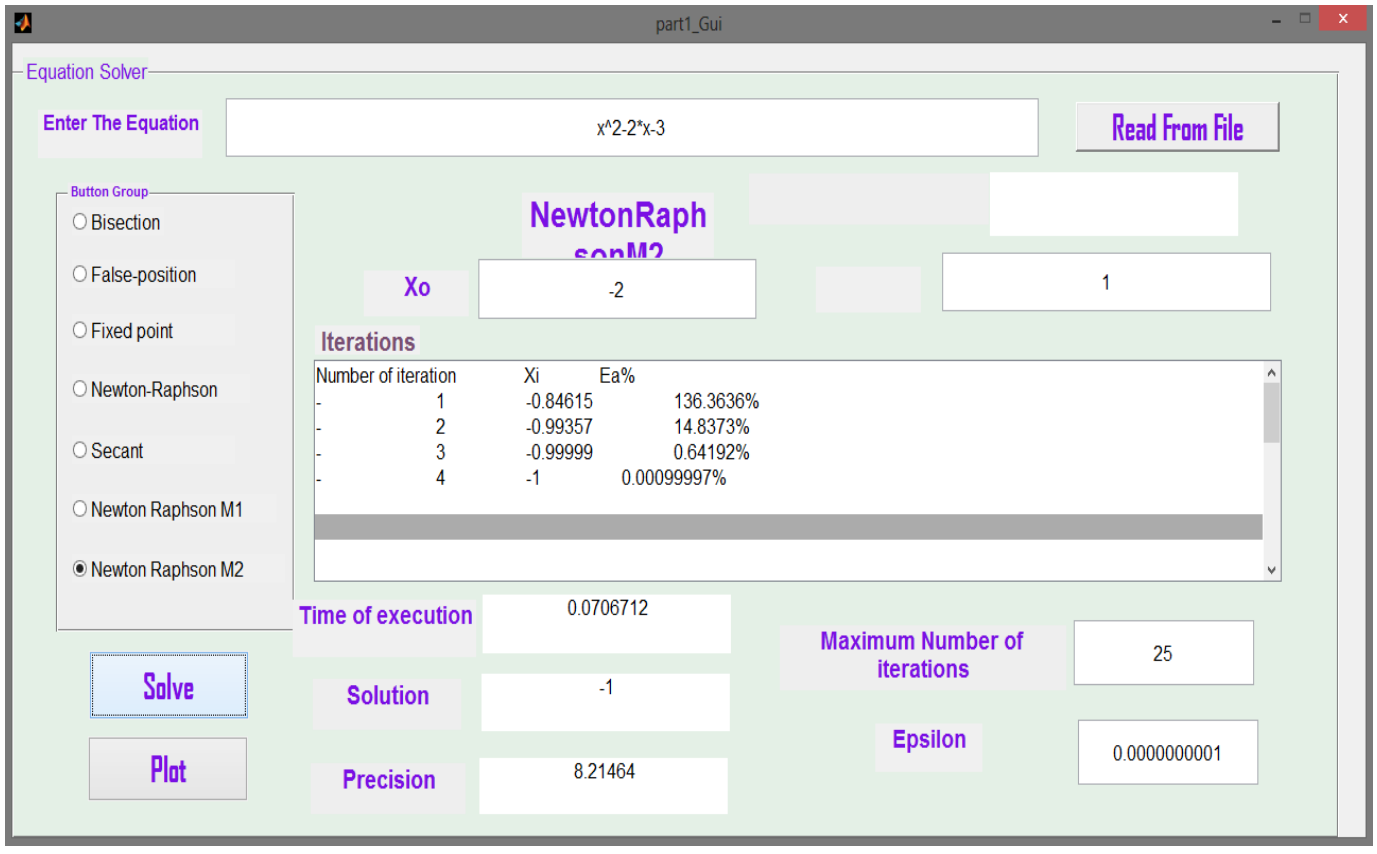
- In Secant



- In Newton Raphson Modified1



- In Newton Raphson Modified2



Comparison and analysis of all methods:

Method	Time	Root	Precision	Iteration	Relative Error
Bisection	0.111345	-1.00002	6.69344	17	0.0045776%
False Position	0.019123	3.00003	7.23384	9	0.002665%
Fixed Point	0.0766798	3	inf	13	0.0011111%
Newton Raphson	0.0606904	-1	4.12043	4	0.05999%
Newton Raphson M 1	0.0323055	-1	-1.4611	3	0%
Newton Raphson M2	0.070612	-1	8.21464	4	0%
Secant	0.0260523	2.99998	3.82668	6	0.080474%

We can see that false Position take the least time of execution ,Newton Raphson Modified 1 and 2 got the accurate root , Newton Raphson Modified 1 take the least number of iteration and the least precision is in Newton Raphson Modified 1 method so we can say that Newton Raphson Modified 1 is the accurate and the fastest method in this example.

Conclusion:

For Bracketing Method:

- They always converge but bisection is slower but the false position converge slowly when the function is too sloppy like X^10 -1.
- The bisection converges slowly and get wrong results when f(xl) is close to zero when xl is close to the exact root.

For open methods:

- Open method converge faster than bracketing methods but these methods can diverge such as Newton Raphson diverge when the equation is sloppy like x^10-1 but it converge faster when x0 is closer to the exact root.
- Newton Raphson converges quadratically unless ther is multiple roots.
- Newton Raphson takes less iteration than fixed point.
- Secant method converges faster than False Position method.

PART TWO

Objectives:

The aim of this assignment is to compare and analyze the behavior of numerical methods studied in class {Gaussian-elimination, LU decomposition, and Gaussian- Jordan}.

Specification:

The program must contain the following features:

- An interactive GUI that enables the user to enter set of linear equations. Reading from files must be available as well.
- A way to choose a method to solve the given set of linear equations.
- A way to enter the needed parameters if any;
- The answer for the chosen method indicating the execution time, solution, precision, and the iterations (if applicable).

DESIGN SPECIFICATIONS

There are two ways to enter the equations

Manual way :

- Enter the number of equations.
- Press ‘ok’ button.
- Enter the coefficients and the result of every equation.
- Press ‘Enter’ button.
- Enter the next equation by the same way in the same edit text then Press ‘Enter’ and so on.
- Press ’Solve’ button to solve the system of the equation.
- Your solutions will appear in the list box and you can get the execution time.

Read File way:

- Press ‘ReadFile’ button.
- The number of equation appears in ‘#equation’ edit text.
- Press ‘Get from file’ to get the coefficients and the result from the file.
- The data appears in its edit text.
- Press ’Solve’ button to solve the system of the equations.
- Your solutions appear in the list box and you can get the execution time.

DATA STRUCTURE USED AND HOW HELPFUL WAS YOUR CHOICE.

1- We have used arrays as a data structure:

We have used two dimensional arrays to solve the system of linear equations, these two dimensional arrays are augmented arrays. So, their entries are the coefficients of the equations and the last column in each of them is the resulted coefficient.

2- We have also used Strings as another data structure to handle Parsing and validating operations,

The program takes the user input as a string to facilitate different validation and other operations.

3- We have also used Cell Array as another data structures, we have used it to hold the solution for the system, we have also used it in reading from file operation as every cell of the array holds a line from the input file.

ANALYSIS FOR THE BEHAVIOR OF DIFFERENT EXAMPLES USING THE ANALYSIS TEMPLATE, AND YOUR CONCLUSION ABOUT THE BEHAVIOR OF EACH METHOD (AT LEAST THREE EXAMPLES).

Method	Five Equations	Four Equations	Three Equations	Two Equations
Gauss Elimination	0.00683474	0.00444223	0.00540778	0.00355428
LU_Decomposition	0.00678178	0.00446645	0.00514176	0.00343039
Gaussian_Jordan	0.00679123	0.00478132	0.00549071	0.00388434
Gaussian_Elimination (Partial Pivoting)	0.00675756	0.00425339	0.00504652	0.00353088
LU_Decomposition (Partial Pivoting)	0.00673499	0.00345247	0.00501901	0.00308218

If the input is a system of five linear equations:

The execution time of each method:

Gauss Elimination: it takes 0.00683474

Gauss Elimination (Partial Pivoting): it takes 0.00675756

Gaussian_Jordan: it takes 0.00679123

LU_Decomposition: it takes 0.00678178

LU_Decomposition (Partial Pivoting): it takes 0.00673499

If the input is a system of four linear equations:

The execution time of each method:

Gauss Elimination: it takes 0.00444223

Gauss Elimination (Partial Pivoting: it takes 0.00425339

Gaussian_Jordan: it takes 0.00478132

LU_Decomposition: it takes 0.00467992

LU_Decomposition (Partial Pivoting): it takes 0.00345247

If the input is a system of three linear equations:

The execution time of each method:

Gauss Elimination: it takes 0.00540778

Gauss Elimination (Partial Pivoting): it takes 0.00504652

Gaussian_Jordan: it takes 0.0054907

LU_Decomposition: it takes 0.00514176

LU_Decomposition (Partial Pivoting): it takes 0.00501901

If the input is a system of two linear equations:

The execution time of each method:

Gauss Elimination: it takes 0.00355428

Gauss Elimination (Partial Pivoting): it takes 0.00353088

Gaussian_Jordan: it takes 0.00388434

LU_Decomposition: it takes 0.0034303

LU_Decomposition (Partial Pivoting): it takes 0.00308218

Conclusion:

Gaussian-Jordan Takes Time more than other methods because it repeats gauss elimination n times so the order of this method is O(n^3).

NUMERICAL METHODS

Gauss Elimination:

It's a numerical method to solve a system of linear equations with any number of coefficients and variables (Solve Ax = b).

It consists of two techniques:

- Forward elimination

  Reduces Ax = b to an upper triangular system Tx = b'

- Back substitution

  Can then solve T x = b' for x

Pseudo Code:

```
start
do for k <- 1 to n-1
do for i <- k to n-1
m <- B(i+1,k)/B(k,k)
do for j <- 1 to  t
B(i+1,j) <- B(i+1,j) - m * B(k,j)
End for
End for
End for
i <- n
x(i,1) <- B(i,t)/B(i,i)
do for i <- n-1 to 1
s <- 0
for k <- n to i+1
s <- s + B(i, k) * x(k, 1)
end for
```

```
x(i,1) <- (B(i,t)-s) / B(i,i)
end for
end
```

Problematic functions and the reason for their misbehavior and your suggestions (if exists).

1- Division by zero

It is possible that during both elimination and back-substitution phases a division by zero can occur.

2- Large Round-off Errors

Solution: Pivoting

So, Gaussian Elimination with Partial Pivoting
And LU_Decomposition with Partial Pivoting can solve that problem by switching rows according to the largest pivot.

- Avoids division by zero.
- Reduces round off error.

Gauss Elimination (Partial Pivoting):

Similar to Gauss Elimination but it solves Gaussian's pitfalls as it:

- Avoids division by zero.
- Reduces round off error.

Pseudo Code:

```
start
do for u <- 1 to n-1
maximum <- abs(B(u, u))
r <- u
 do for i <= u+1 to n
if maximum < abs(B(i, u))
maximum  <- abs(B(i,u));
r <- i
end if
end for
row1(1,:) <- B(r,:)
row2(1,:) <- B(u,:)
B(u,:) <- row1
B(r,:) <- row2

do for k <- u+1: n
m <- B(k,u) / B(u,u)
do for j <- u to t
B(k,j) <- B(k,j) - m * B(u,j);
End for
End for
i <-n
x(i,1) <- B(i,t) / B(i,i);
do for i <- n-1 to 1
```

```
s <- 0
do for k <- n to i+1
s <- s + B(i, k) * x(k)

end for
x(i) <- (B(i,t)-s) / B(i,i);
 end   for
end
```

LU-Decomposition:

Description:

Gauss Elimination can be used to decompose [A] into [L] and [U].

Therefore, it requires the same total FLOPs as for Gauss elimination:

In the order of (proportional to) $N^3$where N is the # of unknowns.

l_{ij}values (the factors generated during the elimination step) can be stored in the lower part of the matrix to save storage. This can be done because these are converted to zeros anyway and unnecessary for the future operations.

Saves computing time by separating time-consuming elimination step from the manipulations of the right hand side.

Provides efficient means to compute the matrix inverse.

Pseudo Code:

```
function x = LUDecomposition(B)
 [n,t] = size(B)
 for k = 1 ->  n-1
 for i = k ->  n-1
 m =  B(i+1,k) / B(k,k)
 for j = 1 -> t
 B(i+1,j) = B(i+1,j) - m * B(k,j)
 endfor
 B(i+1,k) = m
 endfor
 endfor
 if B(i,i) == 0
 disp('This system has no solutions!!')
 return
 endif
 i = n x(i,1) = B(i,t) / B(i,i)
 for i = 1 -> n-1
 s = 0
 for k = 1 ->  i-1
 s = s + B(i, k) * x(k, 1)
 endfor

 if B(i,i) == 0
 disp('This system has no solutions!!')
 return
 endif
 x(i,1) = (B(i,t)-s) / B(i,i)
```

```
endfor
for i = n-1 -> 1
s = 0
for k = n -> i+1
s = s + B(i, k) * x(k, 1)
endfor
x(i,1) = (B(i,t)-s) / B(i,i)
endfor
endfor
```

Problematic functions and the reason for their misbehavior and your suggestions (if exists).

1- Division by zero

It is possible that during both elimination and back-substitution phases a division by zero can occur.

2- Large Round-off Errors

Solution: Pivoting

So, LU_Decomposition with Partial Pivoting can solve that problem by switching rows according to the largest pivot.

- Avoids division by zero.
- Reduces round off error.

LU-Decomposition (Partial Pivoting):

Description:

- Avoids division by zero.
- Reduces round off error.

Pseudo Code:

```
function x = LUPartialPivoting(A)
[n, t] = size(A);
B = A;

for u = 1: n-1
maximum = abs(B(u, u));
r = u;
for i = u+1: n
if maximum < abs(B(i, u))
maximum =abs(B(i,u));
r = i;
end
end

row1(1,:) = B(r,:);
row2(1,:) = B(u,:);
B(u,:) = row1;
B(r,:) = row2;

for k = u+1: n;
m = B(k,u)/B(u,u);
for j = u: t;
B(k,j) = B(k,j) - m * B(u,j);
end
B(k,u)=m;   %
end
```

```
i = n; x(i,1) = B(i,t) / B(i,i);

for i = 1:1:n-1;
s = 0;
for k = 1: 1: i-1;
s = s + B(i, k) * x(k);
end
x(i,1)=(B(i,t)-s)/B(i,i);
end;

for i = n-1: -1: 1;
s = 0;
for k = n: -1: i+1;
s = s + B(i, k) * x(k);
end
x(i) = (B(i,t)-s) / B(i,i);
end;

end
```

Gauss-Jordan

Description:

- It is a variation of Gauss elimination. The major differences are:
  - When an unknown is eliminated, it is eliminated from all other equations rather than just the subsequent ones.
  - All rows are normalized by dividing them by their pivot elements.
  - Elimination step results in an identity matrix.
  - Consequently, it is not necessary to employ back substitution to obtain solution.

Pseudo Code:

```
function x = GaussJordan(B)
[n,t] = size(B)
for k = 1 -> n
for i = 1 -> n
if i != k
item = B(i,k)
scale = B(k,k)
for j = k ->t
%scalling
if scale == 0
disp('This system has no solutions!!')
return
endif
```

```
B(k,j) = B(k,j) / scale
%change row
B(i,j) = B(i,j) - item * B(k,j)
endfor
endif
endfor
endfor
for u = 1 -> n
x(u) = B(u,t)
endfor
endfor
```

Problematic functions and the reason for their misbehavior and your suggestions (if exists).

If the pivot element =0, so I can't scale or normalize its row by dividing it by this pivot.

Solution:

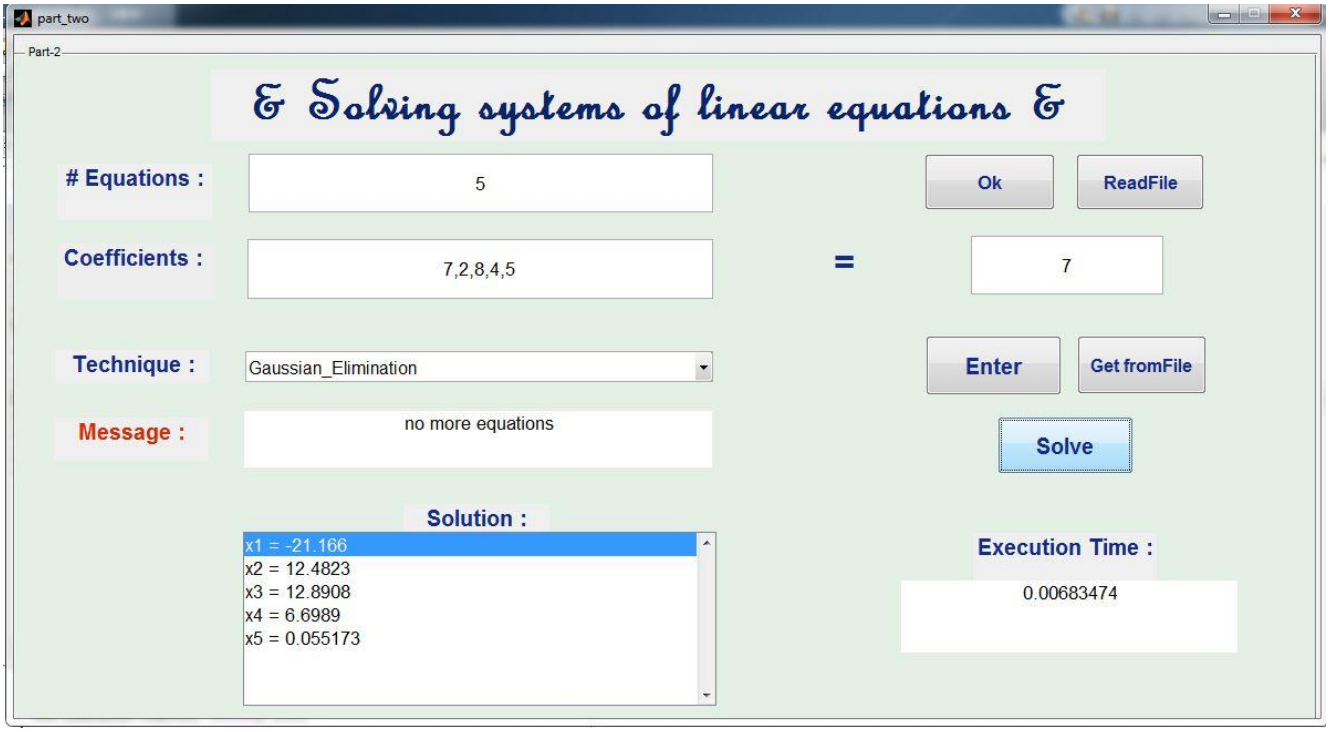
We can replace the row which have zero pivot with another one in order to have non zero pivot and can normalize the row correctly.

SAMPLE RUNS AND SNAPSHOTS FROM YOUR GUI.

In Gaussian Elimination Method:

- In case of system of five linear equations as an input:
  Input file:

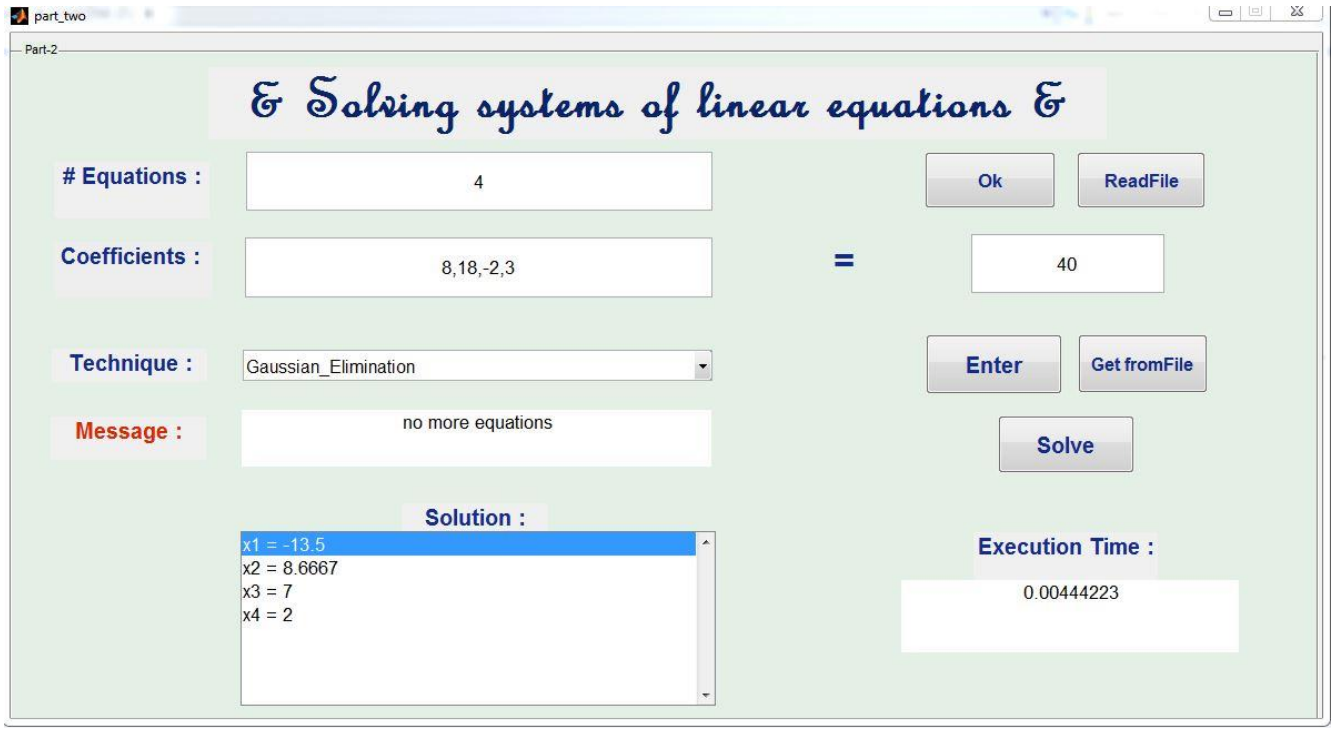
```
5
4,6,2,-2,7
3
2,0,5,-2,5
9
-4,-3,-5,4,8
10
8,18,-2,3,6
50
7,2,8,4,5
7
```



- In case of system of four linear equations as an input:

  Input file:

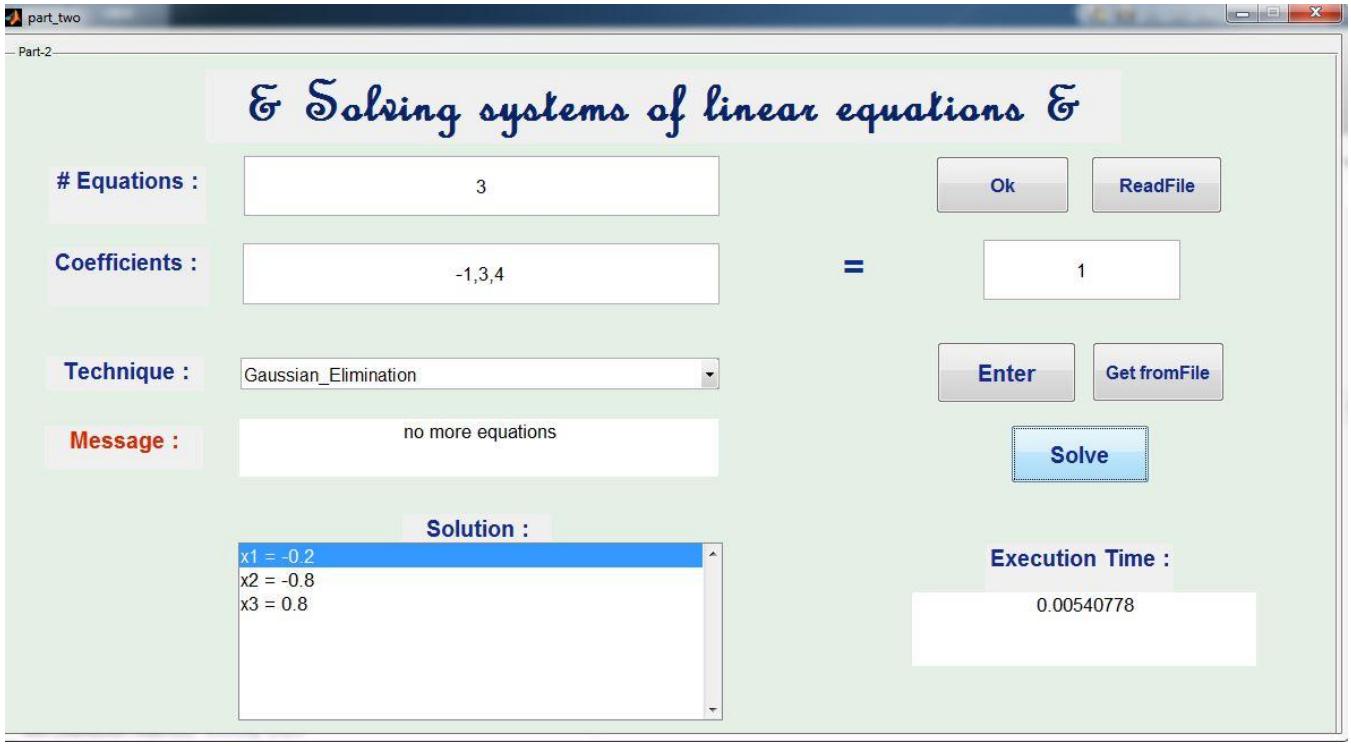
```
4
4,6,2,-2
8
2,0,5,-2
4
-4,-3,-5,4
1
8,18,-2,3
40
```



- In case of system of three linear equations as an input:

  Input file:

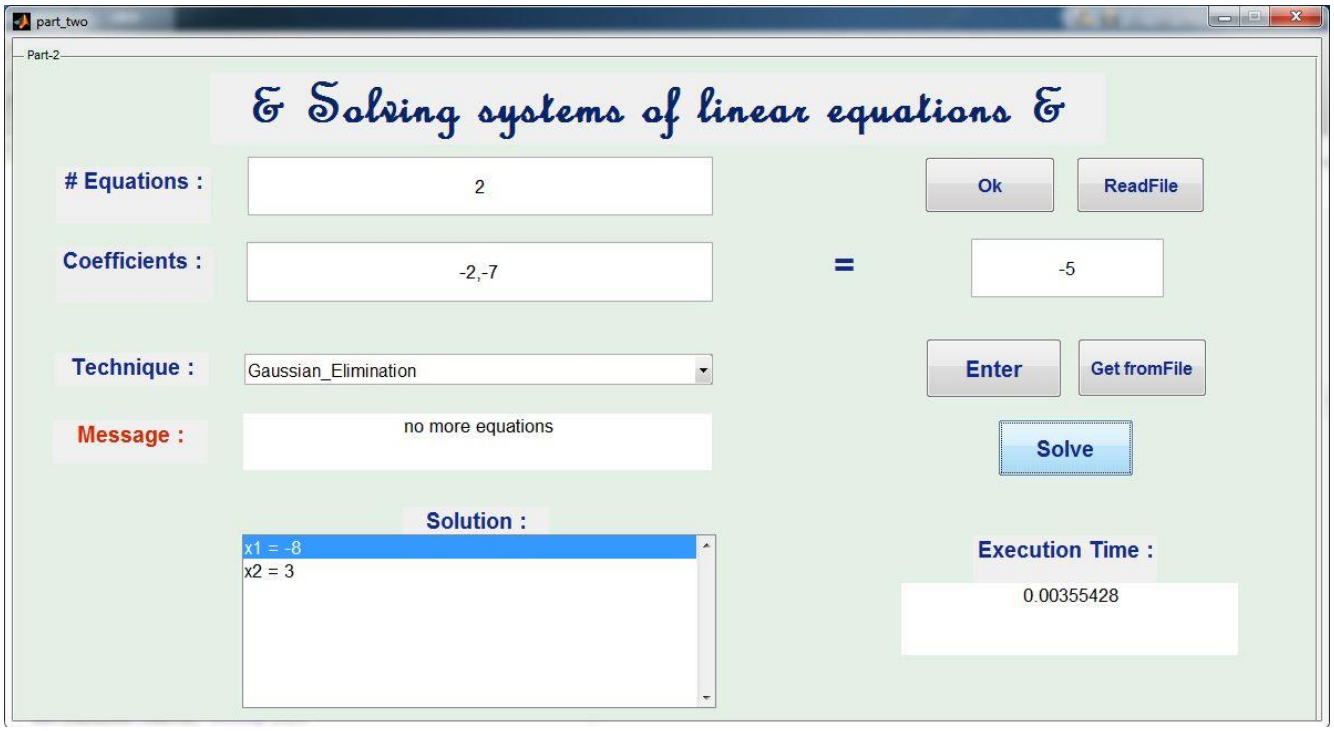
```
3
-1,1,2
1
3,-1,1
1
-1,3,4
1
```



- In case of system of two linear equations as an input:

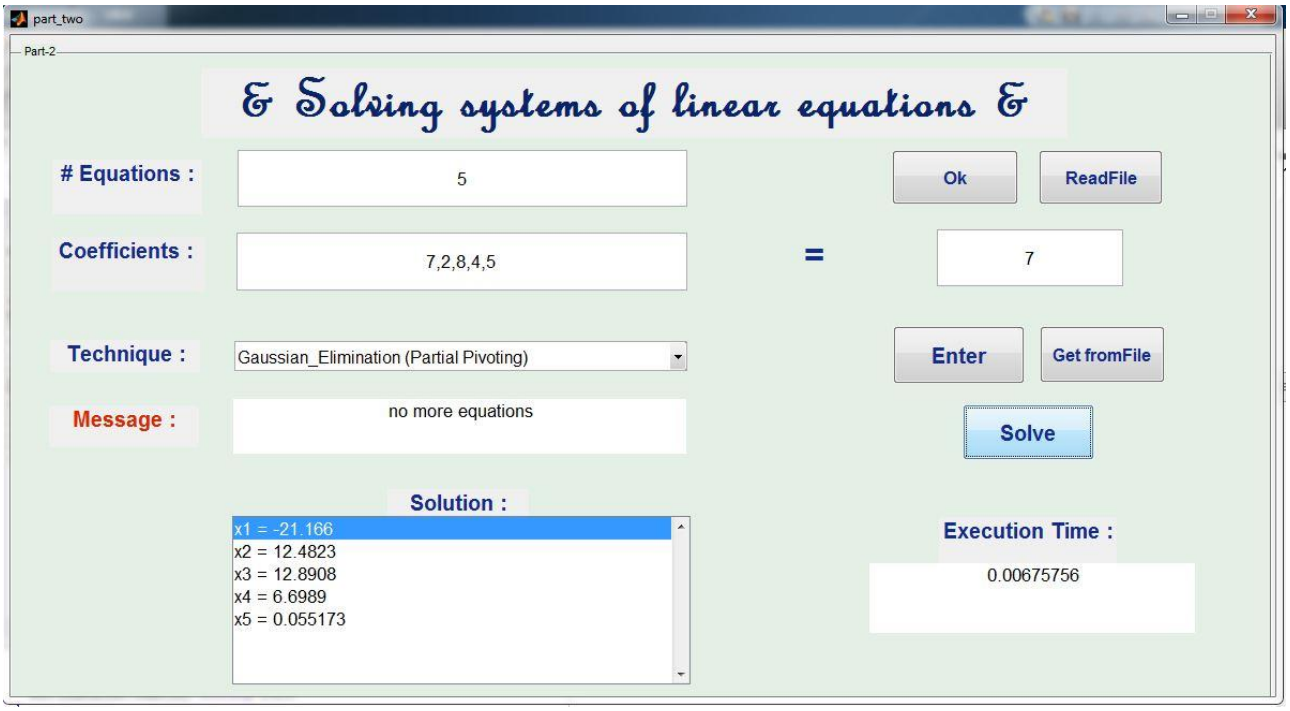
  Input file:

```
2
1,5
7
-2,-7
-5
```

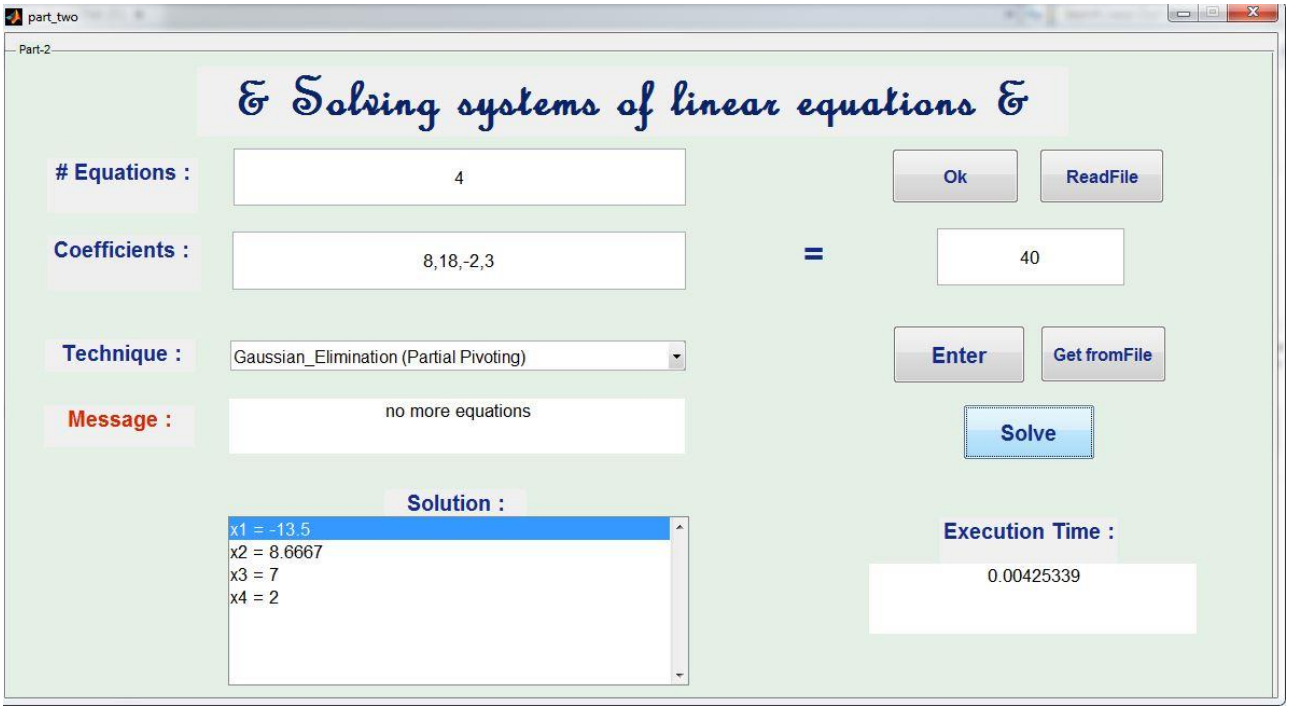


In Gaussian Elimination (Partial Pivoting) Method:

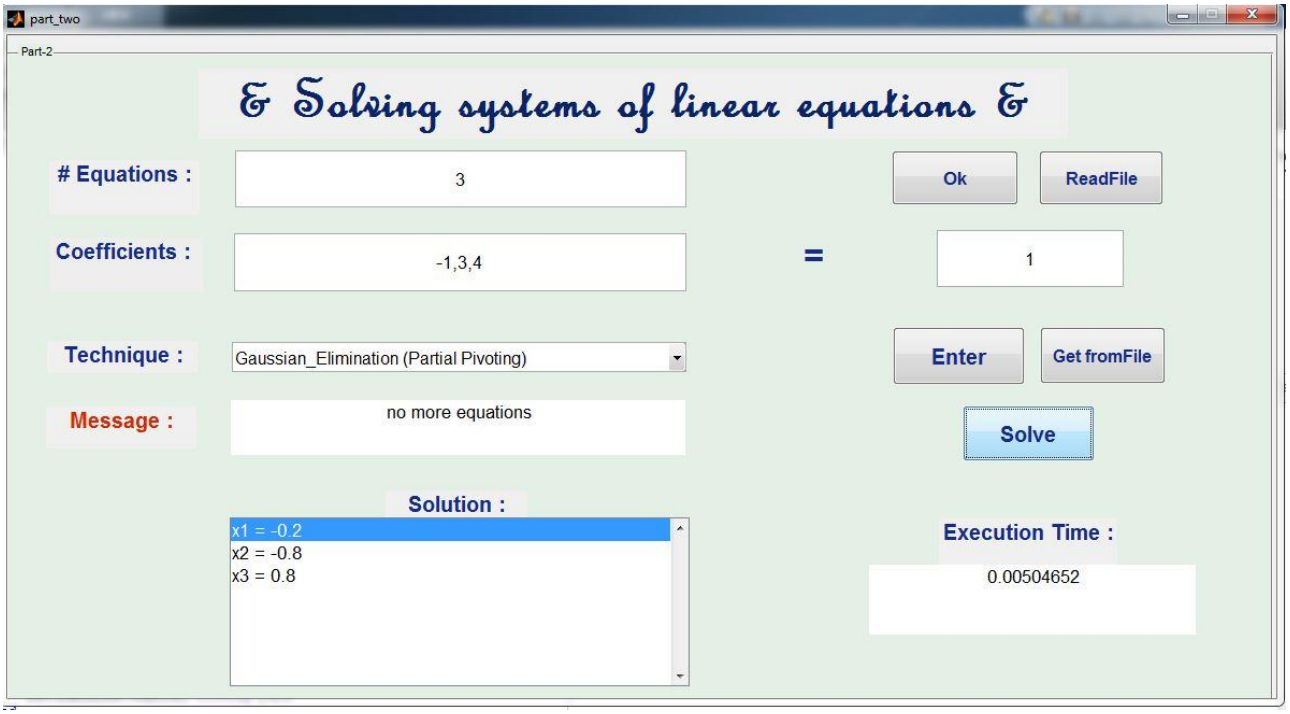
- In case of system of five linear equations as an input:



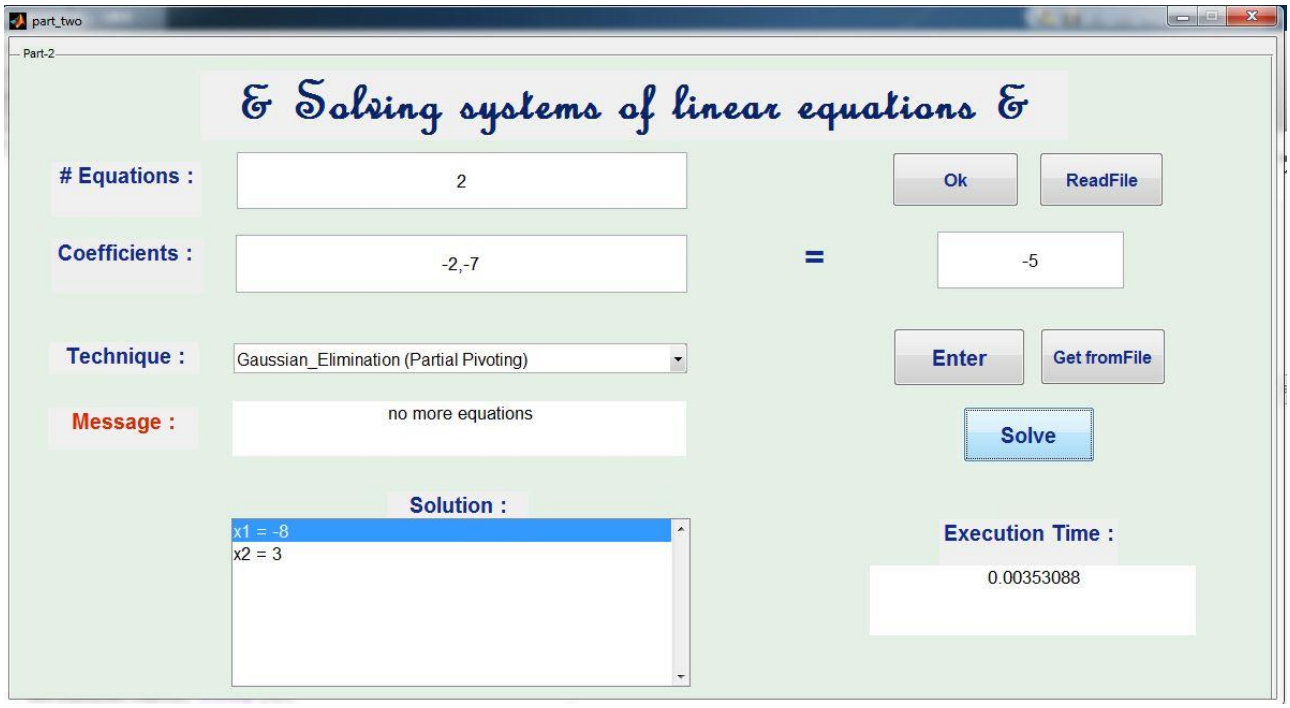
- In case of system of four linear equations as an input:



- In case of system of three linear equations as an input:

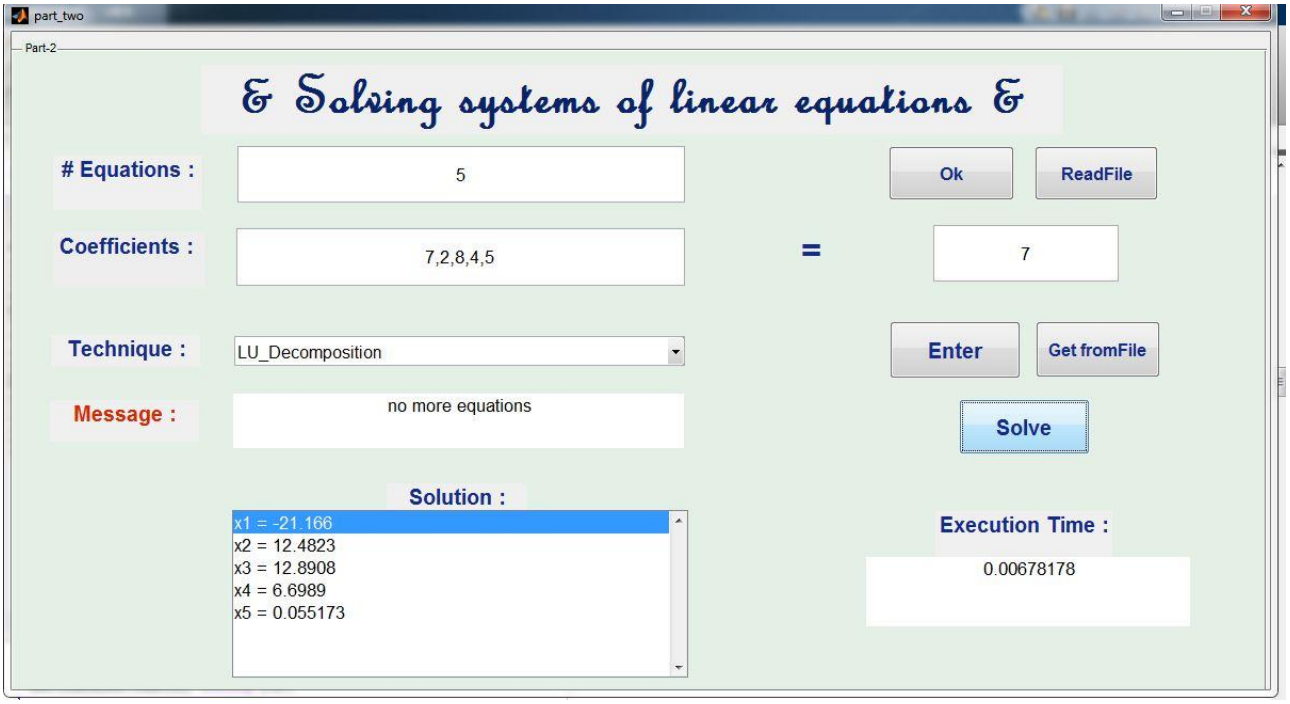


- In case of system of two linear equations as an input:

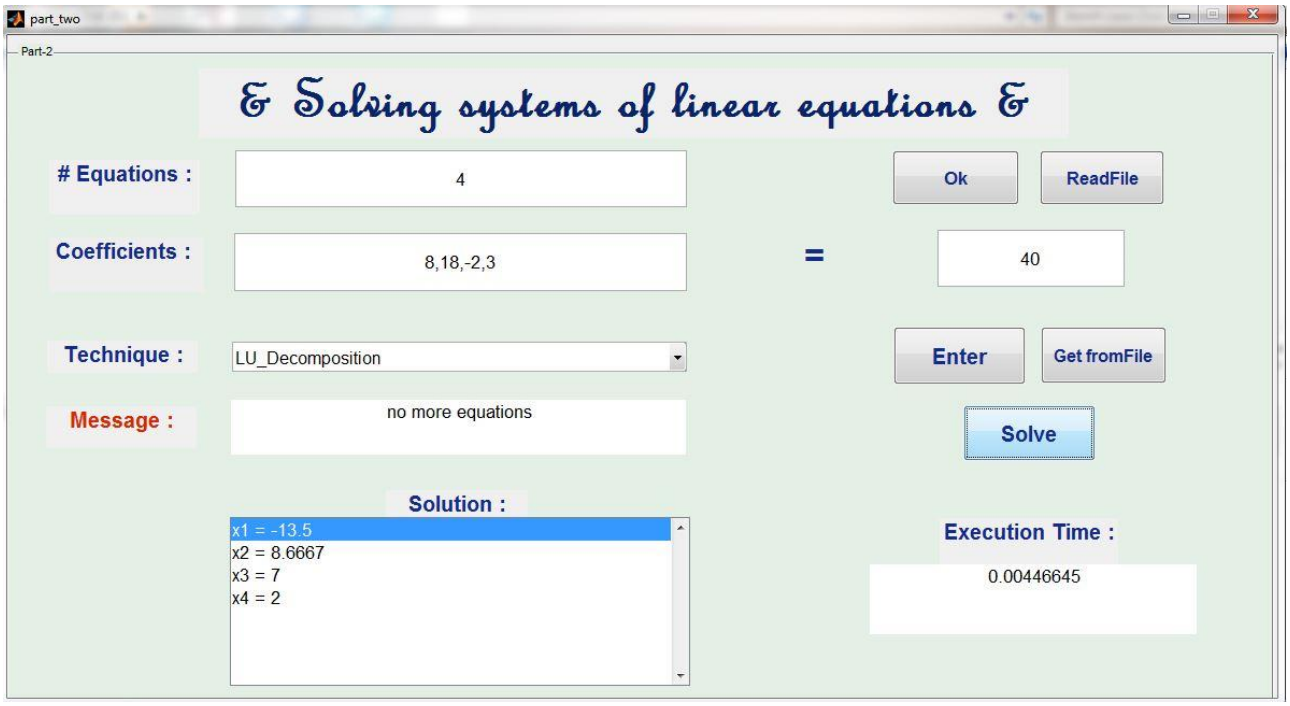


In LU-Decomposition Method:

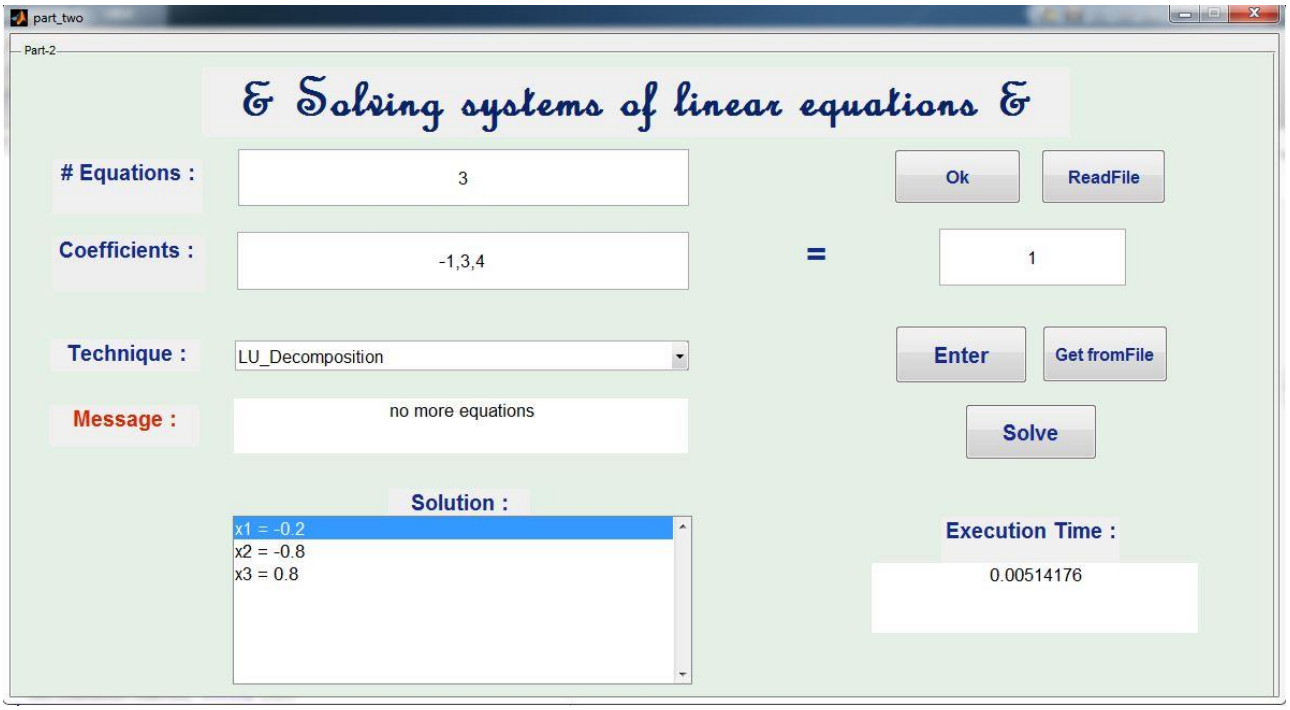
- In case of system of five linear equations as an input:



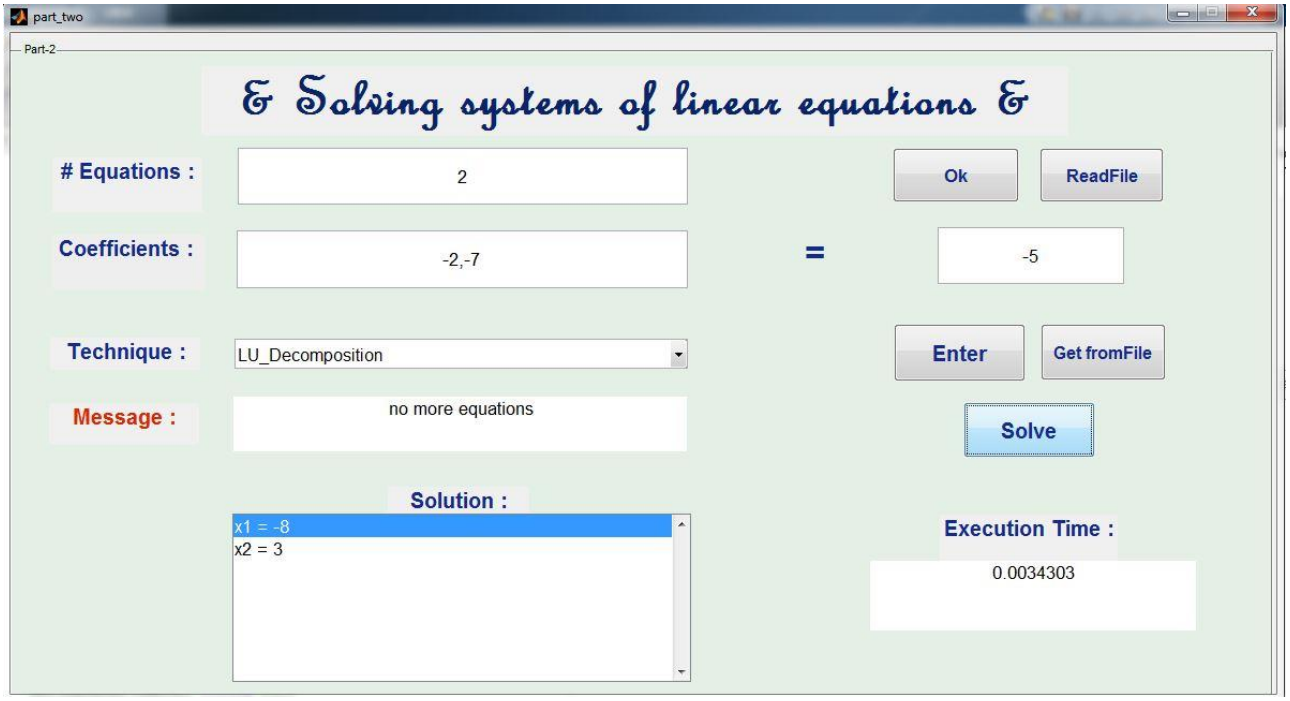
- In case of system of four linear equations as an input:



- In case of system of three linear equations as an input:

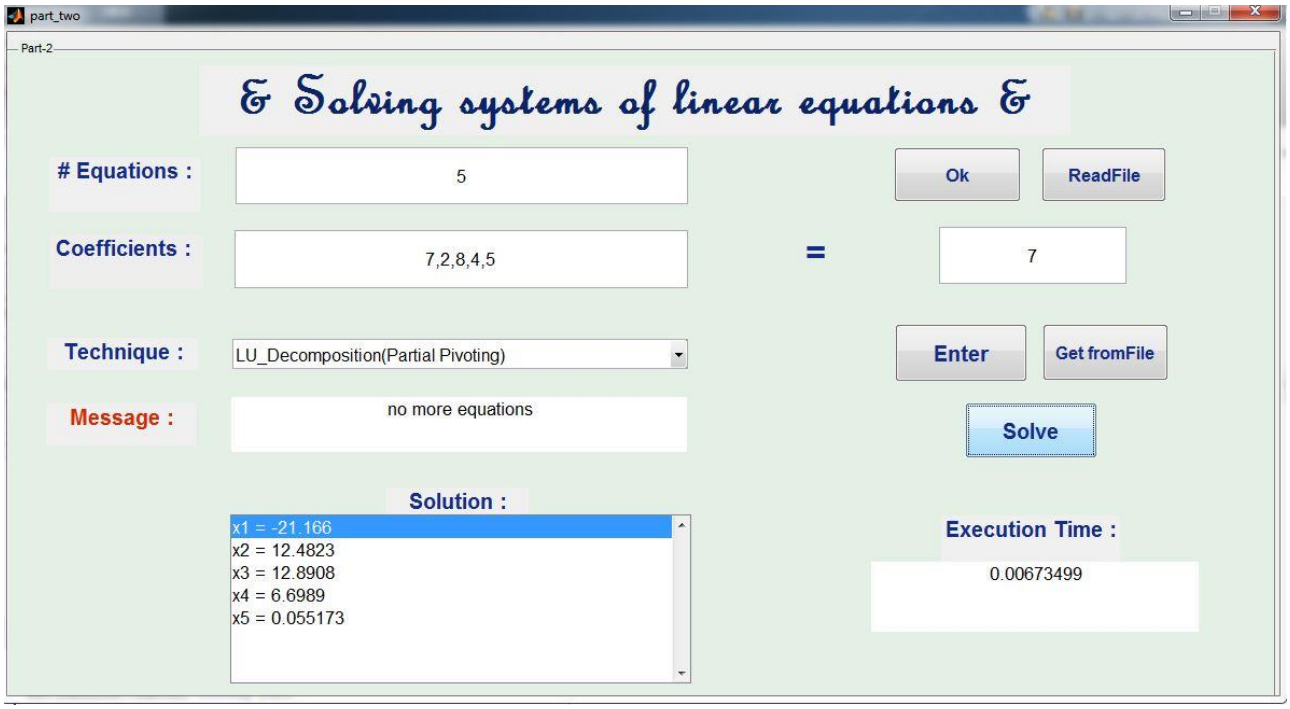


- In case of system of two linear equations as an input:

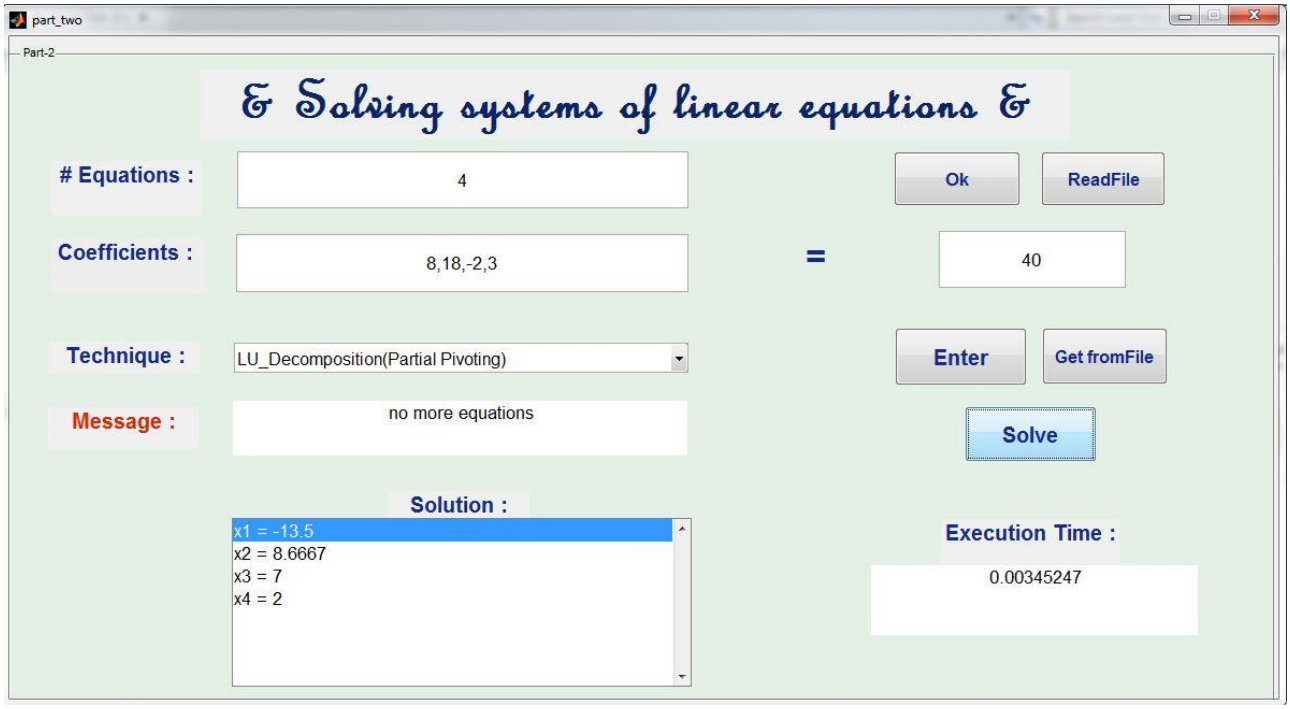


In LU-Decomposition (Partial Pivoting) Method:

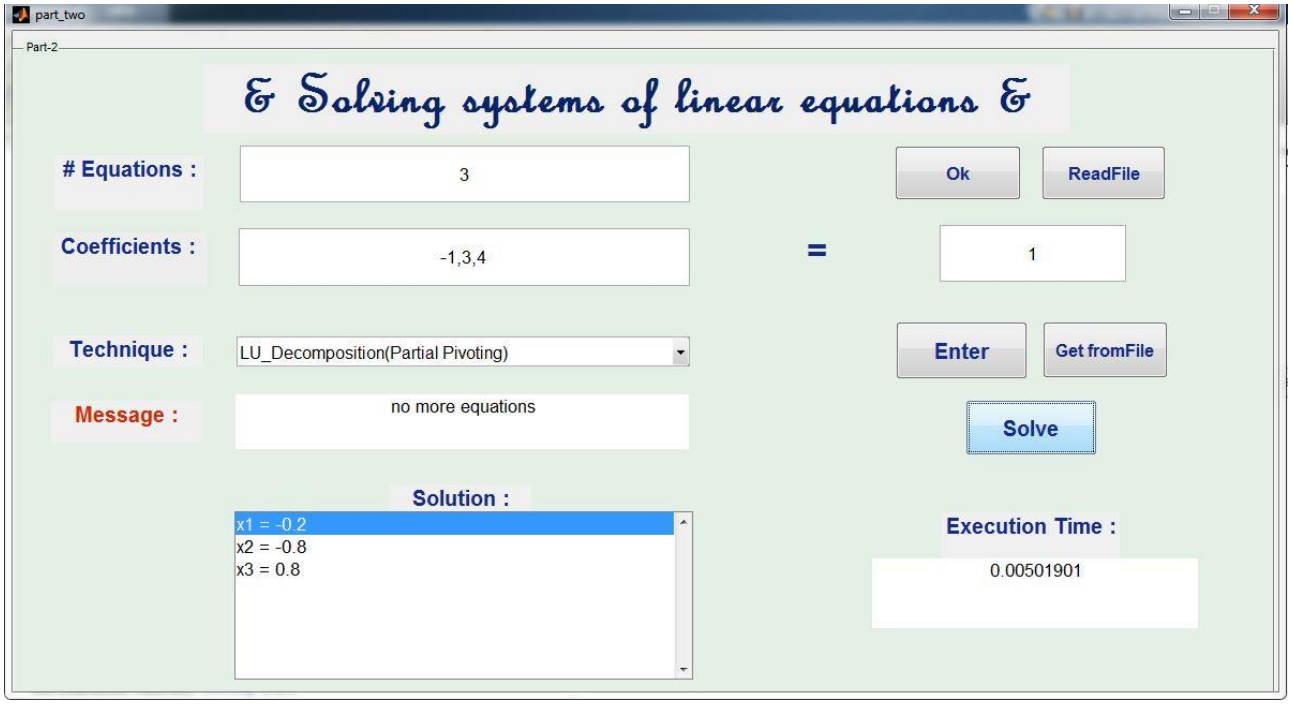
- In case of system of five linear equations as an input:



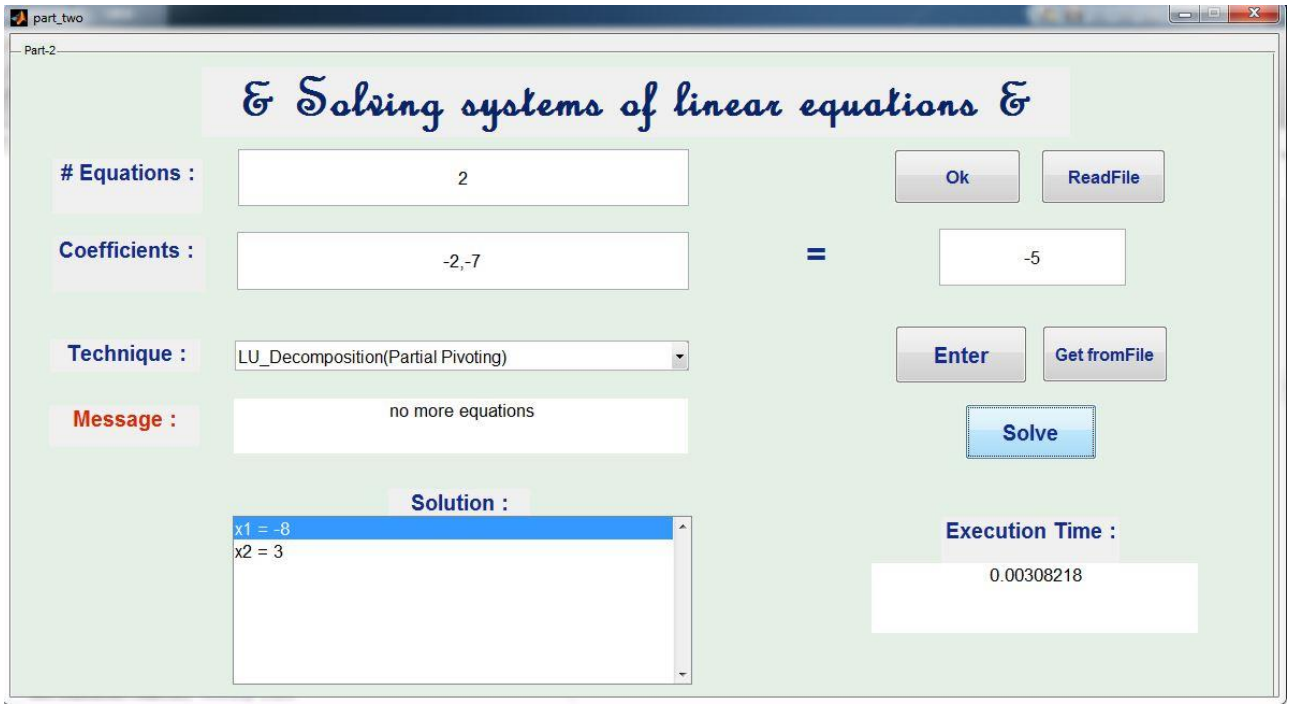
- In case of system of four linear equations as an input:



- In case of system of three linear equations as an input:

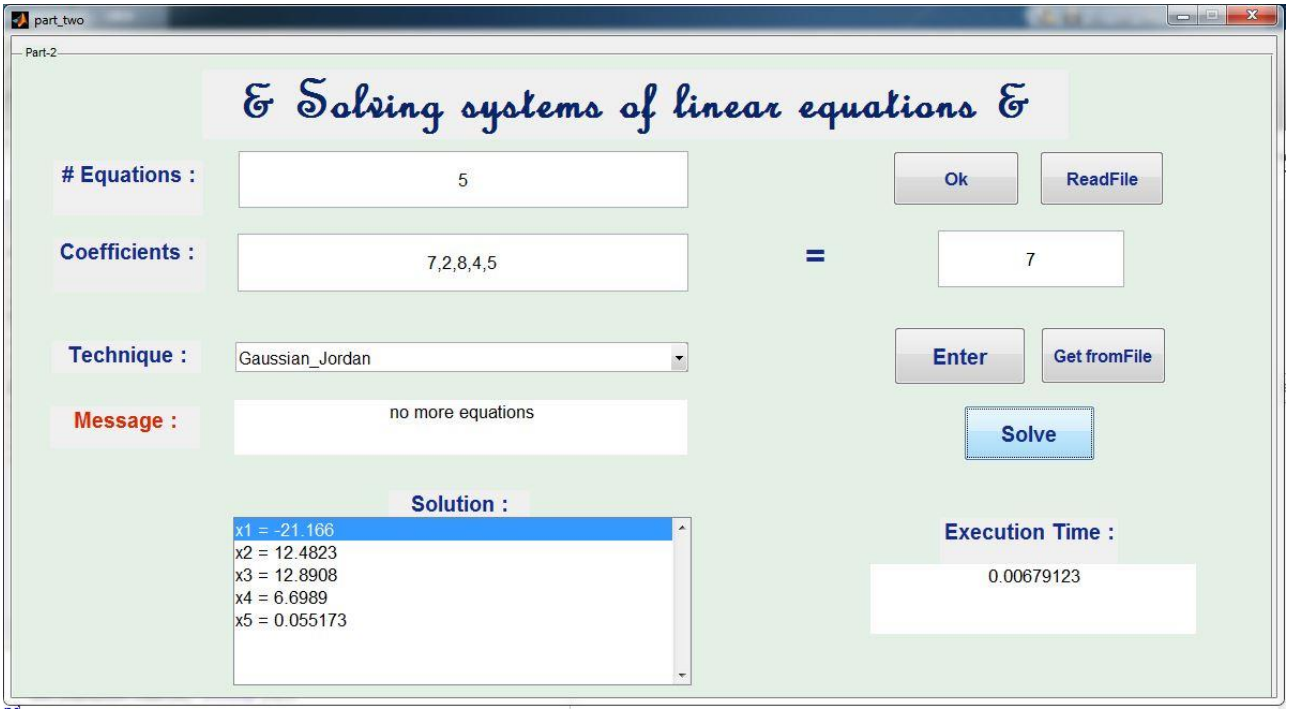


- In case of system of two linear equations as an input:

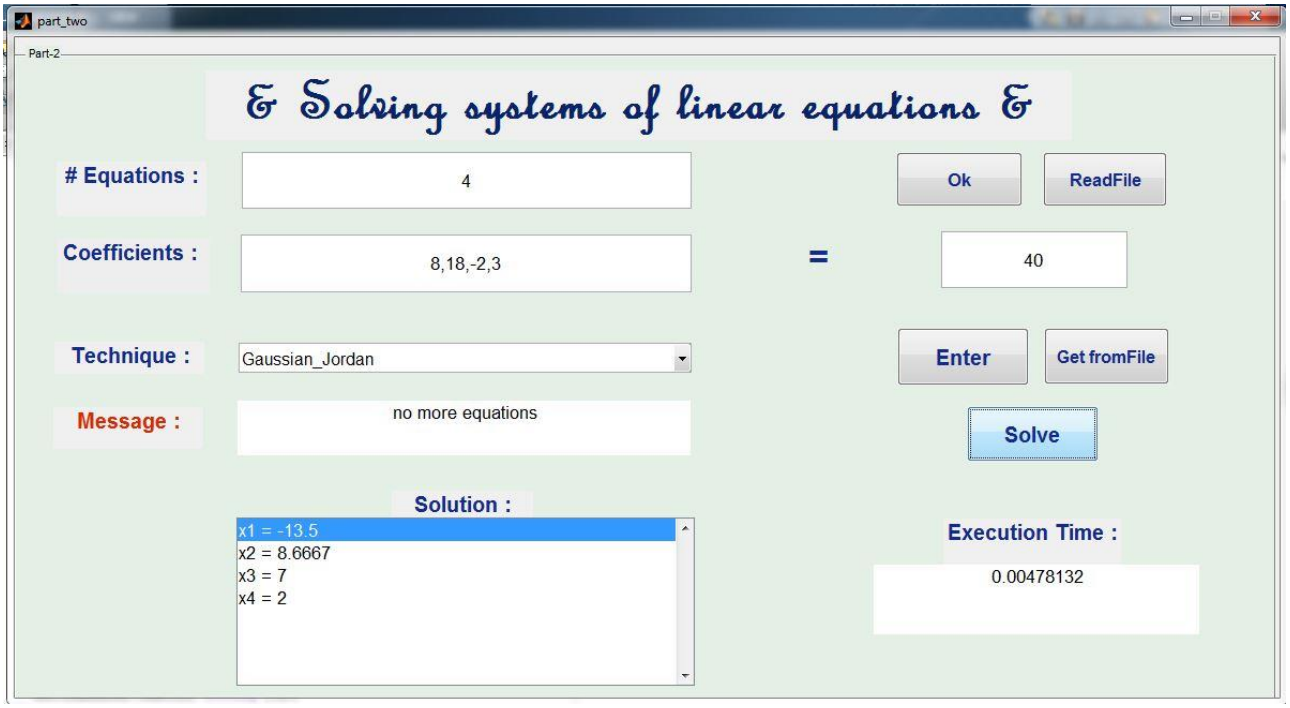


In Gaussian Jordan Method:

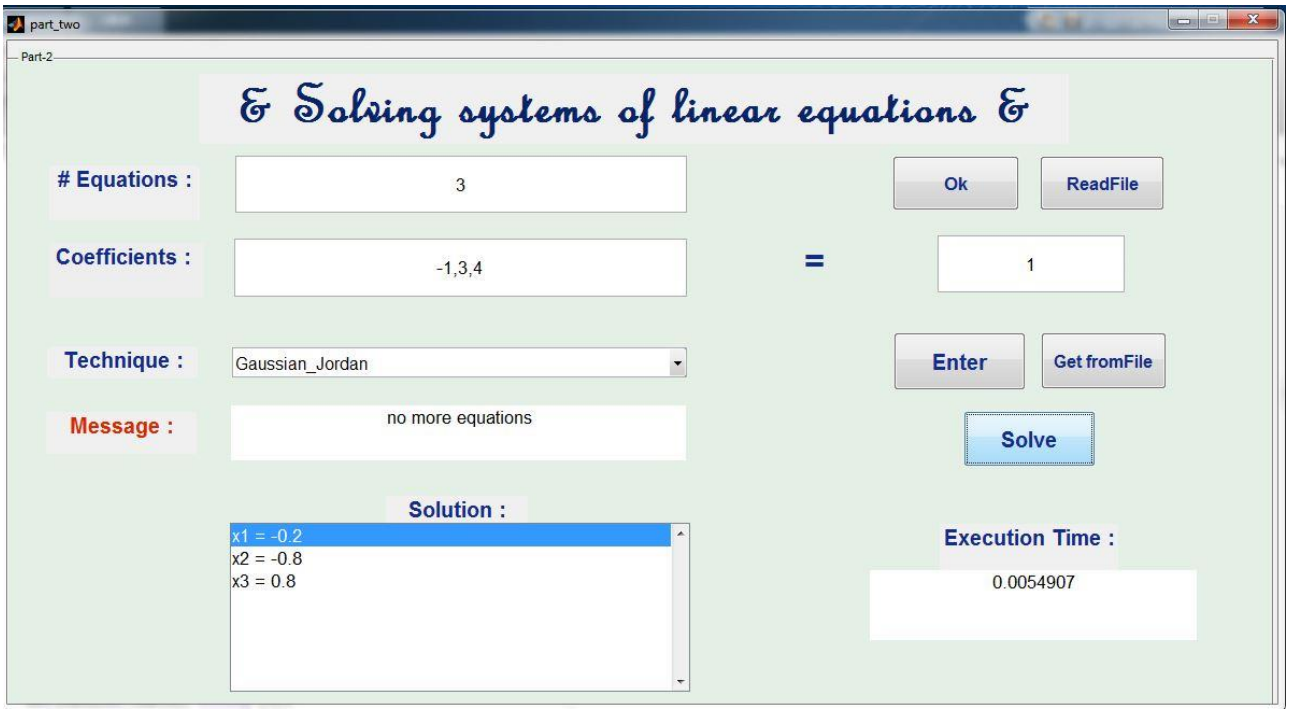
- In case of system of five linear equations as an input:



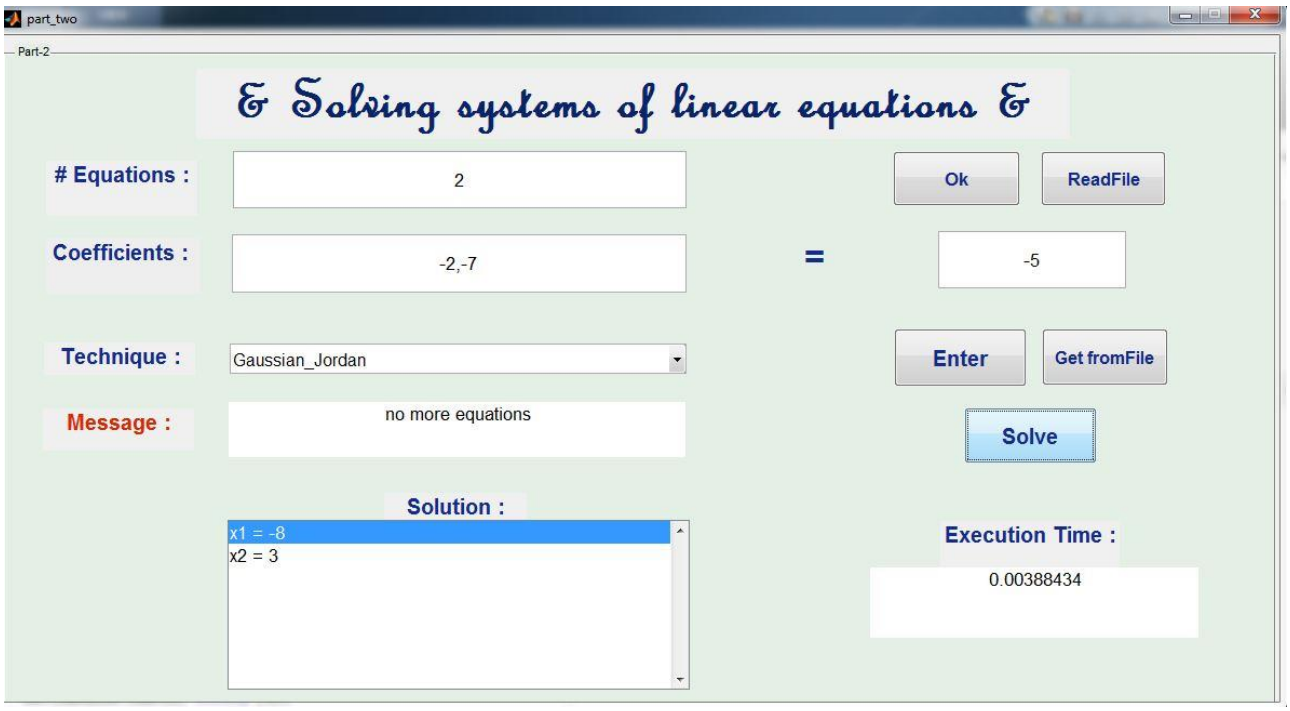
- In case of system of four linear equations as an input:



- In case of system of three linear equations as an input:

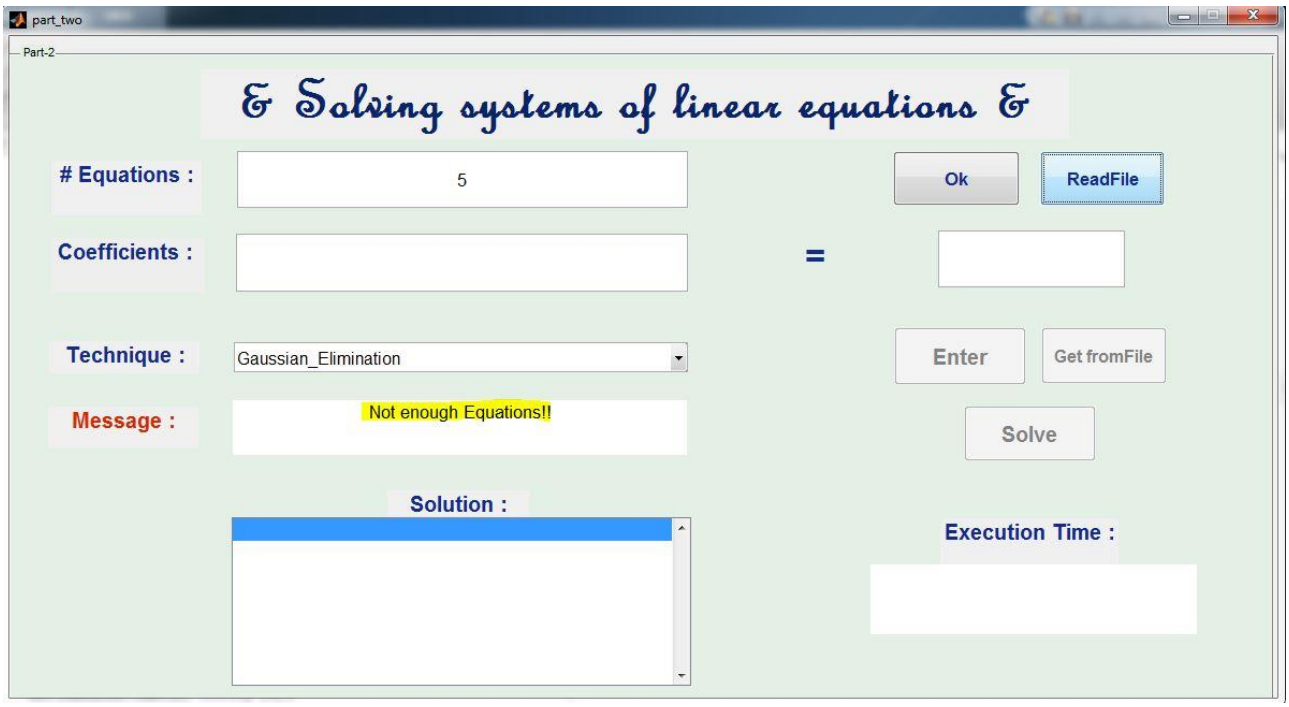


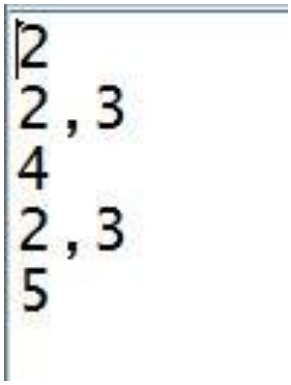
- In case of system of two linear equations as an input:

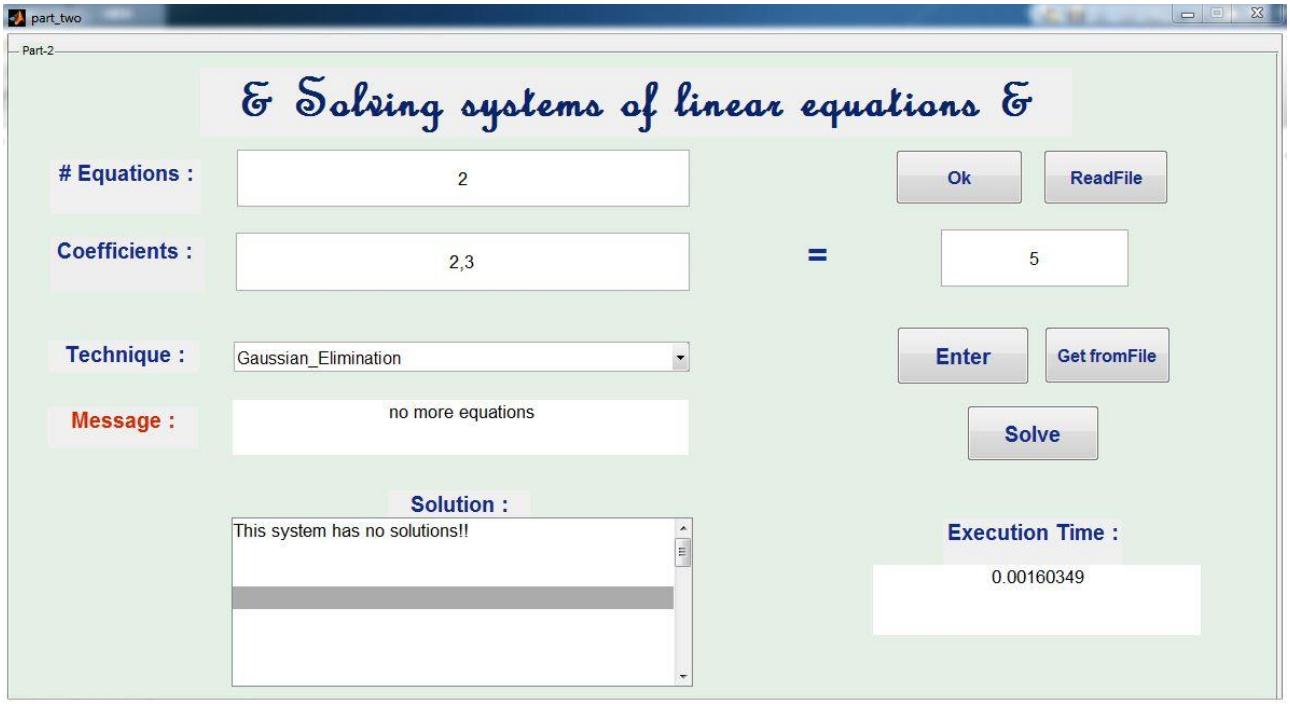


Special Cases:

```
5
4,6,2,-2
8
2,0,5,-2
4
-4,-3,-5,4
1
8,18,-2,3
40
```




2
2,3
4
2,3
5


part_two
Part-2
& Solving systems of linear equations &
# Equations :
2
Ok
ReadFile
Coefficients :
2,3
=
5
Technique :
Gaussian_Elimination
Enter
Get fromFile
Message :
no more equations
Solve
Solution :
This system has no solutions!!
Execution Time :
0.00160349