Numerical Assignment report

Mohamed Ismail Mohamed 53

Kharim Mohamed Mostafa 46

Kamal abdel Aziz Kamal 47

Ahmed Khaled Abdel Sayed 5

Part 1

Problem Statement:

You are required to implement a root finder program which takes as an input the equation (as free text), the technique to use (As drop-down menu) and its required parameters (e.g. interval for the bisection method, initial point for Newton-Raphson and so on). Also, you should implement a general algorithm that takes as an input the equation to solve and outputs all of its roots.

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. (input files format will be given later)
- Differentiation and Parsing is your task.
- A way to choose a method to solve the given equation. (Drop-down menu)
- A plot of the function with
    - boundary functions in case of bisection and false position
    - g(x) with y = x in case of fixed point
    - f'(x) in the remaining cases.
- A way to enter the precision and the max number of iterations otherwise default values are used, the program should stop once any condition is met. (Default Max Iterations = 50, Default Epsilon = 0.0001)
- The answer for the chosen method indicating the number of iterations, execution time, all iterations, approximate root, and precision achieved.
- The program must validate that a root exists in the given interval using the various checks that were studied in class whenever possible before running. In case any check fail, a message should be displayed to the user.
- The program must be able to run in 2 different modes:-
    - Single step mode: where each iteration is done by pressing on "NEXT" button.
    - Fast mode: where the program continues running until conditions are met.
- During both modes, the final outcome should include the full details of all the iterations

Pseudo code:

1- Bisection

```
function [result] = myBisectionAlgorithm (func,lb,ub,maxIterations,precision)
syms f(x);
f(x)=func;
lower_bound=lb;
upper_bound=ub;
xrnew=(lower_bound + upper_bound)/2;
xrold=0;
i=1;
error =abs((xrnew-xrold));
while  error> precision && i<maxIterations

    if (f(upper_bound)*f(xrnew)<0)
        lower_bound=xrnew;

    else
        if (f(lower_bound)*f(xrnew)<0)
        upper_bound=xrnew;
        else
            break;
        end
    end
    xrold = xrnew;
    xrnew = (lower_bound+upper_bound) /2;
    error =abs((xrnew-xrold));

    i=i+1;
end
result = xrnew;
end
```

2- False-position

```
function [result]= myFalsePosition (func,lb,ub,maxIterations,precision)
    syms f(x);
    f(x)=func;
    xl=lb;
    xu=ub;
    xrnew= double(((xl*f(xu))- (xu*f(xl)))/(f(xu) - f(xl)));
    xrold=0;
    i=1;
    error = abs(xrnew-xrold );
    while  error > precision && i< maxIterations
        if (f(xu)*f(xrnew)<0)
            xl=xrnew;
        else
            if (f(xl)*f(xrnew)<0)
            xu=xrnew;
            else
                break;
            end
        end
        xrold = xrnew;
        xrnew= double(((xl*f(xu))- (xu*f(xl)))/(f(xu) - f(xl)));
        i=i+1;
        error = abs(xrnew-xrold );

    end
    result=xrnew;
end
```

3- Fixed point

```
function [result] = FixedPoint(x_i1,func,maxIteration,precision)
    syms f(x)
    f(x) = func;
    syms g(x)
    g(x) = f+x;
    x_i2 = double(g(x_i1));
    iterations = 1;
    error =abs(x_i2-x_i1);


    while error > precision && iterations < maxIteration
        iterations = iterations+1;
        x_i1 = x_i2;
        x_i2 = double(g(x_i1));
        error =abs(x_i2-x_i1);


    end

    result=x_i2;

end
```

4- Newton-Raphson

```
function [result] = Newton_Raphson(x_i1,func,maxIteration,precision)
    syms f(x)
    f(x) =func;
    fx = double(f(x_i1));
    dif = diff(f,x);
    dfdx = double(dif(x_i1));
    x_i2 = x_i1-double(fx/dfdx);
    iterations = 1;
    error =abs(x_i2-x_i1);

    while error > precision && iterations < maxIteration
        if dfdx<.0001
            result=NaN;
            return;
        end
        iterations = iterations+1;
        x_i1 = x_i2;
        f(x_i1)
        fx = double(f(x_i1));
        dfdx = double(dif(x_i1));

        x_i2 = x_i1-double(fx/dfdx);
        error =abs(x_i2-x_i1);


    end
    result = x_i2;
    if iterations>=maxIteration
        result= NaN;
    end
end
```

5- Secant

```
function [result] = Secant (x_i0,x_i1,func,maxIteration,precision)
    syms f(x)
    f(x) = func;
    fx_0 = double(f(x_i0));
    fx_1 = double(f(x_i1));
    if fx_0-fx_1<.0001
        error('Division By Zero');
    end
    x_i2 = x_i1-double(fx_1*(x_i0-x_i1)/(fx_0-fx_1));
    iterations = 1;
    error =abs(x_i2-x_i1);


    while error > precision && iterations < maxIteration
        iterations = iterations+1;
        x_i0 = x_i1;
        x_i1 = x_i2;
        fx_0 = double(f(x_i0));
        fx_1 = double(f(x_i1));
        if fx_0-fx_1<.0001
            error('Division By Zero');
        end
        x_i2 = x_i1-double(fx_1*(x_i0-x_i1)/(fx_0-fx_1));
        error =abs(x_i2-x_i1);

    end

end
```

6- Bierge Vieta

```
function [str_arr] = birgeVietaCaller(a,x0,maxIteration,precision)
    [rooti1,str1] = myBirgeVieta(a,x0);
    str_arr={str1};
    [rooti2,str] = myBirgeVieta(a,rooti1);
    str_arr = cat(1,str_arr,str);
    i = 1;
    while (rooti2-rooti1)>precision && i<maxIteration
        rooti1=rooti2;
        [rooti2,str] = myBirgeVieta(a,rooti1);
        str_arr = cat(1,str_arr,str);
        i=i+1;
    end
end
```

```
function [root,str] = myBirgeVieta(a,Xo)
    x=numel(a);
    [n,m]=size(a);
    b= zeros(n,m);
    c= zeros(n,m);

    b(m)=a(m);
    c(m)=a(m);


    for i=x-1:-1:1
        b(i) = Xo*b(i+1)+a(i);
    end
    for i=x-1:-1:1
        c(i) = (Xo*c(1,i+1)+b(i));
    end
    root = Xo-(b(1)/c(2));

end
```

Some functions might diverge or their roots might not be found using some specific methods and converge or their roots can be found by other functions:

1. **Bisection**
   No function will diverge using bisection. However, some roots might be missed if they are multiple roots or if there is 2 roots close to each other. For example $f(x) = x^2-1$ will have no roots found using bisection methods

2. **False position**
   Its drawbacks is the same as bisection. In addition, some functions might converge too slowly if the starting points are not so good. For example:
   $f(x) = x^4+ 3x -4$ will take too much time to converge if the starting point is a big –ve number. To solve this problem, we can use some bisection first to decrease the range of choosing the starting point.

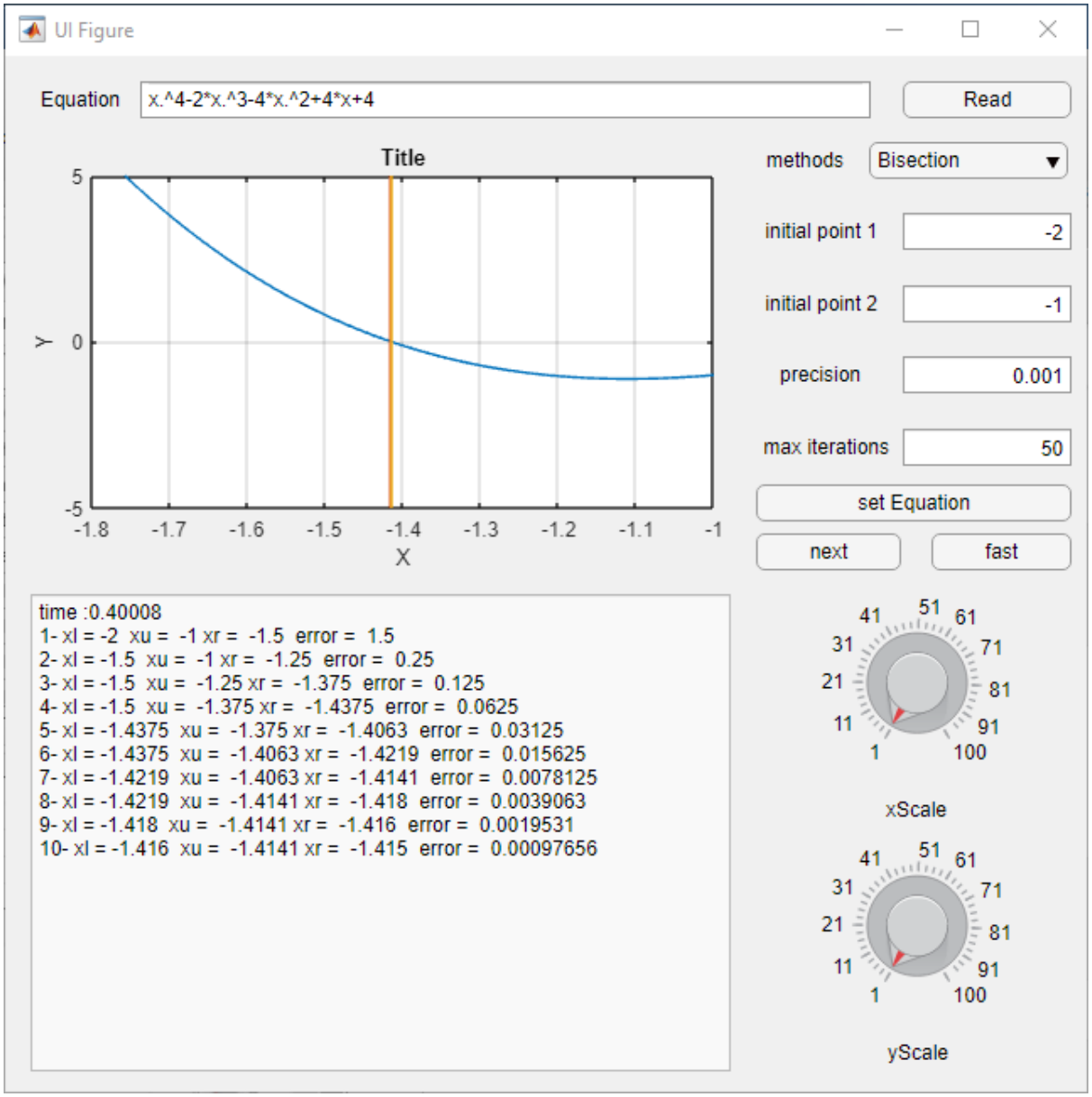
3. **Fixed point**
   Some functions might diverge using fixed point if the g(x) is not chosen correctly. The best method to find out is to calculate g'(x). if it is greater than 1, it will diverge so it is better to use another g(x).

4. **Newton-Raphson**
   Some functions might diverge using this method and some other functions might converge after too much time. For example: 2^x-2 will take forever to get the root 1 if we start from a very big –ve number. The best way to solve it is to use some bisection to decrease the range of guessing a starting point for newton.

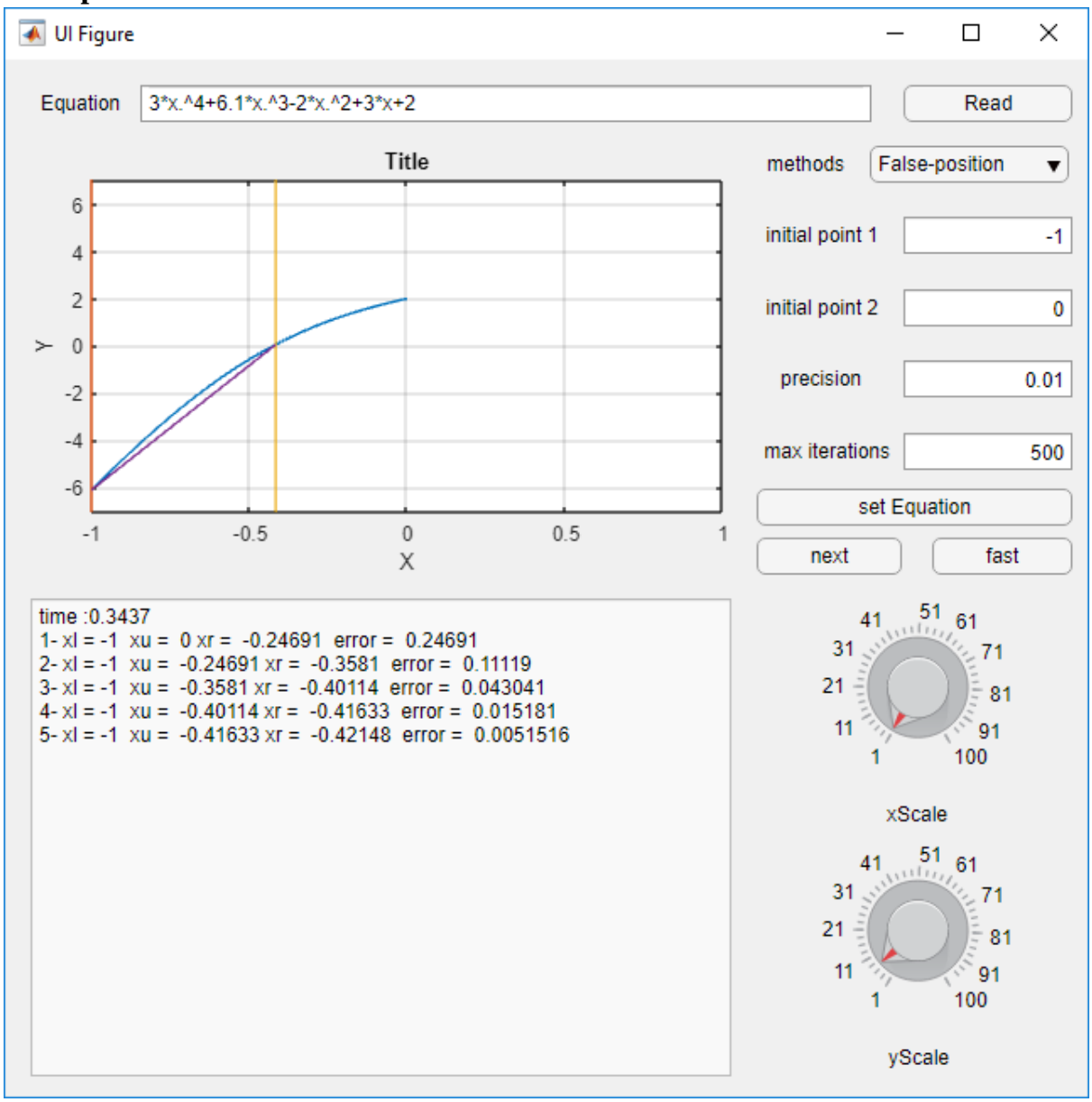
5. **Secant**
   This method is almost like false- position method. It will take too much time for the same functions. And it might diverge if the initial two points are not chosen properly. For example $f(x)= x^2 -1$ will diverge if we choose (2,-2) is the starting points.

Sample Runs:

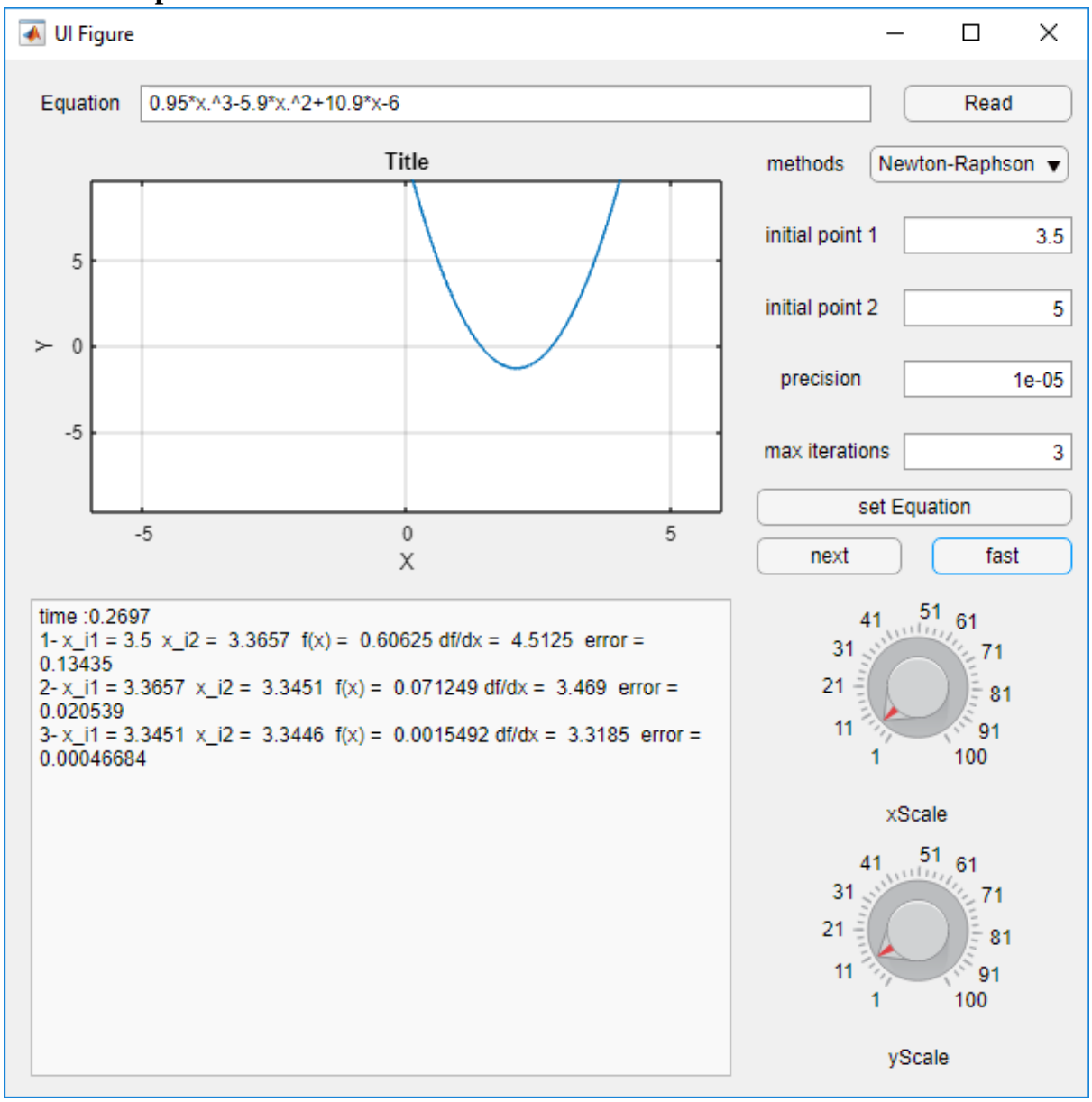
1. Bisection



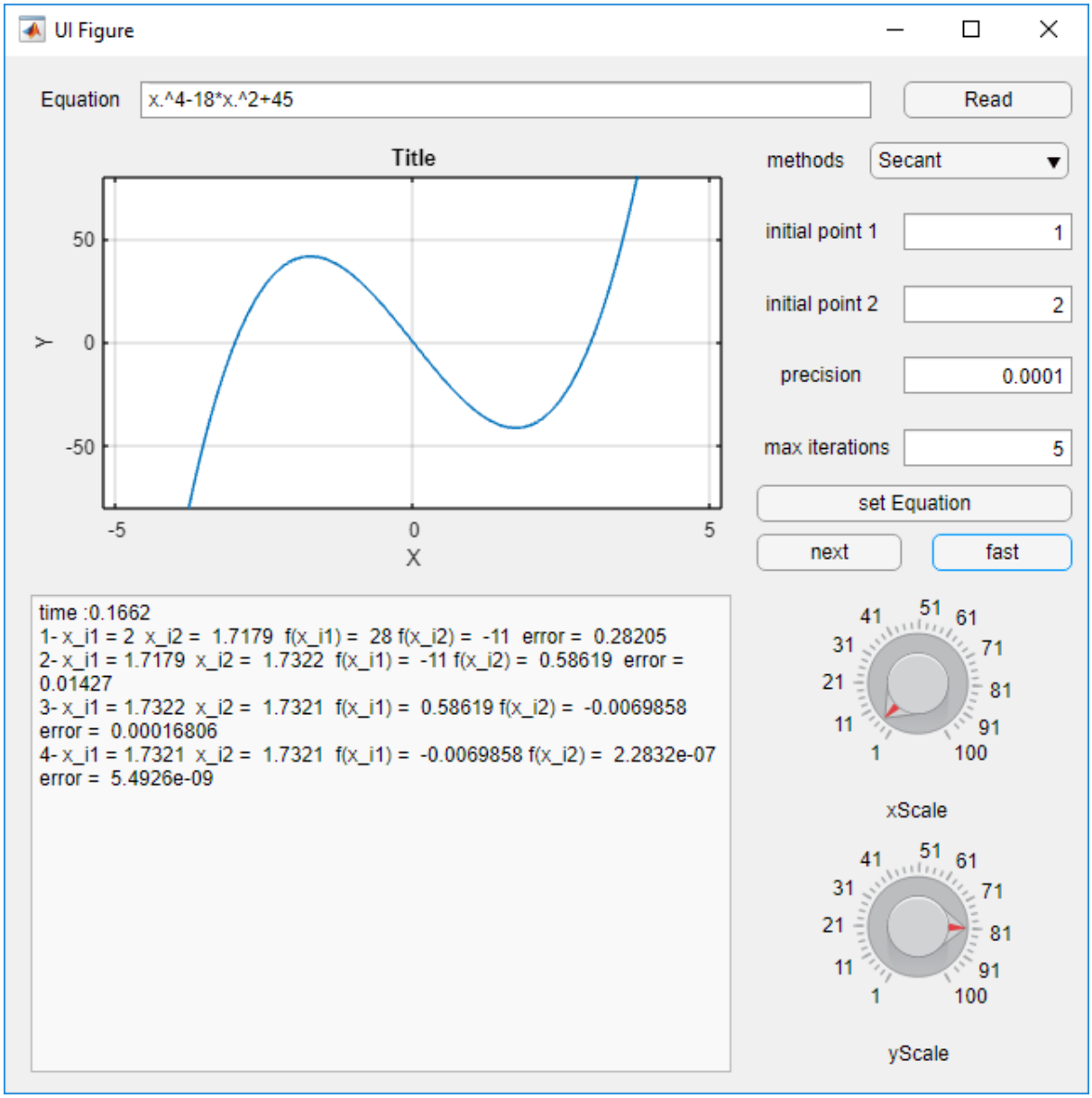
2. False position:



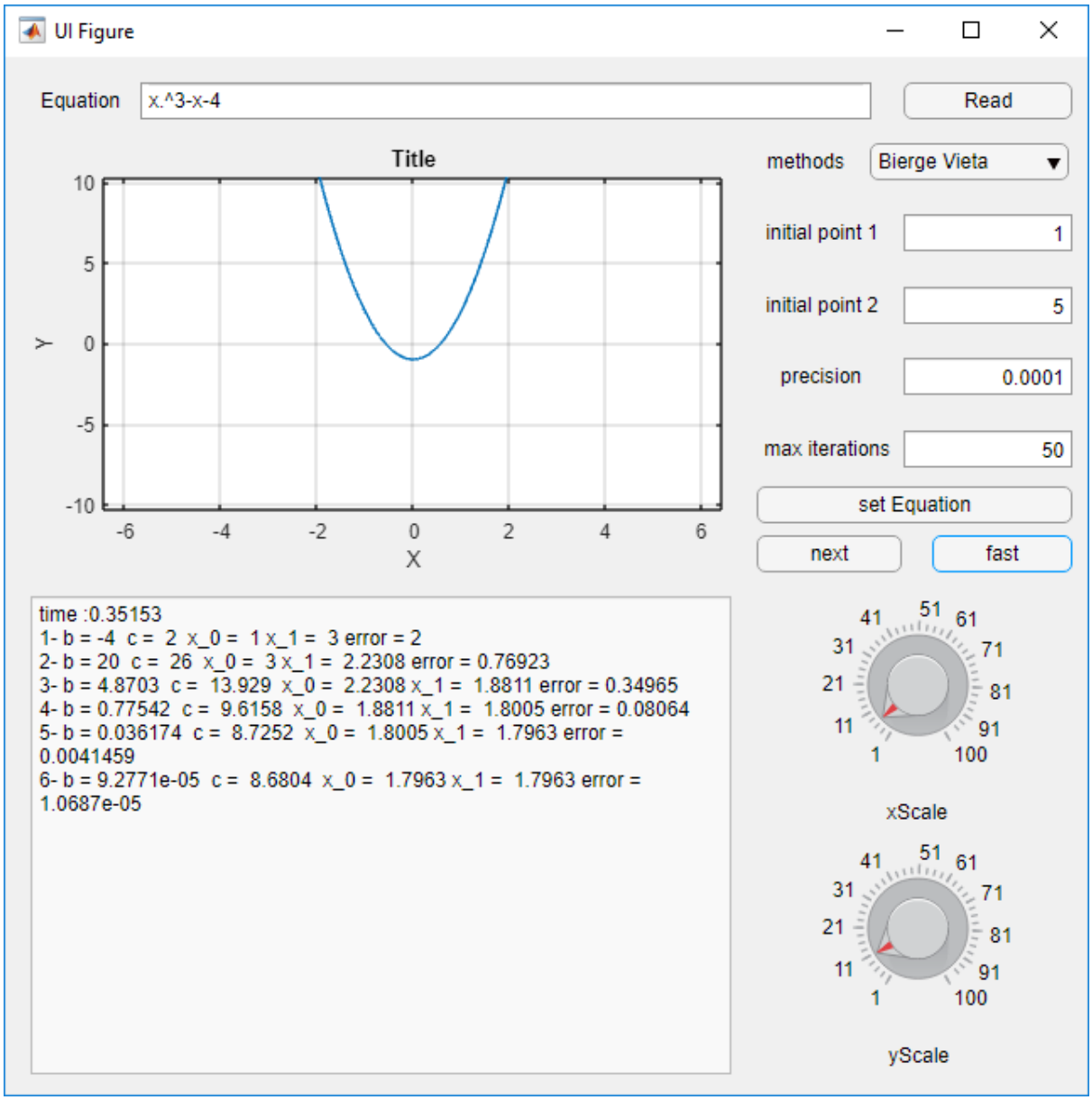
3. Newton-Raphson



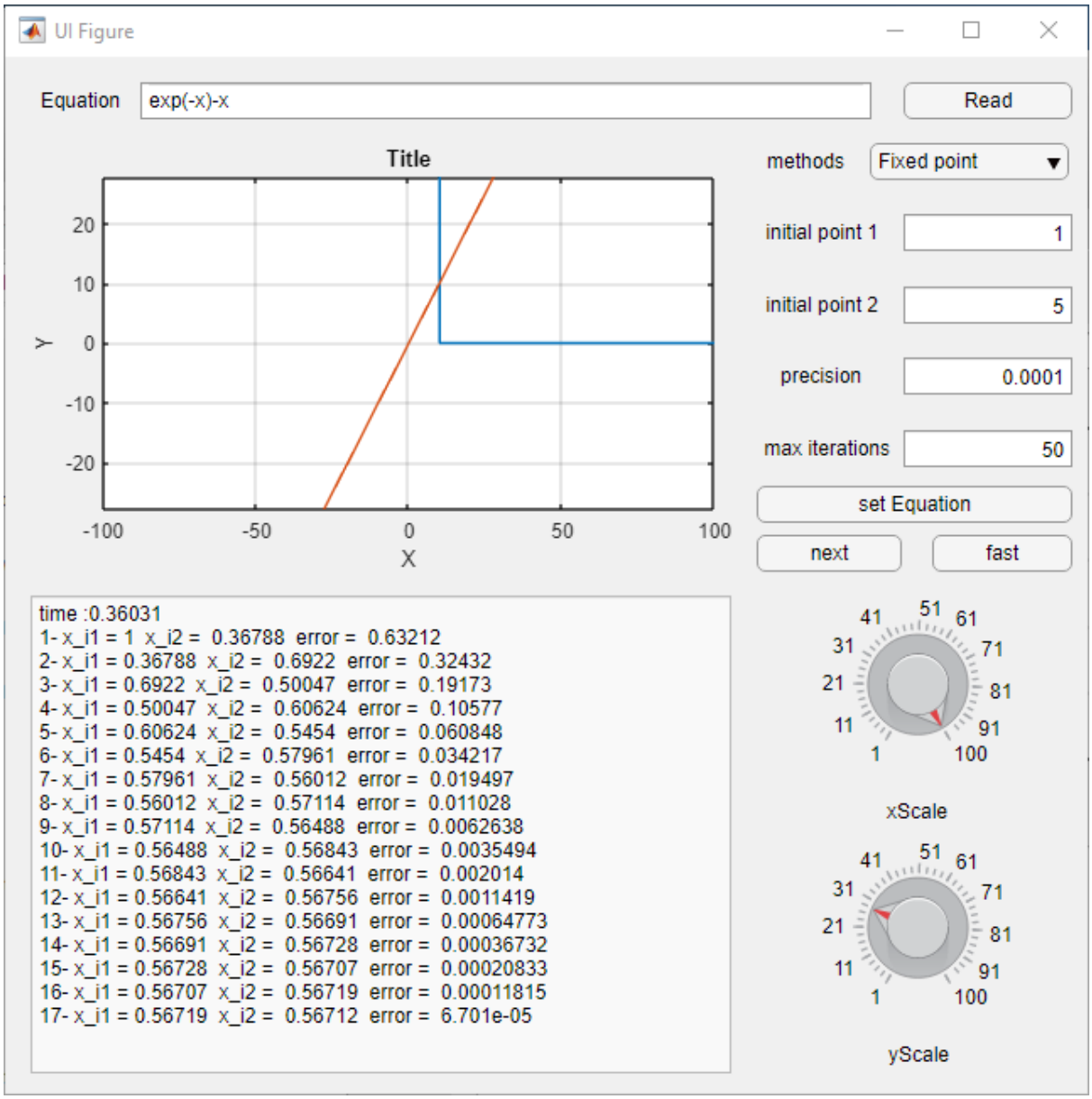
4. Secant


UI Figure
Equation
x.^4-18*x.^2+45
Read
Title
50
0
-50
Y
-5
0
5
X
methods
Secant
initial point 1
1
initial point 2
2
precision
0.0001
max iterations
5
set Equation
next
fast
time :0.1662
1- x_i1 = 2 x_i2 = 1.7179 f(x_i1) = 28 f(x_i2) = -11 error = 0.28205
2- x_i1 = 1.7179 x_i2 = 1.7322 f(x_i1) = -11 f(x_i2) = 0.58619 error = 0.01427
3- x_i1 = 1.7322 x_i2 = 1.7321 f(x_i1) = 0.58619 f(x_i2) = -0.0069858 error = 0.00016806
4- x_i1 = 1.7321 x_i2 = 1.7321 f(x_i1) = -0.0069858 f(x_i2) = 2.2832e-07 error = 5.4926e-09
1
11
21
31
41
51
61
71
81
91
100
xScale
yScale

5. Bridge Vieta



6. Fixed Point



The General Algorithm:

In the general algorithm, we combined the Newton Raphson Algorithm and Bracketing method so that we get nearly all roots. First Newton-Raphson method is run among a specific range with some well-chosen numbers so it gets nearly all the roots. Then, the bracketing method is used among all the range with a big integral to search in. if any root is missed from the Newton-Raphson method, the bracketing method will fund it unless it is a multiple root so it will not be found. The probability of missing a root is so small by this method.

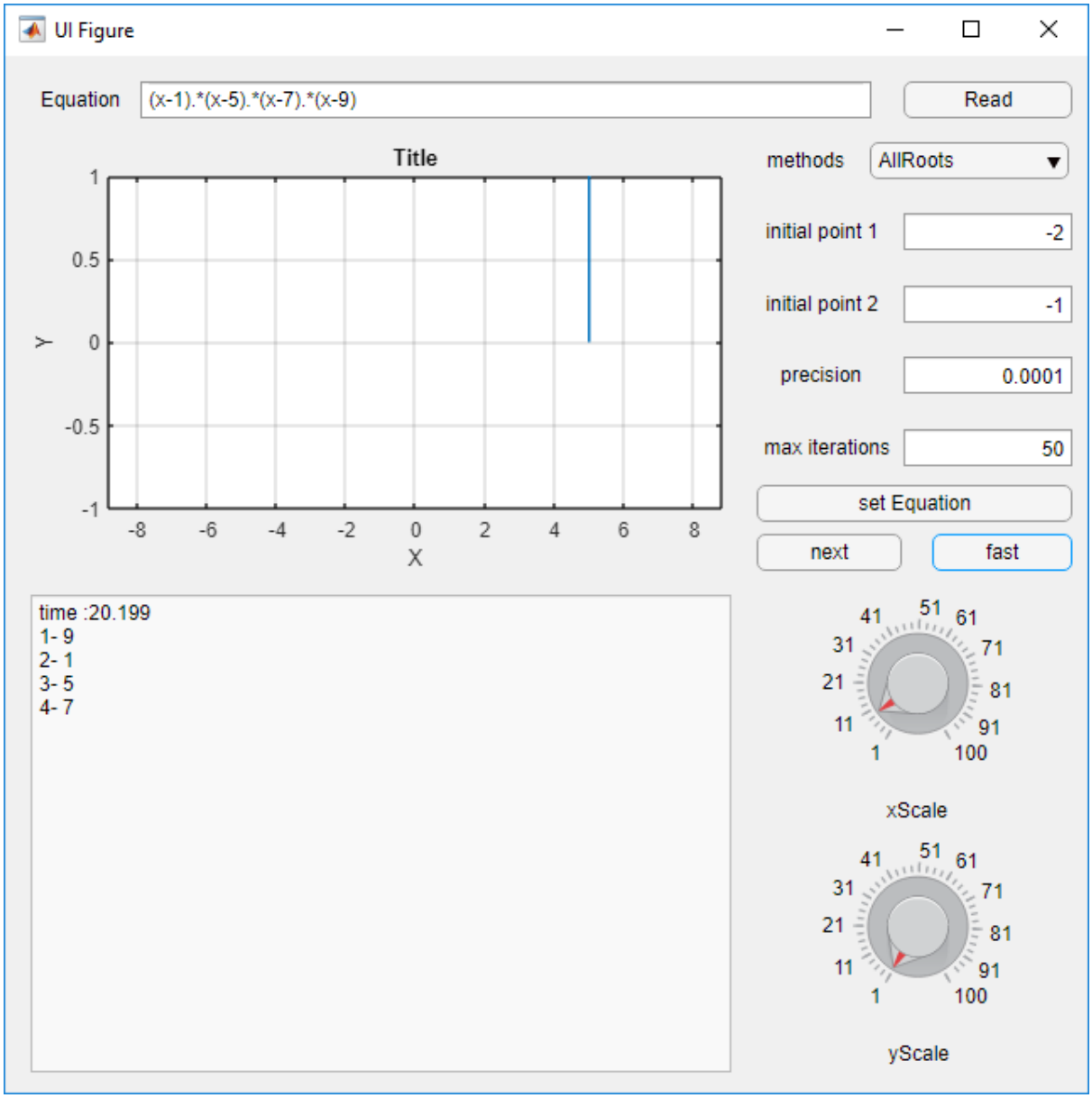
Some function roots might not be found in some conditions:

If the root is a multiple root between two other roots and the distance between them is so small. So the newton might miss it and the bracketing will never find it as it is a multiple root. The solution for this problem is to use newton is linear small distances instead of the algorithm used. Or to use another algorithm that can get the multiple roots along with newton instead of bracketing.

Pseudo code:

```
function result = getAllRoots( f,lb,ub, maxIteration,perscision)

    i=lb,u=ub, n=1;
    while i<=u &&n<50
        if(f(i)==0)
            result{n}=i;
            n++, i+;
           continue;
        end
        temp=Newton_Raphson(i,f,maxIteration,perscision);
        if(isnan(temp))
            q=1;
            while isnan(temp)&& i<u
                i+=50*q;q++;
                temp=Newton_Raphson(i,f,maxIteration,perscision);
            end
        end

        if(!result.contains(temp))
            result{n}=temp;
            n++;
            i=temp+0.1;
       else
           i=i+(i-temp);
        end
    end

    l=Newton_Raphson(ub,f,maxIteration,perscision);
    u=Newton_Raphson(lb,f,maxIteration,perscision);
    if(isnan(u))
        u=ub;
    end
    if(isnan(l))
        l=lb;
    end
    use biesction with the interval(l,u);
    end

end
```

Sample run:


UI Figure
Equation
(x-1).*(x-5).*(x-7).*(x-9)
Read
Title
Y
X
methods
AllRoots
initial point 1
-2
initial point 2
-1
precision
0.0001
max iterations
50
set Equation
next
fast
time :20.199
1- 9
2- 1
3- 5
4- 7
xScale
yScale

Used built in MATLAB function:

num2str : converts numbers to strings

str2func : converts strings to functions

cat : adding elements to an array

matlabFunction : string to function

numel(array) : array size

diff : gets the derivative of a function

zeros : creates an array and sets all initial values to zero

size : array size

fopen : opens file

str2double : converts strings to doubles

textscan : reads from file line by line

regexprep ,strsplit : parsing files

num2cell : converts array of numbers to array of cells

cell2mat : converts array of cells to array of numbers

isnan : checks whether a values is null or not

Part 2

Problem statement:

You are required to implement a program for querying the values of specific points using interpolation which takes as an input the polynomial order, sample point(s), corresponding value(s), the interpolation technique to use (Newton – Lagrange) and the query point(s)

The program must contain the following features:

- An interactive GUI that enables the user to enter an order, a set of data point(s) and corresponding value(s).
- Reading from files must be available as well. (input files format will be given later)
- A way to choose a method for interpolation. (Drop-down Menu)
- A way to enter the query point(s) where we need to find a value.
- The answer for the chosen method indicating the execution time and solution.
- The polynomial function obtained from the interpolation and its plot in the data set range.

Pseudo code:

1. Lagrange:

```
function [ corresponding ] = Lagrange(order,values,correspondings,query )
    syms f(x)
    syms g(x)
    f(x) = 0;
    iterator_1 = 1;
    iterations = 0;
    values = cell2mat(values);
    correspondings = cell2mat(correspondings);
    while iterations<(order+1)
        g(x) = double(correspondings(iterator_1))
        iterator_2 = 1 ;
        iterations_two = 1;
        while iterations_two<(order+1)
            if iterator_1 ~= iterator_2
                g(x) = g(x)
                *(x-values(iterator_2))
                /double((values(iterator_1)-values(iterator_2)))
                iterations_two = iterations_two +1
            end
            iterator_2 = iterator_2+1

        end
        f(x) = f(x)+g(x);
        iterator_1 = iterator_1+1
        iterations = iterations + 1
    end
    corresponding = double(f(query))
end
```

2- Newton

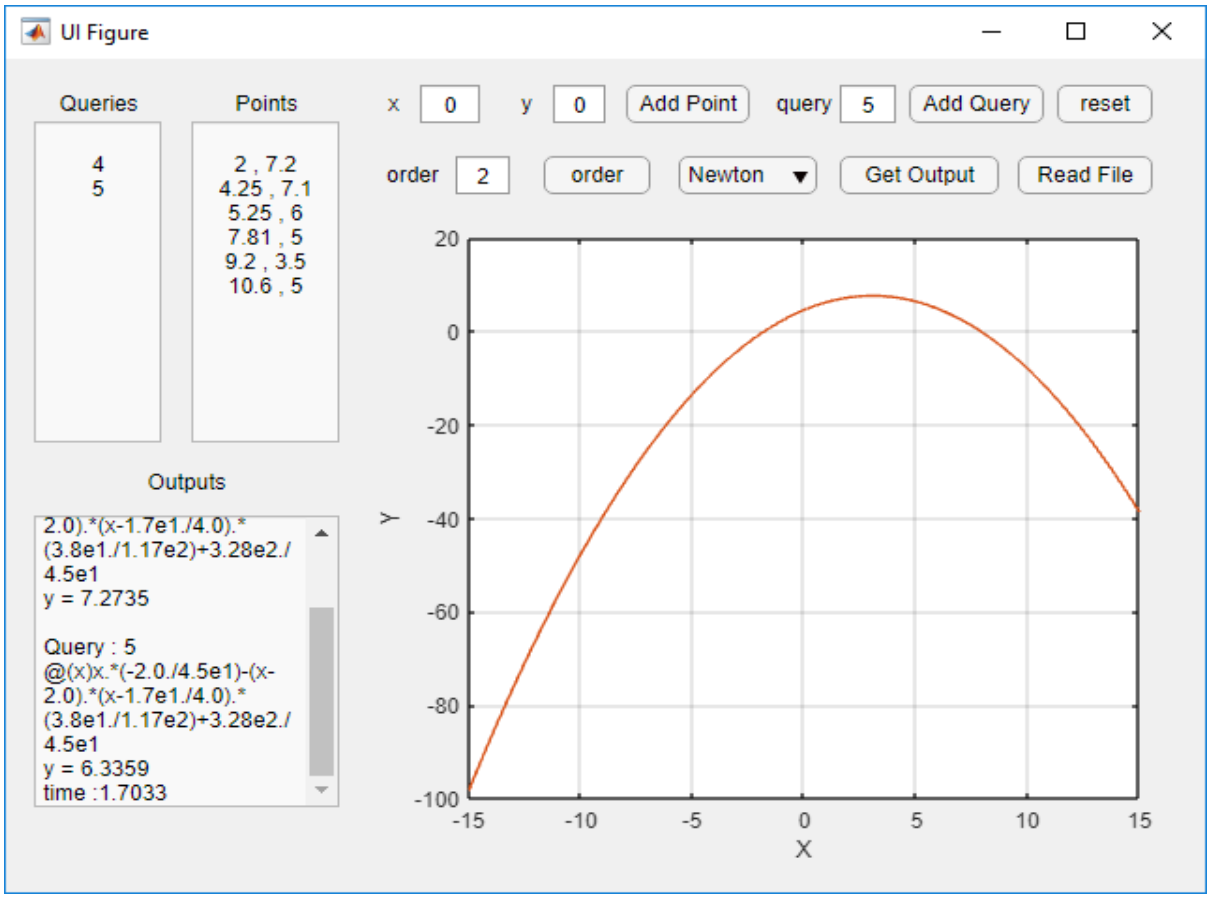
```
function [root,func] = newton_interpolation(order,points,values,point)
    x_array= cell2mat(points);
    y_array=cell2mat(values);
    num=numel(x_array);
    order = order +1;
    table = zeros (num,order);
    b= zeros (num,1);

    for i=1:1:num
        table(i,1)= x_array(i);
        table(i,2)=y_array(i);
    end

    for i=2:1:num
        for j=3:1:(order+1)
            if (i-(j-2)>0)
                table(i,j)= (table(i,j-1)-table(i-1,j-1))/(table(i,1)-table(i-(j-2),1));
            end
        end
    end
    i=1;
    for j=2:1:(order+1)
        b(i,1)=table(j-1,j);
        i=i+1;
    end
    syms f(x);
    syms g(x);

    f(x)=0;
    g(x)=1;

    for i=1:1:num
        j=1;
        g(x)=1;
        while j~=i
            g(x)=g(x)*(x-table(j,1));
            j=j+1;
        end
        f(x)=f(x)+b(i,1)*g(x);
    end
    func=matlabFunction(f);
    root=double(f(point));

end
```

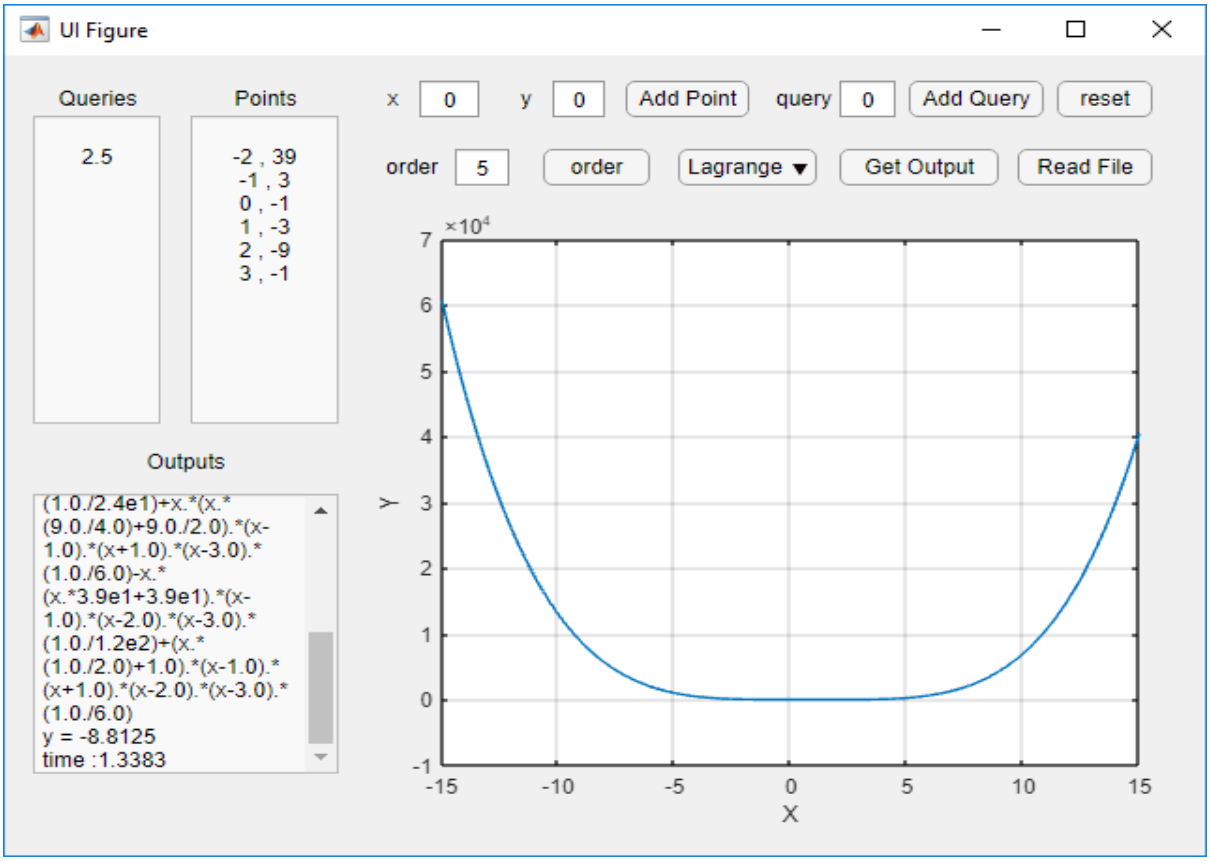
In this part the algorithms used for interpolation are not allowed to work (gives error) in two conditions:-

1- Giving an order larger than number of points+1
2- Having a query on a point larger than the largest point

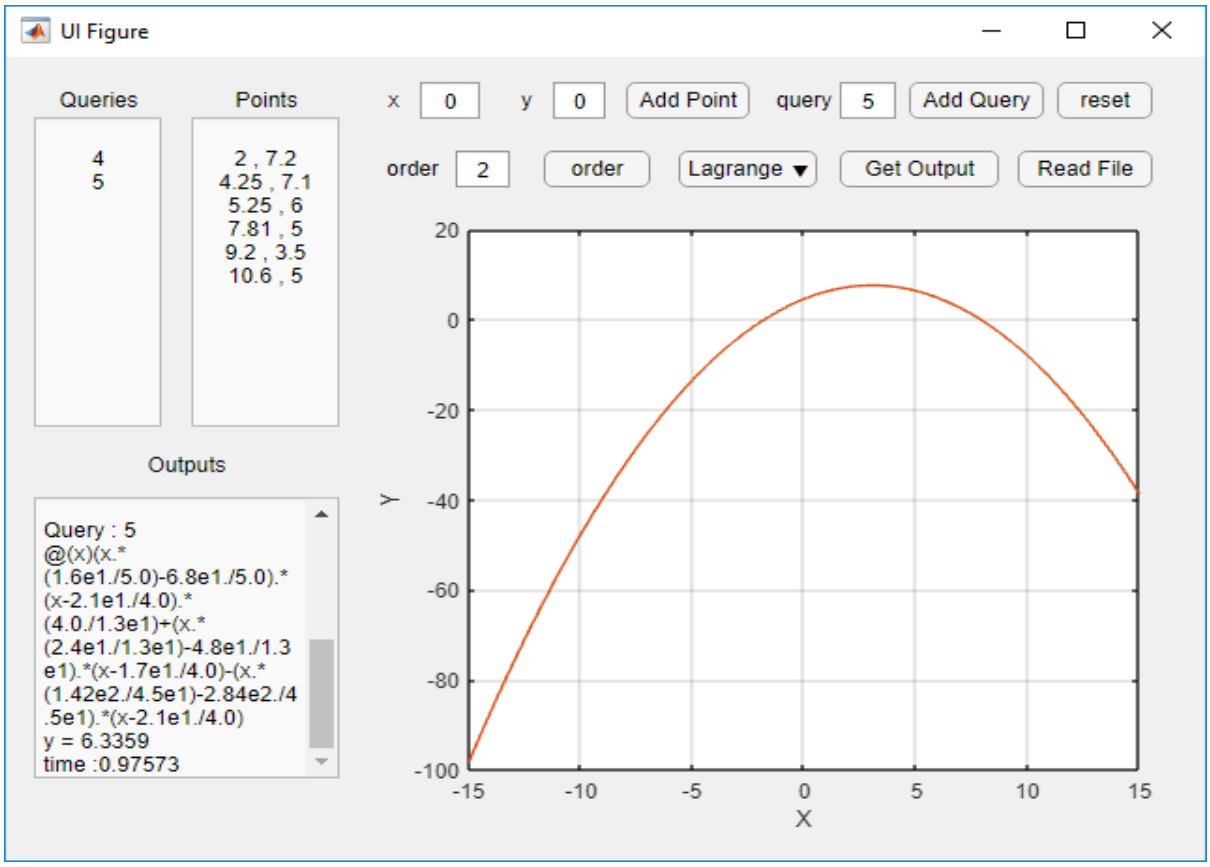
Sample Runs:

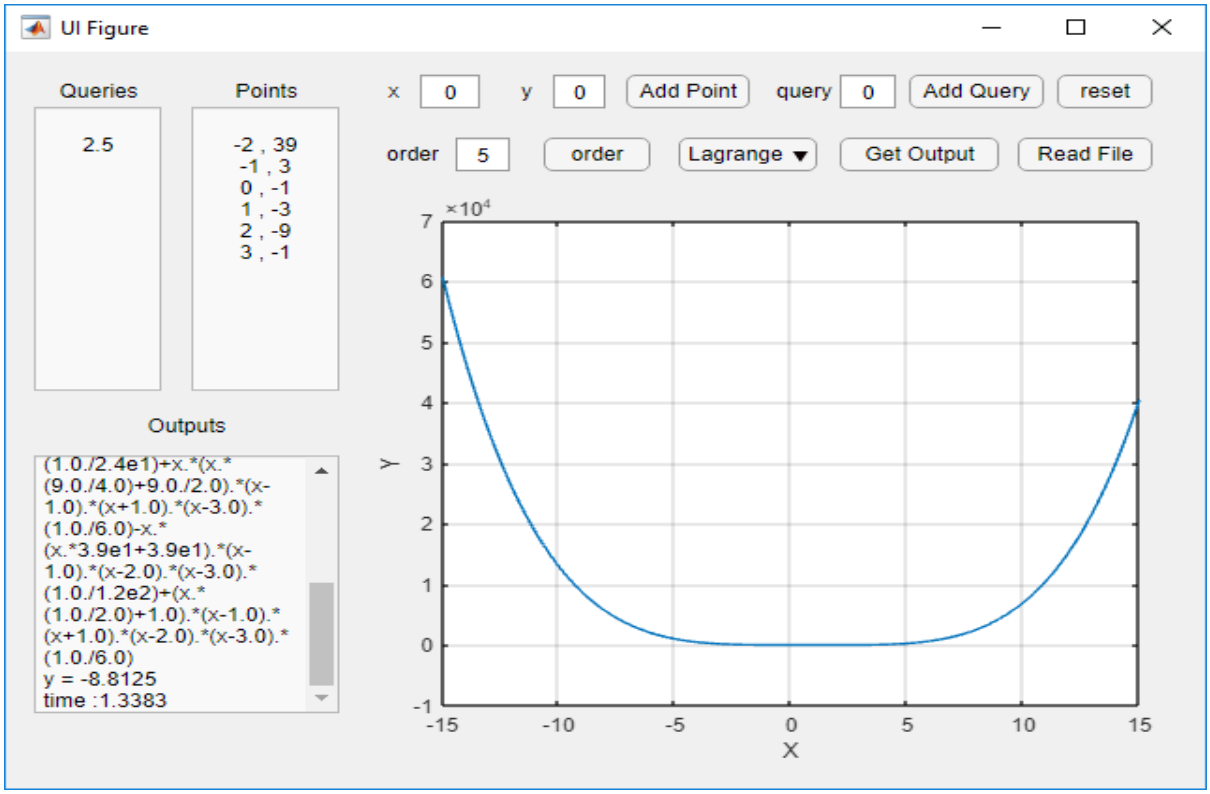
1) Newton





2) Lagrange


UI Figure
Queries
Points
4
5
2 , 7.2
4.25 , 7.1
5.25 , 6
7.81 , 5
9.2 , 3.5
10.6 , 5
x 0 y 0 Add Point query 5 Add Query reset
order 2 order Lagrange Get Output Read File
Outputs
Query : 5
@(x)(x.*
(1.6e1./5.0)-6.8e1./5.0).*
(x-2.1e1./4.0).*
(4.0./1.3e1)+(x.*
(2.4e1./1.3e1)-4.8e1./1.3
e1).*(x-1.7e1./4.0)-(x.*
(1.42e2./4.5e1)-2.84e2./4
.5e1).*(x-2.1e1./4.0)
y = 6.3359
time :0.97573
20
0
-20
-40
-60
-80
-100
Y
-15
-10
-5
0
5
10
15
X


UI Figure
Queries
Points
2.5
-2 , 39
-1 , 3
0 , -1
1 , -3
2 , -9
3 , -1
x 0 y 0 Add Point query 0 Add Query reset
order 5 order Lagrange Get Output Read File
Outputs
(1.0./2.4e1)+x.*(x.*
(9.0./4.0)+9.0./2.0).*(x-
1.0).*(x+1.0).*(x-3.0).*
(1.0./6.0)-x.*
(x.*3.9e1+3.9e1).*(x-
1.0).*(x-2.0).*(x-3.0).*
(1.0./1.2e2)+(x.*
(1.0./2.0)+1.0).*(x-1.0).*
(x+1.0).*(x-2.0).*(x-3.0).*
(1.0./6.0)
y = -8.8125
time :1.3383
×10^4
7
6
5
4
3
2
1
0
-1
Y
-15
-10
-5
0
5
10
15
X