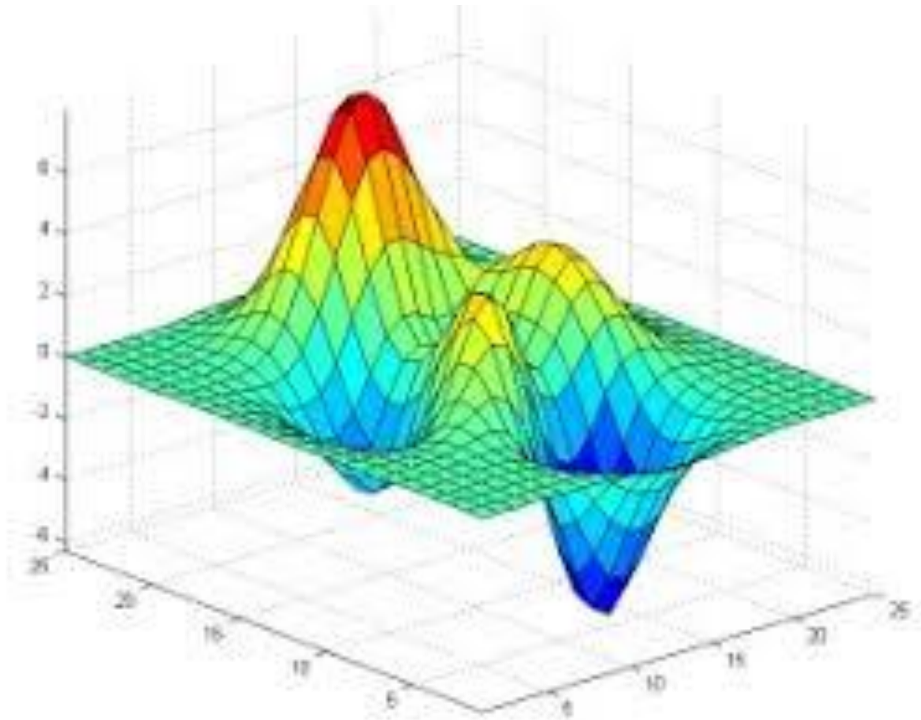


Numerical Methods

PART I

Ziyad Elbanna -27
Abdelrahman Ahmed -36
Abdelrahman Saeed -39
Khalil Esmail -23
Ahmed Elbawab -8
| Numerical Analysis | 11/5/2018

Algorithms and Data structures

PSEUDO-CODES

1. FIXED POINT ITERATION

```
function [root,arr,tt,err] = Fixed_Point(F,G,x1,it,tolerance)
    arr = [];  tt = 0; err = 0;
    for ind =1:1:it
        tt = tt + 1;  Xnew = G(x1);
        arr(end +1) = Xnew;
        %fprintf(' >>>> %12.5f',G(x1));
        if (F(Xnew)==0 )
          %fprintf('breaked from Xnew');
            break;
        end
        if ind>=2
            err = abs((Xnew-x1)/Xnew);
            if (err<tolerance)
               % fprintf('breaked from error');
                break;
            end
        end
        x1=Xnew;
    end
    root=Xnew;
end
```

2. BISECTION

```
function [ out_l, out_u, out_r, bisect_cell_data] = bisectAlgorithm( xl, xu, fx, tolerance, max_iter )

%UNTITLED3 Summary of this function goes here

%   Detailed explanation goes here


flag = 0;

xr_old = 0;

xr = 0;

ea = 0;

iter = 1;

bisect_cell_data = {};

bisect_cell_data(1, :) = {"xl", "xu", "xr", "Ea", "ea"};

display(sprintf("xl\t\t\t\txu\t\t\t\txr\t\t\t\tEa\t\t\t\tea") + "");

while 1

    xr = (xu + xl) / 2;

    if flag

        Ea = abs(xr - xr_old);

        if(xr ~= 0)

            ea = abs((Ea / xr) * 100);

            output = sprintf("%-15.f%-15.5f%-15.5f%-15.5f%-15.5f",  xl, xu, xr, Ea, ea);

            bisect_cell_data(iter, :) = {xl, xu, xr, Ea, ea};

            display("" + output);

        if(Ea < abs(tolerance) | iter > max_iter)

            out_l = xl;

            out_u = xu;
```

```
        out_r = xr;
        break;
      end
      end



   end
   iter  = iter + 1;
   if(fx(xr) * fx(xu) < 0)
      xl  = xr;
   else if(fx(xr) * fx(xl) < 0)
        xu = xr;
      else
        display("no answer");
        break;
      end
   end
   xr_old = xr;
   flag = 1;
end
%%uitable('ColumnName',cell_arr(1, :),'Data',cell2mat(cell_arr(2:end, :)))
```

3. FALSE POSITION

```
function [out_l, out_u, out_r, cdata] = FalsePosition( xl, xu, fx, tolerance, max_iter)
%FALSEPOSITION Summary of this function goes here
%   Detailed explanation goes here
```

```
flag = 0;
xr_old = 0;
xr = 0;
ea = 0;
cdata = {};
cdata(1, :) = {"xl", "xu", "xr", "Ea", "ea"};
iter = 1;

display(sprintf("xl\t\t\t\txu\t\t\t\txr\t\t\t\tEa\t\t\t\tea") + "");

while 1
    xr = xu -  (fx(xu) * (xu - xl))/(fx(xu)- fx(xl));
    if flag
        Ea = abs(xr - xr_old);
        if(xr ~= 0)
            ea = abs((Ea / xr) * 100);
            output = sprintf("%-15.5f%-15.5f%-15.5f%-15.5f%-15.5f",  xl, xu, xr, Ea, ea);
            cdata(iter, :) = {xl, xu, xr, Ea, ea};
            display("" + output);
        if(Ea < abs(tolerance) | iter > max_iter)
            out_l = xl;
            out_u = xu;
            out_r = xr;
            break;
        end
        end
```

```
    end
    if(fx(xr) * fx(xu) < 0)
        xl  = xr;
    else if(fx(xr) * fx(xl) < 0)
            xu = xr;
        else
            display("no answer");
            break;
        end
    end
    xr_old = xr;
    flag = 1;



end
```

4. NEWTON RAPHSON

```
function [ xr, cdata ] = newton( xi, fx, es, max_iter , m )


syms x;
func = diff(fx, x);
ea = 0;


i = 0;
cdata = {};
```

```
cdata(1, :) = {"xi", "Ea",  "ea"};

iter = 2;

while i< max_iter


    xj = xi - m * eval((subs(fx,x,xi))) / eval( (subs(func,x,xi)));

%     fprintf('%f   %f\n', xj, xi);


    if xj ~= 0 & ~isnan(xj) & ~isinf(xj)

        Ea = abs(xj - xi);

        ea = abs(Ea / xj) * 100;

        fprintf('iter: %d,  xi: %.6f,  Ea: %.6f,  ea: %.6f\n',i,xi, Ea, ea);

        cdata(iter, : ) = {xi, Ea, ea};

        if(ea < es)

            break;

        end

    else if isnan(xj)

            if(eval((subs(fx,x,xi))) == 0)

                xj = xi;

                cdata(iter, : ) = {xi, 0, 0};

                break;

            end

        else if isinf(xj)

                xj = xi;

                cdata(iter, : ) = {xi, Ea, ea};

                break;

            end
```

```
        end

    end
    iter = iter + 1;

    xi = xj;

    i = i + 1;
end
xr = xj;

end
```

5. SECANT

```
function [ xr, cdata ] = secant_method(  xk, xi, fx, es, max_iter )
%SECANT_METHOD Summary of this function goes here
%   Detailed explanation goes here

syms x;
Ea = 0;
ea = 100;
i = 1;
cdata  = {};
cdata(1, : ) = {"xi-1", "xi", "xi+ 1", "Ea", "ea"};
```

```
while i<= max_iter

    xj = xi - eval((subs(fx,x,xi))) * (xk - xi) / (eval((subs(fx,x,xk))) - eval((subs(fx,x,xi))));

    Ea = abs(xj - xi);
    ea = abs(Ea / xj) * 100;

    if(ea < es)
        fprintf('iter: %d xi-1: %f,xi: %f,xi+1: %f,Ea: %f,ea: %f\n',i, xk, xi, xj,Ea, ea);
        cdata( i + 1, : ) = {xk, xi, xj, Ea, ea};
        break;
    end
    if(xj ~= 0)
        ea = (Ea / xj)  * 100;
        cdata( i + 1, : ) = {xk, xi, xj, Ea, ea};
  %      fprintf('xj: %f,xi: %f,xk: %f,ea: %f\n',xj, xi, xk, ea);
    end

    fprintf('iter: %d xi-1: %f,xi: %f,xi+1: %f,Ea: %f,ea: %f\n',i, xk, xi, xj,Ea, ea);
    xk = xi;
    xi = xj;
    i = i + 1;
end
xr = xj;
```

end

6. BIERGE VIETA

```
function [ root, cdata ] = bierge_vieta( a_arr, po, err, max_iter)
i = 2;
temp = size(a_arr);
temp = temp(1,2);
cdata = {};
cdata(1, :) = {"po", "Ea", "ea"};
cdata(2, :) = {po, NaN, 100};
ea = 100;
while i <= max_iter + 1
   b_arr = ones(1, temp);
   c_arr = ones(1, temp - 1);
   j = 2;
   while j <= temp
      b_arr(1, j) = a_arr(1, j) + po *  b_arr(1, j - 1);
      j = j + 1;
   end
   j = 2;
   while( j <= temp - 1)
      c_arr(1, j) = b_arr(1, j) + po * c_arr(1,j - 1);
      j  = j + 1;
   end
```

```
    p = po - b_arr(1,end) / c_arr(1,end);
    if(p ~= 0)
        Ea = abs(p - po);
        ea = abs(Ea / p) * 100;
        cdata(i+ 1, :) = {p, Ea, ea};

        if(abs(ea) < abs(err))
            root = p;
            fprintf('iter: %d, xi: %.18f, Ea: %.18f, ea: %.18f\n',i,p,Ea,ea);
            break;
        end
    end
    fprintf('iter: %d, xi: %.18f, Ea: %.18f, ea: %.18f\n',i,po,Ea,ea);

    po = p;
    i = i  + 1;



end
root =  po;
```

THE GENERAL METHOD USED

THE BONUS PART (GAUSSIAN JORDAN)

```
function x= GaussianJordan( a,b )
    n = length(b);
    %b = b';
    rankA = rank(a);
    a = [a b] ;
    rankAB = rank(a);
    if(rankA == rankAB) && rankA==n ,
   x = zeros(n,1);
   for i=1 : n % forward elimination
  for j=1 : n
      if(j==i),
      else
     z = a(j,i) / a(i,i) ;
     a( j , : ) = a( j , : ) - a( i , : )*z ;
      end
  end
   end
   for i=1 : n
  x(i) = a(i,n+1)/a(i,i);
   end
    else
   fprintf('Matrix A is singular matrix');
    end
end
```

Full analysis and run of equations

RUN OF PROBLEMS , EQUATIONS AND PROBLEMATIC FUNCTIONS

In case of bisection method:

- Function changes sign but root does not exist

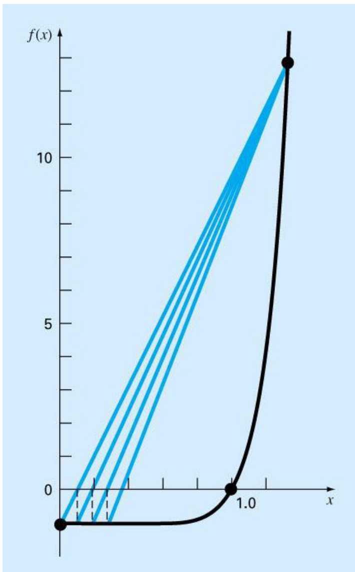
$f(x) = 1/x$

- If a function $f(x)$ is such that it just touches the x-axis it will be unable to find the lower and upper guesses such as: $f(X) = x^2$

In case of false-position method:

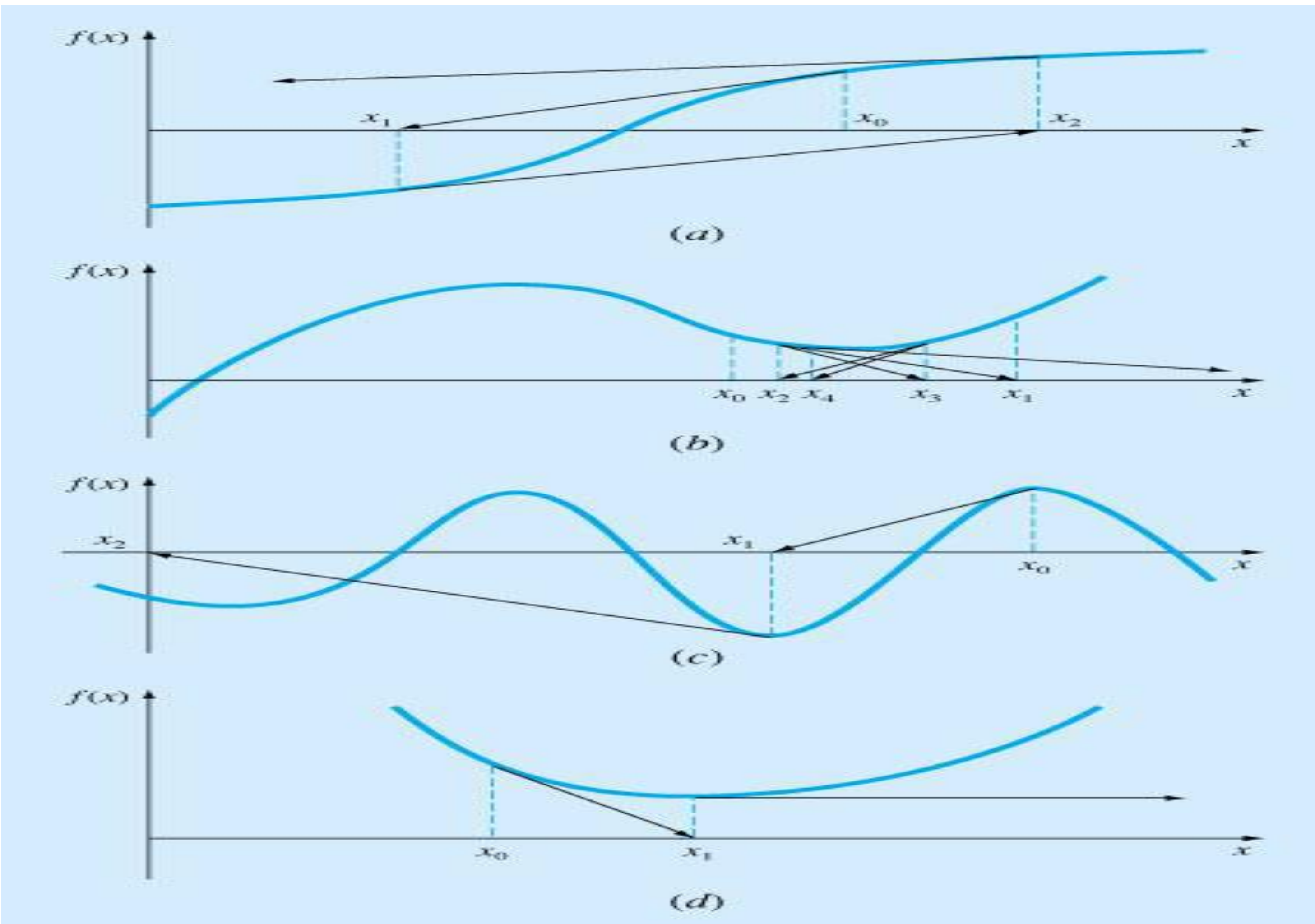
It can sometimes lead to problems and pitfalls such as this example

- Its very ,very slow



In case of Newton raphson method:

- Sometimes its very slow such as : f(x) = x^10 -1, (The Above figure)
- An inflection point (f''(x)=0) at the vicinity of a root causes divergence.(a)
- A local maximum or minimum causes oscillations.(b)
- A zero slope causes division by zero. (d)
- And also it doesn't work for multiple even roots that's why we found a modification for newton Raphson

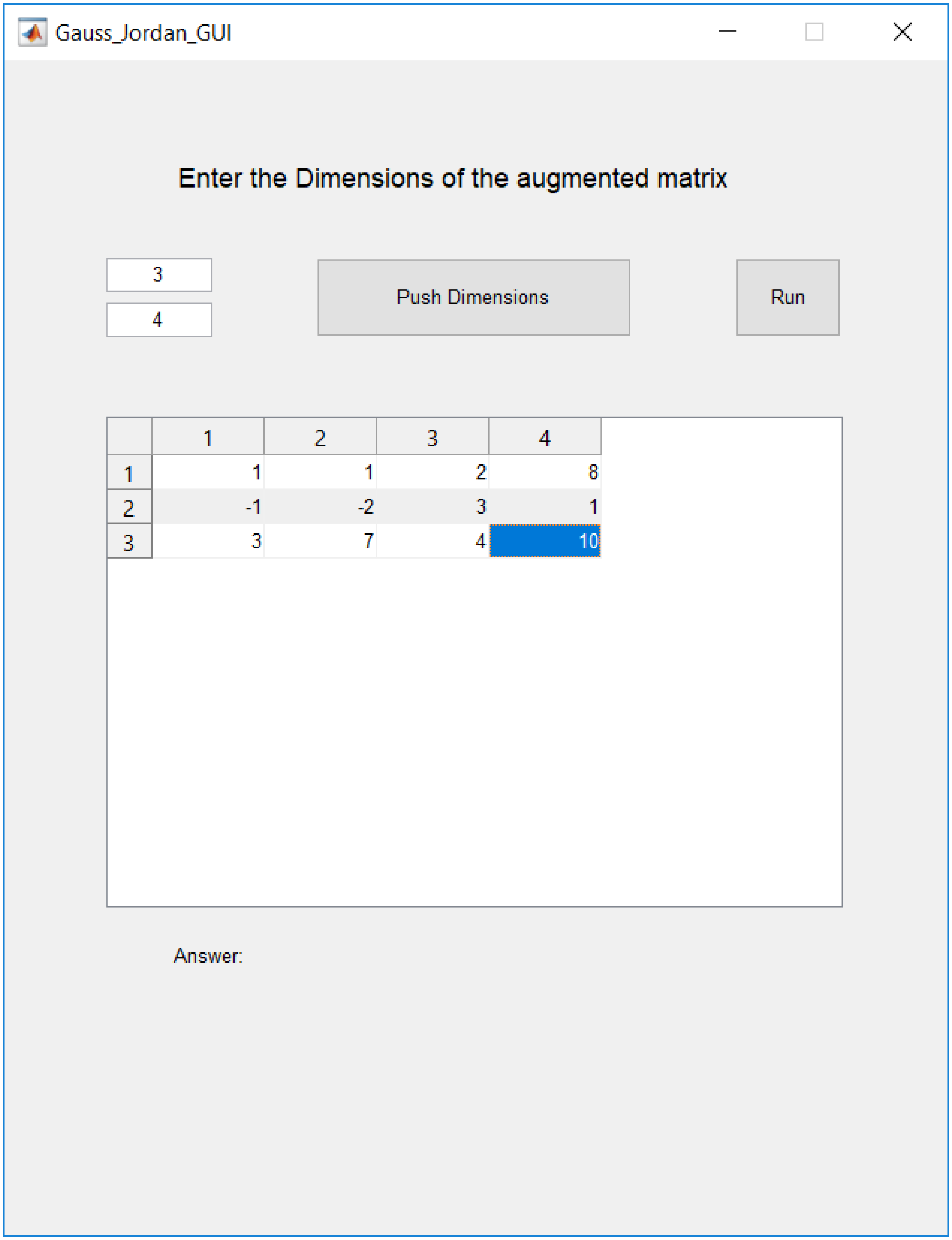


In case of gaussian jordan method:

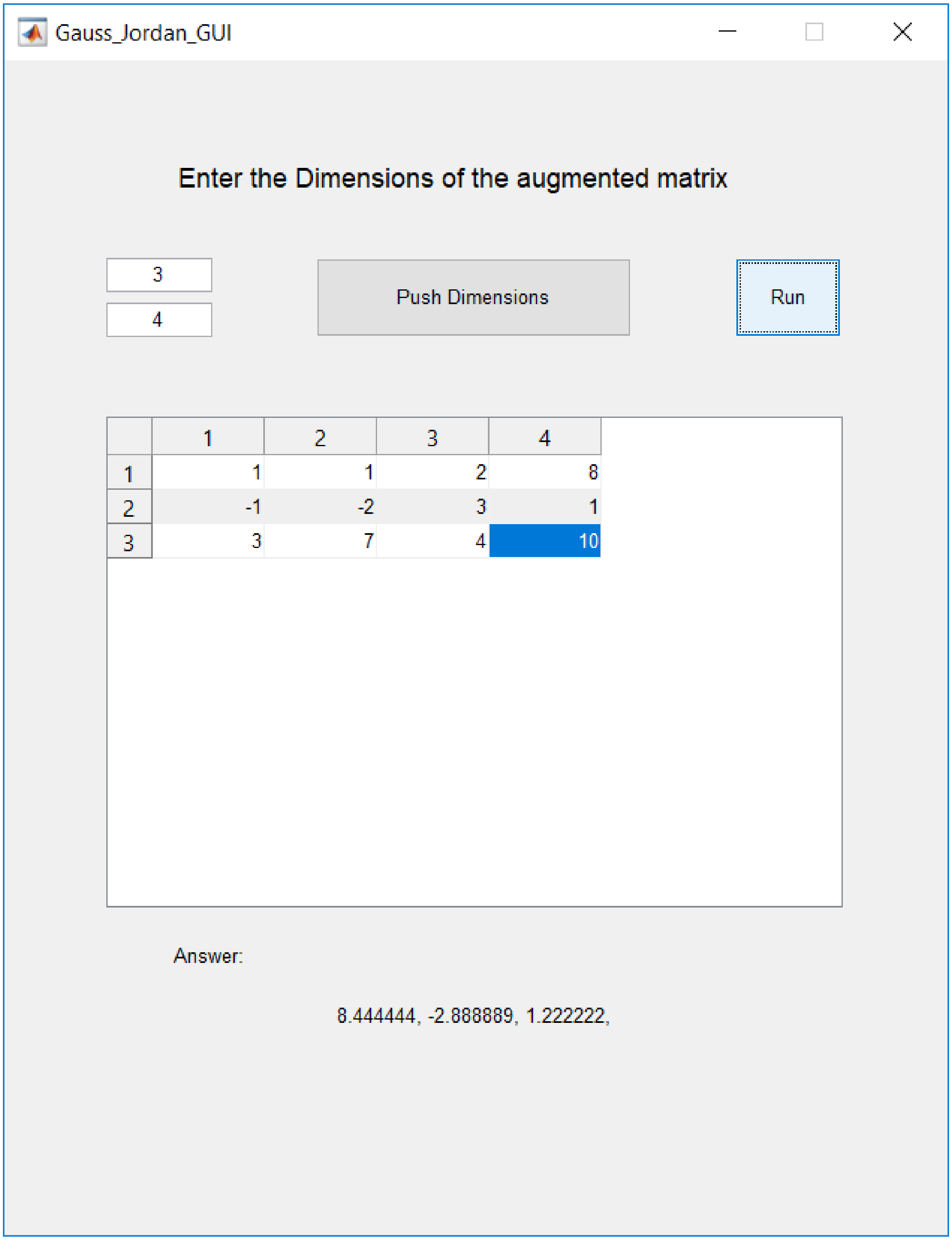
- Division-by-zero can cause problems
- Round-off error is high
- ill-conditioned systems.

SAMPLE RUNS AND SNAPSHOTS

Gauss Jordan method :

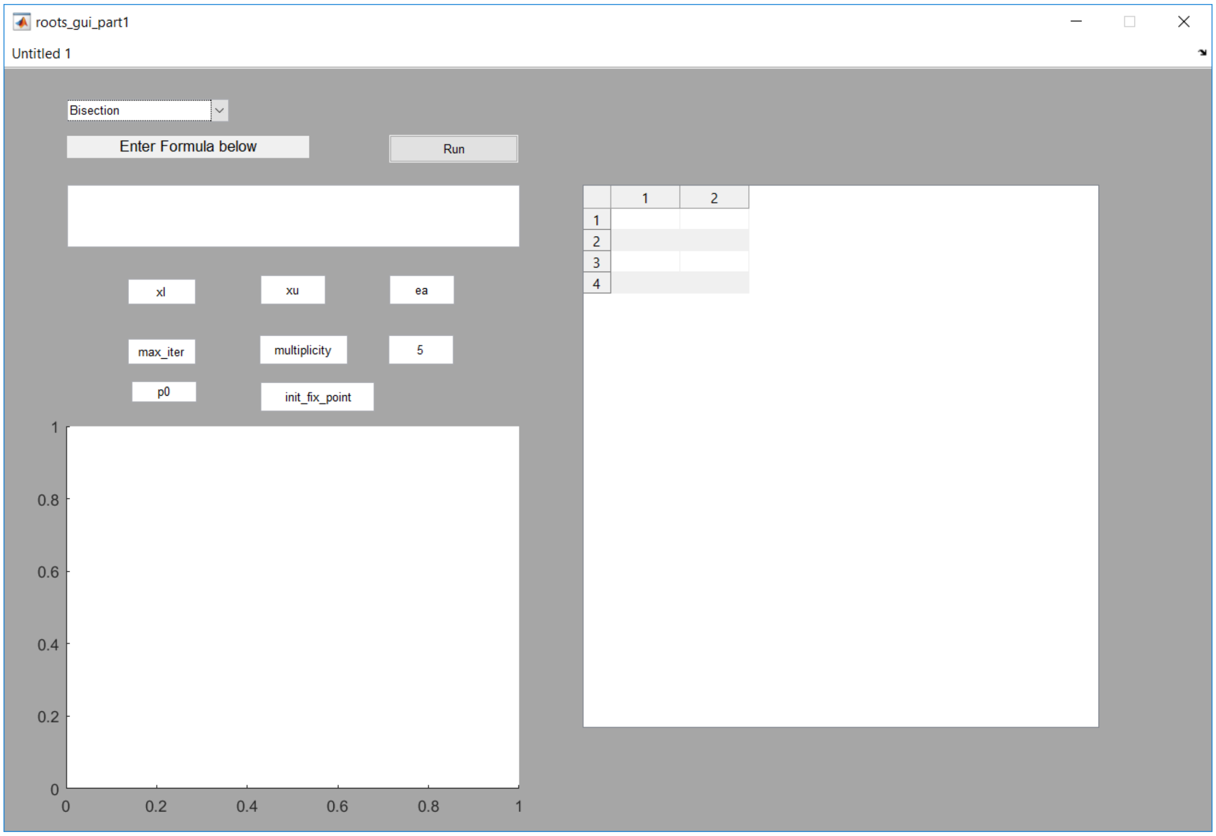


The Answer will be:

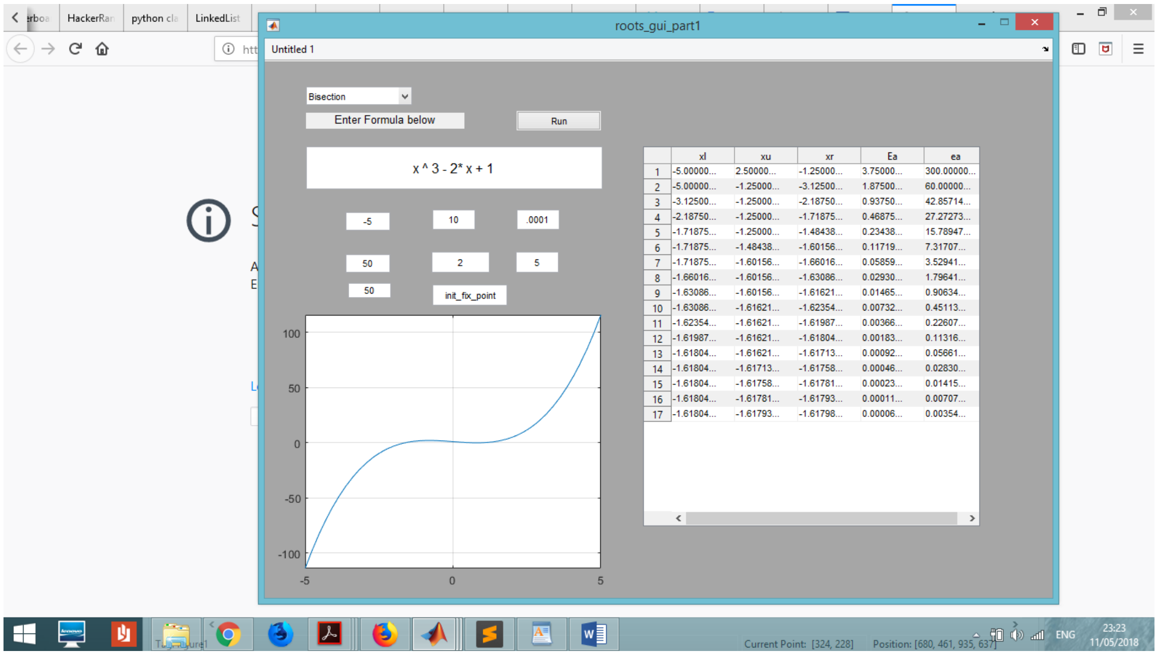


Which are the values of the roots x1,x2,x3

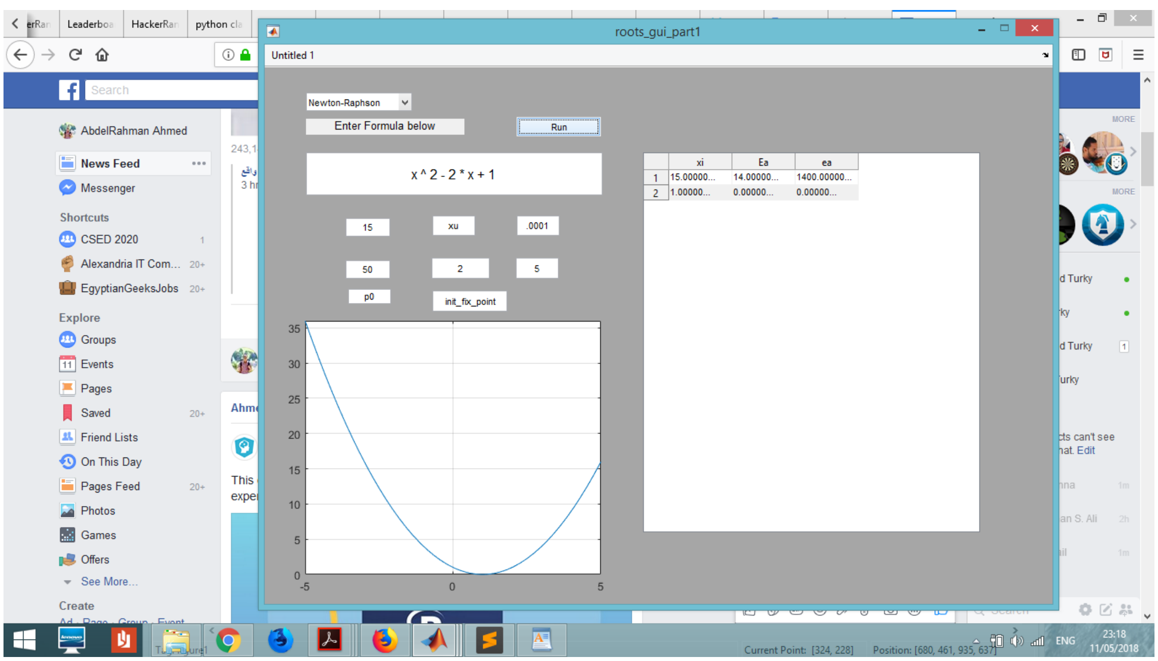
The Whole GUI:



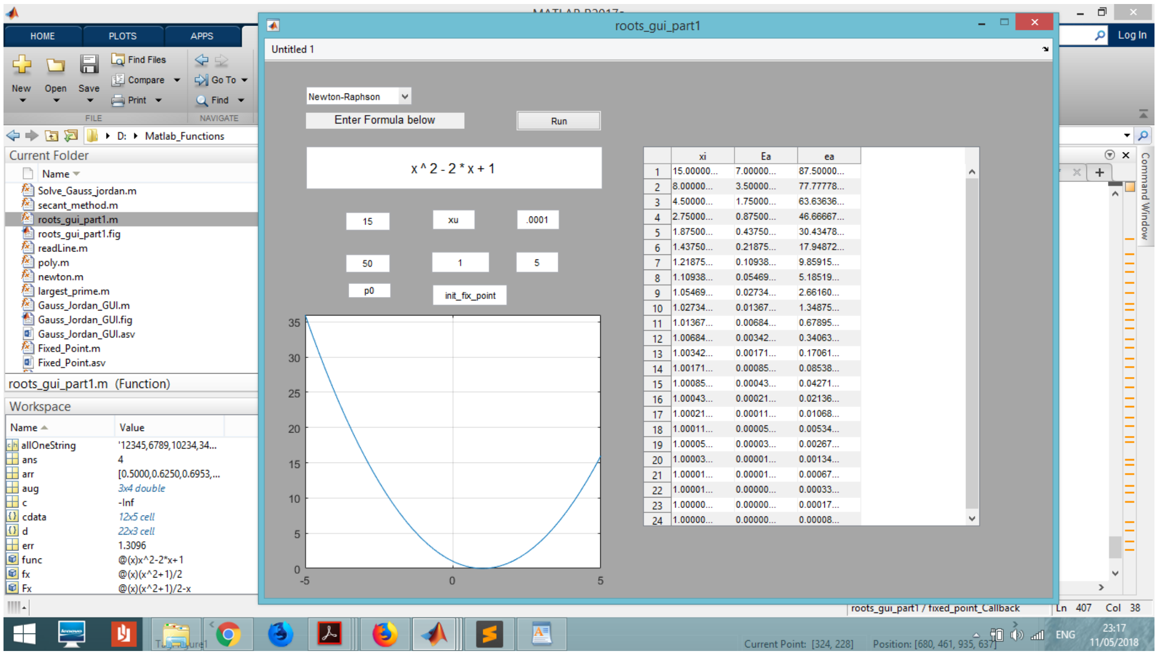
Bisection method :



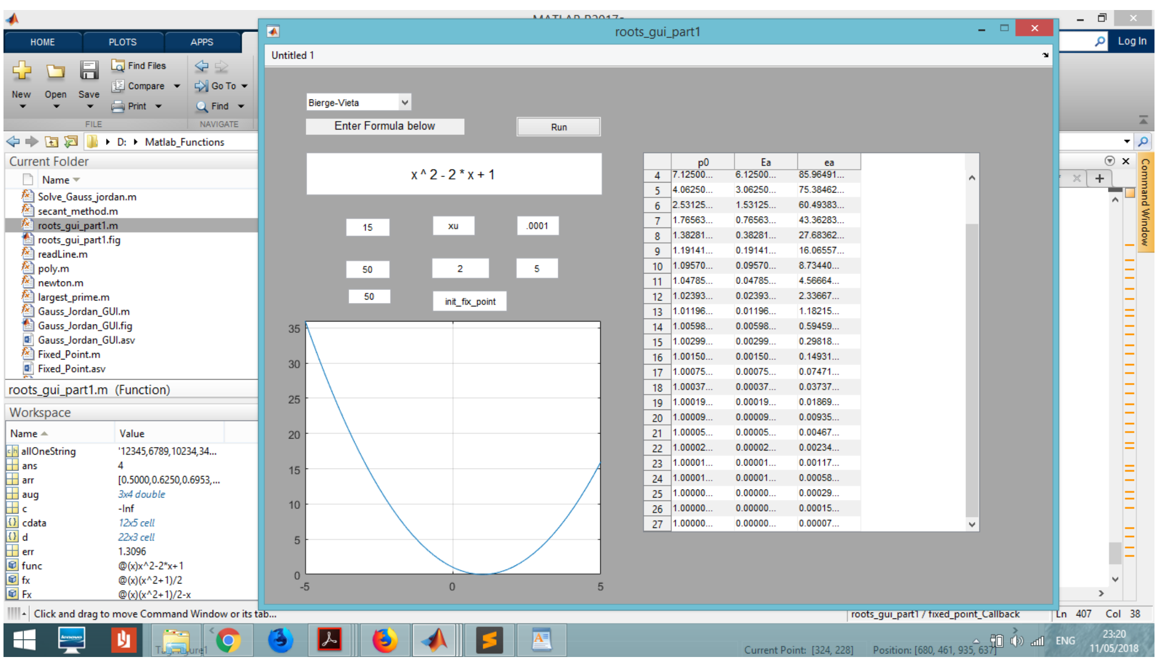
Newton Raphson with m = 2



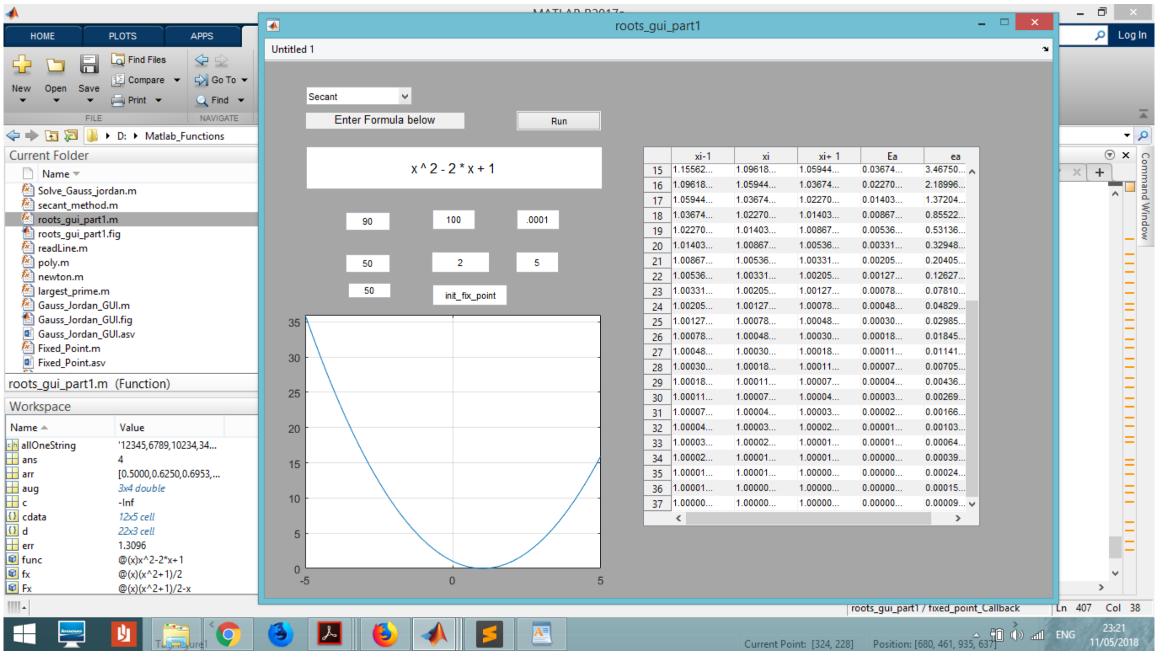
Newton Raphson example:



Bierge vieta



Secant method


roots_gui_part1
Untitled 1
Secant
Enter Formula below
Run
x ^ 2 - 2 * x + 1
init_fix_point
xi-1
xi
xi+ 1
Ea
ea
Current Folder
Workspace
roots_gui_part1.m (Function)

Numerical Assignment

Part 2

Interpolation

Names:

Khalil Ismail Khalil (23)
Ahmed Mohamed El-Bawab (08)
Abdelrahman Ahmes Torki (36)
Abdelrahman Said Ali (39)
Ziad Elbanna (27)

Pseudo-code:

(1)Newton Method

1)Newton_divided_refrence(x , y , order)

```
n >> order + 1
a[1] >> y[1]
Loop k=1 >> n-1
    d[k][1] >> (y[k+1] - y[k])/(x[k+1] - x[k]);
End Loop
Loop j=2 >> n-1
    Loop k=1 >> n-j
        d[k][j] >> (d[k+1][j - 1] - d[k][ j - 1])/(x[k+j] - x[k]);
    End Loop
End Loop
Loop j=2 >> n
    a[j] = d[1, j-1];
```

```
        End Loop
        syms z;
        p >> a[1];
        Loop i=2 >> order+1
                term >> a[i];
                Loop j=1 >> i-1
                        temp >> z - x[j];
                        term >> expand(term * expand(temp)) ;
                End Loop
                p >> expand(p + expand(term));
        End Loop
        x >> vpa(expand(p));
End
```

```
2)querey_Driver( x , y , order , querey )
        P >> Newton_divided_refrence( x , y , order);
        syms z;
        Loop i=1 >> length of querey
                z >> querey(i);
                a[i] >> subs(P);
        End Loop

End
```

(2)Lagrange Method

```
1)Lagrange( O,X,Y )
        syms x
        F >> 0;
        Loop i=1 >> O+1
                l >> 1;
                Loop j=1 >> O+1
                        if (j != i)
                                l >> expand(l* (x - X[j]) / (X[i]-X[j]));
                End Loop
        End Loop
```

```
        F >> expand(F + l * Y[i]);
        F >> vpa(F,10);

End

2)ValueOfLagrange( O,X,Y,V )
        F >> Lagrange(O,X,Y);
        R >> zeros(size(V));
        Loop i = 1 >> length of V
           x>>V(i);
           R(i)>>subs(F);
        End Loop

End
```

Data Structure Used:

In methods (Newton_divided_refrence and Lagrange), we used matrix as data-structure, and we used built in MATLAB functions (syms, expand,vpa) so that we can get Polynomial Function.

In methods (querey_Driver and ValueOfLagrange), we used matrix as data-structure,
and we used built in MATLAB functions (syms,subs) so that we can get output of query values.

Analysis and Runs of Problems:

1)Analysis:

1- Newton Method:

For Newton_divided_refrence(x , y , order):

time = O(n^2 + 3n)

For querey_Driver(x , y , order , querey):

time = O(n)

2- Lagrange Method:

For Lagrange(O,X,Y):

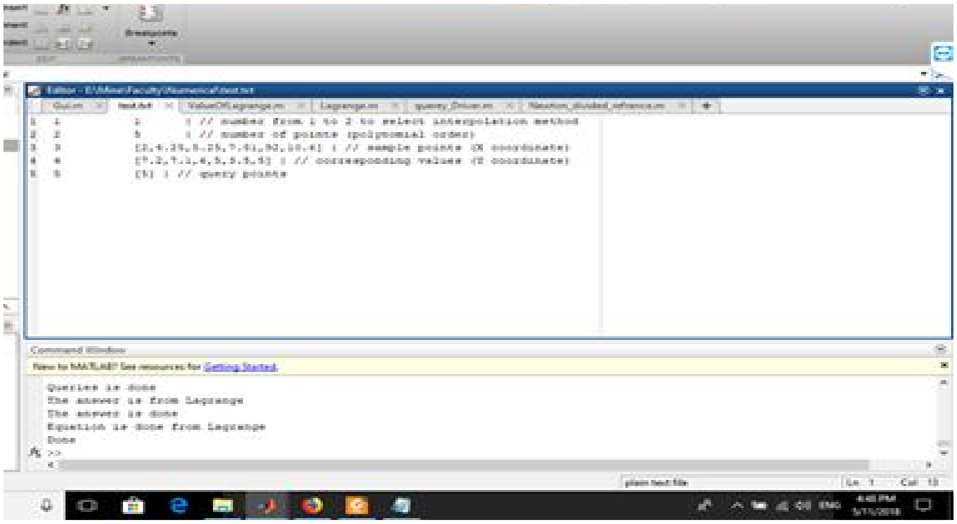
time = O(n^2)

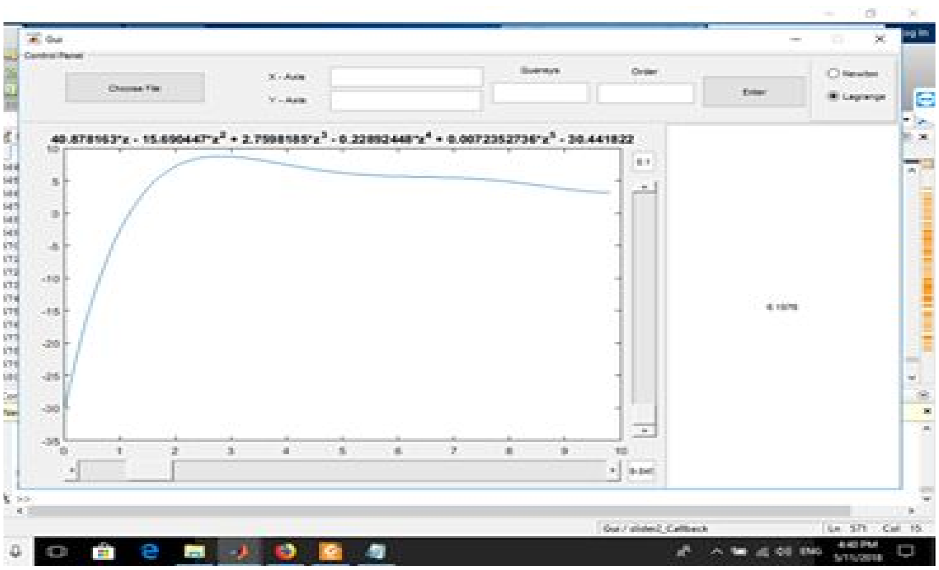
For ValueOfLagrange(O,X,Y,V):

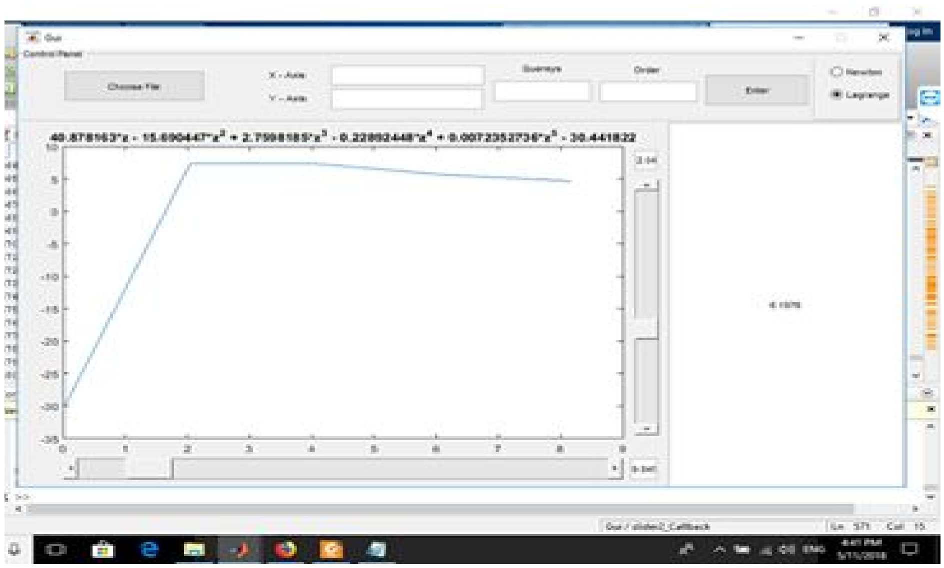
time = O(n)

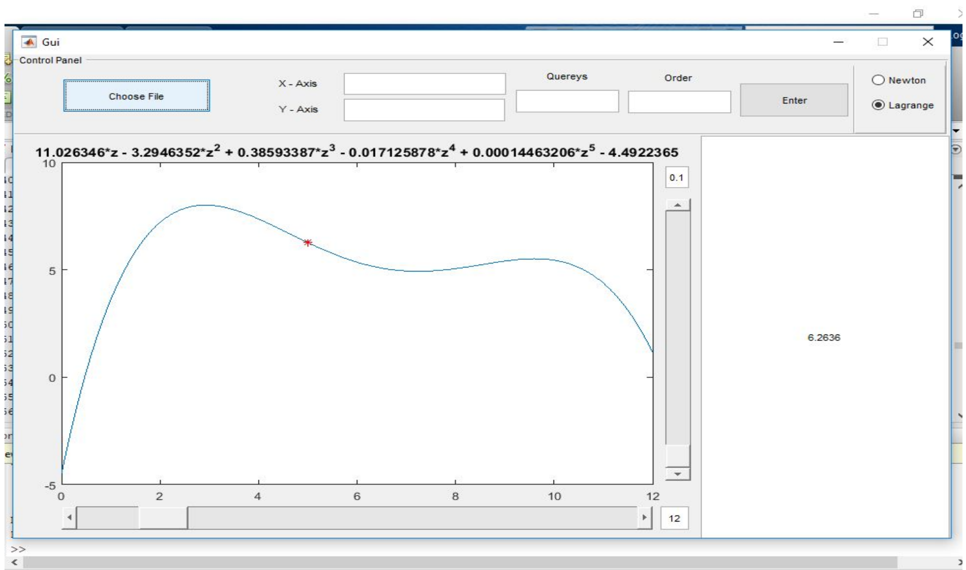
1)Problems:

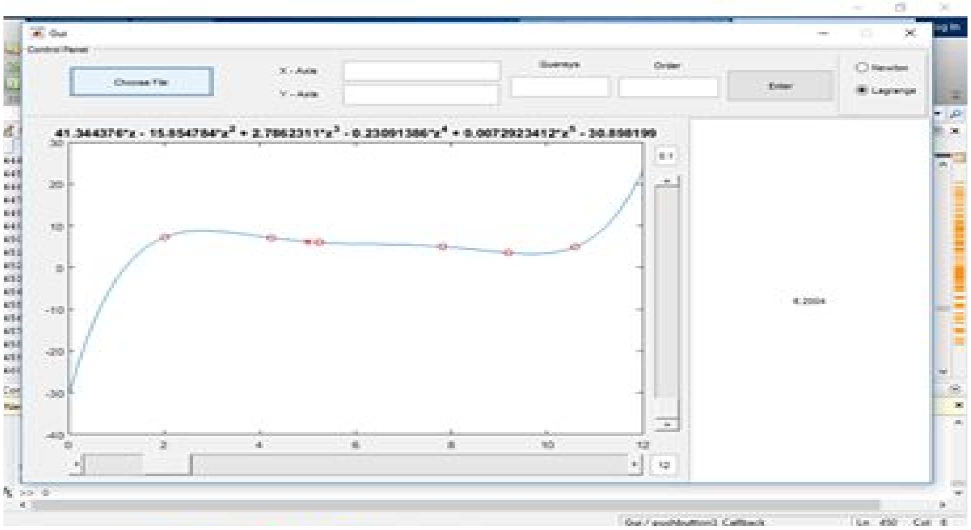
1) Using Newton Method


Command Window
Queries is done
The answer is from Lagrange
The answer is done
Equation is done from Lagrange
Done

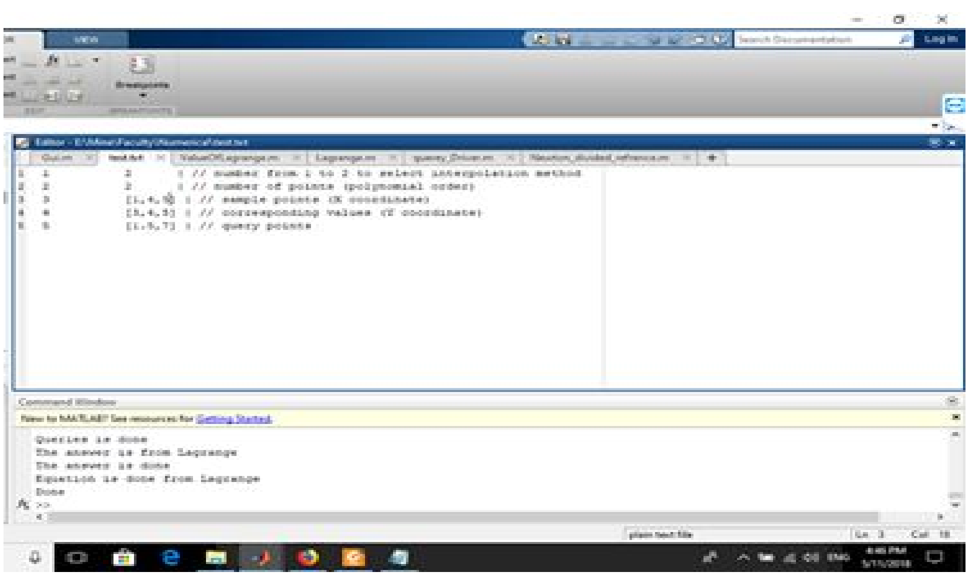

Control Panel
Choose File
X - Axis
Y - Axis
Order
Enter
Newton
Lagrange

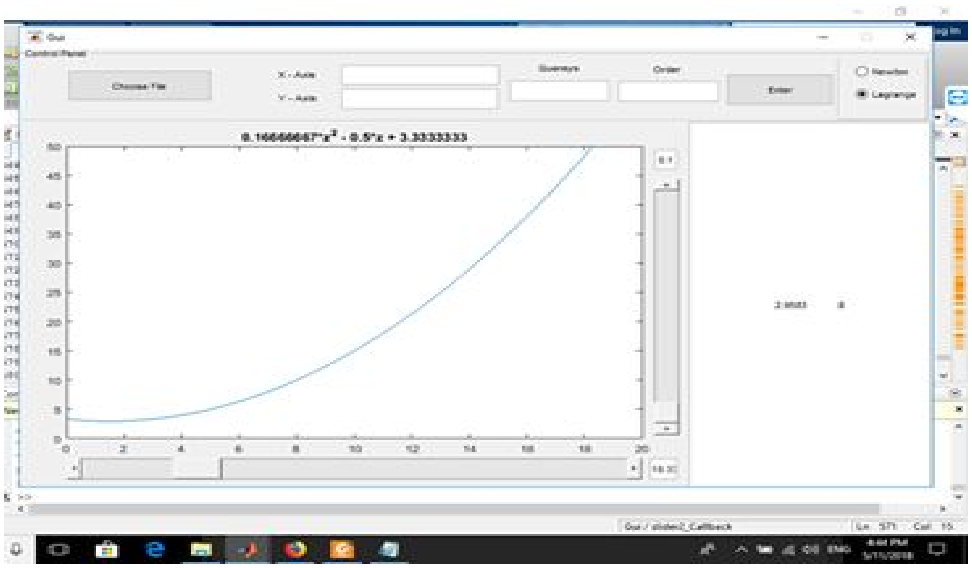

Gui
Control Panel
Choose File
X - Axis
Y - Axis
Quereys
Order
Enter
Newton
Lagrange


Gui
Control Panel
Choose File
X - Axis
Y - Axis
Quereys
Order
Enter
Newton
Lagrange
11.026346*z - 3.2946352*z^2 + 0.38593387*z^3 - 0.017125878*z^4 + 0.00014463206*z^5 - 4.4922365
0.1
12
6.2636

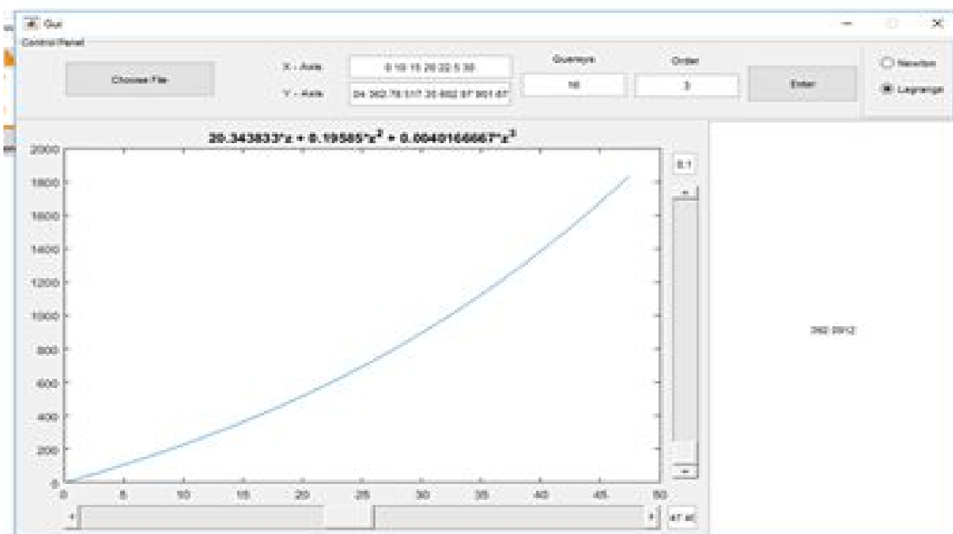


2) using Lagrange Method





3) using Lagrange Method

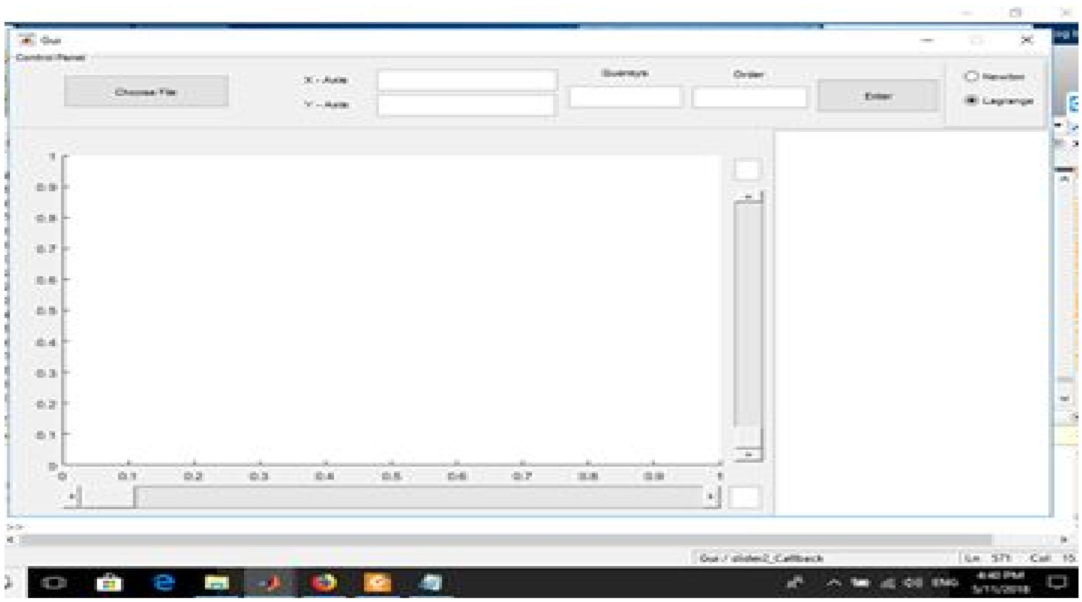


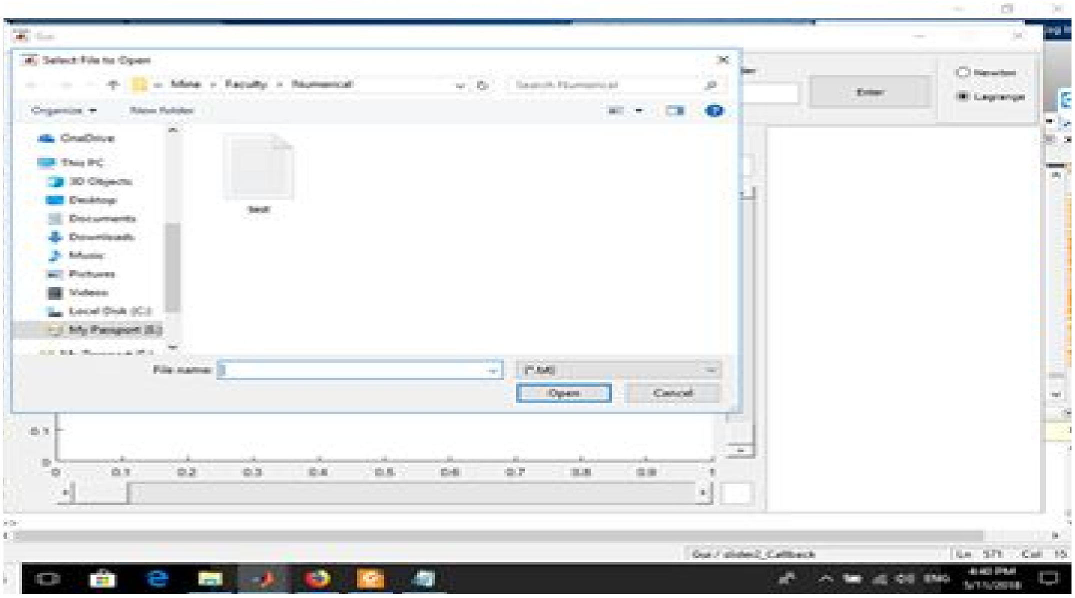
Problematic Functions:

1) Query Values must be in range Otherwise there will be Extrapolation.

2) Order must be less than (length of points - 1).

Sample Runs:




Select File to Open
Mine
Faculty
Numerical
Search Numerical
Organize
New folder
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
My Passport (E:)
test
File name:
Open
Cancel
Enter
Newton
Lagrange
Gui / slider2_Callback
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
ENG