Faculty of Engineering, Alexandria University
Ahmed Magdy Gamal (12)
Bishoy Nader Fathy (22)
Amr Mohamed Nasr Eldine (46)
Marc Magdi Kamel (53)
Michael Raafat Mikhail (55)

Matlab - Roots Finder

17th May 2017

Pseudo-Code Phase 1

- **Bracketing:**

○ Bisection Method

```
function [xxl, xxu, xxr, err, fxxr]  = bisectionMethod(func, l, u, eps, maxI)
    it = 1;
    functon_left = func(l);
    xr = 0;
    while it <= maxI
        xrOld = xr; // last x right
        xr = (l+u)/2;
        xxu(it) = u;
        xxl(it) = l;
        xxr(it) = xr;
        err(it) = abs(xr-xrOld);
        functon_right = func(xr);
        fxxr(it) = functon_right;
        test = functon_right * functon_left;
        if(test > 0)
            l = xr;
            functon_left = functon_right;
        end
        if(test < 0)
            u = xr;
        end
        if(test == 0) // root found
            break;
        end
        if(err(it) <= eps && it > 1) // small error
            break;
        end
        it = it+1;
    end
    if (it <= maxI)
        xxu = xxu(1:it);
        xxl = xxl(1:it);
        xxr = xxr(1:it);
        err = err(1:it);
        fxxr = fxxr(1:it);
    end
    return;
end
```

○ False Position (Regula Falsi)

```
function [xxl, xxu, xxr, err, fxxr, flag] = falsePosition(f, l, u, eps, maxI)
    fl = f(l);
    fu = f(u);
    it = 1;
    initialze();
    while it <= maxI
        if (fl * fu > 0)
            % Cannot detect root
            flag = 1;
            break;
        end
        xrOld = xrNew;
        % false position formula.
        xrNew = ( l * fu - u * fl ) / (fu - fl);
        xxu(it) = u;
        xxl(it) = l;
        xxr(it) = xrNew;
        err(it) = abs(xrNew-xrOld);
        fr = f(xrNew);
        fxxr(it) = fr;
        test = fr * fl;
        if(test > 0) % go to the right.
            l = xrNew;
            fl = fr;
        end
        if(test < 0) % go to the left.
            u = xrNew;
            fu = fr;
        end
        if(test == 0) % one of upper and lower is the root.
            break;
        end


        it = it+1;
    end
    if (it <= maxI)

        % return vectors filled with data.
        xxu = xxu(1:it);
        xxl = xxl(1:it);
        xxr = xxr(1:it);
        err = err(1:it);
        fxxr = fxxr(1:it);
    end
    return;
end
```

- **Open Methods**
    - **Fixed Point**

```
function [xs, err, fxxr] = fixedPoint (f, g, xi, eps, maxI )
    initialze();
    xs(1) = xi;
    err(1) = 0;
    fxxr(1) = f(xi);
    while it <= maxI;
        xOld = xi;
        % new point is a function of the previous point.
        xi = g(xi);
        xs(it) = xi;
        fxxr(it) = f(xi);
        err(it) = abs(xi-xOld);
        if(err(it) <= eps && it > 1) % small error i.e root found.
            break;
        end
        it = it+1;
    end
    if (it <= maxI)
        % return vectors filled
        xs = xs(1:it);
        err = err(1:it);
        fxxr = fxxr(1:it);
    end
    return;
end
```

Newton Raphson

```
function [ xs, err, fxs, dfxs, flag, message] = newtonRaphson(f,dF,xi,eps,maxI)
    it = 1;
    initialize();
    flag = 0;
    xOld = xi;
    while it <= maxI;
        fxs(it) = f(xi);
        dfxs(it) = dF(xi);
        xs(it) = xi;
        err(it) = abs(xi-xOld);
        if(abs(dfxs(it)) < 1E-8) % invalid state
            flag = 1;
            message = 'Division by zero!';
            it = it+1;
            break;
        end;
        if(err(it) <= eps && it > 1)
            break; % root was found.
        end;
        xOld = xi;
        xi = xi - fxs(it)/dfxs(it);
        it = it+1;
    end;
    if (it <= maxI)
        % root found. return the vectors with their data.
        xs = xs(1:it);
        fxs = fxs(1:it);
        dfxs = dfxs(1:it);
        err = err(1:it);
    end
    return;
end
```

Secant

```
function [ x0s, xis, fx0s, fxis, err] = secant(f,x0,xi,eps,maxI)
    initialize();
    it = 1;
    xOld = x0;
    while it <= maxI;
        x0s(it) = x0;
        xis(it) = xi;
        fx0s(it) = f(x0);
        fxis(it) = f(xi);
        err(it) = abs(xi-xOld);
        if(err(it) <= eps && it > 1)
            % root was found.
            break;
        end;
        % next Step
        xOld = xi;
        xi = xi - f(xi)*(x0-xi)/(f(x0) - f(xi));
        x0 = xOld;
        it = it+1;
    end
    if (it <= maxI)

        % root found.
        % return the vectors with their data.
        x0s = x0s(1:it);
        xis = xis(1:it);
        fx0s = fx0s(1:it);
        fxis = fxis(1:it);
        err = err(1:it);
    end
    return;
end
```

- **Birge-Vieta**

```
function [xs, fxs, err] = birgeVieta (f, coeff, x0, eps, maxI )
    initialize();
    b(m) = coeff(m);
    c(m) = coeff(m);
    it = 2;
    xs(1) = x0;
    fxs(1) = f(x0);
    while it <= maxI;
        xOld = x0;
        for i = m-1:-1:1
            b(i) = coeff(i) + xOld * b(i+1);
            c(i) = b(i) + xOld * c(i+1);
        end
        x0 = x0 - b(1) / c(2);
        xs(it) = x0;
        err(it) = abs(x0-xOld);
        fxs(it) = f(x0);
        if(err(it) <= eps)
            % small error i.e. root found.
            break;
        end;
        it = it+1;
    end;

    % return vectors filled with their data.
    xs = xs(1:it);
    err = err(1:it);
end
```

- **General Algorithm**

  We try for a specific number of iterations each iteration we check for 2 intervals of length 100,
- In each interval we check:
  - The boundaries of the interval if they are roots. If not we check:
  - The bisection condition between its two boundaries to find if it can find a root. If not we check
  - The fixed point condition and pass the midpoint of the interval as an initial point to find if it will diverge or not. If it diverges we finally check
  - Newton Raphson with the midpoint of the interval as an initial point.
- If root wasn't found we then go to the following interval in the +ve side and previous interval in the -ve side.

```
function [ xroot, flag, counter, boundL, boundU ] = general_method(f, df, g, dg, eps, iter)
    initialize();
    flag = 0;
    counter = 0;
    while((counter < iter))
        % check interval
        % boundaries(start, -start, start + step, - start - step)

        % apply bisection on both intervals
        [~, ~, rRight, ~, ~, rightFlag] = bisectionMethod(f, start, start + step, eps, iter);
        [~, ~, rLeft, ~, ~, leftFlag] = bisectionMethod(f, -1 * start, (-1 * start) - step, eps, iter);

        if(rightFlag && leftFlag) %No solution Try fixed point
            if (abs(dg((start + step)/2)) < 1)
               [rl, ~, ~] = fixedPoint(f, g, (start + step)/2, eps, iter);
               xroot = rl(end);
               flag = 1;
               return;
            elseif (abs(dg((-start - step)/2)) < 1)
               [rl, ~, ~] = fixedPoint(f, g, (-start - step)/2, eps, iter);
               xroot = rl(end);
               flag = 1;
               return;
            else
               % check Newton with midpoint
               [rl,  ~, ~, ~, newtonFlag, ~] = newtonRaphson(f, df, (start + step)/2, eps, iter);
               if(newtonFlag == 0)
                   xroot = rl(end);
                   flag = 1;

                    return;
                end
                [rl,  ~, ~, ~, newtonFlag, ~] = newtonRaphson(f, df, (-start - step)/2, eps, iter);
                if(newtonFlag == 0)
                    xroot = rl(end);
                    flag = 1;
                    return;
                end
            end
        elseif(leftFlag) % get result from right half of bisection
            flag = 1;
            xroot = rRight(end);
            return;
        elseif(rightFlag) % get result from left half of bisection
            flag = 1;
            xroot = rLeft(end);
            return;
        end
        boundL = -start - step;
        boundU = start + step;
        start = start + step; % change interval
        counter = counter + 1;
    end

end
```

Pseudo-Code Phase 2

- **Gaussian Elimination**

```
function [xxl, xxu, xxr, err, fxxr]  = bisectionMethod(func, l, u, eps, maxI)
    it = 1;
    functon_left = func(l);
    xr = 0;
    while it <= maxI
        xrOld = xr; // last x right
        xr = (l+u)/2;
        xxu(it) = u;
        xxl(it) = l;
        xxr(it) = xr;
        err(it) = abs(xr-xrOld);
        functon_right = func(xr);
        fxxr(it) = functon_right;
        test = functon_right * functon_left;
        if(test > 0)
            l = xr;
            functon_left = functon_right;
        end
        if(test < 0)
            u = xr;
        end
        if(test == 0) // root found
            break;
        end
        if(err(it) <= eps && it > 1) // small error
            break;
        end
        it = it+1;
    end
    if (it <= maxI)
        xxu = xxu(1:it);
        xxl = xxl(1:it);
        xxr = xxr(1:it);
        err = err(1:it);
        fxxr = fxxr(1:it);
    end
    return;
end
```

- ## Gaussian Jordan

```
function [x] = Gaussian_Jordan(a, b) % a the coeff, b the equations
a = [a, b];
n = length(a);
for i = 1 to n-1
    y = a(i,:);
    y = y/y(i);
    a(i,:) = y;
    for j = 1 to n-1
        if (i != j)
           a(j,:) = y * -1 * a(j,i) + a(j,:);
        end
    end
end
x = a(:,length(a))';
end
```

- ## Gauss Seidel

```
function [ x , s, absrel] = Gauss_Seidel(a,b,c,max,eps)
i = 1;
acc = inf;
x = c;
s = zeros(max,length(b));
absrel = zeros(max,length(b));
s(1,:) = c;
while (i <= max) && (acc > eps)
    acc = 0;
    for k = 1 to size(a,1)
        temp = x(k);
        x(k)= b(k);
        for j = 1 to size(a,1)
            if j != k
                x(k) = x(k) - a(k,j) * x(j);
            end
        end
        x(k) = x(k) / a(k,k);
        s(i + 1,k) = x(k);
        absrel(i + 1,k) = abs((temp - x(k)) / x(k))* 100;
        if ( abs(temp - x(k)) > acc)
            acc = abs(temp - x(k));
        end
    end
    i = i + 1;
end
s = s(1:i, :);
absrel = absrel(1:i, :);
end
```

- **LU Decomposition**

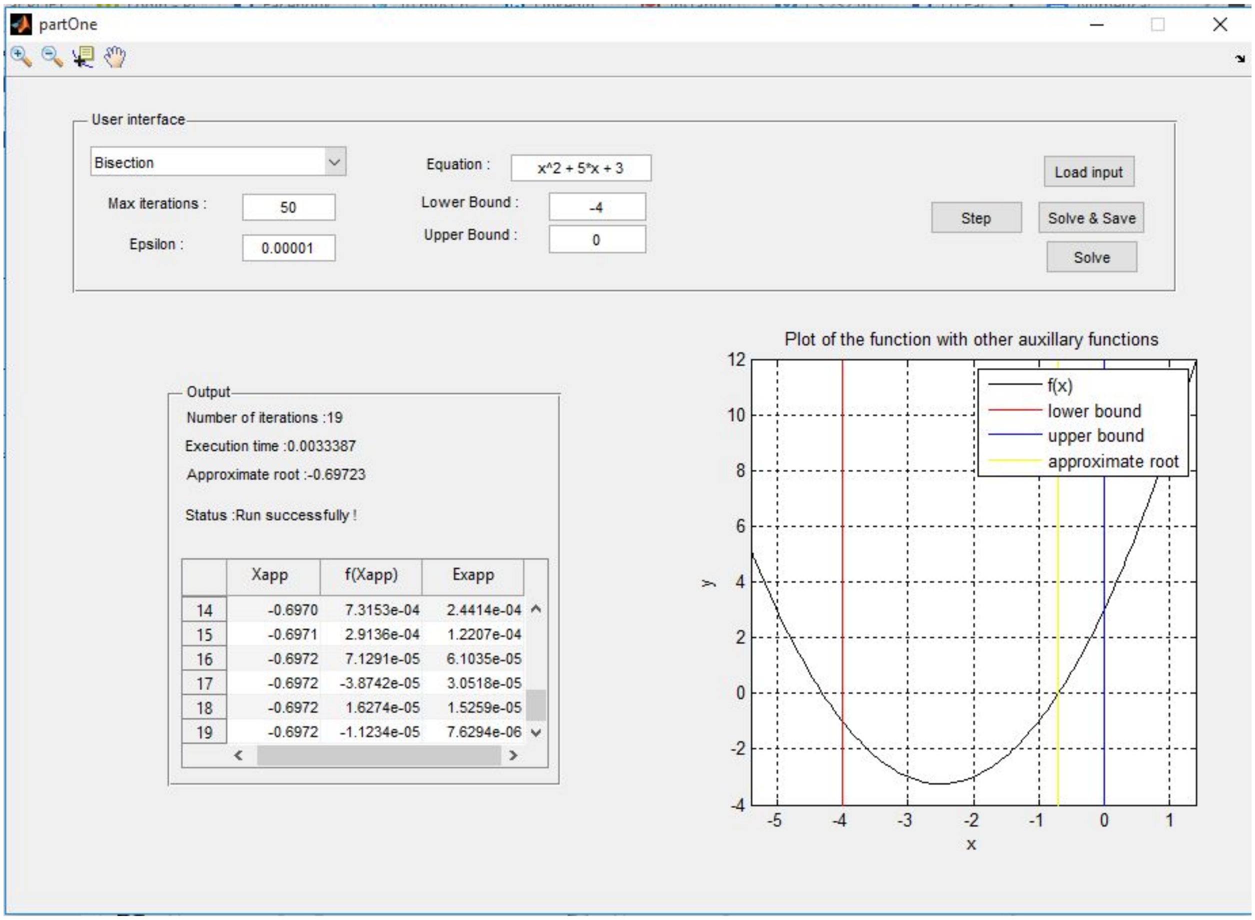
```
function [ x ] = LuDecomposition( a, b )
n = size(a, 1);
lu = a;
for i = 1 to n - 1
    for j = i + 1 : n
        factor = lu(j,i) / lu(i,i);
        lu(j,i:n) = lu(j,i:n) - factor .* lu(i,i:n);
        lu(j,i) = factor;
    end
end
y = zeros(n, 1);
% L Y = B
y(1) = b(1);
for i = 2 : n
    y(i) = b(i);
    for j = 1 : i - 1
        y(i) = y(i) - lu(i,j) *  y(j);
    end
end
% U X = Y
x(n) = y(n) / lu(n,n);
i = n - 1;
while (i > 0)
    x(i) = y(i);
    for j = i + 1 to n
        x(i) = x(i) - lu(i,j) * x(j);
    end
    x(i) = x(i) / lu(i,i);
    i = i - 1;
end

end
```

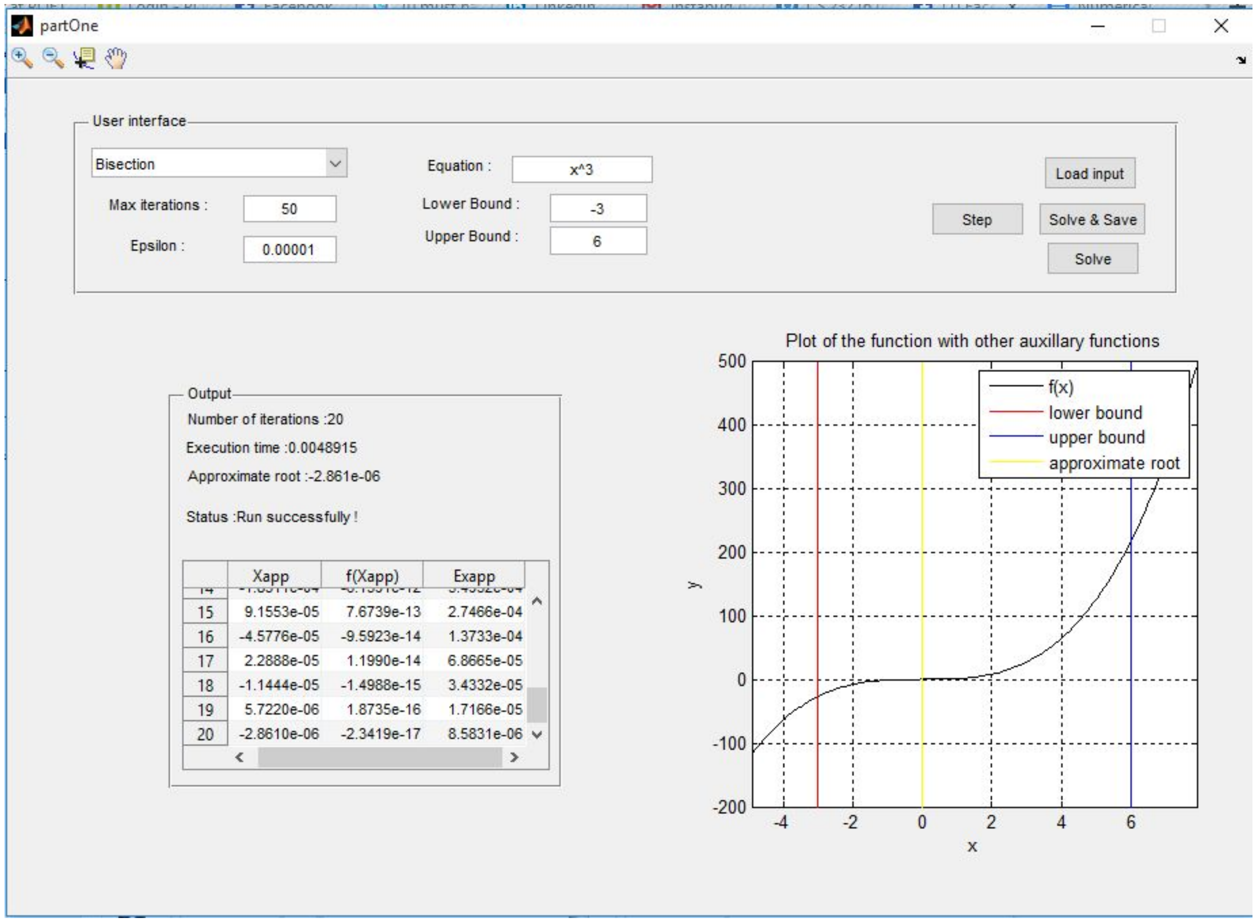
Sample Runs

Part One:

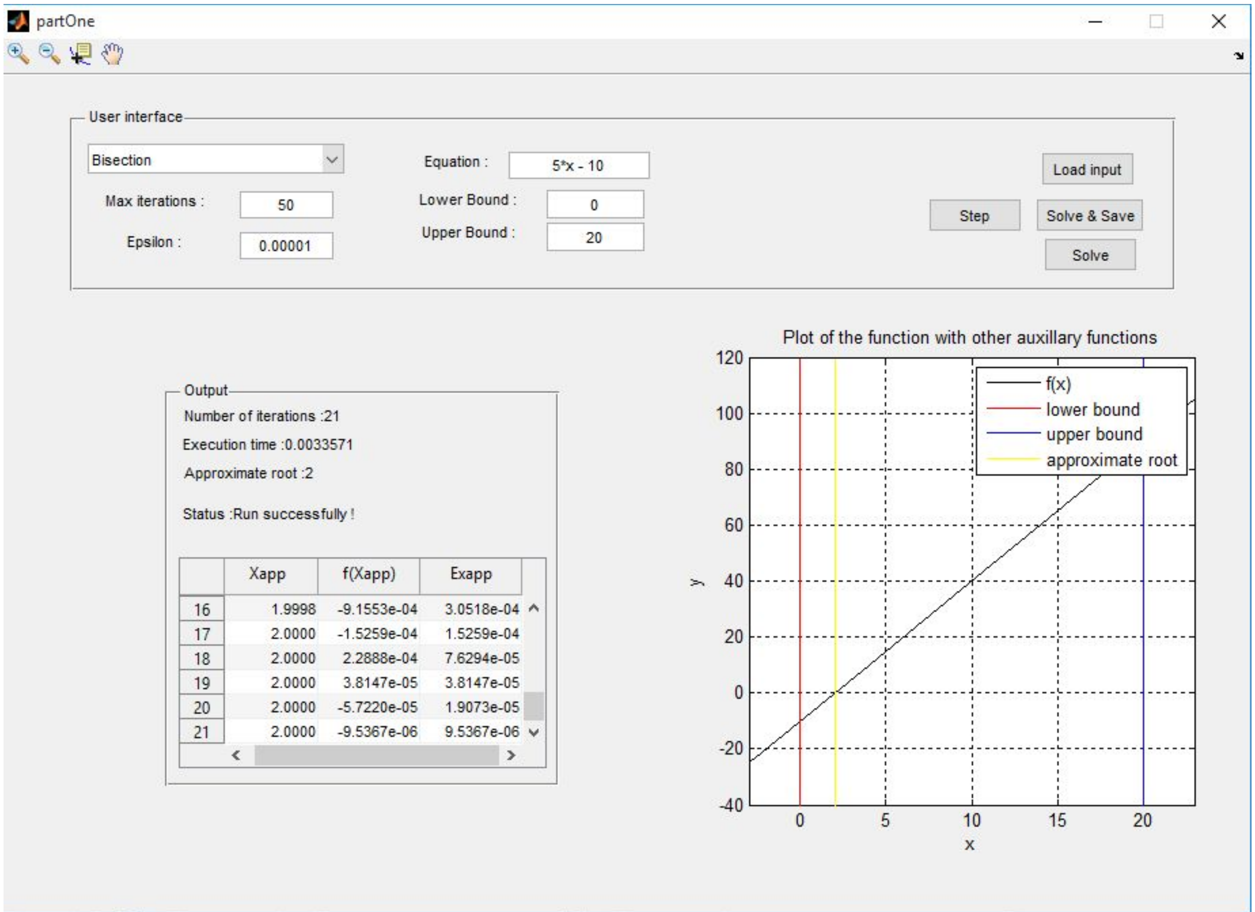
1. Using Bisection to solve x^2 - 5*x + 3



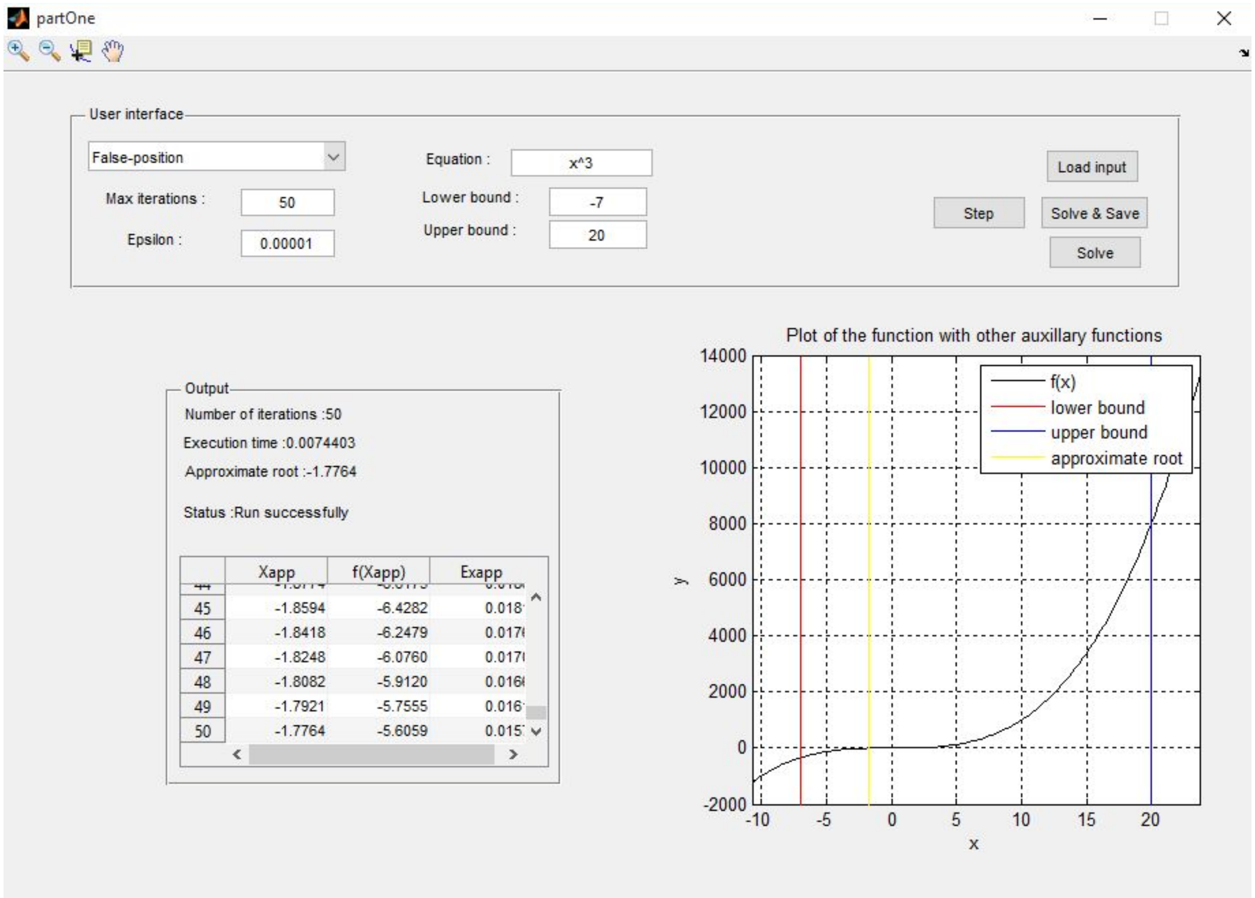
2. Using Bisection to solve x^3



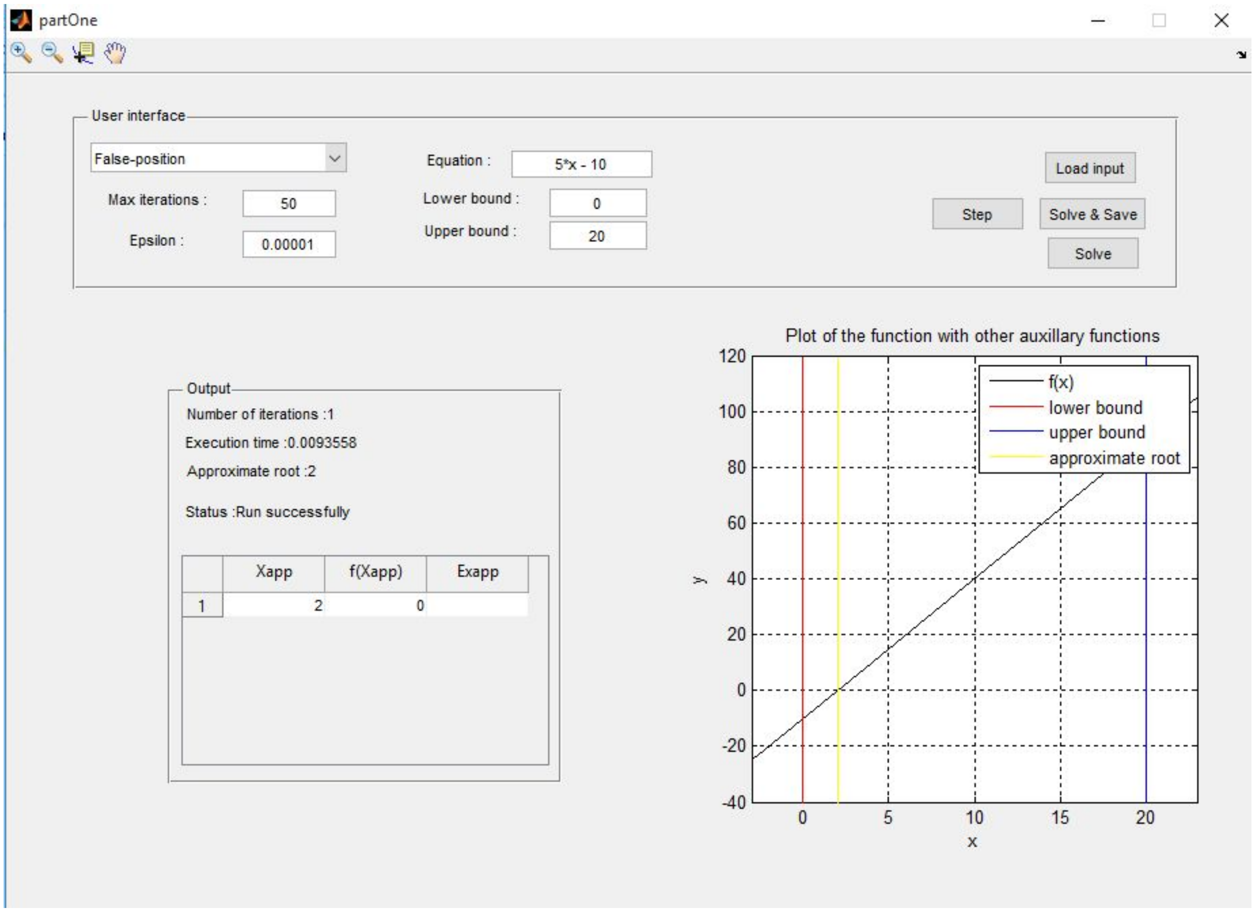
3. Using Bisection to solve 5*x - 10



4. Using False position to solve x*3

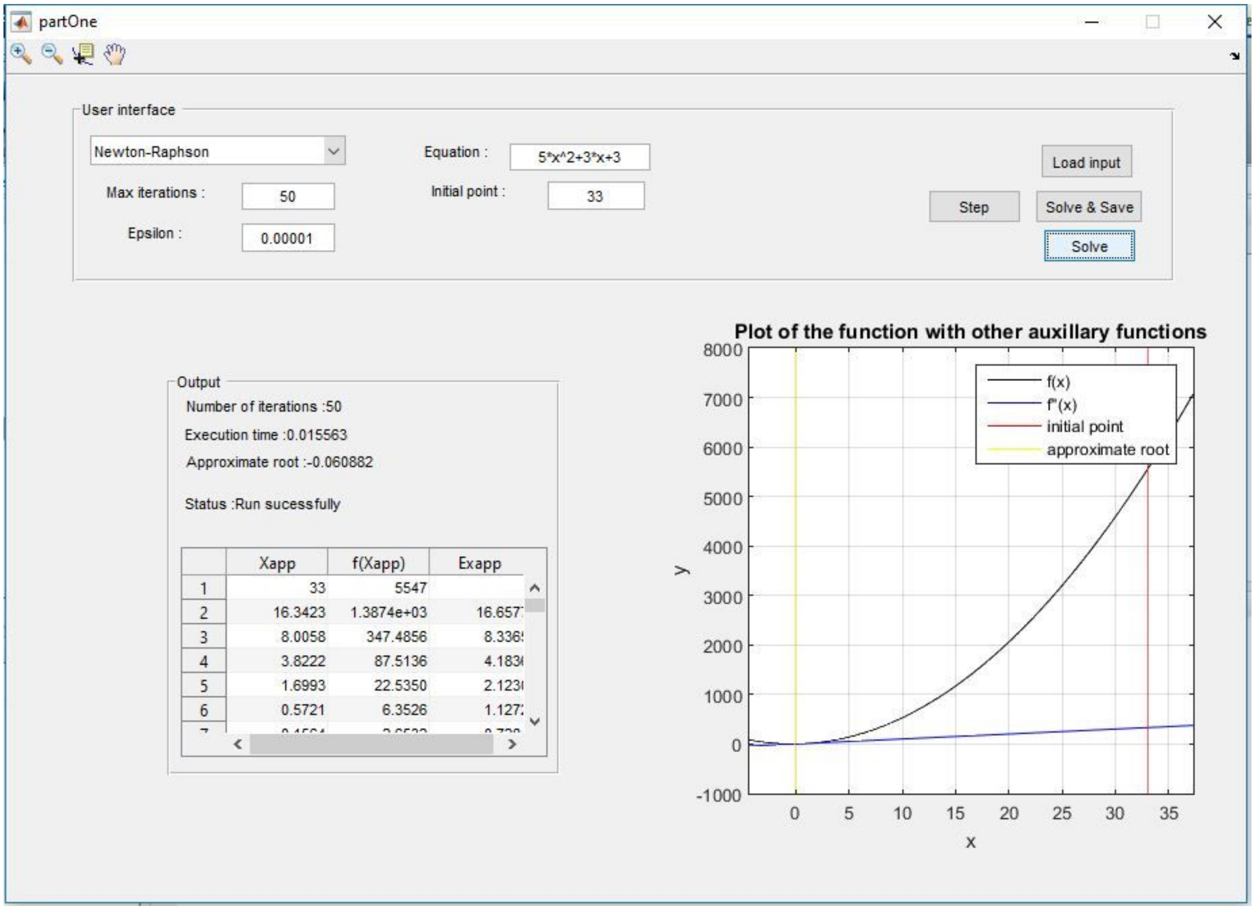


5. Using False position to solve 5*x - 10

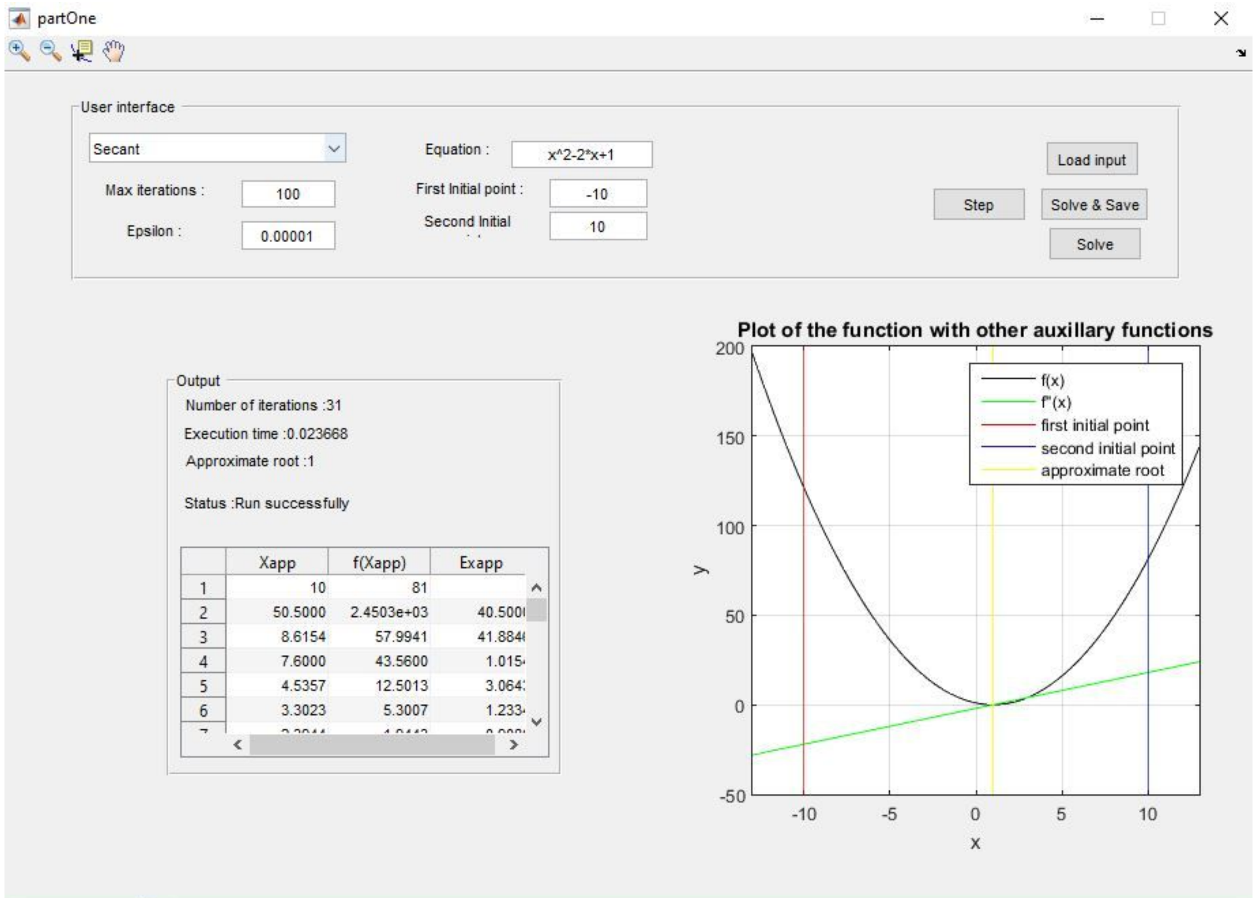


partOne
User interface
False-position
Max iterations : 50
Epsilon : 0.00001
Equation : 5*x - 10
Lower bound : 0
Upper bound : 20
Load input
Step
Solve & Save
Solve
Output
Number of iterations :1
Execution time :0.0093558
Approximate root :2
Status :Run successfully
Xapp
f(Xapp)
Exapp
1 2 0
Plot of the function with other auxillary functions
f(x)
lower bound
upper bound
approximate root
x
y

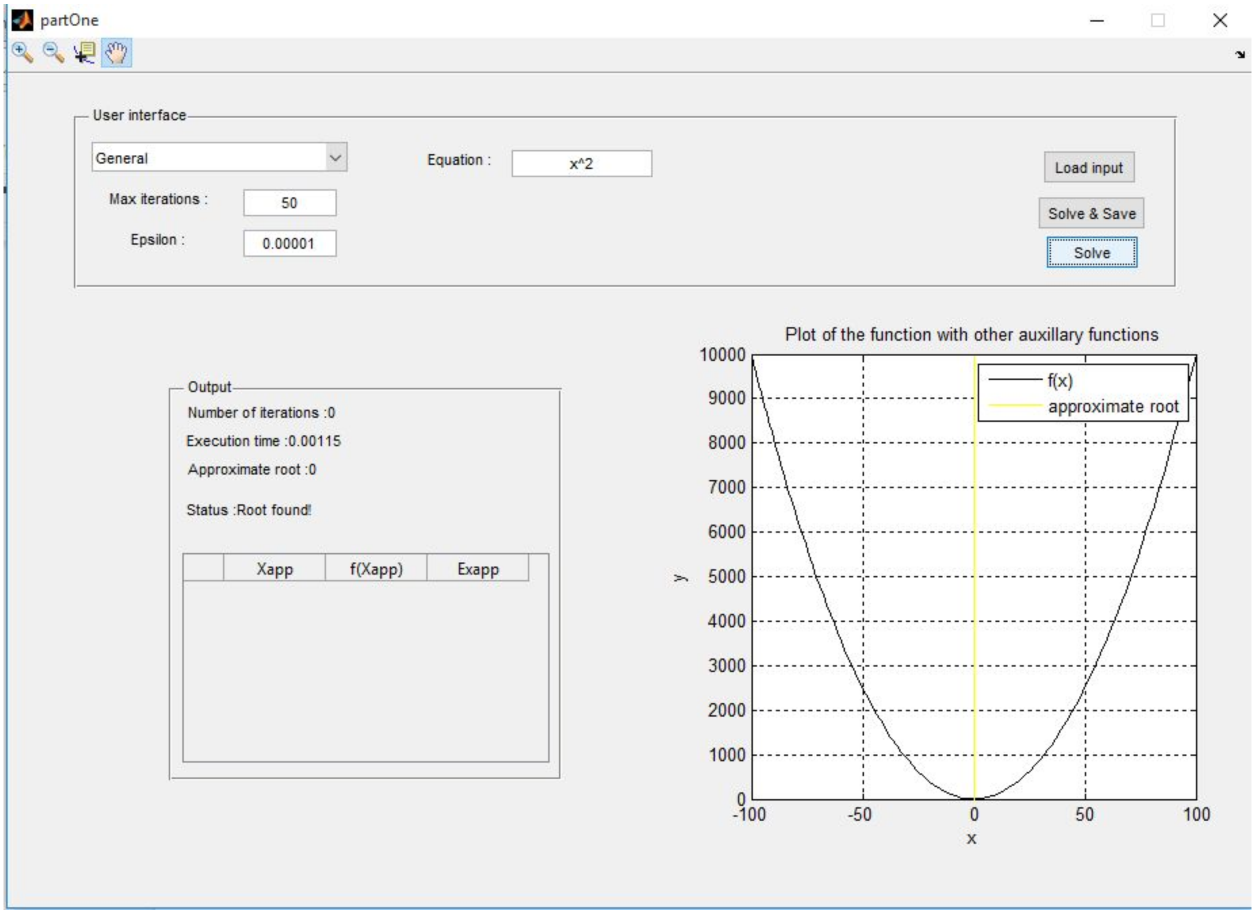
6. Using Newton to solve: 5X^2+3X+3



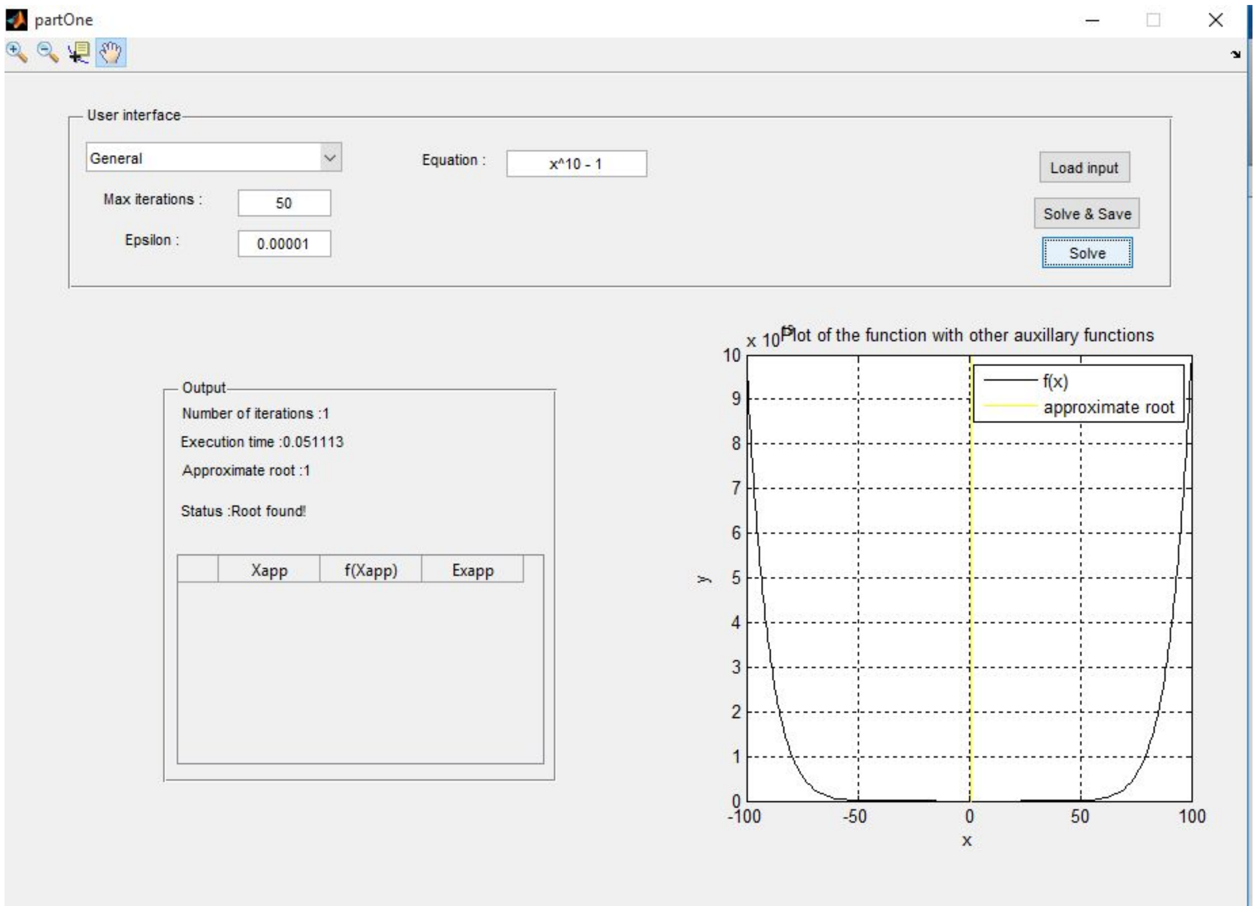
7. Using Secant to solve: X^2-2X+1

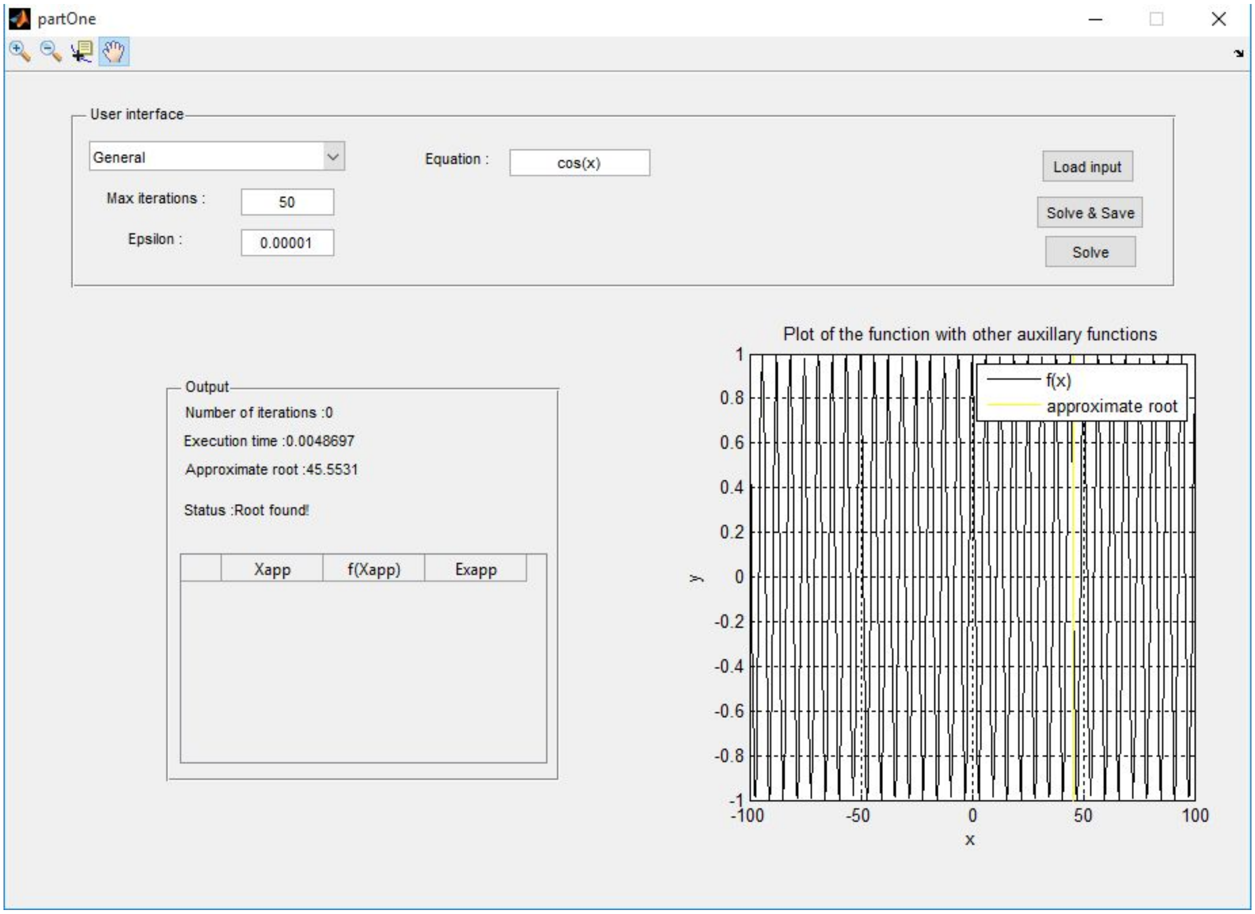


8. General Method was able to get solutions of equations that fall in other methods



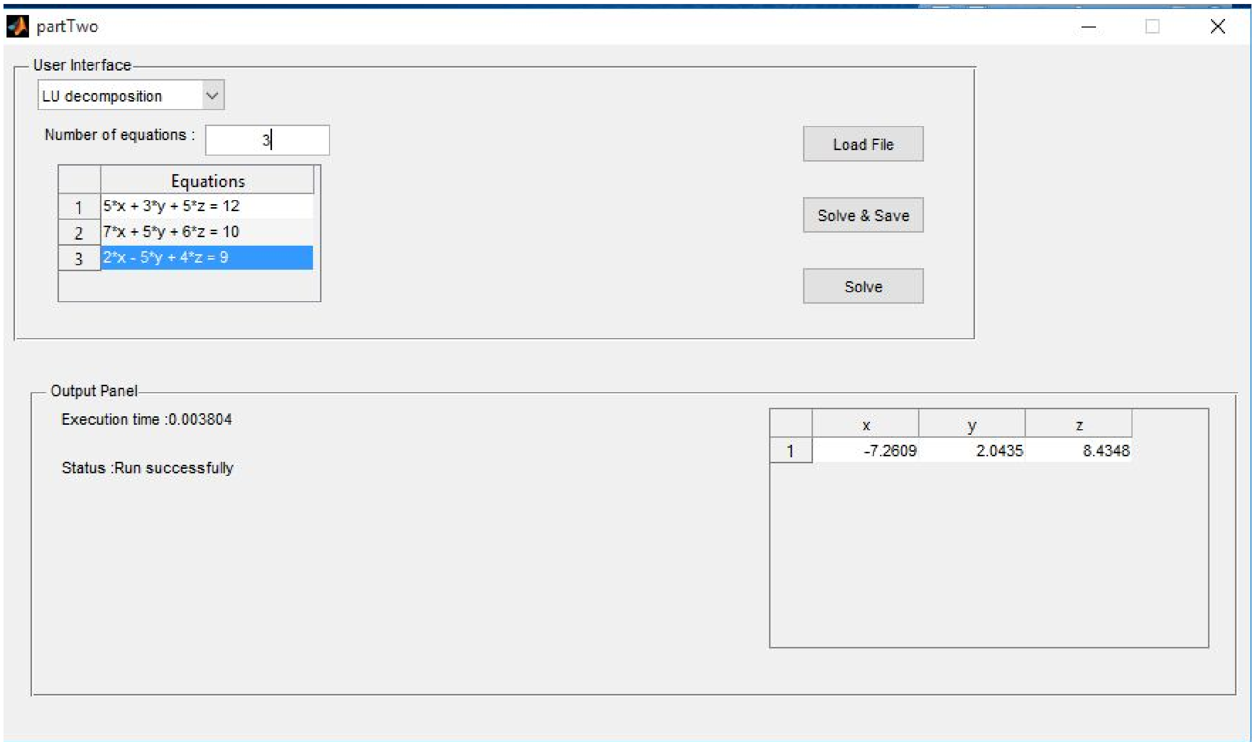
partOne
User interface
General
Equation :
x^2
Load input
Max iterations :
50
Solve & Save
Epsilon :
0.00001
Solve
Output
Number of iterations :0
Execution time :0.00115
Approximate root :0
Status :Root found!
Xapp
f(Xapp)
Exapp
Plot of the function with other auxillary functions
f(x)
approximate root
y
x


partOne
User interface
General
Equation :
x^10 - 1
Load input
Max iterations :
50
Solve & Save
Epsilon :
0.00001
Solve
Output
Number of iterations :1
Execution time :0.051113
Approximate root :1
Status :Root found!
Xapp
f(Xapp)
Exapp
Plot of the function with other auxillary functions
f(x)
approximate root
x
y

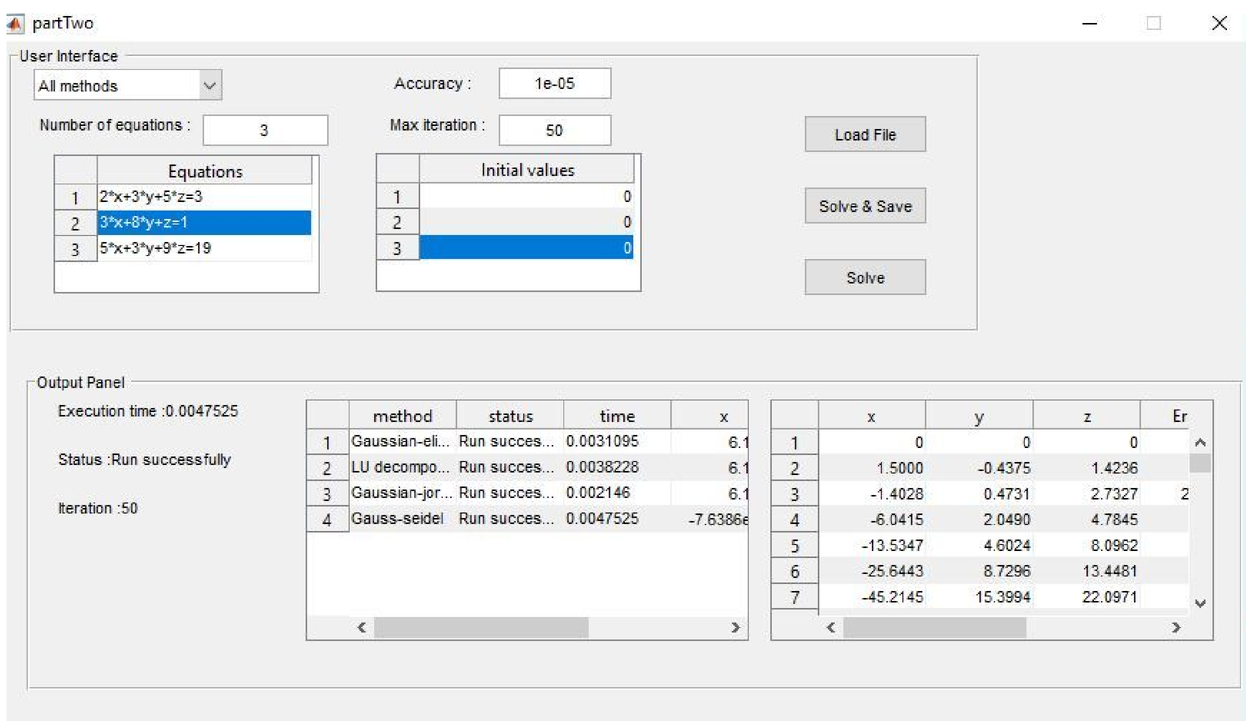


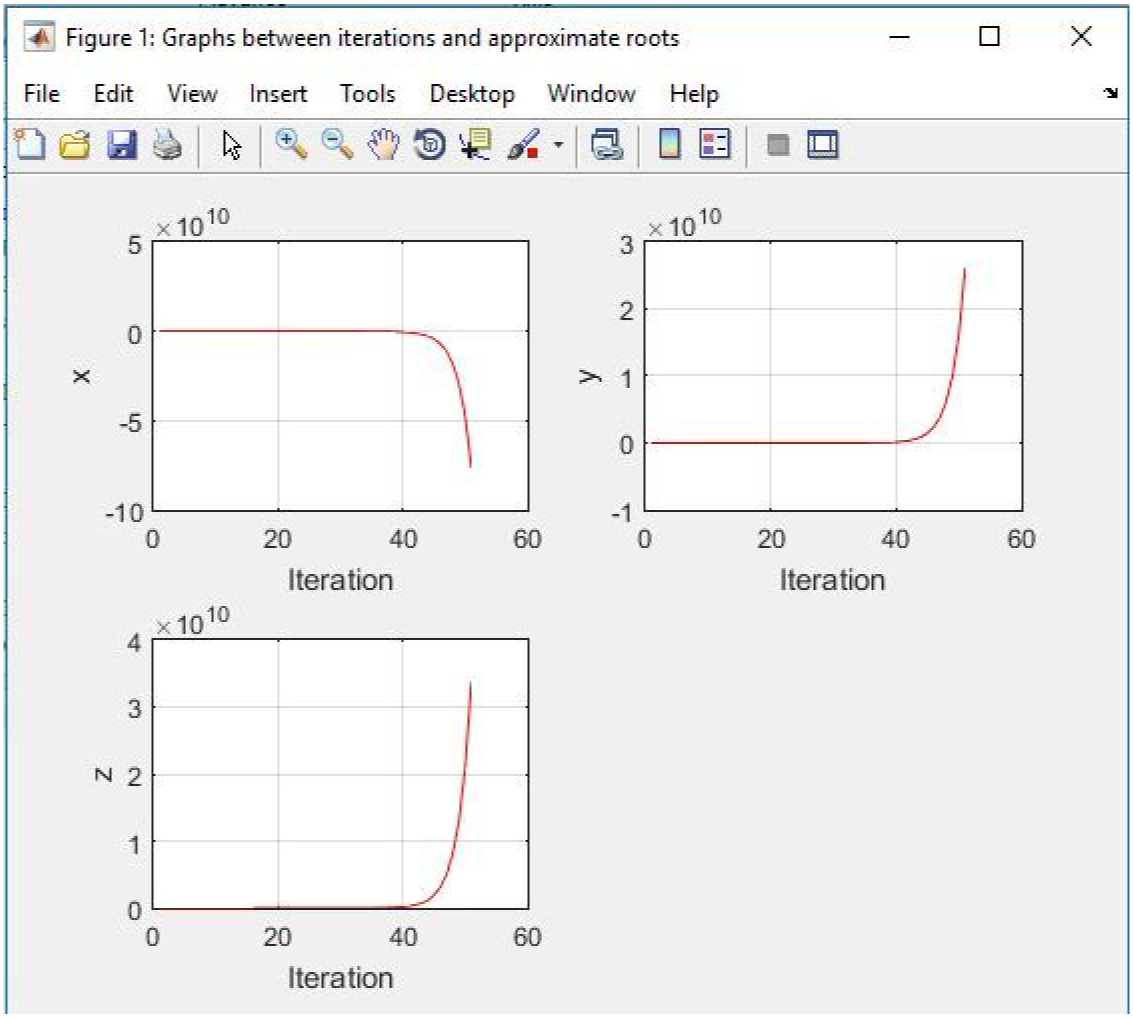
Part Two:

Using LU Decomposition to solve a system of 3 equations



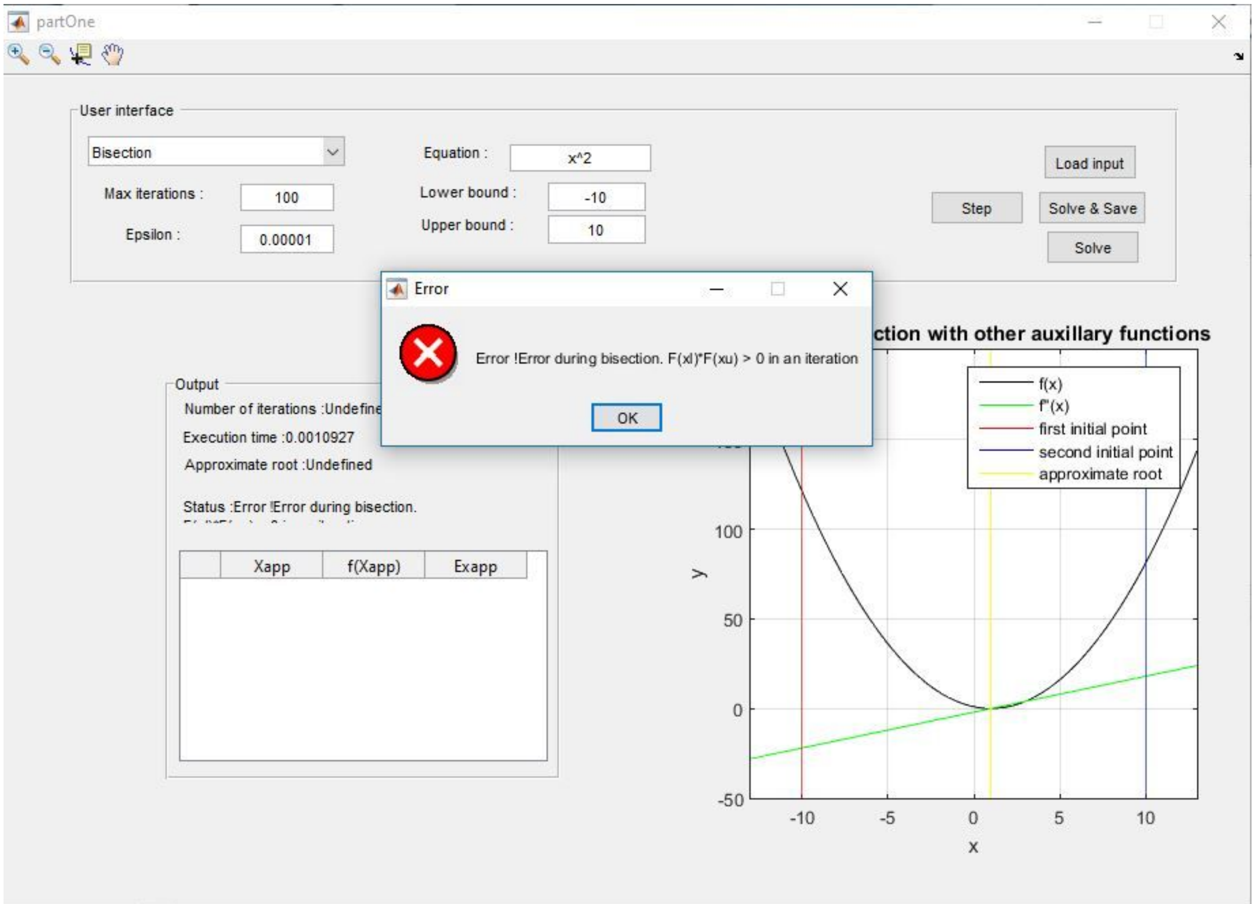
Using All Methods to solve a system of 3 equations



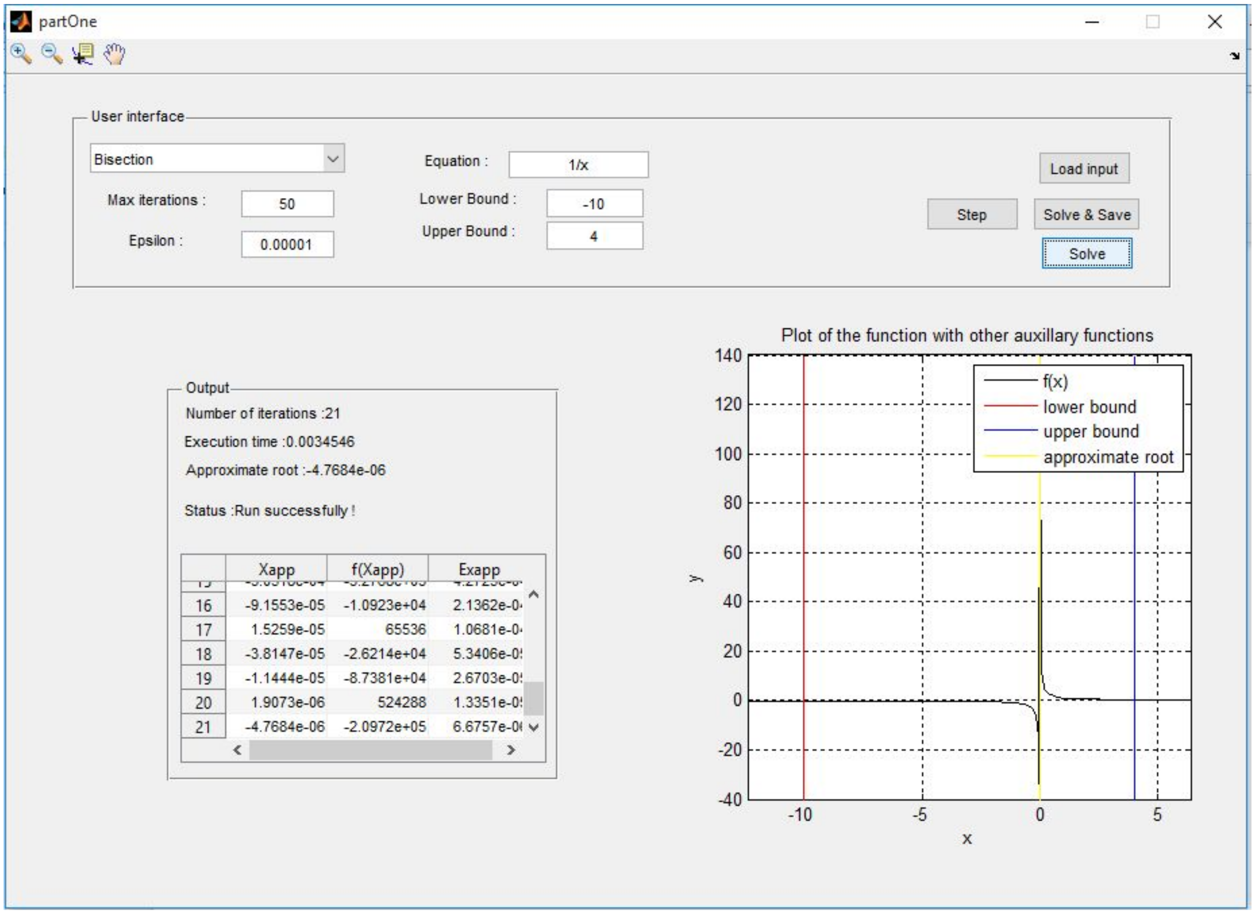


Pitfalls:

1. Bisection:
    1. It cannot detect even number of roots within its range. Eg (X^2 when lower = -10 and upper = +10)

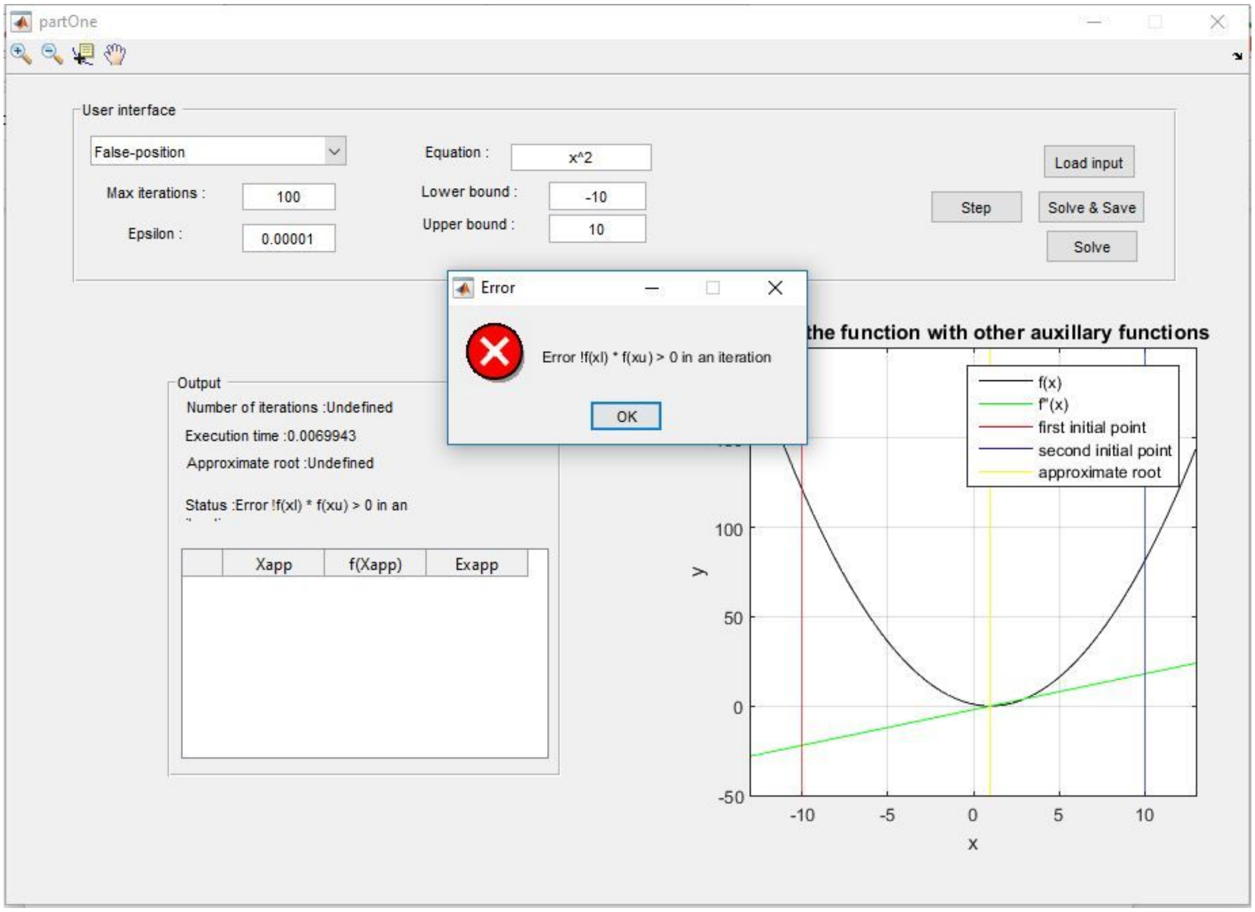


2. Function changes sign but root does not exist

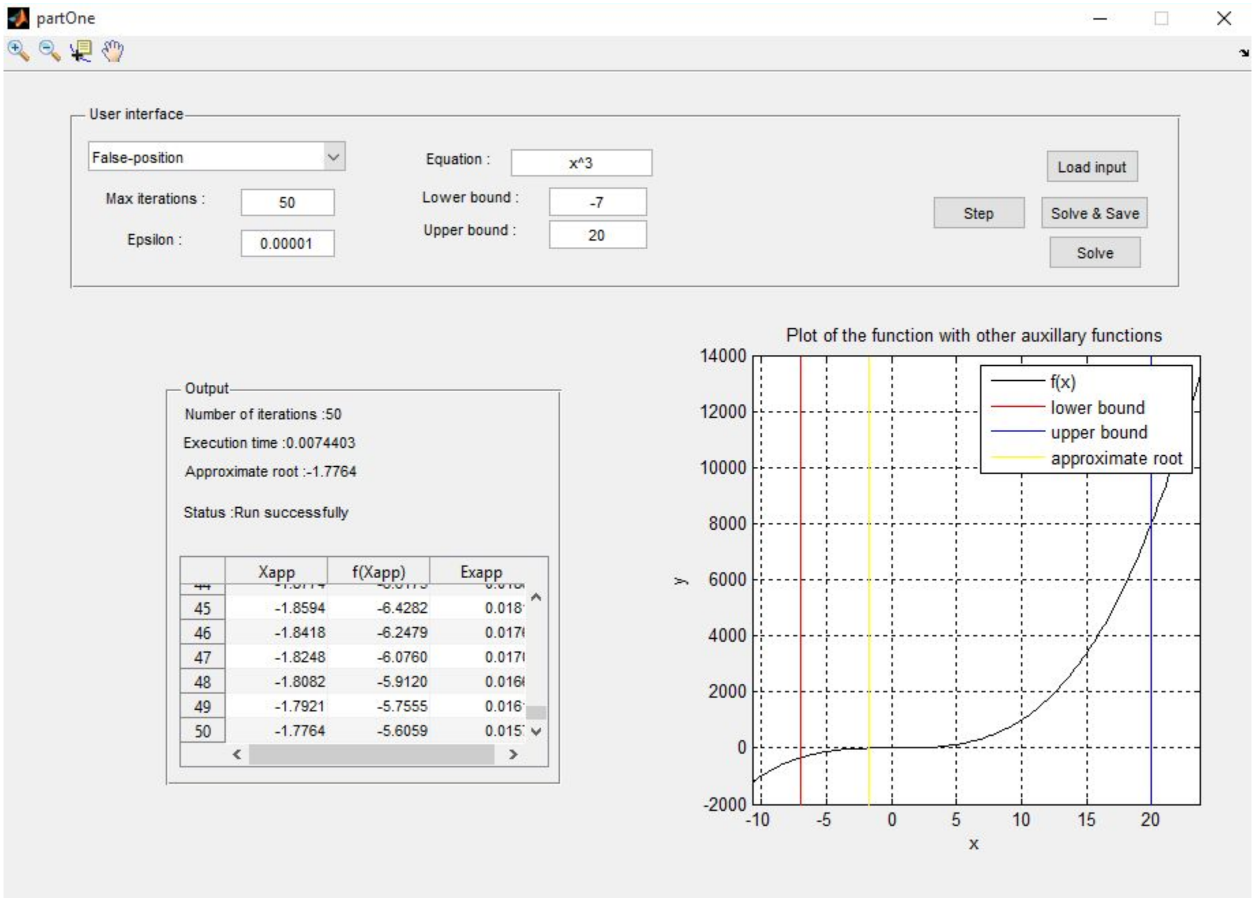


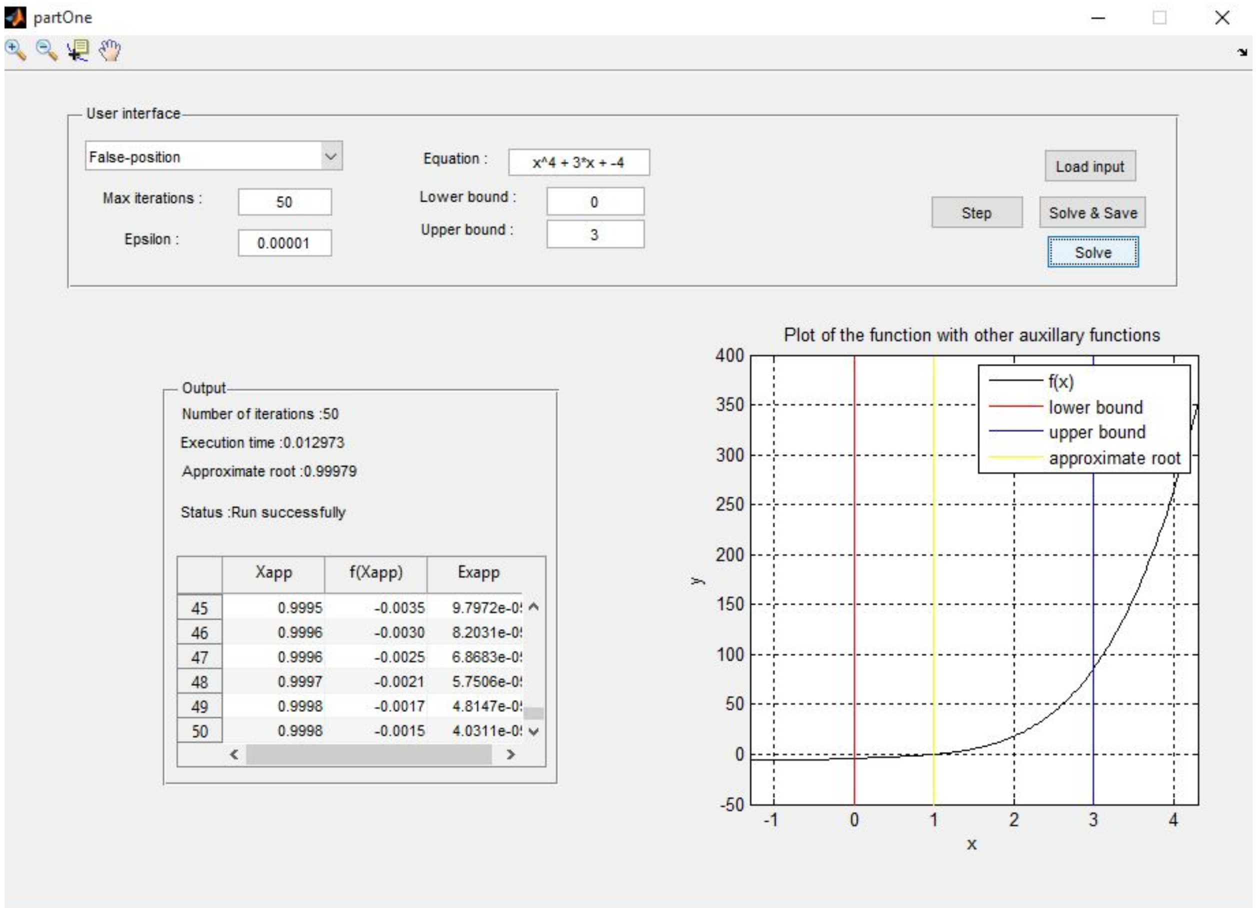
2. Birge Vieta
    1. The problem with this method is that it is able to solve plynomials only
3. False Position

1. Like bisection method it cannot detect even number of roots within its range. Eg (X^2 when lower = -10 and upper = +10)

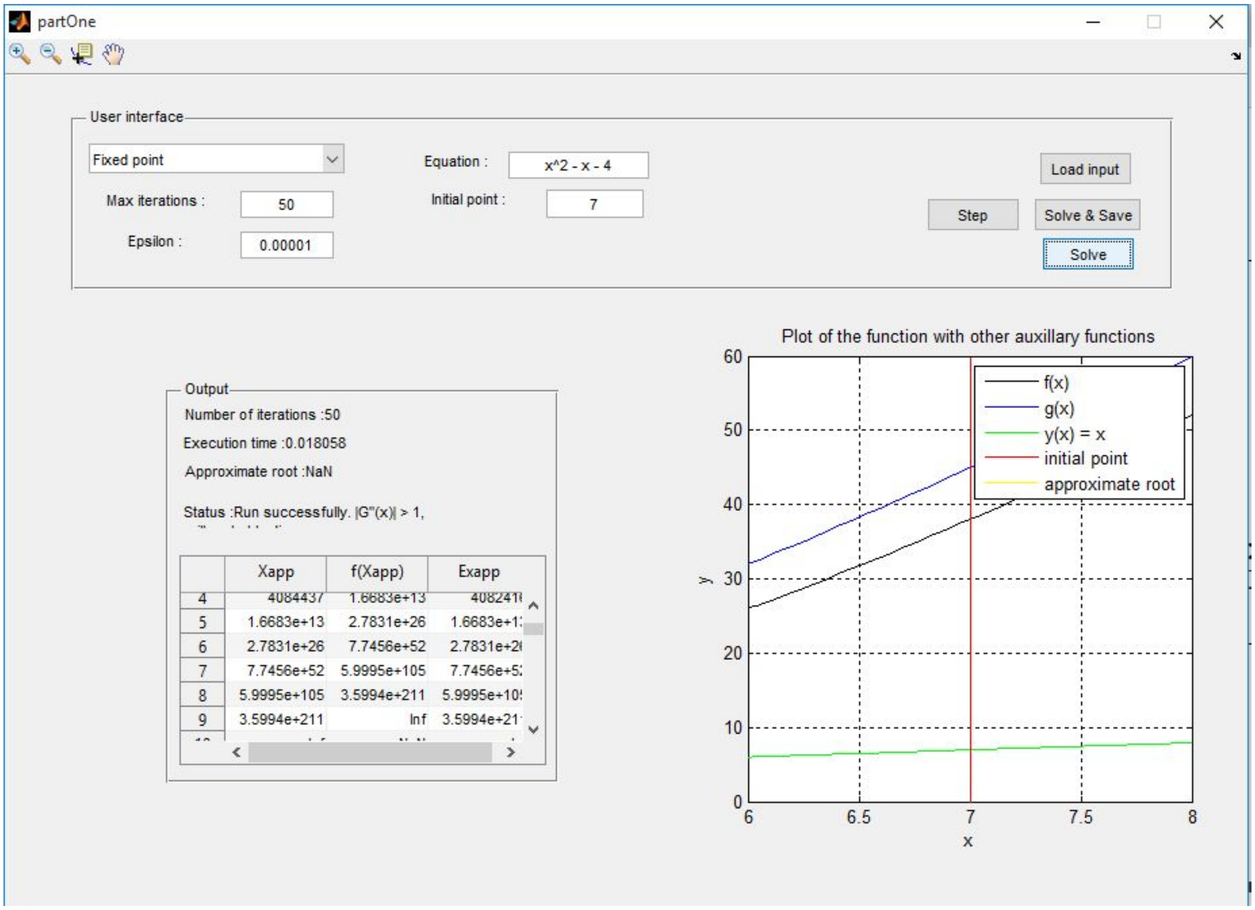


2. Sometimes it is slower than bisection this depends on the function itself (i.e. sometimes we need to perform bisection before false position to get root faster) Example : Here False position took 50 iteration to get the root which is greater than bisection because of the function itself.


partOne
User interface
False-position
Max iterations :
50
Epsilon :
0.00001
Equation :
x^3
Lower bound :
-7
Upper bound :
20
Step
Load input
Solve & Save
Solve
Output
Number of iterations :50
Execution time :0.0074403
Approximate root :-1.7764
Status :Run successfully
Xapp
f(Xapp)
Exapp
45 -1.8594 -6.4282
46 -1.8418 -6.2479
47 -1.8248 -6.0760
48 -1.8082 -5.9120
49 -1.7921 -5.7555
50 -1.7764 -5.6059
Plot of the function with other auxillary functions
f(x)
lower bound
upper bound
approximate root
y
x

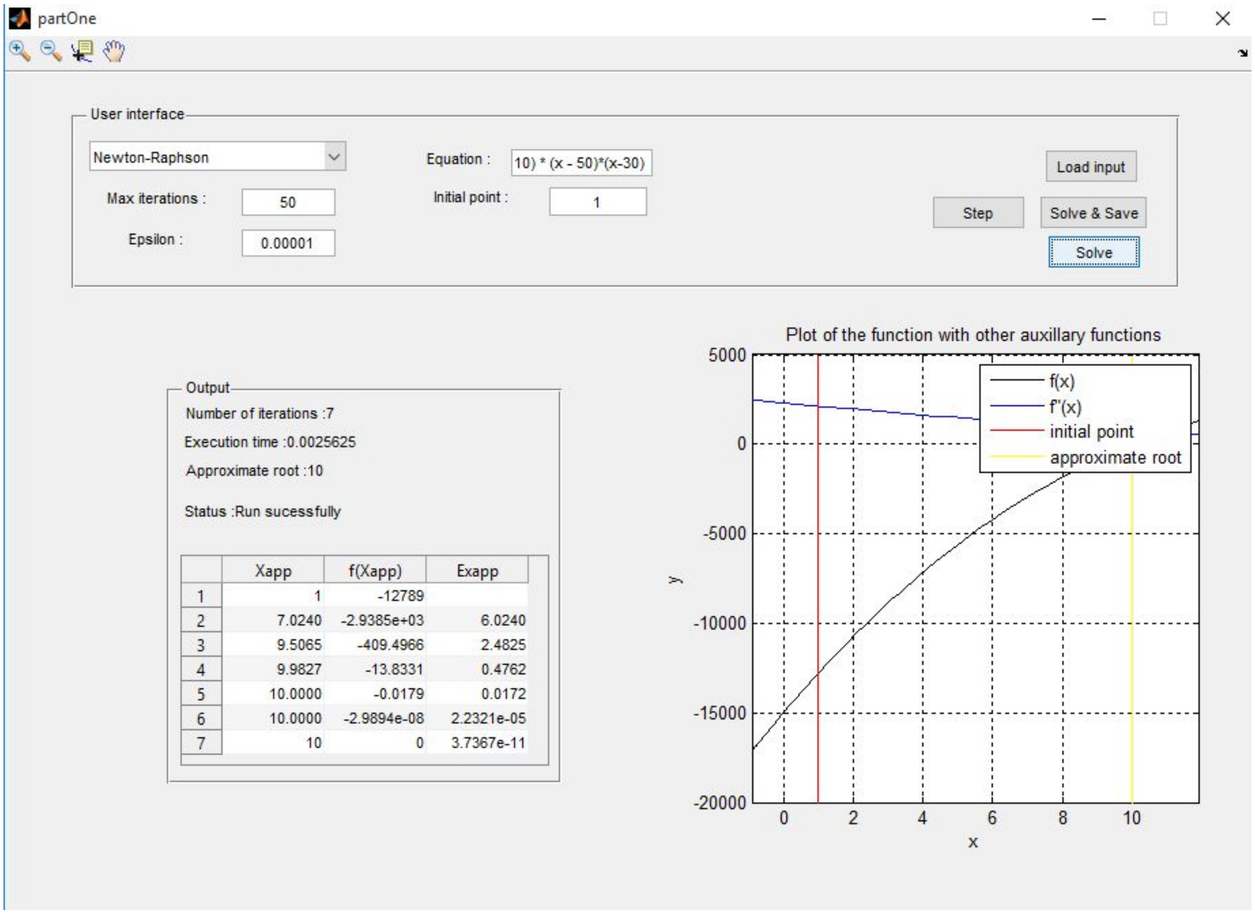


4. Fixed Point
    1. Fixed point not always converges and not always get the approximate root in the same number of steps. This depends on the choice of G(X) as well as the initial point. If abs(G'(x)) is less than 1 then it converges else it diverges.



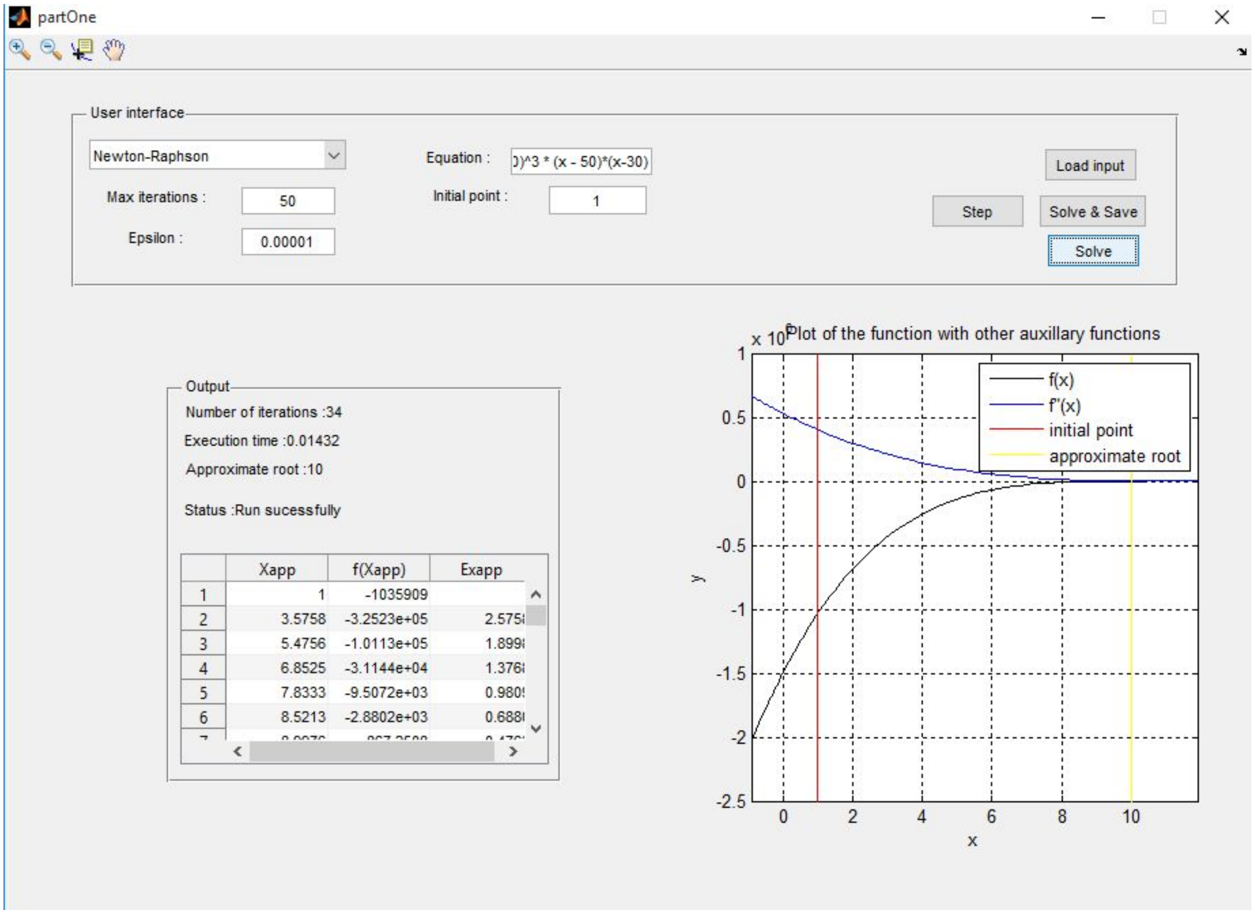
5. Newton Raphson
    1. When there is multiplicity it converges linearly not quadratically.

Solving (x - 10) * (x - 50)*(x-30)

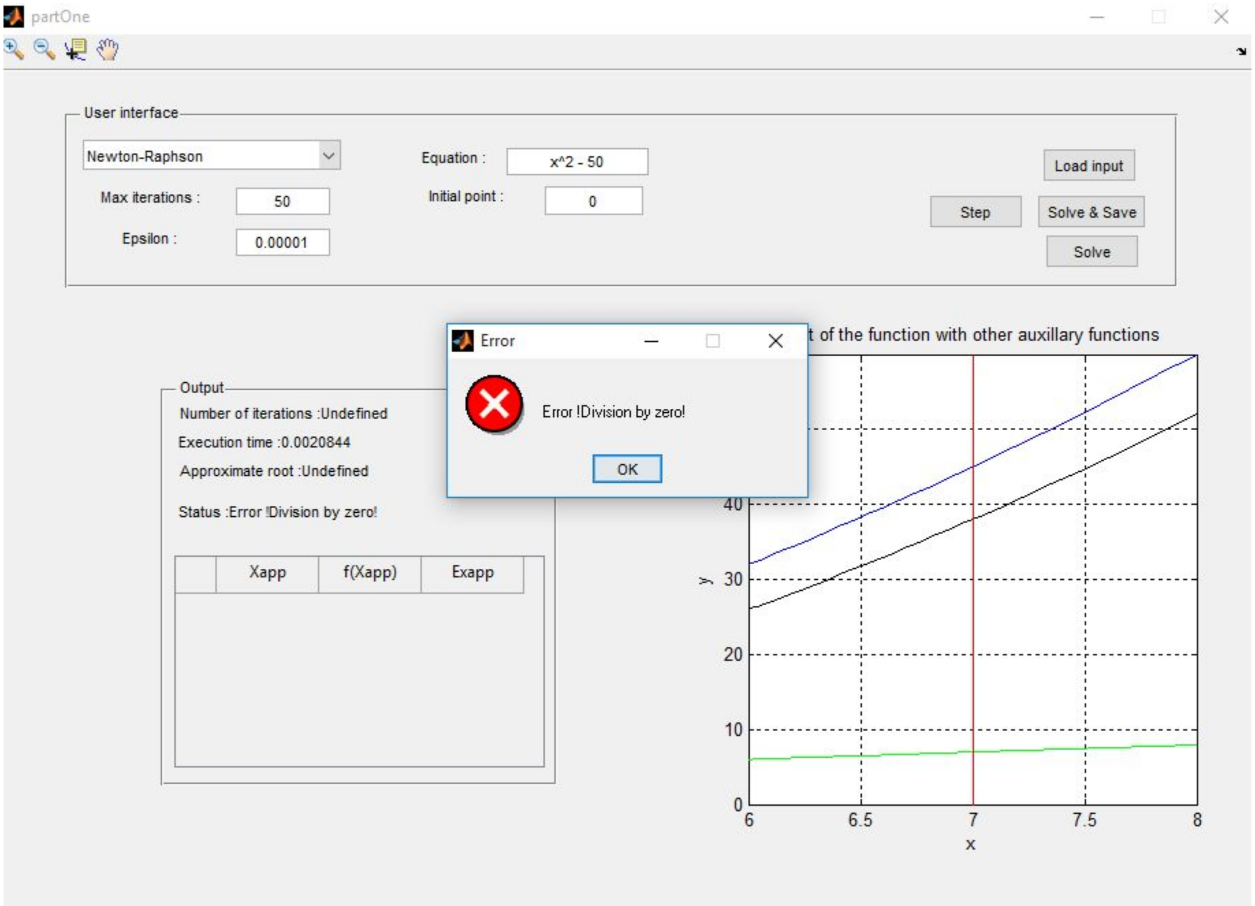


	Xapp	f(Xapp)	Exapp
1	1	-12789	
2	7.0240	-2.9385e+03	6.0240
3	9.5065	-409.4966	2.4825
4	9.9827	-13.8331	0.4762
5	10.0000	-0.0179	0.0172
6	10.0000	-2.9894e-08	2.2321e-05
7	10	0	3.7367e-11

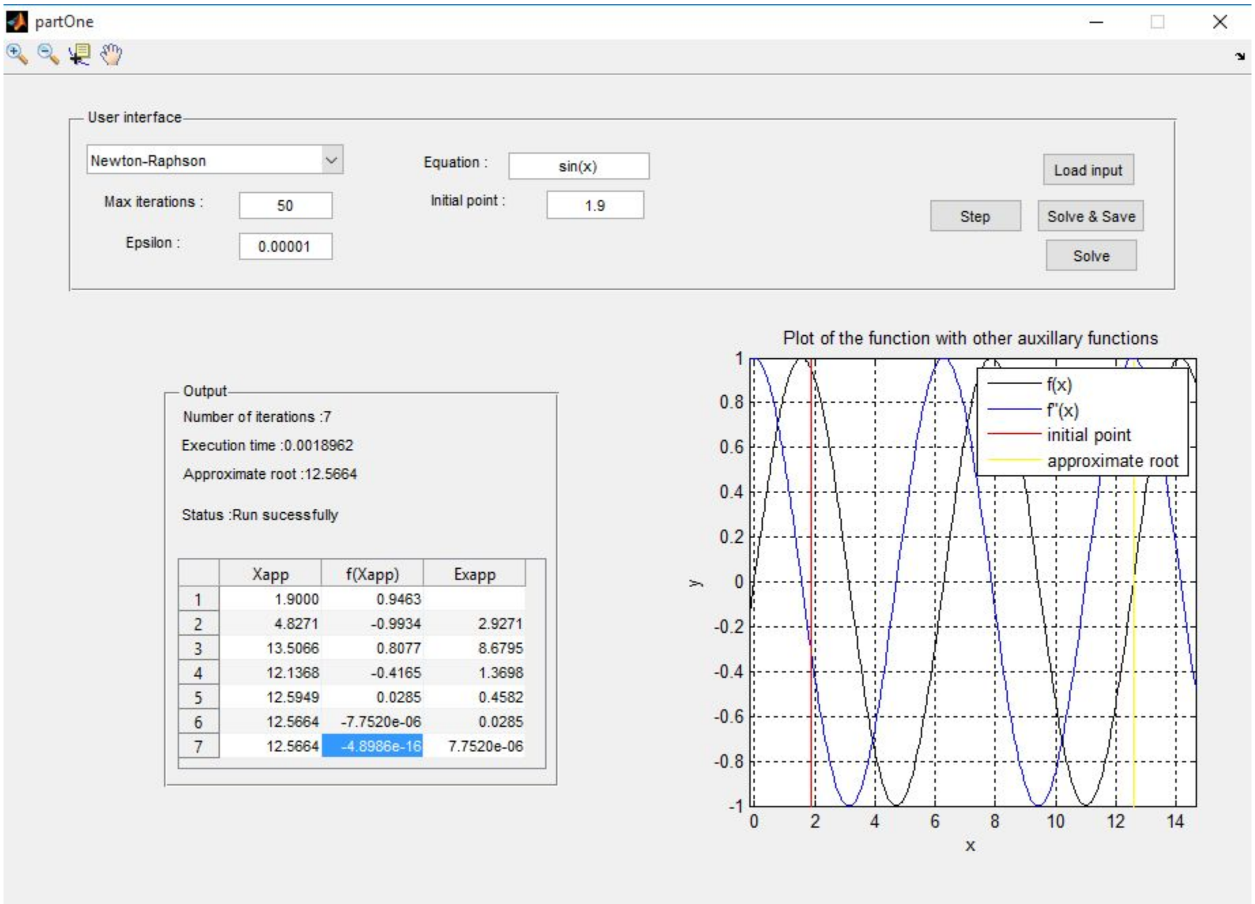
Solving (x - 10)^3 * (x - 50)*(x-30)



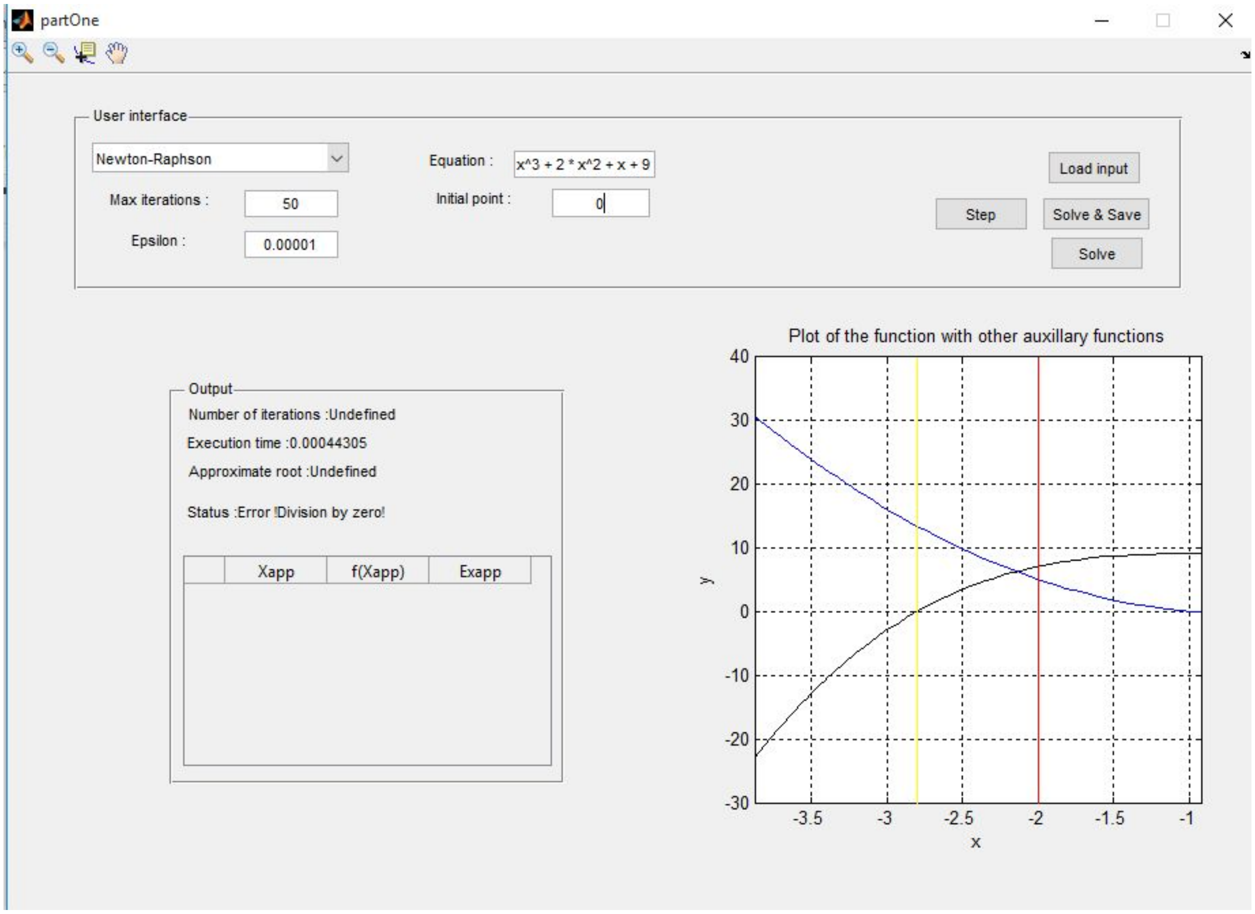
2. In the formula we divide by f'(x) so if it is zero we will get infinity as in inflection point.



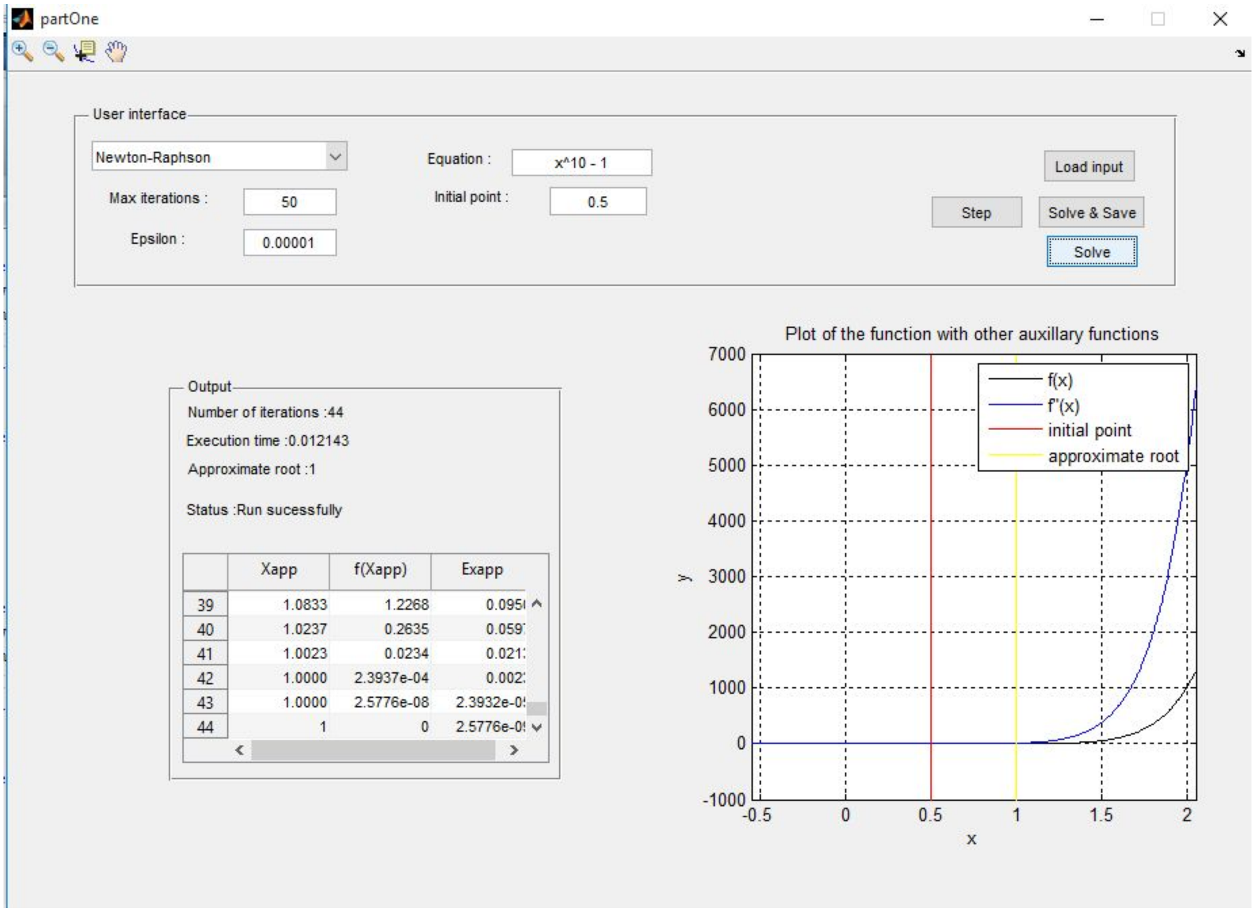
3. We may choose an initial point near a specific root but the method converges to another root.



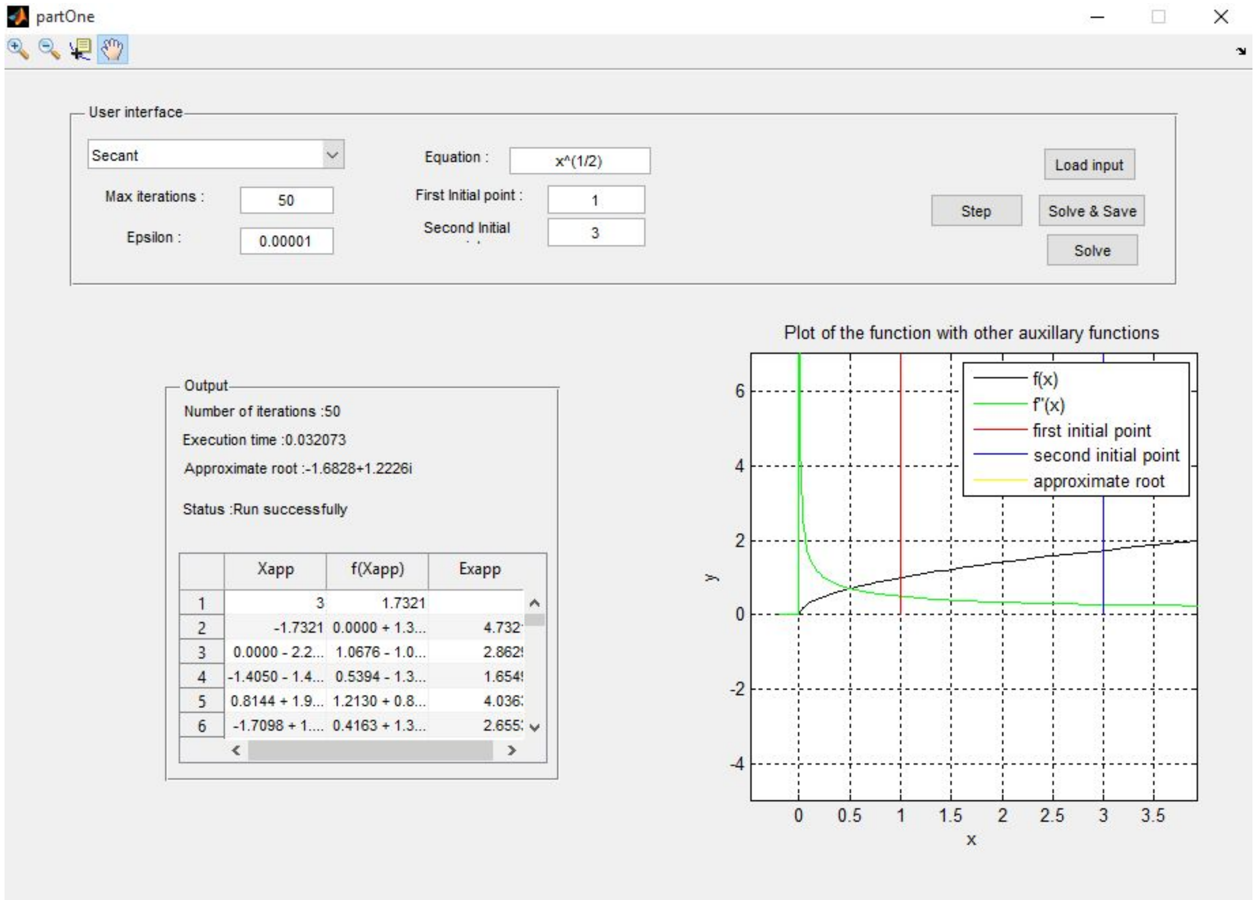
4. Points near local minimum or maximum may cause oscillation then diverges.



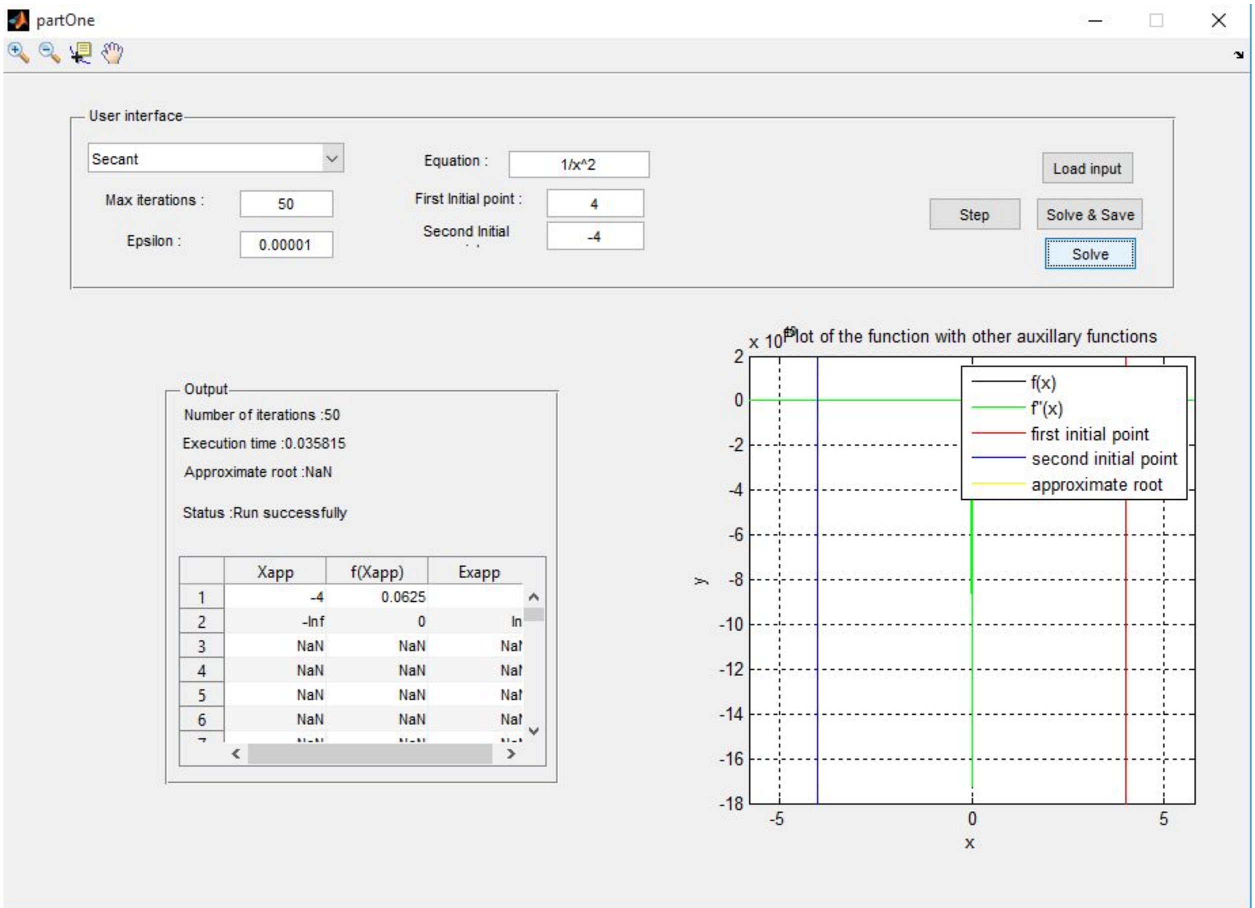
5. Sometimes it’s slow according to the function.



6. Secant
    1. With multiple roots it converges linearly as Newton Raphson not quadratically.
    2. It may diverge according to the function and intial points



3. It depends on 2 initial points if f(xi) = f(xi+1) then it won’t be able to get a new point from interpolation.


partOne
User interface
Secant
Max iterations :
50
Epsilon :
0.00001
Equation :
1/x^2
First Initial point :
4
Second Initial
-4
Load input
Step
Solve & Save
Solve
Output
Number of iterations :50
Execution time :0.035815
Approximate root :NaN
Status :Run successfully
Xapp
f(Xapp)
Exapp
1 -4 0.0625
2 -Inf 0 In
3 NaN NaN NaI
4 NaN NaN NaI
5 NaN NaN NaI
6 NaN NaN NaI
Plot of the function with other auxillary functions
f(x)
f'(x)
first initial point
second initial point
approximate root
y
x