NUMERICAL ANALYSIS

FINAL PROJECT 1

Lara HossamElDin Mostafa	6853
Nourine Mohamed AbouElWafa	7152
Nour Hesham Shaheen	7150
Sohayla Khaled Abouzeid	6851

Table of Contents

Project Description

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}**.

This program is a **root finder** application, that takes the numerical method desired to solve a specific input equation and required initial points with an interactive **GUI** and a support for all functions. The **output is** the number of iterations, execution time, all iterations, approximate root and precision aligned with the theoretical bound of the error.

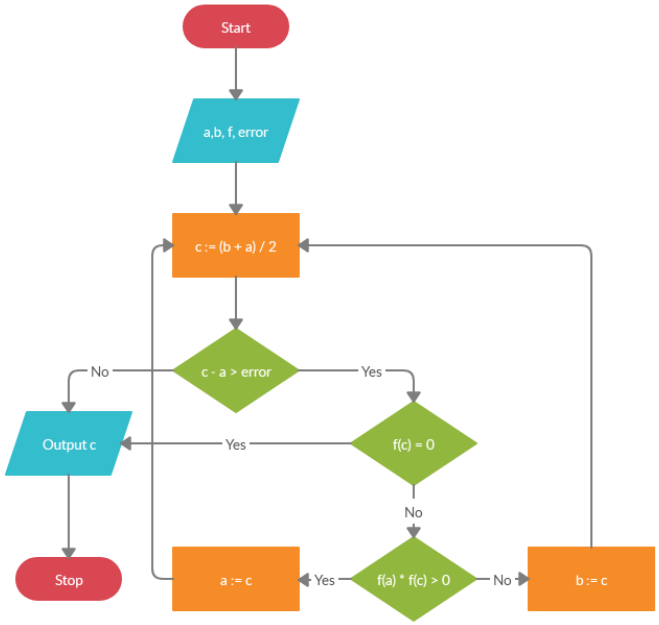
Numerical Methods

Bisection

The bisection method is a closed root-finding method that applies when one knows two values with opposite signs for a continuous function.

Input Required from User: Xupper, Xlower, Number of iterations [Default 50], Error Tolerance [Default 0.00001].

Flowchart



Pseudo Code

```
1. Start
2. Define function f(x)
3. Input
   a. Lower and Upper guesses x0 and x1
   b. tolerable error
   c. Number of iterations
4. If f(x0)*f(x1) > 0 "Initial guesses are wrong." goto 3
   End If
5. Do
   x2 = (x0+x1)/2
   If f(x0)*f(x2) < 0
      x1 = x2
   Else
      x0 = x2
   End If
while abs(f(x2) > e  and i!=max_iter
6. Print root as x2 7. Stop
```

Matlab Code

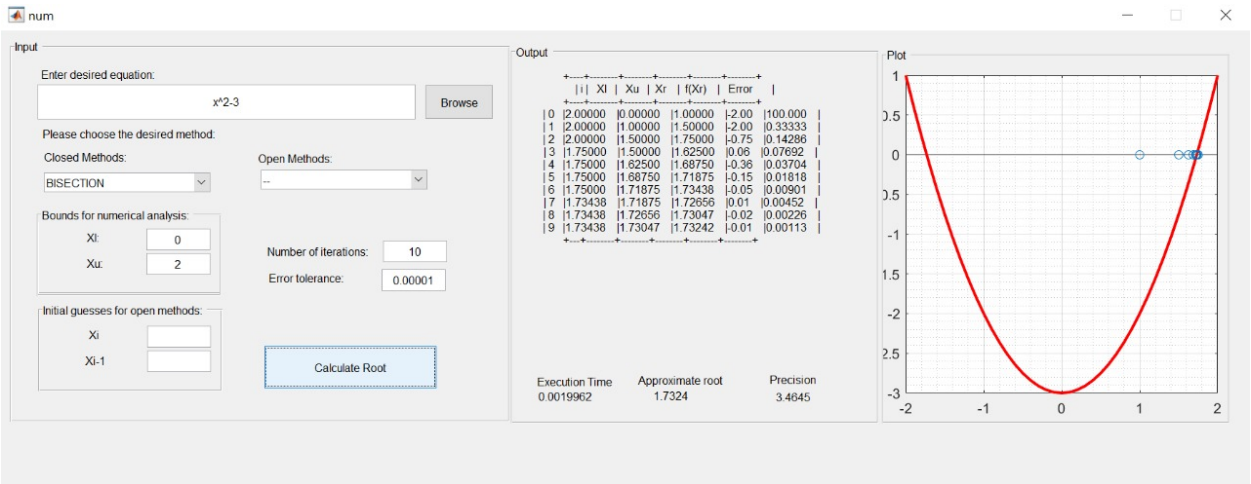
```
function bisection(handles,max,f,es,xu,xl)
% Case bisection:
fxl = f(xl);
fxu = f(xu);
if fxl * fxu < 0
    fprintf('Initial guesses are right. \n');
else
    fprintf('Initial guesses are wrong. \n');
end
xr = (xu + xl) / 2;
ea = 100;
i =0;
xrprev=0;
while (ea > es && i ~= max )
    fxr = f(xr);
    fxl = f(xl);
    if fxl * fxr < 0
        xu = xr;
    elseif fxl * fxr > 0
```

```
        xl = xr;
    end
    xrprev = xr;
    xr = (xu + xl) / 2;
    ea = abs((xrprev - xr) / xr);
    i = i+1;
end
```

Problematic Cases

- When fxu and fxl have the same sign. Guesses are incorrect. User is prompted to try again.

Sample run

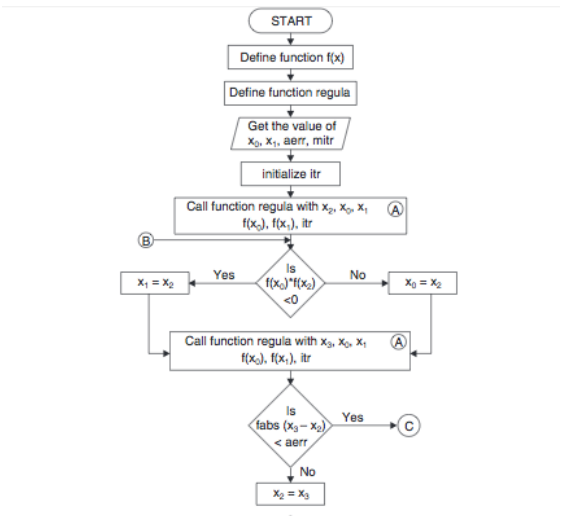


False-Position

The false position method is a closed root-finding method to solve equations with one unknown.

Input Required from User: Xupper, Xlower, Number of iterations [Default 50], Error Tolerance [Default 0.00001].

Flowchart



Pseudo Code

```
1. Start
2. Define function f(x)
3. Input
a. Lower and Upper guesses x0 and x1
b. tolerable error e
c. maximum number of iterations
4. If f(x0)*f(x1) > 0 print "Incorrect initial guesses"
      goto 3
   End If
5. Do
      x2 = x0 - ((x0-x1) * f(x0))/(f(x0) - f(x1))
      If f(x0)*f(x2) < 0
         x1 = x2
      Else
         x0 = x2
      End If
    While abs(f(x2) > e  and i != maxiterations
6. Print root as x2
7. Stop
```

Matlab Code

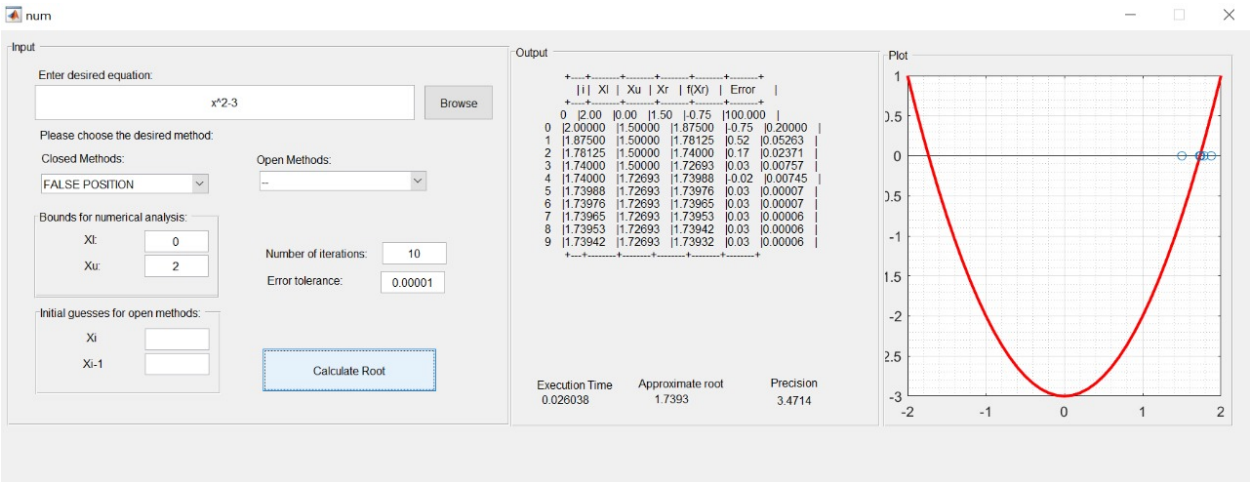
```
function regula_falsi(handles,max,f,es,xu,xl)
% Case regula falsi:
fxl = f(xl);
fxu = f(xu);
if fxl * fxu < 0
    fprintf('Initial guesses are right. \n');
else
    fprintf('Initial guesses are wrong. \n');

end
 xr = xu - (fxu * (xl - xu) / (fxl - fxu));
ea = 100;
i =0;
xrprev=0;
while (ea > es && i ~= max )
    fxr = f(xr);
```

```
    fxl = f(xl);
    if fxl * fxr < 0
        xu = xr;
    elseif fxl * fxr > 0
        xl = xr;
    end
    xrprev = xr;
    xr = xu - (fxu * (xl - xu) / (fxl - fxu));
    ea = abs((xrprev - xr) / xr);
    i = i+1;
end
```

Problematic Cases

- When fxu and fxl have the same sign. Guesses are incorrect. User is prompted to try again.

Sample run



Fixed-Point

Fixed point iteration method is open root finding method for non-linear equation by successive approximation. It requires only one initial guess to start. Since it is open method its convergence is not guaranteed.

Input Required from User: Xi, Number of iterations [Default 50], Error Tolerance [Default 0.00001].

Pseudo Code

```
1. Start
```

```
2. Define function as f(x)
3. Define convergent form g(x)
4. Input:
a. Initial guess x0
b. Tolerable Error e
c. Maximum Iteration N
5. Initialize iteration counter: step = 1
6. Do
  x1 = g(x0)
  step = step + 1
  If step > N
      Print "Not Convergent"
      Stop
  End If x0 = x1
  While abs f(x1) > e
7. Print root as x1 8. Stop
```

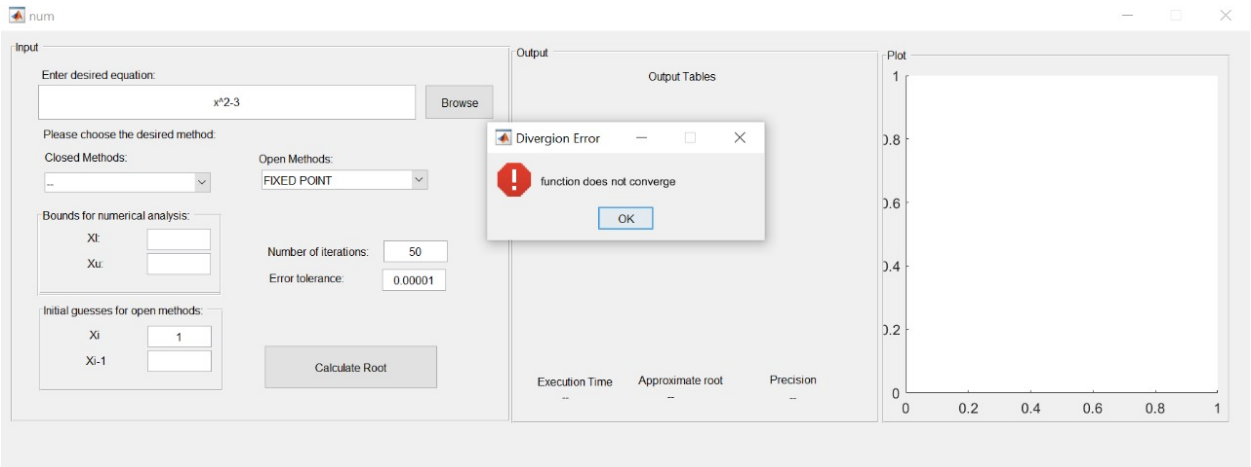
Matlab Code

```
function fixed_point(handles,max,funct_fixed,f,es,xold)
syms y;
syms x;
flag=0;
i=0;
ea=100;
for k=1:length(funct_fixed)
   tf=strcmp( funct_fixed(k),"x");
    if tf==1
           newChr = replaceBetween(funct_fixed,k,k,'y');
           m=str2sym(newChr);
           temp=string(isolate(m==0, y))
           s=str2sym(erase(temp,"y == "));
           d = diff(s,x);
           if d==0
               break;
           end
           derv = matlabFunction(d);
           if abs(derv(xold))<1
                flag=1;
               break;
           end
```

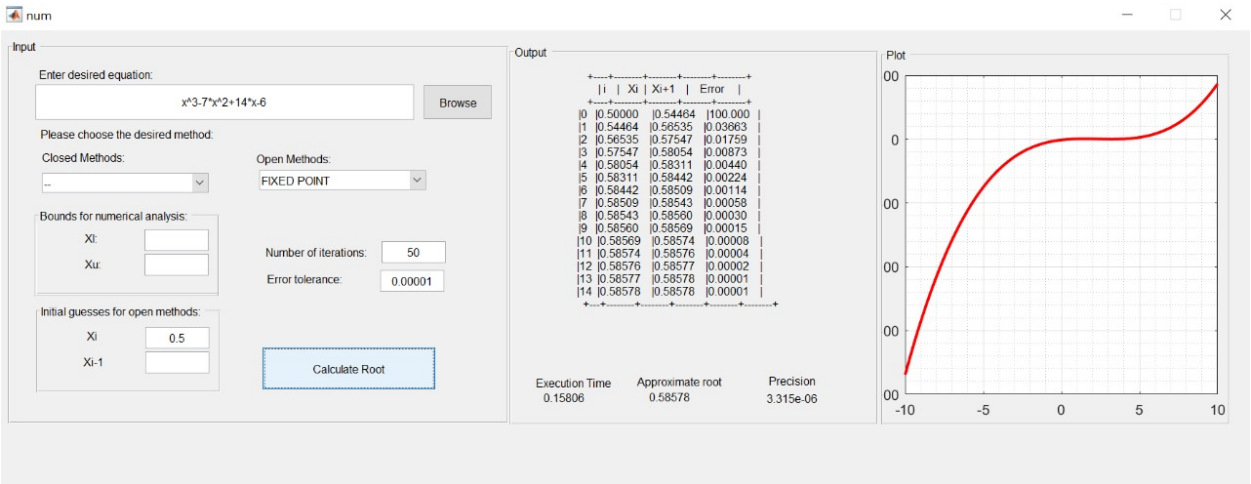
```
    end
end
if flag ==1
    sym x
   gx=matlabFunction(s);
   xnew=gx(xold);
   xprev=0;
    i=1;
   while (ea > es && i ~= max )
       xold=xnew;
       xprev=xnew;
       xnew=gx(xold);
       ea = abs((xprev - xnew)/xnew);
       i=i+1;
   end
```

Problematic Cases

- When g(x) diverges.



Sample run



Newton-Raphson

Newton Raphson Method is an open method and starts with one initial guess for finding real root of non-linear equations.

Input Required from User: Xi, Number of iterations [Default 50], Error Tolerance [Default 0.00001].

Pseudo Code

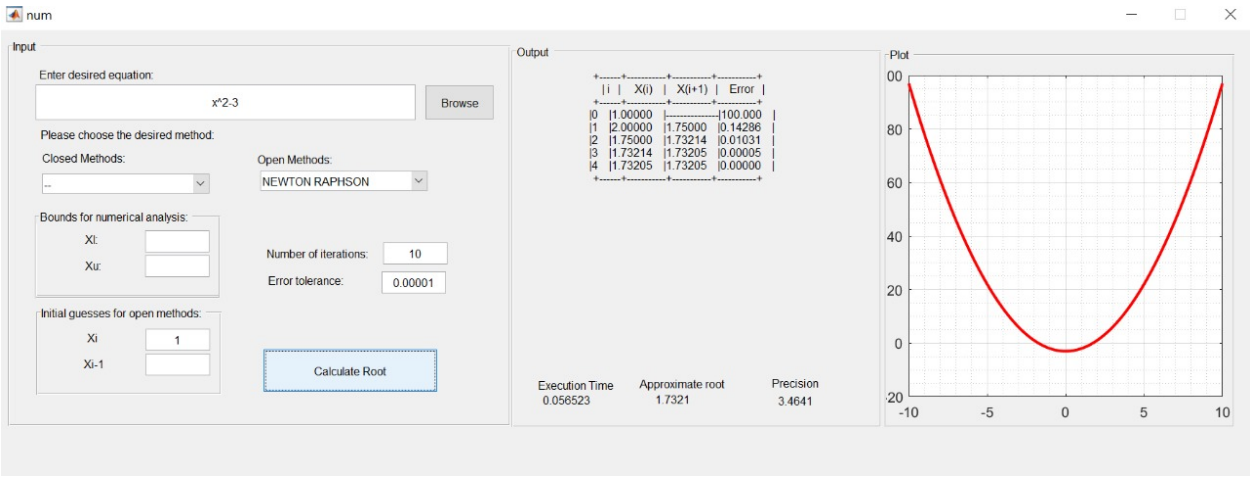
```
1. Start
2. Define function as f(x)
3. Define derivative of function as g(x)
4. Input:
a. Initial guess x0
b. Tolerable Error e
c. Maximum Iteration N
5. Initialize iteration counter step = 1
6. Do
    If g(x0) = 0
        Print "Mathematical Error"
        Stop
    End If
    x1 = x0 - f(x0) / g(x0)
    x0 = x1
    step = step + 1
    If step > N Print
```

```
        "Not Convergent"
         Stop
     End If
  While abs f(x1) > e
7. Print root as x1 8. Stop
```

Matlab Code

```
function newton(handles,max,f,es,xold)
now1 = tic();
% Case newton:
syms x;
d = diff(f,x);
derv = matlabFunction(d);
%x(1) = xold;
xnew = xold - (f(xold)/derv(xold));
i =1;
ea = 100;
xprev=0;
while(ea > es && i ~= max)
    xold=xnew;
    xprev=xnew;
    xnew =  xold - (f(xold)/derv(xold));
    ea = abs((xprev - xnew)/xnew);
    i=i+1;
end
```

Sample run



Secant

Secant Method is an open method and starts with two initial guesses for finding real root of non-linear equations.

Input Required from User: Xi, Xi-1, Number of iterations [Default 50], Error Tolerance [Default 0.00001].

Flowchart

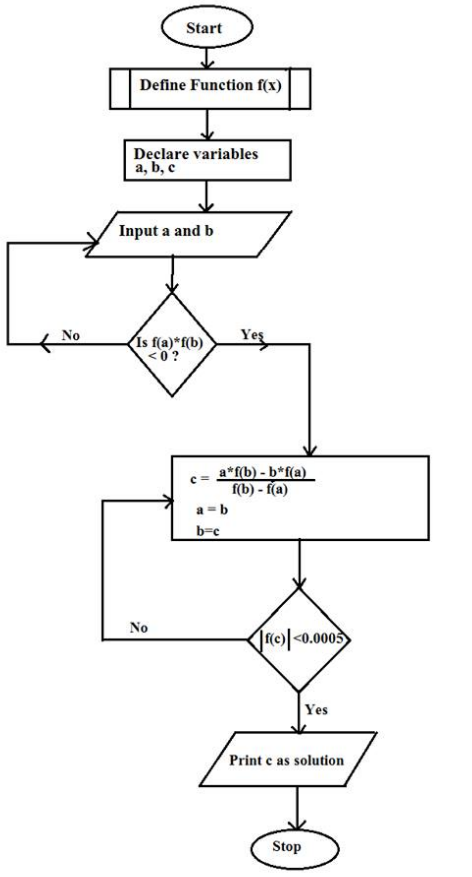


Fig. Flow Chart for Secant Method

Pseudo Code

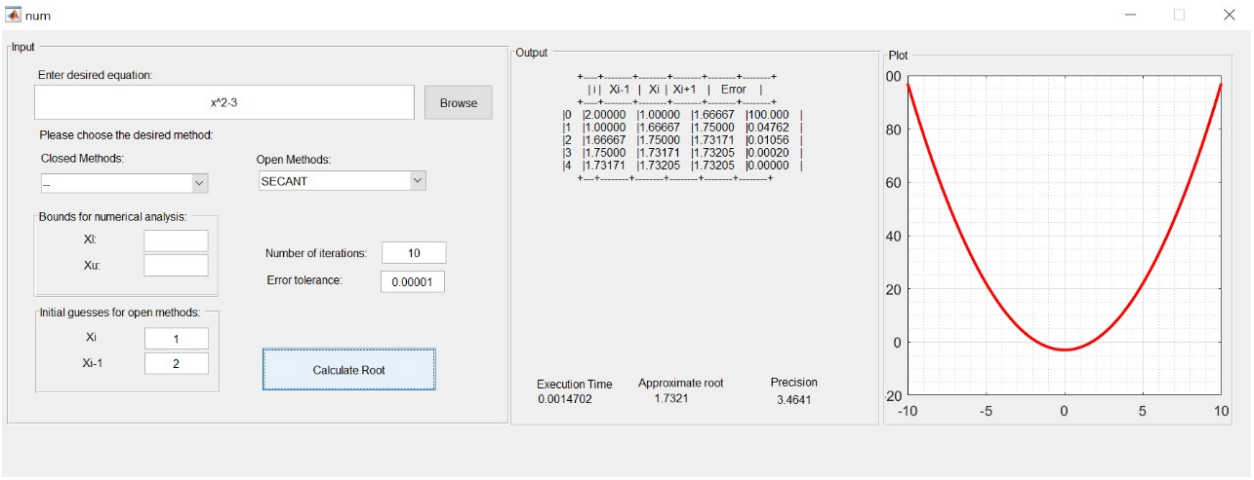
```
1. Start
2. Define function as f(x)
```

```
3. Input:
a. Initial guess x0, x1
b. Tolerable Error e
c. Maximum Iteration N
4. Initialize iteration counter step = 1
5. Do
      x2 = x1 - (x1 - x0) * f(x1) / ( f(x1) - f(x0) )
      x0 = x1
      x1 = x2
      step = step + 1
      If step > N
         Print "Not Convergent"
         Stop
       End If
       While abs f(x2) > e
6. Print root as x2
7. Stop
```

Matlab Code

```
function secant(handles,max,f,es,xold,xold2)
% Case secant:
xnew=xold-((f(xold)*(xold2-xold))/(f(xold2)-f(xold)));
i =0;
ea=100;
xrprev=0;
i=1;
while (ea > es && i ~= max )
   xrprev = xnew;
   xold2=xold
   xold=xnew
   xnew=xold-((f(xold)*(xold2-xold))/(f(xold2)-f(xold)));
   ea = abs((xrprev - xnew) / xnew);
   i = i+1;
end
```

Sample run



Behavior Analysis

	Bisection	False Position	Fixed Point	Newton–Raphson	Secant
Execution Time	0.0019962 s	0.026038 s	-	0.056523 s	0.0014702
Number of iterations	10	10	-	5	5

BONUS: Single Step Mode Simulation for Bisection Method