ENGR 313

Dr. Sherine Elbaradei

Group Project - Part 1

By:

Ahmed A. Agiza (900121143)

Mohammed R. Anany (900120267)

Ola Amr Hussein (900120585)

Rawan Abbas (900120743)

Pseudo-code:

#1

<u>Bisection Method:</u>

```
function bisection(f(x)):
    xl = read lower bound guess
    xh = read higher bound guess
    Validate_the_existence_of_root()
    x0 = (xl + xh) / 2.0
    error = ∞

    iterate while(error > ε):
        if f(xl)*f(x0) < 0 :
            xNew = (xl + x0)  / 2.0
        else:
            xNew = (x0 + xh)  / 2.0
        endif

        error = |xNew – x0| / xNew
        x0 = xNew

    endloop

    return xNew

endfunction
```

Secant Method:

```
function secant(f(x)):
    x0 = read lower bound guess
    x1 = read higher bound guess
    x1 = nextGuess(f(x), x0, x1)
    error = ∞

    iterate while(error > ε):
        xNew = nextGuess(f(x), x0, x1)
        error = |xNew – x1| / xNew
        x0 = x1
        x1 = xNew
    endloop

    return xNew

endfunction

function nextGuess(f(x), x0, x1):

    return x1 - f(x1)*((x1 – x0)/(f(x1) – f(x0)))

endfunction
```

__False-Position Method:__

```
function falsePosition(f(x)):

    xl = read lower bound guess
    xh = read higher bound guess
    Validate_the_existence_of_root()
    x0 = nextGuess(f(x), xl, xh)
    error = ∞

    iterate while(error > ε):

        if f(xl) * f(x0) < 0 :
            xNew = nextGuess(f(x), xl, x0)
        else
            xNew = nextGuess(f(x), x0, xh)
        endif

        if f(xl) * f(xNew) < 0 :
            xh = xNew
        else
            xl = xNew
        endif

        error = |xNew – x0| / xNew
        x0 = xNew

    endloop

    return xNew

endfunction

function nextGuess(f(x), x0, x1):

    return x1 - f(x1)*((x1 – x0)/(f(x1) – f(x0)))

endfunction
```

Newton-Raphson Method:

```
function newtonRaphson(f(x), f’(x)):

    x0 = read lower initial guess
    x0 = nextGuess(f(x), f’(x), x0)
    error = ∞

    iterate while(error > ε):
        xNew = nextGuess(f(x), f’(x), x0)
        error = |xNew – x0| / xNew
        x0 = xNew
    endloop

    return xNew

endfunction

function nextGuess(f(x), f’(x), x0):

    return x0 – f(x0)/f’(x0)

endfunction
```

#2

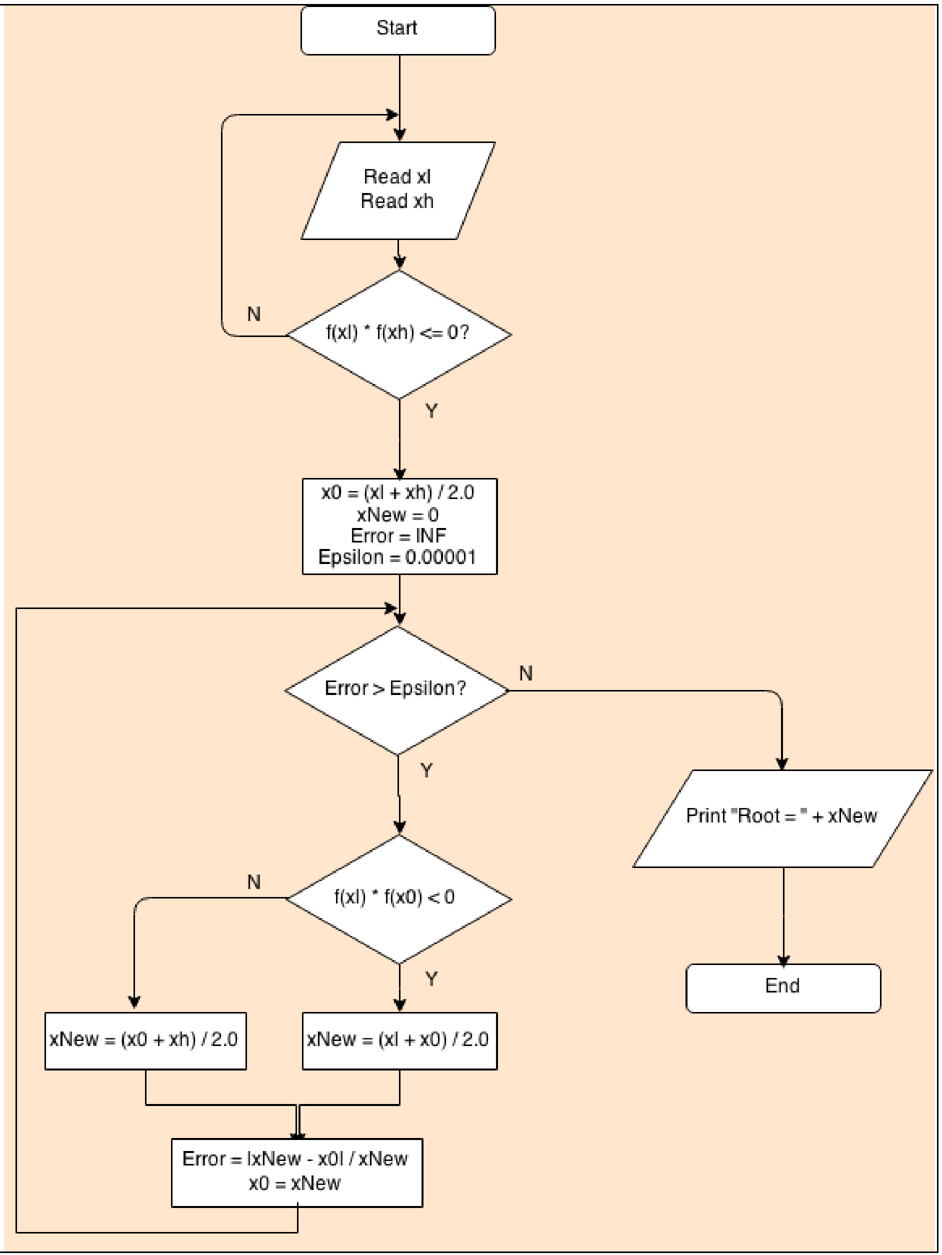
Gauss-Jordan Elimination Method:

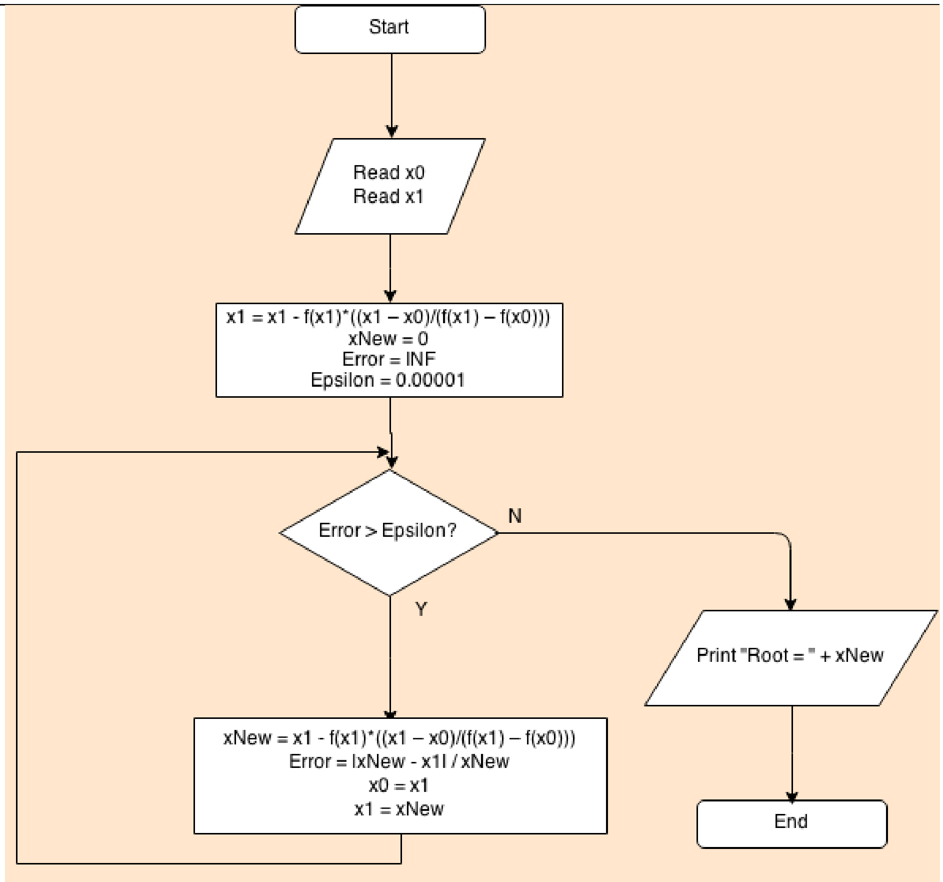
```
function gaussJordan(Matrix coefficients)
    for i: 0 → 3:
        pivot = coefficients[i, i]
        for j: 0 → 3:
            if j = i:
                continue
            factor = -1 * coefficients[j, i] / pivot
            coefficients[j] = coefficients[j] + factor * coefficients[i]
        endloop
    endloop

    return { coefficients[0, 3], coefficients[1, 3], coefficients[2, 3] }
endfuncion
```

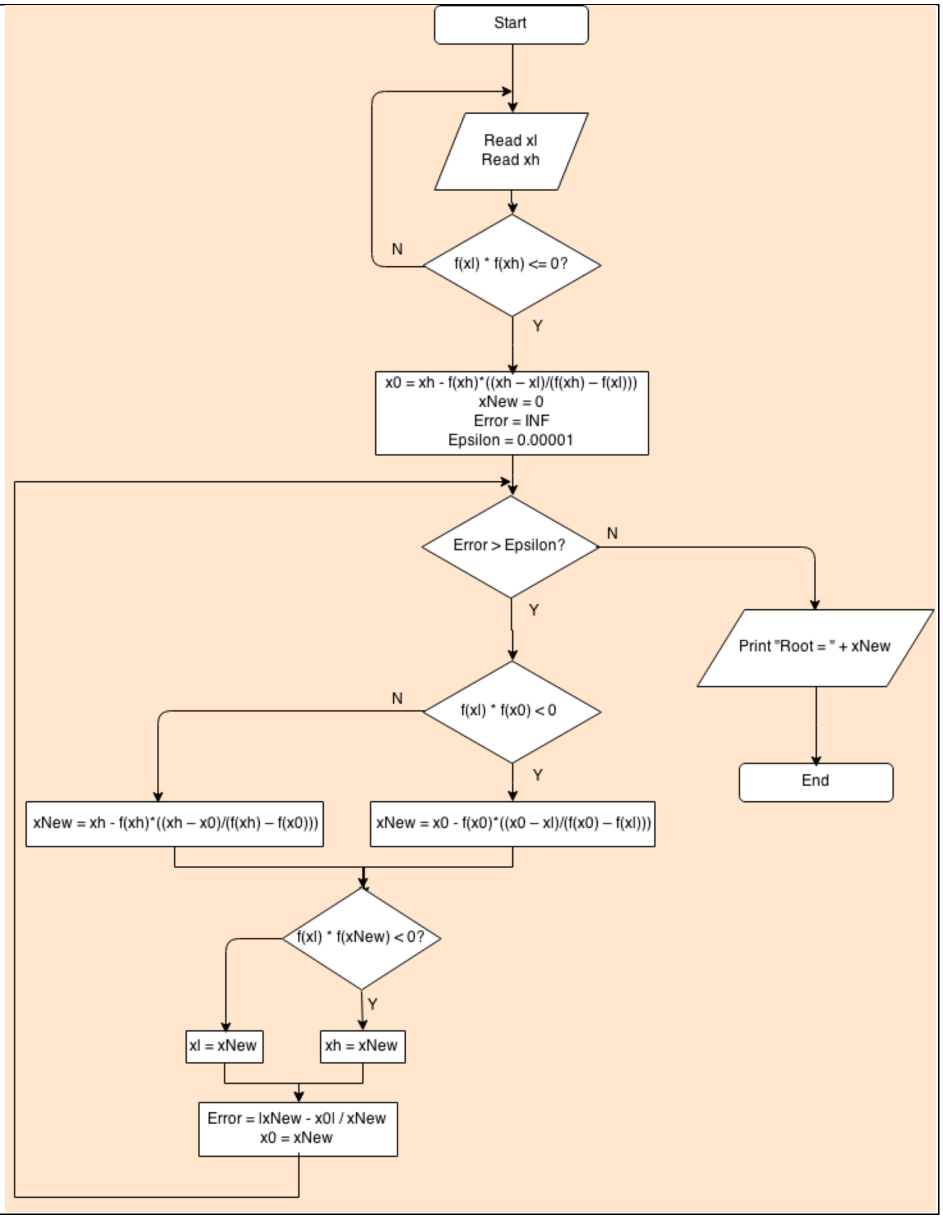
The Jacobi Method:

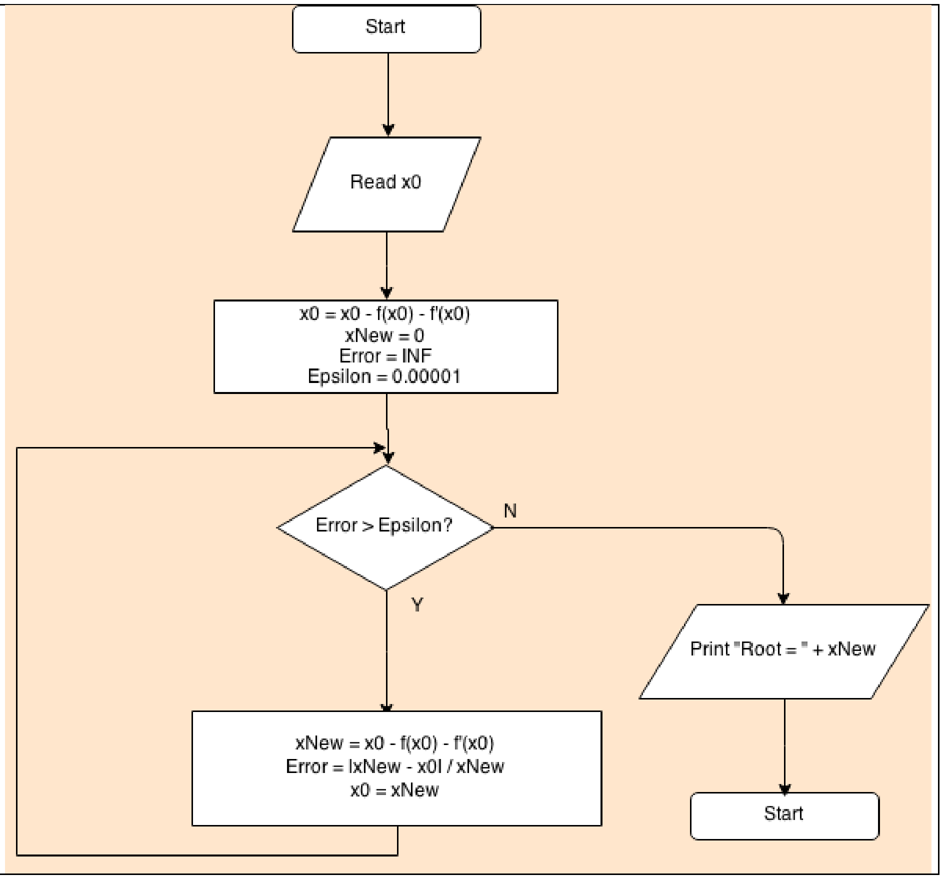
```
function jacobi(Equation f0(x, y), Equation f1(x, y), Equation f2(x, y))
    x0 = x1 = x2 = 0
    err0 = err1 = err2 = error = ∞

    iterate while(error > ε):
        xNew0 = f0(x1, x2)
        xNew1 = f1(x0, x2)
        xNew2 = f2(x0, x1)

        err0 = |xNew0 – x0| / xNew0
        err1 = |xNew1 – x1| / xNew1
        err2= |xNew2 – x2| / xNew2
        error = max(err0, err1, err2)

        x0 = xNew0
        x1 = xNew1
        x2 = xNew2
    endloop
    return {x0, x1, x2}
endfunction
```

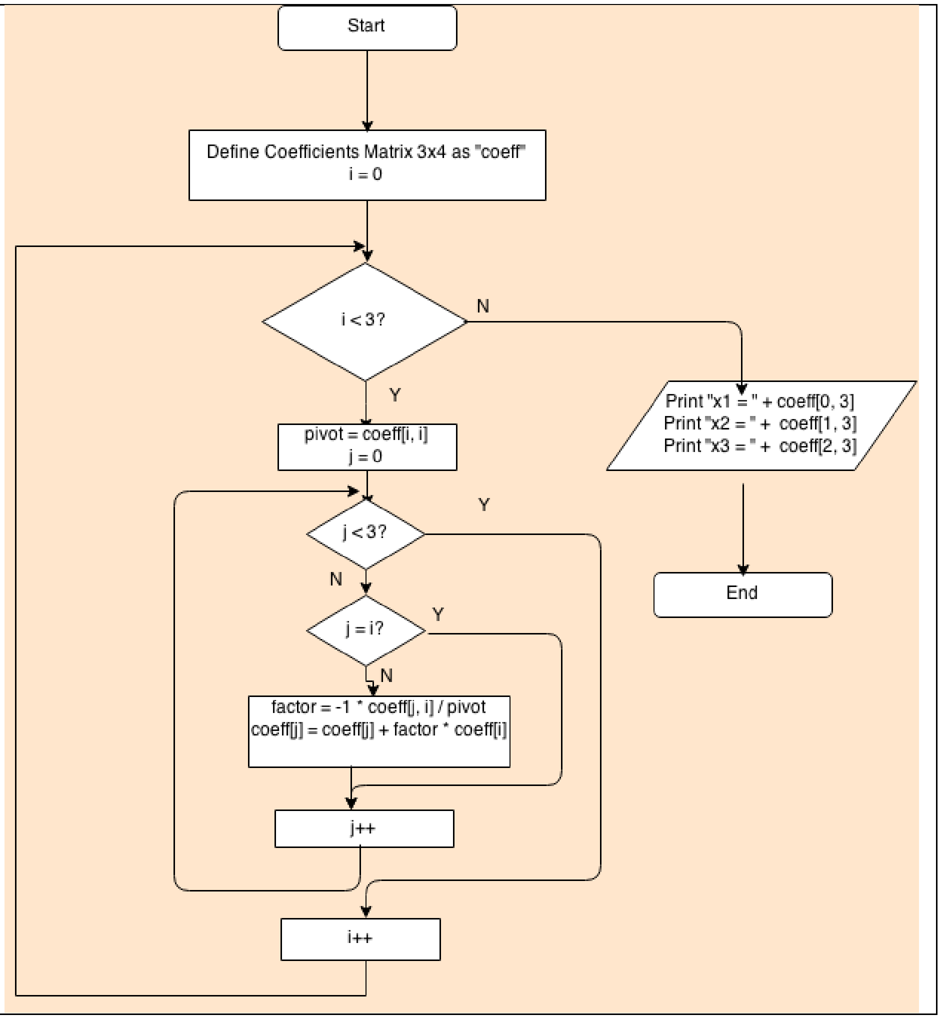
Flowcharts:

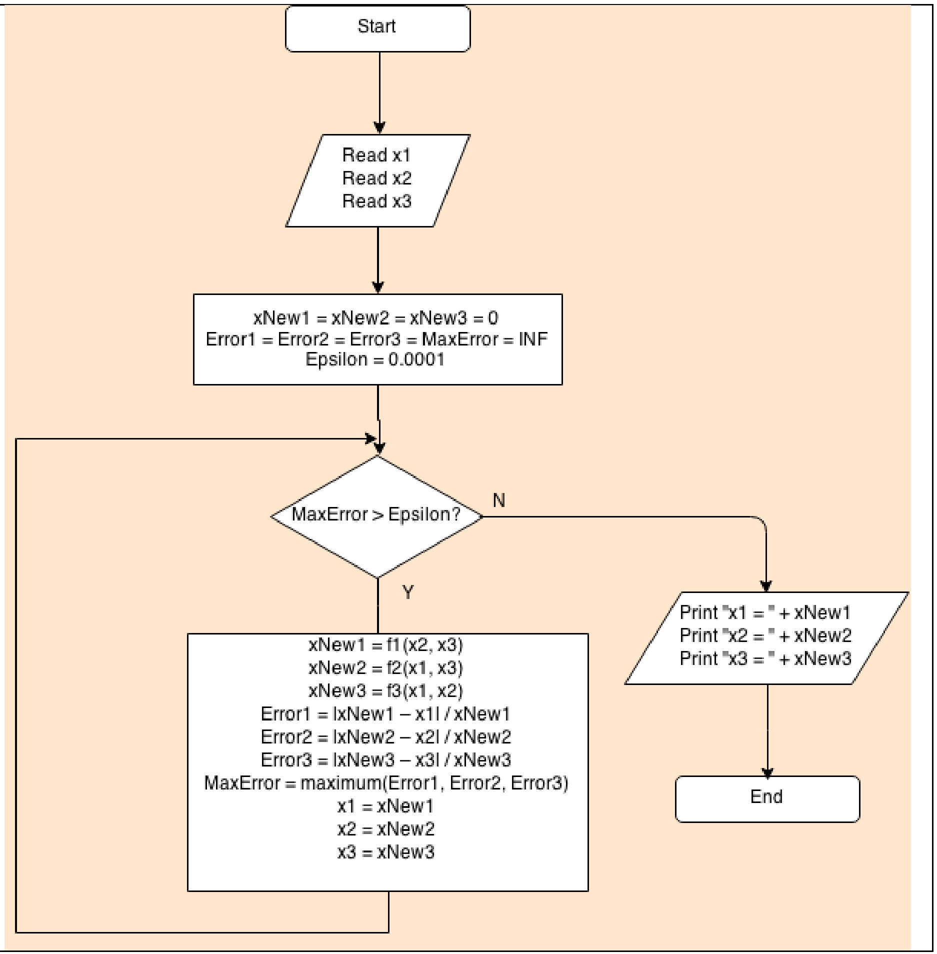
Bisection	

Secant	Start
	Read x0
	Read x1
	x1 = x1 - f(x1)*((x1 – x0)/(f(x1) – f(x0)))
	xNew = 0
	Error = INF
	Epsilon = 0.00001
	Error > Epsilon?
	N
	Y
	Print "Root = " + xNew
	xNew = x1 - f(x1)*((x1 – x0)/(f(x1) – f(x0)))
	Error = lxNew - x1l / xNew
	x0 = x1
	x1 = xNew
	End

False-Position	 Start Read xl Read xh f(xl) * f(xh) <= 0? N Y x0 = xh - f(xh)*((xh – xl)/(f(xh) – f(xl))) xNew = 0 Error = INF Epsilon = 0.00001 Error > Epsilon? N Y Print "Root = " + xNew End f(xl) * f(x0) < 0 N Y xNew = xh - f(xh)*((xh – x0)/(f(xh) – f(x0))) xNew = x0 - f(x0)*((x0 – xl)/(f(x0) – f(xl))) f(xl) * f(xNew) < 0? Y xl = xNew xh = xNew Error = lxNew - x0l / xNew x0 = xNew

Newton-Raphson	 Start
	Read x0
	x0 = x0 - f(x0) - f'(x0)
	xNew = 0
	Error = INF
	Epsilon = 0.00001
	Error > Epsilon?
	N
	Y
	Print "Root = " + xNew
	xNew = x0 - f(x0) - f'(x0)
	Error = IxNew - x0I / xNew
	x0 = xNew
	Start

Gauss-Jordan	 Start Define Coefficients Matrix 3x4 as "coeff" i = 0 i < 3? N Y Print "x1 = " + coeff[0, 3] Print "x2 = " + coeff[1, 3] Print "x3 = " + coeff[2, 3] End pivot = coeff[i, i] j = 0 j < 3? Y N j = i? Y N factor = -1 * coeff[j, i] / pivot coeff[j] = coeff[j] + factor * coeff[i] j++ i++

The Jacobi	

Computer Programs:

C++ files are attached, definitions of variables, functions and procedures are outlined through the comments of the source code.

<u>Recommended Initial Guesses:</u>

Equation1: [0.4, 0.6]

Equation2: [-0.6, -0.4]

Equation3: [0.4, 0.6]

Equation4: [1, 1.2]

Equation5: [0, 0.2]

Test Results:

Gauss-Jordan Elimination Method:

x0 = 0.5, x1 = 8, x2 = -6

error = 0%

iterations: N/A

The Jacobi Method:

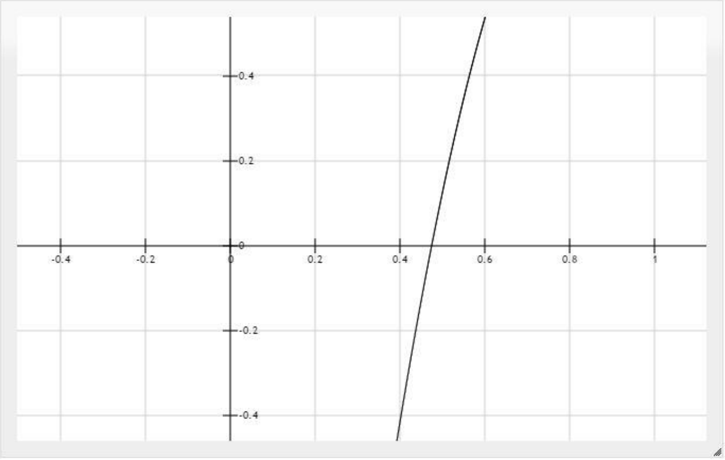
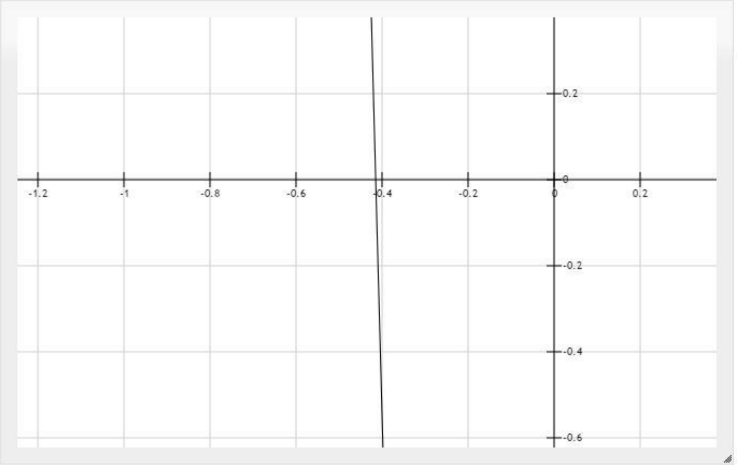
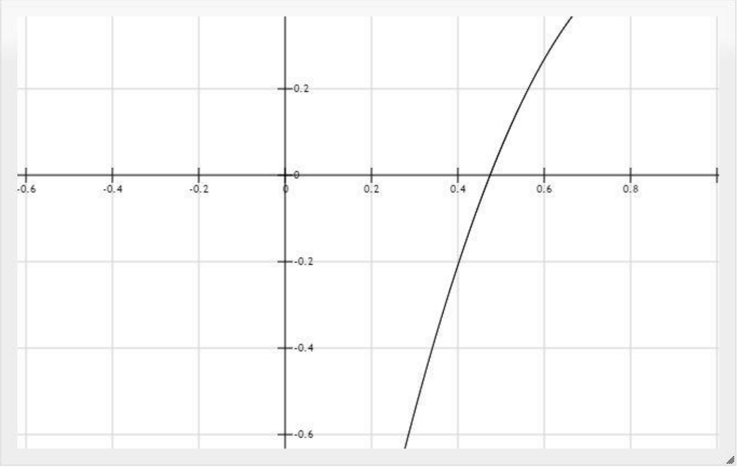
x0 = 0.500022, x1 = 8.00002, x2 = -6

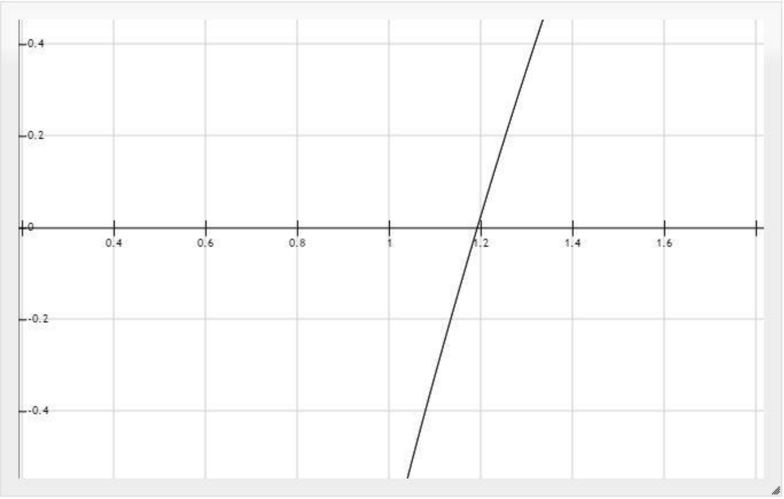
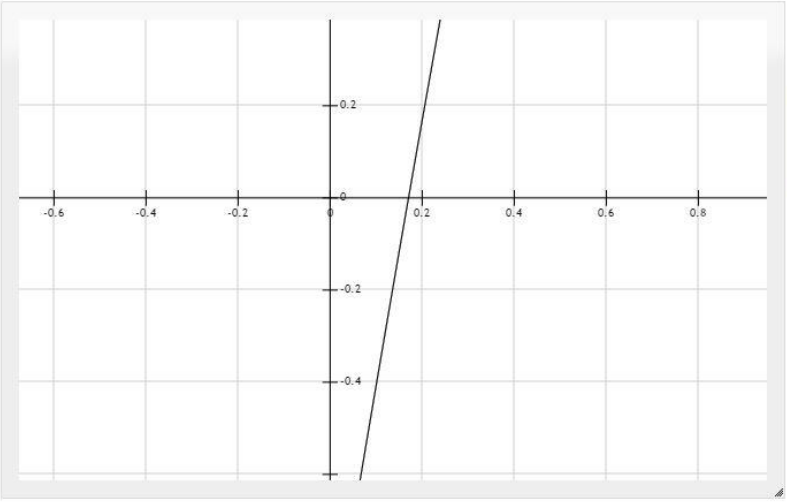
error = 0.00318659%

iterations: 10

Graphs for Initial Guesses:

#1

Equation 1	
Equation 2	
Equation 3	

Equation 4	
Equation 5	 0.2 0 -0.2 -0.4 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8