NUMERICAL METHODS USING ANGULAR

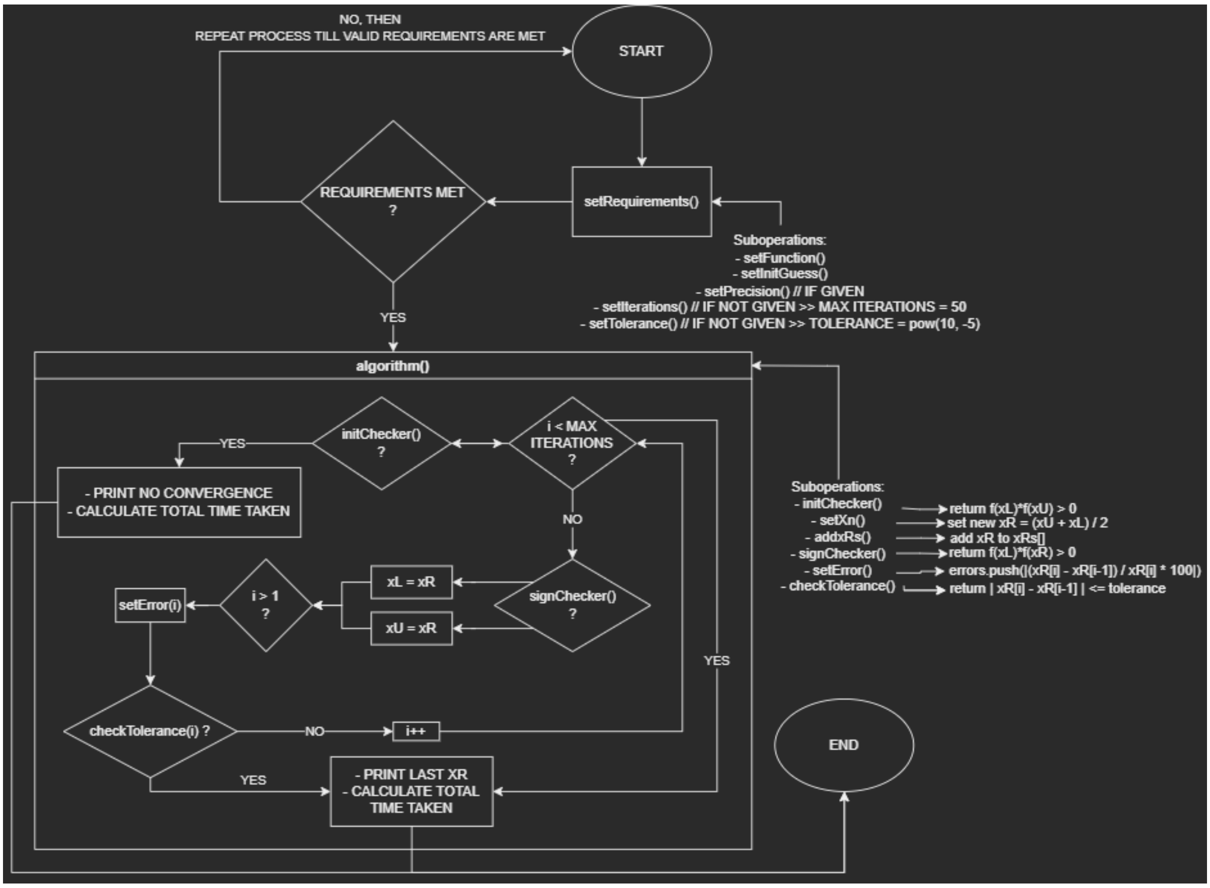
PROJECT - PHASE 1

Zeyad Ahmed Ibrahim Zidan	19015709
Basel Ahmed Awad Ayad	19015513
Yousef Saber Saeed Mohamed	19016924
Ali Hassan Ali Ahmed ELSharawy	19016013
Louay Magdy Abdel-Halim Ali	19016195

For a better reading experience, please consider viewing the report on google documents here.

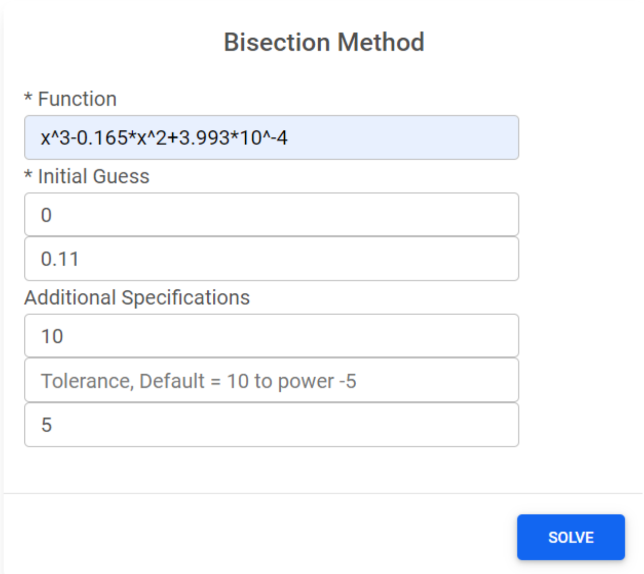
Bisection Method

Flowchart



Sample Runs

- Sample #1



…:

____

- Setting x root for iteration #2: x root = (xl + xu) / 2 = 0.0825
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.0825
- Evaluating error at iteration #2
- Relative Error = 33.333%
- ______________________________________
- Setting x root for iteration #3: x root = (xl + xu) / 2 = 0.06875
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.06875
- Evaluating error at iteration #3
- Relative Error = 20%
- ______________________________________
- Setting x root for iteration #4: x root = (xl + xu) / 2 = 0.061875
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned +. Setting x root to be the new x lower. x lower = 0.061875
- Evaluating error at iteration #4
- Relative Error = 11.111%
- ______________________________________
- Setting x root for iteration #5: x root = (xl + xu) / 2 = 0.065312
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.065312

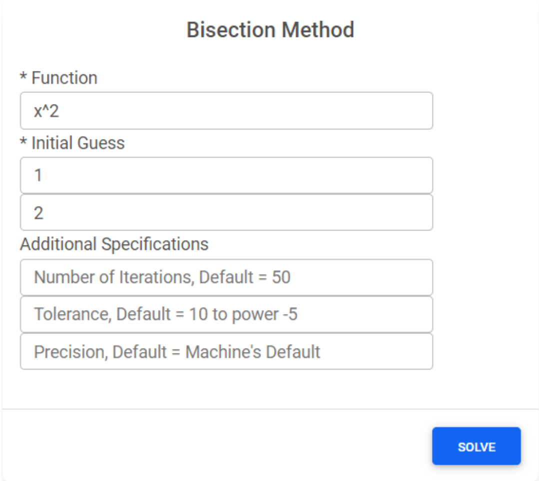
And so on till iteration #10. Note that we have calculated time taken and plotted solutions.

____

- Setting x root for iteration #10: x root = (xl + xu) / 2 = 0.062412
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.062412
- Evaluating error at iteration #10
- Relative Error = 0.17144%

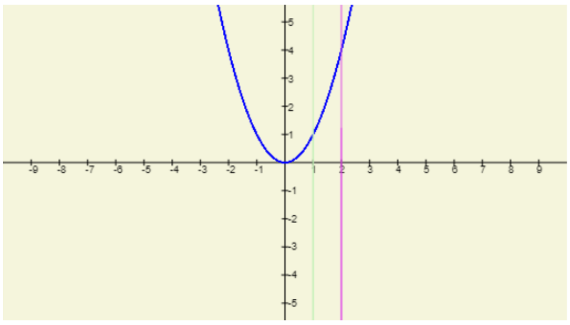
Time taken: 6ms

- Sample #2

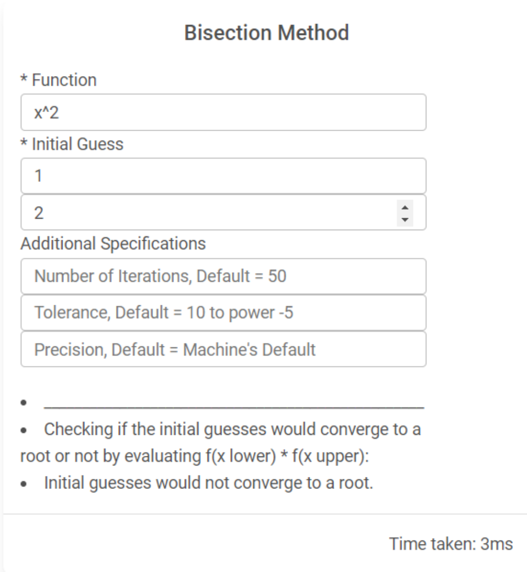


x^2 root = 0

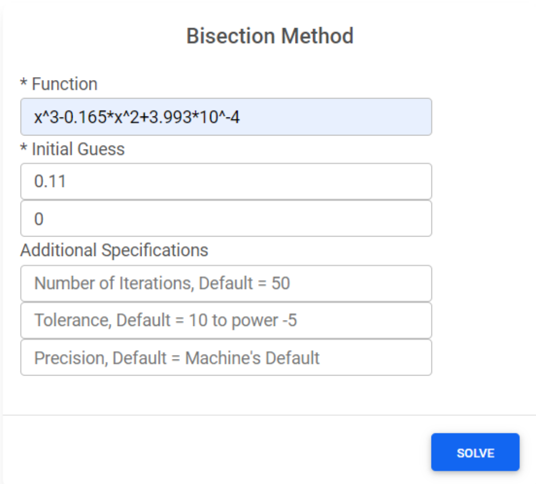
Init Guesses = {1, 2}



The solution

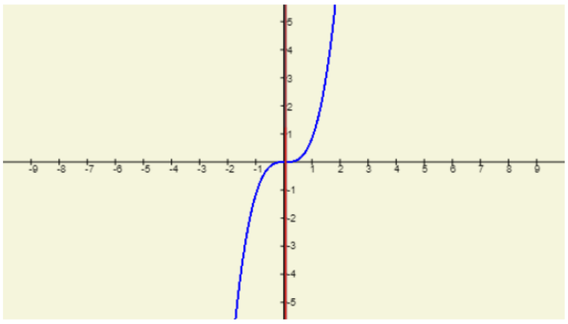


- Sample #3



When a user inputs

xL > XU



Bisection Method

* Function

x^3-0.165*x^2+3.993*10^-4

* Initial Guess

0.11

0

Additional Specifications

Number of Iterations, Default = 50

Tolerance, Default = 10 to power -5

Precision, Default = Machine's Default

- ______________________________
- Checking if the initial guesses would converge to a root or not by evaluating f(x lower) * f(x upper):
- Initial guesses would converge to a root.
- ______________________________
- Setting x root for iteration #1: x root = (xl + xu) / 2 = 0.055
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.055
- ______________________________

- ______________________________
- Setting x root for iteration #12: x root = (xl + xu) / 2 = 0.06238525390625001
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned +. Setting x root to be the new x lower. x lower = 0.06238525390625001
- Evaluating error at iteration #12
- Relative Error = 0.04304778303917882%
- ______________________________
- Setting x root for iteration #13: x root = (xl + xu) / 2 = 0.062371826171875006
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.062371826171875006
- Evaluating error at iteration #13
- Relative Error = 0.021528525296019892%
- ______________________________
- Setting x root for iteration #14: x root = (xl + xu) / 2 = 0.062378540039062506
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned +. Setting x root to be the new x lower. x lower = 0.062378540039062506
- Evaluating error at iteration #14
- Relative Error = 0.010763104079217779%
- Tolerance satisfied. ROOT FOUND! x root = 0.062378540039062506

Time taken: 7ms

Program still functns properly & calculates Xr

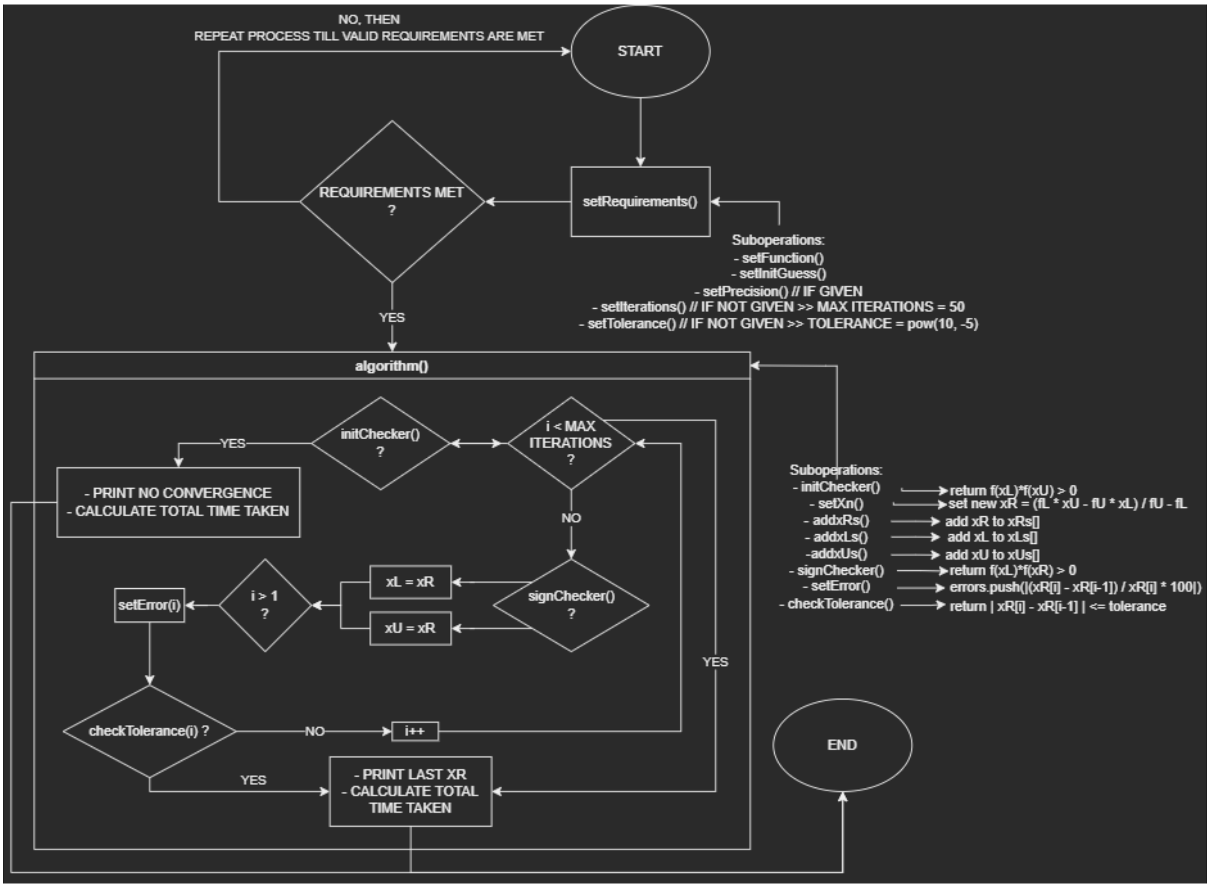
Used Data Structure

- ArrayLists used as a good storage for variables at the runtime
- Maps was useful when evaluating the equation input

False-Position Method

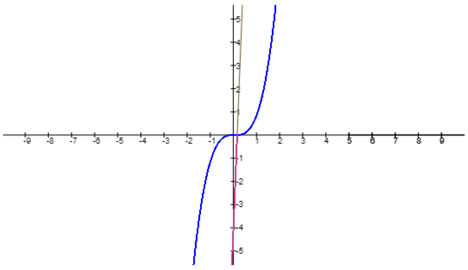
Flowchart

We note that the flowchart of false-position is pretty much like bisection with slight changes.



Sample Runs

- Sample #1



False-Position Method

* Function

x^3-0.165*x^2+3.993*10^-4

* Initial Guess

-5

5

Additional Specifications

Number of Iterations, Default = 50

Tolerance, Default = 10 to power -5

Precision, Default = Machine's Default

- ______________________________
- Checking if the initial guesses would converge to a root or not by evaluating f(x lower) * f(x upper):
- Initial guesses would converge to a root.
- ______________________________
- Setting x root for iteration #1: x root = (xl + xu) / 2 = 0.16498402799999987
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16498402799999987
- ______________________________
- Setting x root for iteration #2: x root = (xl + xu) / 2 = 0.1649680734394242
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.1649680734394242
- Evaluating error at iteration #2
- Relative Error = 0.009671301993783899%
- ______________________________

Continue…:

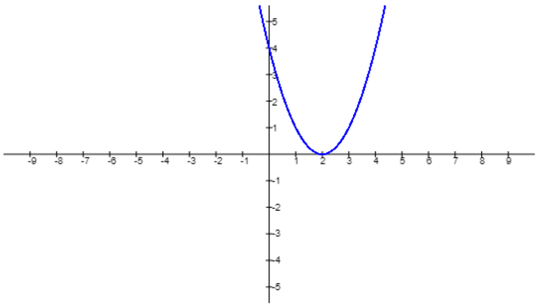
____
- Setting x root for iteration #3: x root = (xl + xu) / 2 = 0.16495213629240127
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16495213629240127
- Evaluating error at iteration #3
- Relative Error = 0.009661679673364246%
- ______________________________
- Setting x root for iteration #4: x root = (xl + xu) / 2 = 0.16493621653311102
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16493621653311102
- Evaluating error at iteration #4
- Relative Error = 0.009652070130424664%
- ______________________________
- Setting x root for iteration #5: x root = (xl + xu) / 2 = 0.164920314135785
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.164920314135785
- Evaluating error at iteration #5
- Relative Error = 0.009642473341958632%
- ______________________________
- Setting x root for iteration #6: x root = (xl + xu) / 2 = 0.16490442907470615
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16490442907470615
- Evaluating error at iteration #6
- Relative Error = 0.009632889285017985%
- ______________________________

And so on till all iterations are consumed. Note that we have calculated time taken and plotted solutions

- ___________________________________
- Setting x root for iteration #48: x root = (xl + xu) / 2 = 0.1642525780225253
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.1642525780225253
- Evaluating error at iteration #48
- Relative Error = 0.009241559555729632%
- ___________________________________
- Setting x root for iteration #49: x root = (xl + xu) / 2 = 0.16423741479992152
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16423741479992152
- Evaluating error at iteration #49
- Relative Error = 0.009232501998557073%
- ___________________________________
- Setting x root for iteration #50: x root = (xl + xu) / 2 = 0.16422226783095314
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.16422226783095314
- Evaluating error at iteration #50
- Relative Error = 0.009223456214819408%
- Max iterations consumed. ROOT FOUND! x root = 0.16422226783095314

Time taken: 41ms

- Sample #2
  In this case, the user initial guess outputs positive values when entering the function. False Position would never converge in such a case.



False-Position Method

* Function

(x-2)^2

* Initial Guess

-5

5

Additional Specifications

Number of Iterations, Default = 50

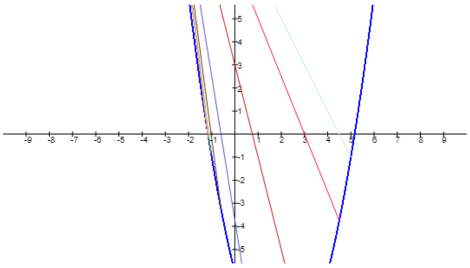
Tolerance, Default = 10 to power -5

Precision, Default = Machine's Default

- ___________________________________
- Checking if the initial guesses would converge to a root or not by evaluating f(x lower) * f(x upper):
- Initial guesses would not converge to a root.

Time taken: 2ms

- Sample #3



False-Position Method

* Function
(x-2)^2-10
* Initial Guess
-3
5
Additional Specifications
Number of Iterations, Default = 50
Tolerance, Default = 10 to power -5
Precision, Default = Machine's Default

- ____
- Checking if the initial guesses would converge to a root or not by evaluating f(x lower) * f(x upper):
- Initial guesses would converge to a root.
- ____
- Setting x root for iteration #1: x root = (xl + xu) / 2 = 4.5
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 4.5
- ____
- Setting x root for iteration #2: x root = (xl + xu) / 2 = 3
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 3
- Evaluating error at iteration #2
- Relative Error = 50%
- ____
- Setting x root for iteration #3: x root = (xl + xu) / 2 = 0.75
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = 0.75
- Evaluating error at iteration #3
- Relative Error = 300%
- ____
- Setting x root for iteration #4: x root = (xl + xu) / 2 = -0.6
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -0.6
- Evaluating error at iteration #4
- Relative Error = 225.00000000000006%
- ____
- Setting x root for iteration #5: x root = (xl + xu) / 2 = -1.0263157894736843
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -1.0263157894736843
- Evaluating error at iteration #5
- Relative Error = 41.53846153846155%
- ____
- Setting x root for iteration #6: x root = (xl + xu) / 2 = -1.131147540983607
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -1.131147540983607
- Evaluating error at iteration #6
- Relative Error = 9.267734553775767%
- ____

And so on till convergence on iteration #13.

- Setting x root for iteration #11: x root = (xl + xu) / 2 = -1.1622595607293897
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -1.1622595607293897
- Evaluating error at iteration #11
- Relative Error = 0.005359281473255834%
- ____
- Setting x root for iteration #12: x root = (xl + xu) / 2 = -1.162273585102753
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -1.162273585102753
- Evaluating error at iteration #12
- Relative Error = 0.0012066327190982536%
- ____
- Setting x root for iteration #13: x root = (xl + xu) / 2 = -1.1622767426741225
- Checking whether f(x lower) * f(x root) is positive, negative, or zero:
- Evaluation returned -. Setting x root to be the new x upper. x upper = -1.1622767426741225
- Evaluating error at iteration #13
- Relative Error = 0.00027167121680662%
- Tolerance satisfied. ROOT FOUND! x root = -1.1622767426741225

Time taken: 8ms

Used Data Structure

- ArrayLists used as a good storage for variables at the runtime
- Maps was useful when evaluating the equation input

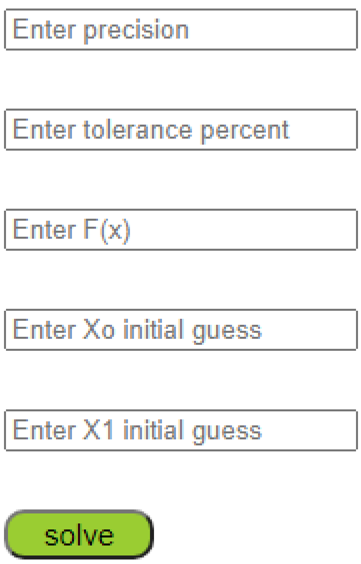
Secant Method:

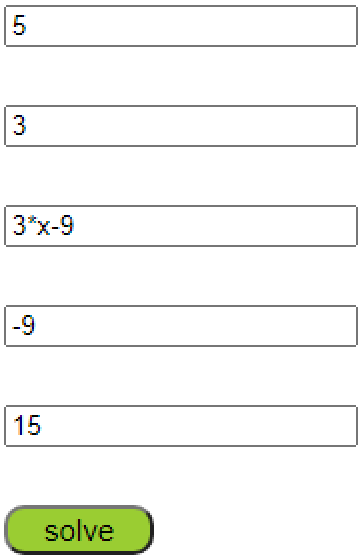
- **Data Structures used**: Array.
- **Pseudo code:**

```
Secant Function
        beginTime = time of beginning
        While(maxIteration > 0 && relativeError > tolerance  &&  !diverge )

                prevRelative = (x - prevX)/x
                nextX = x - (( F(x) * (prevX - x)) / (F(prevX)-F(x) ))
                relativeError =(nextX - x) / nextX

                If( relativeError > prevRelative)
                        divergeTimes--
                End If
                If( divergeTimes = 0)
                        diverge = true
                End If

                prevCoff.push(prevX);
                prevCoffSub.push( F(prevX) )
                coff.push(x);
                coffSub.push( F(x) )
                nextCoff.push( nextX )
                nextCoffSub.push( F(nextX) )
                relatives.push( relativeError * 100
                prevX = x
```

```
                x = nextX
                maxIteration–

                runtime = beginTime - EndTime
        End While
    End Function
```

- **Sample run:**




5
3
3*x-9
-9
15
solve

Secant Method
(you can modify and solve again without refresh)

Run Time = 516 milli second

Iteration: 0

X0 = -9 || X1 = 15 || X2 = 3

F(X0) = -36 || F(X1) = 36 || F(X2) = 0

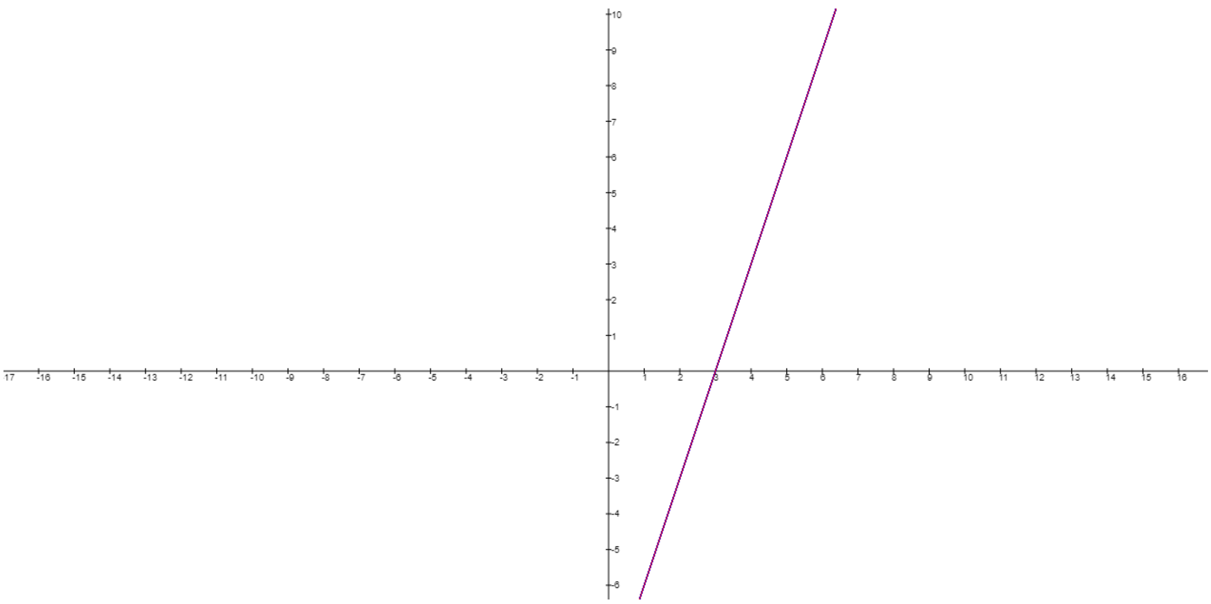
Relative Error = 400%

Iteration: 1

X1 = 15 || X2 = 3 || X3 = 3

F(X1) = 36 || F(X2) = 0 || F(X3) = 0

Relative Error = 0%



3

0.01

x^2+3*x-9

-18

20

solve

Secant Method
(you can modify and solve again without refresh)

Run Time = 638 milli second

Iteration: 0

X0 = -18 || X1 = 20 || X2 = -70.2

F(X0) = 261 || F(X1) = 451 || F(X2) = 4708.44

Relative Error = 128%

Iteration: 1

X1 = 20 || X2 = -70.2 || X3 = 29.6

F(X1) = 451 || F(X2) = 4708.44 || F(X3) = 955.96

Relative Error = 337%

Iteration: 2

X2 = -70.2 || X3 = 29.6 || X4 = 55

F(X2) = 4708.44 || F(X3) = 955.96 || F(X4) = 3181

Relative Error = 46.2%

Iteration: 3

X3 = 29.6 || X4 = 55 || X5 = 18.7

F(X3) = 955.96 || F(X4) = 3181 || F(X5) = 396.78999999999996

Relative Error = 194%

Iteration: 4

X4 = 55 || X5 = 18.7 || X6 = 13.5

F(X4) = 3181 || F(X5) = 396.78999999999996 || F(X6) = 213.75

Relative Error = 38.5%

Iteration: 5

X5 = 18.7 || X6 = 13.5 || X7 = 7.43

F(X5) = 396.78999999999996 || F(X6) = 213.75 || F(X7) = 68.4949

Relative Error = 81.69999999999999%

Iteration: 6

X6 = 13.5 || X7 = 7.43 || X8 = 4.57

F(X6) = 213.75 || F(X7) = 68.4949 || F(X8) = 25.594900000000003

Relative Error = 62.6%

Iteration: 7

X7 = 7.43 || X8 = 4.57 || X9 = 2.86

F(X7) = 68.4949 || F(X8) = 25.594900000000003 || F(X9) = 7.759599999999999

Relative Error = 59.8%

Iteration: 8

X8 = 4.57 || X9 = 2.86 || X10 = 2.12

F(X8) = 25.594900000000003 || F(X9) = 7.759599999999999 || F(X10) = 1.8544000000000018

Relative Error = 34.9%

Iteration: 9

X9 = 2.86 || X10 = 2.12 || X11 = 1.89

F(X9) = 7.759599999999999 || F(X10) = 1.8544000000000018 || F(X11) = 0.24210000000000065

Relative Error = 12.2%

Iteration: 10

X10 = 2.12 || X11 = 1.89 || X12 = 1.86

F(X10) = 1.8544000000000018 || F(X11) = 0.24210000000000065 || F(X12) = 0.03960000000000008

Relative Error = 1.6099999999999999%

Iteration: 11

X11 = 1.89 || X12 = 1.86 || X13 = 1.85

F(X11) = 0.24210000000000065 || F(X12) = 0.03960000000000008 || F(X13) = -0.027499999999999858

Relative Error = 0.541%

Iteration: 12

X12 = 1.86 || X13 = 1.85 || X14 = 1.85

F(X12) = 0.03960000000000008 || F(X13) = -0.027499999999999858 || F(X14) = -0.027499999999999858

Relative Error = 0%