Alex Chen
Professor Raheja
CS 3010
November 11th, 2018

Project 2: Find Roots of Polynomials

Introduction

For this project, we tested five methods for finding the roots of polynomials. These methods include: *bisection, false-position, newton-raphson, secant, & modified-secant.* The next section will discuss the results and comparisons of the methods. Later, the data obtained from the iterations will be displayed in tables and graphical formats.

It's important to note that the graphs of *iteration vs approximate* error uses additional data by reducing the relative error to .0001 (.01%). This allows us to examined more iterations and see how the methods compare as the it gets closer to the root (I included some *zoomed in* graphs that show the method's behaviors as the approximate error gets extremely small). In addition, values from function evaluations are computed and stored as doubles. These values are rounded to 4 decimal places when displayed in the tables so that it fits in the console for display.

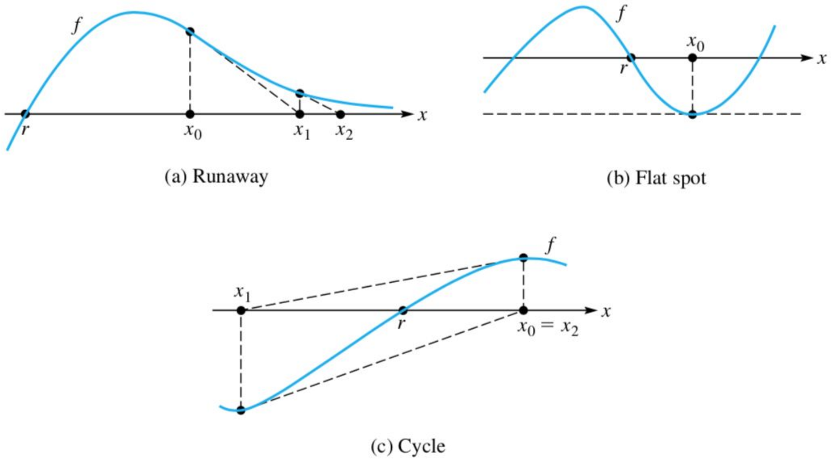
From this point on:

- **Function A**: $2x^3 – 11.7x^2 + 17.7x – 5$
- **Function B**: x + 10 – xcosh(50/x)

Newton-Raphson Method

Newton's method is 1 of the 3 open methods that were implemented. This method utilizes the derivative of functions in order to derive the next guess. Similar to all open methods, Newton's method does not need initial points that are near the root and is very efficient in finding the root. For *function A,* it was one of the fastest methods in finding the root and the same can be said for *function B*.

The drawback of this method is choosing the appropriate initial value. Due to the fact that this method utilizes the derivative of functions, in situations where the derivative is zero, this method will fail. Newton's method also might encounter problems when the initial value is near local minimas/maximas as well as inflection points. **These initial guesses might recede from the target root and diverge towards another root**. For convergence analysis, the Newton's method does not guarantee convergence. Some of the problematic situations are displayed in the figure below (Figure 3.6 from the book). Overall, Newton's method offers fast convergence on the root but the initial guess is very important. In situations where a good initial guess can be made, Newton's method would be ideal.



(a) Runaway

(b) Flat spot

(c) Cycle

FIGURE 3.6 Failure of Newton's method due to bad starting points

Secant Method

The secant method is another open method that's similar to Newton's method. Instead of using the derivative of functions, this method utilizes the secant line from two points. This method is useful when the derivative is hard or impossible to find. The graphs for *function A* show that the secant method was fast for some of the roots while slow for others. And this method is relatively slow for *function B.* This result depends heavily on the initial values as they do not need to bracket the root. Better initial values will improve the speed dramatically and should be comparable to Newton's method in most cases.

Similar to Newton's method, the drawback of the secant method is division-by-zero error. If the two points have the same functional value, the derived secant line will be parallel to the x-axis and division-by-zero error will occur. Therefore, the secant method will not guarantee convergence. This method should be considered with Newton's method and is appropriate when the derivative is hard to obtain.

Modified Secant Method

The modified secant method essentially utilizes the same mechanism as the secant method with minor modifications. A delta value is used to determine the next guess and this value will impact the performance of this method. A small delta value will result in inefficiencies regarding round offs, and a large delta value can result in divergence (Therefore, this method does not guarantee convergence). From the graphs of *function A* and *function B*, the modified secant method is very inefficient for some roots and average for others. It would be interesting to analyze the effects of the delta value on the performance more carefully. In some of the graphs, the modified secant method was significantly faster than the secant method. If one can find a delta value that is catering for the particular root, this method can be very effective.

Bisection Method

The bisection method is 1 of the 2 bracket methods that were implemented. The initial values were chosen to be relatively close to the actual roots. The closer the initial values bracket the roots, the faster the bisection method will be in finding the true root. This is because the bisection method essentially cuts the bracket range in half and picking one of them in each literation. Furthermore, the method may be more effect in finding roots that are finite/rational based on it’s mechanism.

For *convergence analysis*, this method will guarantee to find the root if the initial values brackets the root. On the other hand, this method is relatively slow when compared to the other methods. For *function A,* the bisection method was one of the slowest in finding the first two roots. And for *function B,* the method was faster at first because of the effective initial values (120,130) but it becomes the slowest as the approximate error decreased. This observation asserts the idea that the bisection method is relatively steady in finding the root. The derived values, *c,* will get closer to the root in a steady fashion as the iterations increase but it won’t be as fast as the open methods in the following sections.

False-Position

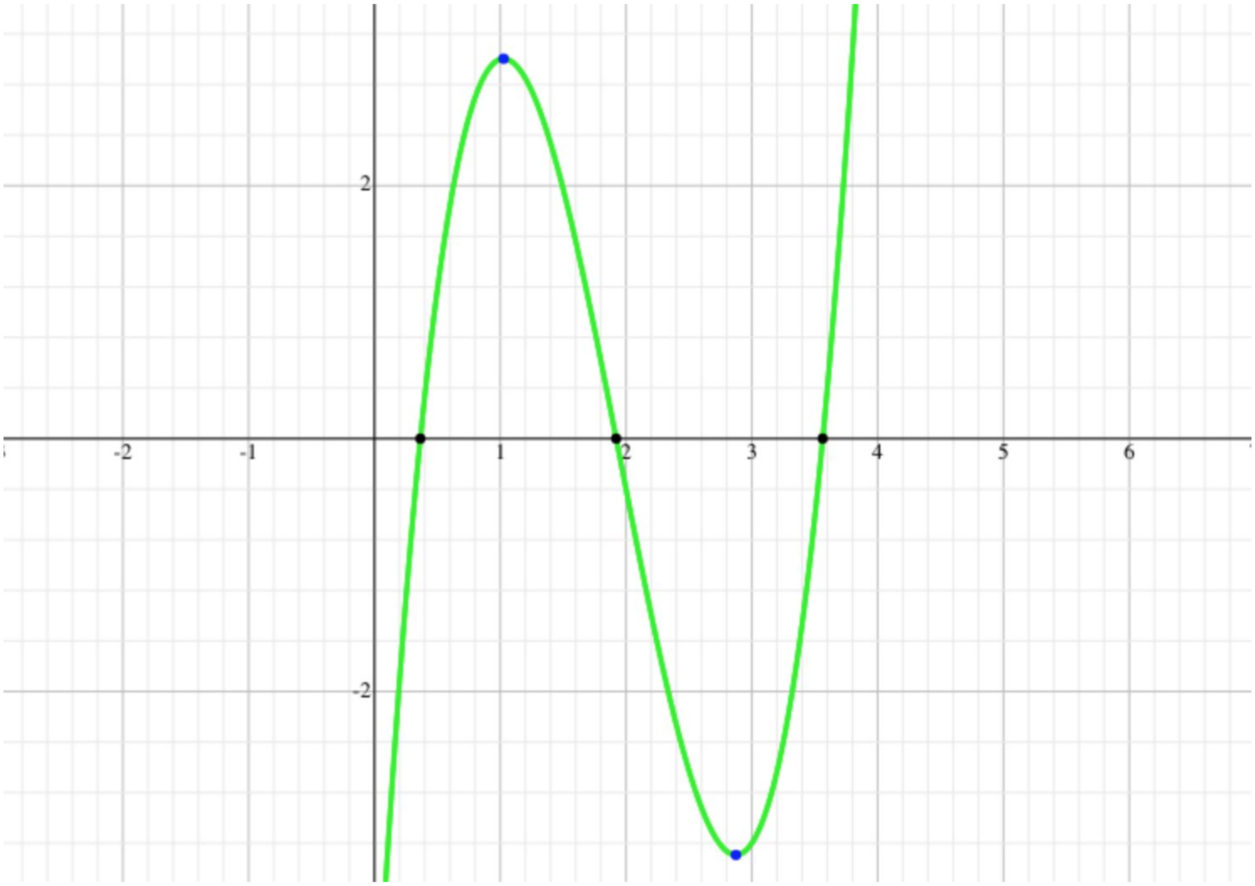
The false-position method is the second bracketing method we implemented. This method is similar to the bisection method but the *c* value is derived differently. The *c* value is selected based on a trigonomic evaluation of where the secant line intersects the x-axis. From my results, this method is one of the fastest methods that was used. It came in first for three of the roots and it beat the bisection method in all of the test. This may be because of the initial values picked for the experiments; as poor initial values can lead to poor convergence. Depending on the function’s curvature, the bracket points can become fixed and the speed of the method can reduce significantly. In general, the false-position method can be very effective when performed on appropriate functions.

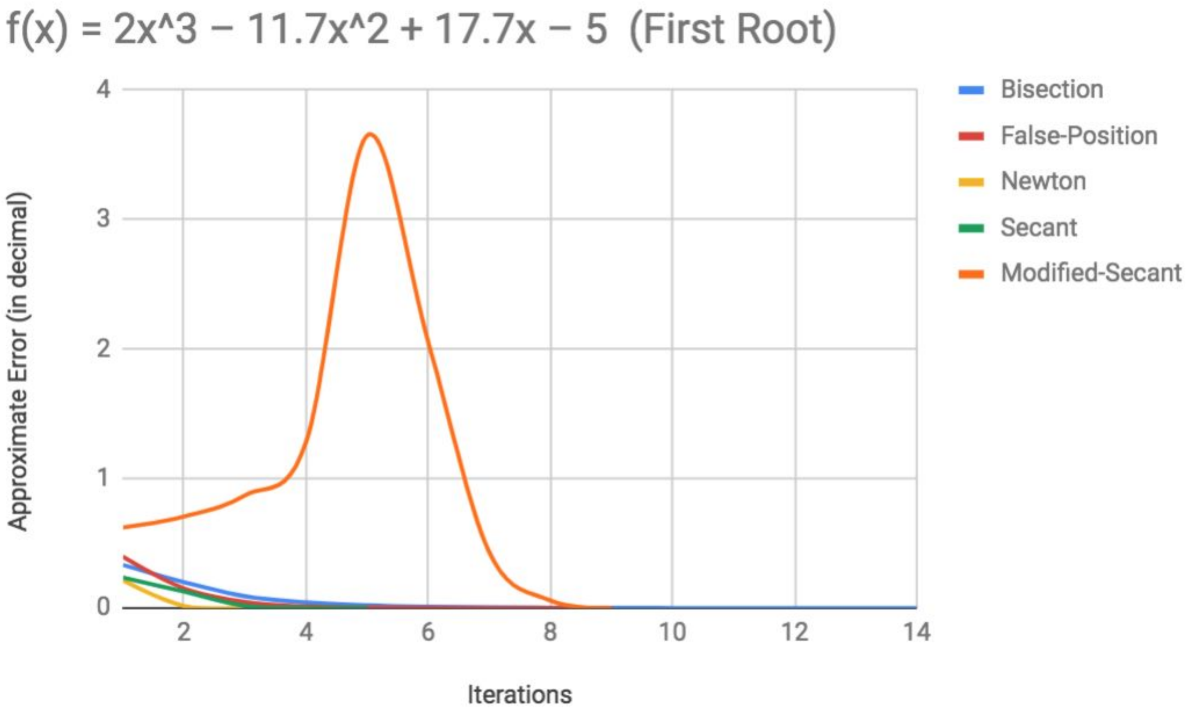
Data Types

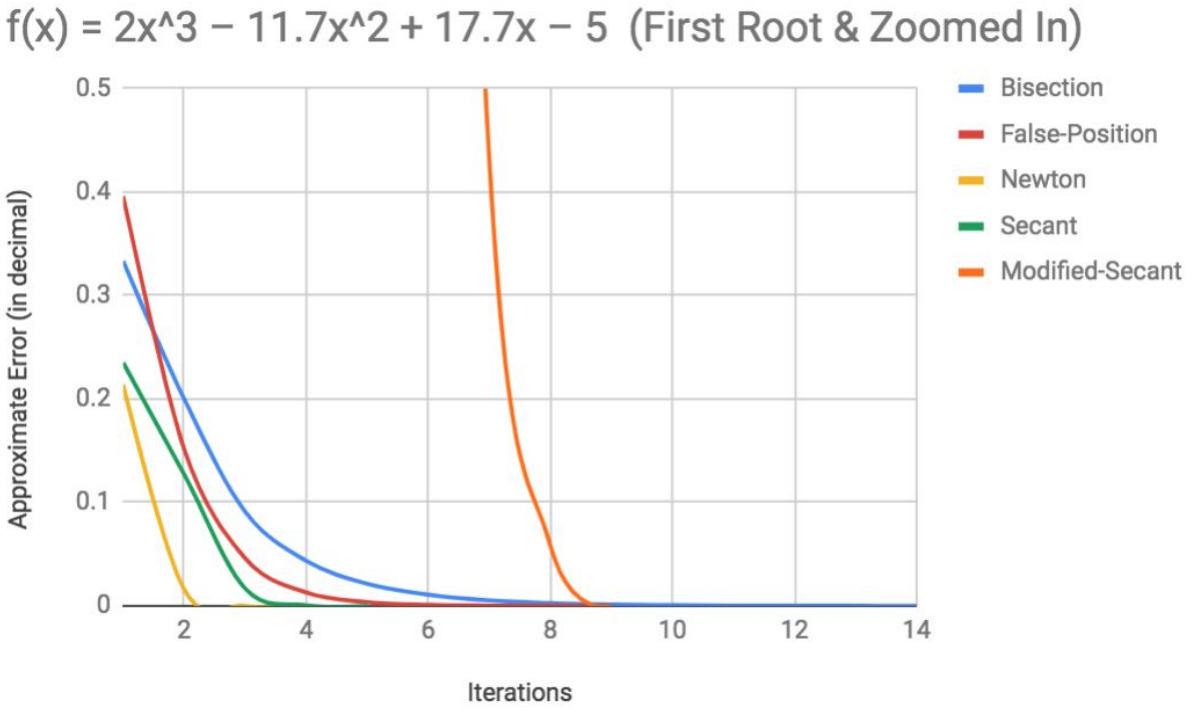
For this project, the main concern I had was the round off errors. Given that I had to store decimal values, the options were either *floats* or *doubles.* Floats in Java are four bytes in size and have a seven decimal digit precision. Doubles are eight bytes in size and have a sixteen decimal digit precision. I decided to use doubles to store my data because it allowed for smaller round off errors.

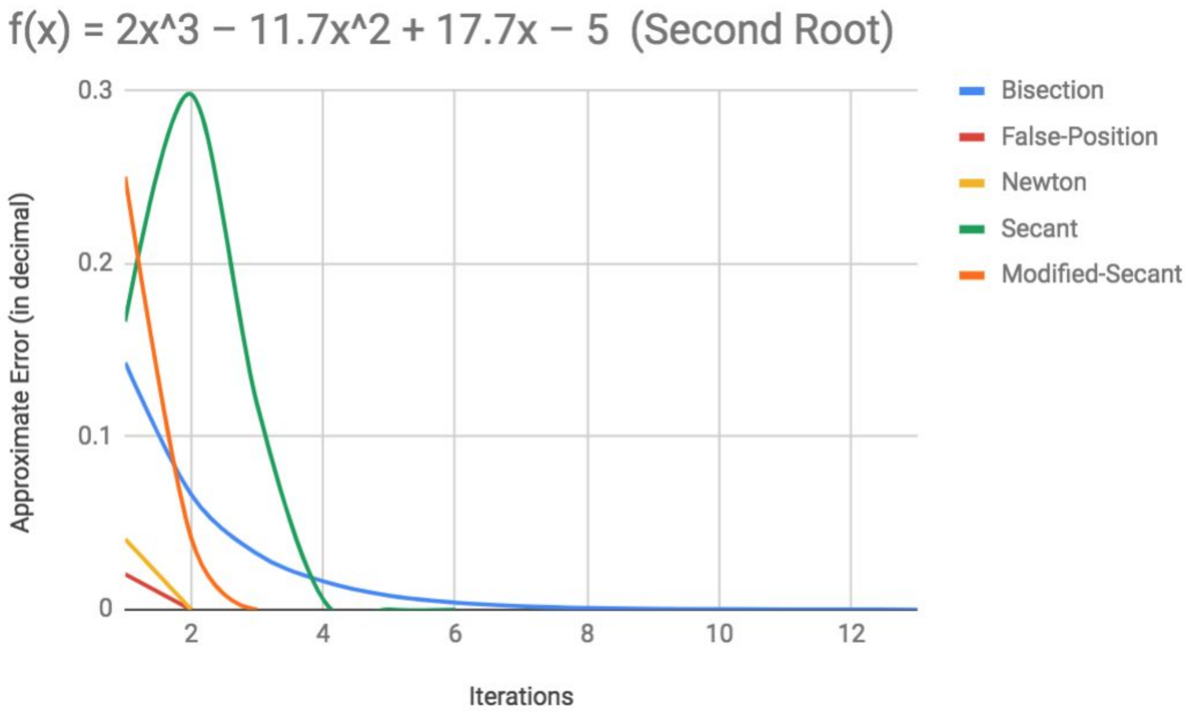
Data

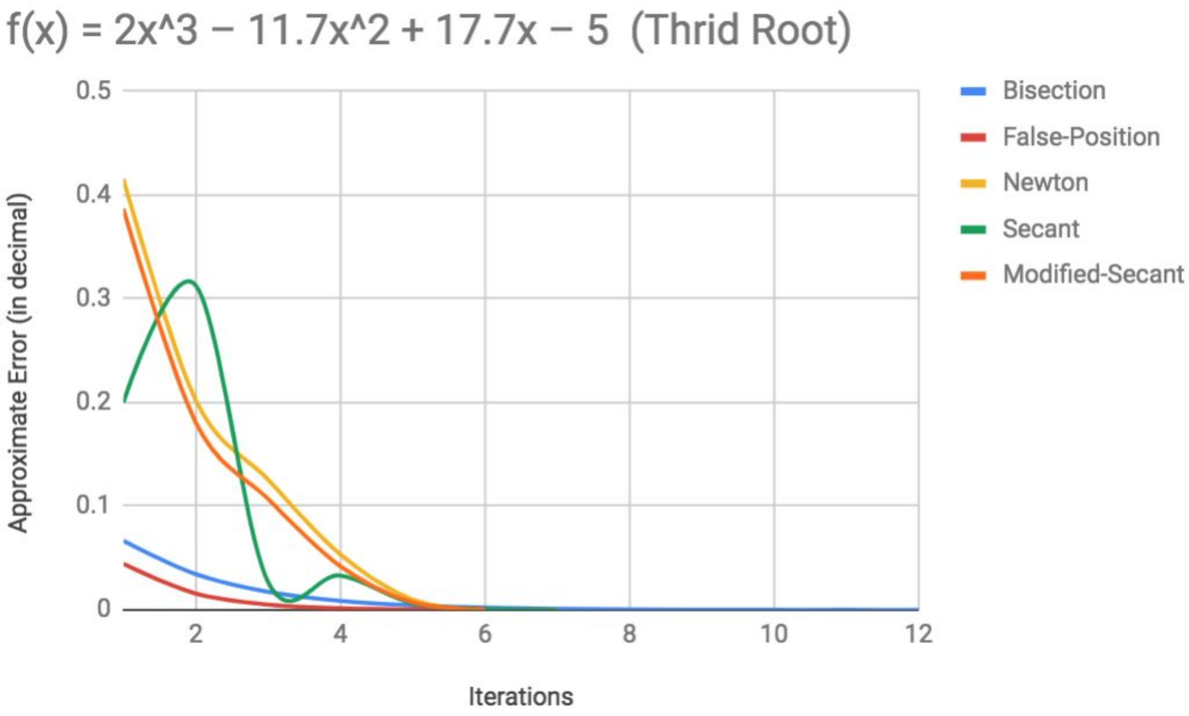
Function A



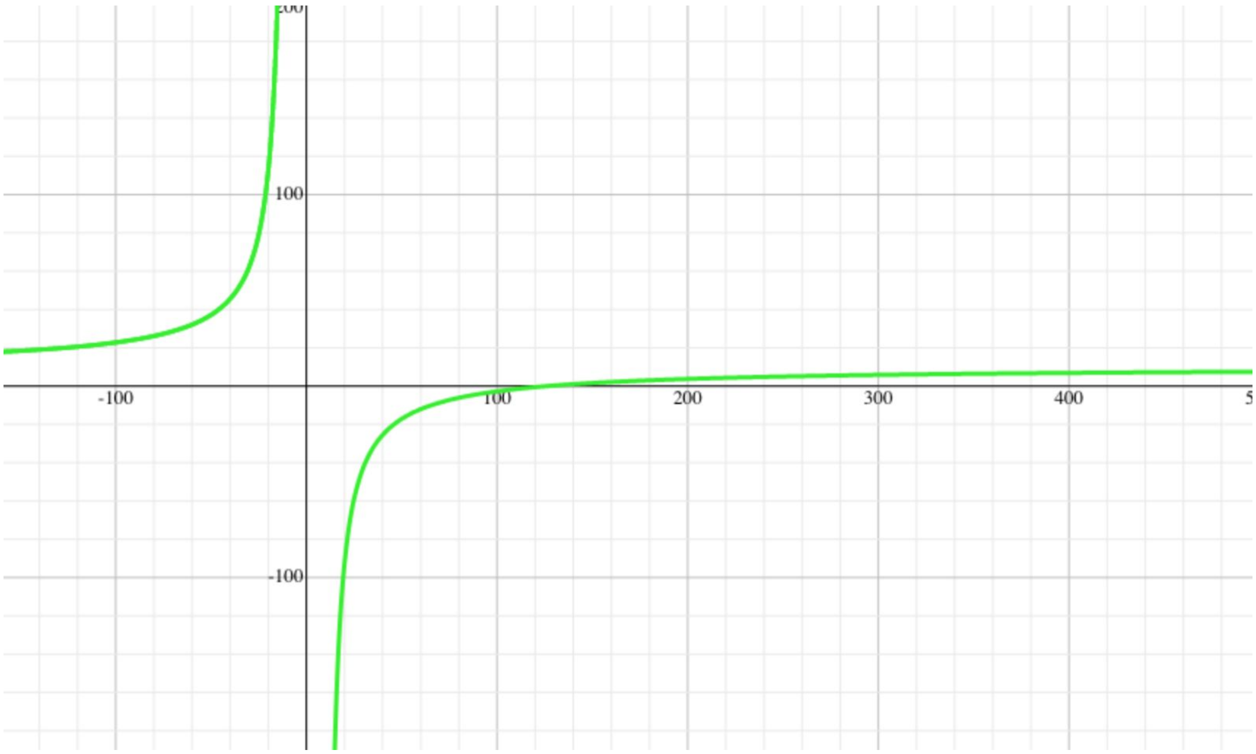

f(x) = 2x^3 – 11.7x^2 + 17.7x – 5 (First Root)
Approximate Error (in decimal)
0
1
2
3
4
Iterations
2
4
6
8
10
12
14
Bisection
False-Position
Newton
Secant
Modified-Secant

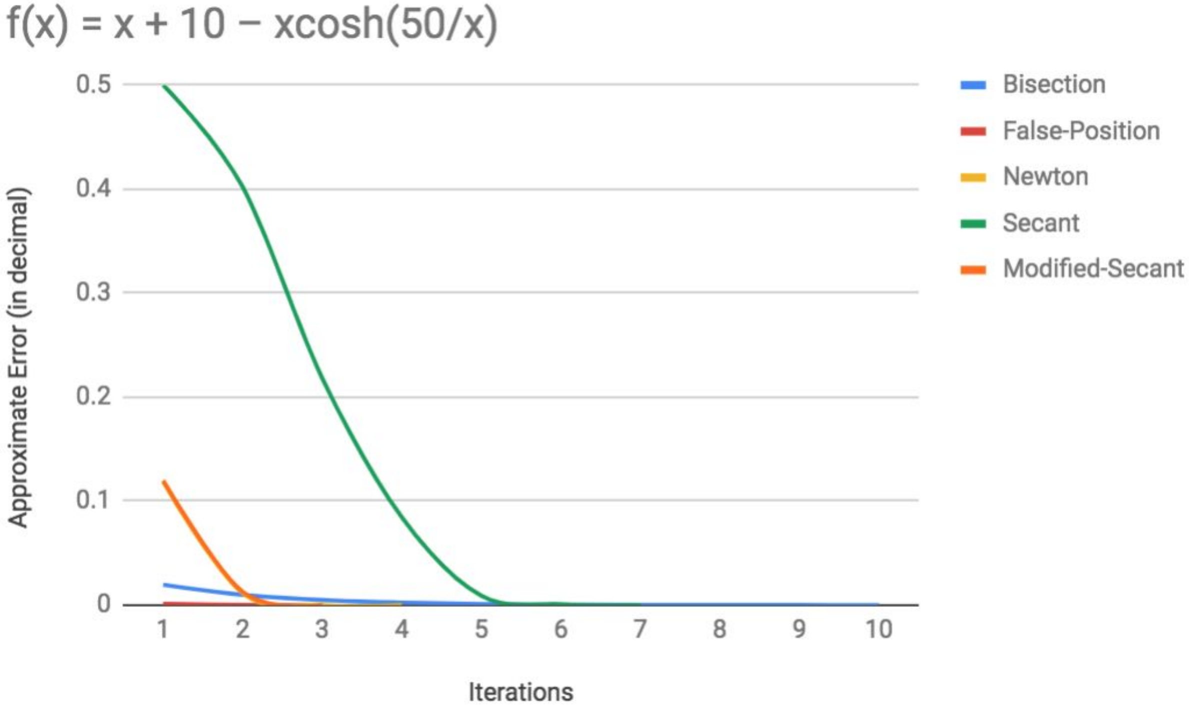

f(x) = 2x^3 – 11.7x^2 + 17.7x – 5 (First Root & Zoomed In)
Approximate Error (in decimal)
0
0.1
0.2
0.3
0.4
0.5
Iterations
2
4
6
8
10
12
14
Bisection
False-Position
Newton
Secant
Modified-Secant

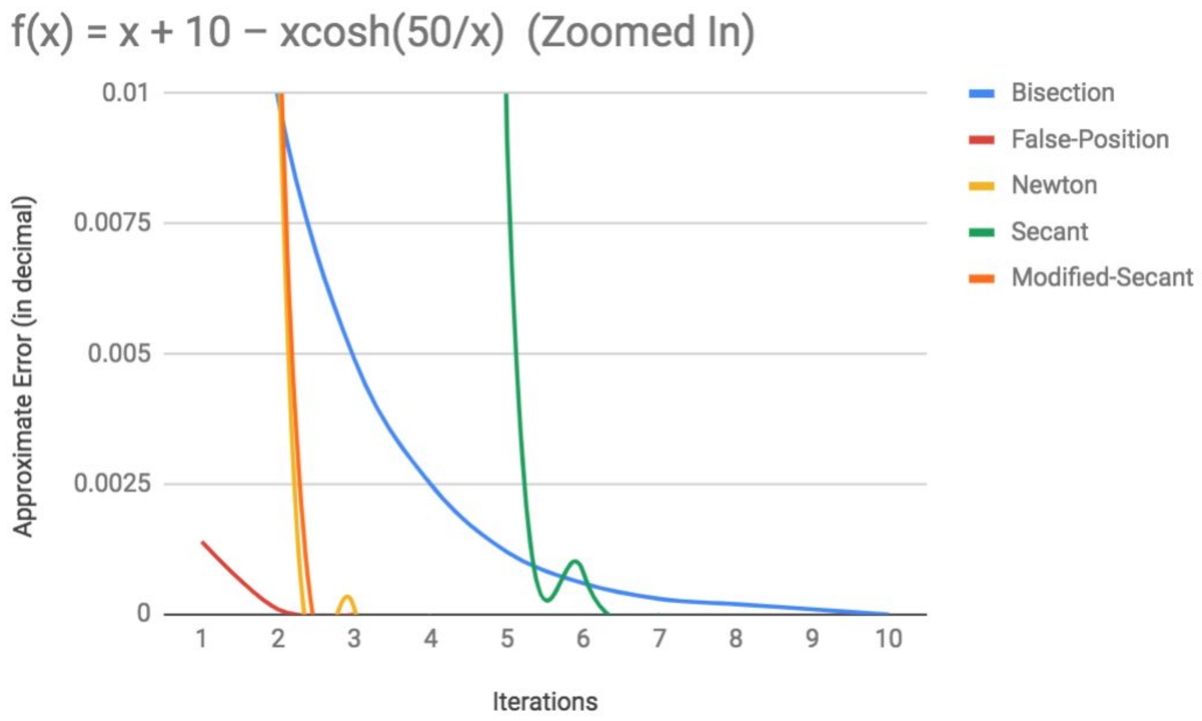

f(x) = 2x^3 – 11.7x^2 + 17.7x – 5 (Second Root)
Approximate Error (in decimal)
0.3
0.2
0.1
0
2
4
6
8
10
12
Iterations
Bisection
False-Position
Newton
Secant
Modified-Secant


f(x) = 2x^3 – 11.7x^2 + 17.7x – 5 (Thrid Root)
Approximate Error (in decimal)
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
Iterations
Bisection
False-Position
Newton
Secant
Modified-Secant

Function B


200
100
-100
-100
100
200
300
400
5


f(x) = x + 10 − xcosh(50/x)
0.5
0.4
0.3
0.2
0.1
0
Approximate Error (in decimal)
1
2
3
4
5
6
7
8
9
10
Iterations
Bisection
False-Position
Newton
Secant
Modified-Secant



Tables

Method: Bisection
Function: $2x^3 - 11.7x^2 + 17.7x - 5$
Root (found by program): 0.365234375

n	a	b	f(a)	f(b)	c	f(c)	Ea
1	0.0000	0.5000	-5.0000	1.1750	0.2500	-1.2750	
2	0.2500	0.5000	-1.2750	1.1750	0.3750	0.0977	0.3333
3	0.2500	0.3750	-1.2750	0.0977	0.3125	-0.5503	0.2000
4	0.3125	0.3750	-0.5503	0.0977	0.3438	-0.2169	0.0909
5	0.3438	0.3750	-0.2169	0.0977	0.3594	-0.0573	0.0435
6	0.3594	0.3750	-0.0573	0.0977	0.3672	0.0208	0.0213
7	0.3594	0.3672	-0.0573	0.0208	0.3633	-0.0181	0.0108
8	0.3633	0.3672	-0.0181	0.0208	0.3652	0.0014	0.0053

Method: Bisection
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 1.921875

```
 n        a          b          f(a)        f(b)        c          f(c)       Ea
---------------------------------------------------------------------------------------
 0     1.0000     2.0000     3.0000     -0.4000     1.5000     1.9750
 1     1.5000     2.0000     1.9750     -0.4000     1.7500     0.8625     0.1429
 2     1.7500     2.0000     0.8625     -0.4000     1.8750     0.2383     0.0667
 3     1.8750     2.0000     0.2383     -0.4000     1.9375    -0.0806     0.0323
 4     1.8750     1.9375     0.2383     -0.0806     1.9062     0.0791     0.0164
 5     1.9062     1.9375     0.0791     -0.0806     1.9219    -0.0007     0.0081
```

Method: Bisection
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 3.59375

```
 n        a          b          f(a)        f(b)        c          f(c)       Ea
---------------------------------------------------------------------------------------
 0     3.0000     4.0000    -3.2000      6.6000     3.5000    -0.6250
 1     3.5000     4.0000    -0.6250      6.6000     3.7500     2.3125     0.0667
 2     3.5000     3.7500    -0.6250      2.3125     3.6250     0.6867     0.0345
 3     3.5000     3.6250    -0.6250      0.6867     3.5625    -0.0069     0.0175
 4     3.5625     3.6250    -0.0069      0.6867     3.5938     0.3303     0.0087
```

Method: Bisection
Function: $x + 10 – x\cosh(50/x)$
Root (found by program): 126.25

```
 n        a            b            f(a)        f(b)        c             f(c)       Ea
---------------------------------------------------------------------------------------------
 0     120.0000     130.0000     -0.5682     0.2655     125.0000     -0.1340
 1     125.0000     130.0000     -0.1340     0.2655     127.5000      0.0698     0.0196
 2     125.0000     127.5000     -0.1340     0.0698     126.2500     -0.0311     0.0099
```

Method: False Position

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 0.3655871767763859

```
  n        a          b         f(a)        f(b)        c          f(c)        Ea
--------------------------------------------------------------------------------------
  0      0.0000     1.0000    -5.0000     3.0000     0.6250     1.9805
  1      0.0000     0.6250    -5.0000     1.9805     0.4477     0.7585     0.3961
  2      0.0000     0.4477    -5.0000     0.7585     0.3887     0.2298     0.1517
  3      0.0000     0.3887    -5.0000     0.2298     0.3716     0.0646     0.0460
  4      0.0000     0.3716    -5.0000     0.0646     0.3669     0.0178     0.0129
  5      0.0000     0.3669    -5.0000     0.0178     0.3656     0.0049     0.0036
```

Method: False Position

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 1.9217409206219755

```
  n        a          b         f(a)        f(b)        c          f(c)        Ea
--------------------------------------------------------------------------------------
  0      1.0000     2.0000     3.0000    -0.4000     1.8824     0.2009
  1      1.8824     2.0000     0.2009    -0.4000     1.9217     0.0003     0.0205
  2      1.9217     2.0000     0.0003    -0.4000     1.9217     0.0000     0.0000
```

Method: False Position

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 3.5551013438441474

```
  n        a          b         f(a)        f(b)        c          f(c)        Ea
--------------------------------------------------------------------------------------
  0      3.0000     4.0000    -3.2000     6.6000     3.3265    -1.9689
  1      3.3265     4.0000    -1.9689     6.6000     3.4813    -0.7959     0.0444
  2      3.4813     4.0000    -0.7959     6.6000     3.5371    -0.2671     0.0158
  3      3.5371     4.0000    -0.2671     6.6000     3.5551    -0.0840     0.0051
```

Method: False Position
Function: x + 10 – xcosh(50/x)
Root (found by program): 126.64240160719689

```
 n        a           b          f(a)       f(b)        c          f(c)       Ea
-------------------------------------------------------------------------------------
 0    120.0000    130.0000    -0.5682     0.2655    126.8156    0.0148
 1    120.0000    126.8156    -0.5682     0.0148    126.6424    0.0008    0.0014
```

Method: Newton-Raphson
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 0.36509824336223623

```
 n       Xn        Xn+1       f(Xn)      f'(Xn)       Ea
-----------------------------------------------------------------
 0     0.0000     0.2825    -5.0000    17.7000
 1     0.2825     0.3593    -0.8886    11.5686     0.2138
 2     0.3593     0.3651    -0.0581    10.0671     0.0158
 3     0.3651     0.3651    -0.0003     9.9571     0.0001
```

Method: Newton-Raphson
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 1.9217409317757095

```
 n       Xn        Xn+1       f(Xn)      f'(Xn)       Ea
----------------------------------------------------------------
 0     1.5000     2.0064     1.9750    -3.9000
 1     2.0064     1.9215    -0.4327    -5.0959     0.0442
 2     1.9215     1.9217     0.0012    -5.1101     0.0001
 3     1.9217     1.9217    -0.0000    -5.1102     0.0000
```

Method: Newton-Raphson
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 3.5631621003251914

```
  n        Xn        Xn+1        f(Xn)       f'(Xn)       Ea
-----------------------------------------------------------------------
  0      3.0000     5.1333     -3.2000      1.5000
  1      5.1333     4.2698     48.0901     55.6867      0.2023
  2      4.2698     3.7929     12.9562     27.1724      0.1257
  3      3.7929     3.5998      2.9476     15.2634      0.0536
  4      3.5998     3.5643      0.3980     11.2164      0.0100
```

Method: Newton-Raphson
Function: x + 10 – xcosh(50/x)
Root (found by program): 126.63243603998873

```
  n        Xn          Xn+1        f (Xn)       f'(Xn)        Ea
-----------------------------------------------------------------------
  0     140.0000    125.1707      0.9761      0.0658
  1     125.1707    126.6149     -0.1199      0.0830      0.0114
  2     126.6149    126.6324     -0.0014      0.0810      0.0001
  3     126.6324    126.6324     -0.0000      0.0810      0.0000
```

Method: Secant
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 0.3650982433622361

```
  n        Xn        Xn-1        Xn+1        f(Xn)      f(Xn-1)       Ea
-------------------------------------------------------------------------------
  0      0.5000     0.0000     0.4049      1.1750     -5.0000
  1      0.4049     0.5000     0.3592      0.3810      1.1750      0.2350
  2      0.3592     0.4049     0.3653     -0.0589      0.3810      0.1271
  3      0.3653     0.3592     0.3651      0.0023     -0.0589      0.0167
```

```
 4     0.3651     0.3653     0.3651     0.0000     0.0023     0.0006
 5     0.3651     0.3651     0.3651    -0.0000     0.0000     0.0000
```

Method: Secant
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 1.9217409317757086

```
 n       Xn        Xn-1       Xn+1       f(Xn)     f(Xn-1)      Ea
-------------------------------------------------------------------------------------
 0     1.5000     1.2500     2.1371     1.9750     2.7500
 1     2.1371     1.5000     1.9107    -1.0884     1.9750     0.2981
 2     1.9107     2.1371     1.9219     0.0562    -1.0884     0.1185
 3     1.9219     1.9107     1.9217    -0.0006     0.0562     0.0058
 4     1.9217     1.9219     1.9217     0.0000    -0.0006     0.0001
 5     1.9217     1.9217     1.9217     0.0000     0.0000     0.0000
```

Method: Secant
Function: $2x^3 – 11.7x^2 + 17.7x – 5$
Root (found by program): 3.5631608248620545

```
 n       Xn        Xn-1       Xn+1       f(Xn)     f(Xn-1)      Ea
-------------------------------------------------------------------------------------
 0     5.0000     4.0000     3.8081    41.0000     6.6000
 1     3.8081     5.0000     3.7078     3.1822    41.0000     0.3130
 2     3.7078     3.8081     3.5887     1.7277     3.1822     0.0270
 3     3.5887     3.7078     3.5662     0.2747     1.7277     0.0332
 4     3.5662     3.5887     3.5632     0.0320     0.2747     0.0063
 5     3.5632     3.5662     3.5632     0.0007     0.0320     0.0008
```

Method: Secant

Function: x + 10 – xcosh(50/x)

Root (found by program): 126.63243603998916

```
 n        Xn         Xn-1          Xn+1        f(Xn)     f(Xn-1)       Ea
--------------------------------------------------------------------------------------
 0    200.0000    100.0000     142.6328      3.7174     -2.7626
 1    142.6328    200.0000     117.0617      1.1461      3.7174     0.4022
 2    117.0617    142.6328     127.8874     -0.8415      1.1461     0.2184
 3    127.8874    117.0617     126.7310      0.1006     -0.8415     0.0847
 4    126.7310    127.8874     126.6314      0.0080      0.1006     0.0091
 5    126.6314    126.7310     126.6324     -0.0001      0.0080     0.0008
```

Method: Modified Secant

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 0.36509823932946267

```
 n        Xn        d*Xn       Xn+1          f(Xn)       f(Xn+(dXn))        Ea
-----------------------------------------------------------------------------------------------
 0      1.0000     0.0100    -11.3355        3.0000        3.0024
 1    -11.3355    -0.1134     -6.9873    -4622.1182     -4742.6124      0.6223
 2     -6.9873    -0.0699     -4.0951    -1382.1475     -1415.5388      0.7063
 3     -4.0951    -0.0410     -2.1889     -411.0295      -419.8597      0.8708
 4     -2.1889    -0.0219     -0.9591     -120.7745      -122.9242      1.2821
 5     -0.9591    -0.0096     -0.2063      -34.5047       -34.9442      3.6500
 6     -0.2063    -0.0021      0.1956       -9.1663        -9.2133      2.0545
 7      0.1956     0.0020      0.3434       -1.9704        -1.9443      0.4304
 8      0.3434     0.0034      0.3647       -0.2204        -0.1849      0.0585
 9      0.3647     0.0036      0.3651       -0.0037         0.0326      0.0010
```

Method: Modified Secant

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 1.9217409317756742

```
 n        Xn        d*Xn        Xn+1         f(Xn)      f(Xn+(dXn))       Ea
-------------------------------------------------------------------------------------
 0      1.5000     0.0150      2.0013       1.9750        1.9159
 1      2.0013     0.0200      1.9214      -0.4064       -0.5083        0.0415
 2      1.9214     0.0192      1.9217       0.0015       -0.0967        0.0002
 3      1.9217     0.0192      1.9217       0.0000       -0.0983        0.0000
```

Method: Modified Secant

Function: $2x^3 - 11.7x^2 + 17.7x - 5$

Root (found by program): 3.5631608265524632

```
 n        Xn        d*Xn        Xn+1         f(Xn)      f(Xn+(dXn))       Ea
-------------------------------------------------------------------------------------
 0      3.0000     0.0300      4.8926      -3.2000       -3.1493
 1      4.8926     0.0489      4.1429      35.7632        38.0973       0.1809
 2      4.1429     0.0414      3.7423       9.7305       10.7367        0.1071
 3      3.7423     0.0374      3.5911       2.2031        2.7481        0.0421
 4      3.5911     0.0359      3.5647       0.3004        0.7098        0.0074
 5      3.5647     0.0356      3.5632       0.0162        0.4039        0.0004
 6      3.5632     0.0356      3.5632       0.0005        0.3870        0.0000
 7      3.5632     0.0356      3.5632       0.0000        0.3865        0.0000
```

Method: Modified Secant

Function: x + 10 – xcosh(50/x)

Root (found by program): 126.63243604536649

```
 n        Xn          d*Xn        Xn+1          f(Xn)      f(Xn+(dXn))       Ea
-------------------------------------------------------------------------------------
 0     140.0000      1.4000     125.0177        0.9761        1.0673
 1     125.0177      1.2502     126.6276       -0.1326       -0.0296        0.0127
 2     126.6276      1.2663     126.6325       -0.0004        0.1012        0.0000
```