Project 1: Root-Finding Methods

CS 3010 - Numerical Methods

Summer 2020

Instructor: Dr. Amar Raheja

Computer Science Department

California State Polytechnic University, Pomona

Submitted On: 15-JUL-2020

Submitted By:

Damian Ugalde

Instructions:

Write programs for all the methods (Bisection, Newton-Raphson, Secant, False-Position and Modified Secant) for locating roots. Make sure that you have clever checks in your program to be warned and stop if you have a divergent solution or stop if the solution is very slowly convergent after a maximum number of iterations.

Use your programs to find the roots of the following functions and plot three graphs for each one of the functions. The first graph should plot the given function. Use any plotting software. This will give you the idea of the root/s of the function. The first graph should show the true percent relative error (y –axis) vs. the number of iterations (x-axis) for all the methods (only if the true root is given to you in the problem) and the second graph should show a similar plot but using the approximate percent error instead of the true percent relative error.

(a) $f(x) = 2x^3 - 11.7x^2 + 17.7x - 5$

This function has 3 positive roots, all of which lie between 0 and 4. Find the roots. Implement the methods until ea < 1%. Let the maximum iterations be 100. For the modified secant method, use δ = 0.01. Plot the graphs for the approximate % relative error for different roots. Each graph should have 5 error curves, one for each method.

(b) $f(x) = x + 10 - x\cosh\left(\frac{50}{x}\right)$

For this function, plot the % approximate relative error for each method similar to the part (a). Use δ = 0.01 for modified secant method. Figure out the initial points for the other methods. As a hint, the root lies in the interval [120, 130]

Write a report that shows the print outs of all the tables for each method and the graphs as well. Talk about the starting points and convergence to the root for the different methods used.

Point out any interesting/strange behaviors you might observe while using these numerical methods. Comment on the data types used to calculate the roots in your program.

Please note: You will have a plot for the function itself for both part (a) and (b) and 3 plots (one for each root and each plot with 5 curves) for part (a) and 1 plot(for the single root) for part (b).

Points distribution: 10% for each root finding method, 25% for the write-up and 25% for the plots

Program Technique:

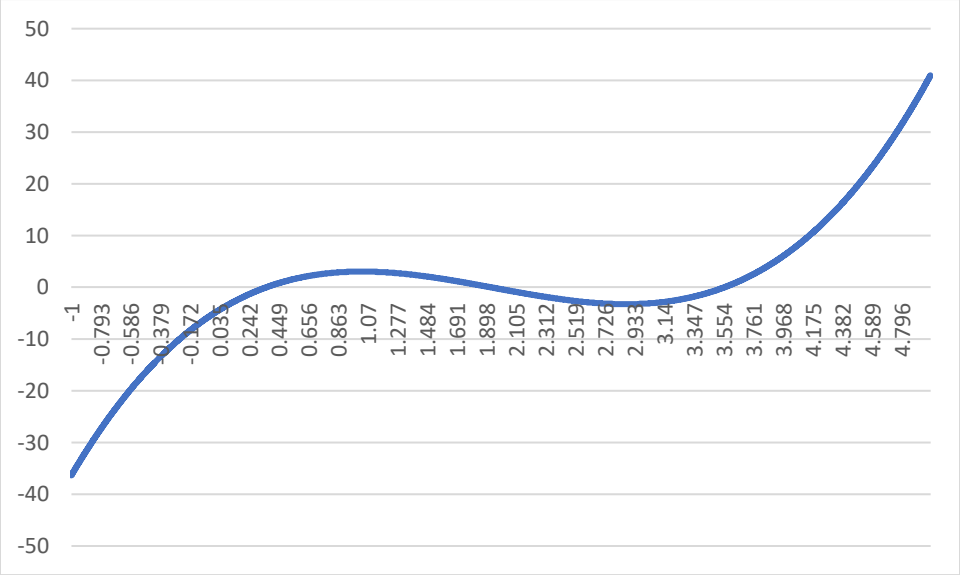
The program first separates the function by selecting an interval that contains all of the roots. This interval is partitioned into smaller intervals that check if there is a sign change for the value of the function between this small interval. If it does detect a sign change, it will call the five different root-finding techniques. These techniques will then find the root.

This technique gives surprising precision with few iterations. This occurs because the domain where the root lies is very small, so there are less values to check. The initial pass through the large domain of the function does take some time, but it is beneficial, because it also helps determine the number of roots in a function.

Graph A

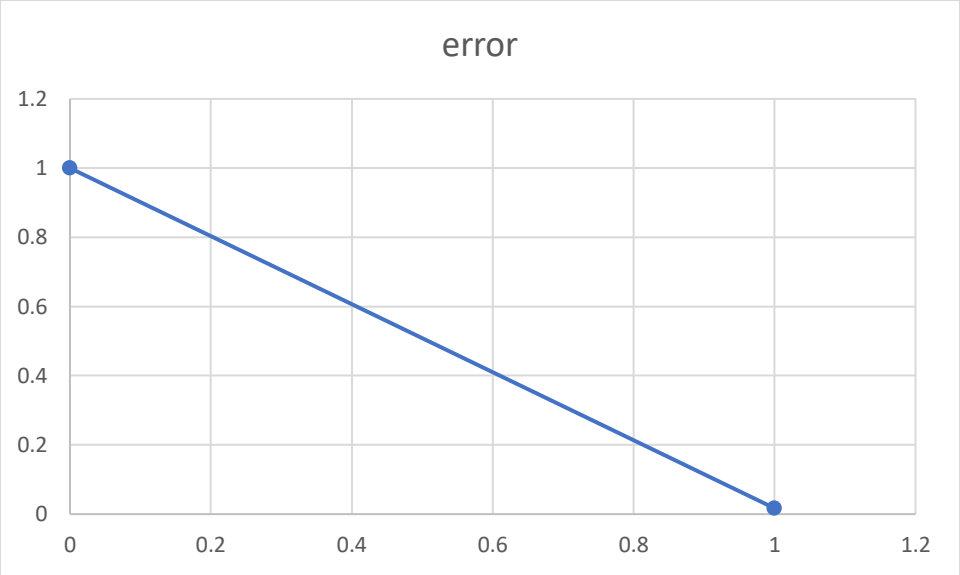
For each method in Graph A, there are three graphs shown. These graphs represent the error to iterations for each of the roots (3 of them).

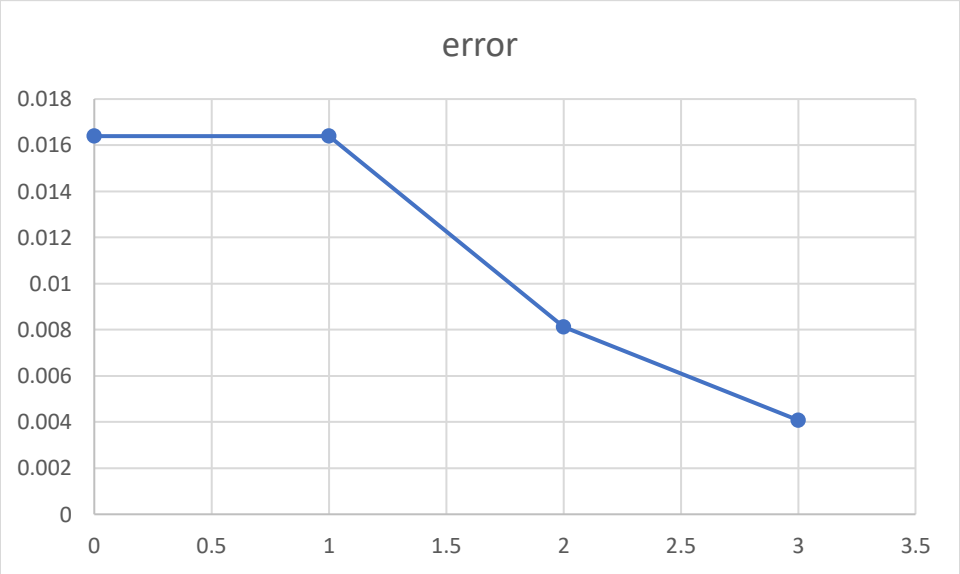
First, here is the graph as it was plotted by the program. We can see 3 occasions where it crosses the x-axis.

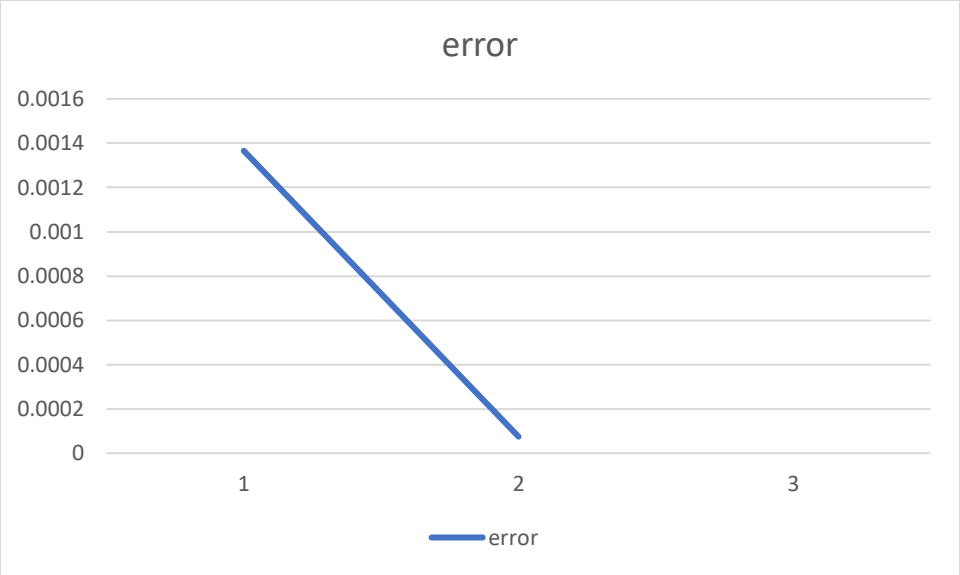


Bisection:

The bisection method takes about three iterations to find the root. It is supposed to be the least efficient of the methods by halving the distance each iteration. The reason that this method takes so short is because we reduce the domain where it should look for the root.


error
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2

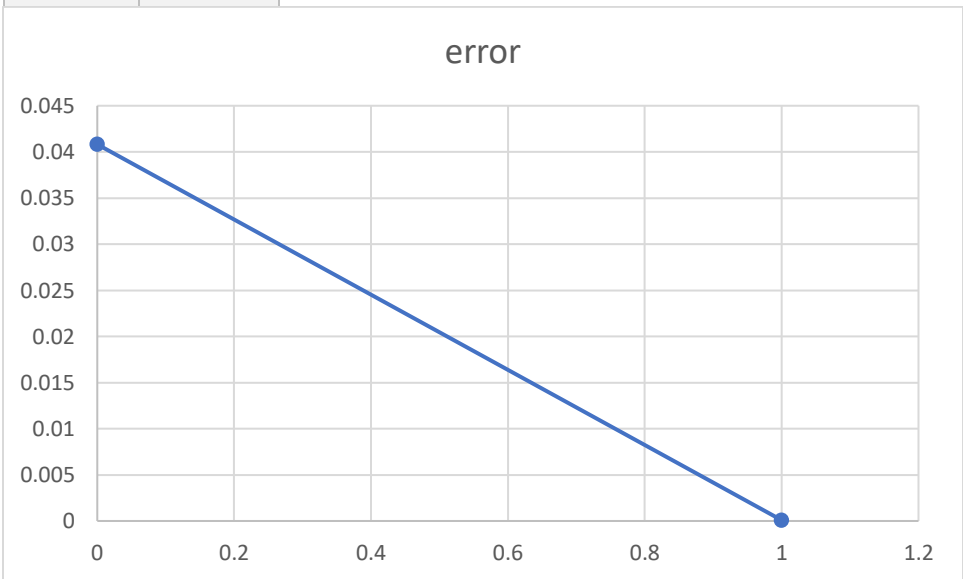

error
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
0.5
1
1.5
2
2.5
3
3.5

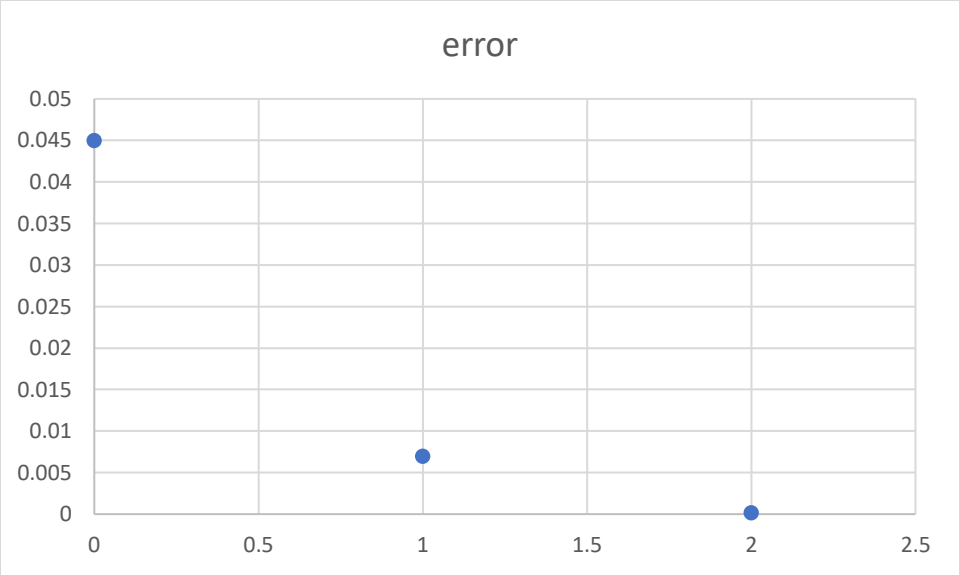


Newton-Raphson:

Newton-Raphson also takes a small interval to find since we are using a small range. The advantage of using this method is if the derivative is easy to calculate. If the derivative is hard to find, the modified secant is better as it is used as an approximation of the derivative. The issue with this method is if the original value of x has a derivative close to 0, the next point can't be found, since the tangent line won't reach the x-axis quickly enough and will alternate and not converge.

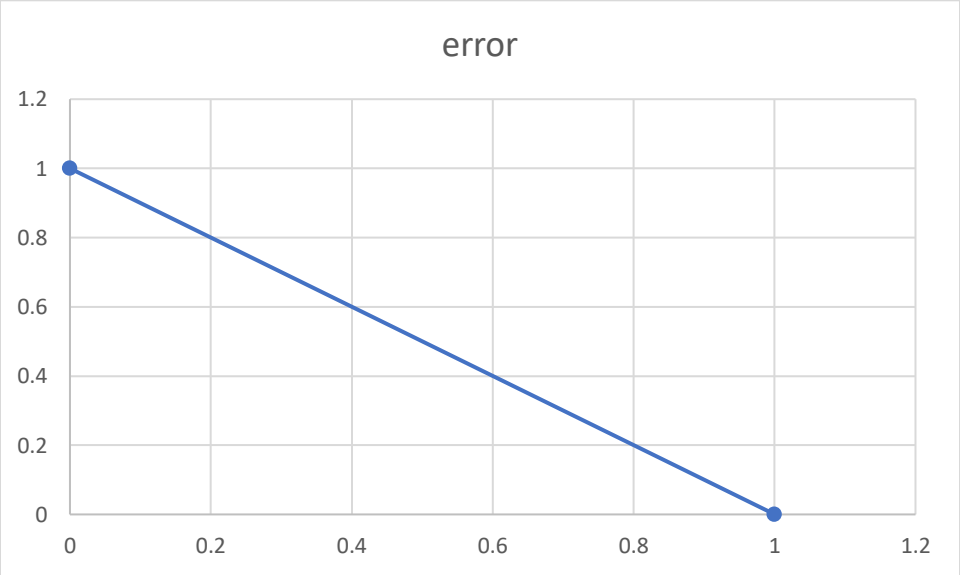
iteration	error
0	0.027388

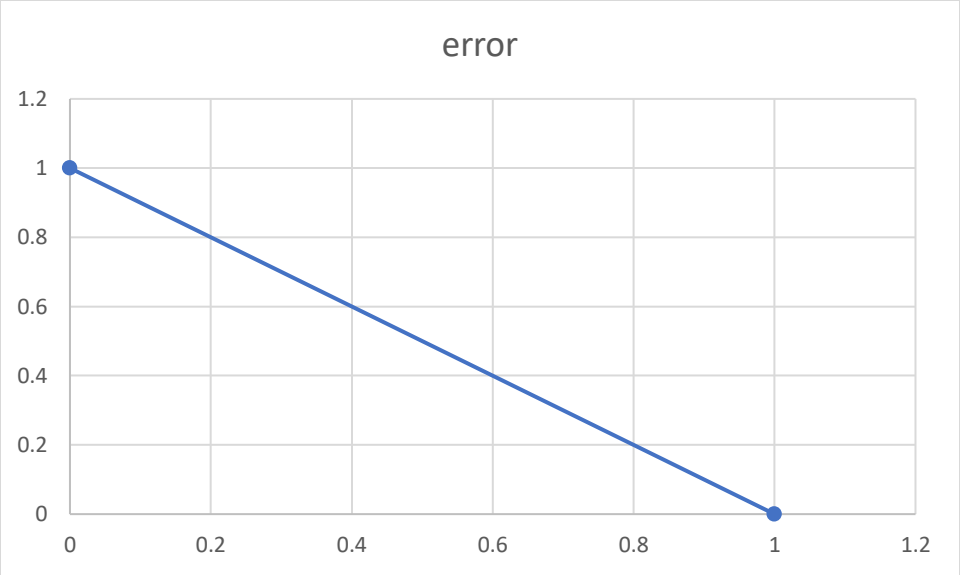


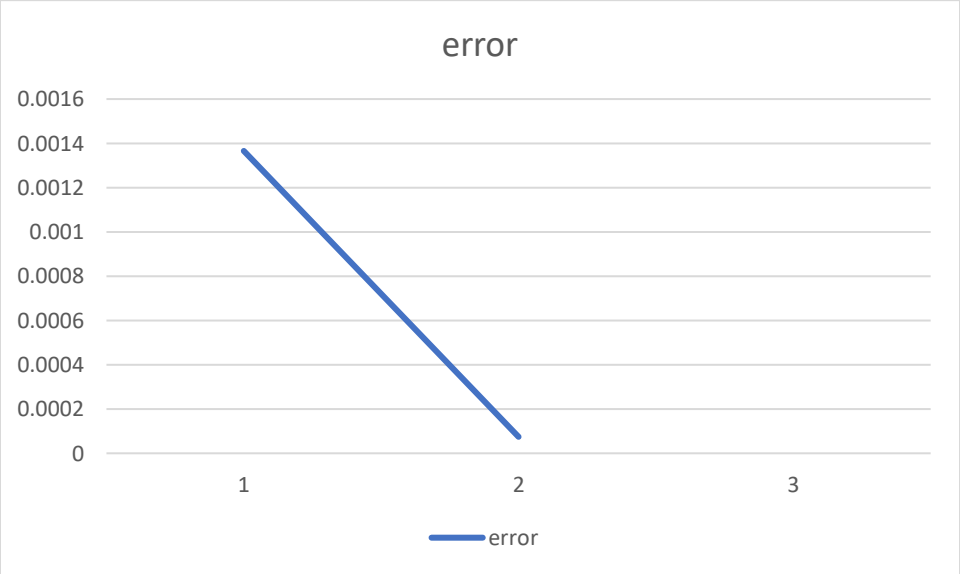


Secant:

Secant method was very efficient with a small range of values. It can find the root very quickly without having to use the derivative like in Newton-Raphson.

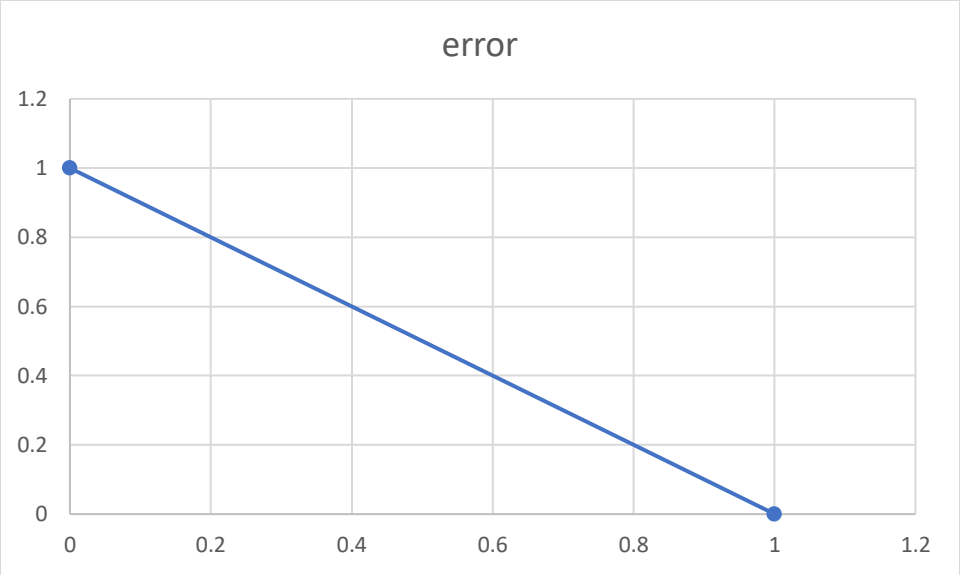


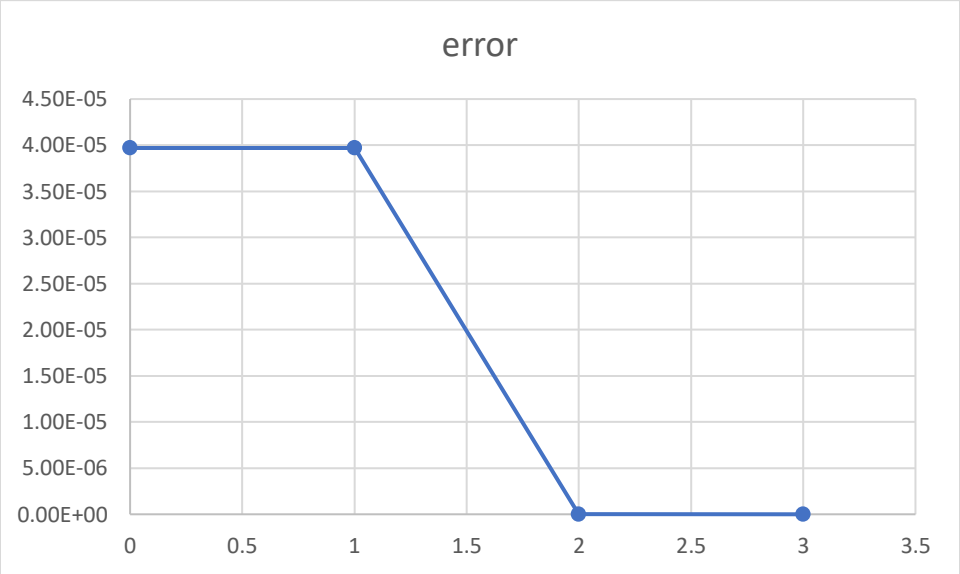


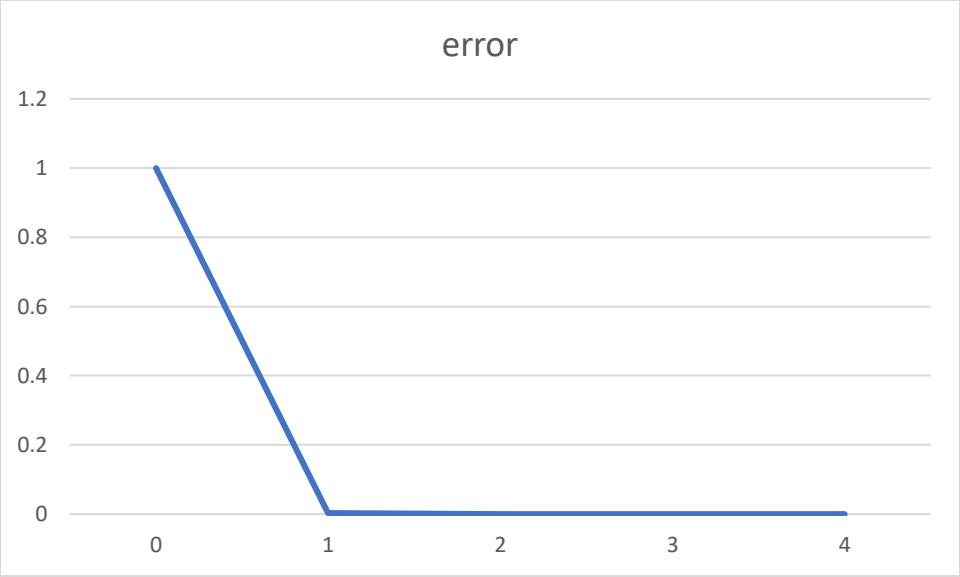


False Position:

False position is good to estimate the derivative for functions that are hard to find the derivative for.

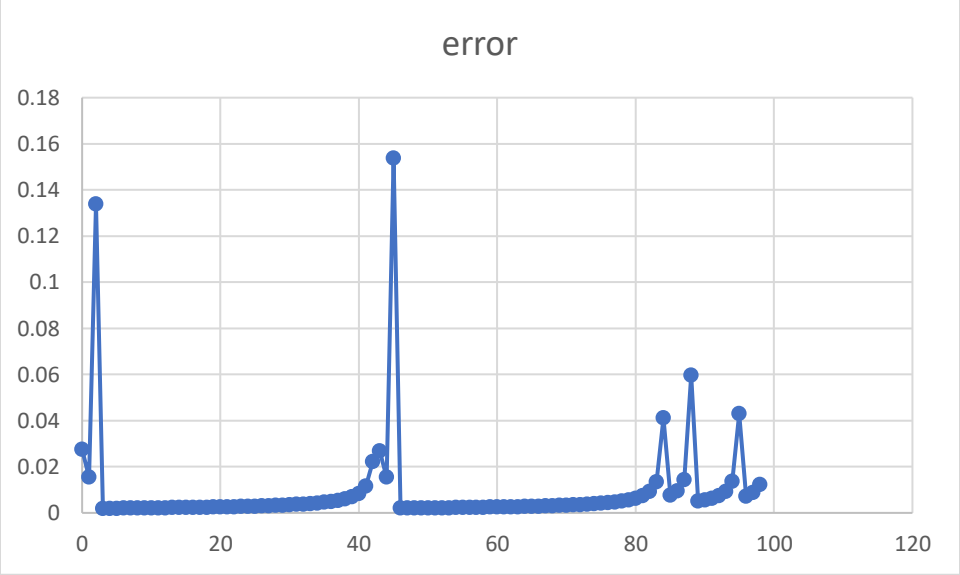

error
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2

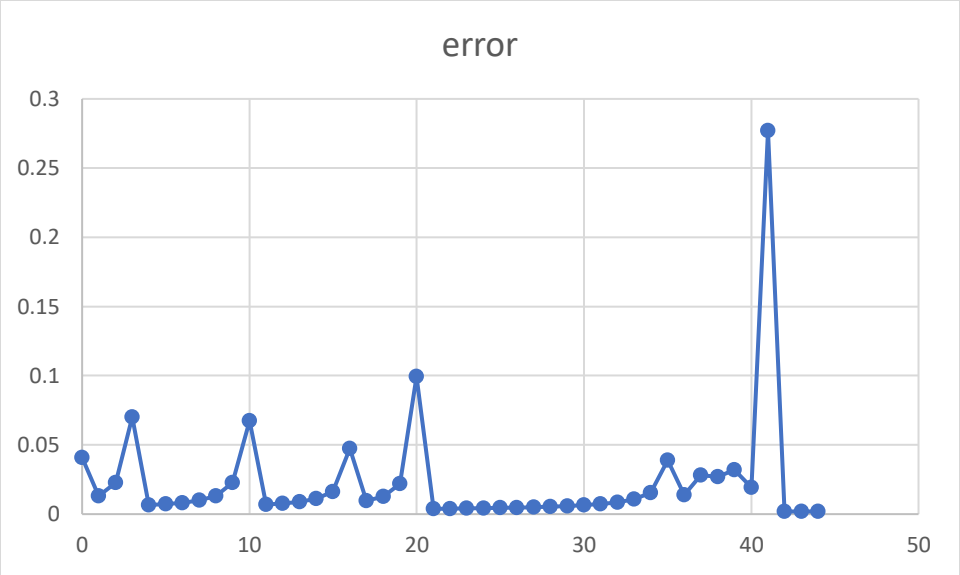

error
4.50E-05
4.00E-05
3.50E-05
3.00E-05
2.50E-05
2.00E-05
1.50E-05
1.00E-05
5.00E-06
0.00E+00
0
0.5
1
1.5
2
2.5
3
3.5

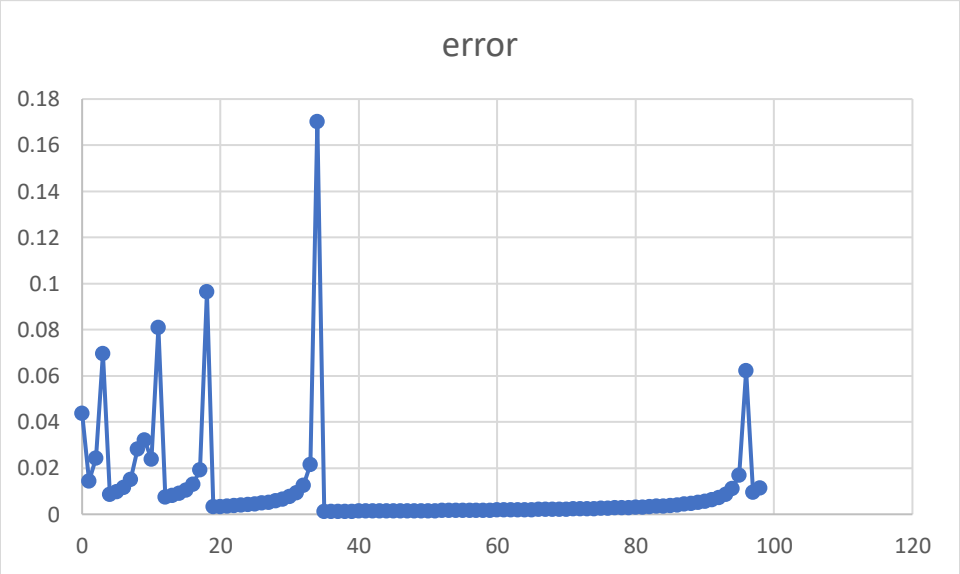


Modified Secant:

Modified Secant had an interesting case. As it alternates and tries to find a solution, it will converge to a solution. However, if the program keeps running, it will move away from the root with the next guess. In this case, a check is placed when the solution is converging. For illustration, I left it running to see what happens, but the program does have a check for convergence and will stop at around iteration 10 for the first example.

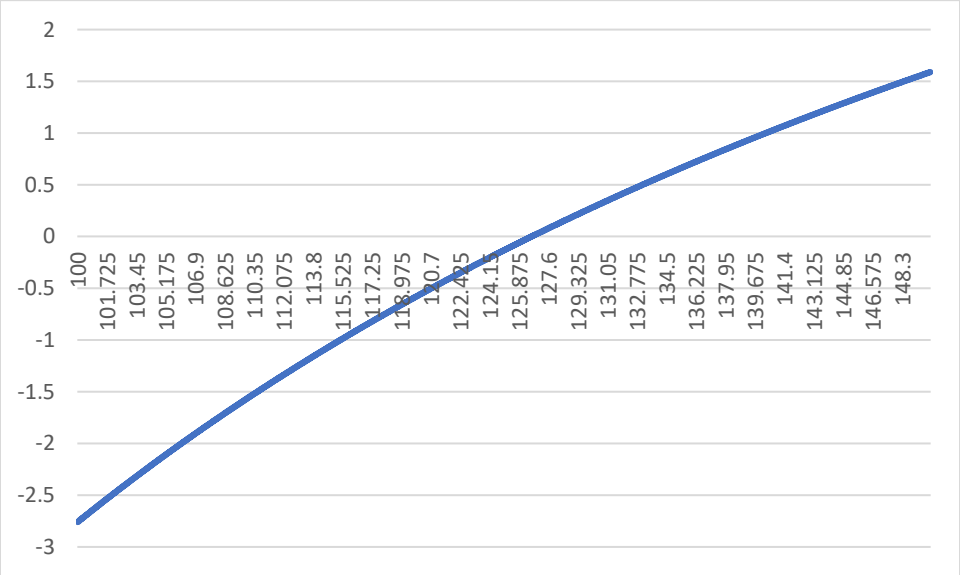



error
0.3
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50


error
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
20
40
60
80
100
120

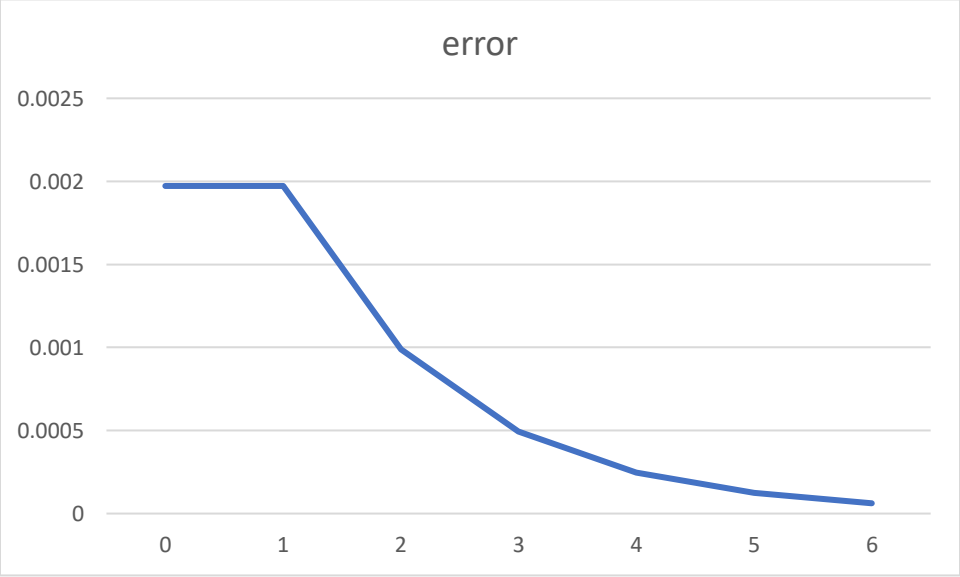
Graph B

For this graph, only the portion close to the root is shown, as it is the section we are most interested in.



Bisection:

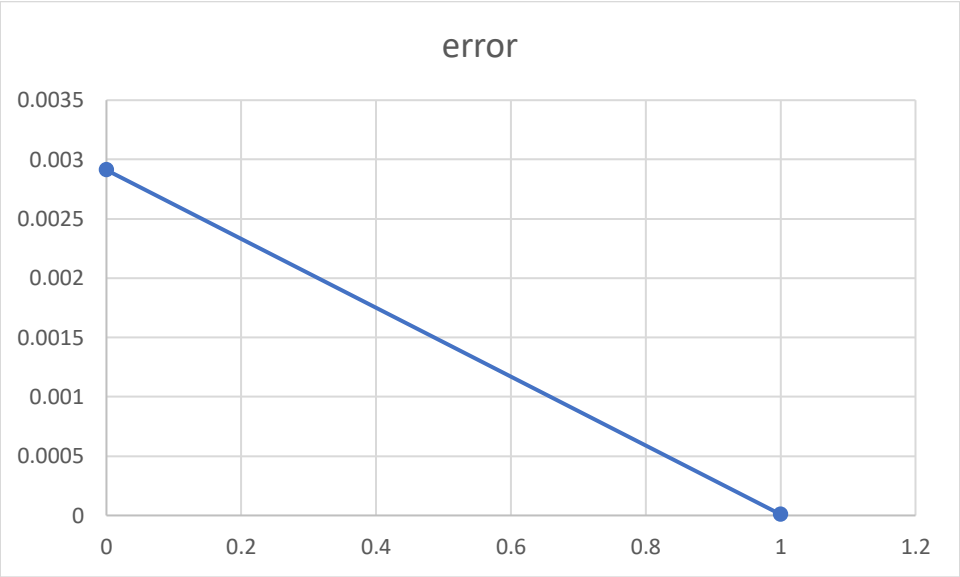
The bisection method seems to approach a solution in an exponential form. This happens because the domain that we are using is getting cut in half with each iteration.



Newton-Raphson:

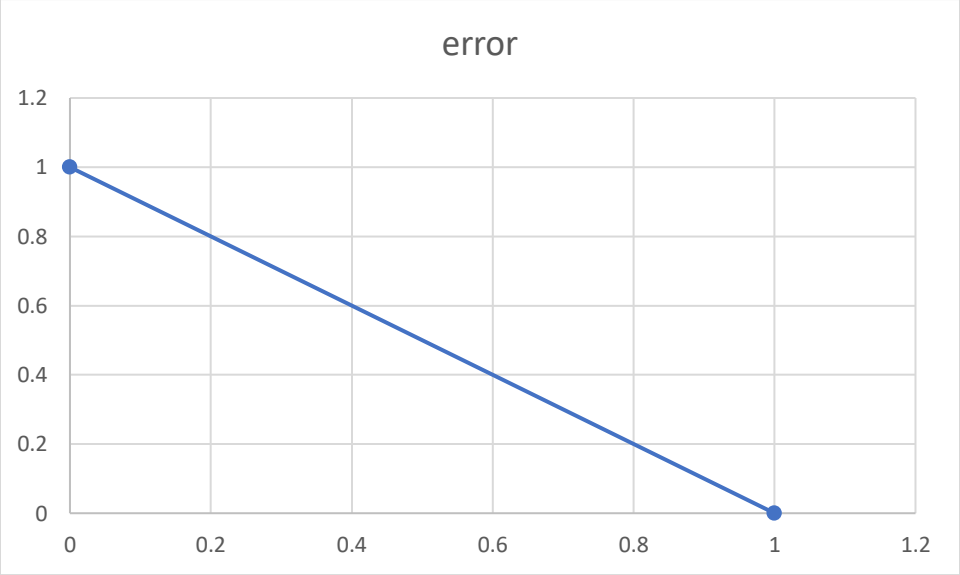
Newton-Raphson is surprisingly quick, because the derivative is easy to evaluate and because the interval given where the function lies is very small [126,127]. But this again

could have problems if the derivative is close to zero, as the method won't be able to find the next value for the root approximation.



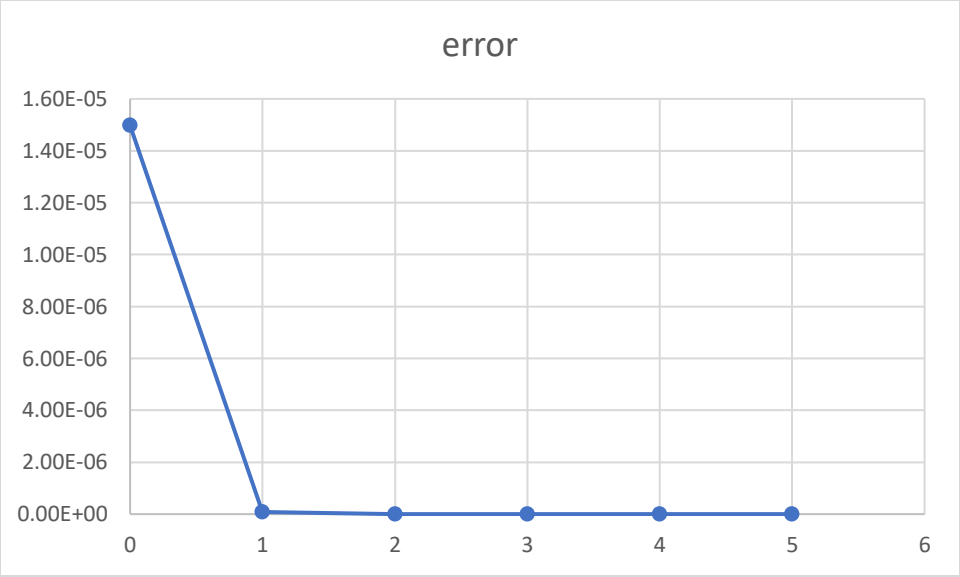
Secant:

Secant also approaches quickly because of the short range. If the range is expanded, the function could keep alternating between values that completely skip the root and won't converge.



False Position:

For false position, it found a solution quickly and was stuck trying to find a very precise solution.



Modified Secant:

Again as in graph A, the function converges quickly to a solution, but if the method is not stopped, it will vary wildly and will just converge back to the solution. The best method is to stop it once the solution converges to a value. In this case, I left it running to illustrate, but it would probably stop at around iteration 20.