Justin Ordonez

CS 301-01

Project 1

Bisection method for function 1 started at a=0 and b=1 to approximate the first root = 0.365, and it did so in about 6 iterations. For root = 1.922, the initial values were a=1 and b=2, and the convergence still took 6 iterations. For root = 3.563, the initial values were a= 3 and b=4, and the convergence took 6 iterations again. 6 iterations were taken for convergence for function 2 as well. The approximate error for the function was kept the same as the pseudocode given by the book. However, because of this, I observed that the Bisection method would always have 6 iterations for this error threshold, and I did not find cases outside of this that could indicate that this was coincidence.

False position for function 1, the approximation for root = 0.365 started at a=0 and b= 1 as well, and converged in about 5 iterations. For function 1, root = 1.922, the approximation had initial values of a= 1 and b=2, and it converged in 3 iterations. In the third iteration, the error for the approximation was 0. For function 1, root = 3.563,the approximation started at a = 3 and b=4, and converged in 3 iterations. False position converged more quickly than bisection on average, and the convergence to root 1.922 was very quick and precise, since one of the given starting points was b=2, which is only .088 away from the root. For function 2, the initial values were a=0 and b=1, and convergence was confirmed after 2 iterations.

Newton-Raphson Method took 9 iterations to converge for function 1 root = .365 with starting point x=1. For function 1 root = 1.922, the starting point was x=2, and only 2 iterations were taken for convergence, with an error of 0. This is probably also due to the close proximity of the starting point and the root, as it was for false-position method. For function 1 root = 3.563, 6 iterations were taken for convergence, with sixth iteration having an error of 0.001. For function 2, Newton-Raphson Method converged in 3 iterations with a starting point of x=1.

Secant method for function 1 with root = 0.365 and starting points a=0 and b=1 converged after 6 iterations. For function 1 with root = 1.922 with starting points a= 1 and b=2, convergence was achieved after 4 iterations. Technically. Convergence was achieved after 3 iterations, but the pseudocode counts the initial b value as an iteration. This is true for the rest of the Secant calculations as well. Function 1 with root = .3563 with starting points a=3 and b=4 took 6 iterations to converge. With starting points of a = 0 and b = 1 for Function 2, Secant method took 2 iterations to converge, with a low true approximate percent error of .0002.

Modified secant method for function 1 with root = 0.365 had initial values of x = 1 and delta = 0.01, and took 9 iterations for convergence. Delta was never changed for the required inputs, so I will refrain from repeatedly listing it. At root = 1.922, the initial value used to calculate the root was x= 1, and convergence was reached after two iterations. This outcome is very similar to that of Newton-Raphson method under these same conditions, with each first iteration differing by only 0.001. For root = 3.563 for function 1, the starting point was x = 3, and 5 iterations were taken for convergence. Function 2 at root = 0.56714329 took 2 iterations, and had the exact same graph as normal secant method.

As for any strange behaviors for certain methods, I found that the modified secant method was undefined when x=0 for function 1 after the first iteration. This is probably because the function had a division by 0. Every iteration following iteration 1 read out as NaN. When I changed the input to x=1 instead, the function behaved as predicted. If tested across a range from a = 0 to b = 4, secant method actually returns the middle root of 1.922, whereas Bisection and False-Position return the higher root of 3.563.

As a side note:

The program outputs of each method show values up to 3 decimal places rounded, but internally, these are not truncated values. They use double data types, though I’m sure for these purposes, floats would have worked fine.