Project 2 Write-Up

The intent of this project was to effectively write a Newton algorithm for determining the root of a given function and to compare the number of iterations needed to reach a certain tolerance of the root between the Bisection and Newton root-finding methods.

Phase 1 of this project was purely devoted to writing the Newton function to solve the root of the function g(x) = (x – 3)^3 = 0. In the test\_newton.m script, I passed 4 variables to the main newton.m function: the initial guess, the function g, the derivative of g (dgdx), and the tolerance. The initial guess for the phase 1 function was 2 and the tolerance was 10e^-5. In the core Newton method, I initially found the values of f and f prime for the initial value 2. Then I began a for-loop with a max iteration count of 100, which should never be reached, and set a variable x equal to the value of the first iteration of the Newton method. X had an initial value of 2.3333 with a g(x) value of -0.2963. I then attained the updated values of f and f prime by using the newly found x as the x-value for these functions. Then I displayed the iteration count, the x value, and the g(x) value for the variable x. Lastly, before the loop repeated, I checked to see if the absolute value of g(x) was less than the tolerance. If it was, then it broke out of the for-loop and ended the method; otherwise it repeated another iteration. The function resulted in 8 total iterations under the tolerance 10e^-5, with a final x value of 2.96098156 and an ending g(x) value of -5.9403e^-5.

Phase 2 was the application process for the written Newton method function, but with a different function, tolerance, and initial guess. I computed the root T of the following function, initially with the Bisection method, and then with the Newton method, under 3 different tolerances (10e^-4, 10e^-7, 10e^-10).

f(x) = .08(pi)(5.67e^-8)T^4 + 2(pi)T – 2085.938

With a derivative of:

dfdx(x) = .32(pi)( 5.67e^-8)T^3 + 2(pi)

I first computed the root of f(x) with the Bisection method under the 3 different tolerances. With each decreasing tolerance, the iterations were 18, 29, and 39, with respective approximate roots of 310.819626, 310.819521, and 310.819521. The Newton method, on the other hand, took 2, 3, and 3 iterations under the 3 different tolerances with respective approximate roots of 310.819535, 310.819521, and 310.819521. In the end, both methods found very similar values, but the Newton method completed the same task with much fewer iterations, and therefore it is proven that the Newton method converges much faster than the Bisection method. The Bisection method algorithm included the test\_bisect.m, bisection.m, and f.m scripts. The bisection.m function accepted parameters of the given function f, lower bound a, upper bound b, max iterations, and the tolerance. It first initializes the average, x, of a and b, and checks to see if the tolerance is an ample size. It then checks to see if the average x is positive or negative, and if it is positive, it sets the upper bound b to x and repeats in a for-loop. If x is negative, then it sets the lower bound a to x, and repeats in a for-loop. The for loop continues until either the max iterations have been reached, or if the tolerance has been reached.

The Newton method follows the same algorithm as in phase 1, except it accepts f.m and dfdx.m as the 2 functions, and it began each time with an initial guess of 300.

It is easy to see that the Newton method converges much faster than the Bisection method, and the nonlinear tolerance is a large factor in this. The smaller the tolerance gets, the wider the gap is between the number of iterations between the Bisection and Newton methods.

In my opinion, the main difference between the two approaches that separated them by so many iterations is that the Newton method takes a much more intuitive approach to finding the root of a problem than the bisection method. The bisection method has no strategy other than to continuously cut itself in half until it finds a root close enough to the given tolerance. It has no derivative or slope pointing it in the right direction and by how far it should go in that direction. The Newton method does take advantage of the function’s derivative, and can therefore make a safe assumption about the direction of the closest root. This is why the Newton method converges so much faster than the Bisection method. There are problems, however, with the Newton method; it has the possibility of failing if it is given a poor initial guess. Even though this is possible though, the Newton method is always preferable to the Bisection method whenever it is able to be applied because it has such a faster convergence rate.