Mauricio Deguchi
UID: 405582562
CEE/MAE 20
November 2, 2021

Homework 5

1 Finding roots of nonlinear equations: Bisection Method

1.1 Introduction

This script uses the bisection method to find the root of a nonlinear function. The bisections method uses an algorithm which sections off our interval, creating a smaller and smaller interval as the algorithm repeats.

1.2 Models and Methods

First, we established our function which we want to evaluate in a dedicated function script. This function will contain only a single input and single output, just as our nonlinear equation would. We then define a second function which captures our bisection algorithm. This algorithm takes up 4 inputs: our desired function, two bounding values for our interval, and a tolerance for our approximation of the root. In this function, we implement our algorithm depicted in the pseudocode which was provided in the problem statement. Aside from the pseudocode provided, we must also define our output variables where xRoot is our value p which corresponds to the final x value from the while loop (the root of our function) and iter which is equal to k.

1.3 Calculations and Results

Once we have set up our 2 functions, we can run our biSection function to find our root and the number of iterations:

```
[xRoot, numIter] = biSection(@fun1, 0, 1, 10e-5)
```

This returns our root which is checked by using the fzero function on our evaluated function fun1. The algorithm in our biSection function takes the average of our boundary values and finds the corresponding value of our function. Depending on the sign of our value, the algorithm then redefines the boundary in which we search and the algorithm is repeated until our value is within the tolerance value to zero. In this example, our root evaluated to 0.2019 which is very close to the true value of 0.2020.

1.4 Discussion

The biggest challenge during this problem was understanding how to implement a function as an input for another function. Aside from that, this problem was quite interesting and helped introduce me to functions and how to use them across scripts. The premise of the problem was also quite interesting, the use of the bisection method introduced me to a search mechanism which essentially halves the amount of points we must search for each iteration.

2 Finding roots of nonlinear equations: Fixed Point Iteration Method

2.1 Introduction

Similar to the prior problem, we want to use an algorithm to approximate the roots of nonlinear functions. Here, we use another method, the fixed-point iteration method, which uses a guess value as its starting point to approximate a nearby root.

2.2 Models and Methods

Just like the previous problem, here we must define two functions. The first function will establish the nonlinear equation we want to analyze the roots of. This function is set up such that it requires a single input for any value of x and it will output a single value corresponding to the value of the function at that point.

Meanwhile, our second defined function implements the fixed-point iteration method. This function takes in 4 inputs (our desired function, a guess value, a tolerance, and a maximum number of iterations) and returns 2 outputs (xStar which is the value where our function intersects the line y = x, and xRoot which is the x position of the root of our function).

2.3 Calculations and Results

The algorithm first takes in our function and guess value; then, the loop begins by evaluating our guess value through our modified function g(x) and ensuring that this our guess point is not sufficiently close enough to the intersection point at y = x. If this does not occur, the cycle begins and we then redefine our guess x as the value of our output for our initial guess x. The cycle then restarts with the added condition that we must not have exceeded the maximum number of iterations specified. The loop runs until we find our root out we exceed the number of iterations. With the conditions provided in part ii, we obtain:

xStar =

1.498700925407042

xRoot =

1.498695355521898

With the conditions specified in section iii, we must define a new function ‘fun3’ in the same way we defined ‘fun2’ and then we can run our fixed-point algorithm with the new conditions. When this is executed, we obtain the following when x0 = 3:

xStar =

4.683823131060242

xRoot =

1.000815952260083

And when x0 = 0:

xStar =

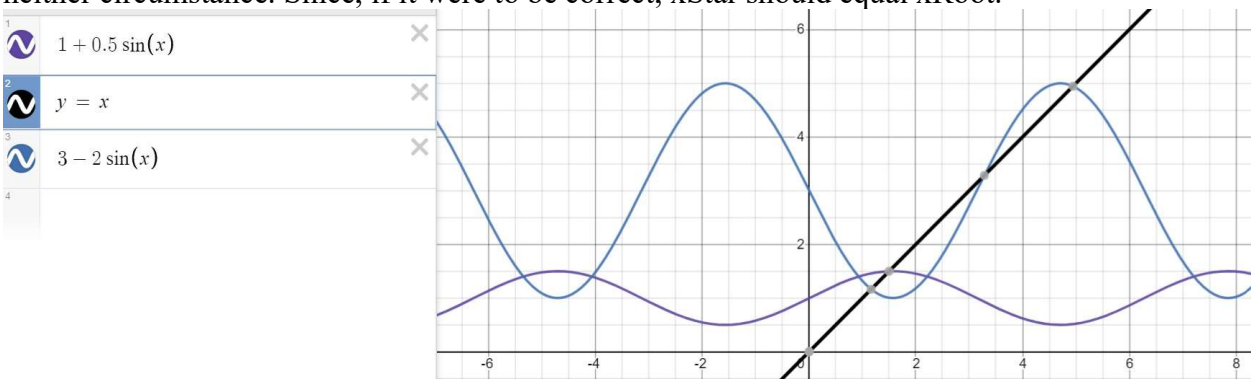
1.000815952260083

xRoot =

4.683823131060242

2.4 Discussion

From the last two results presented, we can see that this simulation result is not correct under neither circumstance. Since, if it were to be correct, xStar should equal xRoot.



By plotting the graphs of ‘fun2’ (which ran smoothly through the simulation), ‘fun3’, and y = x, we can see one key distinction between ‘fun2’ and ‘fun3’ (purple and blue respectively): fun3 intersects the y = x line on multiple occasions while fun2 only intersects the y = x line once, at the root. This is likely the source of error when using our fixed-point method; since our method depends on this intersection, when it attempts to find the intersection, the algorithm could be looking for to estimate between the various points of intersection and deem one as correct.