Adrian Loekman
UID: 105785820
CEE/MAE M20
October 22, 2021

HOMEWORK 5

1. Finding Roots of Nonlinear Equations
   a. Bisection Method
      i. Introduction

The goal of this code is to generate a set of functions and a main script to calculate the root of a nonlinear equation on a given equation. This part specifically uses the bisection method so solve for the root.

      ii. Model and Methods

The function that will be used in this problem is $f(x) = x^3 - sin(x) + 4x + 1 = 0$. So, a function called f1.m is created it takes an input, x, and an output, fx. The input will be any real number and fx is the output by substituting x into the function.

```
fx = x.^3 - sin(x) - 4*x + 1;
```

The value of fx could either be an array or a single value depending on the characteristics of x.

Next, another function is introduced. The function is called bisection.m. the purpose of this function is to create a root finding algorithm using bisection method. This function is created separate from the main script because it could get very long and unnecessary to be put in the main script because it promotes functionality and usage. The inputs of the function are the function f, left boundary a, right boundary b, and tolerance tol. There will be two outputs in this function, thus square brackets '[ ]' are used. The outputs are the xRoot itself and numIter, the number of iterations taken to get to the root.

```
nMax = floor((log(e0) - log(tol))/log(2)); % maximum
iterations
k = 1;
```

The first part of the code is to define the maximum iterations of the code. This can be obtained by using the natural log of epsilon, $\epsilon_0 = b - a$ subtracted by the natural log of the tolerance and divide the product by the natural log of 2. Then, start the iteration from 1.
The while loop has a condition that as long as the iteration has not reached the maximum iterations allowed, the loop will proceed. This will proceed the bisection where the root, p, is the addition of a + b divided by 2, or simply the average point from the root.

```
if f(p) == 0 || e0/2 < tol % p is the root if therse
conditions are true
        disp(p);
```

```
    end
```

If we evaluate the function and get a 0 or the value of epsilon/2 is less than the tolerance, then it means that we've reached the root of the equation, and the iteration stops. Otherwise, the iteration proceeds and checks the sign of f(p) and f(a). If they have the same sign (i.e. both are negative or positive) then, p is set as the new left boundary (a = p), otherwise p is set as the new right boundary (b = p). Then the iteration will go on until the max iteration is met.

```
if (f(p)*f(a)) > 0
        a = p;
    else
        b = p;
end
```

Using f(p) * f(a) > 0 is to check the signs of the function. Since we're not considering the values of the function itself, we can use it greater than zero. We use this because we know that if the same signs are multiplied, they will have a positive result.
Lastly in this function, the outputs, xRoot and NumIter, will be determined by p and k respectively.

The next and last code now is the main script. Since we know the left boundary, right boundary, and tolerance, we can set those as parameters. Then we can proceed with the function call. The function call is quite simple,

```
[xRoot, numIter] = biSection(@fun1, a, b, tol);
```

This will output xRoot and numIter in the main script workspace. Notice that there's a @ before the function is defined. This is because it uses a function handle. This means that we are defining an anonymous function which would work with any value of x as long as it's a real number.
We can verify our result with a MATLAB built-in function fzero and calculate the error.

```
xRoot_fzero = fzero(@fun1,[a b]);

error = (abs(xRoot - xRoot_fzero))/ xRoot_fzero;
```

Last but not least are the outputs of the function using fprintf and those will be the xRoot from bisection method and fzero, the number of iterations, and the verify the answer using the error.

iii. Calculations and Results

When the code is run, we obtain 2 values of roots, the first from the bisection method and the second from fzero.

```
The Root of the Function via Bisection is : 0.2019195557
The Root of the Function via fzero is : 0.2019203968
Number of Iterations: 17
Error: 4.165919e-06
```

From the results above, we can clearly see that the x value of the root is similar. This will be discussed more in the section below. In conclusion, the bisection method is a valid method to find a x root for a function if a range is given. As the range gets smaller, the number of iterations required to reach the root is smaller.

iv. Discussion

In conclusion, the bisection method is a valid method to find a x root for a function if a range is given. As the range gets smaller, the number of iterations required to reach the root is smaller. The bisection is a good method because it 'cuts' the range of interval into half and evaluates the function at the middle point. If the value of the function is exactly 0 or less than the tolerance, it is safe to approximate that the value is actually the root. If not, this depends on the value of the function at middle point, p, and function at left boundary a. This will determine whether p will be the left or right boundary depending on the value of p with respect to a and b.

The built-in MATLAB function, fzero, can help finding the root of a nonlinear function. As seen on the value in the section above, we obtain that the value of the root using both methods output the same value. Implicitly, these means that both methods are precise and accurate. There is always a small margin of error, but it is small enough to neglect the error. Thus, bisection and fzero is a function that uses a function (anonymous function with a function handle) and a range of values (i.e. x) to calculate the root of the function by 'cutting' the integral into halves until the evaluated function at that half point is equal to 0.

b. Fixed Point Iteration Method

i. Introduction

The goal in this part of the problem is identical to the problem above, which is to find the root of a nonlinear equation. However, in this part another method is used. This part used fixed point and Newton iteration method. Although it provides the same goal, this method and the method in part a. uses a different approach.

ii. Model and Methods

In part b, we are introduced to a new method to find the root of the function. The function is defined as $f(x) = 0$. Using fixed point, we will modify the function so that we have a function that is $x = g(x)$. Similar to the function

Adrian Loekman
UID: 105785820
CEE/MAE M20
October 22, 2021

above, we will also define a function which will be used in the main script. For part i and ii, the function will be `gx = 1 + 0.5*sin(x);` and put this into a function m-file fun2. For part ii the function would be `gx = 3 + 2*sin(x);` and set that to a different function m-file fun3.

The function fixPoint.m is now introduced to proceed with the root finding algorithm. The inputs for the function are: the function itself fun(2 or 3), the initial guess (x0), tolerance (tol), and maximum iteration (maxIter). The outputs would be the value of xStar from the fixed point method and xRoot, which is the root real root of the function itself. First, I defined a new variable, xVal, to define the initial guess value and k = 1 to start the iteration. The while loop coming up has 2 conditions, `abs(fun3(xVal)-xVal) > tol && (iter < maxIter)` which tells us that the value g(x) subtracted by x has to be greater than the tolerance and the iteration must be less than the maximum iteration for the while loop to run. Once the condition is no longer true, the root is gotten.

```
xNew = fun3(xVal); % new root, x(i+1) is g(xi)
xVal = xNew; % update values
iter = iter + 1; % update iteration
```

Inside the while loop, we will define a new x value (i.e. x(i+1)) with the g(x) from our function fun2. Then update the value of xVal with xNew and update the iteration. Do this until the conditions of the while loop are longer met. The outputs xStar will be the xVal and the xNew will be xNew. In some cases, these number will be the same values with a very small error.

In the main function for part ii, we have the parameters (inputs) and do the function call with the given inputs. Finally for part ii, we can print the outputs using fprintf and show the values for xStar and xRoot.

In part iii, we are using the same function fixPoint but since we have a different function, we need a new function m-file to define the new equation gx = fun(x). The function stated in the problem has a form of 3 + 2sin(x) - x = 0, so we need to change it to a new function where x = g(x). Then we can write a function where gx = 3 + 2sin(x). gx will be used to find the root of the equation.

In the main script, we will initialize the new parameters. Since we have 2 initial guesses, we can use an array. Next, we can do function calls for each respective initial guess. The next step is to print out the function output from both initial guesses.

An additional part in this main script is to output 2 figures in order to answer the question stated in part iii of the problem statement. The figure would be y

= x along with f1(x) = 1 + 0.5 sin(x) and f2(x) = 3 + 2 sin(x). Each function will be plotted along with y = x. For the x values, I used an array from 0 to 5 with step 0.01.

iii. Calculations and Results

When part i and ii is run, an output showing the root with the initial guess will be printed on the command window.

```
The Fixed point from initial guess 0.000 is : 1.4987009254
The Root from initial guess 0.000 is : 1.4986953555
```

In this part, the fixed point means the xStar and the root is xRoot. This shows that the root of the function f(x) = 1+0.5sin(x)-x is 1.49870.

When part iii is run, an output showing the root with the initial guesses will be printed on the command window. In addition to that, there are figures with plots that will help drawing an explanation for part iii.

```
The Fixed point from initial guess 3.000 is : 1.0008159523
The Root from initial guess 3.000 is : 4.6838231311
The Fixed point from initial guess 0.000 is : 4.6838231311
The Root from initial guess 0.000 is : 1.0008159523
```

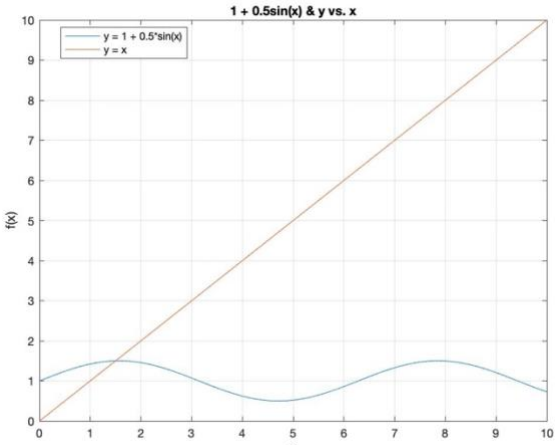

Figure 1 plot of 1+5sin(x) and y = x

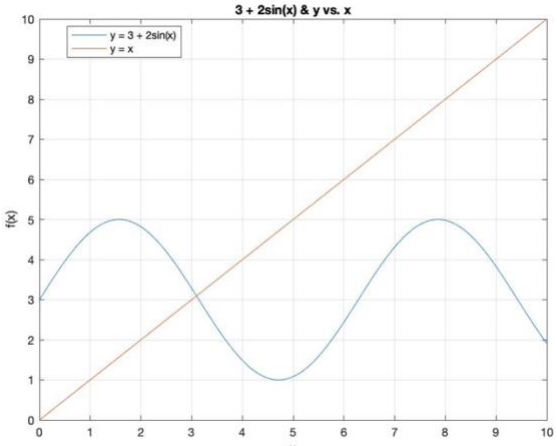

Figure 2 plot of 3+2sin(x) and y = x

Note from the print statements that the fixed point and the root does not have the same value.

iv. Discussion

Adrian Loekman
UID: 105785820
CEE/MAE M20
October 22, 2021

From part i and ii, we can obtain the root of the function and see that the value of xStar and xRoot is identical. xRoot is indeed the root of the function $f(x) = 1 + 0.5\sin(x) - x$. This statement can also be verified using figure 1. The purpose of finding the intersection between $y = 1+0.5\sin(x)$ and $y = x$ is because if we make both equations equal, we can get the root. The intersection is actually the root of the function. From figure 1, we see that the figure's intersection and the value we found using fixed point method is the same. Thus, we can use this algorithm for this particular nonlinear equation.

In part 3 however, the simulation with both initial guesses do not work. This implies that the root finding algorithm, fix point method, has limitations and cannot be used for every equation there is. This is similar to our previous method, bisection. Bisection method can only be used if we know the interval of where the root lies. Since of these limitations, we can check the true value of the intersection using figure 2. From figure 2, we can see that the true value of the root is actually 3.0944 but from fixpoint method, we get 4.683 and 1.0008. The main issue is that fixed point method does not always guarantee convergence. The reason why it worked for the previous function in part ii is because of the slope of the function at the initial guess. Thus, we can use it to test convergence of a function.

In this homework, I can substantially learn about root finding with functions. The only drawback from these methods is much information must be known prior to the method itself. This information is integral to getting convergence. In other words, convergence can be obtained only if the right value for each parameter is defined close to the root itself.