**BME 313L: Introduction to Numerical Methods in Biomedical Engineering**

**Lab Report**

**Lab\_4: Chapter 6. Roots: Open Methods**

**Last name, First name: Liu, Vincent**

**EID: VL5649**

**Lab Section: 14035 (Tuesday 9:30-12:30)**

**Problem 1 (Newton-Raphson)**

Use Newton-Raphson method to determine root of



using an initial guess of x=4.5 and . Start with the BME313L\_Lab4\_Problem1\_Name.m file and enter the required commands. Besides functions abs(), fprintf(), sin(), and cos() do not use any other built-in functions. Also, Anonymous function(*fhandle = @(arglist) expression*) is not allowable. As output, print a table on the Command Window with the results (first row of the required format is already inside the script file). The table should begin with labels and then start with iteration 1, not iteration 0.

For example, the first two rows should look like:

Iteration xi+1 f(xi) f’(x)) |Ea|

1 2.300098 -2.15e+001 2.4548e-001 74.67

(The following is your answer)

**MATLAB Code:**

x = 4.5; %initial guess

n = 0; %initialize counter

e = 1;

while e > .000003 %loop while error is high

n = n+1; %update counter

iter(n) = n; %store n

fxi(n) = cos(x) - .5\* x^2;

fprimex(n) = -sin(x)-x;

xi = x - fxi(n)/fprimex(n); %newton's method

e = abs((xi - x)/xi); %approximate error

x = xi; %update guess

nextxi(n) = x; %stores xi and error

Ea(n) = e\*100; %makes percent and stores

end

A = [iter;nextxi;fxi;fprimex;Ea]; %matrix of all the arrays

fprintf('Iteration \t xi+1 \t\t f(xi) \t\t f''(x) \t\t|Ea|\n') %prints column titles

fprintf('\t%d\t\t %f\t %f\t %f \t%g\n',A) %prints columns

**MATLAB Function:**

The purpose of this function was to use an open root finding method (Newton-Rhapson) to find the root of the function cos(x)-.5x^2. To do so, we needed an initial guess and to calculate the value of the function and the derivative of the function at the guess in order to update our guess. We could set this up as a loop so that it would keep updating our guess until our guess fell below a certain threshold. By setting up arrays along the way, we can then output values for every step along the way in a table

x = 4.5; %initial guess

This line of code sets our initial guess as 4.5, which was given to us as part of the problem.

n = 0; %initialize counter

e = 1;

These 2 lines of code initialize our counter as well as our error value. This ensures that our counter starts with the number of 1 within the loop and that the condition for the loop does not automatically terminate because it takes e as 0.

while e > .000003 %loop while error is high

This line of code creates a loop to continuously run until a condition is broken. In this case, the condition was our given error criterion, causing the function to update guesses until our error was below .0003% or .000003.

n = n+1; %update counter

iter(n) = n; %store n

The first line of code updates our counter for each iteration in the loop. This counter can then be used to populate certain values within arrays used later. The second line of code then keeps track of the number of iterations used by storing it as an array.

fxi(n) = cos(x) - .5\* x^2;

fprimex(n) = -sin(x)-x;

These 2 lines of code calculate

xi = x - fxi(n)/fprimex(n); %newton's method

This line of code uses newton’s method to update our guess for the root. Since f(x) and f’(x) were already previously calculated, we can just use those variables to perform newton’s method.

e = abs((xi - x)/xi); %approximate error

This line of code calculates the approximate error. This value is stored as the variable e, the same variable used to terminate our while loop.

x = xi; %update guess

nextxi(n) = x; %stores xi and error

These 2 lines of code update the guess and store it in an array to be put into a table later.

Ea(n) = e\*100; %makes percent and stores

This line of code creates an array of approximate error values. Because I took the numerical value of the approximate error rather than the percentage, I multiply 100 to get the percentage before storing it into the array.

end

This line of code closes the while loop.

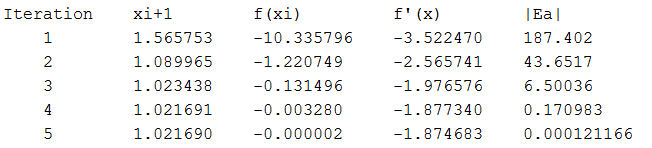
A = [iter;nextxi;fxi;fprimex;Ea]; %matrix of all the arrays

fprintf('Iteration \t xi+1 \t\t f(xi) \t\t f''(x) \t\t|Ea|\n') %prints column titles

fprintf('\t%d\t\t %f\t %f\t %f \t%g\n',A) %prints columns

These last 3 lines of code create a matrix of all the array values to be outputted, create titles for columns on the table, and then output the matrix as a table.

**Results:**

****

**Discussion:**

As shown by the results, with only 1 guess and newton’s method, we can reach the root of a function. With each successive iteration, the value of x logarithmically approached the root. In this case, it took 5 iterations to fall below the required error value of .0003%. This method works very fast to converge to the root, requiring only 5 iterations to go from 187% error to .00012%; however, this is only specific to certain equations like this one, and in some situations the newton-rhapson method might even not converge.

From this, we learned how to both use open root methods as well as how to use them without passing them as anonymous functions to function files. We reviewed implementing conditional loops in order to efficiently iterate functions until the condition was broken. Furthermore, we learned how to implement arrays without fixed values (value changes with each iteration). Lastly, we refreshed our knowledge on how to format and output arrays as tables using the fprintf function in MATLAB.

**Problem 2 (Modified Secant)**

Enzymes are biological catalysts that can be immobilized in porous matrices within reactors through which substrates can be diffused and reacted. Such enzyme immobilized reactors have been conceptualized as extracorporeal devices for the selective elimination or transformation of specific components in blood, such as bilirubin in jaundice and the anticoagulant heparin (Lavin et al, 1985; Sung et al, 1986). The mass balance of substrates on an immobilized enzyme particle leads to an overall relationship between the efficiency of reaction, , and the Thiele modulus, , which represents the ratio of substrate reaction rate to its diffusion rate. A large value for  indicates that diffusion limits the overall substrate consumption rate, while a small value for  indicates that reaction rate limits the overall consumption of the substrate. For a first-order reaction, this relationship becomes:



Determine and plot the values for the Thiele modulus when the efficiency of metabolite reaction varies from 30% to 60% at 10% increments. Use modified secant method for this problem using an initial guess of x=1, and  for all the values of metabolite reaction efficiencies. Start with the BME313L\_Lab4\_Problem2\_Name.m file and enter the required commands. Besides functions abs(), fprintf(), zeros(), disp(), plot() and tanh(), do not use any other built-in functions. Also, Anonymous function (*fhandle = @(arglist) expression*) is not allowable. As output, print a table on the Command Window with the results (first row of the required format is already inside the script file). The table should begin with labels and then start with iteration 1, not iteration 0. For example, the first two rows should look like:

Efficiency Value Iteration xi+1 xi+del\_xi f(xi) f(xi+del\_xi) |Ea|

3.0e-01 1 1.378 1.447 9. 02 5.702 74

(The following is your answer)

**MATLAB Code:**

delta = .05; %given pertubation

eff = 0.3:.1:.6; %efficiency values

fprintf('Efficiency Value\tIteration \t xi+1 \t\t\txi+del\_xi \t f(xi) \t f(xi+del\_xi) \t |Ea|\n') %table column

for z = 1:length(eff) %vector with same length as eff

n = 0; %intialize counter

x = 1; %initial guess

e = 1; %intializes/reintializes error

while e > .0003 %loop for conditional (error)

n = n + 1; %counter

EfficiencyValue(n) = eff(z); %efficiency value used in calculation

f = 1/x\*(1/tanh(3\*x)-1/(3\*x)) - eff(z); %function

Iteration(n) = n; %iteration array

xidelxi(n) = x + delta \* x; %xi + delta\*xi

fxi(n) = f; %stores value of function

fxidelxi(n) = 1/(xidelxi(n))\*(1/tanh(3\*(xidelxi(n)))-1/(3\*(xidelxi(n)))) - eff(z); %function with pertubation

nextxi(n) = x - delta\*x\*fxi(n)/(fxidelxi(n)-fxi(n)); %calculates next guess

e = abs((nextxi(n)-x)/nextxi(n)); %calculates error

Ea(n) = e\*100; %error array

x = nextxi(n); %updates guess

end

A = [EfficiencyValue;Iteration;nextxi;xidelxi;fxi;fxidelxi;Ea]; %matrix for table

fprintf(' \t%1.1d \t\t\t%d \t\t %f\t\t%f\t %+f \t %+f \t\t%+f\n',A) %print columns

fprintf('\n') %print space between efficiency values

root(z) = nextxi(n); %creates a vector of calculated roots

EfficiencyValue = 0; %resets arrays to be used

Iteration = 0;

nextxi = 0;

xidelxi = 0;

fxi = 0;

fxidelxi = 0;

Ea = 0;

end

plot(root,eff) %plots roots against efficiency values

xlabel('Roots') %makes plot understandable

ylabel('Efficiency Value')

**MATLAB Function:**

The purpose of this function was to use another root finding function (modified secant) to calculate the root for a function relating the efficiency of an enzyme to its reaction and diffusion rates. In this problem, we had to calculate phi for multiple efficiency values, requiring us to nest a while loop within a foor loop. To set this problem up, we had a while loop that would continualy update the gueses for the modified secant function until the error condition was met. This was nested within a for loop so that it could be run for every efficiency value that we needed to test. These values could then be outputted in a table and graphed to show the change in reaction/diffusion rates as a result of efficiency.

delta = .05; %given perturbation

This first line of code sets the given perturbation, .05, as the variable delta. This value was given to us as part of the problem and is important to be used in the modified secant method to avoid using a second guess.

eff = 0.3:.1:.6; %efficiency values

This line of code generates a vector of all the efficiency values that we need to calculate for.

fprintf('Efficiency Value\tIteration \t xi+1 \t\t\txi+del\_xi \t f(xi) \t f(xi+del\_xi) \t |Ea|\n') %table column

This line of code prints out the first row of the table with titles for each of the columns. While it may seem out of place here, I had to put it outside of the for loop in order to only print it once, so it ended up here.

for z = 1:length(eff) %vector with same length as eff

This line of code sets up a for loop to repeat everything within it for a vector, z, of the same length as the efficiency vector generated earlier. This could have been accomplished in a similar fashion by creating the efficiency vector here instead, but I needed to be able to have a counter for the value within the efficiency vector being used and this was my way of doing so.

n = 0; %intialize counter

x = 1; %initial guess

e = 1; %intializes/reintializes error

These 3 lines of code initialize the counter, the guess, and the error, respectively. By having these within the for loop but outside of the while loop, we can effectively reinitialize all of these values for every calculation of efficiency.

while e > .0003 %loop for conditional (error)

This line of code sets up a for loop to reiterate the modified secant function until the condition for error is met.

n = n + 1; %counter

This line of code updates the counter.

EfficiencyValue(n) = eff(z); %efficiency value used in calculation

This line of code populates an array with the current value of efficiency that the code is working with.

f = 1/x\*(1/tanh(3\*x)-1/(3\*x)) - eff(z); %function

This line of code is the function that was given to model the efficiency. It is set equal to 0 so that we can solve for the root.

Iteration(n) = n; %iteration array

This line of code creates an array of the current iteration.

xidelxi(n) = x + delta \* x; %xi + delta\*xi

This line of code calculates xi+delta\*xi to be used in the modified secant method. It is also stored in an array to be used later when making a table of values.

fxi(n) = f; %stores value of function

This line of code stores the current value of the function with the current guess in an array.

fxidelxi(n) = 1/(xidelxi(n))\*(1/tanh(3\*(xidelxi(n)))-1/(3\*(xidelxi(n)))) - eff(z); %function with perturbation

This line of code plugs x+delta\*x into the given function as one of the values needed in the modified secant method. It is also stored in an array to be used later when generating a table.

nextxi(n) = x - delta\*x\*fxi(n)/(fxidelxi(n)-fxi(n)); %calculates next guess

This line of code uses the modified secant method in order to calculate the next guess, which is also stored as an array.

e = abs((nextxi(n)-x)/nextxi(n)); %calculates error

This line of code calculates the approximate error of the guesses and stores it as the variable e, which is also the variable used as the conditional in the while loop.

Ea(n) = e\*100; %error array

This line of code stores the error into an array to be used in the table.

x = nextxi(n); %updates guess

This line of code updates the guess to be used in the next set of iterations

end

This line of code closes the while loop.

A = [EfficiencyValue;Iteration;nextxi;xidelxi;fxi;fxidelxi;Ea]; %matrix for table

fprintf(' \t%1.1d \t\t\t%d \t\t %f\t\t%f\t %+f \t %+f \t\t%+f\n',A) %print columns

fprintf('\n') %print space between efficiency values

These 3 lines of code create the matrix to be outputted, output the matrix, and then print a blank line for formatting reasons. By printing a blank line after outputting the matrix, there is a gap between separate matrices of different efficiency values.

root(z) = nextxi(n); %creates a vector of calculated roots

This line of code pulls the last guess of x, or the value closest to the root, and stores it in the array called root. The number of the value in the array is the variable z used in the for loop.

EfficiencyValue = 0; %resets arrays to be used

Iteration = 0;

nextxi = 0;

xidelxi = 0;

fxi = 0;

fxidelxi = 0;

Ea = 0;

These 7 lines of code reset the 7 arrays used in the table for every calculation of efficiency. In doing so, we don’t have to worry about previous values being left in the arrays resulting in confusing tables, while working in the for loop (but outside the while loop).

end

This line closes the for loop.

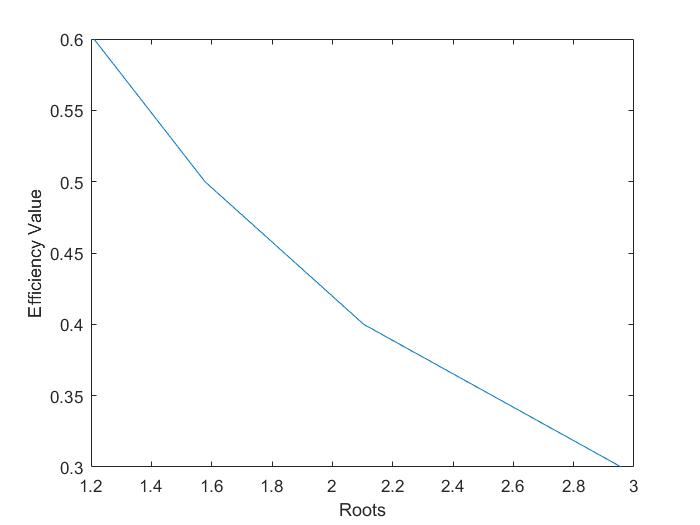
plot(root,eff) %plots roots against efficiency values

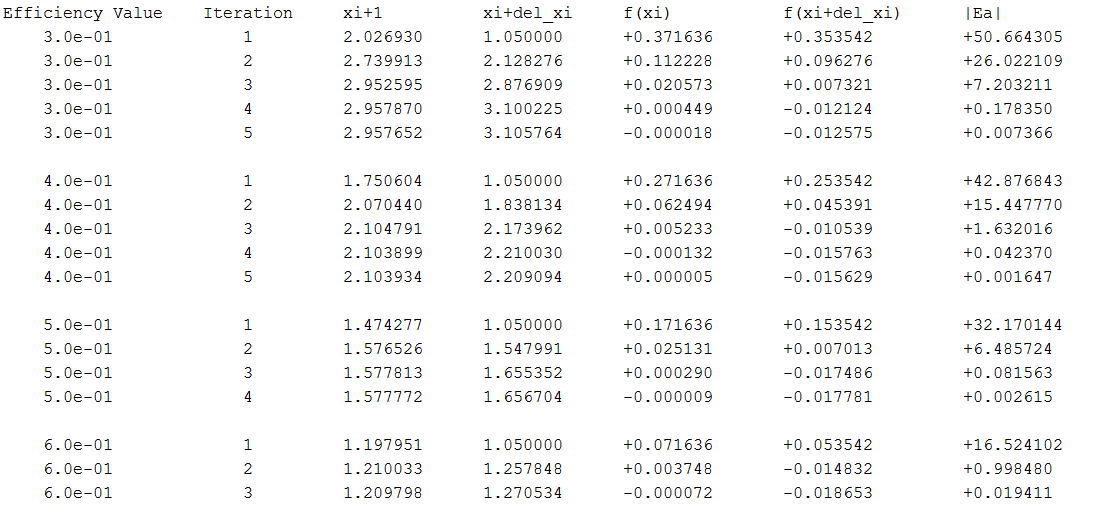
xlabel('Roots') %makes plot understandable

ylabel('Efficiency Value')

These last 3 lines of code plot the efficiency values against the roots pulled from each respective efficiency value’s calculations.

**Results:**



****

**Discussion:**

As shown by the results, as the efficiency of an enzyme decreased, so did the root. The root in this case, was phi, a value relating the enzymes reaction and diffusion rates, which makes sense logically. If the enzymes reaction and diffusion rates are low, so it the enzyme’s efficiency. Like other open root finding methods, the error value seems to decrease logarithmically. The results also show that the modified secant method only needs 1 guess in order to reach the root.

From this problem, we learned how to nest a loop within another loop. We also learned how to use the secant method to solve for roots and reviewed how to plot arrays against each other. Furthermore, we reviewed how to populate arrays of unpredetermined lengths. Lastly, we reviewed how to format and create tables using MATLAB’s fprintf function.

**Problem 3 (Secant)**

When laser light is incident on tissue, the tissue acts as a transfer medium that can reflect, absorb, scatter, and transmit various portions of the incident wave of radiation. Diffusion equations can be constructed to describe the propagation of light intensity as a function of distance. Imagine an isotropic light source from a laser fiber within a tissue that can both absorb and scatter light. If the tissue acts as an infinite medium, under conditions of predominant scattering, the light fluence rate (in W/cm2) at a large distance r from the fiber can be shown to be:



where the penetration depth, ; andis the tissue absorption coefficient. The anisotropy of the medium, g=0.9; absorption coefficient,=0.1cm-1, scattering coefficient,=100cm-1. Find the distance away from the point source of laser where the light intensity drops to 30% of that of the source. Note that .

Use Secant Method with the two initial estimates as xi-1 = 0.12 and xi = 0.13. Use = 0.01% for this problem. Start with the BME313L\_Lab4\_Problem3\_Name.m file and enter the required commands. Besides functions abs(), exp(), sqrt(), and fprintf(), do not use any other built-in functions. Also, Anonymous function (*fhandle = @(arglist) expression*) is not allowable. As output, print a table on the Command Window with the results (first row of the required format is already inside the script file). The table should begin with labels and then start with iteration 1, not iteration 0.

For example, the first two rows should look like:



(The following is your answer)

**MATLAB Code:**

g = .9; %given constants

mua = .1;

mus = 100;

xi1 = .12; %given guesses

xi = .13;

phir = .3; %finding for this value

D = (mua + mus\*(1-g))/3;

delta = sqrt(D/mua);

e = 1; %starts error at something other than 0

n = 0; %initialize counter

while e > .0001

n = n + 1; %counter

iter(n) = n; %array for counter

fxi = exp(-xi/delta)/4/pi/D/xi-phir; %function

fxin(n) = fxi; %array

fxi1 = exp(-xi1/delta)/4/pi/D/xi1-phir; %function w/ previous fxi

fxi1n(n) = fxi1; %array

nextxi = xi - fxi\*(xi1-xi)/(fxi1-fxi); %secant method

xi1 = xi; %updates guesses

xi1n(n) = xi1; %array for guess

xi = nextxi; %updates guesses

xin(n) = xi; %array for guess

e = abs((xi - xi1n(n))/xi); %calculates error

Ea(n) = e \* 100; %error array

end

A = [iter;xin;xi1n;fxin;fxi1n;Ea]; %matrix of arrays

fprintf('Iteration \t xi \t\t xi\_1 \t f(xi) \t\t f(xi\_1) \t|Ea|\n') %creates table

fprintf('\t%d\t\t%f \t%f \t%+f \t%+f \t%f\n',A)

**MATLAB Function:**

The purpose of this function was calculate where the light intensity in tissue had an intensity of 30%. To do so, we can set the equation given, equal to 0 and solve for roots. In this problem, we were asked to use the secant method in order to so. Iterations could be repeated if we set up a while loop and then we could output a table of values,

g = .9; %given constants

mua = .1;

mus = 100;

These first 3 lines of code are parameters given as part of the problem.

xi1 = .12; %given guesses

xi = .13;

These 2 lines of code are the initial 2 guesses needed as part of the secant method.

phir = .3; %finding for this value

This line of code was another one of the parameters given in the problem. Because of the way the problem was written, the ratio of phi naught to phi r could just be written as phi r, given as .3.

D = (mua + mus\*(1-g))/3;

delta = sqrt(D/mua);

These 2 lines of code calculate more values to be used in the function used to model light in tissue. These equations were all given by the problem.

e = 1; %starts error at something other than 0

n = 0; %initialize counter

These 2 lines of code initialize the error value (so that the while loop doesn’t terminate immediately) and initialize the counter to keep track of the number of iterations.

while e > .0001

This line of code creates the conditional while loop to repeat while the approximate error condition is not met.

n = n + 1; %counter

iter(n) = n; %array for counter

These 2 lines of code update the counter and store the counter in an array.

fxi = exp(-xi/delta)/4/pi/D/xi-phir; %function

fxin(n) = fxi; %array

These 2 lines of code calculate the current value if you plug in the current guess for the root and store it into an array.

fxi1 = exp(-xi1/delta)/4/pi/D/xi1-phir; %function w/ previous fxi

fxi1n(n) = fxi1; %array

These 2 lines of code calculate the previous value if you plug in the previous guess and solve for the root. It is then stored into an array. This value is calculated to be used as part of the secant method later on

nextxi = xi - fxi\*(xi1-xi)/(fxi1-fxi); %secant method

This line of code uses the secant method in order to calculate the next guess of x

xi1 = xi; %updates guesses

xi1n(n) = xi1; %array for guess

xi = nextxi; %updates guesses

xin(n) = xi; %array for guess

These 4 lines of code update the guesses for the root of the function and store the same values into arrays to be outputted as part of the table.

e = abs((xi - xi1n(n))/xi); %calculates error

Ea(n) = e \* 100; %error array

These 2 lines calculate the error and store it in an array. Because the e value calculated is a decimal it has to be multiplied by 100 before being stored.

end

This line of code closes the while loop.

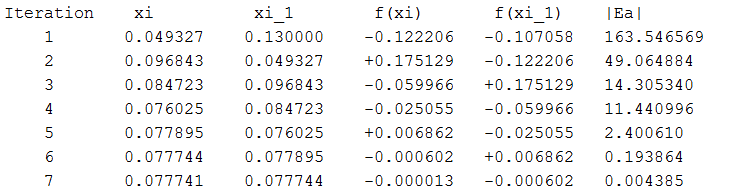
A = [iter;xin;xi1n;fxin;fxi1n;Ea]; %matrix of arrays

fprintf('Iteration \t xi \t\t xi\_1 \t f(xi) \t\t f(xi\_1) \t|Ea|\n') %creates table

fprintf('\t%d\t\t%f \t%f \t%+f \t%+f \t%f\n',A)

These last 3 lines of code create a matrix out of all the arrays to be outputted, print a line of code with the column titles for the table, and format and output the matrix of the values in the table.

**Results:**

****

**Discussion:**

As shown by the results, we can solve for the root of a function using 2 guesses. These 2 guesses, however, do not necessarily have to be on either side of the root (exhibit sign change). As seen in the table, in some situations, the values are both negative, a result of the secant method not being a closed bracketing method. Like other open root methods, the approximate error decreases in a logarithmic sort of fashion.

From this problem, we reviewed how to implement open root finding methods without using anonymous functions to pass to another function file. We also reviewed how to use conditional while loops to efficiently reiterate calculations such as updating the guess of the root using a root finding method. Furthermore, we reviewed arrays of unfixed lengths and outputting tables using the fprintf function.