Problem 1:

1. Write a function that takes an interval and returns the two points on the x axis used in one iteration of a golden section search

**function** **[**outLower**,**outUpper**]** **=** GoldenRatio**(**inLower**,**inUpper**)**

%function [outLower,outUpper] = GoldenRatio(inLower,inUpper)

% Golden Ratio is a function which returns the two inner values which

% are needed for Golden ration given two intial lower and upper

% bounds

% Inputs: inLower, the lower bound for "x"

% inUpper, the upper bound for "x"

% Outputs:outLower, the lower GR value

% outUpper, the upper GR value

%Follow the formula used in class

dist **=** **(**inUpper **-** inLower**)** **/** 1.618**;**

outLower **=** inUpper **-** dist**;**

outUpper **=** inLower **+** dist**;**

**end**

1. Write a script that calls the function from 2(a) three times, starting with the interval [5,7]. Discard the left-most, right-most, and left-most sections during the three iterations, in order.

%Code for Problem 1

xLower **=** 5**;**

xUpper **=** 7**;**

%Run the first time

**[**outLower**,** outUpper**]** **=** GoldenRatio**(**xLower**,** xUpper**);**

%Throw out the leftmost

xLower **=** outLower**;**

fprintf**(**"First Run: xLower is %.4f and xUpper is %.4f\n"**,**...

xLower**,** xUpper**);**

%Run a second time

**[**outLower**,** outUpper**]** **=** GoldenRatio**(**xLower**,** xUpper**);**

%Throw out the rightmost

xUpper **=** outUpper**;**

fprintf**(**"Second Run: xLower is %.4f and xUpper is %.4f\n"**,**...

xLower**,** xUpper**);**

%Run a third tim

**[**outLower**,** outUpper**]** **=** GoldenRatio**(**xLower**,** xUpper**);**

%Throw out the leftmost

xLower **=** outLower**;**

fprintf**(**"Third Run: xLower is %.4f and xUpper is %.4f\n"**,**...

xLower**,** xUpper**);**

|  |  |  |  |
| --- | --- | --- | --- |
|  | First Run | Second Run | Third Run |
| Lower | 5.7639 | 6.2360 | 6.0557 |
| Upper | 6.2361 | 6.5279 | 6.2361 |

The final bounds are [6.0557, 6.5279].

Problem 2:

1. =

**function** **[**root**,** numIterations**]** **=** NewtonRaphson**(**intGuess**,** func**,**...

derivFunc**,** error**,** maxNumIterations**)**

%function [root, numIterations] = NewtonRaphson(intGuess, func,...

% derivFunc, error, maxNumIterations)

%

% Function implements the Newton-Raphson open root finding method

% Inputs: intGuess, starting guess for the function

% func, function using to find roots of

% derivFuc, derivative of the function

% error, desired approximate relative error threshold

% maxNumIterations, if reaches this then breaks from func

% Outputs:root, root location

% numIterations, numb of iterations it took to find the root

%Set intial values at the start

numIterations **=** 0**;**

root **=** intGuess**;**

%Loop until maxNumIterations reached

**while(**numIterations **<=** maxNumIterations**)**

%Result of plugging in the root

y **=** func**(**root**);**

%Derivative at the point

yPrime **=** derivFunc**(**root**);**

%Calculate where root would be on a straight line

x **=** root **-** **(**y **/** yPrime**);**

%If it is close to same number calculate just before, assume root found

**if** **(**abs**((**x **-** root**)** **/** x**)** **<** error**)**

root **=** x**;**

**return**

**end**

%Set as current root

root **=** x**;**

**end**

**end**

%Function operating on

func = @(x) x^10 - 10 \* x^5 + 0.5 \* exp(x) - .45;

%The derivative

derivFunc = @(x) 10 \* x^9 - 50 \* x^4 + 0.5 \* exp(x);

%Set of intial guesses

intGuess = linspace(-3, 3, 600);

%Find roots of each guess

roots = zeros(1, 600);

funcPoint = zeros(1, 600);

for i = 1:600

roots(i) = NewtonRaphson(intGuess(i), func, derivFunc, exp(-6), 100);

funcPoint(i) = derivFunc(intGuess(i));

end

%Make a graph of which root found for each intial guess

figure();

plot(intGuess, roots);

title("Roots Found Based on Different Intial Guesses");

xlabel("Interval of Roots Tested from -3 to 3");

ylabel("Found Root");

hold on;

figure();

fplot(func, [-3 3]);

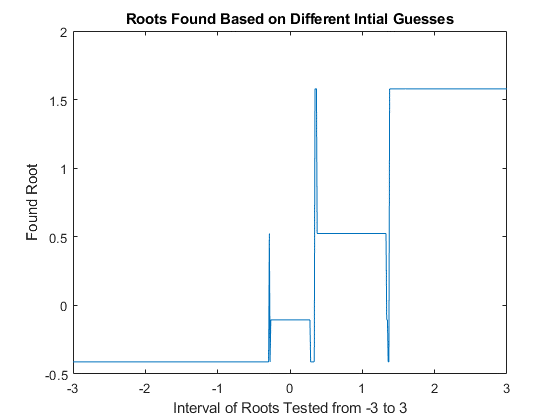
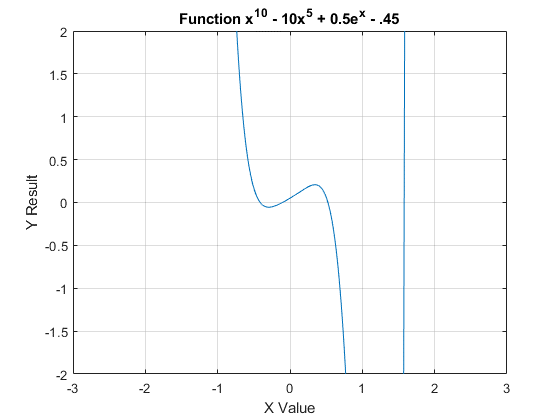
ylim([-2 2]);

grid on;

title("Function x^1^0 - 10x^5 + 0.5e^x - .45");

xlabel("X Value");

ylabel("Y Result");



The graph on the left shows the function of which we are trying to find the roots (zeros) of. The graph on the right helps visualize what root each initial guess returned. 600 different initial values were tested on the interval of -3 to 3 to create the graph on the right. What it shows is that near each local minimum or maximum, the closest root may not be found. This is a result of the derivative being flat at those points and therefore going to far in the horizontal direction i.e. moving away from the closest root. A specific example is x at -0.28548 which returns y is 0.52431 even though y at -.41199 or -.10565 would be closer.