**Homework 1: Errors, Plotting, and Roots (Part 1)**

Problem 1:

**function** **[**error**]** **=** CalculateError**(**true**,** approximate**)**

%

% [absolutRelErr] = CalculateError(true, approx)

%

% Calculates the absolute relative error between two vectors

% Input:

% true -the true/expected value

% Output:

% approx -the calculated value

% Calculates the error using the absolute relative error equation

error **=** abs**((**true **-** approximate**)** **/** true**)** **\*** 100**;**

**end** % End of function CalculateError

Problem 2:

For this part, the errors are computed using the function created before. All the angles are found with their corresponding sines. Then the CalculateError function is used to find the error values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | -0.7854 | -0.7828 | -0.7802 | -0.775 | -0.7749 | -0.7723 |
| sin(x) | -0.7071 | -0.7053 | -0.7034 | -0.7015 | -0.6997 | -0.6978 |
| Absolute Relative Error (%) | 11.0721 | 1.0.9927 | 10.9136 | 10.8349 | 10.7565 | 10.6785 |

% Find the vector of angles from -pi/4 to pi/4 spaced with 600 points

angleVector **=** linspace**((-**pi**/**4**),** **(**pi**/**4**),** 600**);**

% Matrix to hold true sin values of each angle

sineVector **=** zeros**(**1**,** 600**);**

% Matrix to hold absolute relative errors

errorVector **=** zeros**(**1**,** 600**);**

% Iterate through all angle values

**for** i **=** 1**:**600

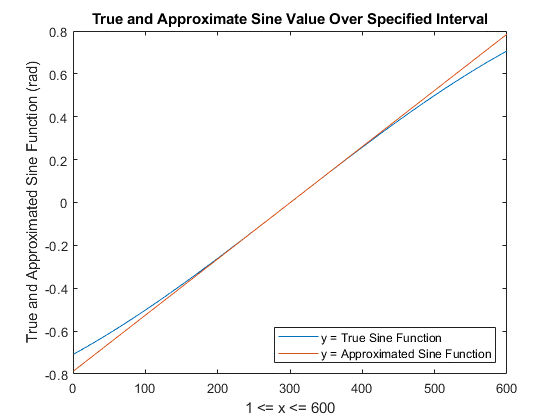
sineVector**(**1**,** i**)** **=** sin**(**angleVector**(**1**,** i**));** % Finds the sin of each of the angles

errorVector**(**1**,** i**)** **=** CalculateError**(**sineVector**(**1**,** i**),** angleVector**(**1**,** i**));** % Finds the absolute relative error between guess and true

**end**

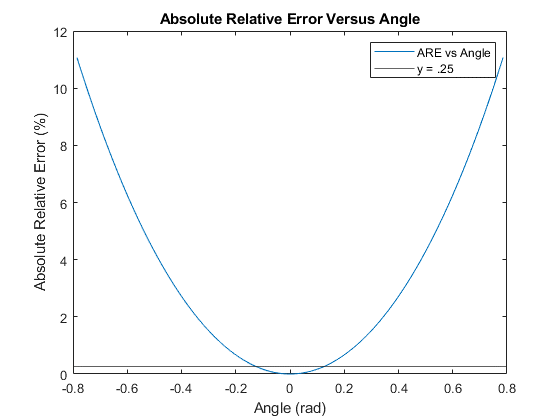
Problem 3:

a)



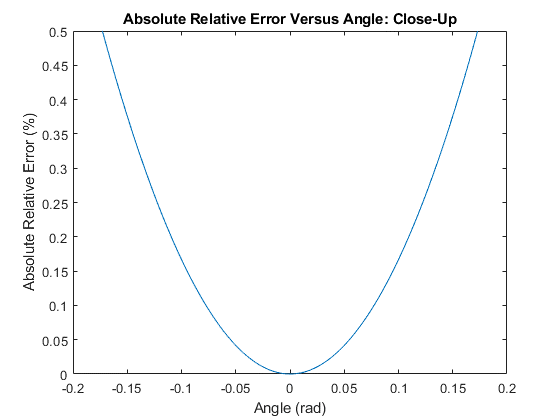
From -.2 to .2, the values of the true and the approximation are equal to each other and their individual lines are almost in distinguishable from each other. The range where this approximation is “good,” therefore, is angles between -.2 and .2 in radians. In degrees, this range is -11.4592 to 11.4592.

b)



The parabola represents the values in absolute relative error values vs angle values. The linear horizontal line is at .25%. The area below this line represents “good” guesses because of their small error.

c)



This graph is a zoomed in version of part c. We can see the values on the graph better. Our original statement from part a which claimed the guess is “good” if the angle in radian is between -.2 and .2. It can clearly be seen hear that the error for that range is just over half a percent, so not much.

Problem 4:

a)

**function** **[**bestGuess**,** numIters**]** **=** Bisection**(**func**,** lowerBound**,** upperBound**,** maxFinalBracket**)**

%

% [bestGuess, numIters] = Bisection(func, lowerBound, upperBound, maxFinalBracket)

%

% This function will perform the root finding method of bisection.

% Input:

% func - the function which the root should be found of

% lowerBound - the bottom starting value

% upperBound - the upper starting value

% maxFinalBracket - error or how method knows when it is "good"

% Output:

% bestGuess - final guess of the root

% numIters - number of iterations to find bestGuess

% number of times the loop must run through

numIters **=** 0**;**

% keep track of current point lowerBound is at on function

lower **=** func**(**lowerBound**);**

% function loops until bounds are below specified threshold

**while** **((**upperBound **-** lowerBound**)** **>** maxFinalBracket**)**

numIters **=** numIters **+** 1**;**

bestGuess **=** **(**upperBound **+** lowerBound**)** **/** 2**;**

curGuess **=** func**(**bestGuess**);**

% Determine which side of the point guess falls on

**if** **((**lower **\*** curGuess**)** **>** 0**)** % same side of point

lower **=** curGuess**;**

lowerBound **=** bestGuess**;**

**else**

upperBound **=** bestGuess**;**

**end**

**end**

**end**

b)

principal **=** 25000**;**

numMonths **=** 60**;**

payment **=** 625**;**

%Use the monthly interest formula - desired rate and calculate the result

func **=** **@(**interest**)** **(**principal **\*** interest **/** 1200**)** **/** ...

**(**1 **-** **(**1 **+** interest **/** 1200**)^(-**1 **\*** numMonths**))** **-** payment**;**

%Determines where the function is equal to zero with bracket of 1

**[**bestGuess**,** numIters**]** **=** Bisection**(**func**,** 1**,** 35**,** 1**);**

fprintf**(**"The best guess is: %d\nNumber of iterations is: %d\n\n"**,**...

bestGuess**,** numIters**);**

%Do again with bracket of .1

**[**bestGuess**,** numIters**]** **=** Bisection**(**func**,** 1**,** 35**,** .1**);**

fprintf**(**"The best guess is: %d\nNumber of iterations is: %d\n\n"**,**...

bestGuess**,** numIters**);**

%Do again with bracket of .01

**[**bestGuess**,** numIters**]** **=** Bisection**(**func**,** 1**,** 35**,** .01**);**

fprintf**(**"The best guess is: %d\nNumber of iterations is: %d\n\n"**,**...

bestGuess**,** numIters**);**

For max bracket of 1: best guess is 1.746875e+01 and number of iterations is 6

For max bracket of .1: best guess is 1.726953e+01 and number of iterations is 9

For max bracket of .01: best guess is 1.727783e+01 and number of iterations is 12

c)

For a max bracket of 1, Elapsed time is 0. 000325 seconds.

For a max bracket of .1, Elapsed time is 0. 000204 seconds.

For a max bracket of .01, Elapsed time is 0. 000202 seconds.

The program seems to get faster after each resolution, but this is not a good representation because of how fast the program runs. Based off of what was said in class, MATLAB’s tic toc command is not accurate for measuring times less than a millisecond.