!--GNU Octave was used--!

?!-GNU Octave Copy-To-Clipboard is not working properly so the output may not look as it should be

**Problem 1**

Call b1(.m)

In order to run the function with the given parameters from 1(b)

**-Files-**

**b1.m** - Outputs the function bisection\_method with the output: c, n, error

%This script calls the bisection\_method with the given functions,

%intervals, etc.

%Function: f(x) = 2sin(x) - x

%Interval: [pi/2, pi]

%Accuracy: Error < 10^-5

%Maximum Iteration: Not given.

[c, n, error] = bisection\_method(@(x) 2\*sin(x) - x, pi/2, pi, 10^-5, 999);

disp("c: "), disp(c)

disp(" ")

disp("n: "), disp(n)

disp(" ")

disp("error: "), disp(error)

disp(" ");

**sign.m -** Input(x)

If x > 0, return 1

If x < 0, return -1

If x = 0, return 0

function [output] = sign (input)

if(input > 0)

output = 1;

elseif(input < 0)

output = -1;

else

output = 0

endif

endfunction

**Reason is written in bisection\_method**

**bisection\_method.m**

#Function F

#Two numbers: a, b

#Error tolerance: tol

#Maximum number of iterations, N

#Root: c

#Number of iterations: n

#Bound error: err

#E = |x.n - c|

#Is it |x.n - x.n+1| ?

#For function as input to work, you need to apply the Anonymous Function syntax

#Which is... @(x) 2\*sin(x) - x

function [c, n, err] = bisection\_method (f, a, b, tol, N)

n = 1;#Initial Iteration: 1

fa = f(a);

#This is E = |x.n - c| The difference between the next c and the

err = (b-a)/2;

while n < N

#Initialize

#a + (b-a)/2 is used for the roundoff error in computer arithmetic

c = a + (b-a)/2;

fc = f(c);

#End the loop if c == 0

if(f(c) == 0)

break;

endif

#If fa\*fc is higher than 0, then 0 is located in the right hand

#If it is lower than 0, then 0 is located in the left hand

#Function sign() is used for sign in order to remove the possibility overflow

if((sign(fa)\*sign(fc)) > 0)

a = c;

fa = fc;

else

b = c;

endif

#If next error is less than the tolerance, then exit immediately

#without incrementing the iteration counter, n

err = (b-a)/2;

if(err > tol)

n++;

else

break;

endif

endwhile

endfunction

**-Output-**

>> b1

c:

1.8955

n:

17

error:

5.9921e-006

**Analysis/Results/Discussion Section (ARD Section)**

1(c): It took 17 steps

1(d): -1.8955, 0 , 1.8955

The Bisection Method is a root-finding problem that comes from the foundation of the Intermediate Value Theorem. The IVT states that when there is K between *a* and *b* where *f(a)* and *f(b)* are of opposite signs, then there exist *f(K) = 0*. The Bisection simply takes this theorem to action by bisecting through intervals and subintervals. Thus, it gets closer and closer to the number *K*.

The function and parameters make entire use of the bisection method and succeeds in converging to the desired root well within the interval given and error accuracy.

As for the results, it took 17 steps before the method stopped due to error accuracy.

**Problem 2**

Call b2(.m)

In order to run the function with the given parameters from 2(b)

**-Files-**

**b2.m**

%Same as 1b.m

%Function: 2\*sin(x) = g(x)

%Initial Guess: 3\*pi/4

%Tolerance: Error < 10^-5

%Interval: [pi/2, pi]

%Maximum Iterations: 999

[c, n, err] = fixed\_point\_iteration(@(x) 2\*sin(x), 3\*pi/4, 10^-5, 999);

disp("c: "), disp(c)

disp(" ")

disp("n: "), disp(n)

disp(" ")

disp("error: "), disp(error)

disp(" ");

**fixed\_point\_iteration.m**

#Function: g

#Initial guess: x.0

#Error Tolerance: tol

#Maximum Iteration: N

#Current iteration: n

#Basic Idea: g(p) = p

function [c, n, err] = fixed\_point\_iteration(g, x0, tol, N)

#Initialize The Needed Variables

#Initialize n

n = 1;

c = -1;

while n < N

p = g(x0);

err = abs(p - x0);

#disp(err)

if(err < tol)

c = p;

break;

endif

x0 = p;

n++;

endwhile

endfunction

**-Output-**

>> b2

c:

1.8955

n:

25

error:

7.0036e-006

**Analysis/Results/Discussion Section (ARD Section)**

2(c): It took 25 steps

For this function to use f(x) = 2sin(x) - x into a successful g(x), g(x) = 2sin(x) was used. By having the initial guess as 3\*/2, which is in the interval of [/2, ], the fixed point iteration takes 25 steps to get there before blocked by an error of 7.0036e-006 < error accuracy < 10^-5.

The fixed point iteration is a method which attempts to find the number x where g(x) = x. This method goes hand to hand with the root finding function and thus, you can create many different g(x) from f(x) and x to create a fitting function that handles this method. Because this relies on the root finding method, this also has congruency with the Intermediate Value Theorem.

**Problem 3**

Call a3(.m)

In order to run the function with the given parameters from 3(a)

Call d3(.m)

In order to run the function for 3(d)

**-Files-**

**a3.m**

%Same as others

[c, n, err] = fixed\_point\_iteration(@(x) x^3 - 5\*x - 7, -4, 10^-5, 100);

disp("c: "), disp(c)

disp(" ")

disp("n: "), disp(n)

disp(" ")

disp("error: "), disp(err)

disp(" ");

**d3.m**

[c, n, err] = fixed\_point\_iteration(@(x) x - (x^3 - 6\*x - 7)/(3\*x^2 - 6), -0.5, 10^-5, 100);

disp("c: "), disp(c)

disp(" ")

disp("n: "), disp(n)

disp(" ")

disp("error: "), disp(err)

disp(" ");

**-Output-**

**3(a)**

>> a3

c:

-1 <- <- <- % This means that c was never found

n:

100

error:

NaN

**3(d)**

>> d3

c:

2.9006

n:

61

error:

9.3898e-011

**Analysis/Results/Discussion Section (ARD Section)**

3(b): No. It diverged. The error goes to to infinity to NaN.

3(c): Perhaps, if your x0 is the real root of the equation but that is unlikely. All other initial guesses will surely diverge into infinity. The reason for this is that most of the g(x) goes to extremely high negative or positive number whenever x becomes the point to apply. Another reason is that it does not follow the criteria for the Fixed Point Theorem especially when g(x) must be for each x.

3(d): g(x) = x -

It took 61 steps

There is not much else to say other than the fact that creating multiple g(x) from f(x) is important in solving the fixed point iteration. It is definitely flawed due to the irregular function of plugging in the output of the g(x) as x but it is definitely doable.