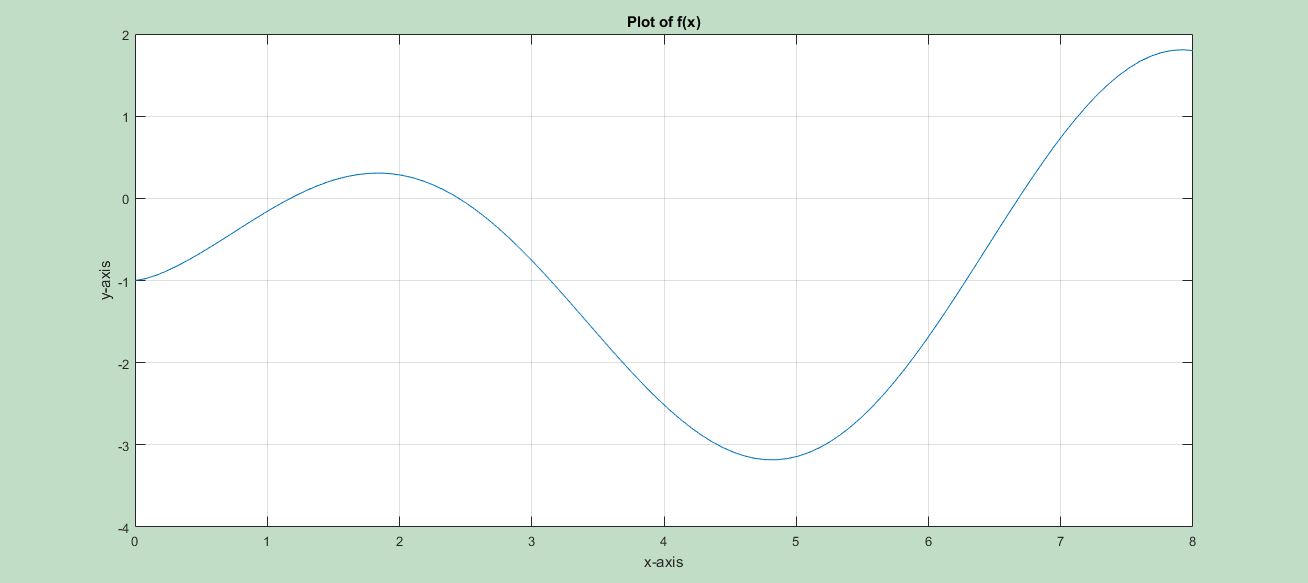
Johnny Donza MEC 320

Programming Assignment 1 Matlab

Submission Number: **b55ab682-ad6f-4376-bd8c-e27a33207292**

**Task 1**

1. The final root for the false –position method was determined to be 1.1748. Whereas the final root for the bisection method was 6.6953. One would normally expect the values for these roots to be approximately the same. However, as indicated by the plot of the function, there are actually multiple existing roots. All that happened here was that the root finding methods instituted for this task were only able to calculate two of these roots based off of the pre-determined bounds.
2. The final approximate error for the bisection method and the false-position method was 0.1753 % and 8.0374 % respectively.
3. Plot of function.



1. Matlab Code for Bisection Method:

close all

clear

clc

%Bisection Method

%Initial Lower and Upper Bound Guess

xu = 7;

xl = 1;

%Begin loop for 8 iterations

i = 0;

for n = 1:8

i = i+1;

Xr = (xu+xl)/2;

func\_Xr = (sqrt(Xr)\*sin(Xr))-1;

func\_xl = (sqrt(xl)\*sin(xl))-1;

func\_xu = (sqrt(xu)\*sin(xu))-1;

if func\_xl\*func\_Xr < 0 %If true, the root lies in the lower sub-interval

xu = Xr;

elseif func\_xl\*func\_Xr > 0 %If true, the root lies in the upper sub-interval

xl = Xr;

else %Only true if func\_xl\*func\_Xr = 0

xroot = Xr;

end

ea = abs((xu-xl)/(xu+xl)) \*100;

end

disp('Final Root =');

disp(Xr);

disp('Approximate Error =')

disp(ea)

1. Matlab Code for the False-Position Method:

close all

clear

clc

%False Position Method

%Initial Lower and Upper Bound Guess

xu = 7;

xl = 1;

%Begin loop for 8 iterations

i = 0;

for n = 1:8

i = i+1;

func\_xl = (sqrt(xl)\*sin(xl))-1;

func\_xu = (sqrt(xu)\*sin(xu))-1;

Xr = xl - (func\_xl\*((xu-xl)/(func\_xu - func\_xl)));

func\_Xr = (sqrt(Xr)\*sin(Xr))-1;

if func\_xl\*func\_Xr < 0 %If true, the root lies in the lower sub-interval

xu = Xr;

elseif func\_xl\*func\_Xr > 0 %If true, the root lies in the upper sub-interval

xl = Xr;

else %Only true if func\_xl\*func\_Xr = 0

xroot = Xr;

end

ea = abs((xu-xl)/(xu+xl)) \*100;

end

disp('Final Root =');

disp(Xr);

disp('Approximate Error =')

disp(ea)

1. Matlab Code for Plot:

% Plot the function

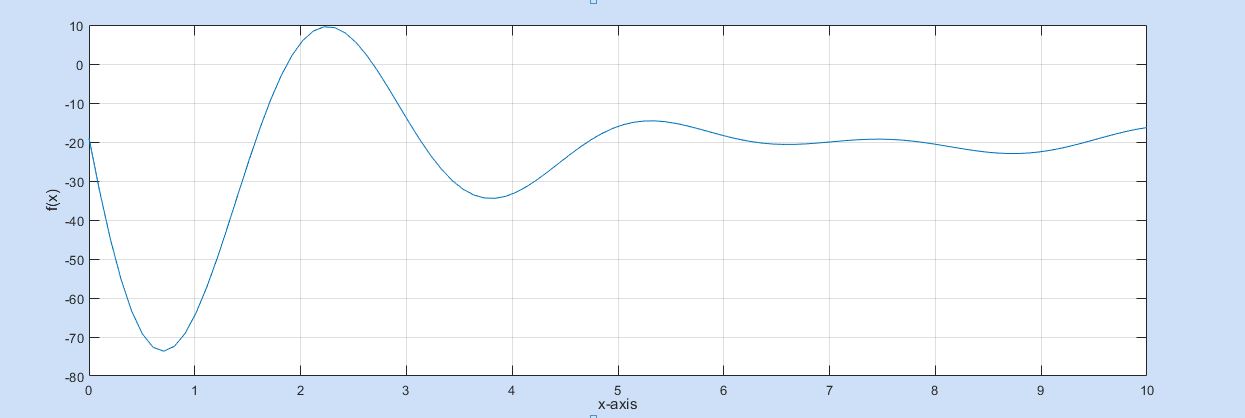
x = linspace(0,8);

y = (sqrt(x).\*sin(x))-1;

plot(x,y)

grid on

**Task 2**

1. The step size that I picked for my incremental search was .1.
2. By analyzing the graph of this function, I determined that there was two positive roots.
3. The roots of this function were 1.8722 and 2.6840.
4. It took 4 iterations for each root to be determined.
5. The Secant Method.
6. Matlab Code for Secant Method:

close all

clear

clc

%Secant Method

%Equation

x = .1;

h = .1;

fx = (20\*exp(-x/5))\*cos(x)\*sin(x)\*(x-7) + erfc(x) - 20;

xi = x + h;

fxi = (20\*exp(-xi/5))\*cos(xi)\*sin(xi)\*(xi-7) + erfc(xi) - 20;

%Incremental Search

while fx\*fxi > 0

fx = (20\*exp(-x/5))\*cos(x)\*sin(x)\*(x-7) + erfc(x) - 20;

h = .1;

xi = x + h;

fxi = (20\*exp(-xi/5))\*cos(xi)\*sin(xi)\*(xi-7) + erfc(xi) - 20;

x = xi;

end

%When x = 1.8, fx = -3.9348

%When x = 1.9, fx = 1.3469

%Initial Guesses

xjj = 1.8; % xj-1

xj= 1.9; % xj

Es = .001; %Stopping Criterion

Ea = abs(((xj - xjj)/(xj))\*100);

i = 0;

while Ea > Es

i = i+1;

fj = (20\*exp(-xj/5))\*cos(xj)\*sin(xj)\*(xj-7) + erfc(xj) - 20;

fjj = (20\*exp(-xjj/5))\*cos(xjj)\*sin(xjj)\*(xjj-7) + erfc(xjj) - 20;

xjjj = xj - ((fj\*(xj-xjj))/(fj - fjj)); %Definition of secant (xj+1)

Ea = abs(((xjjj - xj)/(xjjj))\*100);

xj = xjjj;

end

disp('Root 1 =');

disp(xjjj);

%Start at initial guess of 2 and do the same thing

%Equation

x = 2;

h = .1;

fx = (20\*exp(-x/5))\*cos(x)\*sin(x)\*(x-7) + erfc(x) - 20;

xi = x + h;

fxi = (20\*exp(-xi/5))\*cos(xi)\*sin(xi)\*(xi-7) + erfc(xi) - 20;

%Incremental Search

while fx\*fxi > 0

fx = (20\*exp(-x/5))\*cos(x)\*sin(x)\*(x-7) + erfc(x) - 20;

h = .1;

xi = x + h;

fxi = (20\*exp(-xi/5))\*cos(xi)\*sin(xi)\*(xi-7) + erfc(xi) - 20;

x = xi;

end

%When x = 2.6, fx = 3.1104

%When x = 2.7, fx = -0.6358

%Initial Guesses

xjj = 2.6; % xj-1

xj= 2.7; % xj

Es = .001; %Stopping Criterion

Ea = abs(((xj - xjj)/(xj))\*100);

ii = 0;

while Ea > Es

ii = ii+1;

fj = (20\*exp(-xj/5))\*cos(xj)\*sin(xj)\*(xj-7) + erfc(xj) - 20;

fjj = (20\*exp(-xjj/5))\*cos(xjj)\*sin(xjj)\*(xjj-7) + erfc(xjj) - 20;

xjjj = xj - ((fj\*(xj-xjj))/(fj - fjj)); %Definition of secant (xj+1)

Ea = abs(((xjjj - xj)/(xjjj))\*100);

xj = xjjj;

end

disp('Root 2 =');

disp(xjjj);

1. Matlab Code for Graph:

x = linspace(0,10);

y = (20.\*exp(-x/5)).\*cos(x).\*sin(x).\*(x-7) + erfc(x) - 20;

plot(x,y)

grid on

**Task 3**

1. The following information taken from the command window from my code shows the values of the derivative of the function g(x) for each iteration within the while loop.

gprime =

0.0779

gprime =

0.1390

gprime =

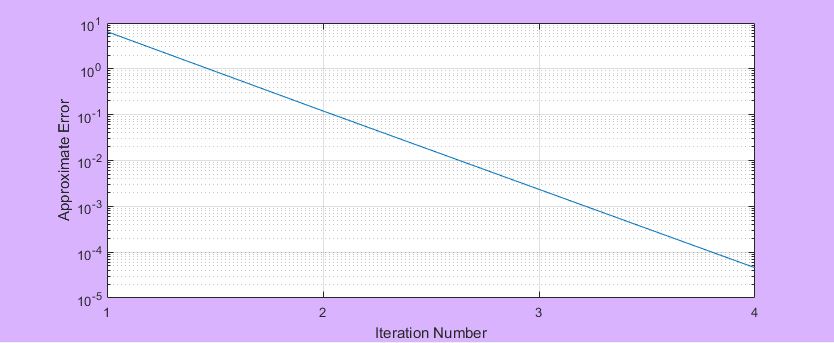
0.1396

gprime =

0.1396

g(x) is linearly converging because the absolute value of gprime is less than one for each iteration.

1. Plot of approximate error versus the iteration number. As seen below, the graph confirms that with each iteration of the while loop, the approximate error is getting smaller and smaller. Therefore concluding that we are approaching an actual value.



1. For the fixed-point method, my code performed 4 iterations to find the root. Whereas for the modified Newton-Raphson method my code performed 6 iterations to find the root.
2. For both root finding methods implemented for this task, my final root was determined to be 0.9685.
3. Matlab Code for Fixed-Point Method:

close all

clear

clc

%Fixed Point Method

%Initial Guess

x0 = 0.1;

%Define stopping criterion

Es = 0.001; %Units in percent

xi = (log(x0^2+6))/2;

xii = (log(xi^2+6))/2;

gprime = abs((xi - xii)/(x0 - xi));

Ea = abs((xii - xi)/(xii))\*100;

i = 0;

while Ea > Es

i = i+1;

xi = (log(x0^2+6))/2;

xii = (log(xi^2+6))/2;

Ea(i) = abs((xii - xi)/(xii))\*100;

gprime = abs((xi - xii)/(x0 - xi))

x0 = xii;

end

disp('g(x) is linearly converging because the absolute value of gprime is less than one.');

disp('Root =');

disp(xii);

semilogy(Ea);

1. Matlab Code for Modified Newton-Raphson Method:

close all

clear

clc

%Modified Newton Raphson Method

%Initial Guess

x0 = 0.1;

%Define stopping criterion

Es = 0.001; %Units in percent

%Define Equations

xi = x0;

fx = xi^2 + 6 - exp(2\*xi);

fxprime = 2\*xi - 2\*exp(2\*xi); % First derivative of function

fxdprime = 2 - 4\*exp(2\*xi); % Second derivative of function

xii = xi - (fx\*fxprime)/((fxprime^2) - (fx\*fxdprime));

Ea = abs((xii - xi)/(xii))\*100;

i = 0;

while Ea > Es

fx = xi^2 + 6 - exp(2\*xi);

fxprime = 2\*xi - 2\*exp(2\*xi); % First derivative of function

fxdprime = 2 - 4\*exp(2\*xi); % Second derivative of function

xii = xi - (fx\*fxprime)/((fxprime^2) - (fx\*fxdprime));

Ea = abs((xii - xi)/(xii))\*100;

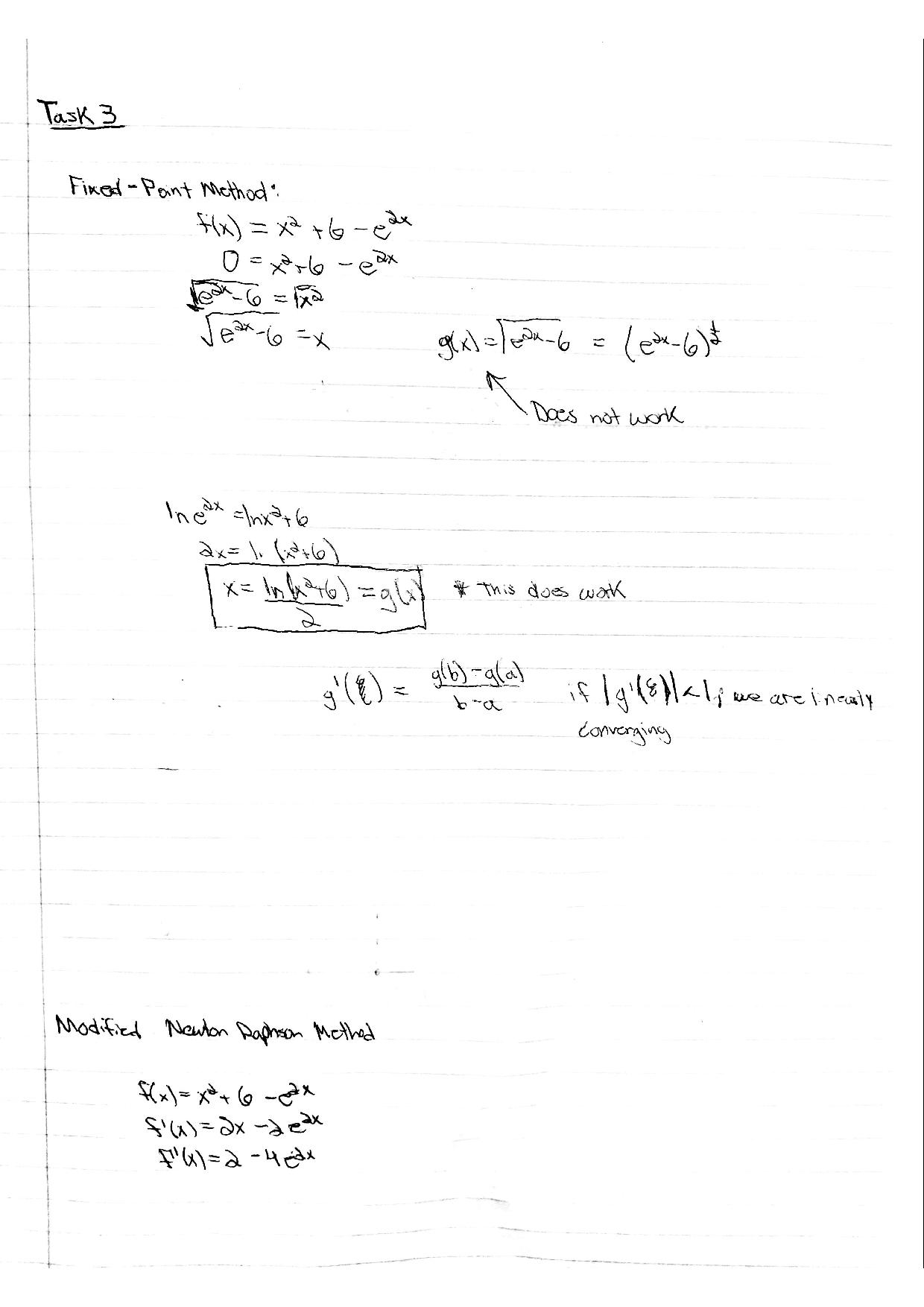
xi = xii;

i = i+1;

end

disp('Root =');

disp(xii);

****

**Task 4**

1. My code was able to identify 5 roots.
2. The roots were 1.5166, 4.8174, 7.6675, 11.3971, and 13.4997.
3. The largest subdivision that can be utilized without missing any of the roots is 1. This was determined experimentally within the code by changing my step size in my incremental search function. From this, I was able to determine that by utilizing a step size of two, I only was able to determine 3 roots while utilizing a step size of 1 determined all 5 roots.
4. The total number of iterations it took to determine each root is displayed below. These values were calculated by adding together the two numbers displayed in the command window prior to each root display.

|  |  |
| --- | --- |
| Root | Number of Iterations |
| 1.5166 | 13 |
| 4.8174 | 16 |
| 7.6675 | 4 |
| 11.3971 | 6 |
| 13.4997 | 4 |

1. Matlab Code for Task 4:

close all

clear

clc

%Incremental Search

i = 0;

for x = 0:1:20

fx = (50 \* exp(-x/5) \* cos(x)) - 2;

h = 1;

xi = x + h;

fxi = (50 \* exp(-xi/5) \* cos(xi)) - 2;

if fx\*fxi < 0

i = i+1;

upper(i) = xi; %This stores my upper bound limits into an array

lower(i) = x; %This stores my lower bound limits into an array

end

end

%Begin Root Finding Methods

%Define Stopping Criterion

Es = 5; %Units in Percent

for r = 1:i

lowerbound = lower(r);

upperbound = upper(r);

Ea = 100; %No error for first iteration, put value at 100 so loop will perform

j=0;

%Begin False Position Method

while Ea > Es

j = j+1;

fxl1 = (50 \* exp(-lowerbound/5) \* cos(lowerbound)) - 2;

fxu1 = (50 \* exp(-upperbound/5) \* cos(upperbound)) - 2;

xr = upperbound - ((fxu1\*(lowerbound-upperbound))/(fxl1 - fxu1));

fxr = (50 \* exp(-xr/5) \* cos(xr)) - 2;

if fxl1 \* fxr > 0

lowerbound = xr; %Root lies in upper sub-interval

elseif fxl1 \* fxr < 0

upperbound = xr; %Root lies in lower sub-interval

else %Only true if fxl1\*fxr = 0

xroot = xr;

end

Ea = abs((upperbound - lowerbound)/(upperbound + lowerbound))\*100;

end

disp('Number of Iterations for False Position Method =');

disp(j);

%Being Modified Secant Method

Esnew = .00005; %Units in percent

x0 = xr;

delta = 0.01;

Eanew = 100;

k = 0;

while Eanew > Esnew

k = k+1;

fx0 = (50 \* exp(-x0/5) \* cos(x0)) - 2;

x1 = x0 + x0\*delta;

fx1 = (50 \* exp(-x1/5) \* cos(x1)) - 2;

x11 = x0 - ((delta\*fx0\*x0)/(fx1-fx0));

Eanew = abs((x11 - x0)/(x11))\*100;

x0 = x11;

end

disp('Number of Iterations for the Modified Secant Method =');

disp(k);

disp('Root =')

disp(x11);

end