

UWTSD

Alternative Fuels and Combustion Modelling

Task 2

ACHA7002

2515979

07/12/25

Contents

1	Bisection	1
1.1	$fx = \sin x \sinh(x)$	1
1.2	$fx = R^2 - x^{21/2} - x \tan(x)$	1
2	Newton's method	2
2.1	Newton b $fx = ex - (1 + x + (x^2/2))$	2
2.2	Newton c $fx = 10 \log x - x$	2
3	Intersect plot $fx = 100e - x - 5 \sin(\pi x/2)$	3
4	Bracketing	3
4.1	Bisect $fx = 100e - x - 5 \sin(\pi x/2)$	3
4.2	Newton $fx = 100e - x - 5 \sin(\pi x/2)$	4
5	Appendices	4
5.1	Convergence table Bisection $fx = \sin x \sinh(x)$	4
5.2	Convergence table Bisection $fx = R^2 - x^{21/2} - x \tan(x)$	5
5.3	Convergence table Newton b $fx = ex - (1 + x + (x^2/2))$	5
5.4	Convergence table Newton c $fx = 10 \log x - x$	5
5.5	Convergence table Bisect $fx = 100e - x - 5 \sin(\pi x/2)$	6
5.6	Convergence table Newton $fx = 100e - x - 5 \sin(\pi x/2)$	6
5.7	Your code and functions	6

1 Bisection

A function was created based on equation 1.1 to solve roots using the bisection method

$$x_2 = (x_1 + x_3)/2 \quad \text{Equation 1-1}$$

1.1 $f(x) = \sin(x) \sinh(x)$

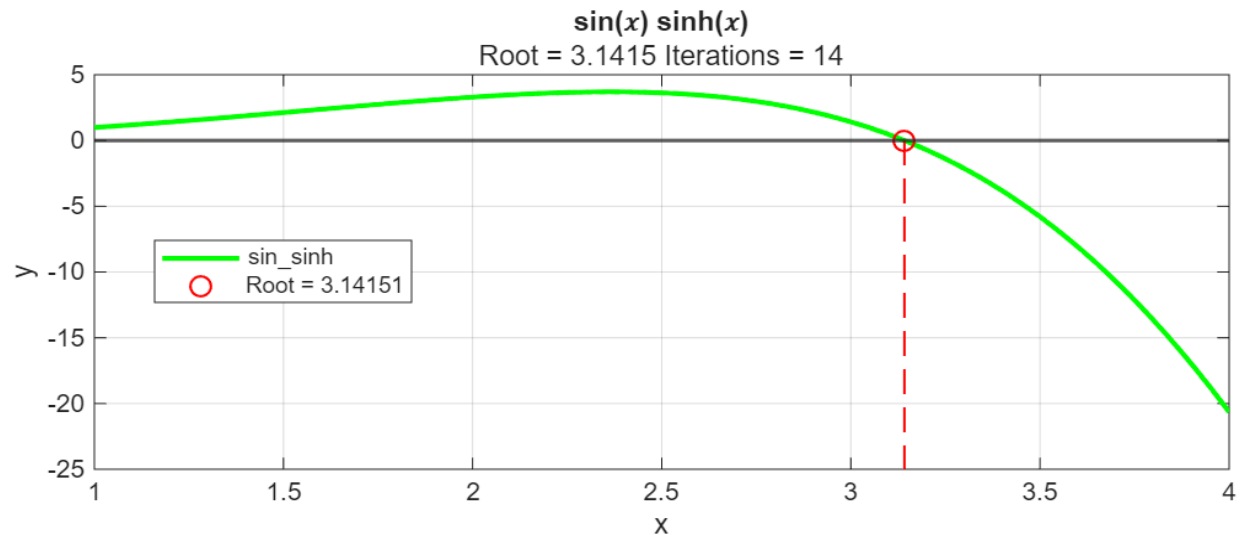


Figure 1-1 this shows the function labelled in the legend and the root

1.2 $f(x) = (R^2 - x^2)^{1/2} - x \tan(x)$

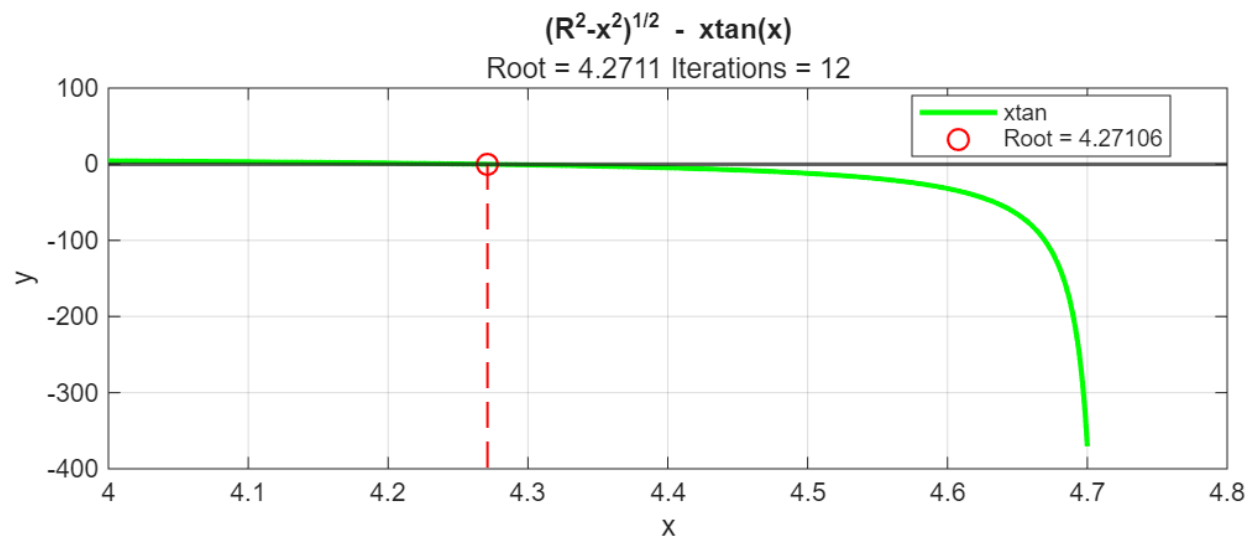


Figure 1-2 this shows the function labelled in the legend and the root

2 Newton's method

A function was made from equation 2.1 to evaluate roots with the newton method to use this method, functions and their derivative functions were created.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{Equation 2-1}$$

2.1 Newton b $f(x) = e^x - (1 + x + (x^2/2))$

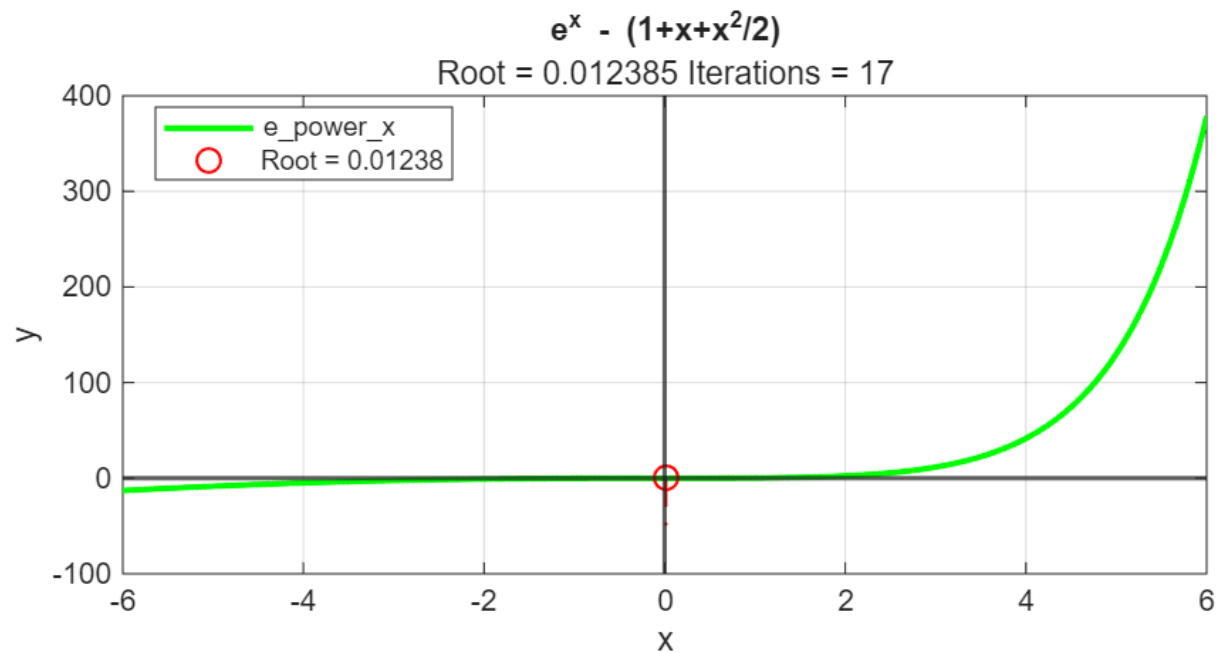


Figure 2-1 this shows the function labelled in the legend and the root

2.2 Newton c $f(x) = 10 \log(x) - x$

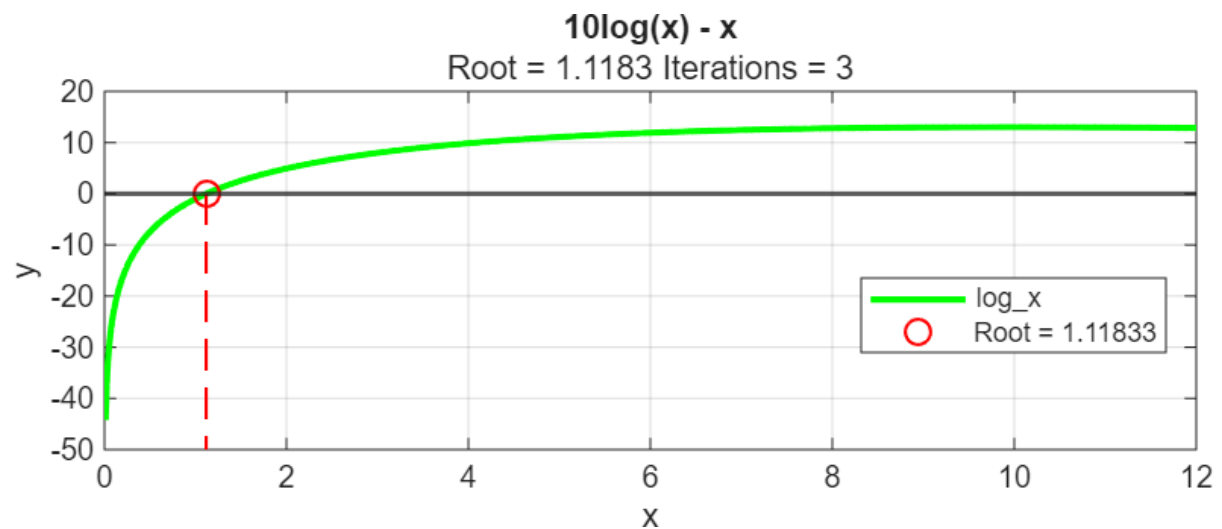


Figure 2-2 this shows the function labelled in the legend and the root

3 Intersect plot $f(x) = 100e^{-x} - 5\sin(\pi x/2)$

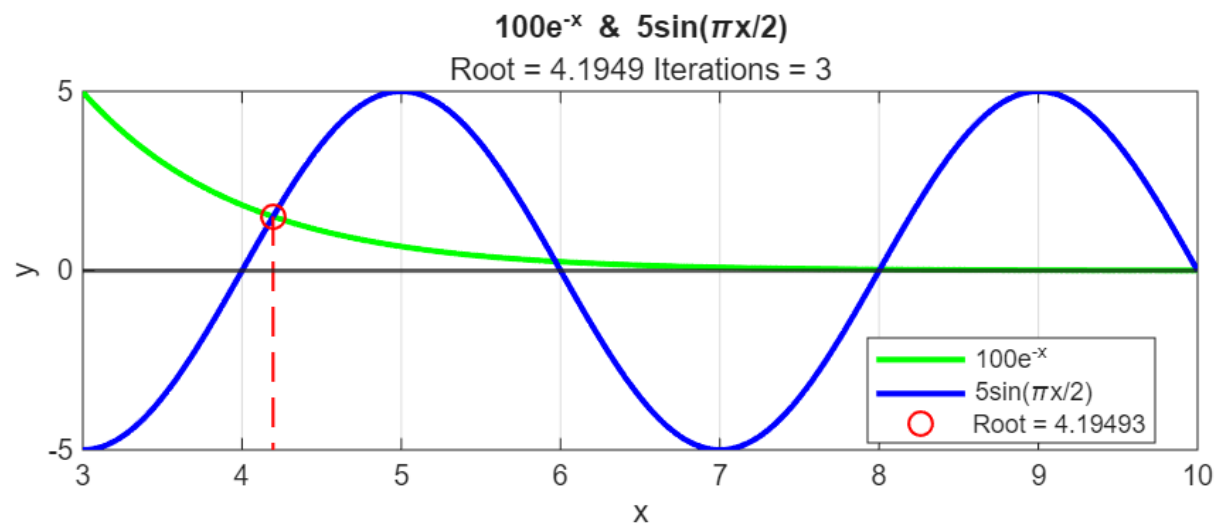


Figure 3-1 plot of function intersection

4 Bracketing

4.1 Bisect $f(x) = 100e^{-x} - 5\sin(\pi x/2)$

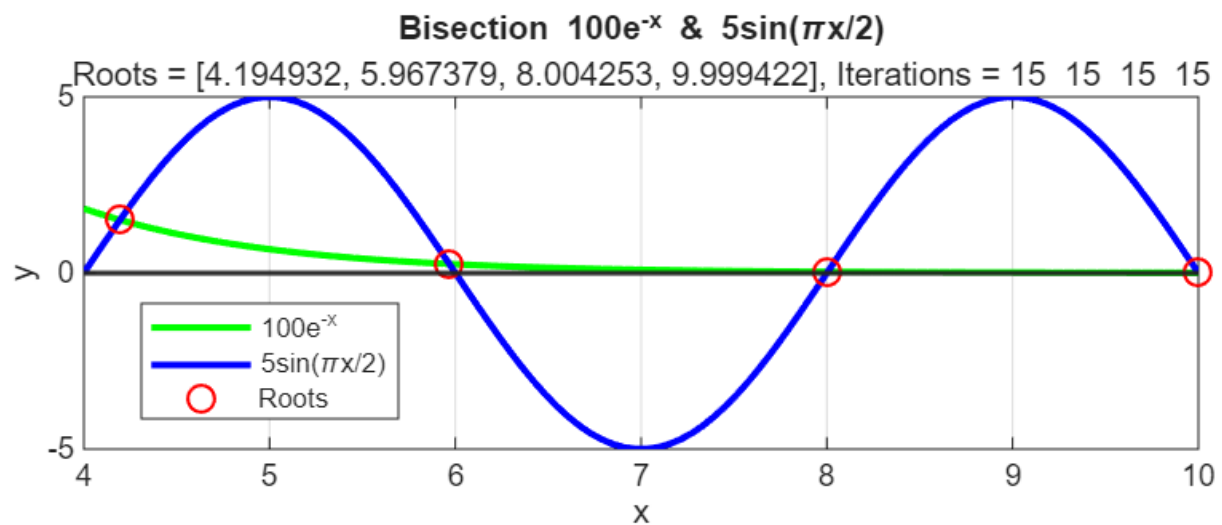


Figure 4-1 multiple roots Bisect for functions shown in legend

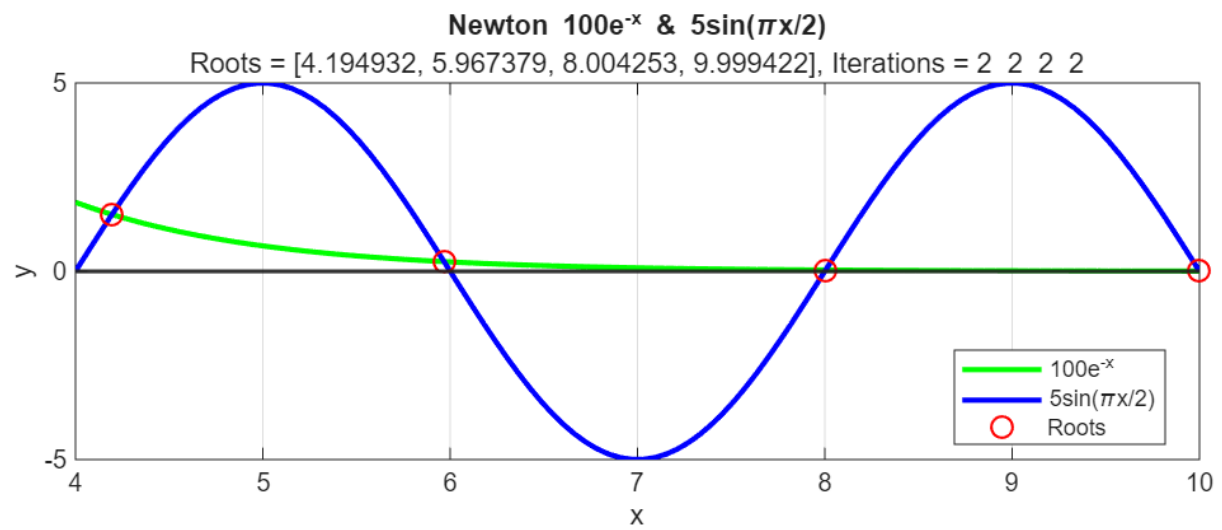
4.2 Newton $f(x) = 100e^{-x} - 5\sin(\pi x/2)$ 

Figure 4-2 multiple roots Newton for functions shown in legend

5 Appendices

5.1 Convergence table Bisection $f(x) = \sin(x) \sinh(x)$

Sin_Sinh	
Iteration	Mid-Point
1	3.25
2	2.875
3	3.0625
4	3.15625
5	3.109375
6	3.132812
7	3.144531
8	3.138672
9	3.141602
10	3.140137
11	3.140869
12	3.141235
13	3.141418
14	3.14151

5.2 Convergence table Bisection $f(x) = (R^2 - x^2)^{1/2} - x \tan(x)$

xtan	
Iteration	Mid-Point
1	4.375
2	4.3125
3	4.28125
4	4.265625
5	4.273438
6	4.269531
7	4.271484
8	4.270508
9	4.270996
10	4.27124
11	4.271118
12	4.271057

5.3 Convergence table Newton b $f(x) = e^x - (1 + x + (x^2/2))$

e^x	
Iteration	Output
1	4.087773
2	3.24102
3	2.487381
4	1.849522
5	1.337187
6	0.944907
7	0.656151
8	0.449883
9	0.305706
10	0.206451
11	0.138834
12	0.093096
13	0.062307
14	0.041646
15	0.027812
16	0.018563
17	0.012385

5.4 Convergence table Newton c $f(x) = 10 \log(x) - x$

log(x)	
Iteration	Output
1	1.111111
2	1.118299
3	1.118326

5.5 Convergence table Bisect $f(x) = 100e^{-x} - 5\sin(\pi x/2)$

Bisect				
Iteration	Output 1	Output 2	Output 3	Output 4
1	4.19697	5.954545	8.015152	9.984848
2	4.189394	5.962121	8.007576	9.992424
3	4.193182	5.965909	8.003788	9.996212
4	4.195076	5.967803	8.005682	9.998106
5	4.194129	5.966856	8.004735	9.999053
6	4.194602	5.96733	8.004261	9.999527
7	4.194839	5.967566	8.004025	9.99929
8	4.194957	5.967448	8.004143	9.999408
9	4.194898	5.967389	8.004202	9.999467
10	4.194928	5.967359	8.004232	9.999438
11	4.194943	5.967374	8.004247	9.999423
12	4.194935	5.967381	8.004254	9.999416
13	4.194931	5.967378	8.00425	9.999419
14	4.194933	5.967379	8.004252	9.999421
15	4.194932	5.967379	8.004253	9.999422

5.6 Convergence table Newton $f(x) = 100e^{-x} - 5\sin(\pi x/2)$

Newton				
Iteration	Output 1	Output 2	Output 3	Output 4
1	4.194842	5.967379	8.004234	9.999444
2	4.194932	5.967379	8.004253	9.999422

5.7 Your code and functions

5.7.1 Call Script

```
%% ---House Keeping---
clc; clear; close all;

%% ---Bisection Method---
%% Sin_Sinh Test ---ALL WORKING
x = linspace(1,4,1000);
y1 = sin_sinh(x);

x1 = 1;
x2 = 4;
imax = 30;
tol = 1e-4;

[Root,Cnt] = bisect('sin_sinh',x1,x2,imax,tol);
Cnt = Cnt-1; % added this line as the bisections cause they show as an extra
iteration than what they actually done as it still adds 1 to the counter after
finding the root
fprintf('Root ≈ %.10f after %d iterations (tol = %g)\n', Root, Cnt, tol);

figure;
plot(x,y1,'g',LineWidth=2);
grid on; hold on;
plot(Root,sin_sinh(Root),'ro',LineWidth=1,MarkerSize=8);
```

```

yline(0,LineWidth=1.5);
ylo=-25;
plot([Root Root],[0 ylo],'r--',LineWidth=1);
legend('sin\sinh',sprintf('Root = %.5f', Root),'Location', 'best');
xlabel("x");
ylabel("y");
title('sin(x) sinh(x)');
subtitle(['Root = ' num2str(Root), ' Iterations = ' num2str(Cnt)]); % fix this to
add count
hold off;

%% xtan Test ---ALL WORKING
x = linspace(4,4.7,1000);
y2 = xtan(x);

x1 = 4;
x2 = 4.5;
imax = 30;
tol = 1e-4;

[Root2, Cnt2] = bisect('xtan',x1,x2,imax,tol);
Cnt2 = Cnt2-1;
fprintf('Root ≈ %.10f after %d iterations (tol = %g)\n', Root2, Cnt2, tol);

figure(2);
plot(x,y2,"g",LineWidth=2);
grid on; hold on;
plot(Root2,xtan(Root2),'ro',LineWidth=1,MarkerSize=8);
yline(0,LineWidth=1.5);
ylo2=-400;
plot([Root2 Root2],[0 ylo2],'r--',LineWidth=1);
legend('xtan',sprintf('Root = %.5f', Root2),'Location', 'best');
xlabel("x");
ylabel("y");
title('(R^2-x^2)^(1/2) - xtan(x)');
subtitle(['Root = ' num2str(Root2), ' Iterations = ' num2str(Cnt2)]); % fix this to
add count
hold off;

%% ---Newton Method---
%% e_power_x Test ---ALL WORKING
x = linspace(-6,6,1000);
y3 = e_power_x(x);

x0 = 5;
imax = 20;
tol = 1e-6;

[Root3,Cnt3] = newton(x0,'e_power_x','dy_dx_e_power_x',imax,tol);
fprintf('Root ≈ %.10f after %d iterations (tol = %g)\n', Root3, Cnt3, tol);

figure(3);
plot(x,y3,"g",LineWidth=2);
grid on; hold on;
plot(Root3,e_power_x(Root3),'ro',LineWidth=1,MarkerSize=8);
yline(0,LineWidth=1.5);
xline(0,LineWidth=1.5);
ylo3=-50;
plot([Root3 Root3],[0 ylo3],'r--',LineWidth=1);

```

```

legend('e\power\x',sprintf('Root = %.5f', Root3),'Location','best');
xlabel("x");
ylabel("y");
title('e^x - (1+x+x^2/2)');
subtitle(['Root = ' num2str(Root3), ' Iterations = ' num2str(Cnt3)]); % fix this to
add count
hold off;

%% log_x Test ---ALL WORKING
x = linspace(0,12,1000);
y4 = log_x(x);

x0 = 1;
imax = 20;
tol = 1e-6;

[Root4,Cnt4] = newton(x0,'log_x','dy_dx_log_x',imax,tol);
fprintf('Root ≈ %.10f after %d iterations (tol = %g)\n', Root4, Cnt4, tol);

figure(4);
plot(x,y4,"g",LineWidth=2);
grid on; hold on;
plot(Root4,log_x(Root4),'ro',LineWidth=1,MarkerSize=8);
yline(0,LineWidth=1.5);
ylow4=-50;
plot([Root4 Root4],[0 ylow4],'r--',LineWidth=1);
legend('log\x',sprintf('Root = %.5f', Root4),'Location','best');
xlabel("x");
ylabel("y");
title('10log(x) - x');
subtitle(['Root = ' num2str(Root4), ' Iterations = ' num2str(Cnt4)]); % fix this to
add count
hold off;

% NOTES FOR WORK: Try create a function that will replace the f(x)/f'(x)
% parts, this may speed up the process and affect results

%% ---Newton Intersection Method--- ALL WORKING
clc; clear; clear -all;

x = linspace(3,10,1000);
y5_Left = 100*exp(-x);
y5_Right = 5*sin(pi*x/2);

x0 = 4;
imax = 20;
tol = 1e-6;

[Root5_x,Cnt5] = newton(x0,'e_and_sin','dy_dx_e_and_sin',imax,tol);
fprintf('Root ≈ %.10f after %d iterations (tol = %g)\n', Root5_x, Cnt5, tol);

Root5_y = 100*exp(-Root5_x); % Since LHS=RHS, Root5_y could also be calculated
with 5*sin(pi*x/2)

figure(5);
plot(x,y5_Left,"g",LineWidth=2); % Plot of LHS
grid on; hold on;
plot(x,y5_Right,"b",LineWidth=2); % Plot of RHS

```

```

plot(Root5_x,Root5_y,"ro",LineWidth=1,MarkerSize=8); % Plot of root

yline(0,LineWidth=1.5); % Adds y-axis line

ylow5=-5; % Indicates the lower limit displayed
plot([Root5_x Root5_x],[Root5_y ylow5],'r--',LineWidth=1); % Plots a dashed line
to the bottom of graph

xlabel("x");
ylabel("y");
legend('100e^{-x}', '5sin(\pix/2)', sprintf('Root = %.5f',
Root5_x), 'Location', 'best');
title('100e^{-x} & 5sin(\pix/2)');
subtitle(['Root = ' num2str(Root5_x), ' Iterations = ' num2str(Cnt5)]); % fix this
to add count
hold off;

% Increased Accuracy
tol_c = 1e-10;

[Root6, Cnt6] = newton(x0, 'e_and_sin', 'dy_dx_e_and_sin',imax,tol_c);

fprintf('Part (c): Root ≈ %.12f after %d iterations (tol = %g)\n',Root6, Cnt6,
tol_c);

% NOTES FOR WORK: need to analyse and comment on part c (increased accuracy)

%% ---Bracketing---
clc; clear; clear -all;
%% Bisection

% Parameters
x = linspace(4,10,100);
f = 'e_and_sin';
df = 'dy_dx_e_and_sin';
tol = 1e-6;
imax = 100;
y6_Left = 100*exp(-x);
y6_Right = 5*sin(pi*x/2);

% Part a&b Calculate
[Roots, Cnt7] = Multiple_Roots_Bisection(x,f,df,imax,tol); % I first wrote everything
here then moved to its own dedicated function to tidy Testing workspace

% Part C - Plot
figure(6);
plot(x,y6_Left,"g",LineWidth=2); % Plot of LHS
grid on; hold on;
plot(x,y6_Right,"b",LineWidth=2); % Plot of RHS

for k = 1:length(Roots)
    xr = Roots(k);
    yr = 100*exp(-xr);

    plot(xr,yr,"ro",LineWidth=1,MarkerSize=8)
end

root_str = sprintf('%.6f, ', Roots);

```

```

root_str = root_str(1:end-2);
Cnt7 = Cnt7-1;

xlabel("x")
ylabel("y")
yline(0,LineWidth=1.5);
legend("100e^{-x}", "5sin(\pix/2)", "Roots", "Location", "best");
title("Bisection 100e^{-x} & 5sin(\pix/2)")
subtitle(['Roots = [' root_str '], Iterations = ' num2str(Cnt7)]);

% NOTE FOR WORK:

%% Newton
% Part a&b Calculate
[Roots, Cnt8] = Multiple_Roots_Newton(x,f,df,imax,tol); % I first wrote everything
here then moved to its own dedicated function to tidy Testing workspace

% Part C - Plot
figure(7);
plot(x,y6_Left,"g",LineWidth=2); % Plot of LHS
grid on; hold on;
plot(x,y6_Right,"b",LineWidth=2); % Plot of RHS

for k = 1:length(Roots)
    xr = Roots(k);
    yr = 100*exp(-xr);

    plot(xr,yr,"ro",LineWidth=1,MarkerSize=8)
end

root_str = sprintf('%.6f, ', Roots);
root_str = root_str(1:end-2);
Cnt8 = Cnt8-1;

xlabel("x")
ylabel("y")
yline(0,LineWidth=1.5);
legend("100e^{-x}", "5sin(\pix/2)", "Roots", "Location", "best");
title("Newton 100e^{-x} & 5sin(\pix/2)")
subtitle(['Roots = [' root_str '], Iterations = ' num2str(Cnt8)]);

```

5.7.2 Bisection Function

```

function [Root, Count] = bisect(f,x1,x2,imax,tol)

% Inputs:
% f = function to be evaluated (the one were finding the roots for)
% a,b = interval limits (our guesses)
% imax = iteration maximum
% tol = convergence criterion

% Output:
% Root = position of the root
% counter = number of iterations required

% Initial Guesses
f1=feval(f,x1); % x1 = left hand guess

```

```

f2=feval(f,x2); % x2 = Right hand guess

% ---Root Checks---

if f1 == 0 % If f1=0, then that means the root is exactly on the left guess
    Root = x1; % Sets root equal to left guess
    Count = 0; % Sets counter equal to zero since no iterations were needed
    fprintf("Exact root at left guess: x = %.8f\n", Root); % Displays where the
root is
    return;
end

if f2 == 0 % If f2=0, then that means the root is exactly on the Right guess
    Root = x2; % Sets root equal to right guess
    Count = 0; % Sets counter equal to zero since no iterations were needed
    fprintf("Exact root at Right guess: x = %.8f\n", Root); % Displays where the
root is
    return;
end

if f1*f2>0 % If the product of f1 & f2 is positive, then that means there is no
root between the left and right guesses
    error('No root in the specified interval') % If root is not between guesses,
this error is displayed

end

% Initial Parameters
Count=0; % Sets up counter

while (abs(x2-x1)>tol && Count<imax) % Runs until either the root is found within
the tolerance selected OR the max number of iterations (imax) is exceeded

    Mid_Point=(x1+x2)/2; % xMid_Point = the average (mid-point) of x1 & x2
    f3=feval(f,Mid_Point); % fn is basically the y value at xn (i.e f1 is y at x1,
f2 is y at x2, fMid_Point is y at Mid_Point)

    % Output to screen
    fprintf('ITERATION %i \r',Count)
    fprintf('Lower guess = %10.6f \r',x1)
    fprintf('Upper guess = %10.6f \r',x2)
    fprintf('Mid-point = %10.6f \r \r',Mid_Point)

    % ---Mid_Point decision---
    if f1*f3<0 % i.e = -ve, root is in left half

        x2=Mid_Point; % Since root is between x1 & Mid_Point, the new x2 becomes
Mid_Point

    else % Root is in right half, so

        x1=Mid_Point; % Since root is between x2 & Mid_Point, the new x1 becomes
Mid_Point

    end

    Count=Count+1; % Adds 1 to the counter

end

```

```
% ---Final Estimate Display---
Root=Mid_Point;
fprintf('FINAL ESTIMATE OF THE ROOT = %10.6f \r \n',Root)

if Count == imax && (abs(x2 - x1) > tol)
    warning('Bisection reached imax without meeting tolerance.');
```

5.7.3 Sin Sinh Function

```
function [y] = sin_sinh(x)

y = sin(x).*sinh(x);

end
```

5.7.4 Xtan Function

```
function [y] = xtan(x)

y = sqrt((10.^2)-(x.^2))-(x.*tan(x));

end
```

5.7.5 Newton Function

```
function [Root, counter] = newton(x0,f,df,imax,tol)

% Inputs:
% x = initial guess at the root
% f = function to be evaluated
% df = derivative of f with respect to x
% imax = iteration maximum
% tol = convergence criterion

% Output:
% Root = position of the root

% Newton's algorithm
% x = x + dx
% dx = -fx(x)/dfx(x)
% NB: both f and df are user coded functions - you will need to write functions
for both!!!
% initialise parameters, i.e. x=x0; fx=feval(f,x); %etc...

x = x0;
fx = feval(f,x);
counter = 0;

fprintf('Counter\t x\t\t f(x)\n');
fprintf('%d\t %.8f\t %.3e\n',counter,x,fx);

while (abs(fx)>tol && counter<imax)
```

```

    dfx = feval(df,x); % The derivative at current x value

    % CHECK: verify that the derivative is not zero.
    if df == 0
        error('Derivative Zero - Fail')
    end

    x = x-fx/dfx;
    fx = feval(f,x); % Updates f(x)
    counter = counter+1; % Adds 1 to counter

    fprintf('%d\t %.8f\t %.3e\n',counter,x,fx);
end

if abs(fx)>tol
    warning('Did not converge within imax');
end

Root = x;

end

```

5.7.6 e^x Function

```

function [y] = e_power_x(x)

y = exp(x)-(1+x+((x.^2)/2)); %exp(x) = e^x

end

```

5.7.7 $dy/dx e^x$ Function

```

function [dy] = dy_dx_e_power_x(x)

dy = exp(x)-(1+x);

end

```

5.7.8 $\log(x)$ Function

```

function [y] = log_x(x)

y = 10*log(x)-x; % Natural log

end

```

5.7.9 $dy/dx \log(x)$

```

function [dy] = dy_dx_log_x(x)

dy = (10/x)-1; % Assuming natural log

```

end

5.7.10 Bracketing Bisection Function

```
function [Clean_Roots, Count] = Multiple_Roots_Bisect(x,f,df,imax,tol)

% Parameters
fx = feval(f,x);

% Part a - Bracketing
Brackets = []; % This stores rows as [a b], these are the brackets (there is a sign
change inside so therefore a root)

for k = 1:length(x)-1 % Loops from the first to the penultimate x, this is to check
pairs for a sign change that indicates a root
    a = x(k); % left bracket
    b = x(k+1); % Right bracket
    fa = fx(k);
    fb = fx(k+1);

    if fa == 0 % This indicates a root in exactly one of a
        Brackets = [Brackets; a a]; % ok<AGROW> % a is the interval, since the
root is in one a, this means the use of bisection or newton may be needed
% the %ok<AGROW> gets rid of the growing array warning message
    elseif fa*fb < 0 % Product has a negative sign
        Brackets = [Brackets; a b]; % ok<AGROW> % a b is the interval, root is
between a & b

    end
end

if isempty(Brackets) % This happens when fa*fb is > 0 (Positive)
    warning('No root within interval') % If product is positive, there is not a
root within the given interval
end

% Brackets Display
fprintf('Found %d Brackets:\n', size(Brackets,1)); % Shows the number of brackets
n displays them
disp(Brackets);

% Part b - Solvers
Roots_Bisect = []; % These are just prep arrays for storing values of the roots and
iteration counts for both
Count_Bisect = [];

for i = 1:size(Brackets,1) % Loops through each of the brackets
    a = Brackets(i,1); % Pulls out the left bracket
    b = Brackets(i,2); % Pulls out the right bracket

    if a == b % Root is both a&b cause they're the same value
        Root_b = a; % Sets root to be at a, could also set to be =b, doesn't
matter as they're both the same
        Count_b = 0; % Count equals 0 since no solver was run
    else
        [Root_b, Count_b] = bisect(f,a,b,imax,tol); % Runs Bisect method
    end
end
```

```

    Roots_Bisect(end+1) = Root_b; %#ok<AGROW> %Stores results as the roots
    Count_Bisect(end+1) = Count_b; %#ok<AGROW> % Stores results as the iteration
    counts
end

Count = Count_Bisect;
All_Roots = Roots_Bisect; % This combines both as the solvers results then deletes
any EXACT duplicates (doesn't always work cause each one might have a slight
difference i.e. newton=2.3541 and bisection=2.35421)

Clean_Roots = []; % Creates an array for all the clean roots
for Refined_Roots = All_Roots % This loop goes through all the roots and only adds
them to the clean list if: 1-the list is empty, OR 2-the roots differ by more
than the tolerance(tol), this sorts the issue as the non EXACT duplicates
    if isempty(Clean_Roots) || all(abs(Clean_Roots-Refined_Roots)>tol)
        Clean_Roots(end+1) = Refined_Roots; %#ok<AGROW>
    end
end

% Display
fprintf('\nRefined roots (within tol = %g):\n', tol); % Displays all the results
for k = 1:length(Clean_Roots) % Loops through all the roots and prints them in the
form
    Refined_Roots = Clean_Roots(k);
    fprintf('Root %d: x = %.10f, f(x)=%.3e\n', k, Refined_Roots, feval(f,
Refined_Roots)); % K=root number (i.e. 1,2,3,...)
end

end
end

```

5.7.11 Bracketing Newton Function

```

function [Clean_Roots, Count] = Multiple_Roots_Newton(x,f,df,imax,tol)

% Parameters
fx = feval(f,x);

% Part a - Bracketing
Brackets = []; % This stores rows as [a b], these are the brackets (there is a sign
change inside so therefore a root)

for k = 1:length(x)-1 % Loops from the first to the penultimate x, this is to check
pairs for a sign change that indicates a root
    a = x(k); % left bracket
    b = x(k+1); % Right bracket
    fa = fx(k);
    fb = fx(k+1);

    if fa == 0 % This indicates a root is exactly on a
        Brackets = [Brackets; a a]; %#ok<AGROW> % a is the interval, since the
root is on a, this means the use of bisection or newton is not needed
% the %#ok<AGROW> gets rid of the growing array warning message
    elseif fa*fb < 0 % Product has a negative sign
        Brackets = [Brackets; a b]; %#ok<AGROW> % a b is the interval, root is
between a & b
    end
end

```

```

end

if isempty(Brackets) % This happens when fa*fb is > 0 (Positive)
    warning('No root within interval') % If product is positive, there isnae a
    root within the given interval
end

% Brackets Display
fprintf('Found %d Brackets:\n', size(Brackets,1)); % Shows the number ae brackets
n dispays them
disp(Brackets);

% Part b - Solvers
Roots_Newton = []; % These aw just prep arays fur storing values ae the roots n
iteration counts fur both
Count_Newton = [];

for i = 1:size(Brackets,1) % Loops throught each ae the brackets
    a = Brackets(i,1); % Pulls oot the left bracket
    b = Brackets(i,2); % Pulls oot the left bracket

    if a == b % Root is both a&b cause theyre the same value
        Root_n = a; % Sets root tae be at a, could also set to be =b, disnae
        matter as theyre both the same
        Count_n = 0; % Count equals 0 since nae solver was run
    else

        % Newton starts at midpoint
        x0 = (a+b)/2; % Average of a & b (midpoint of bracket), this is the initial
        guess set fur newton

        [Root_n, Count_n] = newton(x0,f,df,imax,tol); % Runs Newton method
    end

    Roots_Newton(end+1) = Root_n; %#ok<AGROW> % Stores results ae the roots
    Count_Newton(end+1) = Count_n; %#ok<AGROW> % Stores results ae the iteration
counts
end

Count = Count_Newton;
All_Roots = Roots_Newton; % This combines both ae the solvers results then deletes
any EXACT duplicates (disnae always work cause each wan might hiv a slight
difference i.e. newton=2.3541 and bisection=2.35421)

Clean_Roots = []; % Creates an array fur aw the clean roots
for Refined_Roots = All_Roots % This loop goes through aw the roots n only adds
them tae the clean list if: 1-the list is empty, OR 2-the roots differ by mare
than the tolerance(tol), this sorts the issue ae the non EXACT duplicates
    if isempty(Clean_Roots) || all(abs(Clean_Roots-Refined_Roots)>tol)
        Clean_Roots(end+1) = Refined_Roots; %#ok<AGROW>
    end
end

% Display
fprintf('\nRefined roots (within tol = %g):\n', tol); % Displays aw the results
for k = 1:length(Clean_Roots) % Loops through aw the roots and prints them in the
form
    Refined_Roots = Clean_Roots(k);

```

```
    fprintf('Root %d: x = %.10f, f(x)=%.3e\n', k, Refined_Roots, feval(f,  
Refined_Roots)); % K=root number (i.e. 1,2,3,...)
```

```
end
```

```
end
```