



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Task 2

Module code: ACHA7002

Student Number: 2410890

Date: 7-12-2025

Ryan, I think I cooked up in task 2, and I completely ruined it by myself and I think I don't have much time to recreate it. I submit it just codes.

I apologise in advance

1 Bisection

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

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

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

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

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.

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

$$2.1 \quad \text{Newton b } f(x) = e^x - (1 + x + (x^2/2))$$

$$2.2 \quad \text{Newton c } f(x) = 10 \log(x) - x$$

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

Bracketing

$$3.1 \quad \text{Bisect } f(x) = 100e^{-x} - 5\sin(\pi x/2)$$

$$3.2 \quad \text{Newton } f(x) = 100e^{-x} - 5\sin(\pi x/2)$$

4 Appendices

4.1 Your code and functions

bisection method

```
function root = bisect(f, a, b, imax, tol)
```

```
% Input: f - function handle
```

```
%      a, b - interval limits
```

```
%      imax - maximum number of iterations
```

```
%      tol - convergence criterion
```

```
% Output: root - the position of the root
```

```
% Initialize parameters
```

```
x1 = a;
```

```
x3 = b;
```

```
f1 = feval(f, x1);
```

```
f3 = feval(f, x3);
```

```
% Verify a root exists in the interval
```

```
if f1 * f3 > 0
```

```
    error('No root in the specified interval');
```

```
end
```

```
% Iteration loop
```

```
for ndx = 1:imax
```

```
    % Midpoint
```

```
x2 = (x1 + x3) / 2;  
f2 = feval(f, x2);  
  
% Check for convergence  
if abs(f2) < tol || abs(x3 - x1) / 2 < tol  
    root = x2;  
    return;  
end  
  
% Determine which subinterval contains the root  
if f1 * f2 < 0  
    x3 = x2;  
    f3 = f2;  
else  
    x1 = x2;  
    f1 = f2;  
end  
end  
  
% If maximum iterations reached without convergence  
error('Exceeded maximum number of iterations');  
end
```

```
function y = func_b(x)
```

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

```
end
```

```
function y = func_c(x)
```

```
    R = 10;
```

```
    y = sin(x) - R ./ x;
```

```
end
```

```
% Main script
```

```
clear; clc;
```

```
% Parameters
```

```
tol = 1e-6;
```

```
imax = 50;
```

```
% Function (b):  $f(x) = \sin(x) * \sinh(x)$ 
```

```
a_b = 1;
```

```
b_b = 4;
```

```
root_b = bisect(@func_b, a_b, b_b, imax, tol);
```

```
fprintf('Root of  $f(x) = \sin(x) * \sinh(x)$  in [%f, %f]: %f\n', a_b, b_b, root_b);
```

```
% Test function evaluations
```

```
fprintf('f(1) = %f, f(4) = %f\n', func_b(1), func_b(4));
```

```
fprintf('f(4) = %f, f(4.7) = %f\n', func_c(4), func_c(4.7));
```

```
% Plot func_b

x_b = linspace(1, 4, 1000);

y_b = func_b(x_b);

figure(1);

plot(x_b, y_b, 'b');

hold on;

yline(0, '--k');

title('f(x) = sin(x) * sinh(x)');

xlabel('x');

ylabel('f(x)');

grid on;
```

```
% Plot func_c

x_c = linspace(4, 4.7, 1000);

y_c = func_c(x_c);

figure(2);

plot(x_c, y_c, 'b');

hold on;

yline(0, '--k');

title('f(x) = sin(x) - R/x');

xlabel('x');

ylabel('f(x)');
```

grid on;

% Main script for Newton's Method and plotting

clear; clc;

% Define functions for (b) and (c) and their derivatives

func_d = @(x) exp(x) - (1 + x + x.^2 / 2);

dfunc_b = @(x) exp(x) - (1 + x);

func_e = @(x) 10 * log(x) - x;

dfunc_c = @(x) 10 ./ x - 1;

% Newton's Method implementation

newton = @(x0, f, df, imax, tol) newton_method(x0, f, df, imax, tol);

% Parameters

tol = 1e-6;

imax = 20;

% Function (b): $f(x) = \exp(x) - (1 + x + x^2/2)$

x_b = linspace(4, 6, 1000);

y_c = func_b(x_b);

figure(1);

```

%plot(x_b, y_b, 'b', 0, func_b(0), 'root', 'MarkerSize', 10, 'LineWidth', 2);

grid on; grid minor;

title(['Root of f(x) = e^x - (1 + x + x^2/2): ', num2str('root_b', '%.6f')]);

xlabel('x (unit)'); ylabel('f(x) (unit)');

legend('f(x)', 'Root', 'Location', 'Best'); axis tight;

% Function (c): f(x) = 10*log(x) - x

x0_c = 1; % Initial guess

%root_c = newton(x0_c, func_c, dfunc_c, imax, tol);

fprintf('Root of f(x) = 10*log(x) - x: %.6f\n', 'root_c');

% Plot for function (b)

x_b = linspace(4, 6, 1000); % Adjust interval based on root location

y_b = func_b(x_b);

figure(1);

%plot(x_b, y_b, 'b', root_b, func_b(root_b), 'ro', 'MarkerSize', 10, 'LineWidth', 2);

grid on; grid minor;

%title(['Root of f(x) = e^x - (1 + x + x^2/2): ', num2str(root_b, '%.6f')]);

xlabel('x (unit)');

ylabel('f(x) (unit)');

legend('f(x)', 'Root', 'Location', 'Best');

axis tight;

% Plot for function (c)

```

```
x_c = linspace(0.5, 2, 1000); % Adjust interval based on root location

y_c = func_c(x_c);

figure(2);

%plot(x_c, y_c, 'b', root_c, func_c(root_c), 'ro', 'MarkerSize', 10, 'LineWidth', 2);

grid on; grid minor;

%title(['Root of f(x) = 10*log(x) - x: ', num2str(root_c, '%.6f')]);

xlabel('x (unit)');

ylabel('f(x) (unit)');

legend('f(x)', 'Root', 'Location', 'Best');

axis tight;


% Newton Method Function

function root = newton_method(x0, f, df, imax, tol)

    % Newton's method implementation

    x_old = x0;

    count = 0;


    while count < imax

        % Evaluate function and derivative

        f_val = f(x_old);

        df_val = df(x_old);


        % Check derivative is not zero

        if abs(df_val) < 1e-12

            error('Derivative near zero, Newton method cannot proceed');
```

```
end

% Compute new x using Newton's formula

x_new = x_old - f_val / df_val;

% Check for convergence

if abs(x_new - x_old) < tol

    root = x_new;

    fprintf('Converged after %d iterations\n', count + 1);

    return;

end

% Update values for the next iteration

x_old = x_new;

count = count + 1;

% Display iteration progress

fprintf('Iteration %d: x = %.6f, f(x) = %.6f\n', count, x_new, f_val);

end

% If no convergence

error('Newton method did not converge within maximum iterations');

end
```

Newton's Method: Intersection of Two Functions

```

% f(x) = 100 e-x - 5 sin(pi x / 2)

% Solve for f(x) = 0 → intersection of 100 e-x and 5 sin(pi x/2)

clear;

clc; close all;

% (b) Plot both functions for range 3 ≤ x ≤ 10

x = linspace(3, 10, 1000);

y_left = 100 * exp(-x);           % 100 e-x
y_right = 5 * sin( (pi*x)/2 );    % 5 sin(pi x / 2)

figure(1); clf;

%plot(x, y_left, 'b', 'LineWidth', 2); hold on;

%plot(x, y_right, 'r', 'LineWidth', 2);

grid on; grid minor;

xlabel('x'); ylabel('y');

legend('100 e-x', '5 sin(\pi x/2)', 'Location', 'Best');

title('Intersection of 100 e-x and 5 sin(\pi x / 2)');

% (c) Newton's Method to find root near x0 = 4

f = @(x) 100*exp(-x) - 5*sin((pi*x)/2);

```

```
df = @(x) -100*exp(-x) - 5*(pi/2)*cos((pi*x)/2);

x0 = 4;          % given initial guess

tol = 1e-6;

imax = 20;

root = newton_method(x0, f, df, imax, tol);

fprintf('\nRoot found near x0 = 4 → x = %.10f\n', root);

% Plot the root on the graph

y_root = 100 * exp(-root); % either side of equation (should be equal)

%plot(root, y_root, 'ko', 'MarkerSize', 10, 'LineWidth', 2);

title(['Root at x = ', num2str(root, '%.6f')]);

%function root = newton_method(x0, f, df, imax, tol)

x = x0;

for i = 1:imax

    fx = f(x);

    dfx = df(x);
```

```

if dfx == 0
    error('Derivative became zero — Newton Method fails.');
```

end

```

x_new = x - fx/dfx;

fprintf('Iteration %02d: x = %.10f, f(x) = %.10f\n', ...
    i, x_new, f(x_new));

if abs(x_new - x) < tol
    root = x_new;
    return;
end

x = x_new;

end
```

Bracketing & Multiple Roots

```

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

% Goal: Find ALL roots in interval [4, 10]

clear;

clc; close all;
```

```
%% Define the function and its derivative
```

```
f = @(x) 100*exp(-x) - 5*sin((pi*x)/2);
```

```
df = @(x) -100*exp(-x) - 5*(pi/2)*cos((pi*x)/2);
```

```
% (a) Bracketing Routine: find sign changes in f(x)
```

```
x = linspace(4, 10, 1000);
```

```
fx = f(x);
```

```
brackets = []; % store intervals where root exists
```

```
for i = 1:length(x)-1
```

```
    if fx(i) * fx(i+1) < 0 % sign change = root in (x(i), x(i+1))
```

```
        brackets = [brackets; x(i), x(i+1)];
```

```
    end
```

```
end
```

```
disp('Bracketing intervals found:');
```

```
disp(brackets);
```

```
% (b) Use Bisection + Newton to find all roots in interval
```

```
tol = 1e-6;
```

```
imax = 50;

all_roots = zeros(size(brackets, 1), 1);

for k = 1:size(brackets, 1)

    a = brackets(k,1);
    b = brackets(k,2);

    % --- Bisection Method first ---
    root_bisect = bisection_method(f, a, b, imax, tol);

    % --- Newton Method for refinement ---
    root_newton = newton_method(root_bisect, f, df, imax, tol);

    all_roots(k) = root_newton;

    fprintf('Root %d found at x = %.8f\n', k, root_newton);

end

% (c) Plot equation (**) and mark all roots
%  $100 e^{-x}$  and  $5 \sin(\pi x / 2)$ 
```

```

y_left = 100 * exp(-x);

y_right = 5 * sin((pi*x)/2);


figure(1); clf;

plot(x, y_left, 'b', 'LineWidth', 2); hold on;

plot(x, y_right, 'r', 'LineWidth', 2);


% Mark each root
for k = 1:length(all_roots)

    xr = all_roots(k);

    yr = 100 * exp(-xr); % same value on intersection

    plot(xr, yr, 'ko', 'MarkerSize', 10, 'LineWidth', 2);

end


grid on; grid minor;

xlabel('x');

ylabel('y');

legend('100 e^{-x}', '5 sin(\pi x / 2)', 'Roots');

title('Multiple Roots of 100 e^{-x} = 5 sin(\pi x / 2)');


Function Below


function root = bisection_method(f, a, b, imax, tol)


fa = f(a);

```

```
fb = f(b);

if fa * fb > 0

    error('Bisection error: No sign change in interval.');
```

end

```
for i = 1:imax

    m = (a+b)/2;

    fm = f(m);

    if abs(fm) < tol || (b-a)/2 < tol

        root = m;

        return;

    end

    if fa * fm < 0

        b = m;

        fb = fm;

    else

        a = m;

        fa = fm;

    end

end

root = (a+b)/2;
```

end

```
%function root = newton_method(x0, f, df, imax, tol)
```

```
x = x0;
```

```
for i = 1:imax
```

```
    fx = f(x);
```

```
    dfx = df(x);
```

```
    if dfx == 0
```

```
        error('Derivative became zero — Newton fails.');
```

```
    end
```

```
    x_new = x - fx/dfx;
```

```
    if abs(x_new - x) < tol
```

```
        root = x_new;
```

```
        return;
```

```
    end
```

```
    x = x_new;
```

```
end
```