

Task 2

Table of Contents

1 Bisection $f(x) = \sin(x)\sinh(x)$; Limits = [1 4] [3].....	1
1 Bisection $f(x) = (R^2-x^2)^{1/2} - \tan(x)$ [3].....	3
2 Newtons $f(x) = e^x - (1+x+(x^2/2))$ [3].....	3
2 Newtons $f(x) = 10\log(x)-x$ [3].....	4
3 Newtons Intersect [6].....	4
4 Bracketing Bisection [6].....	4
4 Bracketing Newton [6].....	6
Functions.....	6

1 Bisection $f(x) = \sin(x)\sinh(x)$; Limits = [1 4] [3]

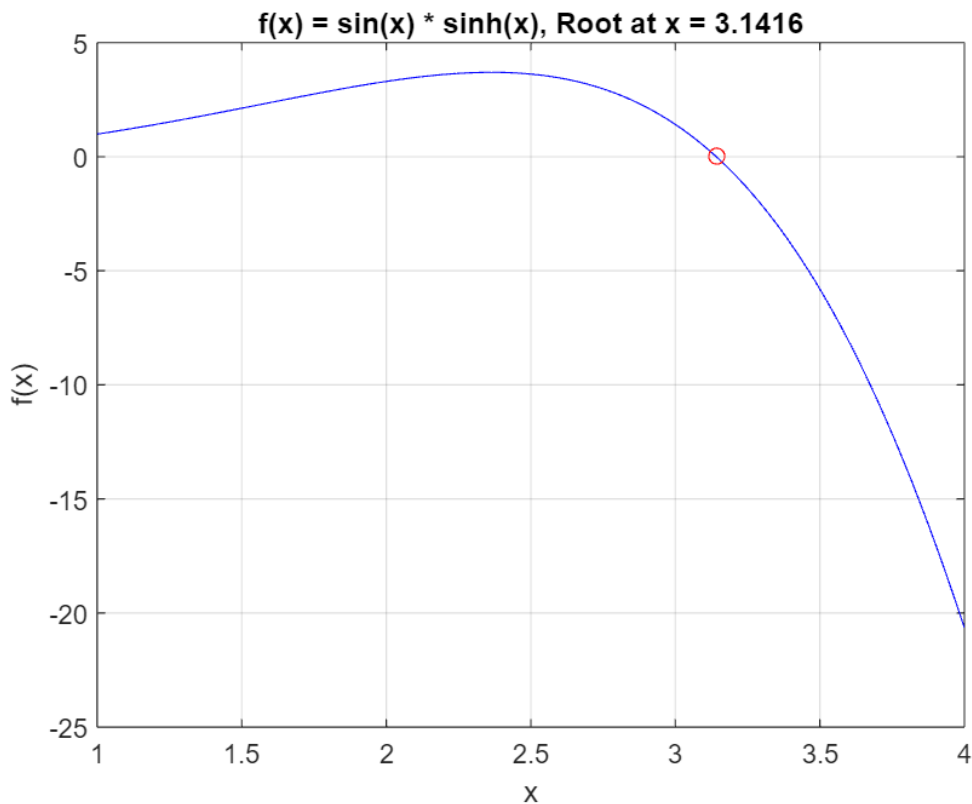
```
% Main Script
clc;
clear;

% Parameters for Bisection
imax = 50; % Maximum iterations
tol = 1e-6; % Tolerance

% Interval and Function for Case (b)
a_b = 1;
b_b = 4;
f_b = @func_b;

% Find Root for Function (b)
root_b = bisection(f_b, a_b, b_b, imax, tol);

% Plot Function (b)
x_b = linspace(a_b, b_b, 100);
y_b = func_b(x_b);
figure;
plot(x_b, y_b, 'b-', root_b, func_b(root_b), 'ro');
xlabel('x');
ylabel('f(x)');
title(['f(x) = sin(x) * sinh(x), Root at x = ', num2str(root_b)]);
grid on;
```



```
% Interval and Function for Case (c)
```

```
a_c = 4;  
b_c = 4.7;  
f_c = @func_c;
```

```
% Find Root for Function (c)
```

```
root_c = bisection(f_c, a_c, b_c, 1000, 1e-6);
```

```
Error using Task2_hand_in>bisection  
No root in the specified interval
```

```
% Plot Function (c)
```

```
x_c = linspace(a_c, b_c, 100);  
y_c = func_c(x_c);  
figure;  
plot(x_c, y_c, 'r-', root_c, func_c(root_c), 'go');  
xlabel('x');  
ylabel('f(x)');  
title(['f(x) = sin(x) - 10 / x, Root at x = ', num2str(root_c)]);  
grid on;
```

```
% Root Finding with Bisection Method and Function Evaluations
```

```
% Define Function for Bisection Method
```

1 Bisection $f(x) = (R^2 - x^2)^{1/2} - x \tan(x)$ [3]

2 Newtons $f(x) = e^x - (1 + x + (x^2/2))$ [3]

```
% Newton Method Implementation
```

```
% (d) Main script to call Newton's method
```

```
% Function (b) parameters
```

```
x0_b = 1.5; % Adjusted initial guess
```

```
tol_b = 1e-6; % Desired accuracy
```

```
imax_b = 100; % Increased maximum number of iterations
```

```
% Function (c) parameters
```

```
x0_c = 2; % Adjusted initial guess
```

```
tol_c = 1e-6; % Desired accuracy
```

```
imax_c = 100; % Increased maximum number of iterations
```

```
% Solve for root using Newton's method
```

```
root_b = newton(x0_b, @func_b, @dfunc_b, imax_b, tol_b);
```

```
fprintf('Root for function b: %.6f\n', root_b);
```

```
root_c = newton(x0_c, @func_c, @dfunc_c, imax_c, tol_c);
```

```
fprintf('Root for function c: %.6f\n', root_c);
```

```
% (e) Plot function (b)
```

```
x_b = linspace(0, 3, 100);
```

```
y_b = arrayfun(@func_b, x_b);
```

```
figure;
```

```
plot(x_b, y_b, 'r-', root_b, func_b(root_b), 'go');
```

```
xlabel('x');
```

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

```
title(['Function:  $f(x) = e^x - (1 + x + x^2 / 2)$ , Root at  $x =$ ', num2str(root_b)]);
```

```
grid on;
```

```
% Plot function (c)
```

```
x_c = linspace(0.5, 5, 100);
```

```
y_c = arrayfun(@func_c, x_c);
```

```
figure;
```

```
plot(x_c, y_c, 'b-', root_c, func_c(root_c), 'ro');
```

```
xlabel('x');
```

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

```
title(['Function:  $f(x) = 10 \cdot \log(x) - x$ , Root at  $x =$ ', num2str(root_c)]);
```

```
grid on;
```

2 Newtons $f(x) = 10\log(x)-x$ [3]

3 Newtons Intersect [6]

```
clc;
clear;

% Plot the two functions in equation (**)
x = linspace(3, 10, 1000);
y_left = 100 * exp(-x); % Left-hand side of the equation
y_right = 5 * sin(pi * x.^2); % Right-hand side of the equation

figure;
plot(x, y_left, 'b-', 'LineWidth', 1.5); hold on;
plot(x, y_right, 'r-', 'LineWidth', 1.5);
xlabel('x');
ylabel('y');
legend('100 e^{-x}', '5 sin(\pi x^2)');
title('Intersection of 100 e^{-x} and 5 sin(\pi x^2)');
grid on;

% (c) Use Newton's method to find the root near x0 = 4
x0 = 4; % Initial guess
tol = 1e-10; % Desired accuracy
imax = 20; % Maximum iterations

% Call Newton's method
root = newton(x0, @func, @dfunc, imax, tol);

% Plot the root on the graph
y_root = 100 * exp(-root); % Evaluate y-value of root
plot(root, y_root, 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'g');
title(['Root at x = ', num2str(root, '%.10f')]);

% Display the root and number of iterations
disp(['Root found: x = ', num2str(root, '%.10f')]);
```

4 Bracketing Bisect [6]

```
clc;
clear;

% (e) Main script to run everything
```

```

% Interval [4, 10]
a = 4;
b = 10;

% Bracket the roots
intervals = bracketing(@func, a, b, 100);

% Initialize root finding results
roots_bisection = [];
roots_newton = [];

% For each interval found by bracketing, use Bisection and Newton to find the root
for i = 1:size(intervals, 1)
    % Extract the interval
    a_interval = intervals(i, 1);
    b_interval = intervals(i, 2);

    % Use Bisection method
    root_b = bisect(@func, a_interval, b_interval, 100, 1e-6);
    roots_bisection = [roots_bisection, root_b];

    % Use Newton's method (starting with the midpoint of the interval)
    x0 = (a_interval + b_interval) / 2;
    root_n = newton(x0, @func, @dfunc, 100, 1e-6);
    roots_newton = [roots_newton, root_n];
end

% Plot the functions and roots
x_vals = linspace(4, 10, 1000);
y_vals = func(x_vals);
figure;
plot(x_vals, y_vals, 'b-', 'LineWidth', 1.5); hold on;
plot(roots_bisection, func(roots_bisection), 'go', 'MarkerSize', 8,
'MarkerFaceColor', 'g');
plot(roots_newton, func(roots_newton), 'ro', 'MarkerSize', 8, 'MarkerFaceColor',
'r');
xlabel('x');
ylabel('f(x)');
legend('100 e^{-x} - 5 sin(\pi x^2)', 'Roots (Bisection)', 'Roots (Newton)',
'Location', 'best');
title('Roots of the Function 100 e^{-x} - 5 sin(\pi x^2)');
grid on;

% Display the roots found
disp('Roots found using Bisection:');
disp(roots_bisection);
disp('Roots found using Newton's method:');
disp(roots_newton);

```

4 Bracketing Newton [6]

Functions

```
function root = newton(x, f, df, imax, tol)
    % Input: x - initial guess at the root
    %         f - function to be evaluated
    %         df - derivative of f with respect to x
    %         imax - maximum number of iterations
    %         tol - convergence criterion
    % Output: root - the root of the function

    count = 0; % Iteration counter
    fold = feval(f, x); % Initial function value
    dfold = feval(df, x); % Initial derivative value

    % Newton's method iteration loop
    while abs(fold) > tol && count < imax
        if dfold == 0
            error('Derivative is zero, unable to proceed!');
        end

        % Update root estimate
        dx = -fold / dfold;
        x = x + dx;

        % Update function and derivative values
        fold = feval(f, x);
        dfold = feval(df, x);

        count = count + 1;

        % Display iteration progress
        fprintf('Iteration: %i, Root estimate: %.6f, Error: %.6f\n', count, x,
abs(fold));
    end

    if count >= imax
        error('Maximum iterations reached without convergence. Try a better initial
guess or increase imax.');
```

```
    end

    % Final root
    root = x;
end
```

```

% (b) Function  $f(x) = e^x - (1 + x + x^2 / 2)$  and its derivative
function y = func_b(x)
    y = exp(x) - (1 + x + x^2 / 2);
end

function y_prime = dfunc_b(x)
    y_prime = exp(x) - (1 + x); % Derivative of f(x)
end

% (c) Function  $f(x) = 10 \cdot \log(x) - x$  and its derivative
function y = func_c(x)
    y = 10 * log(x) - x;
end

function y_prime = dfunc_c(x)
    y_prime = 10 / x - 1; % Derivative of f(x)
end

function root = bisect(f, a, b, imax, tol)
    % Bisection Method to find the root of a function
    % Inputs:
    %   f - Function handle
    %   a, b - Interval limits
    %   imax - Maximum number of iterations
    %   tol - Convergence tolerance
    % Output:
    %   root - Position of the root

    x1 = a;
    x3 = b;
    f1 = feval(f, x1);
    f3 = feval(f, x3);

    % Check if root exists in the interval
    if f1 * f3 > 0
        error('No root in the specified interval');
    end

    for ndx = 1:imax
        x2 = (x1 + x3) / 2; % Midpoint
        f2 = feval(f, x2);

        % Check which subinterval contains the root
        if f1 * f2 < 0
            x3 = x2;
            f3 = f2;
        else
            x1 = x2;
            f1 = f2;
        end
    end
end

```

```

        end

        % Check for convergence
        d = abs(x3 - x1);
        if d < tol
            root = x2;
            return;
        end
    end

    error('Bisection method did not converge within the maximum iterations');
end

% Define Function (b):  $f(x) = \sin(x) * \sinh(x)$ 
function y = func_b(x)

```

Function 'func_b' has already been declared within this scope.

```

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

% Define Function (c):  $f(x) = \sin(x) - R / x$ 
function y = func_c(x)
    R = 10; % Given constant
    y = sin(x) - R ./ x;
end

function root = newton(x, f, df, imax, tol)
    % Input: x - initial guess at the root
    %         f - function to be evaluated
    %         df - derivative of f with respect to x
    %         imax - maximum number of iterations
    %         tol - convergence criterion
    % Output: root - the root of the function

    count = 0; % Iteration counter
    fold = feval(f, x); % Initial function value
    dfold = feval(df, x); % Initial derivative value

    % Newton's method iteration loop
    while abs(fold) > tol && count < imax
        if dfold == 0
            error('Derivative is zero, unable to proceed!');
        end

        % Update root estimate
        dx = -fold / dfold;
        x = x + dx;

        % Update function and derivative values
        fold = feval(f, x);

```

```

        dfold = feval(df, x);

        count = count + 1;

        % Display iteration progress
        fprintf('Iteration: %i, Root estimate: %.6f, Error: %.6f\n', count, x,
abs(fold));
    end

    if count >= imax
        error('Maximum iterations reached without convergence. Try a better initial
guess or increase imax.');
```

```

    end

    % Final root
    root = x;
end

% (b) Function  $f(x) = e^x - (1 + x + x^2 / 2)$  and its derivative
function y = func_b(x)
    y = exp(x) - (1 + x + x^2 / 2);
end

function y_prime = dfunc_b(x)
    y_prime = exp(x) - (1 + x); % Derivative of f(x)
end

% (c) Function  $f(x) = 10 \cdot \log(x) - x$  and its derivative
function y = func_c(x)
    y = 10 * log(x) - x;
end

function y_prime = dfunc_c(x)
    y_prime = 10 / x - 1; % Derivative of f(x)
end

% (a) Define the function and its derivative
function y = func(x)
    y = 100 * exp(-x) - 5 * sin(pi * x.^2); % Use .^ for elementwise power
end

function y_prime = dfunc(x)
    y_prime = -100 * exp(-x) - 10 * pi * x .* cos(pi * x.^2); % Use .* for
elementwise multiplication
end

% (b) Bracketing routine to find intervals with roots
function intervals = bracketing(func, a, b, n)
    x = linspace(a, b, n); % Create n points between a and b
    y = func(x); % Evaluate the function at the points

```

```

intervals = []; % Initialize the intervals list

% Check for sign change between consecutive points
for i = 1:n-1
    if sign(y(i)) ~= sign(y(i+1)) % Sign change detected
        intervals = [intervals; x(i), x(i+1)];
    end
end
end

% (d) Bisection method for finding roots
function root = bisect(func, a, b, imax, tol)
    count = 0;
    fa = func(a);
    fb = func(b);

    if sign(fa) == sign(fb)
        error('No root in the specified interval');
    end

    while (b - a) / 2 > tol && count < imax
        c = (a + b) / 2;
        fc = func(c);

        if fc == 0
            break;
        elseif sign(fc) == sign(fa)
            a = c;
            fa = fc;
        else
            b = c;
            fb = fc;
        end

        count = count + 1;
    end

    root = (a + b) / 2;
end

function multiple_roots
    % Function definition
    func = @(x) 100 * exp(-x) - 5 * sin(pi * x.^2);
    dfunc = @(x) -100 * exp(-x) - 10 * pi * x .* cos(pi * x.^2); % Derivative of
    f(x)

    % Interval for root finding

```

```

a = 4;
b = 10;

% Bracketing: Approximate intervals containing roots
intervals = bracketing(func, a, b, 100);

% Initialize parameters
tol = 1e-6;
imax = 50;

% Arrays to store root positions
bisection_roots = [];
newton_roots = [];

% Refine roots using both methods
for i = 1:size(intervals, 1)
    % Bisection method
    root_b = bisect(func, intervals(i, 1), intervals(i, 2), imax, tol);
    bisection_roots = [bisection_roots; root_b];

    % Newton method
    x0 = (intervals(i, 1) + intervals(i, 2)) / 2; % Midpoint as initial guess
    root_n = newton(func, dfunc, x0, imax, tol);
    newton_roots = [newton_roots; root_n];
end

% Plot the function
figure;
x = linspace(a, b, 1000);
y = func(x);
plot(x, y, 'b-', 'LineWidth', 1.5); hold on;

% Mark Bisection method roots
y_roots_bisection = func(bisection_roots); % Should all be ~0
plot(bisection_roots, y_roots_bisection, 'ro', 'MarkerSize', 8, 'LineWidth', 2,
'DisplayName', 'Bisection Roots');

% Mark Newton-Raphson method roots
y_roots_newton = func(newton_roots); % Should all be ~0
plot(newton_roots, y_roots_newton, 'go', 'MarkerSize', 8, 'LineWidth', 2,
'DisplayName', 'Newton-Raphson Roots');

% Add title and labels
title('Roots of  $f(x) = 100e^{-x} - 5\sin(\pi x^2)$ ');
xlabel('x');
ylabel('f(x)');
grid on;
legend('f(x)', 'Bisection Roots', 'Newton-Raphson Roots', 'Location', 'Best');

% Display roots

```

```

fprintf('Roots found using Bisection:\n');
disp(bisection_roots);

fprintf('Roots found using Newton-Raphson:\n');
disp(newton_roots);
end

% Bracketing function
function intervals = bracketing(func, a, b, num_points)
    x = linspace(a, b, num_points);
    y = func(x);
    intervals = [];
    for i = 1:(num_points - 1)
        if y(i) * y(i + 1) < 0 % Sign change indicates a root
            intervals = [intervals; x(i), x(i + 1)];
        end
    end
end

% Bisection method
function root = bisect(func, a, b, imax, tol)
    for i = 1:imax
        c = (a + b) / 2;
        if abs(func(c)) < tol || (b - a) / 2 < tol
            root = c;
            return;
        end
        if func(a) * func(c) < 0
            b = c;
        else
            a = c;
        end
    end
    error('Bisection method did not converge within maximum iterations.');
```

```

end

% (c) Newton's method for finding roots
function root = newton(x0, func, df, imax, tol)
    count = 0;
    fold = func(x0);
    dfold = df(x0);

    while abs(fold) > tol && count < imax
        if dfold == 0
            error('Derivative is zero. Method fails.');
```

```

        end

        % Newton-Raphson update
        dx = -fold / dfold;
        x0 = x0 + dx;
    end
end

```

```

        fold = func(x0);
        dfold = df(x0);
        count = count + 1;
    end

    if count >= imax
        error('Maximum iterations reached without convergence.');
```

end

```

    root = x0;
end

% Newton-Raphson method
function root = newton(func, dfunc, x0, imax, tol)
    for i = 1:imax
        f_val = func(x0);
        df_val = dfunc(x0);
        if abs(f_val) < tol
            root = x0;
            return;
        end
        if df_val == 0
            error('Derivative is zero. Newton method failed.');
```

end

```

        x0 = x0 - f_val / df_val;
    end
    error('Newton-Raphson method did not converge within maximum iterations.');
```

end