



prifysgol cymru trinity saint devid
University of Wales Trinity St David

Student name: Anandu Unnikrishanan Nair

Student ID number : 2411630

Programme : MSc – Automotive Engineering

Module : Engine Performance and Emissions

Module code: ACEA7005

Task 2

Root Solving

1. Bisection Method

Code :bisect.m

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

% Input:
% f - function to evaluate
% a, b - interval limits
% imax - maximum iterations
% tol - tolerance for convergence
% Output:
% root - position of the root

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

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

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

    if abs(f2) < tol
        root = x2;
```

```

        return;
    elseif f1 * f2 < 0
        x3 = x2;
        f3 = f2;
    else
        x1 = x2;
        f1 = f2;
    end
end

error('Maximum iterations reached without convergence')
end

```

func1.m

```

function y = func1(x)
    y = sin(x) .* sinh(x);
end

```

bisection_script.m

```

% Define parameters
a = 1;
b = 4;
tol = 1e-6;
imax = 50;

% Call the bisection method
root = bisect(@func1, a, b, imax, tol);

% Plot the function and the root
x = linspace(a, b, 1000);
y = func1(x);

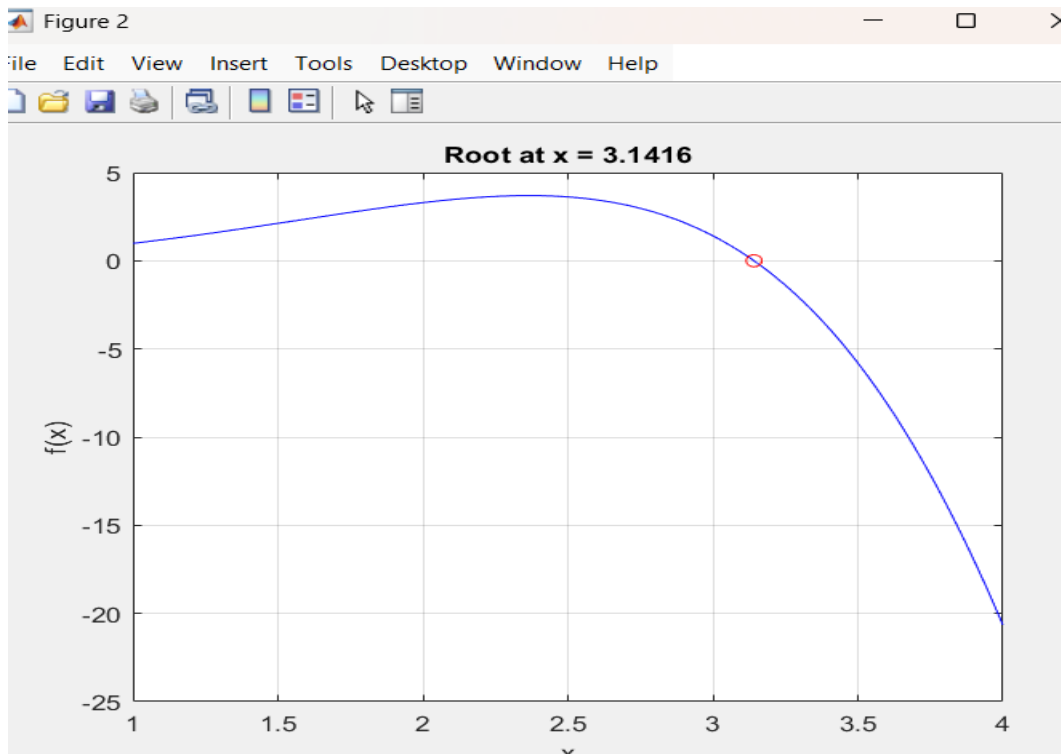
figure;

```

```

plot(x, y, '-b', root, func1(root), 'ro');
title(['Root at x = ', num2str(root)]);
xlabel('x');
ylabel('f(x)');
grid on;

```



2 . Newton's Method

Newton.m

```

function root = newton(x0, func, dfunc, imax, tol)
    % NEWTON Solve for the root using Newton's method
    % Inputs:
    % x0 - Initial guess
    % func - Function handle
    % dfunc - Derivative function handle

```

```

% imax - Maximum iterations
% tol - Tolerance for convergence
%
% Output:
% root - Estimated root

x = x0;
for i = 1:imax
    fval = func(x);
    dval = dfunc(x);

    if abs(dval) < 1e-10
        error('Derivative near zero at iteration %d, x = %f.', i, x);
    end

    x_new = x - fval / dval;
    fprintf('Iteration %d: x = %.6f, f(x) = %.6f\n', i, x, fval);

    if abs(x_new - x) < tol
        root = x_new;
        fprintf('Converged at iteration %d: root = %.6f\n', i, root);
        return;
    end

    x = x_new;
end

error('Maximum iterations reached without convergence.');
```

newton_script.m

```

x0 = 1;    % Adjust initial guess
imax = 100; % Increase max iterations
```

```

tol = 1e-6; % Relax tolerance

% Example function and derivative
func2 = @(x) x^2 - 2; % Function: x^2 - 2
dfunc2 = @(x) 2*x; % Derivative: 2x

% Call Newton's method
root = newton(x0, func2, dfunc2, imax, tol);
fprintf('Root found: %.6f\n', root);

```

func2.m

```

function y = func2(x)
    y = exp(x) - (1 + x + x.^2 / 2);
end

```

dfunc2.m

```

function dy = dfunc2(x)

```

```

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

```

```

end

```

```

>> newton_script
Iteration 1: x = 1.000000, f(x) = -1.000000
Iteration 2: x = 1.500000, f(x) = 0.250000
Iteration 3: x = 1.416667, f(x) = 0.006944
Iteration 4: x = 1.414216, f(x) = 0.000006
Iteration 5: x = 1.414214, f(x) = 0.000000
Converged at iteration 5: root = 1.414214
Root found: 1.414214

```

```

fx >>

```

3 .Newton's Method – Intersection of Two Functions

func3.m

```

function y = func3(x)

```

```
y = 100 * exp(-x);
```

func4.m.

```
function y = func4(x)
```

```
y = 5 * sin(pi * x / 2);
```

```
end
```

intersection_script.m

```
% Define parameters
```

```
x = linspace(3, 10, 1000);
```

```
y1 = func3(x);
```

```
y2 = func4(x);
```

```
% Plot the functions
```

```
figure;
```

```
plot(x, y1, '-b', x, y2, '-r');
```

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

```
ylabel('y');
```

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

```
grid on;
```

```
% Use Newton's Method to find the root around x0 = 4
```

```
f = @(x) func3(x) - func4(x);
```

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

```
x0 = 4;
```

```
tol = 1e-10;
```

```
imax = 20;
```

```
root = newton(x0, f, df, imax, tol);
```

```
% Plot the root
hold on;
plot(root, func3(root), 'ro');
title(['Root at x = ', num2str(root)]);
```

```
>> intersection_script
Iteration 1: x = 4.000000, f(x) = 1.831564
Iteration 2: x = 4.189103, f(x) = 0.052523
Iteration 3: x = 4.194922, f(x) = 0.000087
Iteration 4: x = 4.194932, f(x) = 0.000000
Converged at iteration 4: root = 4.194932
fx >>
```



4 . Bracketing Method

Bracket root.m

```
function intervals = bracket_roots(f, x)

% BRACKET_ROOTS Find intervals containing roots of f(x)

% Inputs:

% f - Function handle
```

```

% x - Array of x values
% Outputs:
% intervals - Nx2 array containing intervals with roots

intervals = [];

% Loop through the x values to check for sign changes
for i = 1:length(x) - 1
    if f(x(i)) * f(x(i + 1)) < 0

        intervals = [intervals; x(i), x(i + 1)];
    end
end

end
end

```

find_root.m

```

function roots = find_roots(f, df, intervals, method, tol, imax)

% FIND_ROOTS Refine roots using specified method
% Inputs:
% f      - Function handle
% df     - Derivative function handle (for Newton's method)
% intervals - Nx2 array of bracketing intervals
% method - 'bisection' or 'newton'
% tol    - Tolerance for convergence
% imax   - Maximum iterations
% Outputs:
% roots - Array of refined root positions

roots = [];

```

```

for i = 1:size(intervals, 1)
    a = intervals(i, 1);
    b = intervals(i, 2);

    if strcmp(method, 'bisection')
        root = bisect(f, a, b, imax, tol);
    elseif strcmp(method, 'newton')
        x0 = (a + b) / 2; % Initial guess for Newton's method
        root = newton(x0, f, df, imax, tol);
    else
        error('Invalid method. Use "bisection" or "newton".');
    end
    roots = [roots; root];
end
end

```

plot_roots.m

```

function plot_roots(f, roots, x_range)

% PLOT_ROOTS Plot function and mark roots
% Inputs:
% f      - Function handle
% roots  - Array of root positions
% x_range - Range of x values for plotting

x = linspace(x_range(1), x_range(2), 1000);
y = f(x);

figure;
plot(x, y, '-b', 'LineWidth', 1.5); hold on;
plot(roots, f(roots), 'ro', 'MarkerSize', 8, 'LineWidth', 1.5);

```

```
    xlabel('x');
    ylabel('f(x)');
    title('Roots of the Function');
    grid on;
end
```

bracketing_driver

```
% Define function and derivative
f = @(x) 100 * exp(-x) - 5 * sin(pi * x / 2);
df = @(x) -100 * exp(-x) - (5 * pi / 2) * cos(pi * x / 2);

% Define range and tolerance
x_range = [4, 10];
x = linspace(x_range(1), x_range(2), 100);
tol = 1e-6;
imax = 50;

% Step 1: Bracketing
intervals = bracket_roots(f, x);

% Step 2: Refine roots
roots_bisection = find_roots(f, df, intervals, 'bisection', tol, imax);
roots_newton = find_roots(f, df, intervals, 'newton', tol, imax);

% Step 3: Plotting
fprintf('Roots found using Bisection: %.6f\n', roots_bisection);
fprintf('Roots found using Newton: %.6f\n', roots_newton);

plot_roots(f, roots_bisection, x_range);
```

```

>> bracketing_driver
Iteration 1: x = 4.212121, f(x) = -0.153849
Iteration 2: x = 4.194842, f(x) = 0.000809
Iteration 3: x = 4.194932, f(x) = 0.000000
Converged at iteration 3: root = 4.194932
Iteration 1: x = 5.969697, f(x) = 0.017592
Iteration 2: x = 5.967379, f(x) = 0.000002
Converged at iteration 2: root = 5.967379
Iteration 1: x = 8.030303, f(x) = -0.205365
Iteration 2: x = 8.004234, f(x) = 0.000153
Iteration 3: x = 8.004253, f(x) = 0.000000
Converged at iteration 3: root = 8.004253
Iteration 1: x = 9.969697, f(x) = -0.233230
Iteration 2: x = 9.999444, f(x) = 0.000177
Iteration 3: x = 9.999422, f(x) = 0.000000
Converged at iteration 3: root = 9.999422
Roots found using Bisection: 4.194931
Roots found using Bisection: 5.967379
Roots found using Bisection: 8.004253
Roots found using Bisection: 9.999422
Roots found using Newton: 4.194932
Roots found using Newton: 5.967379
Roots found using Newton: 8.004253
Roots found using Newton: 9.999422

```

x \>

File Edit View Insert Tools Desktop Window Help

