



University of Wales Trinity St David

Task - 1

Student name: Aangan Nitinkumar Patel

Student ID number: 2511974

Program: MSc – Automotive Engineering

Engineering Module: Alternative fuels and Combustion Modelling

Module code: ACHA7002

Date: 07/12/2025

Contents

1	Plotting and differentiation.....	1
2	Trapezoidal Method.....	2
3	Composite	2
4	Simpsons.....	2
5	Integration results tables.....	3
5.1	Composite convergence iterations	3
5.2	Simpsons Convergence iterations.....	4
6	Appendices.....	4
6.1	Your code script and functions.....	4

1 Plotting and differentiation

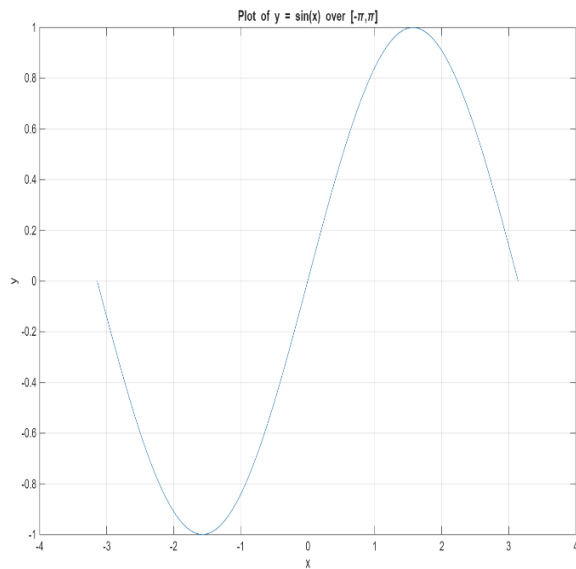


Figure 1 $y = \sin(x)$ plot

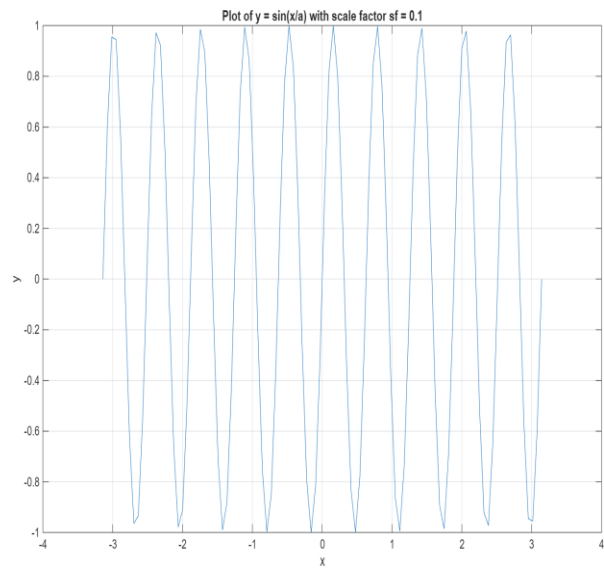


Figure 2 $y = \sin_scale(x, sf)$ plot (0.1 steps)

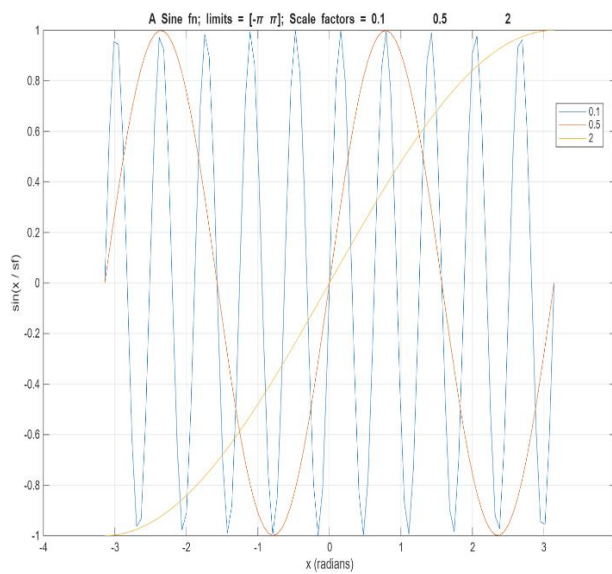


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

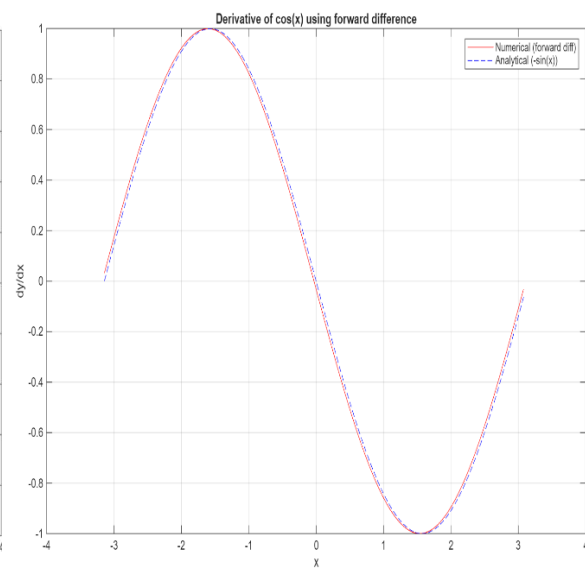


Figure 4 diff cos (x) to -sin(x)

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h_i}$$

Equation 1-1 numerical differentiation

2 Trapezoidal Method

$$I \approx \frac{1}{2}(f_a + f_b)(b - a)$$

Equation 2-1 trapezoidal method

A function was created to integrate 'x sin(x)' between 0 and pi/2 the code is in appendix

$$I = \int_0^{\pi/2} x \sin(x) dx ;$$

IntTrap (your code result)	Analytical (result)	percentage_error_Trap
1.233701	1	23.3701%

3 Composite

A function was created to integrate further functions in a discretised way from equation 3.1.

$$I \approx \frac{1}{2} \Delta x_n (f_a + 2 \sum_{i=1}^{n-1} f_i + f_b)$$

Equation 3-1 composite

Table 3-1 results for integration of $f(x) = 1/x$ using composite method a while loop was used to find when the result was within $1e-6$

	Analytical	Composite Trapezium			Converge $1e-6$
'Result'	0.69314718	0.69563492	0.69317218	0.69315343	0.69314768
'n iterations'	Na	5	50	100	354
'Percentage difference'	Na	0.35890503%	0.00360656%	0.00090167%	0.00007195%

4 Simpsons

A function was created to integrate further functions in a discretised separating odd and even crossing over in a quadratic manner way from equation 4.1

$$I \approx \frac{1}{3} \Delta x_n (f_a + 4 \sum_{i=1, \text{ odd only}}^{n-1} f_i + 2 \sum_{i=1, \text{ even only}}^{n-1} f_i + f_b)$$

Equation 4-1 Simpsons

Table 4-1 results for integration of $f(x) = 1/x$ using Simpson's method a while loop was used to find when the result was within $1e-6$ of the analytical

	Analytical	Simpsons Trapezium			Converge 1e-6
'Result'	0.69314718	0.69316979	0.69314719	0.69314718	0.69314765
'n iterations'	NA	5	50	100	16
'Percentage difference'	NA	0.00326231%	0.00000072%	0.00000005%	0.00006813%

5 Integration results tables

5.1 Composite convergence iterations

N (Intervals)	Numerical Integral	Absolute Error	Percentage Error
5	0.6956349206	0.0024877401	0.3589050269%
50	0.6931721793	0.0000249988	0.0036065573%
100	0.6931534305	0.0000062499	0.0009016731%
Minimum N for 6 d.p.	354	4.987387	0.000072

5.2 Simpsons Convergence iterations

N (Intervals)	Numerical Integral	Absolute Error	Percentage Error
6 (Adjusted from 5)	0.6931697932	0.000022612610	0.003262310009%
50	0.6931471856	0.000000004995	0.000000720627%
100	0.6931471809	0.000000000312	0.000000045073%
Minimum EVEN N for 6 d.p.	16	4.722595	0.000068%

6 Appendices

6.1 Your code script and functions

6.1.1 Function

6.1.1.1 *sin_scale*

```
function y = sin_scale(x, sf)
% sin_scale: scales the input argument of the sine function
% Input:  scalar or vector x, sf = scale factor
% Output: scalar or vector y containing corresponding sine values

% ML intrinsic sine function with element-wise division
y = sin(x ./ sf); % note: ./ so it works for vectors

% The '.' before '/' means "do this for each element of x"
end
```

6.1.1.2 *forward_diff*

```

function [dy, x_d] = forward_diff(x, y)
% forward_diff: Forward difference numerical differentiation
% Inputs:
%   x - vector of x values (evenly spaced)
%   y - vector of y values, y = f(x)
% Outputs:
%   dy - numerical derivative at interior points
%   x_d - x positions corresponding to dy

N = length(x);           % number of points
h = x(2) - x(1);         % step size (assumed constant)

% Forward difference formula:
%  $f'(x_i) \approx (f(x_{i+1}) - f(x_i)) / h$ 
dy = diff(y) / h;        % diff(y) gives y(2:N) - y(1:N-1)

% x locations for derivative (we lose the last point)
x_d = x(1:N-1);
end

```

6.1.1.3 *trap1*

```

function I = trap1(f, a, b)
% trap1: Simple trapezoidal rule (single interval)
% Inputs:
%   f - function handle, e.g. @(x) x.*sin(x)
%   a - lower limit
%   b - upper limit
% Output:
%   I - approximate integral

if a > b
    error('b must be greater than a');
end

h = b - a;                % interval width
I = h * (f(a) + f(b)) / 2; % equation [3] in the notes
end

```

6.1.1.4 *comp_trap*

```

function I = comp_trap(f, a, b, n)
% comp_trap: Composite trapezium rule
% Inputs:
%   f - function handle
%   a - lower limit
%   b - upper limit
%   n - number of sub-intervals
% Output:
%   I - approximate integral

h = (b - a) / n;          % width of each sub-interval
x = a:h:b;                % n+1 points from a to b
y = f(x);                 % function values at each point

% Equation [4]:
%  $I \approx h/2 * (f(a) + 2*\sum(f \text{ at interior points}) + f(b))$ 
I = h * (0.5 * y(1) + sum(y(2:end-1)) + 0.5 * y(end));

```

end

6.1.1.5 comp_simpson

```
function I = comp_simpson(f, a, b, n)
% comp_simpson: Composite Simpson's rule
% Inputs:
%   f - function handle
%   a - lower limit
%   b - upper limit
%   n - number of sub-intervals (MUST be even)
% Output:
%   I - approximate integral

if rem(n, 2) ~= 0
    error('n must be even for Simpson''s rule');
end

h = (b - a) / n;
x = a:h:b;          % n+1 points
y = f(x);

% Simpson's rule:
% I ≈ h/3 * [f0 + fn + 4*(odd indices) + 2*(even indices)]
odd_sum = sum(y(2:2:end-1)); % y2, y4, ..., y_{n-1}
even_sum = sum(y(3:2:end-2)); % y3, y5, ..., y_{n-2}

I = h/3 * (y(1) + y(end) + 4*odd_sum + 2*even_sum);
end
```

6.1.2 script

6.1.2.1 lab1_sin

% lab1_sin.m

```
clc; clear; close all
format short g          % format numerical output

%% Task I (d,e): Plot y = sin(x) over [-pi, pi]

N = 100;                % number of points
x = linspace(-pi, pi, N)'; % column vector from -pi to pi

y = sin(x);             % built-in sine function

figure(1)
plot(x, y)
xlabel('x (radians)')
ylabel('sin(x)')
title('y = sin(x) over [-\pi, \pi]')
grid on

%% Task II (e): Plot scaled sine function with sf = 0.1

sf = 0.1;               % scale factor
y_scaled = sin_scale(x, sf); % call user-defined function
```

```

figure(2)
plot(x, y_scaled)
xlabel('x (radians)')
ylabel('y')
title(['Plot of y = sin(x/sf) with scale factor sf = ', num2str(sf)])
grid on

%% Task II (f): Plot multiple scaled sine waves using a for loop

sf = [0.1 0.5 2];           % scale factor values (array)
N_sf = numel(sf);           % number of scale factors

% Initialise an array of zeros to store sine values
% Size is N (rows) by N_sf (columns)
sin_array = zeros(N, N_sf);

for ndx = 1:N_sf
    % Compute scaled sine for each scale factor
    sin_array(:, ndx) = sin_scale(x, sf(ndx));
end

figure(3)
plot(x, sin_array)
xlabel('x (radians)')
ylabel('sin(x / sf)')
title(['A Sine fn; limits = [-\pi \pi]; Scale factors = ', num2str(sf)])
grid on

%% Task IV: Numerical Differentiation using Forward Difference

% Number of points and x range
N = 100;
x = linspace(-pi, pi, N)';

% y = cos(x)
y = cos(x);

% Use our forward difference function
[dy_num, x_d] = forward_diff(x, y);

% Analytical derivative of cos(x) is -sin(x)
dy_exact = -sin(x_d);

% Plot numerical vs analytical derivative
figure(4)
plot(x_d, dy_num, 'r', x_d, dy_exact, 'b--')
legend('Numerical (forward diff)', 'Analytical (-sin(x))')
title('Derivative of cos(x) using forward difference')
xlabel('x')
ylabel('dy/dx')
grid on

%% Numerical Integration - 1.Trapezoidal Method

% Integral I =  $\int_{-\pi/2}^{\pi/2} x \sin(x) dx$ 

```

```

a = 0;
b = pi/2;

% Define integrand as an anonymous function
f = @(x) x .* sin(x);

% Use trapezoidal rule
I_numerical = trap1(f, a, b);

% Analytic value from the notes
I_analytic = 1;

% Percentage error
percentage_error = abs(I_numerical - I_analytic) / I_analytic * 100;

fprintf('--- 1.Trapezoidal Method ---\n');
fprintf('Numerical: %f\n', I_numerical);
fprintf('Analytic : %f\n', I_analytic);
fprintf('Error      : %.4f%%\n\n', percentage_error);

%% Numerical Integration - 2.Composite Trapezium Rule

fprintf('--- 2.Composite Trapezium Rule ---\n');

% (2b)  $I = \int_0^{\pi/2} x \sin(x) dx$  with  $N = 100$ 
f = @(x) x .* sin(x);
a = 0;
b = pi/2;
N = 100;

I_num = comp_trap(f, a, b, N);
I_analytic = 1;
err_pct = abs(I_num - I_analytic) / I_analytic * 100;

fprintf('2b) Integral  $x \sin(x)$ ,  $[0, \pi/2]$ ,  $N = %d$ \n', N);
fprintf('Numerical: %f\n', I_num);
fprintf('Analytic : %f\n', I_analytic);
fprintf('Error      : %.6f%%\n\n', err_pct);

% (2c & 2d)  $I = \int_1^2 1/x dx$  for several N
f = @(x) 1 ./ x;
a = 1;
b = 2;
I_analytic = log(2);
N_values = [5, 50, 100];

fprintf('2c & 2d) Integral  $1/x$ ,  $[1, 2]$ \n');
fprintf(' N      Numerical      Abs Error      Percent Error\n');

for N = N_values
    I_num = comp_trap(f, a, b, N);
    err_abs = abs(I_num - I_analytic);
    err_pct = err_abs / I_analytic * 100;
    fprintf('%3d %12.8f %12.8f %12.8f\n', N, I_num, err_abs, err_pct);
end
fprintf('\n');

% (2e) Find N for ~6 decimal places accuracy

```

```

Target_Error = 0.5e-6;
N = 1;

while true
    I_num = comp_trap(f, a, b, N);
    if abs(I_num - I_analytic) < Target_Error
        break;
    end
    N = N + 1;
end

fprintf('2e) Min N for ~6 d.p. accuracy (trapezium): %d\n', N);
fprintf('Absolute Error: %e\n\n', abs(I_num - I_analytic));

%% Numerical Integration - 3.Composite Simpson's Rule

fprintf('--- 3.Composite Simpson''s Rule ---\n');

% Integral of 1/x from 1 to 2
f = @(x) 1 ./ x;
a = 1;
b = 2;
I_analytic = log(2);

% (3b & 3c) Requested N = 5, 50, 100
N_requested = [5, 50, 100];

fprintf('Req N    Used N    Numerical    Abs Error    Percent Error\n');

for k = 1:length(N_requested)
    N_req = N_requested(k);

    % N must be even for Simpson's rule
    if mod(N_req, 2) == 1
        N = N_req + 1;    % adjust odd N to next even
    else
        N = N_req;
    end

    I_num = comp_simpson(f, a, b, N);
    err_abs = abs(I_num - I_analytic);
    err_pct = err_abs / I_analytic * 100;

    fprintf('%5d %6d %12.8f %12.8f %12.8f\n', ...
        N_req, N, I_num, err_abs, err_pct);
end
fprintf('\n');

% (3d) Find minimum even N for ~6 decimal places accuracy
Target_Error = 0.5e-6;
N = 2;    % smallest even N

while true
    I_num = comp_simpson(f, a, b, N);
    if abs(I_num - I_analytic) < Target_Error
        break;
    end
    N = N + 2;    % keep N even
end

```

end

```
fprintf('3d) Min EVEN N for ~6 d.p. accuracy (Simpson): %d\n', N);  
fprintf('Absolute Error: %e\n', abs(I_num - I_analytic));  
fprintf('\n--- Lab 1 Numerical Tasks Complete ---\n');  
fprintf('\n--- Thank You ---\n')
```