

UWTSD

MSc

Task 1

Name :- ARTH PANDYA

**Module Name:- Alternative fuels and Combustion Modelling
(ACHA7002)**

Student number :- 2412514

Date :- 7/12/25

Contents

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

1 Plotting and differentiation

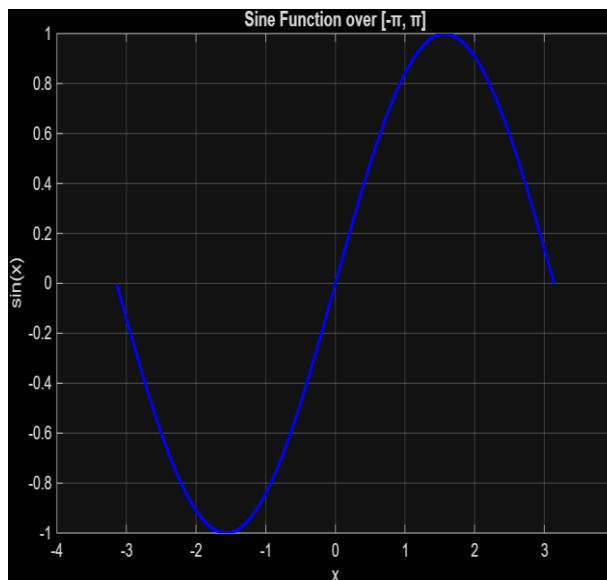


Figure 1 $y = \sin(x)$ plot

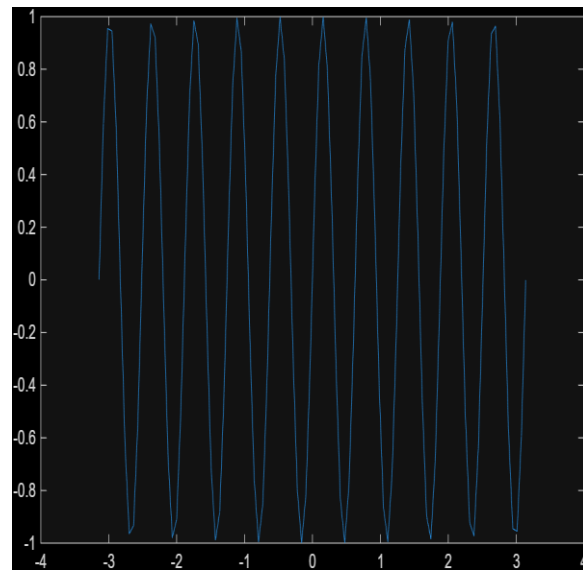


Figure 2 $y = \sin_scale(x, sf)$ plot (0.1 steps)

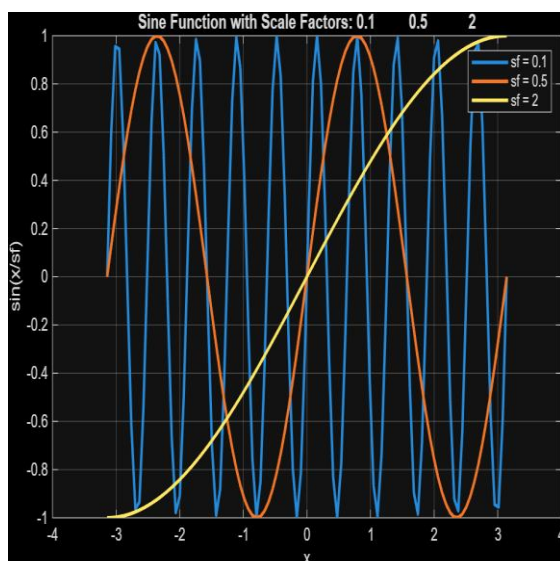


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

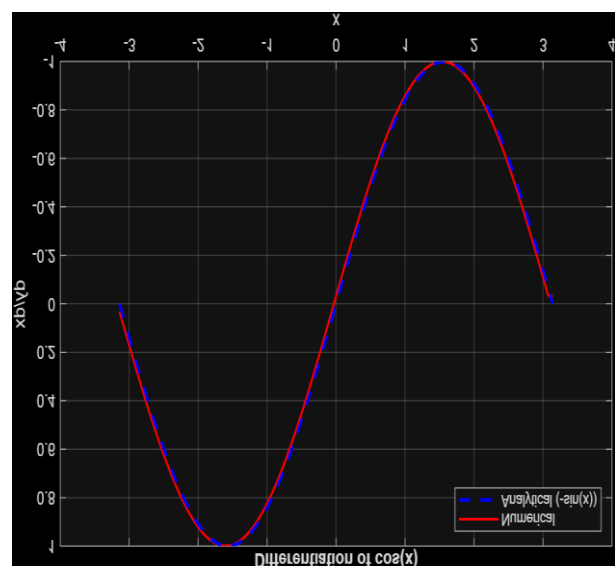


Figure 4 diff cos (x) to -sin(x)

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

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	1.000000	23.370055%

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'		0.6956349206	0.6931721793	0.6931534305	
'n iterations'	Na	5	50	100	
'percentage difference'	Na	0.358905%	0.003607%	0.000902%	

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'		0.6931697932	0.6931471856	0.6931471809	
'n iterations'	NA	5	50	100	
'percentage difference'	NA	0.358905%	0.003607%	0.000902%	

5 Integration results tables

5.1 Composite convergence iterations

n	Numerical	Percent Error
5	0.6956349206	0.358905%
50	0.6931721793	0.003607%
100	0.6931534305	0.000902%

5.2 Simpsons Convergence iterations

n	Numerical	Error
5	0.6931697932	2.261261e-05
50	0.6931471856	4.995009e-09
100	0.6931471809	3.124220e-10

Intervals needed for 6-decimal accuracy: 14

6 Appendices

6.1 Your code script and functions

1. lab1_sin.m (script)

```
1 % File: lab1_sin.m
2 clear
3 format short g
4
5 % Define x vector
6 N = 100;
7 x = linspace(-pi, pi, N)';
8
9 % Calculate y
10 y = sin(x);
11
12 % Plot
13 figure(1);
14 plot(x, y, 'b-', 'LineWidth', 2);
15 xlabel('x');
16 ylabel('sin(x)');
17 title('Sine Function over [-π, π]');
18 grid on;
19
```

2. sin_scale.m (script)

```
1 function y = sin_scale(x, sf)
2 % Input: x - input vector, sf - scale factor
3 % Output: y - scaled sine values
4
5 y = sin(x ./ sf); % ./ means element-wise division
6
7 end
8
9
10 % x = linspace(-pi, pi, 100)';
11 % y = sin_scale(x, 0.1);
12 % figure(2);
13 % plot(x, y);
14
```

3. differentiate_cos.m (function)

```
1 function dy = differentiate_cos1(x, y)
2 % Numerical differentiation using forward difference
3 %  $dy/dx \approx (y(i+1) - y(i)) / h$ 
4
5 h = x(2) - x(1); % step size
6 dy = zeros(size(x));
7
8 % Forward difference for all but last point
9 for i = 1:length(x)-1
10     dy(i) = (y(i+1) - y(i)) / h;
11 end
12
13 % For last point, use backward difference
14 dy(end) = dy(end-1);
15
```

4. trap1.m (script)

```

15     int_numerical = trap('integrand', a, b);
16
17     % Analytical:  $\int x \sin(x) dx = \sin(x) - x \cos(x)$ 
18     % At  $\pi/2$ :  $\sin(\pi/2) - (\pi/2) \cos(\pi/2) = 1 - 0 = 1$ 
19     int_analytical = 1;
20
21     % Error
22     error_percent = abs(int_numerical - int_analytical) / int_analytical;
23
24     fprintf('Numerical Integration Result:\n');
25     fprintf('Numerical value: %f\n', int_numerical);
26     fprintf('Analytical value: %f\n', int_analytical);
27     fprintf('Error: %f%%\n', error_percent);
28

```

5. trap_composite.m (function)

```

function int = trap_composite(f, a, b, n)
% Composite trapezoidal rule with n intervals

dx = (b - a) / n;
x = a:dx:b;
y = feval(f, x);

int = dx/2 * (y(1) + 2*sum(y(2:end-1)) + y(end));
end

```

6. run_trap.m (script)

```

clc; clear;

a = 1;
b = 2;
n_values = [5, 50, 100];

analytical = log(2); % Exact integral

fprintf('Composite Trapezoid Rule Results:\n');
fprintf('n\tNumerical\t\tPercent Error\n');

for n = n_values
    int_num = trap_composite('f_1_over_x1', a, b, n);
    percent_error = abs(int_num - analytical) / analytical * 100;
    fprintf('%d\t%.10f\t%.6f%%\n', n, int_num, percent_error);
end

```

7. integrand.m (function)

```

1 function y = integrand(x)
2 % y = x * sin(x)
3 y = x .* sin(x);
4 end
5
6

```

8. simpson.m (function)

```

1 function int = simpson(f, a, b, n)
2 % Simpson's rule with n intervals (n must be even)
3 % I ≈ (Δx/3)[f(a) + 4Σf(odd) + 2Σf(even) + f(b)]
4
5 if mod(n, 2) ~= 0
6     n = n + 1; % Make n even
7 end
8
9 dx = (b - a) / n;
10 x = a:dx:b;
11 y = f(x); % f is a function handle
12
13 % Odd indices: 2, 4, 6, ...
14 % Even indices: 3, 5, 7, ...
15 int = dx/3 * (y(1) + 4*sum(y(2:2:end-1)) + 2*sum(y(3:2:end-2)) +
16
17 end

```

9. simpson_rule.m (script)

```

1 clc; clear;
2
3 % Interval and exact value
4 a = 1;
5 b = 2;
6 analytical = log(2); % ∫12 (1/x) dx = ln(2)
7
8 % Define f(x) = 1/x as an anonymous function
9 f = @(x) 1 ./ x;
10
11 % Values of n to test
12 n_values = [5 50 100];
13
14 fprintf('\nSimpson Rule Results:\n');
15 fprintf('n\tNumerical\t\tError\n');
16
17 for n = n_values
18     int_num = simpson(f, a, b, n); % pass function handle
19     err = abs(int_num - analytical);
20     fprintf('%d\t%.10f\t%.2e\n', n, int_num, err);
21 end

```

10. f_1_over_x1.m (function)

```
function y = f_1_over_x1(x)
y = 1 ./ x;
end
```