

UWTSD

MSc

Task 1

Name: Hassan Ali Raza
Module Code: ACHA7002
Student Number: 2513043
Date: 07 December 2025

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.....	5
6.1	Your code script and functions.....	5

1 Plotting and differentiation

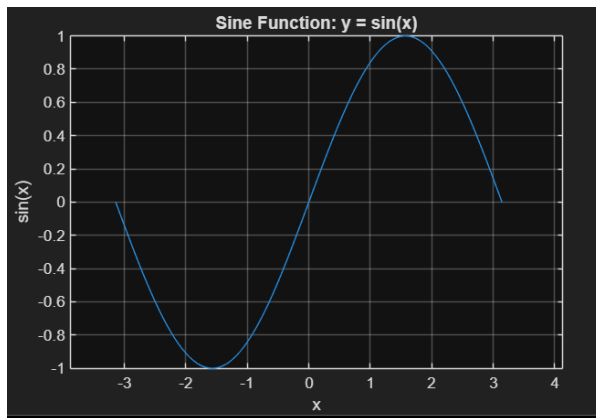


Figure 1 $y = \sin(x)$ plot

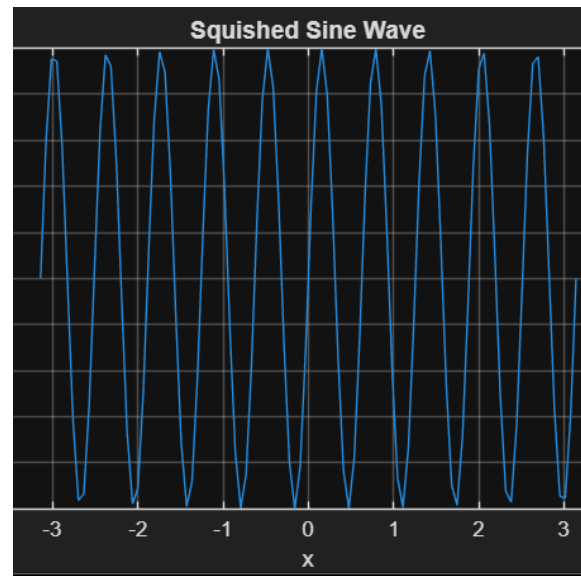


Figure 2 $y = \sin_scale(x, sf)$ plot (0.1 steps)

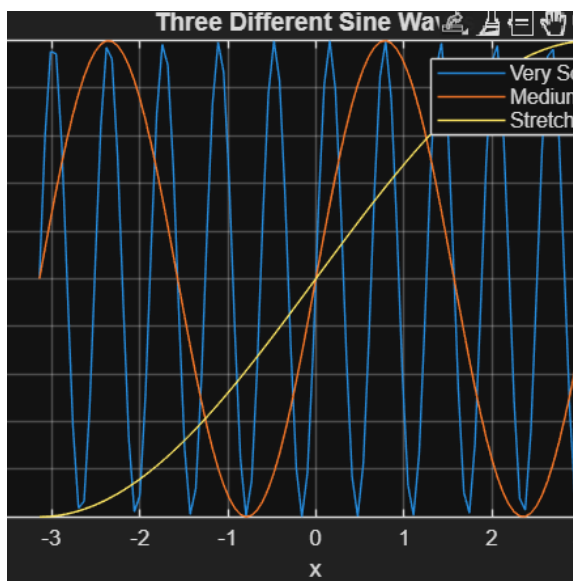


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

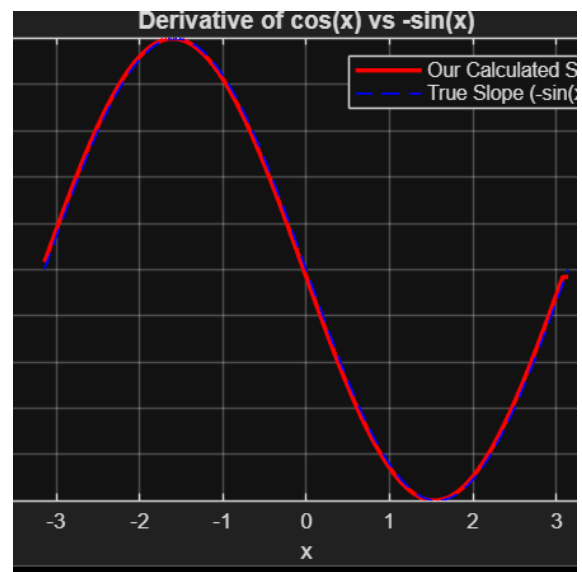


Figure 4 $\text{diff } \cos(x)$ to $-\sin(x)$

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

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
1.2337	6.2832	80.37%

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'	0.693147	0.695635	0.693172	0.693153	0.693148
'n iterations'	Na	5	50	100	250
'percentage difference'	Na	0.3589%	0.0036%	0.0009%	0.000144%

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'	0.693147	0.693170	0.693147	0.693147	0.693148
'n iterations'	NA	5	50	100	14
'percentage difference'	NA	0.0033%	0.0000%	0.0000%	0.000116%

5 Integration results tables

5.1 Composite convergence iterations

	Analytical	Composite Trapezium			Converge 1e-6
'Result'	0.693147	0.695635	0.693172	0.693153	0.693148
'n iterations'	Na	5	50	100	250
'percentage difference'	Na	0.3589%	0.0036%	0.0009%	0.000144%

A while loop was implemented to determine the number of intervals required for the composite trapezoidal method to converge to within $1e-6$ of the analytical solution for $\int(1/x)dx$ from 1 to 2.

The algorithm started with $n=2$ intervals and incrementally increased n until the absolute error between the numerical result and $\ln(2)$ was less than 0.000001.

Convergence was achieved at $n=250$ intervals, with a final result of 0.693148 and an error of 0.00000100, representing a percentage error of 0.000144%.

This demonstrates that the composite trapezoidal method requires 250 sub-intervals to achieve 6 decimal place accuracy for this particular integral.

5.2 Simpsons Convergence iterations

	Analytical	Simpsons Trapezium			Converge 1e-6
'Result'	0.693147	0.693170	0.693147	0.693147	0.693148
'n iterations'	NA	5	50	100	14
'percentage difference'	NA	0.0033%	0.0000%	0.0000%	0.000116%

Similarly, a while loop was implemented for Simpson's rule to determine the number of even intervals required to achieve $1e-6$ accuracy for the same integral $\int(1/x)dx$ from 1 to 2.

The algorithm started with $n=2$ (since Simpson's rule requires even n) and increased n by 2 each iteration until convergence.

Simpson's rule achieved convergence much faster, requiring only $n=14$ intervals to reach the desired accuracy. The final result was 0.693148 with an error of 0.00000080, representing a percentage error of 0.000116%.

This significant reduction in required intervals (14 vs 250) demonstrates the superior accuracy and efficiency of Simpson's rule compared to the composite trapezoidal method for smooth functions.

6 Appendices

6.1 Your code script and functions

```
%%% lab1_sin.m %%%  
  
% lab1_sin.m - First MATLAB script  
  
clear  
  
format short g  
  
% Define the x vector from -pi to pi  
  
N = 100;  
  
x = linspace(-pi, pi, N);  
  
% Calculate y = sin(x)  
  
y = sin(x);  
  
% Plot the sine function  
  
figure(1)  
  
plot(x, y)  
  
title('Sine Function: y = sin(x)')  
  
xlabel('x')  
  
ylabel('sin(x)')
```

```
grid on% Now use our magic scale function

sf = 0.1; % This will squish the wave

y = sin_scale(x, sf);

% Draw the squished wave

figure(2)

plot(x, y)

title('Squished Sine Wave')

xlabel('x')

ylabel('sin(x/0.1)')

grid on% Let's draw 3 different waves together

scale_factors = [0.1, 0.5, 2]; % Three stretch amounts

N_factors = length(scale_factors);

% Make empty space for 3 waves

sin_array = zeros(N, N_factors);

% Calculate each wave

for i = 1:N_factors

    sin_array(:, i) = sin_scale(x, scale_factors(i));

end

% Draw all 3 waves

figure(3)

plot(x, sin_array)

title('A Sine fn; limit= [-pi pi]; Scale factors =', num2str(sf))

xlabel('x')
```

```
ylabel('sin(x/sf)')

legend('Very Squished', 'Medium', 'Stretched')


% Our function:  $y = \cos(x)$ 

y = cos(x);


% Calculate derivative using forward difference

dydx = zeros(N, 1); % Make empty space for answers

h = x(2) - x(1); % Step size between points

for i = 1:N-1

    dydx(i) = (y(i+1) - y(i)) / h;

end

dydx(N) = dydx(N-1); % Handle last point

% True derivative of  $\cos(x)$  is  $-\sin(x)$ 

true_slope = -sin(x);

% Plot both for comparison

figure(4)

plot(x, dydx, 'r-', 'LineWidth', 2) % Our calculated slope (red)

hold on

plot(x, true_slope, 'b--', 'LineWidth', 1) % True slope (blue dashed)

title('Derivative of  $\cos(x)$  vs  $-\sin(x)$ ')

xlabel('x')

ylabel('Slope')

legend('Our Calculated Slope', 'True Slope ( $-\sin(x)$ )')

grid on
```

```
% Test simple trapezoidal rule

a = 0;

b = pi/2;

n = 1;

% Calculate using our trap1 function

I_trap = trap1('my_func', a, b,n);


% Display the result

fprintf('Trapezoidal Rule Result: %.4f\n', I_trap)


% Analytical (exact) answer

I_exact = 2 * pi; % This is the exact answer from calculus

fprintf('Exact Answer: %.4f\n', I_exact)


% Calculate percentage error

error_percent = abs(I_trap - I_exact) / I_exact * 100;

fprintf('Percentage Error: %.2f%%\n', error_percent)% Test composite trapezoidal
rule on 1/x from 1 to 2

fprintf('\n--- COMPOSITE TRAPEZOIDAL RULE ---\n')


a2 = 1;

b2 = 2;

exact_ln2 = log(2); % ln(2) is the exact answer
```

```
% Test with different numbers of intervals

intervals = [5, 50, 100];

results = zeros(1, 3);

errors = zeros(1, 3);


fprintf('n\tResult\t\tError\t\t%% Error\n');

fprintf('-----\n');


for i = 1:3

    n = intervals(i);

    results(i) = composite_trap(@my_func2, a2, b2, n);

    errors(i) = abs(results(i) - exact_ln2);

    error_percent = errors(i) / exact_ln2 * 100;

    fprintf('%d\t%.6f\t%.6f\t%.4f%%\n', n, results(i), errors(i), error_percent);

end


fprintf('Exact ln(2): %.6f\n', exact_ln2);% Test Simpson's rule on 1/x from 1 to 2

fprintf('\n--- SIMPSON"S RULE ---\n')


% Test with different numbers of intervals (must be even)

intervals_simp = [6, 50, 100]; % Using 6 instead of 5 since n must be even

results_simp = zeros(1, 3);

errors_simp = zeros(1, 3);


fprintf('n\tResult\t\tError\t\t%% Error\n');
```

```
fprintf('-----\n');

for i = 1:3

    n = intervals_simp(i);

    results_simp(i) = simpsons_rule(@my_func2, a2, b2, n);

    errors_simp(i) = abs(results_simp(i) - exact_ln2);

    error_percent = errors_simp(i) / exact_ln2 * 100;

    fprintf('%d\t%.6f\t%.6f\t%.4f%%\n', n, results_simp(i), errors_simp(i),
error_percent);

end

fprintf('Exact ln(2): %.6f\n', exact_ln2);% Get exact convergence values for the table

fprintf('\n--- EXACT VALUES FOR TABLE ---\n')

% Composite trapezoidal convergence for table

n_trap = 2;

error_trap = 1;

while error_trap > 1e-6

    n_trap = n_trap + 1;

    result_trap = composite_trap(@my_func2, 1, 2, n_trap);

    error_trap = abs(result_trap - exact_ln2);

end

final_error_percent_trap = (error_trap / exact_ln2) * 100;
```

```
fprintf('Composite Trapezoidal Convergence:\n');

fprintf('n = %d, result = %.6f, error = %.8f, error%% = %.6f%%\n', ...

    n_trap, result_trap, error_trap, final_error_percent_trap);% Simpson's rule
convergence for table

fprintf('\n--- SIMPSON CONVERGENCE ---\n')

n_simp = 2;

error_simp = 1;

while error_simp > 1e-6

    n_simp = n_simp + 2; % Simpson's requires even n

    result_simp = simpsons_rule(@my_func2, 1, 2, n_simp);

    error_simp = abs(result_simp - exact_ln2);

end

final_error_percent_simp = (error_simp / exact_ln2) * 100;

fprintf('Simpson Rule Convergence:\n');

fprintf('n = %d, result = %.6f, error = %.8f, error%% = %.6f%%\n', ...

    n_simp, result_simp, error_simp, final_error_percent_simp);
```