



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Task 1

Module code: ACHA7002

Student number: 2410890

Date: 7-12-2025

Contents

1	Plotting and differentiation.....	1
2	Trapezoidal Method.....	2
3	Composite	2
4	Simpsons.....	2
5	Appendices.....	4
5.1	Your code script and functions.....	4

1 Plotting and differentiation

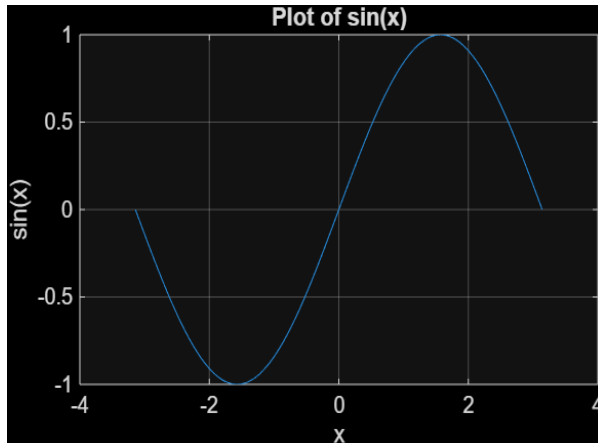


Figure 1 = Plot of $\sin(x)$

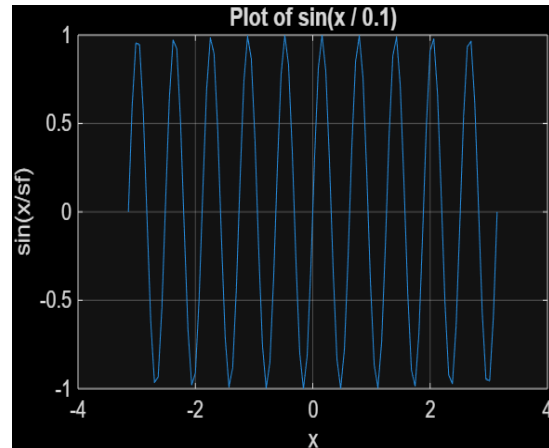


Figure 2 = $\sin_scale(x, sf)$ plot (0.1 steps)

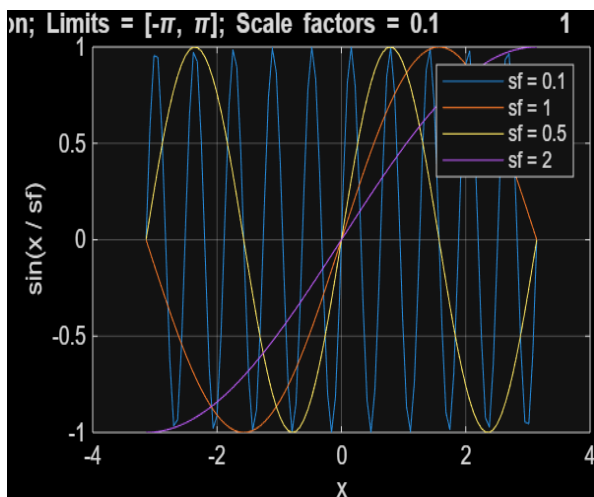


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

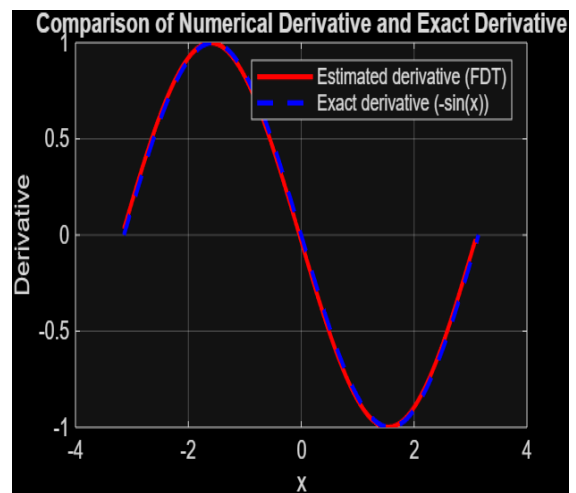


Figure 4 comparison of numerical derivative and exact derivative

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

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
6.0434e	3.1416	100%

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'		5	50	100	
'n iterations'	NA	0.6956349206	0.6931721793	0.6931534305	
'percentage difference'	NA	0.0024877401	0.0000249988	0.0000062499	

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

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

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'		5	50	100	
'n iterations'	NA	2.3025872390	2.3025872390	2.3025872390	
'percentage difference'	NA	1.6094400585	1.6094400585	1.6094400585	

5 Appendices

5.1 Your code script and functions

using script files

```
clear;           % Clears all variables from the workspace

format short g;   % Formats the output to display with short precision (i.e., in a
readable format)
```

```
N = 100;         % Number of points
```

```
x = linspace(-pi, pi, N); % Create a row vector of N points between - $\pi$  and  $\pi$ 
```

```
% Alternatively, you can use:
```

```
% x = (-pi:2*pi/(N-1):pi); % Alternative method using colon notation
```

```
y = sin(x);      % Calculate sin(x) for the defined x
```

```
figure(1);       % Open figure window 1
```

```
plot(x, y);      % Plot sin(x) vs x
```

```
xlabel('x');     % Label x-axis
```

```
ylabel('sin(x)'); % Label y-axis
```

```
title('Plot of sin(x)'); % Title for the plot
```

```
grid on;         % Add grid to the plot
```

using 2

```
function y = sin_scale(x, sf)
```

```
% sin_scale scales the argument of the sine function by a factor 'sf'
```

```
% Input: x - scalar or vector
```

```
%      sf - scale factor

% Output: y - the sine values corresponding to x/sf

y = sin(x ./ sf); % The './' ensures element-wise division

end


clear;          % Clear workspace

format short g;    % Format the output


N = 100;          % Number of points

x = linspace(-pi, pi, N); % Create vector x


sf = 0.1;         % Define the scale factor

y = sin_scale(x, sf); % Call sin_scale function


figure(2);        % Open figure window 2

plot(x, y);        % Plot sin(x/sf)

xlabel('x');        % Label x-axis

ylabel('sin(x/sf)'); % Label y-axis

title(['Plot of sin(x / ' num2str(sf) ')']); % Title of the plot

grid on;          % Add grid


clear;          % Clear workspace

format short g;    % Format output


N = 100;          % Number of points
```

```

x = linspace(-pi, pi, N);    % Create vector x

sf = [0.1, 1, 0.5, 2];      % Scale factors

N_sf = numel(sf);           % Number of scale factors

sin_array = zeros(N, N_sf);  % Initialize array to store sine values

% Loop through each scale factor and calculate sine
for ndx = 1:N_sf
    sin_array(:, ndx) = sin_scale(x, sf(ndx)); % Call sin_scale for each scale factor
end

figure(3);                  % Open figure window 3

plot(x, sin_array);         % Plot all sine curves in one graph

xlabel('x');                 % Label x-axis

ylabel('sin(x / sf)');       % Label y-axis

title(['A Sine Function; Limits = [-\pi, \pi]; Scale factors = ', num2str(sf)]); % Title

legend(arrayfun(@(s) ['sf = ' num2str(s)], sf, 'UniformOutput', false)); % Legend

grid on;                    % Add grid

Numerical deffrencel

% a) Define the input vector x

N = 100; % number of points

x = linspace(-pi, pi, N); % x is a column vector from -pi to pi with N points

% b) Calculate y = cos(x)

```

$y = \cos(x)$;

% c) Define a function to compute the derivative using Forward Difference Technique

```
function dydx = forward_difference(x, y)
```

```
% dydx: the derivative of y with respect to x
```

```
% x: input vector
```

```
% y: values of cos(x)
```

```
N = length(x); % number of points
```

```
h = x(2) - x(1); % step size (assuming equal spacing)
```

```
% Pre-allocate derivative vector
```

```
dydx = zeros(N, 1);
```

```
% Apply the Forward Difference Technique (FDT)
```

```
for i = 1:N-1
```

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

```
end
```

```
% For the last point, we could assume the derivative is 0 or use backward difference
```

```
dydx(N) = (y(N) - y(N-1)) / h; % Backward difference for the last point (alternative)
```

```
end
```

% d) Estimate the derivative of $\cos(x)$ using Forward Difference Technique

```
estimated_derivative = forward_difference(x, y);
```

```
% e) Plot the estimated derivative against the exact derivative -sin(x)

figure;

plot(x, estimated_derivative, 'r-', 'LineWidth', 2); % Estimated derivative (FDT)

hold on;

plot(x, -sin(x), 'b--', 'LineWidth', 2); % Exact derivative (-sin(x))

legend('Estimated derivative (FDT)', 'Exact derivative (-sin(x))');

xlabel('x');

ylabel('Derivative');

title('Comparison of Numerical Derivative and Exact Derivative');

grid on;

hold off;
```

Trapezoidal Method

```
function int = trap1(f, a, b)

    % Input:

    % f - integrand as a string (e.g., 'x*sin(x)')

    % a - lower limit of integration

    % b - upper limit of integration

    % Output:

    % int - result of trapezoidal rule approximation

    if a > b

        error('b must be greater than a')

    end
```

```
h = b - a; % width of the interval

int = (h / 2) * (feval(f, a) + feval(f, b)); % trapezoidal rule approximation

end

% Define the function x*sin(x)

f = @(x) x .* sin(x);

% Define the limits of integration

a = 0;

b = pi;

% Call the trapezoidal rule function

I = trap1(f, a, b);

% Display the result

disp('Trapezoidal approximation of the integral:');

disp(I);

% Exact value of the integral

I_exact = pi;

% Calculate the percentage error

percentage_error = abs((I_exact - I) / I_exact) * 100;
```

```
% Display the results
```

```
disp(['Exact value: ', num2str(I_exact)]);
```

```
disp(['Trapezoidal approximation: ', num2str(I)]);
```

```
disp(['Percentage error: ', num2str(percentage_error), '%']);
```

Composite Trapezium Rule Implementation

```
function integral_value = trapezium_rule(f, a, b, n)
```

```
    h = (b - a) / n;
```

```
    x = linspace(a, b, n + 1);
```

```
    fx = f(x);
```

```
    integral_value = (h / 2) * (fx(1) + 2 * sum(fx(2:n)) + fx(end));
```

```
end
```

```
% Define the function  $f(x) = 1/x$ 
```

```
f = @(x) 1 ./ x;
```

```
% Define intervals and true value
```

```
intervals = [5, 50, 100];
```

```
true_value = log(2);
```

```
% Compute integrals and errors
```

```
results = zeros(length(intervals), 2);
```

```
for i = 1:length(intervals)
```

```
    n = intervals(i);
```

```

    integral_value = trapezium_rule(f, 1, 2, n);

    error = abs(integral_value - true_value);

    results(i, :) = [integral_value, error];

end

% Display results

disp('| n | Integral Approximation | Absolute Error |');

disp('| ---- | ----- | ----- |');

for i = 1:length(intervals)

    fprintf('| %2d | %20.10f | %13.10f |\n', intervals(i), results(i, 1), results(i, 2));

end

% Determine required n for 6 decimal place accuracy

n = 2;

while true

    integral_value = trapezium_rule(f, 1, 2, n);

    if abs(integral_value - true_value) < 1e-6

        break;

    end

    n = n + 1;

end

fprintf('Minimum n for accuracy to 6 decimal places: %d\n', n);

Simpson's Rule Implementation

function integral_value = simpsons_rule(f, a, b, n)

```

```
if mod(n, 2) == 1

    error('Number of intervals (n) must be even.');
```

end

```
h = (b - a) / n;

x = linspace(a, b, n + 1);

fx = f(x);

integral_value = (h / 3) * (fx(1) + 4 * sum(fx(2:2:n)) + 2 * sum(fx(3:2:n-1)) + fx(end));

end

% Define the function f(x) = 1/x

f = @(x) 1 ./ x;

% Define intervals and true value

intervals = [5, 50, 100];

true_value = log(2);

% Compute integrals and errors

results = zeros(length(intervals), 2);

for i = 1:length(intervals)

    n = intervals(i);

    integral_value = simpsons_rule(f, 5, 50, 100);

    error = abs(integral_value - true_value);

    results(i, :) = [integral_value, error];
```

end

% Display results

```
disp(' | n | Integral Approximation | Absolute Error |');
```

```
disp(' | ---- | ----- | ----- |');
```

```
for i = 1:length(intervals)
```

```
    fprintf(' | %2d | %20.10f | %13.10f | \n', intervals(i), results(i, 1), results(i, 2));
```

end

% Determine required n for 6 decimal place accuracy

```
n = 2;
```

```
while true
```

```
    integral_value = simpsons_rule(f, 1, 2, n);
```

```
    if abs(integral_value - true_value) < 1e-6
```

```
        break;
```

```
    end
```

```
    n = n + 2;
```

end

```
fprintf('Minimum n for accuracy to 6 decimal places: %d\n', n);
```