

Task 1

please convert this to pdf for submission

Function have been put at the end

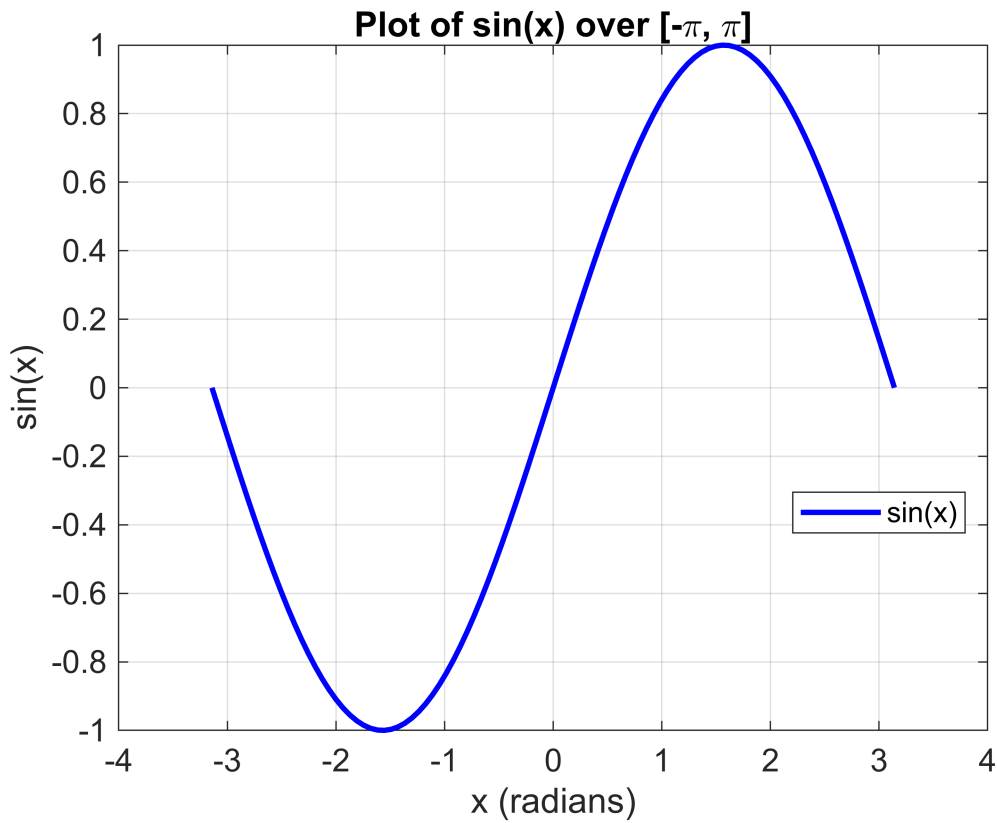
Table of Contents

(i) (e) $y = \sin(x)$ plot [1].....	1
(ii) (e) $y = \sin_scale(x,sf)$ plot [1].....	2
(ii) (e) $y = \sin$ array plot [3].....	3
Numerical Differentiation $y=\cos(x)$ plot [3].....	4
Numerical Intergration (a) $\int \sin(x) dx$ [3].....	5
Numerical Intergration (b) compare [3].....	6
Comp Trap (c) $1/x$ for 5,50,100 [4].....	7
Comp Trap (d) compare [1].....	9
Comp Trap (e) to tolerance [3].....	9
Simpsons (b) $1/x$ for 5,50,100 [4].....	9
Simpsons (c) compare [1].....	10
Simpsons (d) to tolerance [3].....	10

(i) (e) $y = \sin(x)$ plot [1]

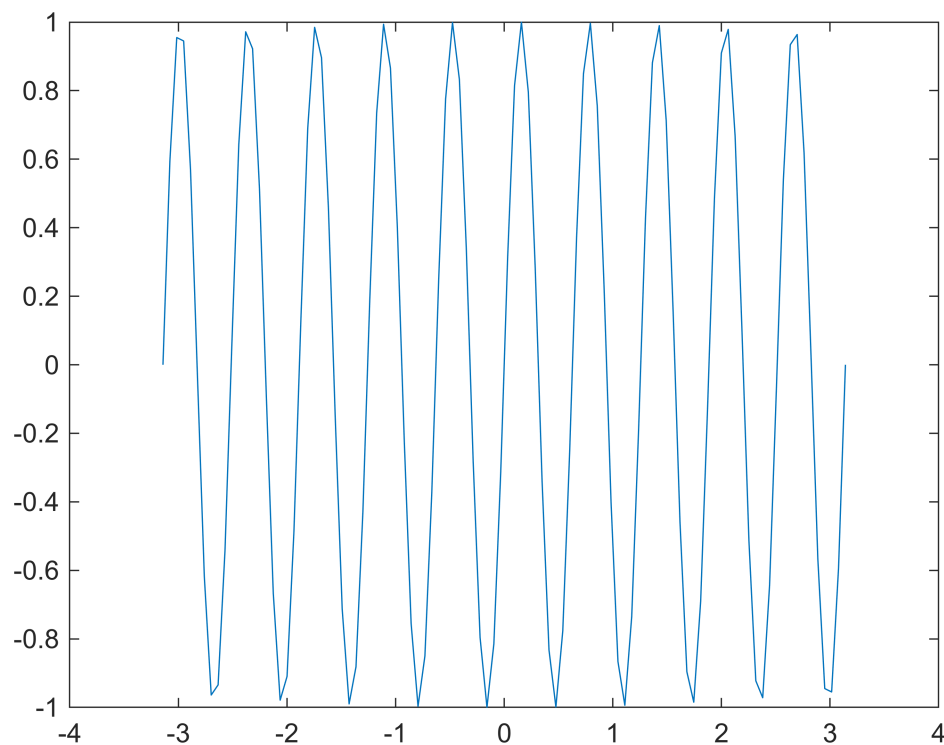
```
clear, clc, clear
format short g
N = 100;
x = linspace(-pi,pi,N)'; % sine function over the interval  $[-\pi \pi]$ 
x = (-pi:2*pi/(N-1):pi)';

y = sin(x);
figure(1)
plot(x, y, 'b-', 'LineWidth', 2) % Blue solid line with increased width
grid on
xlabel('x (radians)', 'FontSize', 12)
ylabel('sin(x)', 'FontSize', 12)
title('Plot of sin(x) over  $[-\pi, \pi]$ ', 'FontSize', 14)
ax = gca;
ax.FontSize = 12; % Increase font size for better readability
legend('sin(x)', 'Location', 'best')
```



(ii) (e) $y = \sin_scale(x, sf)$ plot [1]

```
% Defining a function (II)
sf = 0.1 ;
y = sin_scale(x,sf); % scale x by 0.1
figure(2),
plot(x,y)
```



(ii) (e) $y = \sin$ array plot [3]

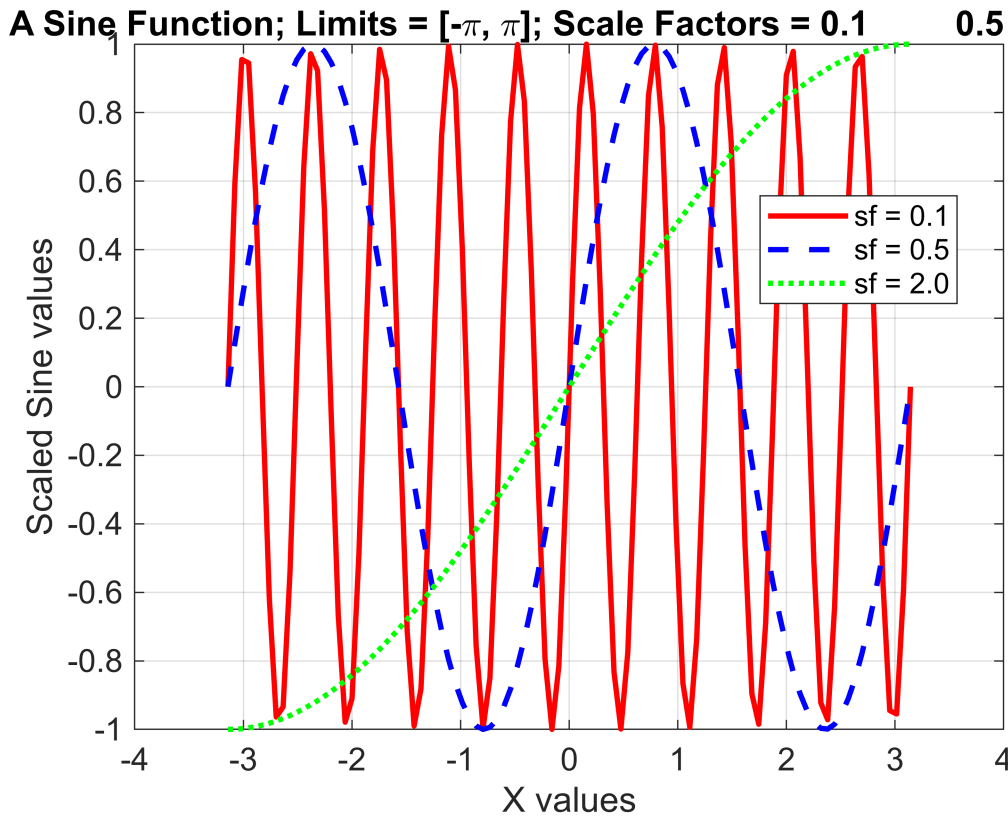
```
%%
sf=[0.1 0.5 2];           % Scale factor values in an array named sf
N_sf = numel(sf);         % Number of elements within sf (ML intrinsic % function)

sin_array = zeros(N,N_sf);
%Initialise an array of zeros to store data. %

for ndx = 1: N_sf
    sin_array(:,ndx) = sin_scale(x,sf(ndx));
end
figure(3)
plot(x, sin_array(:,1), 'r-', 'LineWidth', 2) % Red solid line
hold on
plot(x, sin_array(:,2), 'b--', 'LineWidth', 2) % Blue dashed line
plot(x, sin_array(:,3), 'g:', 'LineWidth', 2) % Green dotted line
hold off

% Add Labels, Title, and Legend
xlabel('X values')
ylabel('Scaled Sine values')
title(['A Sine Function; Limits = [-\pi, \pi]; Scale Factors = ', num2str(sf)])
legend(arrayfun(@(s) sprintf('sf = %.1f', s), sf, 'UniformOutput', false),
'Location', 'Best')
```

```
% Improve Readability
grid on
set(gca, 'FontSize', 12) % Increase font size for readability
```

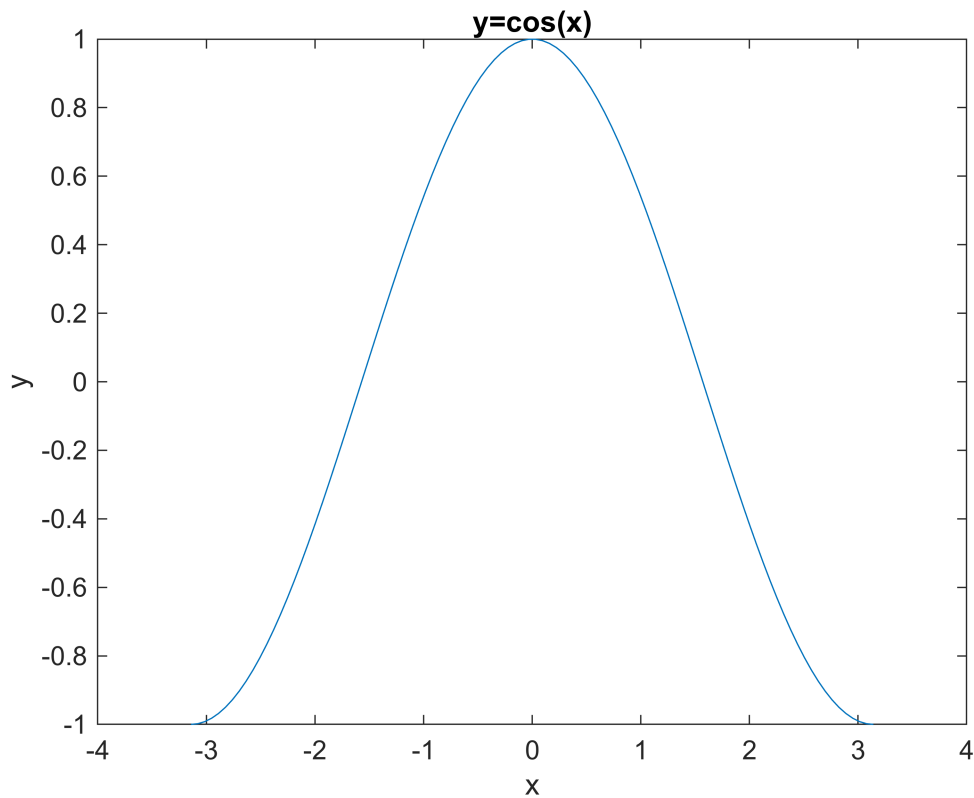


Numerical Differentiation $y=\cos(x)$ plot [3]

```
%% Numerical Differentiation
N = 100;

x = linspace(-pi,pi,N)';

function_y=cos(x);
% Plot function
figure(4)
plot(x,function_y)
xlabel('x')
ylabel('y')
title('y=cos(x)')
hold on
```



Numerical Intergration (a) $\frac{d}{dx} \sin(x) = \cos(x)$ [3]

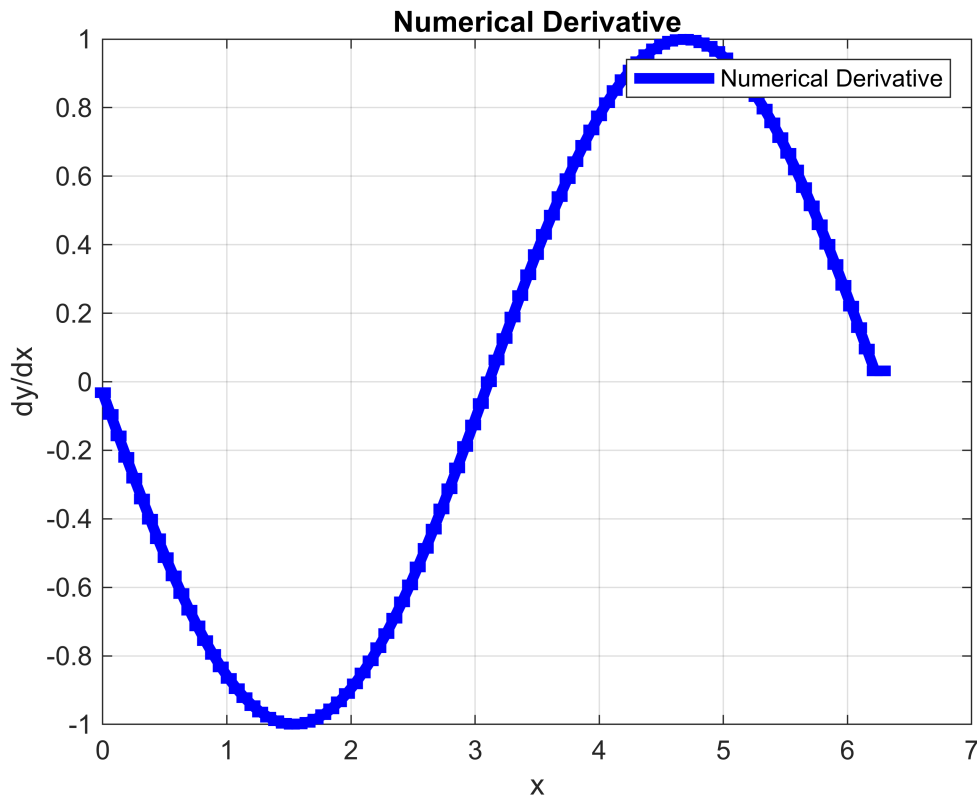
```
% Define x values
x = linspace(0, 2*pi, 100);

% Define function y = cos(x)
function_y = cos(x);

% Call the derivative function
dy_by_dx = numerical_derivative(x, function_y);

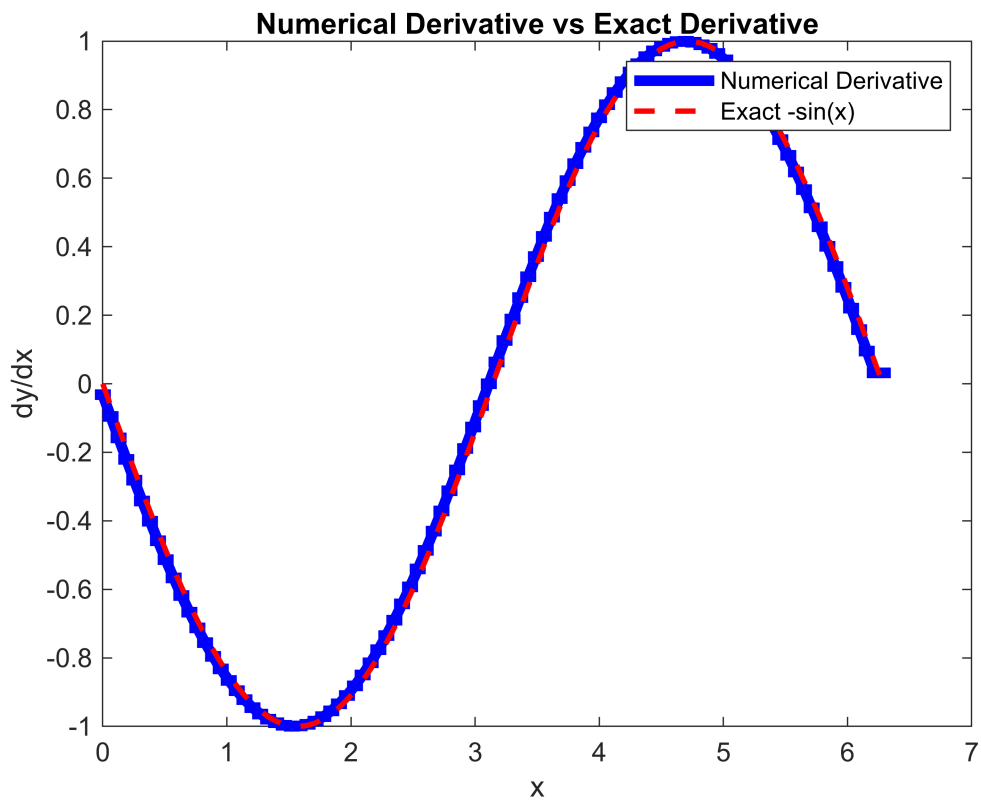
% Plot results
figure;
plot(x, dy_by_dx, 'b-', 'LineWidth', 4, 'Marker', '_'); % Numerical derivative

% Labels
xlabel('x');
ylabel('dy/dx');
title('Numerical Derivative');
legend('Numerical Derivative');
grid on;
```



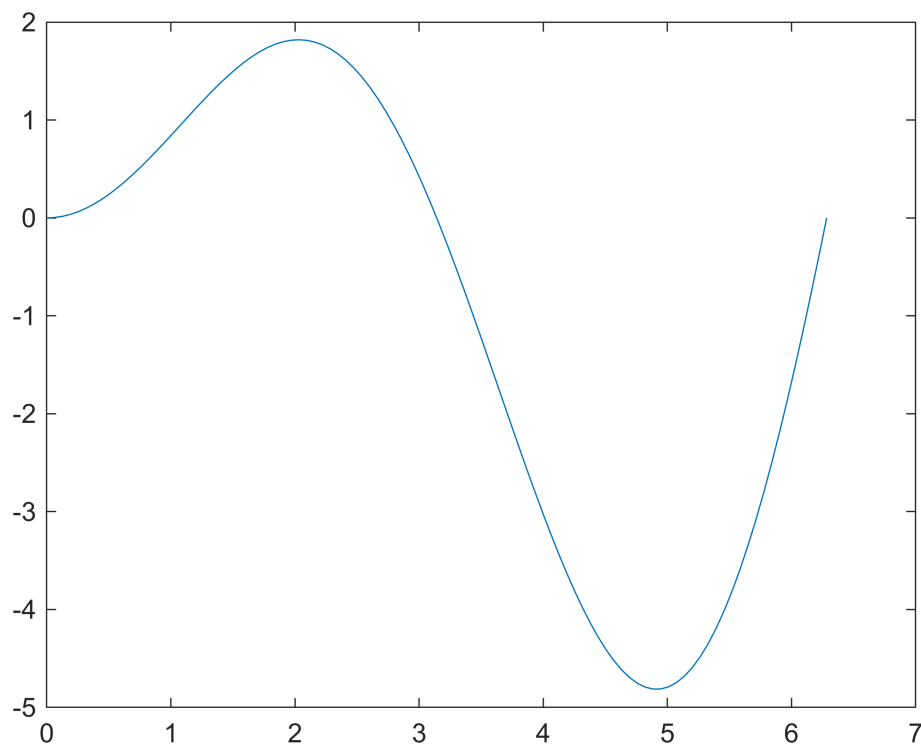
Numerical Intergration (b) compare [3]

```
plot(x, dy_by_dx, 'b-', 'LineWidth', 4, 'Marker', '_'); % Numerical derivative
hold on;
plot(x, -sin(x), 'r--', 'LineWidth', 2); % Exact derivative of cos(x) is -sin(x)
hold off;
% Labels
xlabel('x');
ylabel('dy/dx');
title('Numerical Derivative vs Exact Derivative');
legend('Numerical Derivative', 'Exact -sin(x)');
```



Comp Trap (c) 1/x for 5,50,100 [4]

```
%%  
f = @(x) x .* sin(x);  
a= 0;  
b = pi/2;  
I = int_Sin(x);  
plot(x, I)
```



```
TRAPEZ=trap1('int_Sin',a,b);
% per error
%%
syms x
I = int(x * sin(x), x, 0, pi/2);
difference = ((TRAPEZ-I)/I)* 100
```

```
difference =
    23.37
```

```
disp(['The difference is for trapezoidal Method: ', num2str(difference), '%']);
```

```
The difference is for trapezoidal Method: 23.3701%
```

```
%% Composite Trapezium Rule
Composite = Comp(f, a, b, N); %function
% diff needs to be worked out
%%
intervals = [5, 50, 100];% intervals
[results_trap] = evaluate_function(intervals); % function
% Display results
disp('Trapezoidal Rule Results:');
```

```
Trapezoidal Rule Results:
```

```
disp(results_trap);
```

```
0.69563    0.69317    0.69315
```

```

intervals = [5, 50, 100]; %intervals
results_trap = evaluate_function(intervals);% function
analytical_result = log(2); % analytical result

TRAPEZ=trap1('int_Sin',a,b);
% per error

```

Comp Trap (d) compare [1]

```

% Prepare the table
comparison_table = table(intervals', results_trap', ...
    repmat(analytical_result, size(intervals')), ...
    'VariableNames', {'Intervals', 'Trapezoidal_Result', 'Analytical_Result'});

% Display the comparison table
disp('Comparison of Numerical Integration Results with Analytical Result:');

```

Comparison of Numerical Integration Results with Analytical Result:

```
disp(comparison_table);
```

Intervals	Trapezoidal_Result	Analytical_Result
5	0.69563	0.69315
50	0.69317	0.69315
100	0.69315	0.69315

Comp Trap (e) to tolerance [3]

```

%% Number of sub intervals needed to achieve accuracy to 6 decimal places for
composite trapezium rule
f = @(x) 1 ./ x;
a = 1;
b = 2;
tolerance = 1e-6;
n1 = 1;
while true
    Composite = Comp(f, a, b, n1);
    if abs( Composite - analytical_result) <= tolerance
        break;
    end
    n1 = n1 + 1;
end
disp(['Number of subintervals (n) used for trapezium rule: ', num2str(n1)]);

```

Number of subintervals (n) used for trapezium rule: 250

Simpsons (b) 1/x for 5,50,100 [4]

```
a = 1; % Lower limit
```

```

b = 2;           % Upper limit
n_values = [5,50,100]; % intervals

% array to store results
S = zeros(size(n_values));

% Loop over each value of n and calculate the integral
for i = 1:length(n_values)
    n = n_values(i);
    S(i) = Simpson(f, a, b, n); % Call your Simpson function
end

```

Simpsons (c) compare [1]

```

analytical_result = log(2); % Analytical result for comparison
% Create a table to display the results
results_table = table(n_values', S', repmat(analytical_result, size(S')), ...
    'VariableNames', {'n', 'Numerical_Result', 'Analytical_Result'});
% Display the table
disp("Numerical Result and analytical result for simpson")

```

Numerical Result and analytical result for simpson

```
disp(results_table);
```

n	Numerical_Result	Analytical_Result
5	0.6582	0.69315
50	0.69315	0.69315
100	0.69315	0.69315

Simpsons (d) to tolerance [3]

```

f = @(x) 1 ./ x; % Define the function
a = 1;           % Lower limit
b = 2;           % Upper limit
analytical_result = log(2); % Analytical result ln(2)
tolerance = 1e-6; % Desired accuracy to six decimal places
n = 2;           % Start

% Iteratively increase n
while true
    I = compositeSimpson(f, a, b, n);
    if abs(I - analytical_result) <= tolerance
        break;
    end
    n = n + 1;
end
% Display the result and number of intervals

```

```
disp(['Approximate integral: ', num2str(I)]);
```

Approximate integral: 0.69315

```
disp(['Number of subintervals (n) used: ', num2str(n)]);
```

Number of subintervals (n) used: 14

All functions can be find below

```
function int= int_Sin(x)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
int = x.*sin(x);
end
```

```
function [y]=my_funcnt(x)
    y=x*sin(x);
end
function y = sin_scale (x,sf)
%UNTITLED4 Summary of this function goes here
% Detailed explanation goes here
y = sin(x./sf);
end
```

numerical derivative

```
function dy_by_dx = numerical_derivative(x, function_y) % Numerical Intergration
(a)  $\int \sin(x) dx$  [3]
% Compute numerical derivative using finite differences
dy_by_dx = zeros(length(x), 1); % Preallocate derivative array

for i = 1:length(x)-1
    delta_y = function_y(i+1) - function_y(i);
    delta_x = x(i+1) - x(i);

    % Store the result in dy_by_dx
    dy_by_dx(i) = delta_y / delta_x;
end

% For the last point, copy the previous value to maintain array size
dy_by_dx(end) = dy_by_dx(end-1);
end
```

trap1 function

```
function TRAPEZ=trap1(I,a,b)

if a>b
error("b must be greater than a")
end
h=(b-a);
TRAPEZ=h*(feval(I,a)+feval(I,b))/2;

end
```

Comp

```
function Composite = Comp(f, a, b, n1)
% Step size
h = (b - a) / n1;

% Calculate the function values at the endpoints
f_a = f(a);
f_b = f(b);

% Initialize the sum
sum_f = 0;

% Calculate the sum of the intermediate points
for i = 1:n1-1
    x_i = a + i * h;
    sum_f = sum_f + f(x_i);
end

% Apply the trapezoidal rule
Composite = (h / 2) * (f_a + 2 * sum_f + f_b);

end
```

Simpsons rule

```
function S = Simpson(f, a, b, n)

% Calculate  $\Delta x$ 
delta_x = (b - a) / n;

% Initialize the function values
f_a = f(a);
f_b = f(b);
sum_f_i_odd = 0; % For 4 * sum(f_i) where i is odd
```

```

sum_f_i_even = 0; % For 2 * sum(f_i) where i is even

% for all intervals
for i = 1:n-1
    x_i = a + i * delta_x; % Midpoint for each sub-interval
    f_i = f(x_i);

    if mod(i, 2) == 1 % odd index
        sum_f_i_odd = sum_f_i_odd + f_i;
    else % even index
        sum_f_i_even = sum_f_i_even + f_i;
    end
end

% Combine the sums into the final approximation
S = (delta_x / 3) * (f_a + 4 * sum_f_i_odd + 2 * sum_f_i_even + f_b);
end

```

Simpsons tol

```

function I = compositeSimpson(f, a, b, n)
% Step size
dx = (b - a) / n;

% Calculate the composite Simpson's approximation
sum1 = 0; % sum for terms with coefficient 4
sum2 = 0; % sum for terms with coefficient 2

for i = 1:2:n-1
    sum1 = sum1 + f(a + i * dx);
end

for i = 2:2:n-2
    sum2 = sum2 + f(a + i * dx);
end

% Composite Simpson's Rule formula
I = (dx / 3) * (f(a) + 4 * sum1 + 2 * sum2 + f(b));
format longG

end

```