

Engine Performance and Emissions, ACEA7005, Assignment 1

Task 1

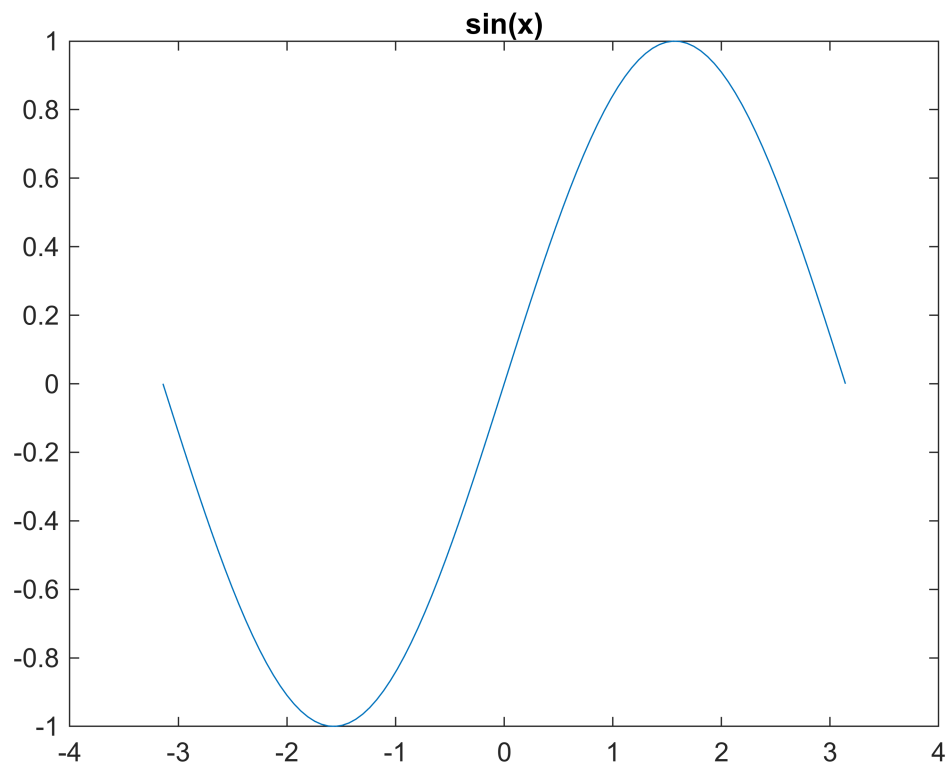
Table of Contents

(i) $y = \sin(x)$ [1].....	1
(ii) $y = \sin_scale(x,sf)$ [1].....	2
(ii) $y = \sin$ array [3].....	3
Numerical Differentiation $y=\cos(x)$ plot [3].....	4
Numerical Intergration (a) $\int \sin(x) dx$ [3].....	6
Numerical Integration (b) compare [3].....	7
Comp Trap (c) $1/x$ for 5,50,100 [4].....	9
Comp Tramp (d) compare [1].....	9
Comp Trap (e) to tolerance [3].....	10
Simpsons (b) $1/x$ for 5,50,100 [4].....	11
Simpsons (c) compare [1].....	11
Simpssons (d) to tolerance [3].....	12

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

```
clear %clear all variables from the workspace
format short g %formats the outputted numerical data

N = 100;
x = linspace(-pi,pi,N);
y = sin(x);
plot (x,y)
title("sin(x)")
```



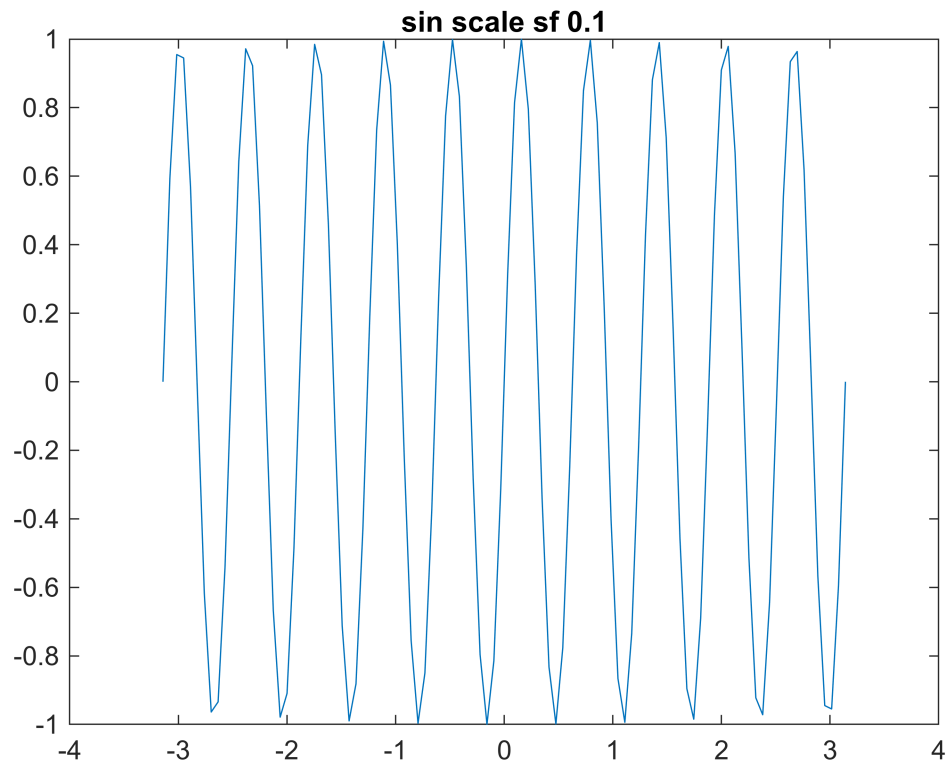
(ii) **y = sin_scale(x,sf) [1]**

```
close;
clear;
clc;

N = 100;
x = linspace(-pi,pi,N);

function y = sin_scale(x,sf)
y = sin(x./sf);
end

sf = 0.1;
y = sin(x./sf);
plot(x,y)
title("sin scale sf 0.1")
```

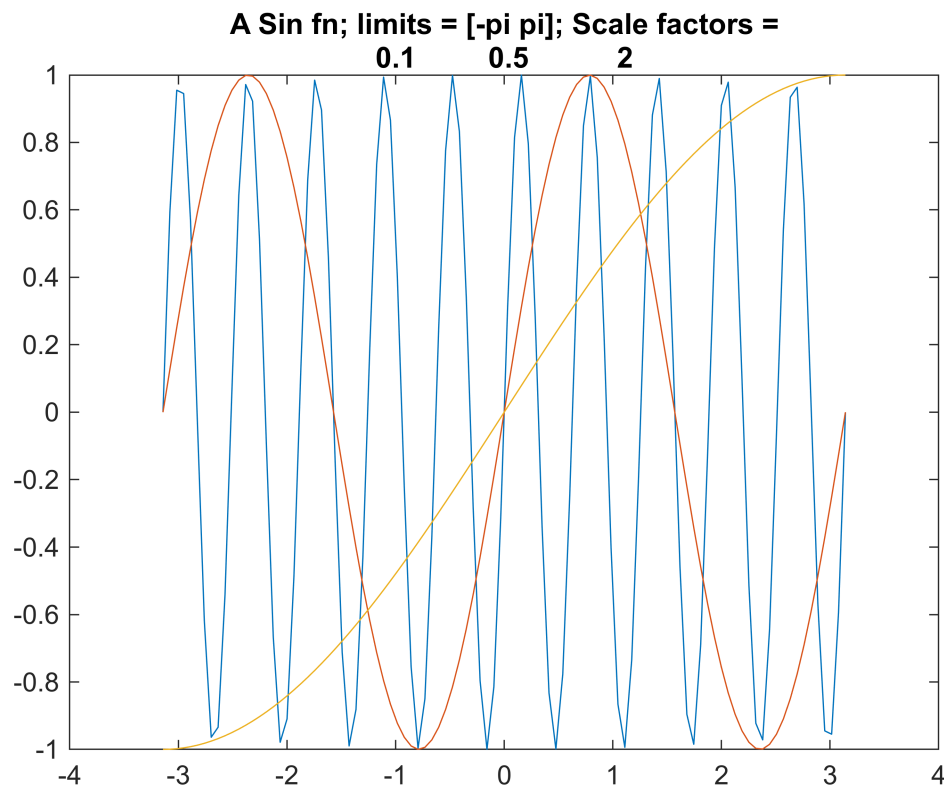


(ii) $y = \sin$ array [3]

```
sf = [0.1, 0.5, 2];
N_sf = numel(sf);
sin_array = zeros(N,N_sf);

for ndx = 1:N_sf
    sin_array(:,ndx) = sin_scale(x,sf(ndx));
end

figure(3);
plot(x,sin_array);
title(["A Sin fn; limits = [-pi pi]; Scale factors =",num2str(sf)]);
```



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

```
close all;
clear;
clc;

% Step (a): Define the input vector x
N = 100; % Number of points
x = linspace(-pi, pi, N)'; % Input vector from -pi to pi with N points

% Step (b): Calculate y = cos(x)
y = cos(x);

% Step (c): Define a function that inputs both x and y vectors
% (This function will calculate the forward difference derivative)

function dy_dx = forward_difference(x, y)
    h = x(2) - x(1); % Step size, assuming constant
    N = length(x);    % Number of points
    dy_dx = zeros(N, 1); % Preallocate the derivative vector

    % Apply the forward difference formula for the first N-1 points
    for i = 1:N-1
```

```

dy_dx(i) = (y(i+1) - y(i)) / h;
end
% For the last point, we cannot use forward difference, so we set it as NaN
dy_dx(N) = NaN; % You may handle this differently if needed
end

% Step (d): Use the forward difference to estimate the derivative
dy_dx_estimate = forward_difference(x, y);

% Step (e): Plot the estimated derivative and compare with the analytical solution
figure;
plot(x, dy_dx_estimate, 'r', 'DisplayName', 'Numerical Derivative (FDT)');

hold on;

plot(x, -sin(x), 'b--', 'DisplayName', 'Analytical Derivative (-sin(x))');
plot(x, cos(x), "b", "DisplayName", "Cos(x)")
legend;

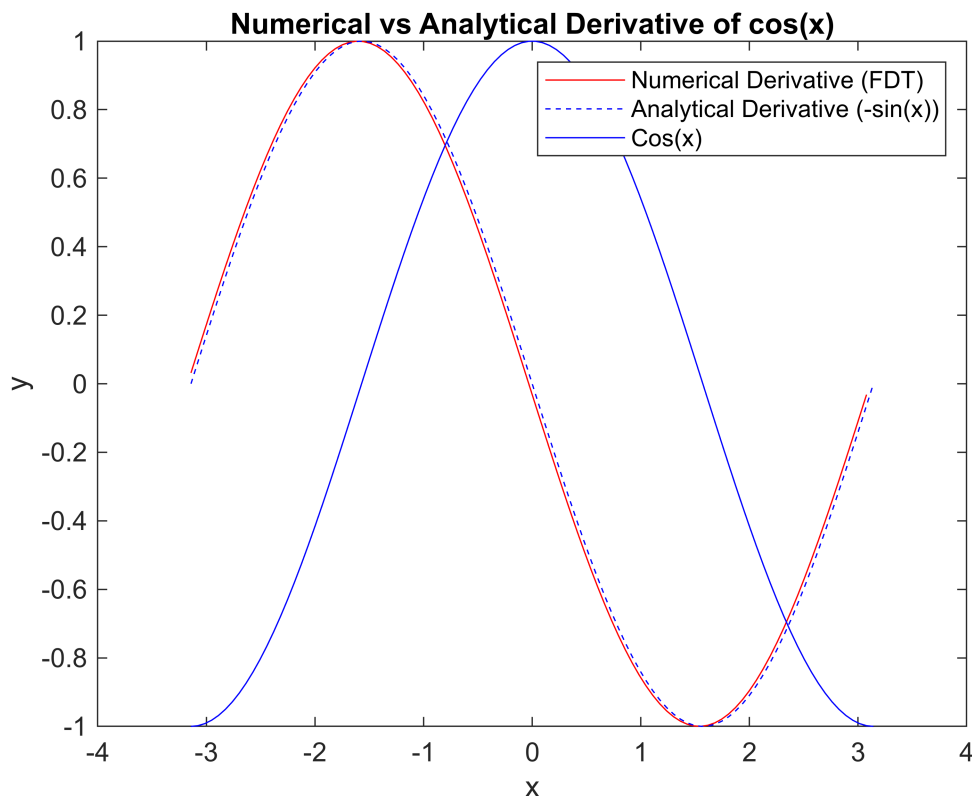
xlabel('x');

ylabel('y');

title('Numerical vs Analytical Derivative of cos(x)');

hold off;

```



Numerical Intergration (a) $\diamond = \int \diamond \sin(\diamond) \diamond x$ [3]

```

close all;
clear;
clc;

N=100;
x = linspace(0,pi/2,N);
number_of_trapiziums = 10;

a=0;
b= pi/2;
%%%%%%%%%%%%%
function [y] = my_function(x)
    y = x.*sin(x);

end
%%%%%%%%%%%%%
if a>b
    error("B must be greater than a")
end

integration = my_function(x);

delta_x = (b-a) / number_of_trapiziums;
sum_of_areas = 0;
hold on

for loop = 1:number_of_trapiziums

    % Draw a rectangle
    x_left = a+(loop-1)*delta_x;
    x_right = a+loop*delta_x;
    y_left = my_function(x_left);
    y_right = my_function(x_right);

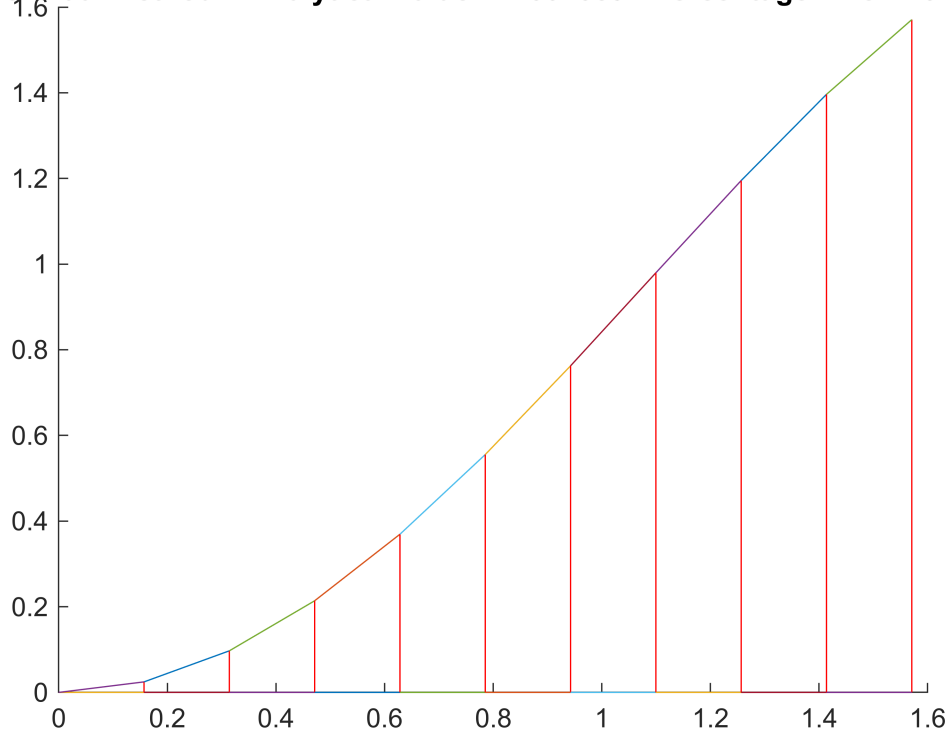
    %Modify y_left and y_right to mimic the various rules
    plot([x_left,x_left],[0 y_left],'r')
    plot([x_right, x_right],[0 y_right],'r')
    plot([x_left, x_right],[0, 0])
    plot([x_left, x_right],[y_left, y_right])

    areas_of_section = delta_x*(y_left + y_right)/2;
    sum_of_areas = sum_of_areas+areas_of_section;
end
title("Trapezoidal Method. - Analytical Value = 1.002059 - Percentage Error = 0.205871")

```

hold off

Trapezoidal Method. - Analytical Value = 1.002059 - Percentage Error = 0.20587'



```
f = @(x) x .* sin(x);  
area = integral(f, a, b);
```

Numerical Integration (b) compare [3]

```
Percentage_error = (abs(( 1 - (sum_of_areas / area) ) .* 100));  
%fprintf("Trapezoidal Method = %f",sum_of_areas);  
%fprintf("True Answer area under the curve x*sin(x) from 0 to pi/2 is: = %f ",area);  
%fprintf("The Percentage Error = %f ",Percentage_error);  
Table = table(["Trapezoidal method"; "Analytical Integration"; "Percentage Error"],  
[sum_of_areas; area; Percentage_error], 'VariableNames', {'Description', 'Value'});  
disp(Table);
```

Description	Value
"Trapezoidal method"	1.0021
"Analytical Integration"	1
"Percentage Error"	0.20587

```
% Part (a): Composite Trapezium Rule Function  
function I = composite_trapezium(f, a, b, n)
```

```

% f - function handle for the integrand
% a - lower limit of integration
% b - upper limit of integration
% n - number of intervals (sub-divisions)

% Calculate the step size
dx = (b - a) / n;

% Evaluate the function at the endpoints
fa = f(a);
fb = f(b);

% Sum the function values at each interior point
sum_interior = 0;
for i = 1:(n-1)
    x = a + i * dx;
    sum_interior = sum_interior + f(x);
end

% Apply the Composite Trapezium Rule formula
I = (dx / 2) * (fa + 2 * sum_interior + fb);

% Plotting the function and trapezoids
%
% %-
%
% x_values = linspace(a, b, n+1);
% y_values = f(x_values);
% figure;
% plot(x_values, y_values, 'b', 'LineWidth', 1.5); % Plot f(x)
% hold on;
%
% % Plot trapezoids for the area approximation
% for i = 1:n
%     x_trap = [x_values(i), x_values(i), x_values(i+1), x_values(i+1)];
%     y_trap = [0, y_values(i), y_values(i+1), 0];
%     fill(x_trap, y_trap, 'c', 'FaceAlpha', 0.3, 'EdgeColor', 'k'); %
Trapezoid fill
% end
%
% % Customize plot
% title('Composite Trapezium Rule Approximation for Integral of 1/x from 1 to
2');
% xlabel('x');
% ylabel('f(x) = 1/x');
% legend('f(x) = 1/x', 'Trapezoid Approximation');
% grid on;
% hold off;

%-

```

end

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

```
n=100; %Intervals
g=@(x) x.*sin(x);
traps = composite_trapezium(g, a, b, n);
P_Error = ((traps/area)-1)*100;
Table = table(["Trapezoidal method"; "Analytical Integration"; "Percentage Error"],
[traps; area; P_Error], 'VariableNames', {'Description', 'Value'});
disp(Table);
```

Description	Value
"Trapezoidal method"	1
"Analytical Integration"	1
"Percentage Error"	0.0020562

```
% Calculating and Comparing with Analytic Solution
f = @(x) 1 ./ x; % Define function 1/x
a = 1;           % Lower limit
b = 2;           % Upper limit
h = 5;           % Number of intervals
j = 50;
n = 100;
inter=[5, 50, 100];
% Calculate the numerical integral
I_numeric = composite_trapezium(f, a, b, h);
II_numeric = composite_trapezium(f, a, b, j);
III_numeric = composite_trapezium(f, a, b, n);
% Analytical solution
I_analytic = log(2); % Integral of 1/x from 1 to 2 is log(2)

% Calculate the percentage error
error_percent_5 = abs((I_numeric - I_analytic) / I_analytic) * 100;
error_percent_50 = abs((II_numeric - I_analytic) / I_analytic) * 100;
error_percent_100 = abs((III_numeric - I_analytic) / I_analytic) * 100;
fprintf('Numerical Integral: %.6f\n', I_numeric);
fprintf('Analytic Integral: %.6f\n', I_analytic);
fprintf('Percentage Error: %.6f%%\n', error_percent);
```

Comp Tramp (d) compare [1]

```
% Create the table
Description = ["Trapezoidal Method"; "Analytical Integration"; "Percentage Error"];
Intervals = [h; j; n];
Values = [I_numeric; II_numeric; III_numeric];
```

```
AnaLit=[III_numeric; III_numeric; III_numeric];
```

```
% Display the table
```

```
Table = table(Description, Intervals, Values, AnaLit, 'VariableNames',  
{'Description', 'Intervals', 'Value Composite', 'Analytical Values'});  
disp(Table);
```

Description	Intervals	Value Composite	Analytical Values
"Trapezoidal Method"	5	0.69563	0.69315
"Analytical Integration"	50	0.69317	0.69315
"Percentage Error"	100	0.69315	0.69315

Comp Trap (e) to tolerance [3]

```
target_accuracy = 1e-6;  
n_intervals = 10;  
I_previous = composite_trapezium(f, a, b, n_intervals);  
  
while abs(I_previous - I_analytic) > target_accuracy  
    n_intervals = n_intervals * 2; % Double the intervals for better accuracy  
    I_previous = composite_trapezium(f, a, b, n_intervals);  
end  
  
fprintf('Number of intervals needed for accuracy to 6 decimal places: %d\n',  
n_intervals);
```

Number of intervals needed for accuracy to 6 decimal places: 320

```
function I = simpsons_rule(f, a, b, n)  
    % Ensure n is even, as Simpson's Rule requires an even number of intervals  
    % if mod(n, 2) ~= 0  
    %     error('Simpson''s rule requires an even number of intervals.');
```

```
% end  
  
% Step size  
dx = (b - a) / n;  
  
% Evaluate the function at the endpoints  
fa = f(a);  
fb = f(b);  
  
% Sum the function values at each interior point, applying the 4 and 2 weights  
sum_odd = 0;  
sum_even = 0;  
for i = 1:(n-1)  
    x = a + i * dx;  
    if mod(i, 2) == 0  
        sum_even = sum_even + f(x);  
    else
```

```

        sum_odd = sum_odd + f(x);
    end
end

% Apply Simpson's Rule formula
I = (dx / 3) * (fa + 4 * sum_odd + 2 * sum_even + fb);
end

```

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

```

f = @(x) 1 ./ x; % Define function 1/x
a = 1;           % Lower limit
b = 2;           % Upper limit
n_values = [5, 50, 100]; % Different interval counts for comparison
I_analytic = log(2); % Analytical result

% Initialize arrays to store results
intervals = [];
numeric_integral = [];
analytic_integral = [];
error_percent = [];

% Loop over different n values to calculate the integral and store results
for n = n_values
    I_numeric = simpsons_rule(f, a, b, n);
    error = abs((I_numeric - I_analytic) / I_analytic) * 100;

    % Store results
    intervals = [intervals; n];
    numeric_integral = [numeric_integral; I_numeric];
    analytic_integral = [analytic_integral; I_analytic];
    error_percent = [error_percent; error];
end

```

Simpsons (c) compare [1]

```

% Create a table with the results
result_table = table(n_values', numeric_integral, analytic_integral, error_percent,
    ...
        'VariableNames', {'Intervals', 'Numerical_Integral',
        'Analytic_Integral', 'Error_Percent'});

% Display the result table
disp(result_table);

```

Intervals	Numerical_Integral	Analytic_Integral	Error_Percent
5	0.6582	0.69315	5.0417
50	0.69315	0.69315	7.2063e-07
100	0.69315	0.69315	4.5073e-08

Simpssons (d) to tolerance [3]

```
% Determine the number of intervals needed for accuracy to 6 decimal places
target_accuracy = 1e-6;
n_intervals = 10; % Starting with 10 intervals
I_previous = simpsons_rule(f, a, b, n_intervals);

while abs(I_previous - I_analytic) > target_accuracy
    n_intervals = n_intervals + 2; % Increase by 2 to keep it even
    I_previous = simpsons_rule(f, a, b, n_intervals);
end

fprintf('\nNumber of intervals needed for accuracy to 6 decimal places: %d\n',
n_intervals);
```

Number of intervals needed for accuracy to 6 decimal places: 14