



prifysgol cymru trinity saint devid
University of Wales Trinity St David

Student name: Anandu Unnikrishanan Nair
Student ID number : 2411630
Programme : MSc - Automotive Engineering
Module : Engine Performance and Emissions
Module code: ACEA7005

TASK 1

Lab. 1

Using a script file (I)

In this first task we are simply going to plot a function using a script file.

(a) Open a MATLAB script file, save it to your personnel directory as lab1_sin.m. This script will be used to call intrinsic and user-defined functions in ML.

(b) Type:

clear

format short g - -

clears all variables from the workspace

formats the outputted numerical data

(c) MATLAB has many 'in-built' functions that can be easily used; this task we will plot the sine function over the interval $[-\pi \pi]$.

(d) Define the vector/array/table x as:

N = 100;

x = linspace(-pi,pi,N)'; (see help linspace) or as

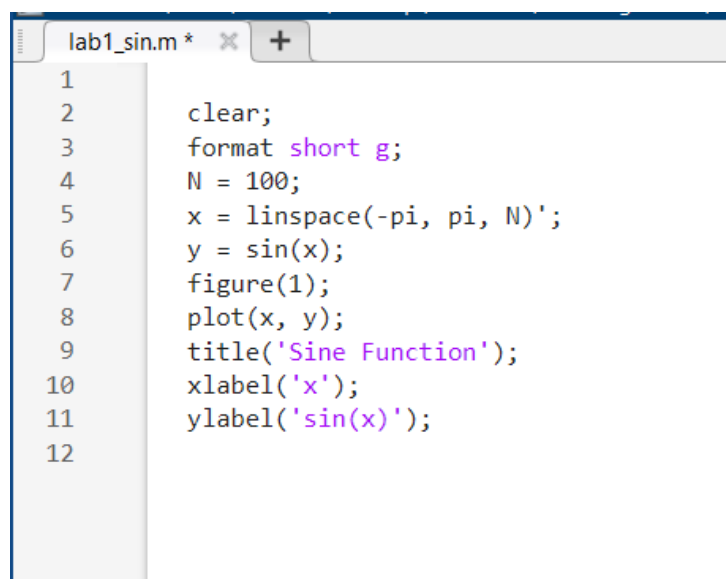
x = (-pi:2*pi/(N-1):pi)';

NB: the semi-colon suppresses output to the command window and the apostrophe transposes the default column vector to a row vector.

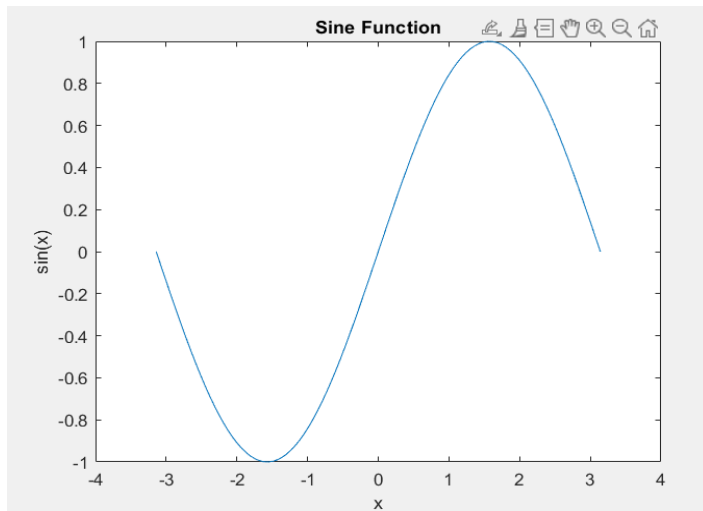
(e) Use the following code to plot sin(x) as a function of x:

y = sin(x); figure(1),

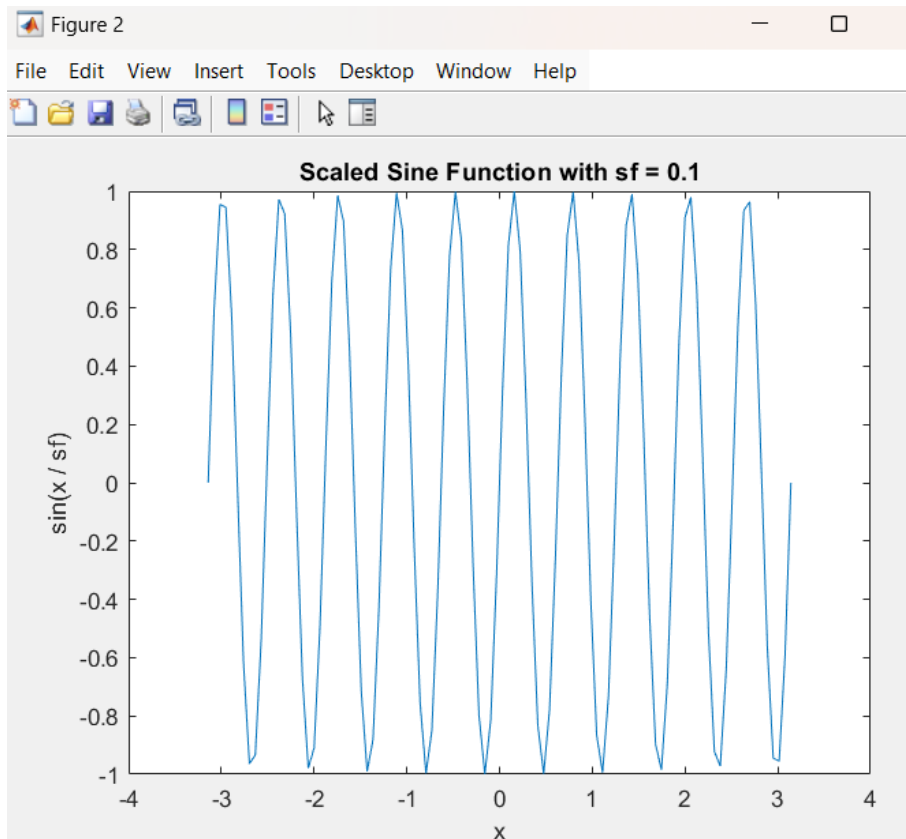
plot(x,y)



```
lab1_sin.m *  +
1
2     clear;
3     format short g;
4     N = 100;
5     x = linspace(-pi, pi, N)';
6     y = sin(x);
7     figure(1);
8     plot(x, y);
9     title('Sine Function');
10    xlabel('x');
11    ylabel('sin(x)');
12
```

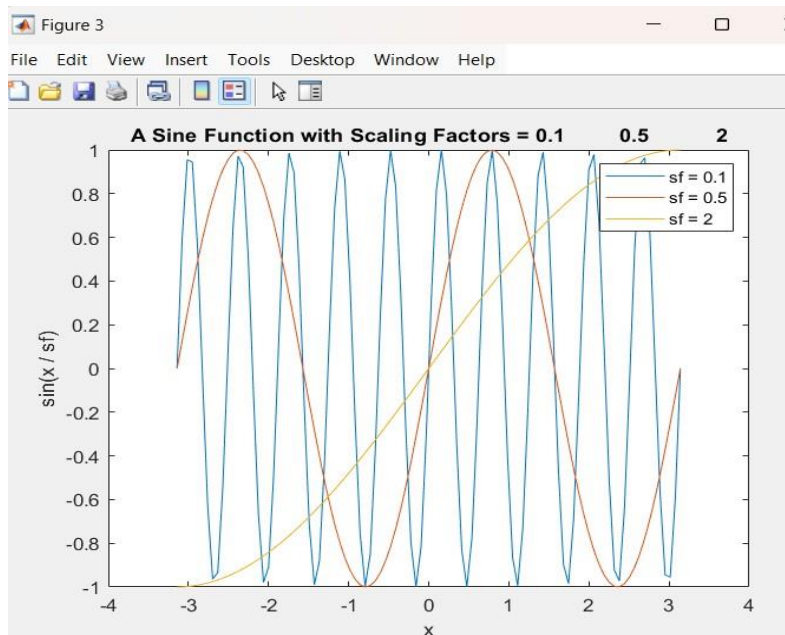


```
lab1_sin.m x sin_scale.m x +
1
2 clear
3 format short g
4
5 N = 100;
6 x = linspace(-pi, pi, N)';
7
8 sf = 0.1;
9 y = sin_scale(x, sf);
10
11 figure(2);
12 plot(x, y);
13 title('Scaled Sine Function with sf = 0.1');
14 xlabel('x');
15 ylabel('sin(x / sf)');
16
```



F

```
lab1_sin.m x sin_scale.m +
1 clear
2 format short g
3
4 N = 100;
5 x = linspace(-pi, pi, N)';
6
7 sf = [0.1, 0.5, 2];
8 N_sf = numel(sf);
9
10 sin_array = zeros(N, N_sf);
11
12 for ndx = 1:N_sf
13     sin_array(:, ndx) = sin_scale(x, sf(ndx));
14 end
15
16 figure(3);
17 plot(x, sin_array);
18 title(['A Sine Function with Scaling Factors = ', num2str(sf)]);
19 xlabel('x');
20 ylabel('sin(x / sf)');
21 legend(arrayfun(@(s) ['sf = ', num2str(s)], sf, 'UniformOutput', false));
22
```



Numerical Differentiation

```

+1 sin_scale.m lab1_cos_derivative.m forward_diff.m +
1 N = 100;
2 x = linspace(-pi, pi, N)';
3 y = cos(x);
4 dy_dx = forward_diff(x, y);
5 figure;
6 plot(x, dy_dx, 'b', 'DisplayName', 'Numerical Derivative');
7 hold on;
8 plot(x, -sin(x), 'r--', 'DisplayName', 'Analytical Derivative');
9 hold off;
10
11 title('Numerical Derivative of cos(x) vs. Analytical Derivative');
12 xlabel('x');
13 ylabel('Derivative');
14 legend show;

```

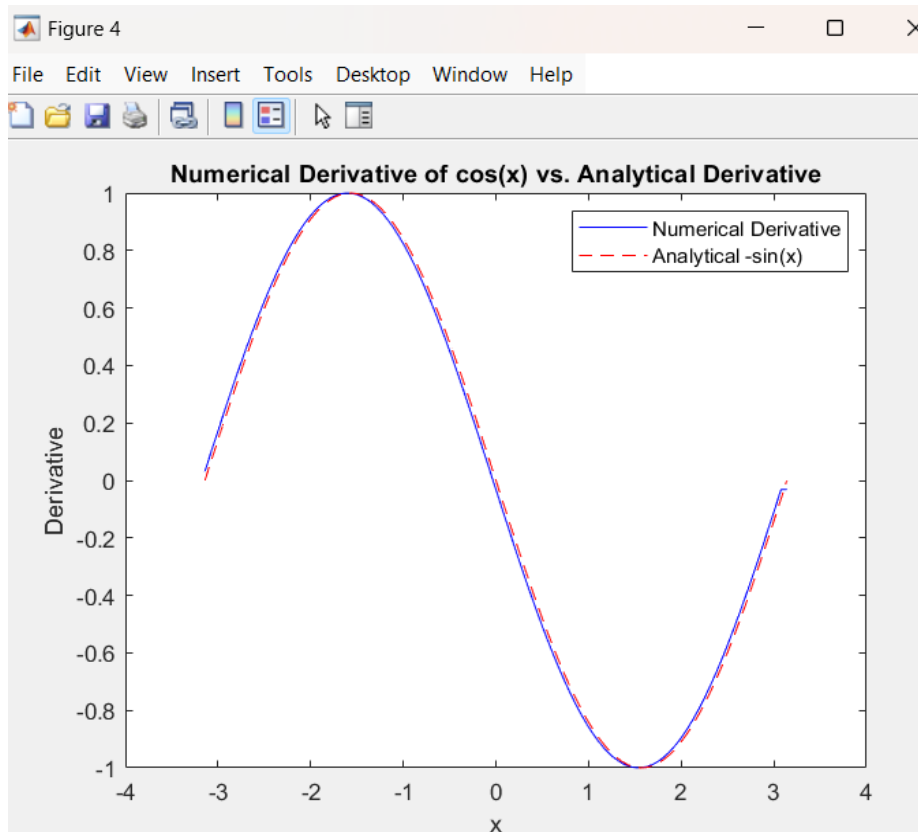
Command Window

```

>> lab1_cos_derivative
>> lab1_cos_derivative
>> lab1_cos_derivative
fr \>

```

```
+1 sin_scale.m x lab1_cos_derivative.m x forward_diff.m x +
1 function dy_dx = forward_diff(x, y)
2
3     N = length(x);
4     h = x(2) - x(1);
5     dy_dx = zeros(N, 1);
6
7     for i = 1:N-1
8         dy_dx(i) = (y(i+1) - y(i)) / h;
9     end
10
11     dy_dx(N) = dy_dx(N-1);
12 end
13
```



Numerical Integration

```

lab1_sin.m x sin_scale.m x lab1_cos_derivative.m x forward_diff.m x trap1.m x f.
1 function int = trap1(f, a, b)
2     if a > b
3         error('b must be greater than a');
4     end
5     h = (b - a);
6     int = h * (feval(f, a) + feval(f, b)) / 2;
7 end

```

```

lab1_sin.m x sin_scale.m x lab1_cos_derivative.m x forward
1 function y = f(x)
2     y = x .* sin(x);
3 end

```

```

lab1_sin.m x sin_scale.m x lab1_cos_derivative.m x forward_diff.m x trap1.m x f.m x main_script.m x +
1 % Main script to evaluate the integral
2 a = 0;
3 b = pi/2;
4 I = trap1('f', a, b);
5 disp(['Numerical integral using trapezoidal method: ', num2str(I)]);
6
7 analytic_value = integral(@(x) x .* sin(x), a, b);
8 disp(['Analytic integral: ', num2str(analytic_value)]);
9
10 percentage_error = abs((I - analytic_value) / analytic_value) * 100;
11 disp(['Percentage error: ', num2str(percentage_error), '%']);

```

Name	Value
a	0
analytic_value	1.0000
analytical_int...	3.1416
b	1.5708
dy_dx	100x1 double
f	@(x)x.*sin(x)
I	1.2337
N	100
N_sf	3
ndx	3
numerical_int...	6.0434e-16
percentage_e...	23.3701
sf	[0.1000,0.5000,2
sin_array	100x3 double
x	100x1 double
y	100x1 double

Command Window

```

>> main_script
Numerical integral using trapezoidal method: 1.2337
Analytic integral: 1
Percentage error: 23.3701%
fx >>

```

2. Composite Trapezium Rule

A)

```
lab1_sin.m  sin_scale.m  lab1_cos_derivative.m  forward_diff.m
1 function int = composite_trap(f, a, b, n)
2
3     % Step size
4     h = (b - a) / n;
5
6     % Initialize sum with endpoints
7     int = f(a) + f(b);
8
9     % Sum up the values at interior points
10    for i = 1:n-1
11        x = a + i * h;
12        int = int + 2 * f(x);
13    end
14
15    % Final result after multiplying by (h / 2)
16    int = (h / 2) * int;
17 end
18
```

b)

```
+2 lab1_cos_derivative.m  forward_diff.m  trap1.m  main_script.m  f.m  composite_trap.m  calculate_integra
1 % Main script to evaluate the integral
2 a = 0; % lower limit
3 b = pi/2; % upper limit
4 n = 100; % number of intervals
5
6 % Call the composite trapezoidal function
7 I_numeric = composite_trap('f', a, b, n);
8 disp(['Numerical integral using composite trapezoidal method: ', num2str(I_numeric)]);
9
10 % Calculate the analytic value using MATLAB's integral function
11 analytic_value = integral(@(x) x .* sin(x), a, b);
12 disp(['Analytic integral: ', num2str(analytic_value)]);
13
14 % Calculate the percentage error
15 percentage_error = abs((I_numeric - analytic_value) / analytic_value) * 100;
16 disp(['Percentage error: ', num2str(percentage_error), '%']);

Command Window
Percentage error: 0.0020562%
>> calculate_integral
Numerical integral using composite trapezoidal method: 1
Analytic integral: 1
Percentage error: 0.0020562%
```

c)

```

+7 composite_trap.m x calculate_integral.m x f_reciprocal.m * x +
1 function y = f_reciprocal(x)
2     y = 1 ./ x;
3 end

```

```

+5 main_script.m x f.m x composite_trap.m x calculate_integral.m x f_reciprocal.m x evaluate_reciprocal_integral.m * x +
2 % Main script to evaluate the integral for f(x) = 1/x
3 a = 1; % lower limit
4 b = 2; % upper limit
5 intervals = [5, 50, 100]; % different numbers of intervals
6 results = zeros(length(intervals), 2); % To store results and errors
7
8 for i = 1:length(intervals)
9     n = intervals(i);
10    I_numeric = composite_trap('f_reciprocal', a, b, n);
11    results(i, 1) = I_numeric; % Store numerical result
12
13    % Calculate the analytic value
14    analytic_value = log(2); % ln(2)
15    results(i, 2) = analytic_value; % Store analytic result
16
17    % Calculate percentage error
18    percentage_error = abs((I_numeric - analytic_value) / analytic_value) * 100;
19    disp(['Numerical integral for ', num2str(n), ' intervals: ', num2str(I_numeric)]);
20    disp(['Analytic integral: ', num2str(analytic_value)]);
21    disp(['Percentage error: ', num2str(percentage_error), '%']);
22 end

```

Command Window

```

>> evaluate_reciprocal_integral
Numerical integral for 5 intervals: 0.69563
Analytic integral: 0.69315
Percentage error: 0.35891%
Numerical integral for 50 intervals: 0.69317
Analytic integral: 0.69315
Percentage error: 0.0036066%
Numerical integral for 100 intervals: 0.69315
Analytic integral: 0.69315
Percentage error: 0.00090167%
fx >>

```

d)

```

21 end
22 % Create a table for results
23 T = table(intervals', results(:, 1), results(:, 2), ...
24     'VariableNames', {'Intervals', 'Numerical Integral', 'Analytic Integral'});
25 disp(T);

```

Command Window

```

Numerical integral for 100 intervals: 0.69315
Analytic integral: 0.69315
Percentage error: 0.00090167%

```

Intervals	Numerical Integral	Analytic Integral
5	0.69563	0.69315
50	0.69317	0.69315
100	0.69315	0.69315

fx >>

e)

```

26
27 % Determine the number of sub-intervals needed for 6 decimal places of accuracy
28 tolerance = 1e-6;
29 n = 1;
30 error = Inf;
31
32 while error > tolerance
33     I_numeric = composite_trap('f_reciprocal', a, b, n);
34     analytic_value = log(2);
35     error = abs((I_numeric - analytic_value) / analytic_value);
36     n = n + 1;
37 end
38
39 disp(['Number of sub-intervals needed for 6 decimal places of accuracy: ', num2str(n-1)]);

```

Command Window

Analytic integral: 0.69315
 Percentage error: 0.00090167%

Intervals	Numerical Integral	Analytic Integral
5	0.69563	0.69315
50	0.69317	0.69315
100	0.69315	0.69315

Number of sub-intervals needed for 6 decimal places of accuracy: 301
 fx >>

3. Simpson's Rule

a)

```

+6 f.m composite_trap.m calculate_integral.m f_reciprocal.m evaluate_reciprocal_integral.m simpson
1 function I = simpson_rule(f, a, b, n)
2
3
4 % Input validation
5 if mod(n, 2) ~= 0
6     error('n must be an even integer for Simpson's Rule.');
```

b)

```

+8 calculate_integral.m x f_reciprocal.m x evaluate_reciprocal_integral.m x simpson_rule.m x evaluate_simpson_integral.m x
7
8   for i = 1:length(intervals)
9       n = intervals(i);
10      % Ensure n is even for Simpson's Rule
11      if mod(n, 2) ~= 0
12          n = n + 1;
13      end
14      I_numeric = simpson_rule('f_reciprocal', a, b, n);
15      results(i, 1) = I_numeric;
16
17      % Calculate the analytic value
18      analytic_value = log(2); % ln(2)
19      results(i, 2) = analytic_value;
20
21      % Calculate percentage error
22      percentage_error = abs((I_numeric - analytic_value) / analytic_value) * 100;
23      disp(['Numerical integral for ', num2str(n), ' intervals: ', num2str(I_numeric)]);
24      disp(['Analytic integral: ', num2str(analytic_value)]);
25      disp(['Percentage error: ', num2str(percentage_error), '%']);
26   end

```

Command Window

```

>> evaluate_simpson_integral
Numerical integral for 6 intervals: 0.69317
Analytic integral: 0.69315
Percentage error: 0.0032623%
Numerical integral for 50 intervals: 0.69315
Analytic integral: 0.69315
Percentage error: 7.2063e-07%
Numerical integral for 100 intervals: 0.69315
Analytic integral: 0.69315
Percentage error: 4.5073e-08%
fx >>

```

c)

```

+8 calculate_integral.m x f_reciprocal.m x evaluate_reciprocal_integral.m x simpson_rule.m x evaluate_simpson_integral.m x +
20
21      % Calculate percentage error
22      percentage_error = abs((I_numeric - analytic_value) / analytic_value) * 100;
23      disp(['Numerical integral for ', num2str(n), ' intervals: ', num2str(I_numeric)]);
24      disp(['Analytic integral: ', num2str(analytic_value)]);
25      disp(['Percentage error: ', num2str(percentage_error), '%']);
26   end
27
28   % Create a table for results
29   T = table(intervals', results(:, 1), results(:, 2), ...
30       'VariableNames', {'Intervals', 'Numerical Integral', 'Analytic Integral'});
31   disp(T);

```

Command Window

```

Numerical integral for 100 intervals: 0.69315
Analytic integral: 0.69315
Percentage error: 4.5073e-08%

```

Intervals	Numerical Integral	Analytic Integral
5	0.69317	0.69315
50	0.69315	0.69315
100	0.69315	0.69315

d)

```

+8 calculate_integral.m x f_reciprocal.m x evaluate_reciprocal_integral.m x simpson_rule.m x evaluate_simpson_integral.m * x
32
33 % Determine the number of sub-intervals needed for 6 decimal places of accuracy
34 tolerance = 1e-6;
35 n = 2;
36 error = Inf;
37
38 % Initialize the analytic value once before the loop
39 analytic_value = log(2);
40
41 % Loop until the error is within the specified tolerance
42 while error > tolerance
43     I_numeric = simpson_rule('f_reciprocal', a, b, n);
44     error = abs((I_numeric - analytic_value) / analytic_value);
45     n = n + 2;
46 end
47
48 % Display the number of intervals needed to achieve the desired accuracy
49 disp(['Number of sub-intervals needed for 6 decimal places of accuracy: ', num2str(n-2)]);

```

Command Window

Analytic integral: 0.69315
 Percentage error: 4.5073e-08%

Intervals	Numerical Integral	Analytic Integral
5	0.69317	0.69315
50	0.69315	0.69315
100	0.69315	0.69315

Number of sub-intervals needed for 6 decimal places of accuracy: 16