

Task 1

please convert this to pdf for submission

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

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

```
%% I
% Plotting sine function
clear; format short g;

% Define variables
N = 100;
x = linspace(-pi, pi, N)';

% (a) Plot sin(x)
y = sin(x);
figure(1);
plot(x, y);
title('Plot of sin(x)');
xlabel('x');
ylabel('sin(x)');
grid on;

% (b) Define and use scaled sine function
sf = 0.1;
y_scaled = sin_scale(x, sf);
figure(2);
plot(x, y_scaled);
title(['Plot of sin(x/' num2str(sf) ')']);
xlabel('x');
ylabel(['sin(x/' num2str(sf) ')']);
grid on;

% (c) Loop over scale factors
sf_values = [0.1, 0.5, 2];
N_sf = numel(sf_values);
```

```

sin_array = zeros(N, N_sf);

for idx = 1:N_sf
    sin_array(:, idx) = sin_scale(x, sf_values(idx));
end

figure(3);
plot(x, sin_array);
legend(arrayfun(@(sf) ['sf = ' num2str(sf)], sf_values, 'UniformOutput', false));
title('Sine Functions with Different Scale Factors');
xlabel('x');
ylabel('sin(x/sf)');
grid on;

% Scaled sine function definition
function y = sin_scale(x, sf)
    y = sin(x ./ sf);
end

```

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

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

```

%% II: Numerical Differentiation of cos(x)
clear; format short g;

% Define x and y
N = 100;
x = linspace(-pi, pi, N)';
y = cos(x);

% Forward Difference Technique
h = x(2) - x(1);
dy_numerical = zeros(size(y));
dy_numerical(1:end-1) = (y(2:end) - y(1:end-1)) / h;
dy_numerical(end) = dy_numerical(end-1); % Boundary condition

% Analytical derivative
dy_analytical = -sin(x);

% Plot Numerical vs Analytical
figure(1);
plot(x, dy_analytical, 'r', x, dy_numerical, 'bo');
legend('Analytical (-sin(x))', 'Numerical (FDT)');
title('Numerical vs Analytical Derivative of cos(x)');
xlabel('x');
ylabel('Derivative');
grid on;

```

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

```
%% Numerical Differentiation
% (a) Define x and y = cos(x)
clear; format short g;

N = 100;
x = linspace(-pi, pi, N)';
y = cos(x);

figure(1);
plot(x, y);
title('Plot of cos(x)');
xlabel('x');
ylabel('cos(x)');
grid on;

% (b) Numerical Differentiation using Forward Difference Technique
h = x(2) - x(1); % Uniform spacing
dy_numerical = zeros(size(y));
dy_numerical(1:end-1) = (y(2:end) - y(1:end-1)) / h;
dy_numerical(end) = dy_numerical(end-1); % Boundary condition

figure(2);
plot(x, dy_numerical);
title('Numerical Derivative of cos(x)');
xlabel('x');
ylabel('dy/dx');
grid on;

% (c) Comparison with Analytical Derivative (-sin(x))
dy_analytical = -sin(x);

figure(3);
plot(x, dy_analytical, 'r', x, dy_numerical, 'bo');
legend('Analytical (-sin(x))', 'Numerical (FDT)');
title('Comparison: Numerical vs Analytical Derivative of cos(x)');
xlabel('x');
ylabel('Derivative');
grid on;

% (d) Plot Numerical vs Analytical Derivative
figure(1);
plot(x, dy_analytical, 'r', x, dy_numerical, 'bo');
legend('Analytical (-sin(x))', 'Numerical (FDT)');
title('Numerical vs Analytical Derivative of cos(x)');
xlabel('x');
```

```
ylabel('Derivative');
grid on;

% (e) Comparison and observations
disp('Numerical Differentiation Complete. Observe plot for comparison.');
```

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

```
%% Numerical Integration
clear; format short g;

% (a) Trapezoidal Rule Function
function result = trapezoidal_rule(f, a, b)
    h = b - a;
    result = (h / 2) * (f(a) + f(b));
end

% Define function and limits
f = @(x) sin(x); % Function to integrate
a = 0; b = pi; % Integration limits

% Compute numerical and analytical integrals
numerical_integral = trapezoidal_rule(f, a, b);
analytical_integral = 2; % Integral of sin(x) from 0 to pi is 2

% Compute percentage error
percentage_error = abs((numerical_integral - analytical_integral) /
analytical_integral) * 100;

% Display results
disp('Trapezoidal Rule Results:');
disp(['Numerical Integral: ', num2str(numerical_integral)]);
disp(['Analytical Integral: ', num2str(analytical_integral)]);
disp(['Percentage Error (%): ', num2str(percentage_error)]);
```

Numerical Intergration (b) compare [3]

```
% (b) Composite Trapezoidal Rule Function
function result = composite_trapezoidal(f, a, b, n)
    h = (b - a) / n;
    x = linspace(a, b, n + 1);
    y = f(x);
    result = (h / 2) * (y(1) + 2 * sum(y(2:end-1)) + y(end));
end

% Composite Trapezoidal Rule Test
n_values = [5, 50, 100];
disp('Composite Trapezoidal Rule Results:');
```

```

for n = n_values
    integral = composite_trapezoidal(f, a, b, n);
    disp(['n = ' num2str(n) ', Integral = ' num2str(integral)]);
end

```

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

```

%% 3: Composite Trapezium Rule
clear; format short g;

% (a) Composite Trapezoidal Rule Function
function result = Composite_trapezoidal (f, a, b, n)
    h = (b - a) / n;
    x = linspace(a, b, n + 1);
    y = f(x);
    result = (h / 2) * (y(1) + 2 * sum(y(2:end-1)) + y(end));
end

% Define function and limits
f = @(x) 1 ./ x; % Function to integrate
a = 1; b = 2; % Integration limits

% (b) Composite Trapezoidal Rule Test
n_values = [5, 50, 100];
disp('Composite Trapezoidal Rule Results:');
for n = n_values
    integral = composite_trapezoidal(f, a, b, n);
    analytical_integral = log(2); % Exact value of integral of 1/x from 1 to 2
    percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
    disp(['n = ' num2str(n) ', Integral = ' num2str(integral) ', Error (%) = '
num2str(percentage_error)]);
end

% (c) Tabulate Results
results = zeros(length(n_values), 2);
for i = 1:length(n_values)
    n = n_values(i);
    integral = composite_trapezoidal(f, a, b, n);
    analytical_integral = log(2);
    percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
    results(i, :) = [integral, percentage_error];
end
disp('Tabulated Results:');
disp(array2table(results, 'VariableNames', {'Integral', 'Percentage_Error'},
'RowNames', string(n_values)));

```

Comp Trap (d) compare [1]

```

% (d) Determine Sub-intervals for Accuracy
target_error = 1e-6; % Target for six-decimal accuracy
n = 2; % Start with a small even number
while true
    integral = composite_trapezoidal(f, a, b, n);
    percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
    if percentage_error < target_error
        break;
    end
    n = n + 1;
end

disp(['Sub-intervals needed for six-decimal accuracy: ' num2str(n)]);

```

Comp Trap (e) to tolerance [3]

```

disp(['Sub-intervals needed for six-decimal accuracy: ' num2str(n)]);

```

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

```

%% Simpson's Rule
clear; format short g;

% a) Simpson's Rule Function
function result = simpsons_rule(f, a, b, n)
    if mod(n, 2) ~= 0
        error('Number of intervals (n) must be even for Simpson's Rule.');
```

```

    end
    h = (b - a) / n;
    x = linspace(a, b, n + 1);
    y = f(x);
    result = (h / 3) * (y(1) + 4 * sum(y(2:2:end-1)) + 2 * sum(y(3:2:end-2)) +
y(end));
end

% Define function and limits
f = @(x) 1 ./ x; % Function to integrate
a = 1; b = 2; % Integration limits

% b) Test Simpson's Rule for Different Intervals
n_values = [5, 50, 100];
disp('Simpson's Rule Results:');
for n = n_values
    if mod(n, 2) ~= 0, n = n + 1; end % Ensure even intervals
    integral = simpsons_rule(f, a, b, n);
    analytical_integral = log(2); % Exact value of integral of 1/x from 1 to 2

```

```

percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
disp(['n = ' num2str(n) ', Integral = ' num2str(integral) ', Error (%) = '
num2str(percentage_error)]);
end

```

Simpsons (c) compare [1]

```

% c) Tabulate Results
results = zeros(length(n_values), 2);
for i = 1:length(n_values)
    n = n_values(i);
    if mod(n, 2) ~= 0, n = n + 1; end % Ensure even intervals
    integral = simpsons_rule(f, a, b, n);
    analytical_integral = log(2);
    percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
    results(i, :) = [integral, percentage_error];
end
disp('Tabulated Results:');
disp(array2table(results, 'VariableNames', {'Integral', 'Percentage_Error'},
'RowNames', string(n_values)));

```

Simpsons (d) to tolerance [3]

```

% d) Determine Sub-intervals for Accuracy
target_error = 1e-6; % Target for six-decimal accuracy
n = 2; % Start with a small even number
while true
    integral = simpsons_rule(f, a, b, n);
    percentage_error = abs((integral - analytical_integral) / analytical_integral)
* 100;
    if percentage_error < target_error
        break;
    end
    n = n + 2; % Increment by 2 to keep n even
end

disp(['Sub-intervals needed for six-decimal accuracy: ' num2str(n)]);

```