

UWTSD

MSc Motorsport Engineering

Task 1

ACHA7002

2500151

Date

Contents

1	Plotting and differentiation.....	1
2	Trapezoidal Method	2
3	Composite	2
4	Simpsons	2
5	Integration results tables	4
5.1	Composite convergence iterations	4
5.2	Simpsons Convergence iterations	4
6	Appendices.....	4
6.1	Your code script and functions	4

1 Plotting and differentiation

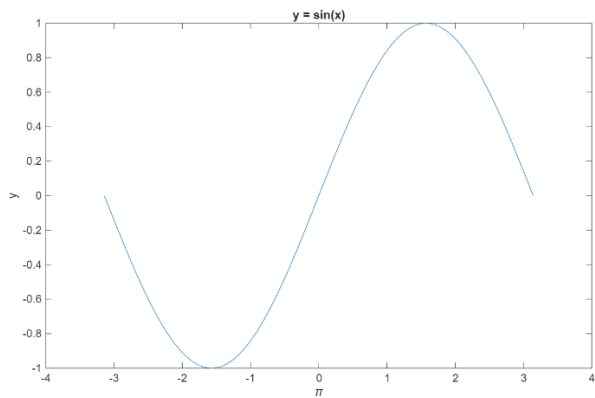


Figure 1 $y = \sin(x)$ plot

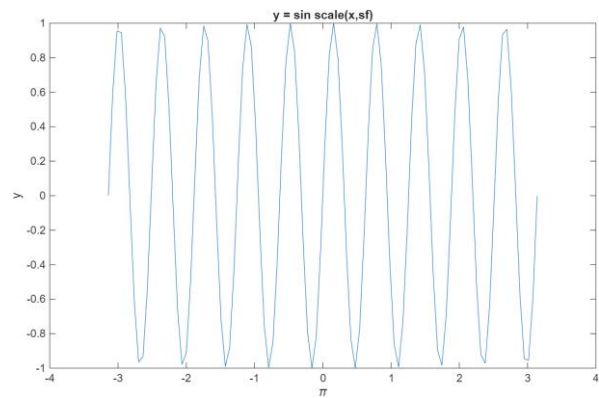


Figure 2 $y = \sin_scale(x, sf)$ plot (0.1 steps)

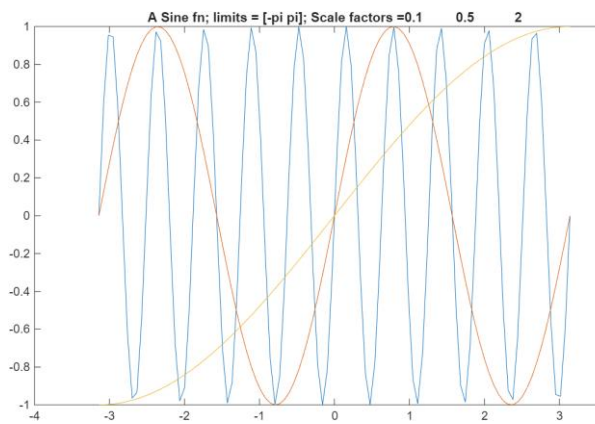


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

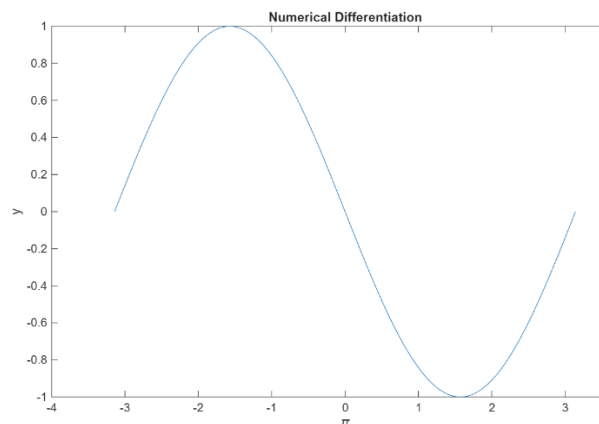


Figure 4 diff $\cos(x)$ to $-\sin(x)$

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

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
1.233700550136170	1.000020979425220	23.367466835071934

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'	0.693147	0.695635	0.693172	0.693153	0.693148
'n iterations'	Na	5	50	100	250
'percentage difference'	Na	0.358905	0.003607	0.000902	0.000144

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

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

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'	0.693147	0.584127	0.693147	0.693147	0.693147
'n iterations'	NA	5	50	100	14
'percentage difference'	NA	15.728290	0.000001	0.000000	0.000116

5 Integration results tables

5.1 Composite convergence iterations

N iterations	Result	Percentage difference
5	0.695635	0.559279
50	0.693172	0.003607
100	0.693153	0.000902
150	0.693150	0.000401
200	0.693149	0.000225
250	0.693148	0.000144

5.2 Simpsons Convergence iterations

N iterations	Results	Percentage difference
2	0.694444	0.187156
4	0.693254	0.015406
6	0.693170	0.003262
8	0.693155	0.001060
10	0.693150	0.000440
12	0.693149	0.000214
14	0.693148	0.000116

6 Appendices

6.1 Your code script and functions

```

%% using a script file (I)

clear, format short g

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

y = sin (x);
figure (1),
plot (x,y)
xlabel('\pi')
ylabel('y')

```

```

title('y = sin(x)')

%% defining a function (II)

sf = 0.1;
y = sin_scale(x, sf);
figure (2),
plot (x,y)
xlabel('\pi')
ylabel('y')
title('y = sin scale(x,sf)')

%%

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 Sine fn; limits = [-pi pi]; Scale factors = ', num2str(sf)])

%% Numerical differentiation
N = 100;
x = linspace(-pi,pi,N)';

y=cos(x);
% h=(2*pi)./N;

df_dt = fdt(x,y);

df_dt_true=-sin(x);

hold on
figure(4),
plot (x,df_dt)
plot (x,df_dt_true)
xlabel('\pi')
ylabel('y')
title('Numerical Differentiation')

hold off

%% Numerical Integration 1. Trapezoidal Method
a = 0;
b = (pi)/2;
N = 100;
x = linspace(a, b, N)';
y = my_funct(x);
fun = 'my_funct';
int = trap1 (fun ,a, b );
int_true = trapz(x, y);
p_error = (abs(int - int_true)/(int_true))*100;

fprintf ('Trapezium rule estimated area = %f \r', int)
fprintf ('real integral area = %f \r', int_true)

```

```

fprintf ('Percentage error =%f \r',p_error)

%% Numerical Integration 2. Composite Trapezium Rule

int_comp = comp_trapz(fun, a, b, N);

% Calculate the percentage error for the composite trapezium rule
p_error_comp = (abs(int_comp - int_true) / int_true) * 100;

fprintf('Percentage error for Composite Trapezium rule = %f\n \r', p_error_comp);
fprintf('Composite Trapezium rule estimated area = %f\n \r', int_comp);

% this is totally useless, I find it funny because 'fra' means 'bro' in Italian
fra = 'fract';
true_integral = integral(@fra, 1, 2);

tolerance = 1e-6;
counter = 1;
max_counter = 300;
int_comp_i = comp_trapz (fra, 1, 2, counter);

% Number of sub-intervals needed to achieve an accuracy to 6 decimal places
while (abs(true_integral-int_comp_i)>tolerance && counter<max_counter)
    int_comp_i = comp_trapz(fra, 1, 2, counter);
    p_error_i = (abs(int_comp_i - true_integral) / true_integral) * 100;

    %output to screen
    fprintf('ITERATIONS %i \r',counter)
    fprintf('Composite trapezium rule with %i intervals =%10.6f \r', counter,
int_comp_i );
    fprintf('Percentage error for Composite Trapezium at 1e-6 tol = %f \r \r',
p_error_i);

    counter = counter + 1;

end

%% Numerical Integration 3. Simpson's Rule
count = 1;
errore = inf;
max_count = 300; % Set a maximum count to prevent infinite loops

int_simp_i = Simpson (fra, 1, 2, counter);

% Number of sub-intervals needed to achieve an accuracy to 6 decimal places
while (errore > tolerance && count<max_count)
    int_simp_i = Simpson (fra, 1, 2, count);
    errore = abs(true_integral-int_simp_i);

    p_error_is = (abs(int_simp_i - true_integral) / true_integral) * 100;

    %output to screen
    fprintf('ITERATIONS %i \r',count)
    fprintf('Simpsons rule with %i intervals =%10.6f \r', count, int_simp_i );
    fprintf('Percentage error with %i intervals for Simpsons rule at 1e-6 tol = %f
\r \r',count, p_error_is);

    count = count + 1;

end

```

```
function y = sin_scale(x,sf)
%input: scalar or vector x, sf scale factor
%output: scalar or vector containing corresponding sine values
%y = sin(x)          ML intrinsic sine function

y = sin(x./sf);

end
```

```
function dy_dx_approx = fdt(x, y)
    % h is the constant step-size
    h = x(2) - x(1);

    % Initialize the approximate derivative vector
    dy_dx_approx = zeros(size(x));

    for i = 1:(length(x) - 1)
        dy_dx_approx(i) = (y(i+1) - y(i)) / h;
        dy_dx_approx(end) = (y(end) - y(end-1)) / h; % Handle the last point
    end

end
```

```
function [y]=my_funcnt(x)

y = x.*sin(x);

end
```

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

%input -f is the integrand input as a string 'f'
%-a and b are upper and lower limits of integration
%output -s is the trapezoidal rule sum, equation 3

if a>b
    error ('b must be greater than a')
end
h=(b-a);
int = h*(feval(fun,a)+feval(fun,b))/2;

end
```

```
function [ff] = comp_trapz(f, a, b, n)

y_values = feval(f, linspace(a, b, n+1)); % Evaluate function at n points between
a and b
ff = (1/2) * ((b - a) / n) * (y_values(1) + 2 * sum(y_values(2:end-1)) +
y_values(end));

end
```

```
function [f]= fract(x)

f=1./x;

end
```

```
function [s] = Simpson (fra, a, b, N)

y_values = feval(fra, linspace(a, b, N+1));
n = N; % Set the number of intervals
h = (b - a) / n; % Calculate the width of each interval
s = h/3 * (y_values(1) + 4 * sum(y_values(2:2:end-1)) + 2 * sum(y_values(3:2:end-2)) + y_values(end));

end
```