

UWTSD

Alternative Fuels and Combustion Modelling

Task 1

ACHA7002

2515979

07/12/25

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

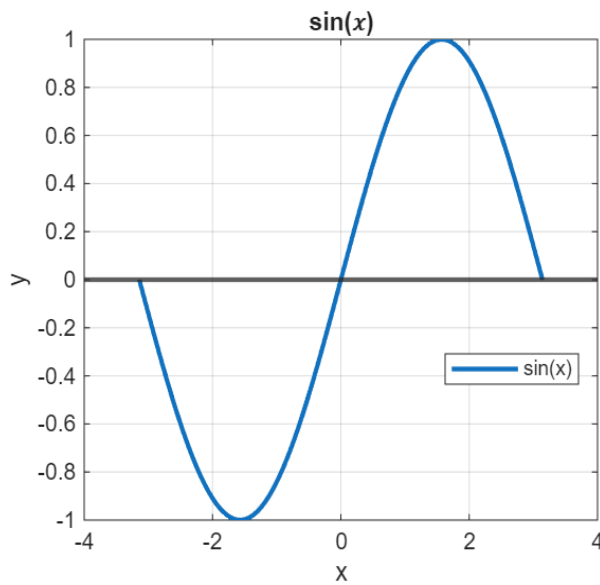


Figure 1 $y = \sin(x)$ plot

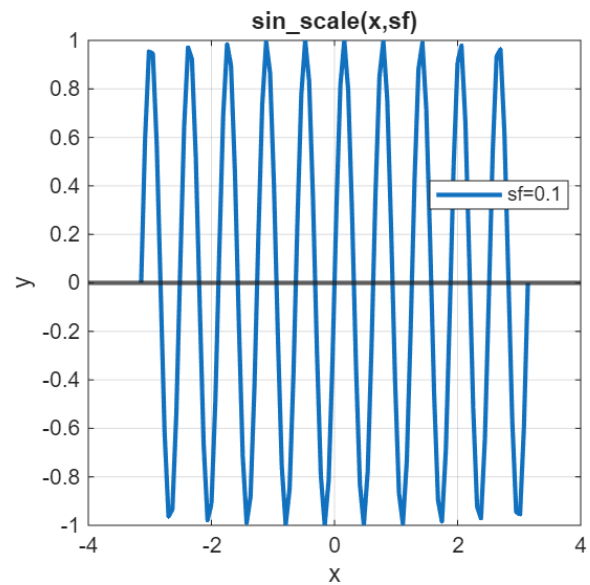


Figure 2 $y = \sin_scale(x, sf)$ plot (0.1 steps)

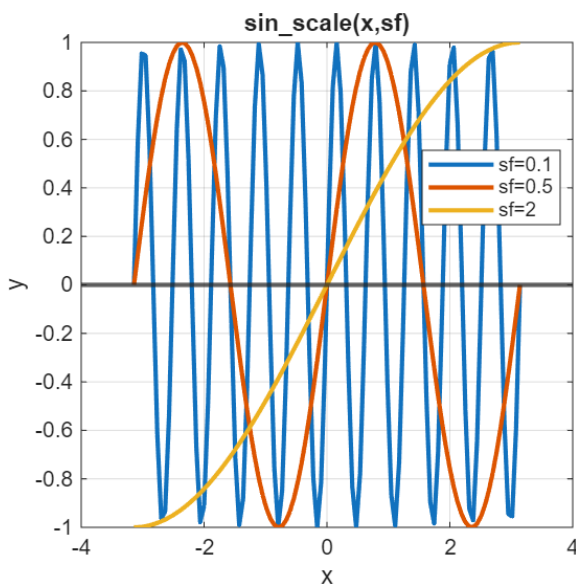


Figure 3 $y = \sin_scale(x, sf)$ plot ([0.1 0.5 2] steps)

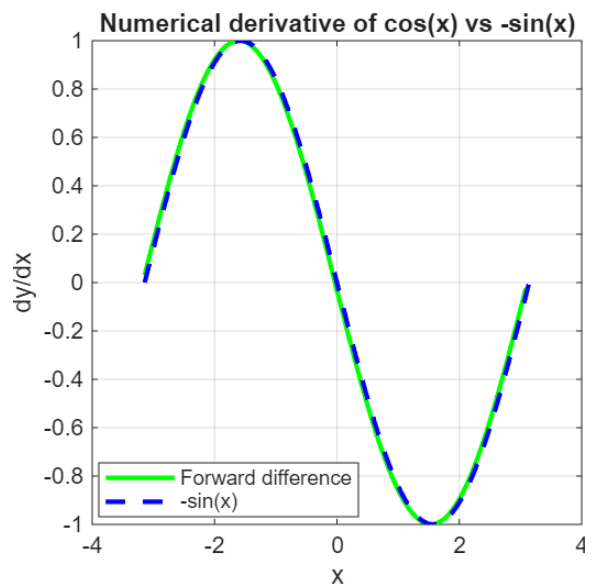


Figure 4 diff cos (x) to $-\sin(x)$

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

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
1.2337	1	23.370055 %

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	0.695634	0.693172	0.693153	0.693148
'n iterations'	Na	5	50	100	250
'percentage difference'	Na	0.35891	0.0036066	0.00090167	0.000144269

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

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

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	0.693169	0.693147	0.693147	0.693147983875
'n iterations'	NA	5	50	100	13
'percentage difference'	NA	0.0032623	7.2063e-07	4.5073e-08	0.000115893877

5 Integration results tables

5.1 Composite convergence iterations

N = 5		N = 50		N = 100	
Iteration	Result	Iteration	Result	Iteration	Result
1	0.266667	1	0.029608	1	0.014901
2	0.409524	10	0.190665	10	0.099857
3	0.534524	20	0.343631	20	0.186491
4	0.645635	30	0.476274	30	0.266214
5	0.695635	40	0.593365	40	0.340048
		50	0.693172	50	0.408803
				60	0.473134
				70	0.533575
				80	0.59057
				90	0.644491
				100	0.693153

5.2 Simpsons Convergence iterations

N = 5		N = 50		N = 100	
Iteration	Result	Iteration	Result	Iteration	Result
1	0.183333	1	0.019804	1	0.00995
2	0.336508	10	0.182322	10	0.09531
3	0.470437	20	0.336472	20	0.182322
4	0.587831	30	0.470004	30	0.262364
5	0.69317	40	0.587787	40	0.336472
		50	0.693147	50	0.405465
				60	0.470004
				70	0.530628
				80	0.587787
				90	0.641854
				100	0.693147

6 Appendices

6.1 Your code script and functions

6.1.1 Call Sheet

```
%% Part I ---ALL WORKING
clear; clc; clear -all;
format short g
```

```
% Variable Definition
```

```
N = 100;
```

```
x = linspace(-pi,pi,N)'; % ' after brackets transposes Row to Column vector
```

```

% Code
y = sin(x);

figure(1),
plot(x,y,'LineWidth',2); hold on; grid on;
yline(0,LineWidth=2);
legend('sin(x)','Location','best');
xlabel("x");
ylabel("y");
title('sin(x)');

%% Part II ---ALL WORKING
sf = 0.1;
y = sin_scale(x,sf);
figure(2);
plot(x,y,'LineWidth',2); hold on; grid on;
yline(0,LineWidth=2);
xlabel("x");
ylabel("y");
legend('sf=0.1','Location','best');
title('sin\_scale(x,sf)')

%For Loop
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) = 3

sin_array = zeros(N,N_sf); % creates zero matrix where Rows = N = 100, Columns =
N_sf = 3

%Initialise an array of zeros to store data. Notice the size of the array is
predicted by the number of values, N and the number of scale factors to be
analysed, N_sf.

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

end

figure(3),plot(x,sin_array,'LineWidth',2); grid on;
yline(0,LineWidth=2);
xlabel("x");
ylabel("y");
legend('sf=0.1','sf=0.5','sf=2','Location','best');
title('sin\_scale(x,sf)')

%% Part III - Numerical differentiation: cos(x) -> -sin(x) ---ALL WORKING

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

dy_estimate = Forward_Difference(x,y);
dy_exact = -sin(x);

% Plots
figure(1);
plot(x,dy_estimate,"g-",LineWidth=2);

```

```

grid on; hold on;
plot(x,dy_exact,"b--",LineWidth=2);
xlabel("x");
ylabel("dy/dx");
title("Numerical derivative of cos(x) vs -sin(x)");
legend("Forward difference","-sin(x)","Location","Best");

%% ---Numerical Methods---
%% Trapezoidal rule: integral of x*sin(x) from 0 to pi/2 ---ALL WORKING

f = @(x) x.*sin(x); % @ makes it a function handle
a = 0;
b = pi/2;
I = xsinx_Integral(a,b);

I_trap = trap1(f,a,b); % numerical result

% Analytical result of  $\int_0^{\pi/2} x \sin(x) dx = 1$ 
I_exact = 1;

percentage_error_Trap = abs(I_trap - I_exact) / I_exact * 100;

fprintf("Percentage error = %.6f %%\n", percentage_error_Trap)

%% Composite trapezoid for x*sin(x), n = 100 ---ALL WORKING

f = @(x) 1./x;
a = 1;
b = 2;
n = 100;

I_trap_comp = trap_composite(f,a,b,n);
I_exact = log(2); % Analitic result

absolute_error = abs(I_trap_comp-I_exact);
percentage_error = absolute_error/abs(I_exact)*100;

fprintf("n=%d I_trap_comp=%.12f I_exact=%.12f absolute_error=%.12e\n",n,I_trap_comp,I_exact,absolute_error,percentage_error)
;

%% Composite trapezoid for f(x) = 1/x from 1 to 2 ---ALL WORKING

n_list = [5 50 100];

I_values_trap_comp = zeros(size(n_list));
percentage_error_trap_comp = zeros(size(n_list));

for k = 1:numel(n_list)
    n = n_list(k);
    I_values_trap_comp(k) = trap_composite(f,a,b,n);
    percentage_error_trap_comp(k) = abs(I_values_trap_comp(k) - I_exact) / I_exact
* 100;
    fprintf("n=%3d I_values_trap(k)=%.12f\n",n,I_values_trap_comp(k))
end

% Table
I_exact_table = I_exact*ones(size(n_list));

```

```

Table =
table(n_list',I_values_trap_comp',I_exact_table',percentage_error_trap_comp','VariableNames',{'Intervals','Composite Trapezium','Exact','Percentage Error'});

disp(Table);
%% Composite trapezoid convergence (tolerance 1e-6) ---ALL WORKING

tol = 1e-6;
n = 1;
error = inf;

while error > tol
    I_n = trap_composite(f,a,b,n);
    error = abs(I_n - I_exact);
    n = n + 1;
end

% The loop exits when err <= tol; n has been incremented once too far
n_conv_trap = n - 1;          % number of intervals needed
I_conv_trap = I_n;           % integral at convergence
absolute_error = abs(I_conv_trap-I_exact);
percentage_error = absolute_error/abs(I_exact)*100;

fprintf("Required n = %d, I = %.12f, Percentage Error = %.12f", n_conv_trap,
I_conv_trap, percentage_error);

%% Simpsons Rule ---NEED TO FIGURE OUT ODD INTERVALS

n_list = [5 50 100];

I_values_simpsons = zeros(size(n_list));
percentage_error_simpsons = zeros(size(n_list));

for k = 1:numel(n_list)
    n = n_list(k);
    I_values_simpsons(k) = simpson_rule(f,a,b,n);
    percentage_error_simpsons(k) = abs(I_values_simpsons(k) - I_exact) / I_exact *
100;
    fprintf("n=%3d I_values_simpsons(k)=%.12f\n",n,I_values_simpsons(k))
end

I_simpsons = simpson_rule(f,a,b,n);

% Table
I_exact_table = I_exact*ones(size(n_list));
Table =
table(n_list',I_values_simpsons',I_exact_table',percentage_error_simpsons','VariableNames',{'Intervals','Simpsons','Exact','Percentage Error'});

disp(Table);

%% Simpsons Tolerance ---ALL WORKING

tol = 1e-6;
n = 1;
error = inf;

while error > tol
    I_n = simpson_rule(f,a,b,n);

```

```

        error = abs(I_n - I_exact);
        n = n + 1;
    end

% The loop exits when err <= tol; n has been incremented once too far
n_conv_simpson = n - 1;          % number of intervals needed
I_conv_simpson = I_n;           % integral at convergence
absolute_error = abs(I_conv_simpson-I_exact);
percentage_error = absolute_error/abs(I_exact)*100;

fprintf("Required n = %d, I = %.12f, Percentage Error = %.12f", n_conv_simpson,
I_conv_simpson, percentage_error);

```

6.1.2 Sin Scale Function

```

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);

% The '.' before an operator implies a vector/array operation

end

```

6.1.3 Forward Difference Function

```

function [df] = Forward_Difference(x,y)
% Inputs:
% x = x vector
% y = y vector

% Output: scalar or vector containing corresponding sine values.

n = numel(x);
if n < 2
    error("x must contain at least two points");
end

h = x(2) - x(1); % Since the step size is constant dont need tae make it a
variable, could if we wur gonnie put in different h values
df = zeros(size(y)); % This preallocates the matrix fur df

df(1:n-1) = (y(2:n)-y(1:n-1))/h;
df(n) = (y(n)-y(n-1))/h;

end

```

6.1.4 xsinx Integral Function

```

function I = xsinx_Integral(a,b)

F = @(x)-x.*cos(x)+sin(x);

```

```
I = F(b)-F(a); % Integral
end
```

6.1.5 Trapezoid Function

```
function Int = trap1(f,a,b)
% Inputs:
% f = function
% a = upper limit
% b = lower limit

% Output:
% Int = integration result

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

Int = 0.5*(feval(f,a)+feval(f,b))*(b-a);

end
```

6.1.6 Composite Trapezoid Function

```
function I = trap_composite(f,a,b,n)
% Inputs:
% f = function to be evaluated
% a = upper limit
% b = lower limit
% n = number of intervals

% Output:
% I = Integrated composite result

if nargin < 4
    error("Must have 4 inputs i.e. (f,a,b,n)")
end

if n <= 0 || floor(n) ~= n
    error("n must be a positive integer")
end

h = (b-a)/n; % This is delta x (step size)
x = a+(1:n-1)*h; % This is the interior points in a row vector
sum_f = sum(f(x)); % This is the sum of f at the interior points, x

I = 0.5*h*(f(a)+2*sum_f+f(b));
end
```

6.1.7 Simpsons Rule Function

```
function I = simpson_rule(f,a,b,n)
% Inputs: same as composite
```

```
% Output: Simpsons rule result

if nargin < 4
    error("Must have 4 inputs i.e. (f,a,b,n)")
end

if n <= 0 || floor(n) ~= n
    error("n must be a positive integer")
end

if mod(n,2) ~= 0
    warning('n must be an even number for simpsons rule, n will increase by 1');
    n = n+1;
end

h = (b-a)/n;
s = f(a)+f(b);

for i = 1:2:n-1
    x_odd = 4*f(a+i*h);
    s = s+x_odd;
end
for i = 2:2:n-2
    x_even = 2*f(a+i*h);
    s = s+x_even;
end

I = h/3*s;

end
```