HW4 Regression, Interpolation, and ODE

Name: Radityo Alhamdika Fadhilah
NIM : 13123136

1.) The specific heat of aluminium varies with temperature in the forowing table

T(°C)	-250	-200	-100	0	100	300
C_p (kJ/kg.K)	0.0163	0.318	0.699	0.870	0.941	1.04

a.) Determine Polynomial function by using lagrange interpolation method.

lagrange interpolation method

$$f_n(x) = \sum_{i=0}^{n} L_i(x)\, f(x_i) \quad \text{where} \quad L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

From $n+1 = 6$ data points, construct 5th-order lagrange polynomial

$$f_5(x) = f(x_0)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_0-x_0)(x_0-x_1)(x_0-x_2)(x_0-x_3)(x_0-x_4)(x_0-x_5)}$$
$$+ f(x_1)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_1-x_0)(x_1-x_1)(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)}$$
$$+ f(x_2)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_2-x_0)(x_2-x_1)(x_2-x_2)(x_2-x_3)(x_2-x_4)(x_2-x_5)}$$
$$+ f(x_3)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)(x_3-x_3)(x_3-x_4)(x_3-x_5)}$$
$$+ f(x_4)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_4-x_0)(x_4-x_1)(x_4-x_2)(x_4-x_3)(x_4-x_4)(x_4-x_5)}$$
$$+ f(x_5)\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_5-x_0)(x_5-x_1)(x_5-x_2)(x_5-x_3)(x_5-x_4)(x_5-x_5)}$$

let : $x_0 = -250 \rightarrow f(x_0) = 0.0163$
$x_1 = -200 \rightarrow f(x_1) = 0.318$
$x_2 = -100 \rightarrow f(x_2) = 0.699$
$x_3 = 0 \rightarrow f(x_3) = 0.870$
$x_4 = 100 \rightarrow f(x_4) = 0.941$
$x_5 = 300 \rightarrow f(x_5) = 1.04$

there is a tendency that $f(x_i)$ is Proportional with x_i

then, lagrange polynomial is

$$f_5(x) = 0{,}0163\frac{(x+200)(x+100)(x-0)(x-100)(x-300)}{(-250+200)(-250+100)(-250-0)(-250-100)(-250-300)}$$
$$+0{,}318\frac{(x+250)(x+100)(x-0)(x-100)(x-300)}{(-200+250)(-200+100)(-200+0)(-200-100)(-200-300)}$$
$$+0{,}699\frac{(x+250)(x+200)(x-0)(x-100)(x-300)}{(-100+250)(-100+200)(-100-0)(-100-100)(-100-300)}$$
$$+0{,}870\frac{(x+250)(x+200)(x+100)(x-100)(x-300)}{(0+250)(0+200)(0+100)(0-100)(0-300)}$$
$$+0{,}941\frac{(x+250)(x+200)(x+100)(x-0)(x-300)}{(100+250)(100+200)(100+100)(100-0)(100-300)}$$
$$+1{,}04\frac{(x+250)(x+200)(x+100)(x-0)(x-100)}{(300+250)(300+200)(300+100)(300-0)(300-100)}$$

$$f_5(x) = 0{,}0163\frac{(x+200)(x+100)x(x-100)(x-300)}{-3{,}60975\cdot 10^{11}}$$
$$+0{,}318\frac{(x+250)(x+100)(x)(x-100)(x-300)}{1{,}5\cdot 10^{11}}$$
$$+0{,}699\frac{(x+250)(x+200)(x)(x-100)(x-300)}{-1{,}2\cdot 10^{11}}$$
$$+0{,}870\frac{(x+250)(x+200)(x+100)(x-100)(x-300)}{1{,}5\cdot 10^{11}}$$
$$+0{,}941\frac{(x+250)(x+200)(x+100)(x-0)(x-300)}{-4{,}2\cdot 10^{11}}$$
$$+1{,}04\frac{(x+250)(x+200)(x+100)(x-0)(x-100)}{6{,}6\cdot 10^{12}}$$

$$f(x) = -4{,}51601\cdot 10^{-14}(x+200)(x+100)(x)(x-100)(x-300)$$
$$+2{,}12000\cdot 10^{-12}(x+250)(x+100)(x)(x-100)(x-300)$$
$$-5{,}825\cdot 10^{-12}(x+250)(x+200)(x)(x-100)(x-300)$$
$$+5{,}80000\cdot 10^{-12}(x+250)(x+200)(x+100)(x-100)(x-300)$$
$$-2{,}24047\cdot 10^{-12}(x+250)(x+200)(x+100)(x)(x-300)$$
$$+1{,}57575\cdot 10^{-13}(x+250)(x+200)(x+100)(x)(x-100)$$

b.) determine the Polynomial Function by using Newton method.

↳ From $n+1 = 6$ data Points, construct 5th order polynomial in Form of

$$f_n(x) = b_0 + b_1(x-x_0) + b_2(x-x_0)(x-x_1) + b_3(x-x_0)(x-x_1)(x-x_2) + b_4(x-x_0)(x-x_1)(x-x_2)(x-x_3) + b_5(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4) + \cancel{b_6(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}$$

where $b_0 = f(x_0)$

$b_1 = f[x_1, x_0]$

$b_2 = f[x_2, x_1, x_0]$

$b_3 = f[x_3, x_2, x_1, x_0]$

$b_4 = f[x_4, x_3, x_2, x_1, x_0]$

$b_5 = f[x_5, x_4, x_3, x_2, x_0]$

Calculation :

$b_0 = f(x_0) = f(-250) = 0.0163$

$$b_1 = f[x_1 - x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{0.318 - 0.0163}{-200-(-250)} = 6.034 \cdot 10^{-3}$$

$$b_2 = f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$

$$= \frac{\left[\frac{0.699 - 0.318}{-100-(-200)}\right] - 6.034 \cdot 10^{-3}}{-100-(-250)} = -1.48266 \cdot 10^{-5}$$

b_3 through b_5 are calculated computationally (see Problem 1 attachment)

$b_3 = f[x_3, x_2, x_1, x_0] = 1.73066 \cdot 10^{-8}$

$b_4 = f[x_4, x_3, x_2, x_1, x_0] = 2.933 \cdot 10^{-12}$

$b_5 = f[x_5, x_4, x_3, x_2, x_1, x_0] = -3.30606 \cdot 10^{-14}$

i	x	y	1st derivative	2nd derivative	3rd derivative	4th derivative	5th derivative
0	-250	0.0163					
			0.006034				
1	-200	0.318		$-1.48266 \cdot 10^{-5}$			
			0.003809		$1.7306 \cdot 10^{-8}$		
2	-100	0.699		$-1.04999 \cdot 10^{-5}$		$2.933 \cdot 10^{-12}$	
			0.001710		$1.8333 \cdot 10^{-8}$		$-3.30606 \cdot 10^{-14}$
3	0	0.870		$-5.0000 \cdot 10^{-5}$		$-1.5249 \cdot 10^{-11}$	
			0.000709		$1.0708 \cdot 10^{-8}$		
4	100	0.941		$-7.1666 \cdot 10^{-7}$			
			0.000495				
5	300	1.04					

therefore, the Polynomial Function achieved using Newton method is

$$f(x) = 0.0163 + (6.304 \cdot 10^{-3})(x+250) + (-1.48266 \cdot 10^{-5})(x+250)(x+200) + (1.73066 \cdot 10^{-8})(x+250)(x+200)(x+100) + (2.933 \cdot 10^{-12})(x+250)(x+200)(x+100)(x) + (-3.30606 \cdot 10^{-14})(x+250)(x+200)(x+100)(x)(x-100)$$

c.) Plot the data Points and the Polinomial Functions (a) and (b) in the same graph.
Is the graph Plot are attached in Problem 1 attachment.

d.) Calculate the value of C_p at $T = 200°C$ and $T = 400°C$ by using the above methods.

$T = 200°C$

- lagrange
  by direct substituting to the lagrange polynomial function, from matlab, we get value $f(200) = 0.9933$
- newton
  Substituting $x = 200$ to $f(x)$ in newton Polynomial equation, we get $f(200) = 0.9933$

which is the same value. lagrange and Newton interpolation method gives the same value since both methods are using highest Posible order of equation.

For $T = 400°C$, using the same Prochedure as $T = 200°C$, we get

- lagrange → $f(400) = 0.9860$
- Newton → $f(400) = 0.9860$

which also the same value.

It is seen from the graph of Problem 1C that lagrange and Newton Polynomial graph are collinear to each other.

Problem 1 Attachment

In this homework I am using both Python and MATLAB to numerically compute the calculations that is difficult to solve manually or to save some time in the calculations.

Problem 1a

```
# lagrange formula

x1 = [-250, -200, -100, 0, 100, 300]
y1 = [0.0163, 0.318, 0.699, 0.870, 0.941, 1.04]


denominators = []
for i in range(len(x1)):
    denominator = 1
    for j in range(len(x1)):
        if i != j:
            denominator *= (x1[i] - x1[j])

    denominators.append(denominator)


lagrange_coefficients = []

for i in range(len(denominators)):
    koef = y1[i] / denominators[i]
    lagrange_coefficients.append(koef)

print(lagrange_coefficients)
```

which gives the output :

```
[-4.516017316017316e-14, 2.12e-12, -5.825e-12, 5.8e-12, -2.24047619047619e-12, 1.575757575757576e-13]
```

Each term of the array is the value of

$$\sum_{i=0}^{n}\left(\frac{f(x_i)}{\prod_{j\neq i}(x_i - x_{j)}}\right)$$

in the equation of Lagrange polynomial

$$f_n(x) = \sum_{i=0}^{n}\left(\frac{f(x_i)}{\prod_{j\neq i}(x_i - x_{j)}}\right)\prod_{j\neq i}(x - x_j)$$

Lagrange polynomial can be expressed in the form of

$$f_n(x) = \sum_{i=0}^{n} C_i \prod_{j\neq i}(x - x_j)$$

Where C_i is Lagrange coefficients. This gives a function of x that can be plotted.

Problem 1b

```
x1 = [-250, -200, -100, 0, 100, 300]
y1 = [0.0163, 0.318, 0.699, 0.870, 0.941, 1.04]

# newton method

# df/dx = df/x2-x1
def first_derivative(x1,y1): # from x1[0] to x1[len(firstderivatives) - 1]
    firstderivatives = []
    for i in range(len(x1) - 1):
        derivative = (y1[i+1] - y1[i]) / (x1[i+1] - x1[i])
        firstderivatives.append(derivative)
    return firstderivatives

first_derivatives = first_derivative(x1,y1)
print(first_derivatives)

# dxnya adalah suku ke2-suku ke 0
def second_derivative (first_derivatives, x1):
    secondderivatives = []
    for i in range(len(first_derivatives) - 1):
        derivative = (first_derivatives[i+1]-first_derivatives[i]) / (x1[i+2] - x1[i])
        secondderivatives.append(derivative)
    return secondderivatives

second_derivatives = second_derivative(first_derivatives, x1)
print(second_derivatives)

def third_derivative(second_derivatives, x1):
    thirdderivatives = []
    for i in range(len(second_derivatives)-1):
        derivative = (second_derivatives[i+1]-second_derivatives[i]) / (x1[i+3] - x1[i])
        thirdderivatives.append(derivative)
    return thirdderivatives

third_derivatives = third_derivative(second_derivatives, x1)
print(third_derivatives)

def fourth_derivative(third_derivatives, x1):
    fourthderivatives = []
    for i in range(len(third_derivatives)-1):
        derivative = (third_derivatives[i+1]-third_derivatives[i]) / (x1[i+4] - x1[i])
        fourthderivatives.append(derivative)
    return fourthderivatives

fourth_derivatives = fourth_derivative(third_derivatives, x1)
print(fourth_derivatives)

def fifth_derivative(fourth_derivatives, x1):
    fifthderivatives = []
    for i in range(len(fourth_derivatives) - 1):
        derivative = (fourth_derivatives[i+1]-fourth_derivatives[i]) / (x1[i+5] - x1[i])
        fifthderivatives.append(derivative)
    return fifthderivatives

fifth_derivatives = fifth_derivative(fourth_derivatives, x1)
print(fifth_derivatives)
```

This gives output:

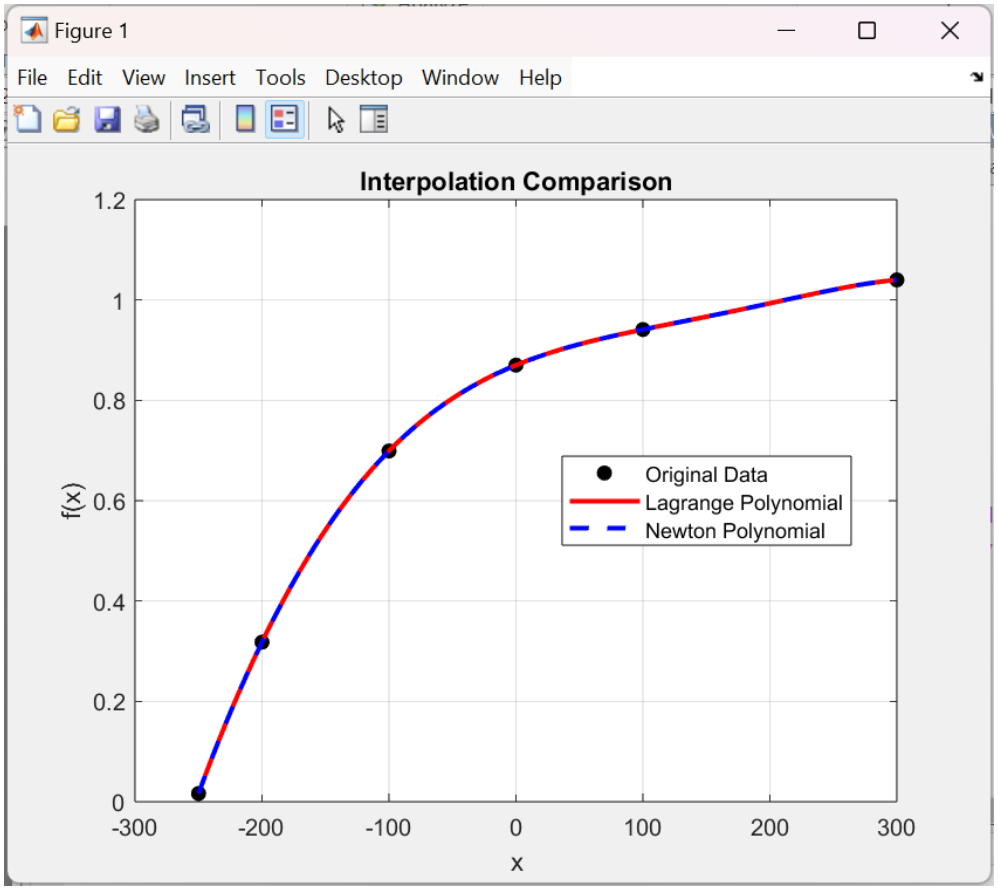
```
[0.006034, 0.0038099999999999996, 0.0017100000000000004, 0.0007099999999999995, 0.0004950000000000004]
[-1.4826666666666672e-05, -1.0499999999999998e-05, -5.000000000000005e-06, -7.166666666666635e-07]
[1.7306666666666695e-08, 1.833333333333331e-08, 1.0708333333333353e-08]
[2.9333333333331838e-12, -1.5249999999999912e-11]
[-3.3060606060605626e-14]
```

Each array on the output represents 1st derivatives, 2nd derivatives, 3rd derivatives, 4th derivatives, and 5th derivative respectively.

Problem 1c

```
% original data
x1 = [-250, -200, -100, 0, 100, 300];
y1 = [0.0163, 0.318, 0.699, 0.870, 0.941, 1.04];

% import all the necessary data obtained from previous calculation
% derivatives from problem 1b computation using newton finite dirference
d1 = [0.006034, 0.0038099999999999996, 0.0017100000000000004, 0.0007099999999999995,
0.0004950000000000004];
d2 = [-1.4826666666666672e-05, -1.0499999999999998e-05, -5.000000000000005e-06, -7.166666666666635e-
07];
d3 = [1.7306666666666695e-08, 1.833333333333331e-08, 1.0708333333333353e-08];
d4 = [2.9333333333331838e-12, -1.5249999999999912e-11];
d5 = -3.3060606060605626e-14;

% lagrange coefficients from problem 1a
lagrange_coeff = [-4.516017316017316e-14, 2.12e-12, -5.825e-12, 5.8e-12, -2.24047619047619e-12,
1.575757575757576e-13];

% lagrange polynomial calculation
n = length(x1);
syms x_sym
y_lagrange = 0; % <<< Initialize before use!

for i = 1:n
    % Build the numerator of the basis polynomial L_i(x)
    numerator = 1;
    for j = 1:n
        if i ~= j
            numerator = numerator * (x_sym - x1(j));
        end
    end
    % Add the term to the polynomial
    y_lagrange = y_lagrange + lagrange_coeff(i) * numerator;
end
disp(y_lagrange);

% newton polynomial calculation. please write the newton polynomial here
syms x_sym
x0 = x1(1);

% Define Newton coefficients
b0 = y1(1);
b1 = d1(1);
b2 = d2(1);
b3 = d3(1);
b4 = d4(1);
b5 = d5;

% Construct Newton Polynomial
term1 = b1 * (x_sym - x1(1));
term2 = b2 * (x_sym - x1(1)) * (x_sym - x1(2));
term3 = b3 * (x_sym - x1(1)) * (x_sym - x1(2)) * (x_sym - x1(3));
term4 = b4 * (x_sym - x1(1)) * (x_sym - x1(2)) * (x_sym - x1(3)) * (x_sym - x1(4));
term5 = b5 * (x_sym - x1(1)) * (x_sym - x1(2)) * (x_sym - x1(3)) * (x_sym - x1(4)) * (x_sym - x1(5));

% Final Newton Polynomial
y_newton = b0 + term1 + term2 + term3 + term4 + term5;
disp(y_newton)

% Convert symbolic expressions to functions
f_lagrange = matlabFunction(y_lagrange);
f_newton = matlabFunction(y_newton);

% Generate smooth x values
x_vals = linspace(min(x1), max(x1), 1000);
y_lag_vals = f_lagrange(x_vals);
y_new_vals = f_newton(x_vals);

% --- Plotting ---
figure;
plot(x1, y1, 'ko', 'MarkerFaceColor', 'k', 'DisplayName', 'Original Data');
hold on;
plot(x_vals, y_lag_vals, 'r-', 'LineWidth', 2, 'DisplayName', 'Lagrange Polynomial');
plot(x_vals, y_new_vals, 'b--', 'LineWidth', 2, 'DisplayName', 'Newton Polynomial');
xlabel('x');
ylabel('f(x)');
title('Interpolation Comparison');
legend('Location', 'best');
grid on;
```

This yield a graph



Original data are plotted as the black dots, lagrange polynomial function is plotted as the red line, and newton polynomial is plotted as blue-dashed line. All the graph are passing through the same points on the original data. Lagrange and newton polynomial function is seen to be collinear each other.

Problem 1d

The value of $T = 200°C$ and $T = 400°C$ are obtained using MATLAB

```
x_val = 200;
y_lagrange = subs(y_lagrange, x_sym, x_val);
disp(double(y_lagrange));
y_newton = subs(y_newton, x_sym, x_val);
disp(double(y_newton))

x_val = 400;
y_lagrange = subs(y_lagrange, x_sym, x_val);
disp(double(y_lagrange));
y_newton = subs(y_newton, x_sym, x_val);
disp(double(y_newton))
```

2.) The data for early steam engines are given in the table below.

Year	efficiency (%)	Type
1718	0.5	Newcomen
1767	0.8	Smeaton
1774	1.4	Smeaton
1775	2.7	Watt
1792	4.5	Watt
1816	7.5	Woolf compound
1828	12.0	Improved Cornish
1834	17.0	Improved Cornish
1878	17.2	Corliss Compound
1906	23.0	Triple expansion

a.) Fit a straight line, quadratic, and qubic regression lines to the data

- Straight line

let $y_i = a_0 + a_1 x_i + \text{error}$

by minimizing the error, we get equation

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \quad \text{and} \quad a_0 = \bar{y} - a_1 \bar{x}$$

assigning x_i = year and y_i = efficiency, calculating the necessary sum,

$n = 10$ $\quad \sum y_i = 86.6$ $\quad \sum x_i^2 = 32745494$

$\sum x_i y_i = 160486.3$ $\quad \bar{x} = 1808.8$

$\sum x_i = 18088$ $\quad \bar{y} = 8.66$

substitute to the equation

$$a_1 = \frac{10(160486.3) - (18088)(86.6)}{10(32745494) - (18088)^2} = 0.1376889$$

$$a_0 = 8.66 - 0.1376889(1808.8) = -240.3917463$$

therefore, straight line linear fit is

$y = -240.3917463 + 0.1376889\,x$

- Quadratic

let the function to fit the data is in form

$y = a_0 + a_1 x + a_2 x^2 + e$

by minimizing e, we get system of equation

$$n a_0 + (\Sigma x_i) a_1 + (\Sigma x_i^2) a_2 = \Sigma y_i$$
$$(\Sigma x_i) a_0 + (\Sigma x_i^2) a_1 + (\Sigma x_i^3) a_2 = \Sigma x_i y_i$$
$$(\Sigma x_i^2) a_0 + (\Sigma x_i^3) a_1 + (\Sigma x_i^4) a_2 = \Sigma x_i^2 y_i$$

in matrix form

$$\begin{bmatrix} n & \Sigma x_i & \Sigma x_i^2 \\ \Sigma x_i & \Sigma x_i^2 & \Sigma x_i^3 \\ \Sigma x_i^2 & \Sigma x_i^3 & \Sigma x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \\ \Sigma x_i^2 y_i \end{bmatrix}$$

calculating the necessary value

$\Sigma x_i^3 = 5.9331 \cdot 10^{10}$ $\quad \Sigma x_i^2 y_i = 2.9757 \cdot 10^8$

$\Sigma x_i^4 = 1.0759 \cdot 10^{14}$ $\quad$ *the rest value are already obtained in the previous calculation

Plug them into the matrix,

$$\begin{bmatrix} 10 & 18088 & 3274594 \\ 18088 & 3274594 & 5.9331 \cdot 10^{10} \\ 3274594 & 5.9331 \cdot 10^{10} & 1.0759 \cdot 10^{14} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 86.6 \\ 160486.3 \\ 2.9757 \cdot 10^8 \end{bmatrix}$$

From calculation with matlab, $a_0 = 858.5184$

$a_1 = -1.0740$

$a_2 = 3.3374 \cdot 10^{-4}$

Subtitute into the fit function,

$y = a_0 + a_1 x + a_2 x^2$

$y = 858.5184 - 1.0740x + 3.3374 \cdot 10^{-4} x^2$

- Qubic

let the function to fit the data is in the form

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

From the same prochedure as previous calculation, we get system equation in form of matrix

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 \\ \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \sum x_i^3 y_i \end{bmatrix}$$

these value and system of equations are solved with matlab.

$a_0 = 5,6323.10^4$

$a_1 = -93,1224$

$a_2 = 0,0512$

$a_3 = -9,3707 \cdot 10^{-6}$

Subtitute the values into the equation

$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$

$y = 5,6323.10^4 - 93,1224x + 0.0512x^2 - 9.3707.10^{-6} x^3$

b.) Compute standard deviation. determine which regression line that has the best fit

Standard deviation of residual is expressed in

$$S_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}}$$

$$S_r = \sum (y_{actual} - y_{predicted})^2$$

n = number of data

$m+1$ = number of regression parameters.

- linear

$$\hookrightarrow S_{td} = \sqrt{\frac{\sum_{i=0}^{10} (y_i - 8.66 + 0.137688 x_i)^2}{10-3}}$$

From matlab calculation, we get S_{td} for linear regression

$$S_{td_{linear}} = 2.8552$$

applying the same prochedure. we get

$$S_{td_{quadratic}} = 2.7684$$

$$S_{tdqubic} = 2.2658.$$

It is seen that the smallest standar deviation is the qubic equation therefore, regression line that has the best fit is qubic equation.

C.) best fit regression line and overlay to the data points can be seen in Problem 2 attachment.

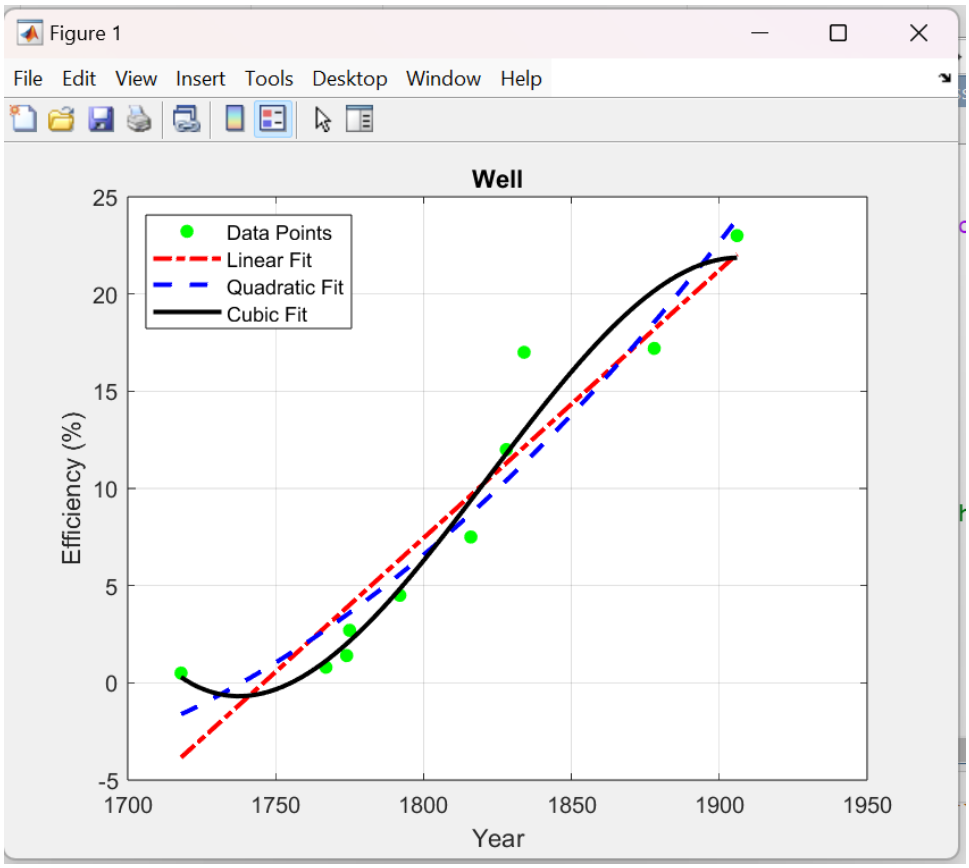
Problem 2 Attachment

MATLAB Code

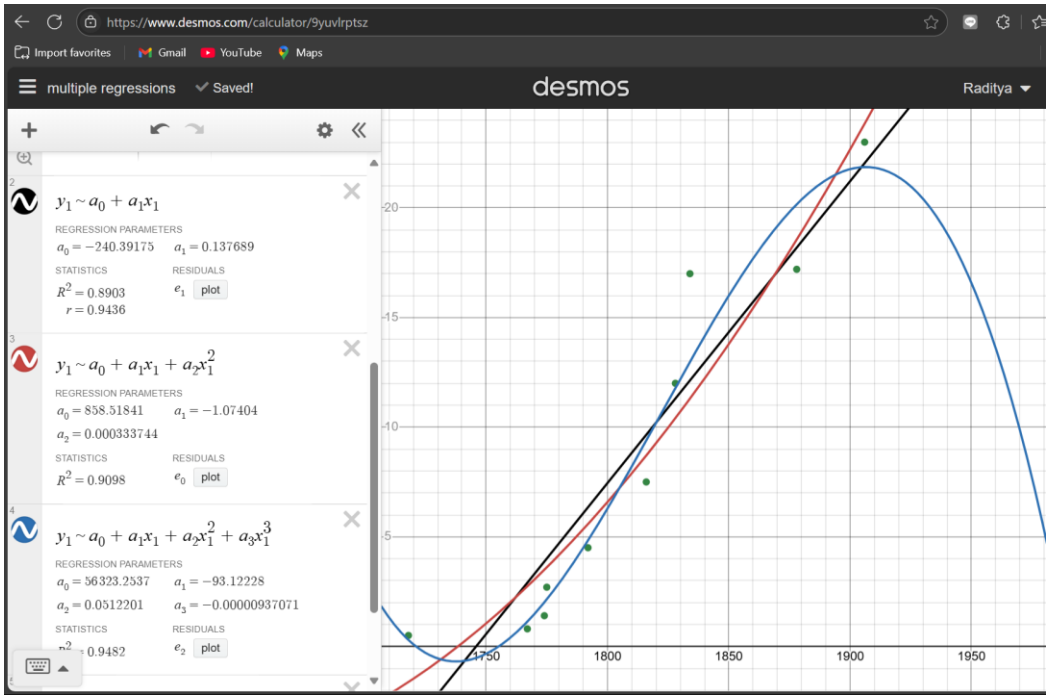
```
year = [1718, 1767, 1774, 1775, 1792, 1816, 1828, 1834, 1878, 1906];
efficiency = [0.5, 0.8, 1.4, 2.7, 4.5, 7.5, 12.0, 17.0, 17.2, 23.0];
type = ["Newcomen", "Smeaton", "smeaton", "Watt", "Watt", "Woolf compound", "Improved Cornish", "Improved
Cornish", "Corliss compound", "Triple expansion"];

x = year;
y = efficiency;

n = length(x);

sum_x = sum(x);
sum_y = sum(y);
sum_xy = sum(x .* y); % this means multply each element x with each coresponding y.
sum_x2 = sum(x.^2);

a1 = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x^2);
a0 = mean(y) - a1 * mean(x);

disp(a0);
disp(a1);

% Create regression line for plotting
x_fit = linspace(min(x), max(x), 100);
y_linear = a0 + a1 * x_fit;

% quadratic
% quadratic regression can be done by completing matrix
% general form y = a0 + a1*x + a2*x^2 + error
% n*a0 + sum(x)*a1 + sum(x^2)*a2 = sum(y)
% sum(x)*a0 + sum(x^2)*a1 + sum(x^3)*a2 = sum(xy)
% sum(x^2)*a0 + sum(x^3)*a1 + sum(x^4)*a2 = sum(x^2 * y)

% add some necessary sum
sum_x3 = sum(x.^3);
sum_x4 = sum(x.^4);
sum_x2y = sum((x.^2) .* y);

% left hand side of the matrix
A = [n       sum_x    sum_x2;
     sum_x   sum_x2   sum_x3;
     sum_x2  sum_x3   sum_x4];

% right hand side of the equation
b = [sum_y; sum_xy; sum_x2y];

coeffs2 = A \ b;
a0_q = coeffs2(1);
a1_q = coeffs2(2);
a2_q = coeffs2(3);

y_quad = a0_q + a1_q * x_fit + a2_q * x_fit.^2;

% cubic regression
sum_x5 = sum(x.^5);
sum_x6 = sum(x.^6);
sum_x3y = sum((x.^3) .* y);

A3 = [n        sum_x    sum_x2   sum_x3;
      sum_x    sum_x2   sum_x3   sum_x4;
      sum_x2   sum_x3   sum_x4   sum_x5;
      sum_x3   sum_x4   sum_x5   sum_x6];

b3 = [sum_y; sum_xy; sum_x2y; sum_x3y];

coeffs3 = A3 \ b3;
a0_c = coeffs3(1);
a1_c = coeffs3(2);
a2_c = coeffs3(3);
a3_c = coeffs3(4);

% Cubic Fit
y_cubic = a0_c + a1_c * x_fit + a2_c * x_fit.^2 + a3_c * x_fit.^3;

%% --- Plot Em All ---
figure;
plot(x, y, 'g.', 'MarkerSize', 20); hold on;
plot(x_fit, y_linear, 'r-.', 'LineWidth', 2);
plot(x_fit, y_quad, 'b--', 'LineWidth', 2);
plot(x_fit, y_cubic, 'k-', 'LineWidth', 2);
```

```
grid on;
xlabel('Year');
ylabel('Efficiency (%)');
title('Well');
legend('Data Points', 'Linear Fit', 'Quadratic Fit', 'Cubic Fit', 'Location', 'NorthWest');
hold off;

% standard deviation calculation
y_pred_linear = a0 + a1 * x;
y_pred_quad = a0_q + a1_q * x + a2_q * x.^2;
y_pred_cubic = a0_c + a1_c * x + a2_c * x.^2 + a3_c * x.^3;

residuals_linear = y - y_pred_linear;
residuals_quad = y - y_pred_quad;
residuals_cubic = y - y_pred_cubic;

n = length(x);
p_linear = 2; % y know
p_quad = 3;
p_cubic = 4;

std_linear = sqrt(sum(residuals_linear.^2) / (n - p_linear));
std_quad = sqrt(sum(residuals_quad.^2) / (n - p_quad));
std_cubic = sqrt(sum(residuals_cubic.^2) / (n - p_cubic));

disp(std_linear)
disp(std_quad)
disp(std_cubic)
```

This gives output in form of a graph of the data plot, straight line, quadratic, and cubic regression plot as well as calculate the standard deviation of each regression. Which turns out that qubic regression has the best fit criteria.



The regressions also can be done with Desmos and the result is the same as the MATLAB calculations



3.)

$$\frac{d^2y}{dx^2} + 0.6\frac{dy}{dx} + 8y = 0$$

Solve the initial value Problem over the interval $t=0$ to $t=5$ where $y(0)=4$ and $y'(0)=0$.

a.) analytically

seek solutions in the form of $y=e^{\lambda x}$ and take the derivatives

$y' = \lambda e^{\lambda x}$; $y'' = \lambda^2 e^{\lambda x}$

Substitute into the equation

$\lambda^2 e^{\lambda x} + 0.6\lambda e^{\lambda x} + 8e^{\lambda x} = 0$

$e^{\lambda x}$ terms cancels out and resulting $\lambda^2 + 0.6\lambda + 8 = 0$

Solve the quadratic equation

$\lambda_1 = -\frac{0.6}{2} + i\left(8 - \frac{1}{4}(0.6)^2\right)$; $\lambda_2 = -\frac{0.6}{2} - i\left(8 - \frac{1}{4}(0.6)^2\right)$

Since the roots is in Form $\lambda = \omega^2 = b - \frac{1}{4}a^2$, assigning $b=8$ and $a=0.6$, general solution of the ODE is

$$y = e^{-\frac{a}{2}x}(A\cos\omega x + B\sin\omega x)$$

Substitute the values.

$$y = e^{-0.3x}\left(A\cos\frac{\sqrt{791}}{10}x + B\sin\frac{\sqrt{791}}{10}x\right)$$

evaluate initial condition

$y(0)=4 \rightarrow e^0(A\cos 0 + B\cdot\sin 0) = 4 \rightarrow \boxed{A=4}$

$$y'(x) = 4\left[(-0.3)e^{-0.3x}\cos\left(\frac{\sqrt{791}}{10}x\right) + \frac{\sqrt{791}}{10}e^{-0.3x}\left(-\sin\left(\frac{\sqrt{791}}{10}x\right)\right)\right]$$
$$+ B\left[(-0.3)e^{-0.3x}\sin\left(\frac{\sqrt{791}}{10}x\right) + \frac{\sqrt{791}}{10}e^{-0.3x}\cos\left(\frac{\sqrt{791}}{10}x\right)\right]$$

$$y'(0) = 0 = 4\left[(-0.3)(1)(1) + 0\right] + B\left[0 + \frac{\sqrt{791}}{10}(1)(1)\right]$$

$$0 = -1.2 + B\frac{\sqrt{791}}{10}$$

$$\boxed{B = \frac{12}{\sqrt{791}}}$$

thus, the particular solution is

$$y = e^{-0.3x}\left[4\cos\left(\frac{\sqrt{791}}{10}x\right) + \frac{12}{\sqrt{791}}\sin\left(\frac{\sqrt{791}}{10}x\right)\right]$$

b.) euler method with $h = 0.5$

$$\frac{d^2y}{dx^2} + 0.6\frac{dy}{dx} + 8y = 0$$

let $\frac{dy}{dx} = \tilde{y}$, thus

$$\frac{d^2y}{dx^2} = \frac{d\tilde{y}}{dx} = -0.6\tilde{y} - 8y$$

constructing taylor formula,

$$\frac{dy}{dx} = \tilde{y} \rightarrow y = y_0 + \tilde{y}h$$

$$\frac{d\tilde{y}}{dx} = -0.6\tilde{y} - 8y \rightarrow \tilde{y} = \tilde{y}_0 + (-0.6\tilde{y} - 8y)h$$

writing in iteration form,

$$y(x_i) = y(x_{i-1}) + \tilde{y}(x_{i-1})h$$

$$\tilde{y}(x_i) = \tilde{y}(x_{i-1}) + (-0.6\tilde{y}(x_{i-1}) - 8y(x_{i-1}))h$$

iteration 1

$y(x_0) = 4$ $\quad h = 0.5$

$\tilde{y}(x_0) = 0$ $\quad x_1 = 0 + h = 0.5$

$$y(x_1) = 4 + 0(0.5) = 4$$

$$\tilde{y}(x_1) = 0 + (-0.6(0) - 8(4))(0.5) = -16$$

iteration 2

$y(x_1) = 4$ $\quad h = 0.5$

$\tilde{y}(x_1) = -16$ $\quad x_2 = x_1 + h = 1$

$$y(x_2) = 4 + (-16)(0.5) = -4$$

$$\tilde{y}(x_2) = -16 + (-0.6(-16) - 8(4)) = -27.2$$

iteration 3 through 10 are performed computationally

x	y
0	4
0.5	4
1	-4
1.5	-17.6
2	-19.12
2.5	15.016
3	77.1512
3.5	90.613
4	-54.264
4.5	-336.907
5	-426.2278

the Plot can be seen on Problem 3 attachment.

It's seen from the graph that numerical solution using $h=0.5$ isn't enough. If we use smaller h, i.e. 0.01, the graph approaches the exact solution from analytical in Problem a.

c.) second order RK method with $h = 0.5$

let $y' = \tilde{y} = \frac{dy}{dt}$, then

$$\frac{dy}{dt} = \tilde{y}$$

$$\frac{d\tilde{y}}{dt} = -0.6\tilde{y} - 8y$$

Initial condition

$y(0) = 4$

$\tilde{y}(0) = 0$

RK Formula

$$y_{i+1} = y_i + k_2^{(1)}$$

$$\tilde{y}_{i+1} = \tilde{y}_i + k_2^{(2)}$$

$$k_1^{(1)} = h \cdot f_1(x_i, y_i, \tilde{y}) = h \cdot \tilde{y}$$

$$k_1^{(2)} = h f_2(x_i, y_i, \tilde{y}_i) = h(-0.6\tilde{y} - 8y)$$

midpoint estimate

$$k_2^{(1)} = h \cdot f_1\left(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_1^{(1)}, \tilde{y}_i + \tfrac{1}{2}k_1^{(2)}\right)$$

$$k_2^{(2)} = h \cdot f_2\left(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_1^{(1)}, \tilde{y}_i + \tfrac{1}{2}k_1^{(2)}\right)$$

Iteration 1

$k_1^{(1)} = h \cdot \tilde{y} = 0.5 \cdot 0 = 0$

$k_1^{(2)} = h(-0.6\tilde{y} - 8y) = 0.5(-0.6 \cdot 0 - 8 \cdot 4) = -16$

↳ $k_2^{(1)} = 0.5\left(0 + \frac{1}{2}(-16)\right) = -4$

$k_2^{(2)} = 0.5\left[-0.6(-8) - 8(4)\right] = -13.6$

↳ $y = y_0 + k_2^{(1)} = 4 + (-4) = 0$

$\tilde{y} = \tilde{y}_0 + k_2^{(2)} = 0 + (-13.6) = -13.6$

Iteration 2

$k_1^{(1)} = 0.5(-13.6) = -6.8$

$k_1^{(2)} = 0.5(-0.6(-13.6) - 8(0)) = 4.08$

↳ $k_2^{(1)} = 0.5\left(-13.6 + \frac{1}{2}(4.08)\right) = -5.78$

$k_2^{(2)} = 0.5\left(-0.6(-11.56) - 8(-3.4)\right) = 17.068$

↳ $y = y_0 + k_2^{(1)} = 0 + (-5.78) = -5.78$

$\tilde{y} = \tilde{y}_0 + k_2^{(2)} = -13.6 + 17.068 = 3.468$

Iteration 3 and so on are calculated with computer.

x	y
0	4
0,5	0
1	-5,779
1,5	1,473
2	7,9762
2,5	-4,1637
3	-10,4639
3,5	8,6848
4	12,9057
4,5	-15,840
5	-14,6094

d.) 4th order Rk

Convert to 1st order system

$$\frac{dy}{dt} = \tilde{y}$$

$$\frac{d\tilde{y}}{dt} = -0.6\tilde{y} - 8y$$

Initial conditions

$y(0) = 4$

$\tilde{y}(0) = 0$

Rk Formula

$$y_{i+1} = y_i + \frac{1}{6}\left(k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)}\right)$$

$$\tilde{y}_{i+1} = \tilde{y}_i + \frac{1}{6}\left(k_1^{(2)} + 2k_2^{(2)} + 2k_3^{(2)} + k_4^{(2)}\right)$$

where

$$k_1^{(1)} = h\, f_1(x_i, y_i, \tilde{y}_i) = h \cdot \tilde{y}_i$$

$$k_1^{(2)} = h\, f_2(x_i, y_i, \tilde{y}_i) = h \cdot (-0.6\tilde{y}_i - 8y_i)$$

$$k_2^{(1)} = h\, f_1\left(x_i + \tfrac{1}{2}h,\ y_i + \tfrac{1}{2}k_1^{(1)},\ \tilde{y}_i + \tfrac{1}{2}k_1^{(2)}\right)$$

$$k_2^{(2)} = h\, f_2\left(x_i + \tfrac{1}{2}h,\ y_i + \tfrac{1}{2}k_1^{(1)},\ \tilde{y}_i + \tfrac{1}{2}k_1^{(2)}\right)$$

$$k_3^{(1)} = h\, f_1\left(x_i + \tfrac{1}{2}h,\ y_i + \tfrac{1}{2}k_2^{(1)},\ \tilde{y}_i + \tfrac{1}{2}k_2^{(2)}\right)$$

$$k_3^{(2)} = h\, f_1\left(x_i + \tfrac{1}{2}h,\ y_i + \tfrac{1}{2}k_2^{(1)},\ \tilde{y}_i + \tfrac{1}{2}k_2^{(2)}\right)$$

$$k_4^{(1)} = h\, f_1\left(x_i + h,\ y_i + k_3^{(1)},\ \tilde{y}_i + k_3^{(2)}\right)$$

$$k_4^{(2)} = h\, f_2\left(x_n + h,\ y_i + k_3^{(1)},\ \tilde{y}_i + k_3^{(2)}\right)$$

Iteration 1

$k_1^{(1)} = 0.5(0) = 0$

$k_1^{(2)} = 0.5(-0.6 \cdot 0 - 8 \cdot 4) = 0.5(-32) = -16$

$k_2^{(1)} = -4$

$k_2^{(2)} = -13.6$

⋮

and so on

Calculate computationally, yield

x	y
0	4
0.5	1.0366
1	-2.4275
1.5	-1.5571
2	1.1538
2.5	1.4810
3	-0.2934
3.5	-1.13192
4	-0.1852
4.5	0.7241
5	0.378

e.) Predictor-corrector with $h = 1.0$.

convert to first order system

$$\frac{dy}{dx} = \tilde{y} \qquad y(0) = 4$$

$$\frac{d\tilde{y}}{dx} = -0.6\tilde{y} - 8y \qquad \tilde{y}(0) = 0$$

- Predictor : $y_{i+1}^{(p)} = y_i + h f(x_i, y_i)$
- Corrector : $y_{i+1} = y_i + \frac{h}{2}\left[f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{(p)})\right]$

apply to the system,

$$Y' = \begin{bmatrix} y' \\ \tilde{y}' \end{bmatrix} = \begin{bmatrix} \tilde{y} \\ -0.6\tilde{y} - 8y \end{bmatrix}$$

Iteration 1

$$f_1 = \tilde{y} = 0$$

$$f_2 = -0.6\tilde{y} - 8y = -0.6(0) - 8(4) = -32$$

↳ Predictor

↳ $y^{(p)} = y_1 + h f_1 = 4 - 1(0) = 4$

$\tilde{y}^{(p)} = \tilde{y} + h f_2 = 0 + 1(-32) = -32$

↳ Predicted

↳ $f_1^{(p)} = \tilde{y}^{(p)} = -32$

$f_2^{(p)} = -0.6(-32) - 8.4 = 19.2 - 32 = -12.8$

↳ Corrector

↳ $y_1(1) = y_1 + \frac{1}{2}h(f_1 + f_1^{(p)}) = 4 + \frac{1}{2}(0 + (-32)) = 4 - 16 = -12$

$\tilde{y}(1) = \tilde{y} + \frac{1}{2}h(f_2 + f_2^{(p)}) = 0 + \frac{1}{2}(-32 + (-12.8)) = -22.4$

$\Rightarrow$ $y(1) = -12$

$\tilde{y}(1) = -22.4$

Iteration 2

$y_1 = -12$

$\tilde{y} = -22.4$

$f_1 = y_2 = -22.4$

$f_2 = -0.6(-22.4) - 8(-12) = 13.44 + 96 = 109.44$

Predictor

↳ $y_i^{(p)} = -12 + 1(-22.4) = -34.4$

$\tilde{y}^{(p)} = -22.4 + 1(109.44) = 87.04$

Predicted

↳ $f_1^{(p)} = 87.04$

$f_2^{(p)} = -0.6(87.04) - 8(-34.4) = -52.224 + 275.2 = 222.976$

Corrector

↳ $y(2) = -12 + \frac{1}{2}(-22.4 + 87.4) = -12 + 32.32 = 20.32$

$\tilde{y}(2) = -22.4 + \frac{1}{2}(109.44 + 222.976) = -22.4 + 166.208 = 143.808$

$\Rightarrow y_1(2) = 20.32$

$\tilde{y}(2) = 143.808$

the remaining iterations are calculated computationally

i	x	y
0	0	4
1	1	-12
2	2	20.32
3	3	39.76
4	4	-543.04
5	5	2923.39

Problem 3 Attachment

Matlab code :

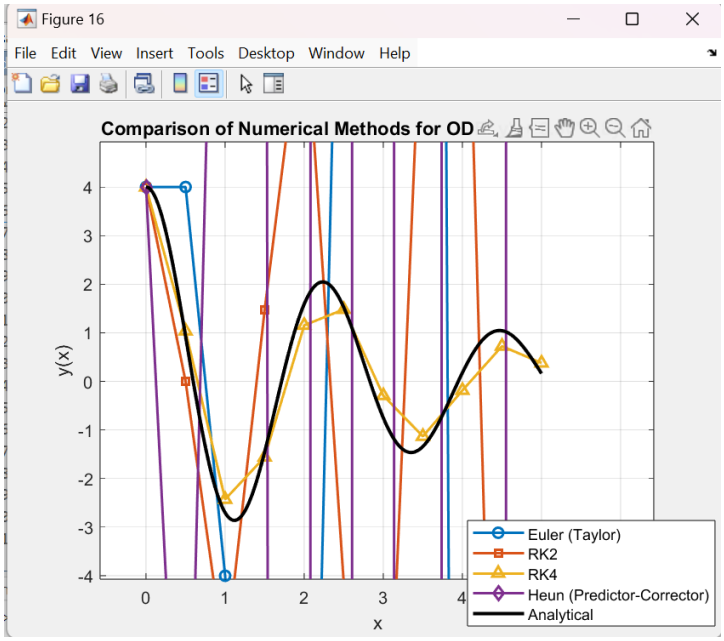
```
format long
clear; clc;
% Parameters
h = 0.5;
x = 0:h:5;
N = length(x);
% Initial conditions
y0 = 4;
dy0 = 0;
%% 1. Euler's Method (Taylor 1st Order Approximation)
y_euler = zeros(1, N);
dy_euler = zeros(1, N);
y_euler(1) = y0;
dy_euler(1) = dy0;

for i = 1:N-1
    ddy = -0.6 * dy_euler(i) - 8 * y_euler(i);
    y_euler(i+1) = y_euler(i) + h * dy_euler(i);
    dy_euler(i+1) = dy_euler(i) + h * ddy;
end

%% 2. Runge-Kutta 2nd Order (Midpoint Method)
y_rk2 = zeros(1, N);
dy_rk2 = zeros(1, N);
y_rk2(1) = y0;
dy_rk2(1) = dy0;

for i = 1:N-1
    f1 = dy_rk2(i);
    f2 = -0.6 * dy_rk2(i) - 8 * y_rk2(i);

    k1_1 = h * f1;
    k1_2 = h * f2;

    k2_1 = h * (dy_rk2(i) + k1_2/2);
    k2_2 = h * (-0.6*(dy_rk2(i) + k1_2/2) - 8*(y_rk2(i) + k1_1/2));

    y_rk2(i+1) = y_rk2(i) + k2_1;
    dy_rk2(i+1) = dy_rk2(i) + k2_2;
end

%% 3. Runge-Kutta 4th Order Method
y_rk4 = zeros(1, N);
dy_rk4 = zeros(1, N);
y_rk4(1) = y0;
dy_rk4(1) = dy0;

for i = 1:N-1
    x_i = x(i);
    y1 = y_rk4(i);
    y2 = dy_rk4(i);

    k1_1 = h * y2;
    k1_2 = h * (-0.6*y2 - 8*y1);

    k2_1 = h * (y2 + k1_2/2);
    k2_2 = h * (-0.6*(y2 + k1_2/2) - 8*(y1 + k1_1/2));

    k3_1 = h * (y2 + k2_2/2);
    k3_2 = h * (-0.6*(y2 + k2_2/2) - 8*(y1 + k2_1/2));

    k4_1 = h * (y2 + k3_2);
    k4_2 = h * (-0.6*(y2 + k3_2) - 8*(y1 + k3_1));

    y_rk4(i+1) = y1 + (k1_1 + 2*k2_1 + 2*k3_1 + k4_1)/6;
    dy_rk4(i+1) = y2 + (k1_2 + 2*k2_2 + 2*k3_2 + k4_2)/6;
end

%% 4. Predictor-Corrector (Heun's Method)
```

```
y_heun = zeros(1, N);
dy_heun = zeros(1, N);
y_heun(1) = y0;
dy_heun(1) = dy0;
h = 1;
for i = 1:N-1
    f1 = dy_heun(i);
    f2 = -0.6 * dy_heun(i) - 8 * y_heun(i);

    y1_pred = y_heun(i) + h * f1;
    dy1_pred = dy_heun(i) + h * f2;

    f1_pred = dy1_pred;
    f2_pred = -0.6 * dy1_pred - 8 * y1_pred;

    y_heun(i+1) = y_heun(i) + (h/2)*(f1 + f1_pred);
    dy_heun(i+1) = dy_heun(i) + (h/2)*(f2 + f2_pred);
end


%% Plot All Results
figure;
plot(x, y_euler, '-o', 'LineWidth', 1.5); hold on;
plot(x, y_rk2, '-s', 'LineWidth', 1.5);
plot(x, y_rk4, '-^', 'LineWidth', 1.5);
plot(x, y_heun, '-d', 'LineWidth', 1.5);

xlabel('x');
ylabel('y(x)');
title('Comparison of Numerical Methods for ODE: y'''' + 0.6y'' + 8y = 0');
legend('Euler (Taylor)', 'RK2', 'RK4', 'Heun (Predictor-Corrector)', 'Location', 'best');
grid on;
```

From the code, we get the values for each method

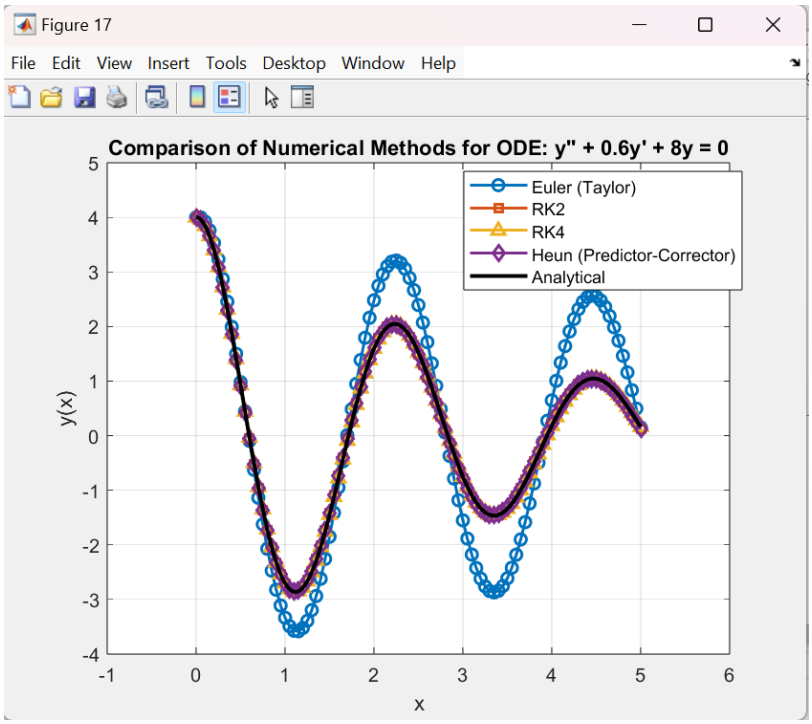
x	Euler_method	RK2_method	RK4_method	Predictor_corrector
0	4	4	4	4
0.5	4	0	1.036666667	
1	-4	-5.78	-2.427559569	-12
1.5	-17.6	1.4739	-1.557105908	
2	-19.12	7.9762555	1.153883434	20.32
2.5	15.016	-4.163730653	1.481014982	
3	77.1512	-10.46393788	-0.293481674	39.7056
3.5	90.61384	8.684894953	-1.131923849	
4	-54.264712	12.90574203	-0.185286249	-543.047552
4.5	-336.9073784	-15.84063742	0.724147919	
5	-426.2278209	-14.60943468	0.378206035	2923.339876

Plot them in the same graph yield



Solution graph with h = 0.5 and h = 0.1 for predictor-corrector

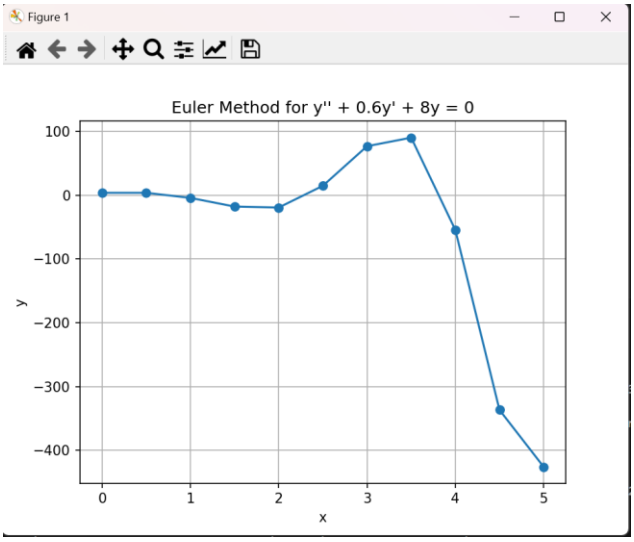
Sure its not accurate. To enhance the accuracy, set h to be much smaller. For example, Iet h is set to be 0.05 for all methods.



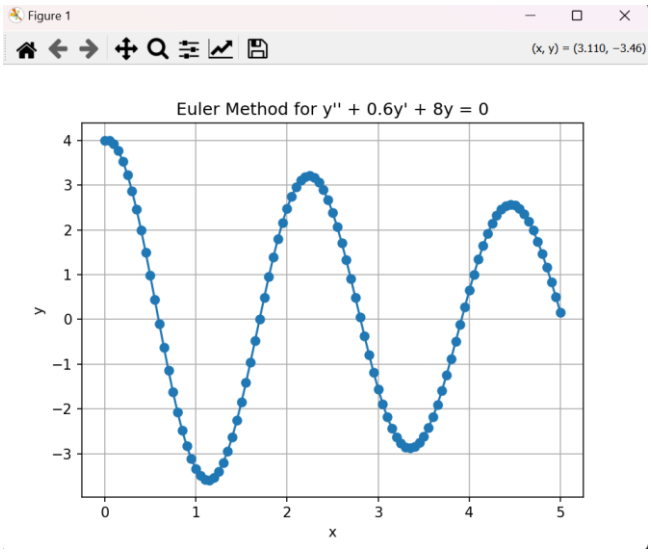
Solution graph using h = 0.05

Individual graph of each methods are shown below

a. Euler method

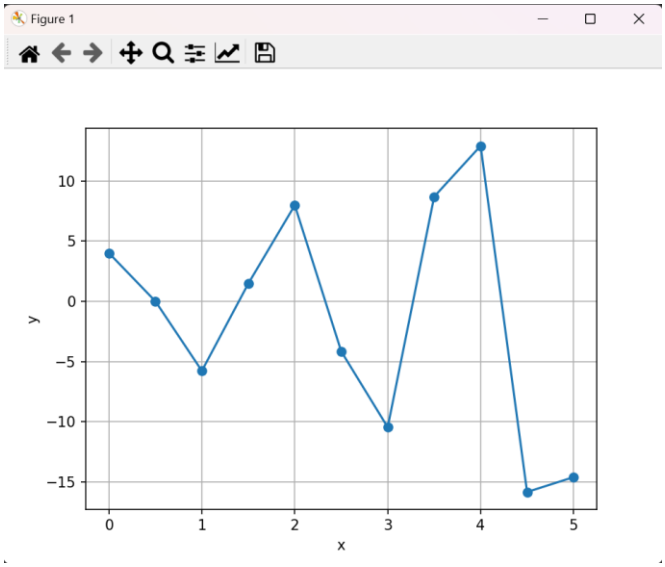


h = 0.5

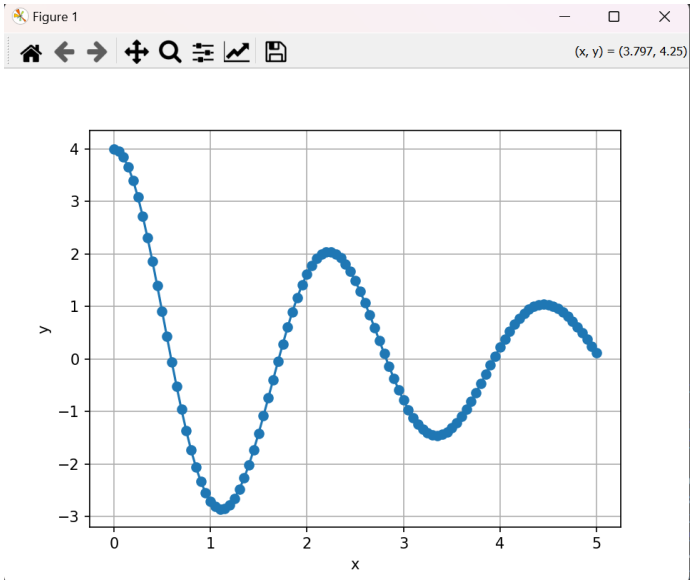


H = 0.05

b. RK2 method

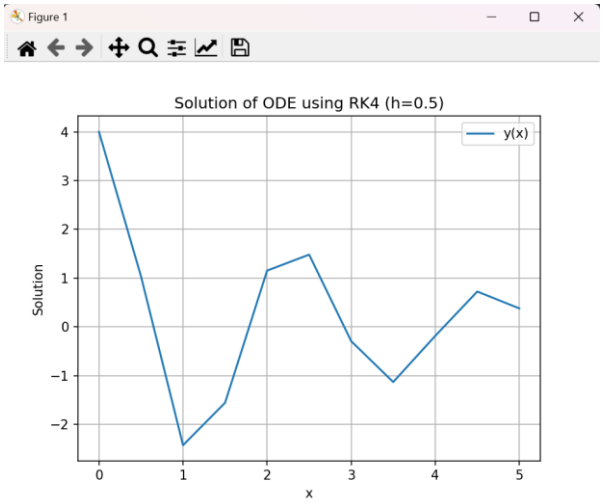


h = 0.5

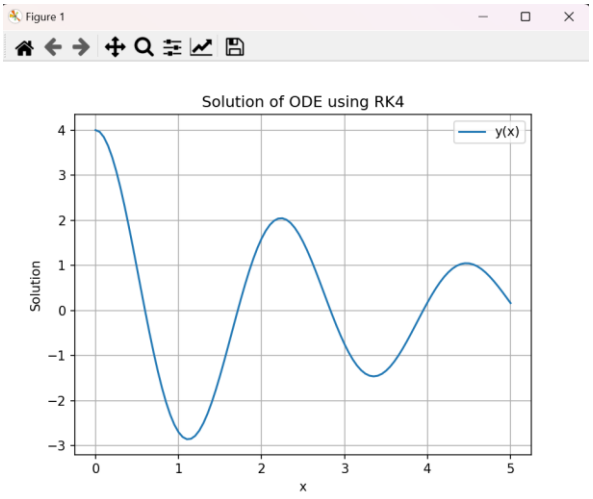


h = 0.05

c. RK4 method



h = 0.5



h = 0.05

d. Predictor-corrector

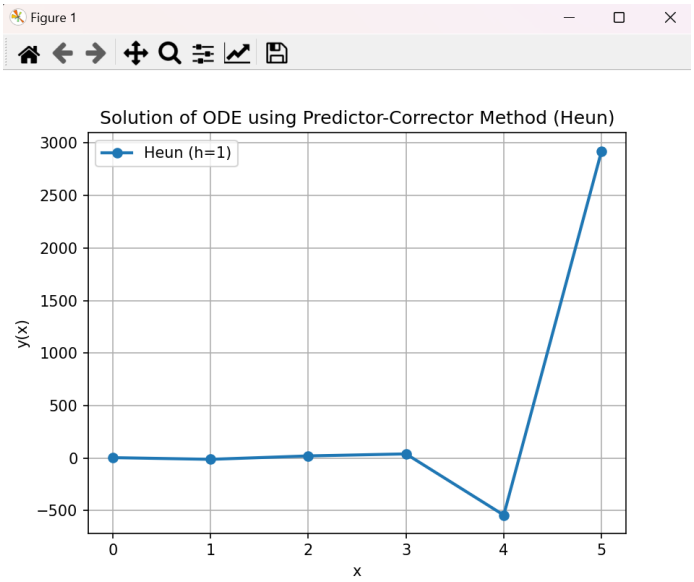


Figure 1
Solution of ODE using Predictor-Corrector Method (Heun)
Heun (h=1)
y(x)
x
3000
2500
2000
1500
1000
500
0
−500
0
1
2
3
4
5

$h = 1$

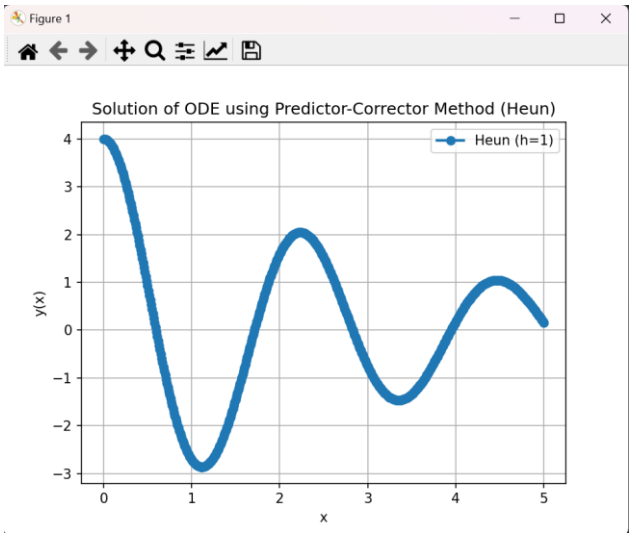


Figure 1
Solution of ODE using Predictor-Corrector Method (Heun)
Heun (h=1)
y(x)
x
4
3
2
1
0
−1
−2
−3
0
1
2
3
4
5

$h = 0.01$

4.) Use numerical integration method to solve

$$\int_1^{100} \frac{e^{-x}}{x}\, dx$$

for various number of partition $N = 10, 20, 40, 100, 1000$

a.) Trapezoidal Rule

$$I = (b-a)\frac{f(a)+f(b)}{2}$$

For each iteration,

$$I = h\left(\frac{f(x_i)+f(x_i+h)}{2}\right)$$

where $h = \frac{100-1}{N}$ (for each N)

b.) Simpson 1/3 rule

$$I = h\left(\frac{f(x_i)+4f(x_i+\frac{1}{2}h)+f(x_i+h)}{6}\right)$$

c.) Simpson 3/8 rule

$$I = h\left(\frac{f(x_i)+3f(x_i+\frac{1}{4}h)+3f(x_i+\frac{3}{4}h)+f(x_i+h)}{8}\right)$$

the calculations are done computationally. See Problem 4 attachment.

Problem 4 attachment

Matlab Code

```
format long
% Define the function
the_function = @(x) exp(-x) ./ x;

integration_range = [1, 100];
N = [10, 20, 40, 100, 1000];

trapezoidal_values = zeros(1, length(N));

% Trapezoidal Rule
for i = 1:length(N)
    partition_width = (integration_range(2) - integration_range(1)) / N(i);
    integration_position = integration_range(1);
    total_area = 0;
    for j = 1:N(i)
        % Trapezoidal formula: 0.5*(b-a)*(f(b) + f(a))
        a = integration_position;
        b = integration_position + partition_width;
        area = 0.5 * partition_width * (the_function(a) + the_function(b));
        total_area = total_area + area;
        integration_position = integration_position + partition_width;
    end
    trapezoidal_values(i) = total_area;
end

% Display trapezoidal values
disp('Trapezoidal Method Results:')
disp(trapezoidal_values)


% Simpson 1/3 Rule
simpson13_value = zeros(1, length(N));

for i = 1:length(N)
    partition_width = (integration_range(2) - integration_range(1)) / N(i);
    integration_position = integration_range(1);
    total_area = 0;
    for j = 1:N(i)
        x0 = integration_position;
        x1 = integration_position + partition_width / 2;
        x2 = integration_position + partition_width;
        area = partition_width * (the_function(x0) + 4 * the_function(x1) + the_function(x2)) / 6;
        total_area = total_area + area;
        integration_position = integration_position + partition_width;
    end
    simpson13_value(i) = total_area;
end

% Display Simpson 1/3 values
disp('Simpson 1/3 Method Results:')
disp(simpson13_value)


% Simpson 3/8 Rule
simpson38_value = zeros(1, length(N));

for i = 1:length(N)
    partition_width = (integration_range(2) - integration_range(1)) / N(i);
    integration_position = integration_range(1);
    oneforth_width = 0.25 * partition_width;
    threeforth_width = 0.75 * partition_width;
    total_area = 0;
    for j = 1:N(i)
        x0 = integration_position;
        x1 = integration_position + oneforth_width;
        x2 = integration_position + threeforth_width;
        x3 = integration_position + partition_width;
        area = partition_width * (the_function(x0) + 3 * the_function(x1) + 3 * the_function(x2) +
the_function(x3)) / 8;
```

```
        total_area = total_area + area;
        integration_position = integration_position + partition_width;
    end
    simpson38_value(i) = total_area;
end

% Display Simpson 3/8 values
disp('Simpson 3/8 Method Results:')
disp(simpson38_value)


% Plotting the integration results
figure;
plot(N, trapezoidal_values, '-o', 'LineWidth', 2);
hold on;
plot(N, simpson13_value, '-s', 'LineWidth', 2);
plot(N, simpson38_value, '-^', 'LineWidth', 2);
hold off;

xlabel('Number of Partitions (N)');
ylabel('Approximated Integral Value');
title('Comparison of Numerical Integration Methods');
legend('Trapezoidal Rule', 'Simpson 1/3 Rule', 'Simpson 3/8 Rule');
xlim([-200, 1100]);
ylim([0, 2.3]);

grid on;
```

This code compute the numerical methods for N = 10, 20, 40, 100, and 1000 in one go.
The results are stored in `trapezoidal_values`, `simpson13_value`, and `simpson38_value` Variables in form of array.

N	Trapezoidal	Simpson 1/3	Simpson 3/8
10	1.821019999	0.609897233	0.488430006
20	0.912677924	0.333714869	0.320455943
40	0.478455633	0.241517621	0.256788135
100	0.273723906	0.220710602	0.227622935
1000	0.219984084	0.21938413	0.219477667

All of three methods converge to the exact solution 0.219…….