1、

(a)

$$s = \frac{x - x_0}{h} = \frac{x - 0.12}{0.12}$$

$$P(x) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 \quad (\text{degree } 2)$$

$$= 0.79168 - 0.01834 \times \frac{x-0.12}{0.12} - 0.01129 \times \frac{\left(\frac{x-0.12}{0.12}\right)\left(\frac{x-0.24}{0.12}\right)}{2!}$$

$$P(0.231) = 0.79168 - 0.01834 \times \frac{0.111}{0.12} - 0.01129 \times \frac{\frac{0.111}{0.12} \times \frac{-0.009}{0.12}}{2} \fallingdotseq 0.77510 \ \#$$

(b)

$$p(x) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0$$

$$P(0.231) = 0.77510 + \frac{\frac{0.111}{0.12} \times \frac{-0.009}{0.12} \times \frac{-0.129}{0.12}}{6} \times 0.00134$$

$$= 0.77512 \ \#$$

(c)

$$\text{Error of part (a)} = \Delta^3 f \times \frac{s(s-1)(s-2)}{3!} = 0.00134 \times \frac{\frac{0.111}{0.12} \times \frac{-0.009}{0.12} \times \frac{-0.129}{0.12}}{6}$$

$$= 0.0000167$$

$$\text{Error of part (b)} = \Delta^4 f \, \frac{s(s-1)(s-2)(s-3)}{4!} = 0.00038 \times \frac{0.111 \times (-0.009) \times (-0.129) \times (-0.249)}{24 \times (0.12)^4}$$

$$= -0.00000245 \ \#$$

(d) Cacalate the error $\frac{x-0.24}{0.12}$

$$x_0 = 0.24 \Rightarrow \text{error} = 0.00172 \times \frac{0.18 \times 0.06 \times (-0.06)}{3!\,(0.12)^3} = -0.0001075$$

$$x_0 = 0.36 \Rightarrow \text{error} = 0.002 \times \frac{0.06 \times (-0.06) \times (-0.18)}{3!\,(0.12)^3} = 0.000125$$

the error start from $x_0 = 0.24$ is smaller, so choose x_0 start from 0.24 is better.

2.

Use the below equation to solve all the S0 ~ Sn (h0 ~ hn-1 is easy to calculate)

Matrix for Computing Cubic Spline Coefficients

$$\begin{bmatrix} h_0 & 2(h_0+h_1) & h_1 & & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & & \\ & & h_2 & 2(h_2+h_3) & h_3 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_{n-1} \\ S_n \end{bmatrix} = 6 \begin{bmatrix} f[x_1,x_2]-f[x_0,x_1] \\ f[x_2,x_3]-f[x_1,x_2] \\ f[x_3,x_4]-f[x_2,x_3] \\ \vdots \\ f[x_{n-1},x_n]-f[x_{n-2},x_{n-1}] \end{bmatrix}$$

$$h_i = x_{i+1} - x_i$$

$$S_i = g''_i(x_i), \quad i = 1, \ldots, n-1$$

Numerical Methods © Wen-Chieh Lin 15

Since the problem says the slope at the ends are 0, we should slightly modify the matrix.

According to the image below, when A=B=0, 2h_0S_0 + h_0S_1 = 6(f [x0, x1] - 0). We can use this to modify the matrices.

- $f'(x_0) = A$ and $f'(x_n) = B$,
- Left: $i = 0$ $\quad 2h_0S_0 + h_0S_1 = 6(f[x_0, x_1] - A)$
- Right: $i = n$ $\quad h_{n-1}S_{n-1} + 2h_{n-1}S_n = 6(B - f[x_{n-1}, x_n])$

I construct the big matrix A and the corresponding matrix b and solve all the S0~Sn first. The first six lines of codes is the modification of matrix because of 0 slope at the ends.

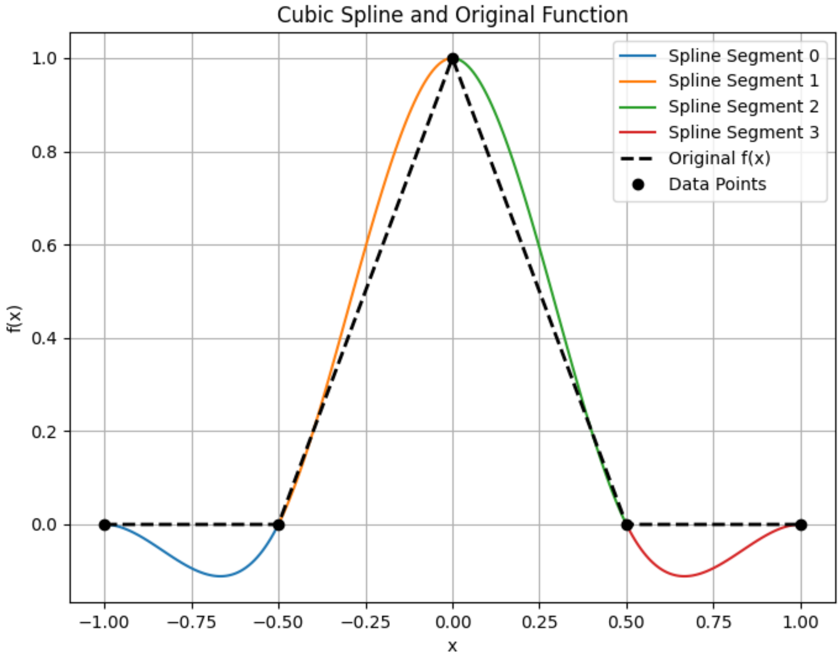
```
A[0,0] = 2*h[0]
A[0,1] = h[0]
b[0] = 6 * ((y_nodes[1] - y_nodes[0]) / h[0] - 0)

A[-1,-2] = h[-1]
A[-1,-1] = 2*h[-1]
b[-1] = 6 * (0 - (y_nodes[-1] - y_nodes[-2]) / h[-1])

for i in range(1, n-1):
    A[i, i-1] = h[i-1]
    A[i, i] = 2*(h[i-1] + h[i])
    A[i, i+1] = h[i]
    b[i] = 6*((y_nodes[i+1] - y_nodes[i]) / h[i] - (y_nodes[i] - y_nodes[i-1]) / h[i-1])

S = np.linalg.solve(A, b)
```

Once we have all the S value, we can use it to solve all the coefficients for each segment

Solve the coefficients in f_i(x) = ai (x - xi)^3 + bi (x-xi)^2 + ci(x-xi) + di for each segment.

```python
def compute_segment_coeffs(x_nodes, y_nodes, S):
    n = len(x_nodes)
    a, b, c, d = [], [], [], []
    for i in range(n - 1):
        h_i = x_nodes[i+1] - x_nodes[i]
        a.append((S[i+1] - S[i]) / (6 * h_i))
        b.append(S[i] / 2)
        c.append((y_nodes[i+1] - y_nodes[i]) / h_i - (2 * h_i * S[i] + h_i * S[i+1]) / 6)
        d.append(y_nodes[i])
    return a, b, c, d
```

Plot the spline curve together with f(x)



3.

Use the equation below to compute the polynomial for each segment

$$\mathbf{P}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Construct x, y and the matrices m1, m2 for the above equation

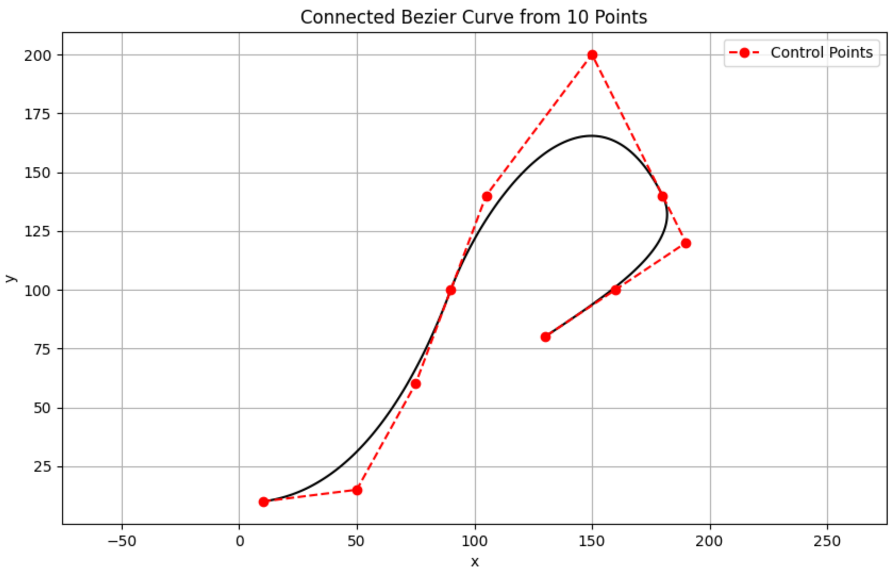
```python
x = np.array([10, 50, 75, 90, 105, 150, 180, 190, 160, 130])
y = np.array([10, 15, 60, 100, 140, 200, 140, 120, 100, 80])

m1 = np.array([
    [2, -2, 1, 1],
    [-3, 3, -2, -1],
    [0, 0, 1, 0],
    [1, 0, 0, 0]
])
m2 = np.array([
    [1, 0, 0, 0],
    [0, 0, 0, 1],
    [-3, 3, 0, 0],
    [0, 0, -3, 3]
])
```

Use the first four points (p0~p3) to compute the first segment and we will reuse point p3 for the second segment. Therefore, I add 3 to the index for each loop. I call the function Bezier_curve for each segment to compute the coefficient using the matrices m1, m2. u_vals contains 100 values evenly between 0 and 1 for plotting each curve.

```
while idx + 3 < len(x):
    ctrl_x = x[idx:idx+4]
    ctrl_y = y[idx:idx+4]

    bx, by = bezier_curve(ctrl_x, ctrl_y, m1, m2, u_vals)
    plt.plot(bx, by, 'black')

    idx += 3
```

```
def bezier_curve(ctrl_x, ctrl_y, m1, m2, u_vals):
    M = m1 @ m2
    coeff_x = M @ ctrl_x.T
    coeff_y = M @ ctrl_y.T

    curve_x = np.polyval(coeff_x, u_vals)
    curve_y = np.polyval(coeff_y, u_vals)
    print(f'coeff_x: {coeff_x}, coeff_y: {coeff_y}')
    return curve_x, curve_y
```

(a) Draw the graph determined by the ten points.



(b)

Since the slope between point 2 and point 3 is the same as the slope between point 3 and point 4. i.e. the points [p2, p3, p4] are colinear. Also, the points [p5, p6, p7] have the same property. Therefore, the curve is C1 continuity, it smoothly connected at points 3 and 6

3 (c)

We know that the Bezier curve for each segment is compute like this.

$$B(u) = \begin{bmatrix}(1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3\end{bmatrix}\begin{bmatrix}p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3}\end{bmatrix} = p_i\cdot(1-u)^3 + p_{i+1}\cdot 3u(1-u)^2 + 3u^2(1-u) + u^3$$

for each segment $u \in [0,1]$

for the first segment we don't have to modify the equation

For the second segment

$u = t-1$ for $t \in [1,2]$

The modified equation $B_2(t) = B(u=t-1) = \begin{bmatrix}(2-t)^3 \\ 3(t-1)(2-t)^2 \\ 3(t-1)^2(2-t) \\ (t-1)^3\end{bmatrix}\begin{bmatrix}p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3}\end{bmatrix}$

for $t \in [1,2)$

Similarly, for the third segment

$$B_3(t) = B[u=t-2] = \begin{bmatrix}(3-t)^3 \\ 3(t-2)(3-t)^2 \\ 3(t-2)^2(3-t) \\ (t-2)^3\end{bmatrix}\begin{bmatrix}p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3}\end{bmatrix}$$

for $t \in [2,3]$

(c)

4. Use the equation below to compute each segment of the B-spline curve

$$B_i(u) = \mathbf{u}^T\mathbf{M}\mathbf{p} = \frac{1}{6}\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2}\end{bmatrix}$$

The implementation of the code is similar to the Bezier Curve I implemented above. We need to change the matrix m and the way we compute the coefficient.

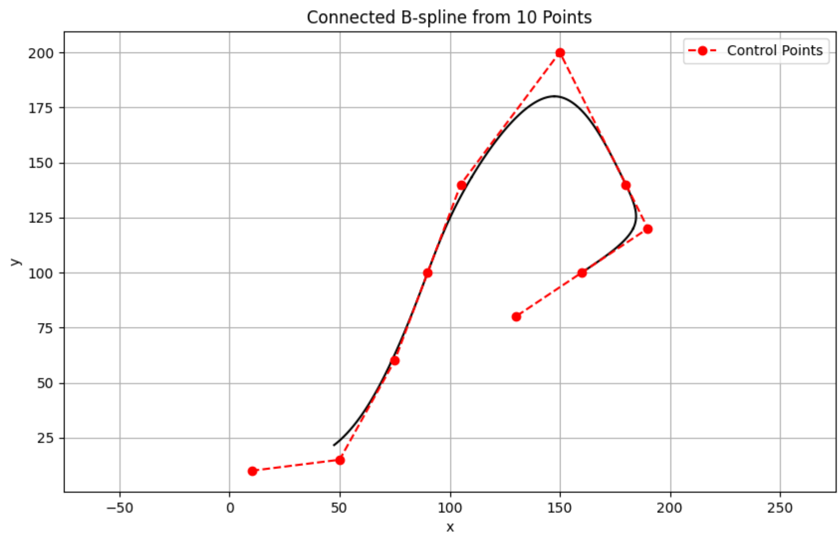
```python
m = np.array([
    [-1, 3, -3, 1],
    [3, -6, 3, 0],
    [-3, 0, 3, 0],
    [1, 4, 1, 0]
])
```

```python
def B_spline(ctrl_x, ctrl_y, m, u_vals):
    coeff_x = (m @ ctrl_x.T) / 6
    coeff_y = (m @ ctrl_y.T) / 6

    curve_x = np.polyval(coeff_x, u_vals)
    curve_y = np.polyval(coeff_y, u_vals)
    print(f'coeff_x: {coeff_x}, coeff_y: {coeff_y}')
    return curve_x, curve_y
```

Also, we need to change the loop we have implemented before since B_spline will share more points between each segments.

```python
idx = 1
while idx + 2 < len(x):
    ctrl_x = x[idx-1:idx+3]
    ctrl_y = y[idx-1:idx+3]

    bx, by = B_spline(ctrl_x, ctrl_y, m, u_vals)
    plt.plot(bx, by, 'black')
    idx += 1
```

(a)



(b)

Since B-spline is C2-continuity, the graph smoothly connected at points 3 and 6.

(c)

4(c) It's the same as what I've done in 3(c)

If we want to move $u \in [0,1]$ to $u \in [1,2]$

Just change all u in the equation to $u-1$

It'll shift the equation right by 1

5.(a)

$$A \quad x = b$$

$$\begin{pmatrix} 0.4 & 0.7 & 1 \\ 1.2 & 2.1 & 1 \\ 3.4 & 4.0 & 1 \\ 4.1 & 4.9 & 1 \\ 5.7 & 6.3 & 1 \\ 7.2 & 8.1 & 1 \\ 9.3 & 8.9 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0.031 \\ 0.933 \\ 3.058 \\ 3.349 \\ 4.870 \\ 5.757 \\ 8.921 \end{pmatrix}$$

Use least-square solution to determine X

$$A^T A \cdot X = A^T b$$

(b)

Solve the above equation to find the least-squares solution

```
A = np.array([[0.4, 0.7, 1],
      [1.2, 2.1, 1],
      [3.4, 4.0, 1],
      [4.1, 4.9, 1],
      [5.7, 6.3, 1],
      [7.2, 8.1, 1],
      [9.3, 8.9, 1],
    ])

B = np.array([0.031, 0.933, 3.058, 3.349, 4.870, 5.757, 8.921])

# Find least squares solution

X = np.linalg.solve(A.T @ A, A.T @ B)
print(f"Least squares solution: {X}")
```

The result:

```
Least squares solution: [ 1.59609218 -0.70238136  0.22066603]
```

The equation: z = 1.59609218 x – 0.70238136 y + 0.22066603

(c)

Find the sum of the least squares of the deviations of the points from the plane

```
Z_fit = A @ X
Z_residuals = B - Z_fit
Z_residuals_sq = Z_residuals**2
Z_residuals_sq_sum = np.sum(Z_residuals_sq)
print(f"Z residuals squared sum: {Z_residuals_sq_sum}")
```

The result:

```
Z residuals squared sum: 0.3193951297516514
```

6

1° $\cos^2(x)$

$$\cos^2(x) \doteq \left(1-\frac{x^2}{2!}+\frac{x^4}{4!}\cdots\right)\left(1-\frac{x^2}{2!}+\frac{x^4}{4!}\cdots\right)$$

$$= 1-x^2+\frac{1}{3}x^4-\frac{2}{45}x^6+\cdots$$

$$= \frac{a_0+a_1x+a_2x^2+a_3x^3}{1+b_1x+b_2x^2+b_3x^3}$$

$\Rightarrow$ $a_0=1$, $b_1=a_1$, $a_2=-1+b_2$, $a_3=-b_1+b_3$

$0=\frac{1}{3}-b_2$, $0=\frac{1}{3}b_1-b_3$, $0=-\frac{2}{45}+\frac{1}{3}b_2$

$\Rightarrow$ $a_0=1$, $a_1=0$, $a_2=-\frac{2}{3}$, $a_3=0$, $b_1=0$, $b_2=\frac{1}{3}$, $b_3=0$

$$\Rightarrow \cos^2(x) \doteq \frac{1-\frac{2}{3}x^2}{1+\frac{1}{3}x^2}$$

2. $\sin(x^4-x)$

$$\sin(x^4-x) \doteq -x+\frac{1}{6}x^3+x^4-\frac{1}{120}x^5 \cdots$$

$$= \frac{a_0+a_1x+a_2x^2+a_3x^3}{1+b_1x+b_2x^2+b_3x^3}$$

$x^0: a_0=0$

$x^1: a_1=-1$

$x^2: a_2=-b_1$

$x^3: a_3=-b_2+\frac{1}{6}$

$x^4: 0=-b_3+\frac{1}{6}b_1+1$

$x^5: 0=\frac{1}{6}b_2+b_1-\frac{1}{120}$

$x^6: 0=\frac{1}{6}b_3+b_2+\frac{1}{120}b_1$

$\Rightarrow$

$a_0=1$

$a_1=-1$

$b_1=\frac{-102}{2153}$

$b_2=\frac{14393}{43060}$

$b_3=\frac{2136}{2153}$

$a_2=\frac{102}{2153}$

$a_3=\frac{-21649}{129180}$

$\Rightarrow$

$$\sin(x^4+x) \doteq \frac{-x+\frac{102}{2153}x^2-\frac{21649}{129180}x^3}{1-\frac{102}{2153}x+\frac{14393}{43060}x^2+\frac{2136}{2153}x^3}$$

3. $$xe^x \doteq x+x^2+\frac{1}{2}x^3+\frac{x^4}{6}+\frac{x^5}{24}+\frac{x^6}{120}\cdots = \frac{a_0+a_1x+a_2x^2+a_3x^3}{1+b_1x+b_2x^2+b_3x^3}$$

$a_0=0$

$a_1=1$

$a_2=1+b_1$

$a_3=\frac{1}{2}+b_1+b_2$

$0=\frac{1}{6}+\frac{1}{2}b_1+b_2+b_3$

$0=\frac{1}{24}+\frac{1}{6}b_1+\frac{1}{2}b_2+b_3$

$0=\frac{1}{120}+\frac{1}{24}b_1+\frac{1}{6}b_2+\frac{1}{2}b_3$

$\Rightarrow$

$a_0=0$

$a_1=1$

$a_2=\frac{3}{5}$

$a_3=\frac{1}{20}$

$b_1=\frac{-1}{6}$

$b_2=\frac{1}{24}$

$b_3=\frac{-1}{120}$

$$\Rightarrow xe^x = \frac{x+\frac{2}{5}x^2+\frac{1}{20}x^3}{1-\frac{3}{5}x+\frac{3}{20}x^2-\frac{1}{60}x^3}$$

7.

(a)

I'll try to find the 8 coefficients for both segments of Hermite curves.

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i+1} \\ x'_i \\ x'_{i+1} \end{bmatrix}$$

It's similar to p3 and p4. First, construct the matrix and the differentiation function we need.

```
m1 = np.array([
    [2, -2, 1, 1],
    [-3, 3, -2, -1],
    [0, 0, 1, 0],
    [1, 0, 0, 0]
])
x = np.array([1, 2, 3])
```

```
def f(x):
    return x*np.exp(-x)
def diff(x):
    return (1-x)*np.exp(-x)
```

Use the control point [p[0], p[1], p'[0], p'[1]] to get the coefficient of the first segment of the Hermite curve. u_vals is used to plot the curve. We can find the second segment using the same way. Note that x'(1), x'(2)and x'(3) should plug in 1 since we let the parametric equation of x = u + 1 and x = u + 2

```
def hermite_interpolation(ctrl_x, ctrl_y, m1, u_vals):
    coeff_x = m1 @ ctrl_x.T
    coeff_y = m1 @ ctrl_y.T

    curve_x = np.polyval(coeff_x, u_vals)
    curve_y = np.polyval(coeff_y, u_vals)
    print(f'coeff_x: {coeff_x}, coeff_y: {coeff_y}')
    return curve_x, curve_y ,coeff_x, coeff_y
```

```
x_input = np.array([1, 2, 1, 1])
y_input = np.array([f(1), f(2), diff(1), diff(2)])
bx, by, cx, cy= hermite_interpolation(x_input, y_input, m1, u_vals)
plt.plot(bx, by, 'black')
```

Find the coefficient of two segments

Eg : the first segment is

x(u) = u + 1   (As we expected)

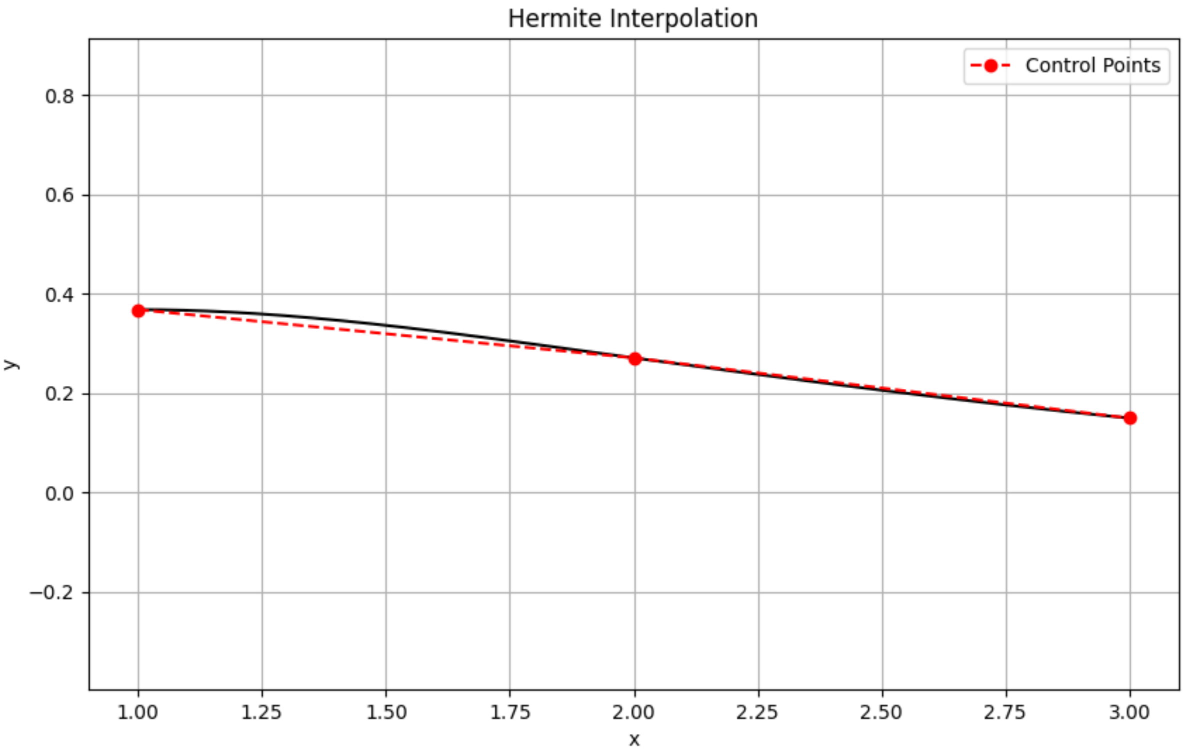
y(u) = 0.059 * u^3 – 0.1563 * u^2 + 0.3679

```
coeff_x: [0 0 1 1], coeff_y: [ 0.05908247 -0.15629134  0.          0.36787944]
```

The second segment:

```
coeff_x: [0 0 1 2], coeff_y: [ 0.0077093   0.00631662 -0.13533528  0.27067057]
```

The Hermite curve:



(b) To find the approximate f(1.5) using Hermite curve, we plug in u = 0.5 to the polynomial of first segment. Since we know that x(0.5) = 1.5 for first segment. The approximate value is close to the true value.

```
The approximate f(1.5) = 0.336191914226910 47
Real value : f(1.5) = 0.33469524022264474
```