[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Python Programming for Numerical Analysis		HW number: #6

6.1. Newton's Polynomial Method

a. 문제

Problem 6.1

The data points in the table lie on the plot of $f(x) = 4.8\cos\dfrac{\pi x}{20}$. Interpolate this data by Newton's method at $x = 0, 0.5, 1.0, \ldots, 8.0$, and compare the results with the "exact" values $y_i = f(x_i)$.

x	0.15	2.30	3.15	4.85	6.25	7.95
y	4.79867	4.49013	4.2243	3.47313	2.66674	1.51909

b. 풀이

sol) 총 6개의 자점이 있으므로 $n = 6-1 = 5$ ⋯ ①

$$
\begin{aligned}
P(x) = {} & a_0 \\
& + a_1(x-x_0) \\
& + a_2(x-x_0)(x-x_1) \\
& + a_3(x-x_0)(x-x_1)(x-x_2) \\
& + a_4(x-x_0)(x-x_1)(x-x_2)(x-x_3) \\
& + a_5(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)
\end{aligned}
$$

- $a_0 = f[x_0] = y_0 = 4.798670$
- $a_1 = f[x_1, x_0] = \dfrac{f[x_1]-f[x_0]}{x_1 - x_0} = -0.143507$
- $a_2 = f[x_2, x_1, x_0] = \dfrac{f[x_2,x_1]-f[x_1,x_0]}{x_2-x_0} = \dfrac{\dfrac{f[x_2]-f[x_1]}{x_2-x_1} - \dfrac{f[x_1]-f[x_0]}{x_1-x_0}}{x_2-x_0} = -0.056411$
- $a_3 = f[x_3, x_2, x_1, x_0] = \dfrac{f[x_3,x_2,x_1]-f[x_2,x_1,x_0]}{x_3-x_0} = \dfrac{\dfrac{f[x_3,x_2]-f[x_2,x_1]}{x_3-x_1} - \dfrac{f[x_2,x_1]-f[x_1,x_0]}{x_2-x_0}}{x_3-x_0} = 0.001229$
- $a_4 = f[x_4, x_3, x_2, x_1, x_0] = \dfrac{f[x_4,x_3,x_2,x_1]-f[x_3,x_2,x_1,x_0]}{x_4-x_0} = 0.000104$
- $a_5 = f[x_5, x_4, x_3, x_2, x_1, x_0] = \dfrac{f[x_5:x_1]-f[x_4:x_0]}{x_5-x_0} = 0.000002$

($x = 0, 0.5, 1, \cdots, 8$ 에서의 입력 결과 및 실제 결과와의 비교는

Python 결과 확인!)

```
Interpolated Values at x = 0, 0.5, ..., 8.0:
  x     P(x)     f(x)       Err
0.0 4.800025 4.800000 -0.000025
0.5 4.785178 4.785203  0.000025
1.0 4.740877 4.740904  0.000027
1.5 4.667361 4.667376  0.000015
2.0 4.565067 4.565071  0.000004
2.5 4.434621 4.434622  0.000001
3.0 4.276829 4.276831  0.000003
3.5 4.092666 4.092673  0.000007
4.0 3.883273 3.883282  0.000009
4.5 3.649941 3.649949  0.000008
5.0 3.394109 3.394113  0.000003
5.5 3.117352 3.117351 -0.000002
6.0 2.821373 2.821369 -0.000004
6.5 2.507994 2.507993 -0.000000
7.0 2.179147 2.179154  0.000007
7.5 1.836868 1.836880  0.000012
8.0 1.483286 1.483282 -0.000004
```

c. 문제

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# 1. Given data points (x_i, y_i)
x_data = np.array([0.15, 2.30, 3.15, 4.85, 6.25, 7.95])
y_data = np.array([4.79867, 4.49013, 4.22430, 3.47313, 2.66674, 1.51909])

# 2. Compute divided difference coefficients for Newton interpolation
def divided_differences(x, y):
    n = len(x)
    coef = y.copy().astype(float)
    for j in range(1, n):
        coef[j:n] = (coef[j:n] - coef[j - 1]) / (x[j:n] - x[j - 1])
    return coef

a = divided_differences(x_data, y_data)

# 3. Generate human-readable Newton interpolation polynomial expression
def interpolation_formula(a, x_data):
    terms = [f"{a[0]:.6f}"]
    for i in range(1, len(a)):
        factors = '*'.join([f"(x - {x_data[j]:.2f})" for j in range(i)])
        terms.append(f"{a[i]:+f}*{factors}")
    return "P(x) = " + ' '.join(terms)

interp_eq = interpolation_formula(a, x_data)
print("Newton Interpolation Formula:")
print(interp_eq)

# 4. Evaluate Newton polynomial at a given x using Horner's method
def newton_eval(a, x_data, x):
    n = len(a)
    result = a[-1]
    for i in range(n - 2, -1, -1):
        result = result * (x - x_data[i]) + a[i]
    return result

# 5. Evaluate at x = 0, 0.5, ..., 8.0 and compare with actual function
x_eval = np.arange(0.0, 8.01, 0.5)
y_eval = [newton_eval(a, x_data, x) for x in x_eval]
real_eval = [4.8 * np.cos(np.pi * x / 20) for x in x_eval]
error_eval = np.array(real_eval) - np.array(y_eval)

# 6. Create table comparing interpolated and actual values
result_df = pd.DataFrame({
    "x": x_eval,
    "P(x)": np.round(y_eval, 6),
    "f(x)": np.round(real_eval, 6),
    "Err": np.round(error_eval, 6)
})
print("\nInterpolated Values at x = 0, 0.5, ..., 8.0:")
print(result_df.to_string(index=False))

# 7. Plot interpolated curve, original data points, and evaluated points
x_dense = np.linspace(0, 8.0, 400)
y_dense = [newton_eval(a, x_data, x) for x in x_dense]
plt.figure(figsize=(10, 6))
plt.plot(x_dense, y_dense, label="Interpolated Curve (P(x))", color='orange')
plt.plot(x_data, y_data, 'ro', label="Original Data Points")
plt.scatter(x_eval, y_eval, color='blue', label="Evaluated Points (step size=0.5)")
plt.xlabel("x")
plt.ylabel("P(x)")
plt.title("5th-order Newton Polynomial Interpolation Graph")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

6.2. Natural Cubic Spline(3차 다항식 보간법)

a. 문제

Problem 6.2

Use a natural cubic spline to determine y at $x = 1.5$. The data points are

x	1	2	3	4	5
y	0	1	0	1	0

b. 풀이

sol.) • natural cubic spline 조건 : $S_1''(x_1) = 0$, $S_4''(x_5) = 0$

$S_1(x)$: $a_1x^3 + b_1x^2 + c_1x + d_1$ | $S_2(x)$: $a_2x^3 + b_2x^2 + c_2x + d_2$ | $S_3(x)$: $a_3x^3 + b_3x^2 + c_3x + d_3$ | $S_4(x)$: $a_4x^3 + b_4x^2 + c_4x + d_4$ ⇒ "계수 16개 구하기"

0 (1), 1 (2), 0 (3), 1 (4), 0 (5)

① 경계 지점에서 함수값 일치
- $S_1(1) = 0$, $S_1(2) = 1$
- $S_2(2) = 1$, $S_2(3) = 0$
- $S_3(3) = 0$, $S_3(4) = 1$
- $S_4(4) = 1$, $S_4(5) = 0$

② 1차 도함수 연속성
- $S_1'(2) = S_2'(2)$
- $S_2'(3) = S_3'(3)$
- $S_3'(4) = S_4'(4)$

③ 2차 도함수 연속
- $S_1''(2) = S_2''(2)$
- $S_2''(3) = S_3''(3)$
- $S_3''(4) = S_4''(4)$

④ Natural Boundry Condition
- $S_1''(1) = 0$
- $S_4''(5) = 0$

$\Rightarrow y(1.5) = S_1(1.5) = a_1(1.5)^3 + b_1(1.5)^2 + c_1(1.5) + d_1$ 구하자. $= 5.58928$

[a_1, b_1, c_1, d_1 구하는 과정]

① 16개 방정식 목록화

② 행렬로 정리

③ 가우스 소거법 사용하여 $a_1 \sim d_1$ 구하기

(계산 과정은 Python code 참고!)

$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{bmatrix} = \begin{bmatrix} -0.714286 \\ 2.142857 \\ -0.428571 \\ -1.000000 \end{bmatrix}$$

$\boxed{\text{답}: 5.58928}$

c. Python Code

```python
import numpy as np

# 보간점
x_vals = [1, 2, 3, 4, 5]
y_vals = [0, 1, 0, 1, 0]

# 방정식 개수 = 16 (4구간 x 4계수)
A = np.zeros((16, 16))
b = np.zeros(16)

# Eq 1~8: 각 구간의 양 끝점에서의 y값 (함수값 일치)
row = 0
for i in range(4):
    x0 = x_vals[i]
    x1 = x_vals[i+1]

    # Si(xi) = yi
    A[row, i*4:i*4+4] = [x0**3, x0**2, x0, 1]
    b[row] = y_vals[i]
    row += 1

    # Si(xi+1) = yi+1
    A[row, i*4:i*4+4] = [x1**3, x1**2, x1, 1]
    b[row] = y_vals[i+1]
    row += 1

# Eq 9~11: C1 연속성 (도함수 연속)
for i in range(3):  # between S1-S2, S2-S3, S3-S4
    x = x_vals[i+1]
    A[row, i*4:i*4+4] = [3*x**2, 2*x, 1, 0]          # S_i'
    A[row, (i+1)*4:(i+1)*4+4] = [-3*x**2, -2*x, -1, 0]  # -S_{i+1}'
    b[row] = 0
    row += 1

# Eq 12~14: C2 연속성 (이차 도함수 연속)
for i in range(3):
    x = x_vals[i+1]
    A[row, i*4:i*4+4] = [6*x, 2, 0, 0]               # S_i''
    A[row, (i+1)*4:(i+1)*4+4] = [-6*x, -2, 0, 0]     # -S_{i+1}''
    b[row] = 0
    row += 1

# Eq 15~16: Natural boundary condition
# S1''(x1) = 0
A[row, 0:4] = [6*x_vals[0], 2, 0, 0]
b[row] = 0
row += 1

# S4''(x5) = 0
A[row, 12:16] = [6*x_vals[-1], 2, 0, 0]
b[row] = 0
# 가우스 소거법으로 해 풀기
def gauss_elimination(A, b):
    n = len(b)
    M = np.hstack([A.astype(float), b.reshape(-1,1)])

    for k in range(n):
        # Pivoting
        max_row = np.argmax(abs(M[k:, k])) + k
        M[[k, max_row]] = M[[max_row, k]]

        # Elimination
        for i in range(k+1, n):
            factor = M[i, k] / M[k, k]
            M[i, k:] = M[i, k:] - factor * M[k, k:]

    # Back substitution
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (M[i, -1] - np.dot(M[i, i+1:n], x[i+1:n])) / M[i, i]

    return x

coeffs = gauss_elimination(A, b)

# 결과 출력 (특히 a1~d1만 확인)
print("Coefficients a1~d1:")
print("a1 =", coeffs[0])
print("b1 =", coeffs[1])
print("c1 =", coeffs[2])
print("d1 =", coeffs[3])
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
_6\hw6_2.py" "
Coefficients a1~d1:
a1 = -0.714285714285718
b1 = 2.1428571428571583
c1 = -0.4285714285714483
d1 = -1.0000000000000002
```

6.3. Lagrange Polynomial Interpolation

a. 문제

Problem 6.3

Density of air ρ varies with elevation h in the following manner:

h (km)	0	3	6
ρ (kg/m^3)	1.225	0.905	0.652

Express $\rho(h)$ as a quadratic function using Lagrange's method.

b. 풀이

sol) $\rho(h) = 1.225 \cdot P_0(h) + 0.905 \cdot P_3(h) + 0.652 \cdot P_6(h)$

- $P_0(h) = \dfrac{(h-3)(h-6)}{(0-3)(0-6)} = \dfrac{(h-3)(h-6)}{18}$
- $P_3(h) = \dfrac{(h-0)(h-6)}{(3-0)(3-6)} = \dfrac{h(h-6)}{-9}$
- $P_6(h) = \dfrac{(h-0)(h-3)}{(6-0)(6-3)} = \dfrac{h(h-3)}{18}$

$$= \frac{1.225}{18}(h-3)(h-6) - \frac{0.905}{9}h(h-6) + \frac{0.652}{18}h(h-3)$$

$$\fallingdotseq \boxed{0.00372h^2 - 0.11783h + 1.225}$$

6.4. Polynomial Interpolation via Linear System

a. 문제

Problem 6.4 *(Interpolation via solving a linear system)*

Polynomial Interpolation via Linear System Let $f(x)$ be a polynomial of degree at most 2 (i.e., quadratic) that passes through the following three points:

$$(1,2), \quad (2,3), \quad (4,1)$$

(a) Assume $f(x) = ax^2 + bx + c$. Write down a system of linear equations based on the interpolation condition $f(x_i) = y_i$ for each given point.

(b) Write the above system in matrix form $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, and solve for $\mathbf{x}$.

(c) Write the explicit form of the interpolating polynomial $f(x)$.

(d) Plot the polynomial $f(x)$ and verify visually that it passes through the given three points.

b. 풀이

sol) (a)

$$\begin{cases} a(1)^2+b(1)+c=2 \\ a(2)^2+b(2)+c=3 \\ a(4)^2+b(4)+c=1 \end{cases} \Rightarrow \begin{cases} a+b+c=2 \\ 4a+2b+c=3 \\ 16a+4b+c=1 \end{cases}$$

(b)

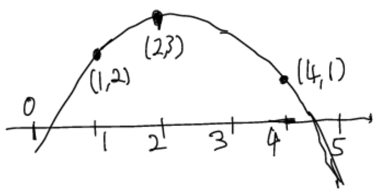
$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 16 & 4 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}}_{b}$$

(c)

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 4 & 2 & 1 & 3 \\ 16 & 4 & 1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & -2 & -3 & -5 \\ 0 & -12 & -15 & -31 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & -2 & -3 & -5 \\ 0 & 0 & 3 & -1 \end{array}\right] \therefore \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -2/3 \\ 3 \\ -1/3 \end{bmatrix}$$

$$\therefore f(x) = -\frac{2}{3}x^2 + 3x - \frac{1}{3}$$

(d)



c. Python Code

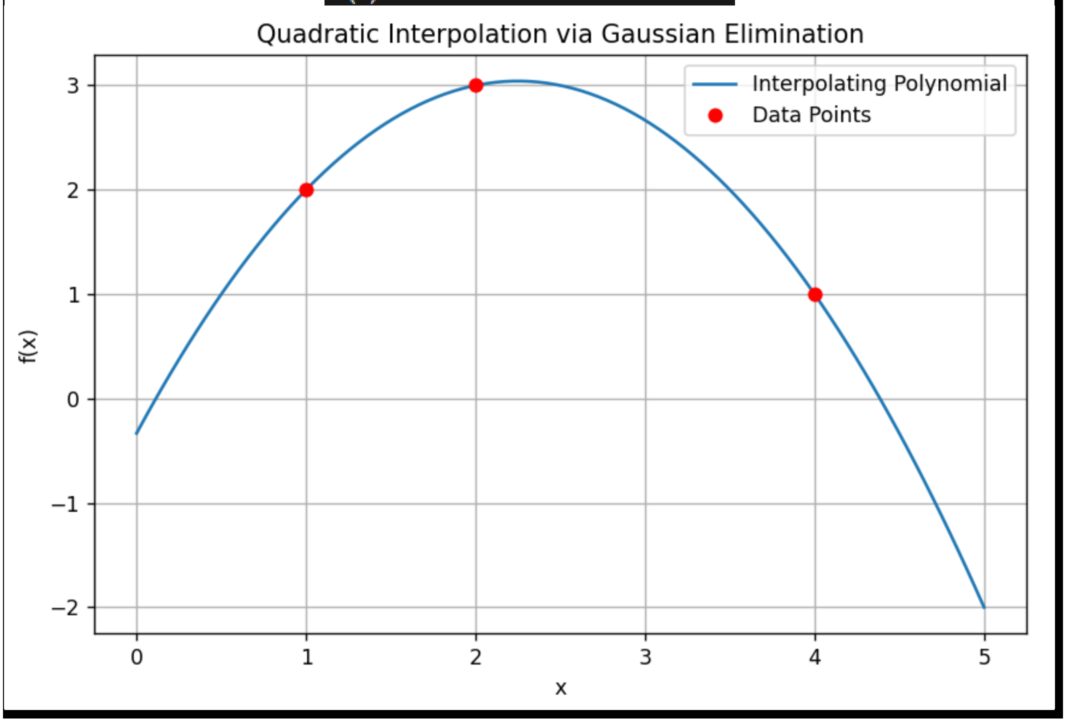
```python
import numpy as np
import matplotlib.pyplot as plt

def gaussian_elimination(A, b):
    A = A.copy().astype(float)
    b = b.copy().astype(float)
    n = len(b)

    # Forward elimination
    for i in range(n):
        # Pivoting
        max_row = np.argmax(np.abs(A[i:, i])) + i
        if i != max_row:
            A[[i, max_row]] = A[[max_row, i]]
            b[[i, max_row]] = b[[max_row, i]]

        # Eliminate below
        for j in range(i+1, n):
            factor = A[j, i] / A[i, i]
            A[j, i:] -= factor * A[i, i:]
            b[j] -= factor * b[i]

    # Back substitution
    x = np.zeros(n)
    for i in reversed(range(n)):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]

    return x

# 행렬 A와 벡터 b 정의
A = np.array([
    [1, 1, 1],
    [4, 2, 1],
    [16, 4, 1]
], dtype=float)

b = np.array([2, 3, 1], dtype=float)

# 가우스 소거법으로 선형 시스템 Ax = b 풀기
x = gaussian_elimination(A, b)

a, b_, c = x
print(f"f(x) = {a:.3f}x^2 + {b_:.3f}x + {c:.3f}")

def f(x):
    return a * x**2 + b_ * x + c

x_vals = np.linspace(0, 5, 100)
y_vals = f(x_vals)

# 원래 주어진 점들
x_data = [1, 2, 4]
y_data = [2, 3, 1]

plt.figure(figsize=(8, 5))
plt.plot(x_vals, y_vals, label='Interpolating Polynomial')
plt.plot(x_data, y_data, 'ro', label='Data Points')
plt.title('Quadratic Interpolation via Gaussian Elimination')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
_6\hw6_4.py" "
f(x) = -0.667x^2 + 3.000x + -0.333
```



6.5. Prediction via Polynomial Interploation

a. 문제

Problem 6.5 *(Prediction via polynomial interpolation)* This method is also applicable to beam deflection profiles, blade surface fitting, and trajectory reconstruction in robotic arms. When more data points are available, higher-order or piecewise polynomial interpolation (e.g., splines) may provide better robustness.

Problem Description An aerospace engineer is analyzing the surface profile of an airfoil. Due to limitations in measurement equipment, only a few height measurements are available along the chord line (length) of the airfoil. The surface is assumed to be smooth and can be approximated by a quartic polynomial:

$$y(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

The following data points (in meters) represent measurements of the airfoil surface at different chord positions:

x (m)	y (m)
0.00	0.00
0.25	0.03
0.50	0.04
0.75	0.03
1.00	0.00

Tasks

(a) Formulate a system of linear equations using the interpolation conditions, and solve for the coefficients a_0, a_1, a_2, a_3, a_4.

(b) Write the explicit form of the polynomial $y(x)$.

(c) Estimate the surface height at $x = 0.6\,\text{m}$. Briefly discuss whether the resulting profile exhibits leading/trailing edge symmetry.

(d) Plot the resulting interpolated polynomial together with the original data points. Use software such as Python/Matlab/Mathematica for visualization.

b. 풀이

(a) $y(0) = 0 \rightarrow a_4 \cdot 0^4 + a_3 \cdot 0^3 + a_2 \cdot 0^2 + a_1 \cdot 0 + a_0 = 0$

$y(0.25) = 0.03 \rightarrow a_4 \cdot (0.25)^4 + a_3 \cdot (0.25)^3 + a_2 \cdot (0.25)^2 + a_1 \cdot (0.25) + a_0 = 0.03$

$y(0.5) = 0.04 \rightarrow a_4 (0.5)^4 + a_3 \cdot (0.5)^3 + a_2 (0.5)^2 + a_1 (0.5) + a_0 = 0.04$

$y(0.75) = 0.03 \rightarrow a_4 (0.75)^4 + a_3 (0.75)^3 + a_2 (0.75)^2 + a_1 (0.75) + a_0 = 0.03$

$y(1) = 0 \rightarrow a_4 + a_3 + a_2 + a_1 + a_0 = 0$

$$\therefore Ax = b \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0.25^4 & 0.25^3 & 0.25^2 & 0.25 & 1 \\ 0.5^4 & 0.5^3 & 0.5^2 & 0.5 & 1 \\ 0.75^4 & 0.75^3 & 0.75^2 & 0.75 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.03 \\ 0.04 \\ 0.03 \\ 0 \end{bmatrix}$$

(b) 가우스 소거법 사용(계산: Python 코드 참고.)

$$\Rightarrow \begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -0.16 \\ 0.16 \\ 0 \end{bmatrix} \; ; \; y(x) = -0.16x^2 + 0.16x$$

(c) $y(0.6) = -0.16(0.6^2 - 0.6)$

$= 0.0384$

(d) Python Code 실행결과 확인!

c. Python Code

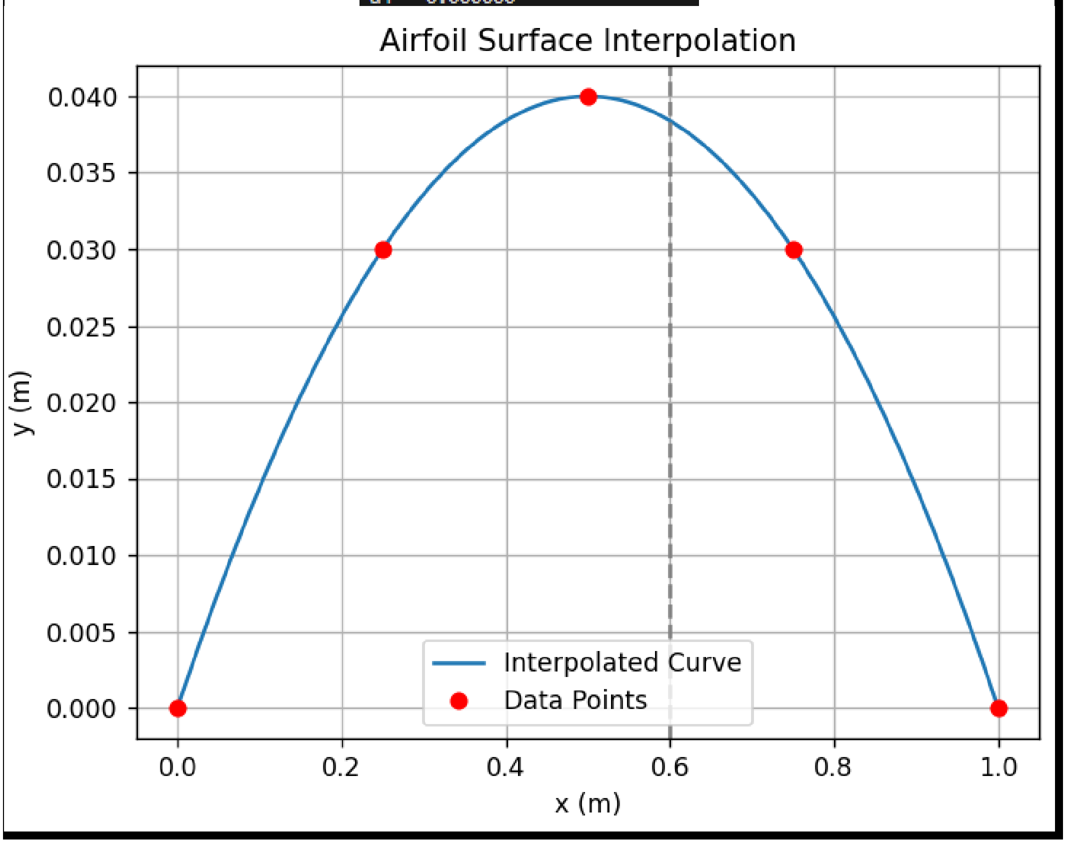
```python
import numpy as np
import matplotlib.pyplot as plt

# Points
x_vals = np.array([0.00, 0.25, 0.50, 0.75, 1.00])
y_vals = np.array([0.00, 0.03, 0.04, 0.03, 0.00])

# Vandermonde
A = np.vander(x_vals, 5)
b = y_vals.copy()

# Augmented matrix [A | b]
Ab = np.hstack((A, b.reshape(-1, 1)))
n = len(b)

# Gauss elimination
def gauss_elimination(Ab):
    # Forward Elimination
    for i in range(n):
        # Pivoting
        max_row = i + np.argmax(np.abs(Ab[i:, i]))
        Ab[[i, max_row]] = Ab[[max_row, i]]

        # Elimination Phase
        for j in range(i+1, n):
            factor = Ab[j, i] / Ab[i, i]
            Ab[j, i:] -= factor * Ab[i, i:]

    # Back Substitution
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (Ab[i, -1] - np.dot(Ab[i, i+1:n], x[i+1:n])) / Ab[i, i]
    return x

# Coefficients
coeffs = gauss_elimination(Ab)

# Result
print("Coefficients (a4 ~ a0):")
for i, a in enumerate(coeffs[::-1]):
    print(f"a{i} = {a:.6f}")


x_data = np.linspace(0, 1, 200)
y_data = np.polyval(coeffs, x_data)

plt.plot(x_data, y_data, label="Interpolated Curve")
plt.plot(x_vals, y_vals, 'ro', label="Data Points")
plt.axvline(0.6, linestyle='--', color='gray')
plt.title("Airfoil Surface Interpolation")
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.grid(True)
plt.legend()
plt.show()
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
Coefficients (a4 ~ a0):
a0 = 0.000000
a1 = 0.160000
a2 = -0.160000
a3 = -0.000000
a4 = 0.000000
```



6.6. (Least Square Method 적용 5) 다변수

a. 문제

b. 풀이

(a), (b)

(a),(b) 1번째 방법 사용:

	$x_0(t)$	$x_1(t)$	$x_2(t)$	$x_3(t)$
	$a_{00}t^3+b_{00}t^2+c_{00}t+d_{00}$	$a_{01}t^3+b_{01}t^2+c_{01}t+d_{01}$	$a_{02}t^3+b_{02}t^2+c_{02}t+d_{02}$	$a_{03}t^3+b_{03}t^2+c_{03}t+d_{03}$
	$y_0(t)$	$y_1(t)$	$y_2(t)$	$y_3(t)$
	$a_{10}t^3+b_{10}t^2+c_{10}t+d_{10}$	$a_{11}t^3+b_{11}t^2+c_{11}t+d_{11}$	$a_{12}t^3+b_{12}t^2+c_{12}t+d_{12}$	$a_{13}t^3+b_{13}t^2+c_{13}t+d_{13}$

$t =$ 0, 0.25, 0.5, 0.75, 1

[Interpolation 조건]

① 위치 연속성
- $x_0(0)=0,\ x_0(0.25)=1$
- $x_1(0.25)=1,\ x_1(0.5)=2.5$
- $x_2(0.5)=2.5,\ x_2(0.75)=4$
- $x_3(0.75)=4,\ x_3(1)=5$
- $y_0(0)=0,\ y(0.25)=2$
- $y_1(0.25)=2,\ y_1(0.5)=1$
- $y_2(0.5)=1,\ y_2(0.75)=2$
- $y_3(0.75)=2,\ y_3(1)=0$

② 속도 연속성
- $x_0'(0.25)=x_1'(0.25)$
- $x_1'(0.5)=x_2'(0.5)$
- $x_2'(0.75)=x_3'(0.75)$
- $y_0'(0.25)=y_1'(0.25)$
- $y_1'(0.5)=y_2'(0.5)$
- $y_2'(0.75)=y_3'(0.75)$

③ 가속도 연속성
- $x_0''(0.25)=x_1''(0.25)$
- $x_1''(0.5)=x_2''(0.5)$
- $x_2''(0.75)=x_3''(0.75)$
- $y_0''(0.25)=y_1''(0.25)$
- $y_1''(0.5)=y_2''(0.5)$
- $y_2''(0.75)=y_3''(0.75)$

④ Natural Boundary Condition
- $x''(0)=0,\ x''(1)=0$
- $y''(0)=0,\ y''(1)=0$

(이어서) 계산과정은 Python 코드 참고할 것.

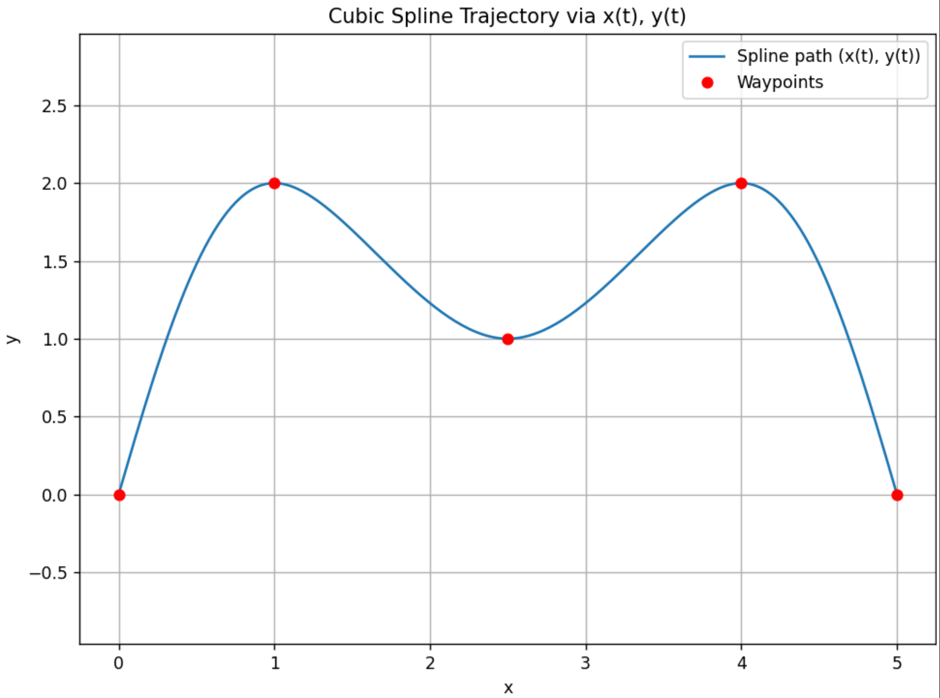
답:

t	$x_i(t)$	$y_i(t)$
[0.0, 0.25)	$8t^3+3.5t$	$-64t^3+12t$
[0.25, 0.5)	$-8(t-0.25)^3+6(t-0.25)^2+5(t-0.25)+1$	$128(t-0.25)^3-48(t-0.25)^2+2$
[0.5, 0.75)	$-8(t-0.5)^3+6.5(t-0.5)+2.5$	$-128(t-0.5)^3+48(t-0.5)^2+1$
[0.75, 1)	$8(t-0.75)^3-6(t-0.75)^2+5(t-0.75)+4$	$64(t-0.75)^3-48(t-0.75)^2+2$

(c)

t	x(t)	y(t)
0.0	0.000	0.000
0.1	0.358	1.136
0.2	0.764	1.888
0.3	1.264	1.896
0.4	1.858	1.352
0.5	2.500	1.000
0.6	3.142	1.352
0.7	3.736	1.896
0.8	4.236	1.888
0.9	4.642	1.136
1.0	5.000	0.000

(c) $t=0, 0.1, 0.2, \cdots, 0.9, 1$ 에 대한 $x(t), y(t)$ 는 (a),(b) 에서 구한 $x_i(t), y_i(t)$ 에 각각 대입하여 구할 것! 계산과정 및 결과는 Python 코드 참고!

(d)	
(e)	1) Cubic Spline의 장점: 왜 경로가 부드러운가? A: Cubic Spline은 수학적으로 다음의 조건을 만족하기 때문: - 위치 연속성(C^0): 경로가 끊기지 않고 연결된다. - 속도 연속성(C^1): 로봇이 방향을 급격히 바꾸지 않고, 매끄럽게 회전한다. - 가속도 연속성(C^2): 로봇이 가속/감속할 때 충격 없이 부드럽게 움직인다.
	2) 하지만 장애물이 있는 환경에서는: A. Cubic Spline은 기본적으로 모든 점을 반드시 통과해야 하므로: - 전역적 최적화 불가능: Spline은 구간 단위로 국소적으로만 조정되므로, 전체 맥락에서 최적 경로를 고려하지 않는다. - 유연성 부족: 경로 수정이 필요할 때 모든 spline을 다시 구해야 한다.

c. Python Code

```python
import numpy as np
import pandas as pd

# 구간 정의
segments = [
    (0.00, 0.25,  # Segment 0
     [8.0, 0.0, 3.5, 0.0],      # x(t) 계수 a,b,c,d
     [-64.0, 0.0, 12.0, 0.0]), # y(t) 계수 a,b,c,d

    (0.25, 0.50,  # Segment 1
     [-8.0, 6.0, 5.0, 1.0],
     [128.0, -48.0, 0.0, 2.0]),

    (0.50, 0.75,  # Segment 2
     [-8.0, 0.0, 6.5, 2.5],
     [-128.0, 48.0, 0.0, 1.0]),

    (0.75, 1.00,  # Segment 3
     [8.0, -6.0, 5.0, 4.0],
     [64.0, -48.0, 0.0, 2.0])
]

# 평가할 t 값
t_values = np.round(np.arange(0.0, 1.01, 0.1), 2)
x_results, y_results = [], []

# 각 t에 대해 해당 구간을 찾아 계산
for t in t_values:
    for (t_start, t_end, x_coeffs, y_coeffs) in segments:
        if t_start <= t <= t_end:
            tau = t - t_start
            a, b, c, d = x_coeffs
            e, f, g, h = y_coeffs
            x_t = a * tau**3 + b * tau**2 + c * tau + d
            y_t = e * tau**3 + f * tau**2 + g * tau + h
            x_results.append(round(x_t, 6))
            y_results.append(round(y_t, 6))
            break

# 결과 출력
df = pd.DataFrame({
    "t": t_values,
    "x(t)": x_results,
    "y(t)": y_results
})

print(df.to_string(index=False))
```