

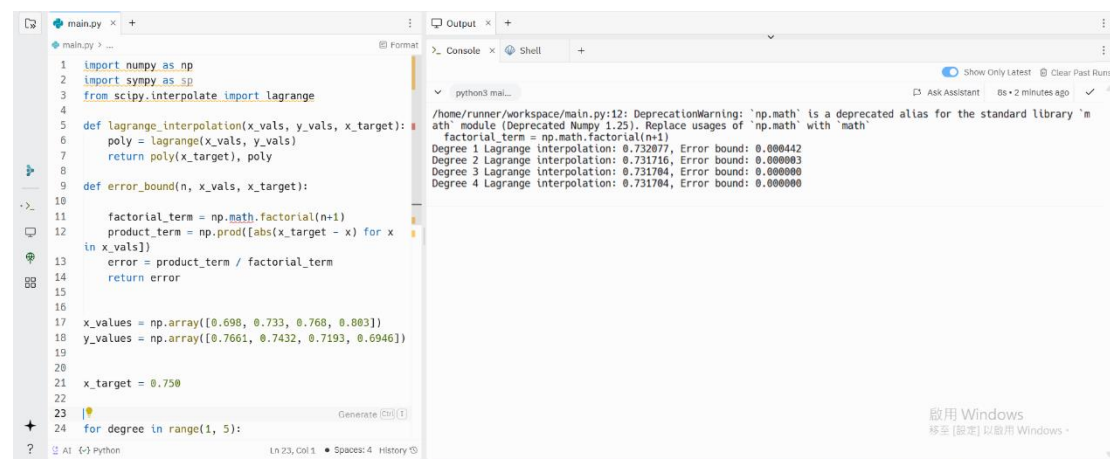
Q1

Degree 1 Lagrange interpolation: 0.732077, Error bound: 0.000442

Degree 2 Lagrange interpolation: 0.731716, Error bound: 0.000003

Degree 3 Lagrange interpolation: 0.731704, Error bound: 0.000000

Degree 4 Lagrange interpolation: 0.731704, Error bound: 0.000000



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an output console on the right. The code in the editor defines a function `lagrange_interpolation` and an `error_bound` function. It then defines `x_values` and `y_values` arrays, and a target `x_target` value. Finally, it iterates over degrees 1 to 5, printing the interpolation result and error bound for each degree. The output console shows the results of these calculations, including a deprecation warning for `np.math`.

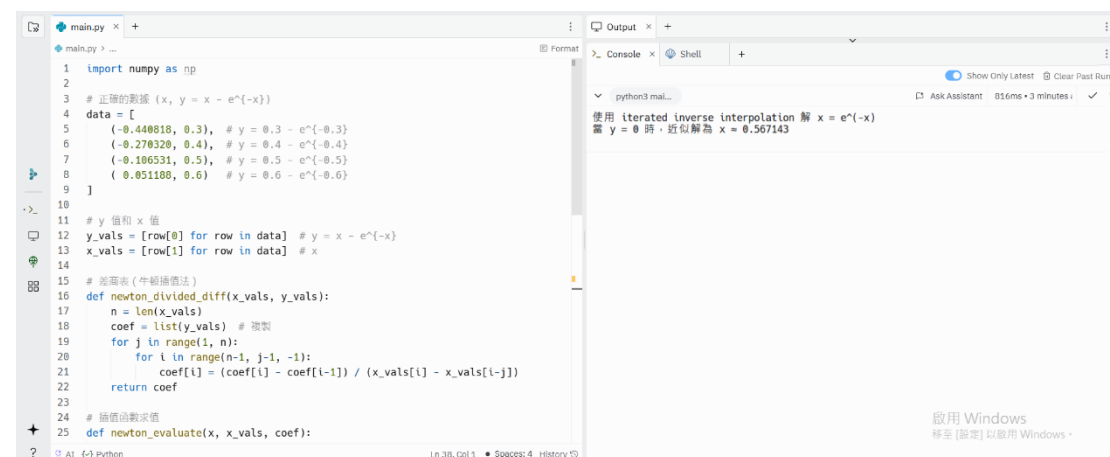
```
1 import numpy as np
2 import sympy as sp
3 from scipy.interpolate import lagrange
4
5 def lagrange_interpolation(x_vals, y_vals, x_target):
6     poly = lagrange(x_vals, y_vals)
7     return poly(x_target), poly
8
9 def error_bound(n, x_vals, x_target):
10
11     factorial_term = np.math.factorial(n-1)
12     product_term = np.prod([abs(x_target - x) for x
13         in x_vals])
14     error = product_term / factorial_term
15     return error
16
17 x_values = np.array([0.698, 0.733, 0.768, 0.803])
18 y_values = np.array([0.7661, 0.7432, 0.7193, 0.6946])
19
20
21 x_target = 0.750
22
23 for degree in range(1, 5):
24     result, poly = lagrange_interpolation(x_values, y_values, x_target)
25     error_bound = error_bound(degree, x_values, x_target)
26     print(f'Degree {degree} Lagrange interpolation: {result}, Error bound: {error_bound}')
```

Output console shows:

```
/home/runner/workspace/main.py:12: DeprecationWarning: 'np.math' is a deprecated alias for the standard library 'math' module (Deprecated Numpy 1.25). Replace usages of 'np.math' with 'math'
factorial_term = np.math.factorial(n-1)
Degree 1 Lagrange interpolation: 0.732077, Error bound: 0.000442
Degree 2 Lagrange interpolation: 0.731716, Error bound: 0.000003
Degree 3 Lagrange interpolation: 0.731704, Error bound: 0.000000
Degree 4 Lagrange interpolation: 0.731704, Error bound: 0.000000
```

Q2

當 $y = 0$ 時，近似解為 $x \approx 0.567143$



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an output console on the right. The code in the editor defines a function `newton_divided_diff` and a function `newton_evaluate`. It then defines a list of data points and uses the Newton-Raphson method to find the root of the function $y = x - e^{-x}$ when $y = 0$. The output console shows the result of the calculation.

```
1 import numpy as np
2
3 # 正確的數據 (x, y = x - e^{-x})
4 data = [
5     (-0.446818, 0.3), # y = 0.3 - e^{-0.3}
6     (-0.270320, 0.4), # y = 0.4 - e^{-0.4}
7     (-0.106531, 0.5), # y = 0.5 - e^{-0.5}
8     (0.051108, 0.6) # y = 0.6 - e^{-0.6}
9 ]
10
11 # y 值和 x 值
12 y_vals = [row[0] for row in data] # y = x - e^{-x}
13 x_vals = [row[1] for row in data] # x
14
15 # 差商表 (牛頓插值法)
16 def newton_divided_diff(x_vals, y_vals):
17     n = len(x_vals)
18     coef = list(y_vals) # 複製
19     for j in range(1, n):
20         for i in range(n-1, j-1, -1):
21             coef[i] = (coef[i] - coef[i-1]) / (x_vals[i] - x_vals[i-j])
22     return coef
23
24 # 插值函數求值
25 def newton_evaluate(x, x_vals, coef):
26     n = len(x_vals)
27     result = coef[0]
28     for i in range(1, n):
29         result = result + coef[i] * (x - x_vals[i-1])
30     return result
```

Output console shows:

```
使用 iterated inverse interpolation 解 x = e^{-x}
當 y = 0 時，近似解為 x = 0.567143
```

Q3

(a)

位置: 768.96 feet

速度: 74.64 ft/s

(b)

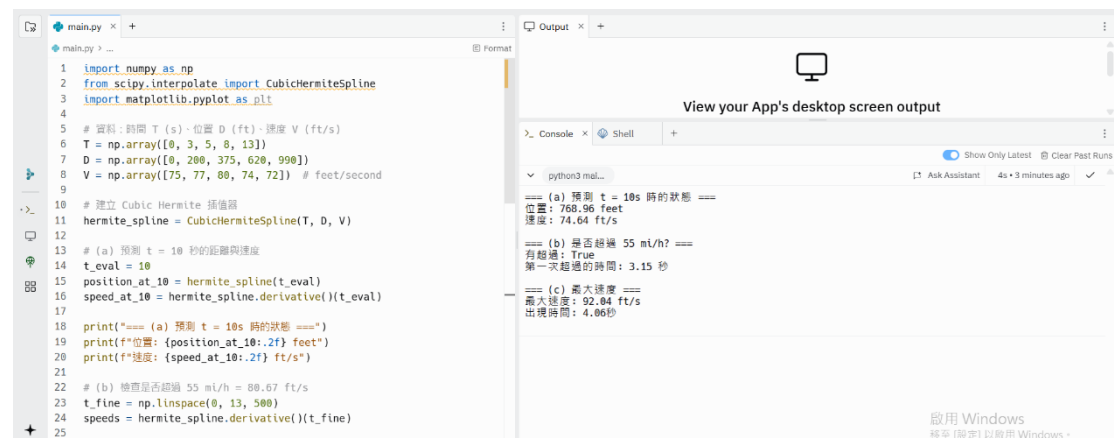
有超過 55mi/h

第一次超過的時間: 3.15 秒

(c)

最大速度: 92.04 ft/s

出現時間: 4.06 秒



The screenshot shows a Python IDE with a file named `main.py`. The code defines time `T` and distance `D` arrays, and velocity `V` array. It uses `CubicHermiteSpline` to create a spline model. Part (a) calculates position and speed at `t = 10` seconds. Part (b) checks if the speed exceeds 55 mi/h and finds the first time it does. Part (c) finds the maximum speed and the time it occurs.

```
1 import numpy as np
2 from scipy.interpolate import CubicHermiteSpline
3 import matplotlib.pyplot as plt
4
5 # 資料: 時間 T (s) - 位置 D (ft) - 速度 V (ft/s)
6 T = np.array([0, 3, 5, 8, 13])
7 D = np.array([0, 200, 375, 620, 990])
8 V = np.array([75, 77, 80, 74, 72]) # feet/second
9
10 # 建立 Cubic Hermite 插值函數
11 hermite_spline = CubicHermiteSpline(T, D, V)
12
13 # (a) 預測 t = 10 秒的距離與速度
14 t_eval = 10
15 position_at_10 = hermite_spline(t_eval)
16 speed_at_10 = hermite_spline.derivative()(t_eval)
17
18 print("=== (a) 預測 t = 10s 時的狀態 ===")
19 print(f"位置: {position_at_10:.2f} feet")
20 print(f"速度: {speed_at_10:.2f} ft/s")
21
22 # (b) 檢查是否超過 55 mi/h = 80.67 ft/s
23 t_fine = np.linspace(0, 13, 500)
24 speeds = hermite_spline.derivative()(t_fine)
25
```

The output window shows the following results:

```
=== (a) 預測 t = 10s 時的狀態 ===
位置: 768.96 feet
速度: 74.64 ft/s

=== (b) 是否超過 55 mi/h? ===
有超過: True
第一次超過的時間: 3.15 秒

=== (c) 最大速度 ===
最大速度: 92.04 ft/s
出現時間: 4.06 秒
```