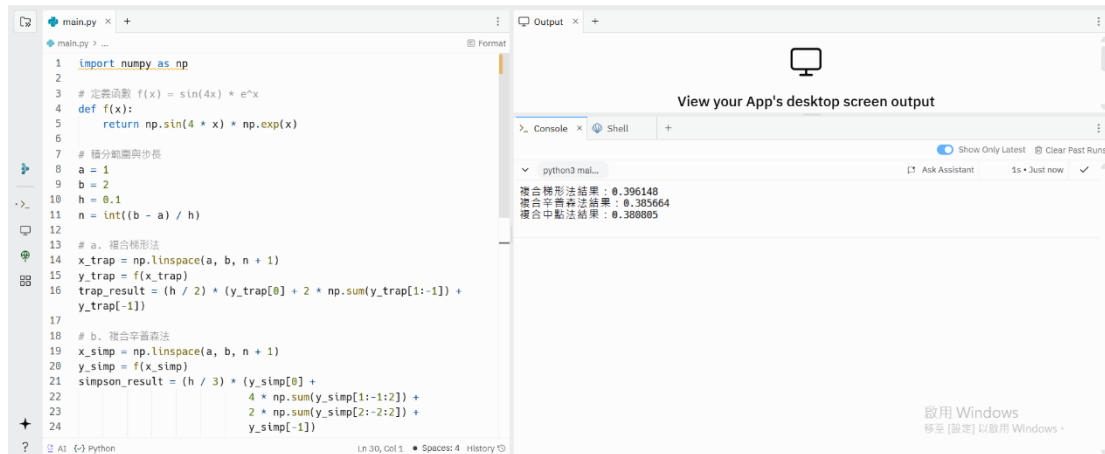


Q1

a. the composite trapezoidal rule : 0.396148

b. the composite Simpsons' method : 0.385664

c the composite midpoint rule : 0.380805



The screenshot shows a Python IDE with a file named `main.py`. The code defines a function `f(x) = sin(4x) * e^x` and implements three numerical integration methods: composite trapezoidal rule, composite Simpson's method, and composite midpoint rule. The output window displays the results for each method.

```
1 import numpy as np
2
3 # 定義函數 f(x) = sin(4x) * e^x
4 def f(x):
5     return np.sin(4 * x) * np.exp(x)
6
7 # 積分範圍與步長
8 a = 1
9 b = 2
10 h = 0.1
11 n = int((b - a) / h)
12
13 # a. 複合梯形法
14 x_trap = np.linspace(a, b, n + 1)
15 y_trap = f(x_trap)
16 trap_result = (h / 2) * (y_trap[0] + 2 * np.sum(y_trap[1:-1]) +
17 y_trap[-1])
18
19 # b. 複合辛普森法
20 x_simp = np.linspace(a, b, n + 1)
21 y_simp = f(x_simp)
22 simpson_result = (h / 3) * (y_simp[0] +
23 4 * np.sum(y_simp[1:-1:2]) +
24 2 * np.sum(y_simp[2:-2:2]) +
25 y_simp[-1])
```

Output:

```
View your App's desktop screen output
python3 mal...
複合梯形法結果: 0.396148
複合辛普森法結果: 0.385664
複合中點法結果: 0.380805
```

Q2

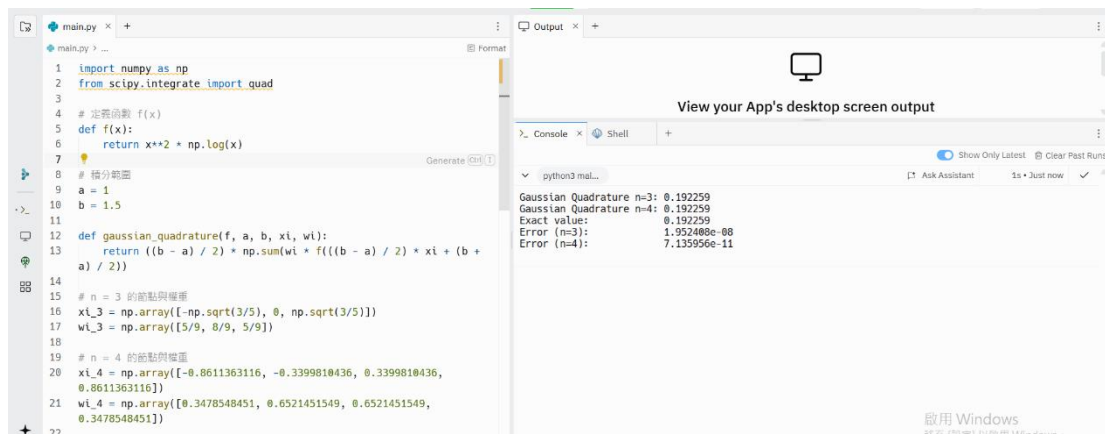
Gaussian Quadrature n=3: 0.192259

Gaussian Quadrature n=4: 0.192259

Exact value: 0.192259

Error (n=3): 1.952408e-08

Error (n=4): 7.135956e-11



The screenshot shows a Python IDE with a file named `main.py`. The code defines a function `f(x) = x^2 * ln(x)` and implements Gaussian Quadrature for n=3 and n=4. The output window displays the results for each method, the exact value, and the errors.

```
1 import numpy as np
2 from scipy.integrate import quad
3
4 # 定義函數 f(x)
5 def f(x):
6     return x**2 * np.log(x)
7
8 # 積分範圍
9 a = 1
10 b = 1.5
11
12 def gaussian_quadrature(f, a, b, xi, wi):
13     return ((b - a) / 2) * np.sum(wi * f(((b - a) / 2) * xi + (b +
14 a) / 2))
15
16 # n = 3 的節點與權重
17 xi_3 = np.array([-np.sqrt(3/5), 0, np.sqrt(3/5)])
18 wi_3 = np.array([5/9, 8/9, 5/9])
19
20 # n = 4 的節點與權重
21 xi_4 = np.array([-0.8611363116, -0.3399810436, 0.3399810436,
22 0.8611363116])
23 wi_4 = np.array([0.3478548451, 0.6521451549, 0.6521451549,
24 0.3478548451])
```

Output:

```
View your App's desktop screen output
python3 mal...
Gaussian Quadrature n=3: 0.192259
Gaussian Quadrature n=4: 0.192259
Exact value: 0.192259
Error (n=3): 1.952408e-08
Error (n=4): 7.135956e-11
```

Q3

Composite Simpson (n=4): 0.5119875440

Gaussian Quad (n=3) : 0.5118655399

Exact value: 0.5118446353

Error (Composite Simpson): 0.0001429087

Error(Gaussian Quad): 0.0000209046

main.py

```
1 import numpy as np
2 from scipy.integrate import dblquad
3 from numpy.polynomial.legendre import leggauss
4
5 # 積分函數
6 def f(x, y):
7     return 2 * y * np.sin(x) + np.cos(x)**2
8
9 # ----- Simpson's Rule (雙重積分) -----
10 def composite_simpson_double(f, a, b, m, n):
11     hx = (b - a) / m
12     result = 0
13     for i in range(m + 1):
14         xi = a + i * hx
15         cx = 1 if i == 0 or i == m else (4 if i % 2 == 1 else 2)
16         y1, y2 = np.sin(xi), np.cos(xi)
17         if y1 > y2:
18             y1, y2 = y2, y1
19         hy = (y2 - y1) / n
20         inner = 0
21         for j in range(n + 1):
22             yj = y1 + j * hy
23             cy = 1 if j == 0 or j == n else (4 if j % 2 == 1 else 2)
24             inner += cy * f(xi, yj)
25         result += cx * (hy / 3) * inner
```

Output

View your App's desktop screen output

python3 mal...

Method	Result	Abs. Error
Composite Simpson (n=4)	0.5119875440	0.0001429087
Gaussian Quad (n=3)	0.5118655399	0.0000209046
Scipy dblquad (Exact)	0.5118446353	(True Value)

Q4

第一題: 0.5259958841

第二題: 0.2744895428

main.py

```
1 import numpy as np
2
3 # 定義第一題的被積分的函數 f(x) = x*(-0.25) * sin(x)
4 def f_a(x):
5     return x*(-0.25) * np.sin(x)
6
7 # 定義第二題的被積分的函數 f(t) = sin(1/t) / t^3
8 def f_b(t):
9     return np.sin(1/t) * t**2
10
11 # 合成 Simpson 法則
12 def composite_simpson(f, a, b, n):
13     h = (b - a) / n
14     result = f(a) + f(b)
15     # 累加奇數位置的項 (x_1, x_3, ..., x_{n-1})
16     for i in range(1, n, 2):
17         result += 4 * f(a + i * h)
18     # 累加偶數位置的項 (x_2, x_4, ..., x_{n-2})
19     for i in range(2, n, 2):
20         result += 2 * f(a + i * h)
21     result *= h / 3
22     return result
23
24 # 設定第一題的積分範圍和步數 (將起點從0稍微偏移)
25 a_a = 0.0001
```

Output

View your App's desktop screen output

python3 mal...

第一題積分結果: 0.5259958841  
第二題積分結果: 0.2744895428