exercise2 (Score: 22.0 / 22.0)

1. Task (Score: 2.0 / 2.0)
2. Test cell (Score: 3.0 / 3.0)
3. Task (Score: 3.0 / 3.0)
4. Test cell (Score: 2.0 / 2.0)
5. Test cell (Score: 2.0 / 2.0)
6. Test cell (Score: 3.0 / 3.0)
7. Test cell (Score: 3.0 / 3.0)
8. Task (Score: 4.0 / 4.0)

Lab 4

1. 提交作業之前，建議可以先點選上方工具列的**Kernel**，再選擇**Restart & Run All**，檢查一下是否程式跑起來都沒有問題，最後記得儲存。
2. 請先填上下方的姓名(name)及學號(stduent_id)再開始作答，例如：

```
name = "我的名字"
student_id= "B06201000"
```

3. 演算法的實作可以參考[lab-4 (https://yuanyuyuan.github.io/itcm/lab-4.html)](https://yuanyuyuan.github.io/itcm/lab-4.html), 有任何問題歡迎找助教詢問。
4. **Deadline: 11/20(Wed.)**

In [1]:

```
name = "陳彥宇"
student_id = "B05303134"
```

Exercise 2

Let $I(f)$ be a define integral defined by

$$I(f) = \int_0^1 f(x)dx,$$

and consider the quadrature formula

$$\hat{I}(f) = \alpha_1 f(0) + \alpha_2 f(1) + \alpha_3 f'(0) \qquad (*)$$

for approximation of $I(f)$.

Part 1.

Determine the coefficients α_j for $j = 1, 2, 3$ in such a way that $\hat{I}$ has the degree of exactness $r = 2$. Here the degree of exactness r is to find r such that

$$\hat{I}(x^k) = I(x^k) \quad \text{for} \quad k = 0, 1, \ldots, r \quad \text{and} \quad \hat{I}(x^j) \neq I(x^j) \quad \text{for} \quad j > r,$$

where x^j denote the j-th power of x.

(Top)

Derive the values of $\alpha_1, \alpha_2, \alpha_3$ in $(*)$. You need to write down the detail in the cell below with Markdown/LaTeX.

Since $\hat{I}(1) = \alpha_1 + \alpha_2$, $\hat{I}(x) = \alpha_2 + \alpha_3$, and $\hat{I}(x^2) = \alpha_2$, and $I(1) = 1$, $I(x) = \frac{1}{2}$, and $I(x^2) = \frac{1}{3}$, let

$$\hat{I}(1) = \alpha_1 + \alpha_2 = 1 = I(1),$$

$$\hat{I}(x) = \alpha_2 + \alpha_3 = \frac{1}{2} = I(x),$$

$$\hat{I}(x^2) = \alpha_2 = \frac{1}{3} = I(x^2).$$

The solution of this linear system is $(\alpha_1, \alpha_2, \alpha_3) = (\frac{2}{3}, \frac{1}{3}, \frac{1}{6})$.

Fill in the tuple variable `alpha_1`, `alpha_2`, `alpha_3` with your answer above.

In [2]:

(Top)

```
'''Hint:
alpha_1 = ?
alpha_2 = ?
alpha_3 = ?
'''
# ===== 請實做程式 =====
alpha_1 = 2/3
alpha_2 = 1/3
alpha_3 = 1/6
# ====================
```

In [3]:

part_1 (Top)

```
print("alpha_1 =", alpha_1)
print("alpha_2 =", alpha_2)
print("alpha_3 =", alpha_3)
### BEGIN HIDDEN TESTS
assert abs(alpha_1 - 2/3) <= 1e-7, 'alpha_1 is wrong!'
assert abs(alpha_2 - 1/3) <= 1e-7, 'alpha_2 is wrong!'
assert abs(alpha_3 - 1/6) <= 1e-7, 'alpha_3 is wrong!'
### END HIDDEN TESTS
```

```
alpha_1 = 0.6666666666666666
alpha_2 = 0.3333333333333333
alpha_3 = 0.16666666666666666
```

(Top)

Part 2.

Find an apppropriate expression for the error $E(f) = I(f) - \hat{I}(f)$, and write your process in the below cell with Markdown/LaTeX.

By Taylor's theorem, for a smooth function f, $f(t) = f(0) + f^{'}(0)t + \frac{f^{''}(\xi)t^2}{2}$ for some $\xi \in (0, 1)$, which implies

$$I(f) = \int_0^1 f(t)dt = \int_0^1 f(0) + f^{'}(0)t + \frac{f^{''}(\xi)t^2}{2}dt = f(0) + \frac{f^{'}(0)}{2} + \frac{f^{''}(\xi)}{6}.$$

Hence,

$$E(f) = I(f) - \hat{I}(f) = \left(f(0) + \frac{f^{'}(0)}{2} + \frac{f^{''}(\xi)}{6}\right) - \left(\frac{2}{3}f(0) + \frac{1}{3}f(1) + \frac{1}{6}f^{'}(0)\right).$$

With some calculation, $E(f) = \frac{1}{3}f(0) - \frac{1}{3}f(1) + \frac{1}{3}f^{'}(0) + \frac{f^{''}(\xi)}{6}$ for some $\xi \in (0, 1)$.

Part 3.

Compute

$$\int_0^1 e^{-\frac{x^2}{2}} dx$$

using quadrature formulas $(*)$, the Simpson's rule and the Gauss-Legendre formula in the case $n = 1$. Compare the obtained results.

Part 3.1

Import necessary libraries

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special.orthogonal import p_roots
```

Part 3.2

Define the function $f(x) = e^{-\frac{x^2}{2}}$ and its derivative.

In [5]:

(Top)

```python
def f(x):
    # ===== 請實做程式 =====
    return np.exp(-1*x**2/2)
    # ====================

def d_f(x):
    # ===== 請實做程式 =====
    return -1*x*f(x)
    # ====================
```

Print and check your functions.

In [6]:

part_3_1_1 (Top)

```python
print('f(0) =', f(0))
print("f'(0) =", d_f(0))
### BEGIN HIDDEN TESTS
assert abs(f(5) - np.exp(-5**2/2)) <= 1e-7, 'f(5) is wrong!'
assert abs(f(10) - np.exp(-10**2/2)) <= 1e-7, 'f(10) is wrong!'
assert abs(d_f(5) - -5*np.exp(-5**2/2)) <= 1e-7, "f'(5) is wrong!"
assert abs(d_f(10) - -10*np.exp(-10**2/2)) <= 1e-7, "f'(10) is wrong!"
### END HIDDEN TESTS
```

```
f(0) = 1.0
f'(0) = 0.0
```

Part 3.3

Compute

$$\int_0^1 e^{-\frac{x^2}{2}} dx$$

with the formula $(*)$.

Fill your answer into the variable `approximation` .

In [7]:

(Top)

```python
# Hint: approximation = ...
# ===== 請實做程式 =====
approximation = alpha_1*f(0)+alpha_2*f(1)+alpha_3*d_f(0)
# ====================
```

Run and check your answer.

In [8]:

part_3_2 (Top)

```python
print("The result of the integral is", approximation)
### BEGIN HIDDEN TESTS
assert abs(approximation - 0.8688435532375445) < 1e-3, "wrong approximation!"
### END HIDDEN TESTS
```

```
The result of the integral is 0.8688435532375445
```

Part 3.4

Compute

$$\int_0^1 e^{-\frac{x^2}{2}} dx$$

with Simpson's rule.

Implement Simpson's rule

In [9]:

(Top)

```
def simpson(
    f,
    a,
    b,
    N=50
):
    '''
    Parameters
    ----------
    f : function
        Vectorized function of a single variable
    a , b : numbers
        Interval of integration [a,b]
    N : (even) integer
        Number of subintervals of [a,b]

    Returns
    -------
    S : float
        Approximation of the integral of f(x) from a to b using
        Simpson's rule with N subintervals of equal length.
    '''
    # ===== 請實做程式 =====
    if N % 2 == 1:
        raise ValueError("N must be an even integer.")
    dx = (b-a)/N
    x = np.linspace(a,b,N+1)
    y = f(x)
    S = dx/3 * np.sum(y[0:-1:2] + 4* y[1::2] + y[2::2])
    return S
    # ====================
```

Run and check your function.

In [10]:

```
        simpson
```

(Top)

```
S = simpson(f, 0, 1, N=50)
print("The result from Simpson's rule is", S)
### BEGIN HIDDEN TESTS
assert abs(S - 0.8556243929705796) < 1e-7, "Wrong answer!"
### END HIDDEN TESTS
```

```
The result from Simpson's rule is 0.8556243929705796
```

Part 3.5

Compute

$$\int_0^1 e^{-\frac{x^2}{2}} dx$$

with the Gauss-Legendre formula using $n = 1$.

In [11]:

(Top)

```python
def gauss(
    f,
    n,
    a,
    b
):
    '''
    Parameters
    ----------
    f : function
        Vectorized function of a single variable
    n : integer
        Number of points
    a , b : numbers
        Interval of integration [a,b]

    Returns
    -------
    G : float
        Approximation of the integral of f(x) from a to b using the
        Gaussian–Legendre quadrature rule with N points.
    '''
    # ===== 請實做程式 =====
    [x, w] = p_roots(n)
    G = 0.5*(b-a)*sum(w*f(0.5*(b-a)*x+(a+b)/2))
    return G
    # ====================
```

Run and check your function.

In [12]:

Gauss-Legendre (Top)

```python
G = gauss(f, 1, 0, 1)
print("The result from Gauss-Legendre is", G)
### BEGIN HIDDEN TESTS
assert abs(G - 0.88) <= 1e-1, "Wrong answer!"
### END HIDDEN TESTS
```

```
The result from Gauss-Legendre is 0.8824969025845955
```

(Top)

Part 3.6

Compare the obtained results of three methods above and write down your observation. You can use either code or markdown to depict.

Since the implement of the Simpson's rule has points $N = 50$, which is much more than the points $N = 3$ in the quadrature formula (*) and the Gauss-Legendre in the sense that the derivative $f'(0)$ is consider as a point, we take the result of the Simpson's rule as the solution to analyize the others' error.

In [13]:

```python
print('The error of the result from the quadrature formula (*) is', abs(approximation-S),'.')
print('The error of the result from the Gauss-Legendre is', abs(approximation-G), '.')
```

```
The error of the result from the quadrature formula (*) is 0.013219160266964902 .
The error of the result from the Gauss-Legendre is 0.01365334934705098 .
```

Hence, by the above comparation, we find that the quadrature formula (*) has the better approximation than the Gauss-Legendre under the points $N = 3$.