4 Лабораторна робота №4. Розробка додатків на Python з графічним інтерфейсом користувача і використання бібліотек Matplotlib, NumPy.

4.1 Мета і завдання

Мета. Ознайомитися з організацією графічного інтерфейсу користувача на основі бібліотеки tkinter.

Завдання:

Вправа 1 (для всіх варіантів).

Встановіть matplotlib. Накресліть діаграму розсіювання (scatter diagram) цих пар (x, y): ((0, 0), (3, 5), (6, 2), (9, 8), (14, 10)). Намалюйте лінійний графік тих самих даних. Намалюйте графік (лінійний графік з маркерами) тих самих даних.

Рішення.

Запустіть VS Code і перейдіть в директорію лабораторних робіт нашого курсу (наприклад, Python_Labs).

Активізуйте вікно терміналу.

В вікні терміналу наберіть:

```
pip install matplotlib
```

Після встановлення бібліотеки створіть новий файл (lab4_Ex1.py) з таким вмістом:

```
import matplotlib.pyplot as plt
x = (0, 3, 6, 9, 14)
y = (0, 5, 2, 8, 10)
fig, plots = plt.subplots(nrows=1, ncols=3)
plots[0].scatter(x, y)
plots[1].plot(x, y)
plots[2].plot(x, y, 'o-')
plt.show()
```

Запустіть файл на виконання. Ви отримаєте наступне:

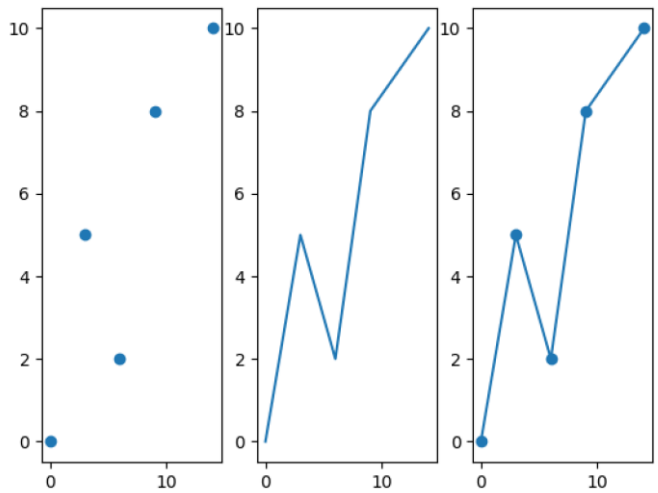

Рис. 3 Результат виконання lab4_Ex1.py

Вправа 2 (для всіх варіантів).

Ми розробимо класи для чисельного інтегрування [2].

Так само, як чисельне диференціювання, чисельне інтегрування є основою обчислювальної математики. Існує безліч методів на вибір, і всі вони можуть бути записані у формі:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} w_i f(x_i).$$

На основі цієї загальної формули різні методи реалізуються шляхом вибору точок інтегрування x_i та відповідних ваг w_i. Наприклад, ***формула трапецій*** має вигляд:

$$x_i = a + ih, \quad w_0 = w_{n-1} = \frac{h}{2},\ w_i = h\ (i \neq 0, n-1),$$

де h = (b − a)/(n − 1), серединний метод (***midpoint rule***) має вигляд:

$$x_i = a + \frac{h}{2} + ih, \quad w_i = h,$$

де h = (b−a)/n, а ***формула Сімпсона*** (Simpson) має вигляд:

$$x_i = a + ih, \quad h = \frac{b-a}{n-1},$$

$$w_0 = w_{n-1} = \frac{h}{6},$$

$$w_i = \frac{h}{3} \text{ for } i \text{ even}, \quad w_i = \frac{2h}{3} \text{ for } i \text{ odd}.$$

Інші методи мають більш складні формули для w_i та x_i, а деякі методи вибирають точки випадковим чином (наприклад, інтегрування Монте-Карло).

Формула чисельного інтегрування може бути реалізована як клас з a, b і n як атрибутами та методом інтегрування для обчислення інтегралу.

Має сенс розмістити весь загальний код у базовому класі та код, специфічний для різних методів, у похідних класах. Тут ми можемо поставити $\sum_j w_j f(x_j)$

в базовий клас (метод integrate) і нехай підкласи доповнюють цей клас кодом, специфічним для певної формули, тобто вибором w_i та x_i. Код цього методу можна помістити всередину методу, наприклад, з назвою construct_rule. Суперклас (базовий клас) для ієрархії чисельного інтегрування може виглядати так:

```
import numpy as np
class Integrator:
    def __init__(self, a, b, n):
        self.a, self.b, self.n = a, b, n
        self.points, self.weights = self.construct_method()
```

```
    def construct_method(self):
        raise NotImplementedError('no rule in class %s' % \
                                  self.__class__.__name__)
    def integrate(self, f):
        s = 0
        for i in range(len(self.weights)):
            s += self.weights[i]*f(self.points[i])
        return s

    def vectorized_integrate(self, f):
        # f must be vectorized for this to work
        return np.dot(self.weights, f(self.points))
```

Зверніть увагу на реалізацію construct_method, яка викликатиме помилку, якщо її викликати, вказуючи, що єдина мета інтегратора - це суперклас, і його не слід використовувати безпосередньо. З іншого боку, ми могли б, звичайно, просто не включати метод construct_method до суперкласу взагалі. Однак підхід, що використовується тут, робить ще більш очевидним, що клас - це просто суперклас і що цей метод потрібно реалізувати у підкласах.

Надклас забезпечує загальну основу для реалізації різних методів, які потім можуть бути реалізовані як підкласи. Метод трапецій та серединний метод можна реалізувати наступним чином:

```
class Trapezoidal(Integrator):
    def construct_method(self):
        h = (self.b - self.a)/float(self.n - 1)
        x = np.linspace(self.a, self.b, self.n)
        w = np.zeros(len(x))
        w[1:-1] += h
        w[0] = h/2; w[-1] = h/2
        return x, w

class Midpoint(Integrator):
    def construct_method(self):
        a, b, n = self.a, self.b, self.n # quick forms
        h = (b-a)/float(n)
        x = np.linspace(a + 0.5*h, b - 0.5*h, n)
        w = np.zeros(len(x)) + h
        return x, w
```

Більш складне правило Сімпсона можна додати до такого підкласу:

```
class Simpson(Integrator):
    def construct_method(self):
        if self.n % 2 != 1:
            print ('n=%d must be odd, 1 is added' % self.n)
            self.n += 1
        x = np.linspace(self.a, self.b, self.n)
        h = (self.b - self.a)/float(self.n - 1)*2
        w = np.zeros(len(x))
        w[0:self.n:2] = h*1.0/3
        w[1:self.n-1:2] = h*2.0/3
        w[0] /= 2
        w[-1] /= 2
```

```
        return x, w
```

Правило Сімпсона є більш складним, оскільки воно використовує різні ваги для непарних і парних точок. Тут ми наводимо всі деталі для повноти, але насправді не потрібно вивчати деталі всіх формул. Важливими частинами тут є дизайн класів та використання ієрархії класів.

Щоб продемонструвати, як можна використовувати класи, обчислимо інтеграл

$$\int_0^2 x^2 dx$$

Використовуючи 101 точку:

```
def f(x):
    return x*x

simpson = Simpson(0, 2, 101)
print(simpson.integrate(f))
trapez = Trapezoidal(0, 2, 101)
print(trapez.integrate(f))
```

Потік виконання програми в цьому випадку може бути не зовсім очевидним. Коли ми створюємо екземпляр за допомогою `simpson = Simpson (0, 2, 101)`, викликається конструктор суперкласу, але цей метод потім викликає construct_method у класі Simpson. Метод `simpson.integrate(f)` потім викликає метод інтегрування, успадкований від суперкласу. Однак, як для користувачів класу, жодна з цих деталей насправді не має значення. Ми використовуємо клас Simpson так, ніби всі методи були реалізовані безпосередньо в цьому класі, незалежно від того, що насправді вони успадковані від іншого класу.

Завдання вправи:

Сформувати файл integral.py, помістивши в нього наведені вище фрагменти коду. Виконати програму і проаналізувати результат. Чи зміниться результат, якщо задати 11 точок? Обчисліть точне значення інтегралу. Поясніть використані в програмі засоби бібліотеки NumPy.

Варіант 1.

Завдання 1 (GUI).

В полі введення задана вага в кілограмах. При натисканні кнопки перевести вагу в грами, в фунти і в унції.

Завдання 2 (matplotlib).

Зобразити 2d графік функції відповідно своєму варіанту та зберегти у .png файл.

Y(x)=x*sin(5*x), x=[-2...5]

Варіант 2.

Завдання 1 (GUI).

Розробіть графічний інтерфейс до однієї з виконаних раніше лабораторних робіт.

Завдання 2 (matplotlib).

Зобразити 2d графік функції відповідно своєму варіанту та зберегти у .png файл.

Y(x)=1/x*sin(5*x), x=[-5...5]

Варіант 3.

Завдання 1 (GUI).

Розробіть графічний інтерфейс до однієї з виконаних раніше лабораторних робіт.

Завдання 2 (matplotlib).

Зобразити 2d графік функції відповідно своєму варіанту та зберегти у .png файл.

Y(x)=2^x*sin(10x), x=[-3...3]

4.4 Питання для самоконтролю

1. Які засоби мова Python надає для роботи з 2D графікою? Які бібліотеки призначені для роботи з графікою?
2. Яким чином можна відобразити графік математичної функції?
3. Як можна налаштувати колір та тип лінії на графіку математичної функції?
4. Яким чином можна відобразити гісторграму?
5. Яким чином можна зберегти зображення у файл?

ЛІТЕРАТУРА

1. Яковенко А.В. Основи програмування. Python. Частина 1: підручник для студ. спеціальності 122 "Комп’ютерні науки". – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.

2. Програмування на мові Python. Методичні вказівки до виконання лабораторних робіт студентами денної та заочної форми навчання спеціальностей 123 "Комп'ютерна інженерія", 125 "Кібербезпека" / Укл.: Є.В. Мелешко – Кропивницький: ЦНТУ, 2017. – 58 с.
3. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
4. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. – Чернігів: ФОП Баликіна С.М., 2020. 180 с.
5. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування. Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.
6. Лутц М. Изучаем Python, том 1, 5-е изд. — СПб.: ООО "Диалектика", 2019. — 832 с.
7. Лутц М. Изучаем Python, том 2, 5-е изд. — СПб.: ООО "Диалектика", 2020. — 720 с.
8. Joakim Sundnes Introduction to Scientific Programming with Python. – Springer, 2020. 148 p.
9. Bill Lubanovic Introducing Python. – O'Reilly Media, 2020. 597 p.