МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Основи роботи з бібліотекою Matplotlib»
з курсу «Обробка даних Python»
для студентів спеціальностей 121 Інженерія програмного забезпечення,
122 Комп'ютерні науки, 124 Системний аналіз,
126 Інформаційні системи і технології

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 06.10.2021 року

Харків
НТУ «ХПІ»
2021

Методичні вказівки до лабораторної роботи «Основи роботи з бібліотекою Matplotlib» з курсу «Обробка даних Python» для студентів спеціальностей 121 Інженерія програмного забезпечення, 122 Комп'ютерні науки, 124 Системний аналіз, 126 Інформаційні системи і технології / уклад. : С. М. Коваленко, С. В. Коваленко, О. В. Шматко. – Харків : НТУ «ХПІ», 2021. – 28 с.

Укладачі: С. М. Коваленко,
С. В. Коваленко,
О. В. Шматко

Рецензент: І. П. Гамаюн

Кафедри програмної інженерії та інформаційних технологій управління, системного аналізу та інформаційно-аналітичних технологій

ВСТУП

Matplotlib – це бібліотека двовимірної графіки для мови програмування Python, за допомогою якої можна створювати високоякісні рисунки. Отримані зображення можуть бути збережені в різних форматах (як растрових, так і векторних) для подальшого використання в наукових публікаціях, веб-додатках, документації тощо. Matplotlib написана і підтримується, в основному, Джоном Хантером і поширюється на умовах BSD-подібної ліцензії. Хоча спочатку Matplotlib створювалась як альтернатива графічним командам MATLAB (що не є вільним програмним забезпеченням), але в подальшому перетворилася в незалежний проект.

Хоча бібліотека Matplotlib побудована на принципах об'єктно-орієнтованого програмування, для сумісності з MATLAB, вона також має і процедурний інтерфейс. В даній лабораторній роботі будуть розглянуті обидва підходи.

Мета: отримати базові знання та навички з роботи з бібліотекою Matplotlib.

1. ТЕОРЕТИЧНІ ОСНОВИ

1.1. Встановлення бібліотеки

Matplotlib не входить до складу стандартних бібліотек Python. Тому попередньо її треба завантажити та встановити [1]. Але в цьому курсі рекомендовано використовувати дистрибутив Anaconda, що має передустановлені бібліотеки для аналізу та візуалізації даних, зокрема Matplotlib.

Matplotlib складається із великої кількості модулів, які в свою чергу складаються із великої кількості класів і функцій, що ієрархічно пов'язані між собою. Зазвичай знайомство з бібліотекою починають з самого високорівневого інтерфейса `matplotlib.pyplot`. Стандартним способом імпорту модуля є команда з аліасом `plt`:

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
```


1.2 Встановлення стилю

Вигляд отриманих графіків і діаграм буде залежати від обраного стилю відображення. Список всіх доступних стилів можна отримати за допомогою команди `plt.style.available`. У відповідь буде виведено список стилів: `['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']`.

Для подальшої роботи з бібліотекою будемо використовувати стиль `'seaborn-whitegrid'`:

```
In [2]: plt.style.use('seaborn-whitegrid')
```

1.3 Створення графіків в Jupyter Notebook

Jupyter Notebook – це інструмент інтерактивного аналізу даних на основі браузера, який може поєднувати текст, код, графіку, елементи HTML та багато іншого в один документ, що виконується.

Створення інтерактивних графіків у Jupyter notebook може виконуватися за допомогою команди `%matplotlib`. У Jupyter notebook ви також можете вбудувати графіку безпосередньо в блокнот із двома можливими варіантами:

- `%matplotlib notebook` приводить до інтерактивних графіків, вбудованих у блокнот
- `% matplotlib inline` приводить до статичних зображень, вбудованих у блокнот

```
In [3]: %matplotlib inline #створюємо статичні
                           зображення
        #Встановлюємо значення незалежної змінної
```

```
x = np.linspace(0, 10, 100)
#Встановлюємо значення першої функції
y = np.sin(x)
#Встановлюємо значення другої функції
z = np.cos(x)
#Рисуємо графік першої функції
plt.plot(x, y)
#Рисуємо графік другої функції
plt.plot(x, z);
```

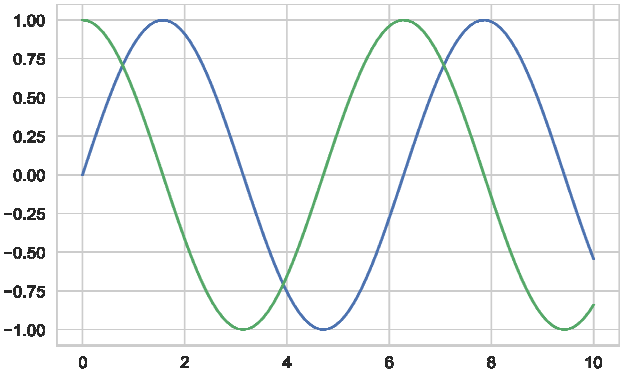


Рисунок 1.1 – Результат виконання коду в `In [3]`

1.4. Частини рисунку в Matplotlib

Робота з графікою в Matplotlib передбачає операції з різними рівнями рисунку: Рисунок (`Figure`) → Область рисування (`Axes`) → Координатна вісь (`Axis`). Також існують елементи рисунка (`Artists`), що присутні на всіх рівнях `Figure` (рис. 1.2).

1.4.1. Рисунок (Figure)

Рисунок (экземпляр класса `plt.Figure`) є об'єктом самого верхнього рівня, на якому розташовується хоча б одна область рисування (`Axes`), елементи рисунка `Artisits` (заголовки, легенда и т.д.) та основа (Canvas).

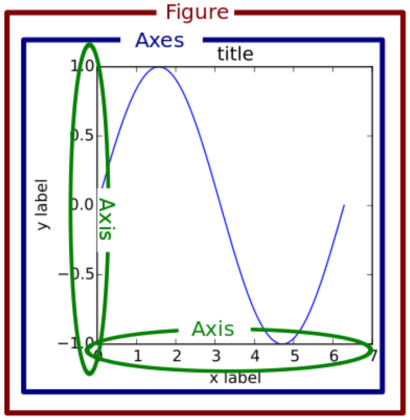


Рисунок 1.2 – Структура рисунку Matplotlib

1.4.2. Область рисування (Axes)

Область рисування є об'єктом середнього рівня, який є, напевно, головним об'єктом роботи з графікою Matplotlib. На рис. 1.3 – це обмежувальна рамка підписами, мітками та сіткою, що у кінцевому результаті буде містити елементи графіка і складе нашу візуалізацію.

На одному рисунку може бути декілька областей рисування, але кожен `Axes` може бути тільки на одному `Figure`.

Кожна область рисування `Axes` містить дві (або три – в разі тривимірних даних) координатні осі (`Axis`), які впорядковують відображення даних.

У найпростішому випадку створити рисунок і область рисування потрібно наступним чином:

```
In [4]: fig = plt.figure()
        ax = plt.axes()
```

1.4.3. Координатна ось (Axis)

Ці об'єкти подібні до числових осей. Вони визначають межі графіків, генерують ціну ділення та засічки (`ticks`) для їх відображення, встановлюють мітки (підписи) для цих засічок (`ticklabels`). Розташування засічок визначається об'єктом `Locator`, а рядки міток форматуються за допомогою `Formatter`. Поєднання

правильного `Locator` та `Formatter` дає дуже точний контроль над місцями засічок та мітками.

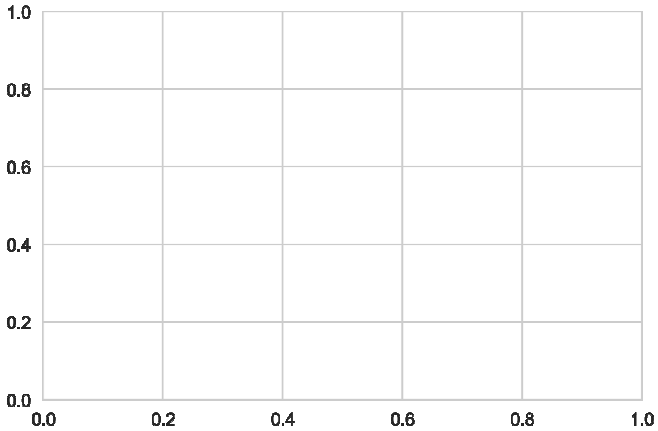


Рисунок 1.3. – Рисунок та пуста область рисування Matplotlib

1.4.4. Елементи рисунка (Artists)

Взагалі все, що ви можете побачити на рисунку, – це `Artists` (навіть об'єкти `Figure`, `Axes` та `Axis`). Елементи рисунка `Artists` включають в себе такі прості об'єкти, як текст (`Text`), плоска лінія (`Line2D`), фігура (`Patch`) та інші. Коли відбувається відображення рисунка (figure rendering), всі елементи рисунка `Artists` наносяться на основу-полотно (`Canvas`). Велика частина з них пов'язується з областю рисування `Axes`.

1.5. Два інтерфейси Matplotlib

Особливістю Matplotlib є його подвійний інтерфейс: структурний інтерфейс в стилі MATLAB і більш потужний об'єктно-орієнтований інтерфейс.

1.5.1. Інтерфейс в стилі MATLAB

Як було зазначено раніше, Matplotlib започатковувся як альтернатива MATLAB, його структурний інтерфейс відображає саме цей факт.

```
In [5]: plt.figure()  #створення загального рисунку

        # створення першої із двох областей рисування
          plt.subplot(2, 1, 1) # (rows, columns,
           panel number)
          plt.plot(x, np.sin(x))

          # створення другої області призначення
           поточних осей
          plt.subplot(212)
          plt.plot(x, np.cos(x));
```

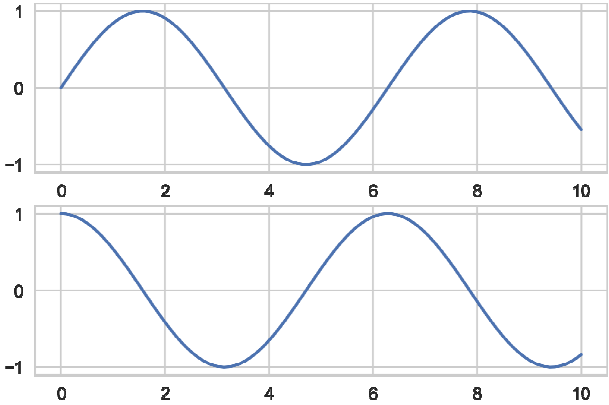


Рисунок 1.4 – Результат виконання коду в `In[5]`

1.5.2. Об'єктно-орієнтований інтерфейс

Об'єктно-орієнтований інтерфейс доступний для більш складних

ситуацій, а також для тих випадків, коли потрібен більший контроль над областю побудови діаграми. В об'єктно-орієнтованому інтерфейсі функції побудови графіків є методами явних об'єктів `Figure` і `Axes`. Щоб відтворити попередній графік (рис. 1.4) з використанням цього стилю побудови, ви можете зробити наступне:

```
In [6]: # Створюємо рисунок та область малювання
        # ax буде масивом із двох об'єктів Axes
        fig, ax = plt.subplots(2)

        #Визиваємо метод plot()для відповідного
        об'єкту
        ax[0].plot(x, np.sin(x))
        ax[1].plot(x, np.cos(x));
```

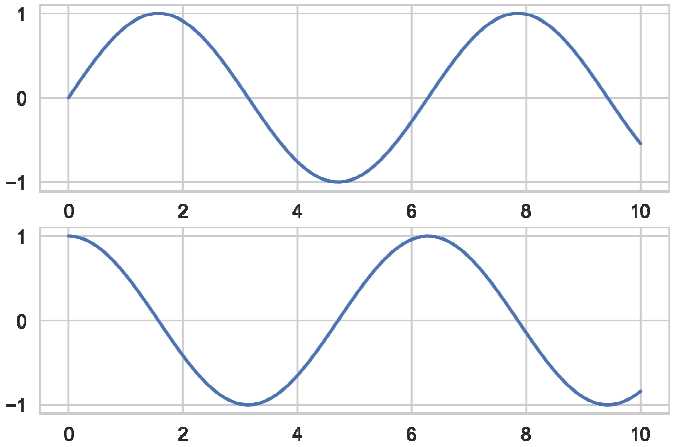


Рисунок 1.5 – Результат виконання коду в `In[6]`

Для більш простих графіків вибір інтерфейсу для використання в значній мірі є питанням переваги, але об'єктно-орієнтований підхід може стати необхідністю з ускладненням графіків.

1.6. Налаштування графіка

1.6.1. Налаштування кольору графіка

Метод `plt.plot()` приймає додаткові аргументи, які можна використовувати для налаштування кольору та стилю ліній. Щоб налаштувати колір, ви можете використовувати ключове слово `color`, яке приймає строковий аргумент, що представляє практично будь-який колір. Колір можна вказати різними способами:

```
In [7]: # призначення кольору по назві
        plt.plot(x, np.sin(x - 0), color='red')
        # короткий код кольору (rgbcmyk)
        plt.plot(x, np.sin(x - 1), color='g'
        # градіції сірого від 0 до 1
        plt.plot(x, np.sin(x - 2), color='0.75')
        # шістнадцятковий код кольору (RRGGBB від
        00 до FF)
        plt.plot(x, np.sin(x - 3), color='#FFDD44')
        # кортеж RGB зі значеннями від 0 до 1
    plt.plot(x, np.sin(x - 4), color=(1.0,0.5,1.0))
        # всі HTML назви кольорів
        plt.plot(x, np.sin(x - 5),
                 color='limegreen');
```

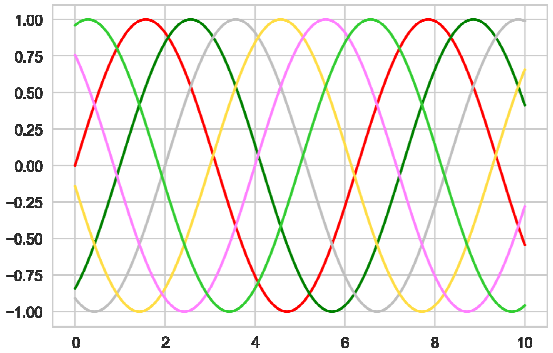


Рисунок 1.6 – Результат виконання коду в `In[7]`

1.6.2. Налаштування стилю ліній

Для встановлення типу ліній необхідно використовувати ключове слово `linestyle`. Стилі можуть бути задані повною назвою та кодами: суцільна (‘solid’,‘-’), пунктирна (‘dotted’,‘:’), штрихова (‘dashed’,‘ --’) та штрих-пунктирна (‘dashdot’,‘-.’).

```
In [8]: # використання повної назви:
        plt.plot(x, x + 0, linestyle=‘solid’)
        # суцільна
        plt.plot(x, x + 1, linestyle=‘dashed’)
        # штрихова
        plt.plot(x, x + 2, linestyle=‘dashdot’)
        #штрих-пункт.
        plt.plot(x, x + 3, linestyle=‘dotted’)
        # пунктирна
        # For short, you can use the following codes:
        plt.plot(x, x + 5, linestyle=‘-’)
        # суцільна
        plt.plot(x, x + 6, linestyle=‘--’)
        # штрихова
        plt.plot(x, x + 7, linestyle=‘-.’)
        # штрих-пунктир
        plt.plot(x, x + 8, linestyle=‘:’);
        # пунктирна
```

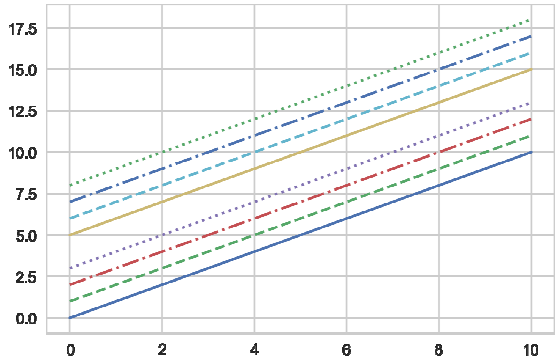


Рисунок 1.7 – Результат виконання коду в `In[8]`

Для скорочення запису можна вводити коди кольору та стилю лінії як один параметр

```
In [9]: # малиновий колір та штрих-пунктирна лінія
        plt.plot(x, np.sin(x), ‘m-.’)
```

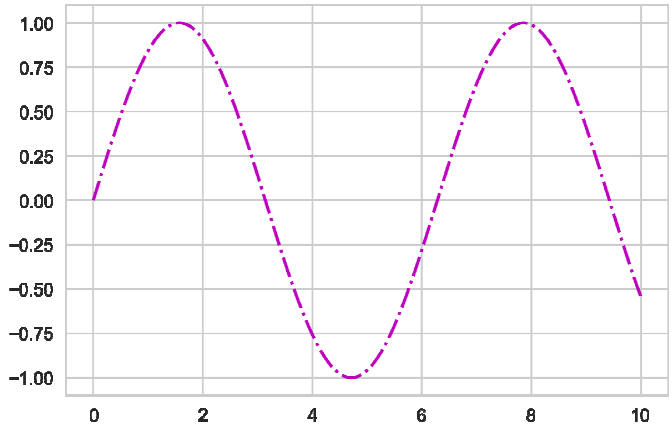


Рисунок 1.8 – Результат виконання коду в `In[9]`

1.6.3. Налаштування лімітів осей

Matplotlib непогано справляється з вибором меж осей для вашого графіка за замовчуванням, але іноді потрібен більш тонкий контроль. Найпростіший спосіб налаштувати межі осі – використовувати методи `plt.xlim()` і `plt.ylim()`:

```
In [10]: plt.plot(x, np.sin(x))
         plt.xlim(-2, 12)
         plt.ylim(-1.5, 1.5);
```

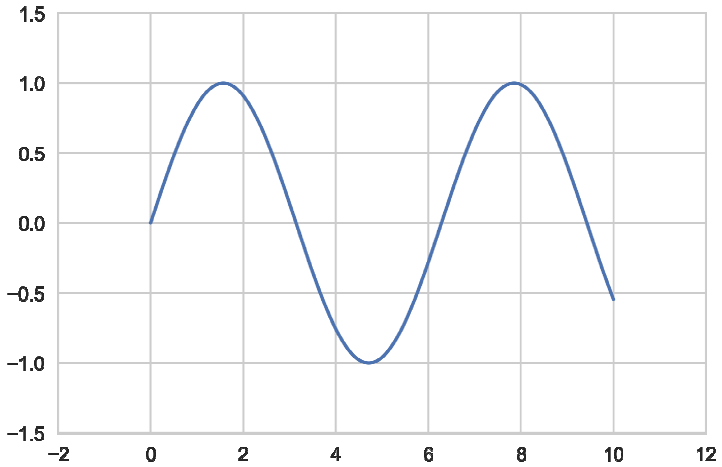


Рисунок 1.9 – Результат виконання коду в `In[10]`

Якщо з якоїсь причини ви хочете, щоб будь-яка вісь відображалася в зворотному порядку, ви можете просто змінити порядок аргументів:

```
In [11]: plt.xlim(10, 0)
         plt.ylim(1.5, -1.5);
```

Також корисним є метод `plt.axis()`, що дозволяє встановити межі `x` і `y` за один виклик, передаючи список`[xmin, xmax, ymin, ymax]`.

```
In [12]: plt.axis([-2, 12, -1.5, 1.5])
```

Результат `In [12]` буде такий же як і на рис. 1.9

1.6.4. Створення підписів на графіках

Для швидкого встановлення підписів існують методи `title` – для заголовку, `xlabel` – для підпису осі `x`, і `ylabel` – для осі `y`. Всі ці

методи підтримують рендеринг формул на мові LaTeX:

```
In [13]: plt.plot(x, np.sin(x**2)**3)
         # LaTeX у заголовку та мітці осі y
         plt.title("Графік функції $sin^3(x^2)$")
         plt.xlabel("x")
         plt.ylabel("$sin^3(x^2)$")
```

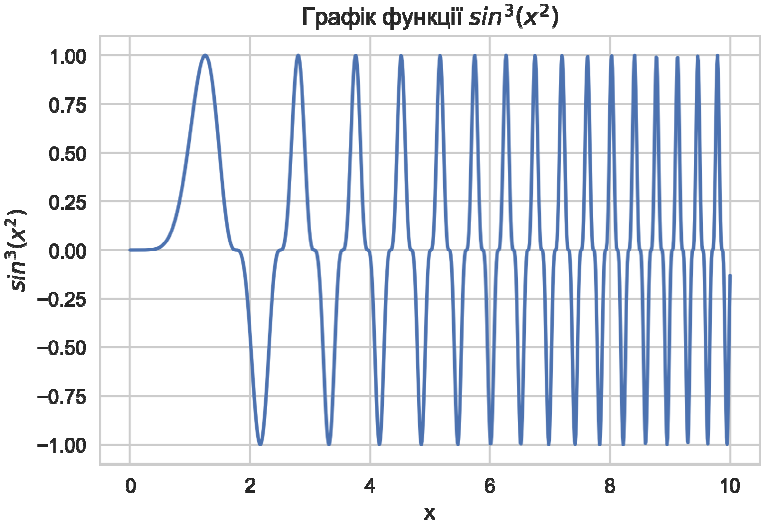


Рисунок 1.10 – Результат виконання коду в `In[13]`

Позицію, розмір і стиль міток можна регулювати, використовуючи додаткові аргументи функції. Для отримання додаткової інформації див. документацію Matplotlib та документацію кожної з цих функцій.

```
In [14]: plt.xlabel?
```

1.6.5. Створення легенди

Коли в межах однієї області рисування показано декілька графіків, може бути корисно створити легенду, яка визначає кожен тип лінії. Знову ж таки, Matplotlib має вбудований спосіб швидкого

створення легенди. Це робиться за допомогою методу `plt.legend()`. Хоча існує кілька дійсних способів використання `plt.legend()`, найпростіше вказати мітку кожного рядка, використовуючи ключове слово `label` функції `plot()`. В легенді також може бути використаний код в LaTeX:

```
In [15]:plt.plot(x, np.sin(x)**2, '-r',
         label='$sin(x^2)$')
        plt.plot(x, np.sin(x)**3, '--m',
         label='$sin(x^3)$')
```

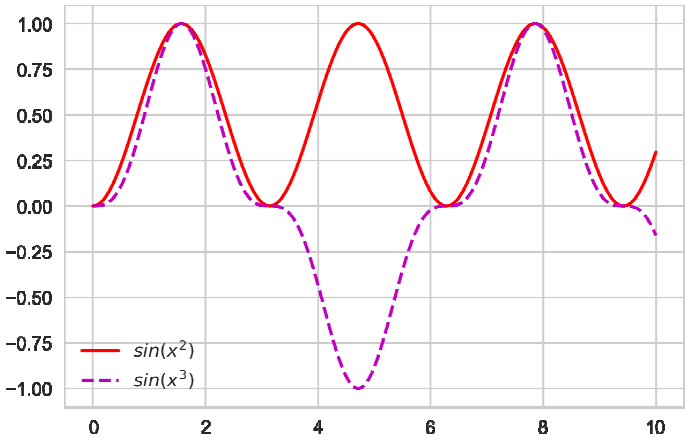


Рисунок 1.11 – Результат виконання коду в `In[15]`

Як видно із рисунка 1.11, функція `plt.legend`() відстежує стиль і колір рядка та ставить їх у відповідність до відповідної мітки. Більше інформації про форматування легенд сюжету можна знайти в докстринг `plt.legend (plt.legend?)`.

1.6.6. Відповідність методів налаштування графіків у структурному та об'єктно-орієнтованому підходах

В розділах 1.6.1. – 1.6.5. приклади були створені з використанням

структурного інтерфейсу. Але більшість `plt` методів безпосередньо транслюються в методи `ax`, наприклад `plt.plot()` → `ax.plot()`, `plt.legend()` → `ax.legend()`. Але є і виключення:

- `plt.xlabel()` → `ax.set_xlabel()`;
- `plt.ylabel()` → `ax.set_ylabel()`;
- `plt.xlim()` → `ax.set_xlim()`;
- `plt.ylim()` → `ax.set_ylim()`;
- `plt.title()` → `ax.set_title()`.

В об'єктно-орієнтованому інтерфейсі часто зручніше використовувати метод `ax.set()` для встановлення всіх цих властивостей одночасно:

```
In [16]: ax = plt.axes()
         ax.plot(x, np.sin(x)**2)
         ax.set(xlim=(0, 6), ylim=(-1, 2),
                xlabel='x', ylabel='$sin^2(x)$',
                title="Об'єктно-орієнтований підхід")
```

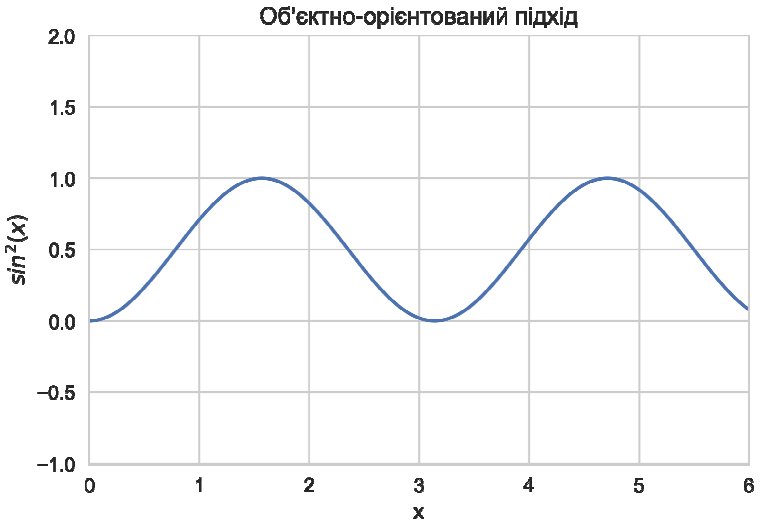


Рисунок 1.12 – Результат виконання коду в `In[16]`

1.7. Деякі види графіків

1.7.1. Лінійні

У всіх попередніх розділах ми розглядали `plt.plot` / `ax.plot` та отримували саме лінійні графіки.

1.7.2. Точкові

1.7.2.1. З використанням `plt.plot` / `ax.plot`

Можна використовувати `plt.plot` / `ax.plot` також і для точкових графіків. Для цього ми повинні не вказувати тип лінії, а вказати тип маркера:

```
In [17]: plt.plot(x, np.sin(x), 'ro')
         #тут o - тип маркера
```

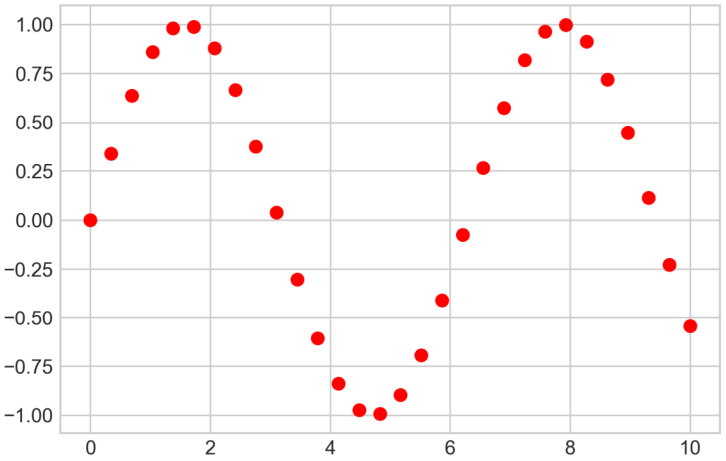


Рисунок 1.13 – Результат виконання коду в `In[17]`

Параметр ‘o’ тут – тип маркера, що використовується для побудови графіка.

Подібно до того, як ми вказували такі параметри, як ‘-’, ‘--’ для управління стилем лінії, стиль маркера має власний набір коротких

рядкових кодів. Повний список доступних символів можна переглянути в документації `plt.plot` або в онлайн-документації Matplotlib. Більшість можливостей досить інтуїтивно зрозумілі, ось низка найпоширеніших:

```
In [18]: rng = np.random.RandomState(0)
         for marker in ['o', '.', ',', 'X', '*', 'v',
                        '^', '<', '>', 's', 'd']:
             plt.plot(rng.rand(3), rng.rand(3),
                 marker, label=f"marker='{marker}'")
         plt.legend()
         plt.xlim(0, 1.5)
```

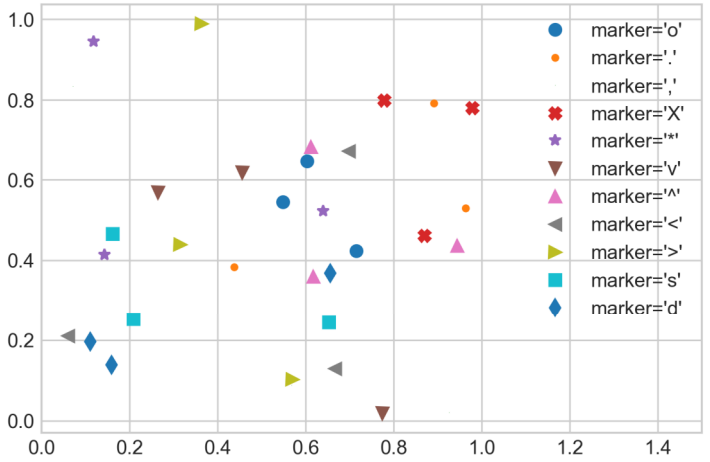


Рисунок 1.14 – Результат виконання коду в `In[18]`

Додаткові аргументи методу `plot()` зазначають широкий діапазон властивостей рядків та маркерів:

```
In [19]: plt.plot(x, np.sin(x), '-h', color='gray',
                  markersize=16, linewidth=6,
                  markerfacecolor='yellow',
```

```
                markeredgecolor='k',
                markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```

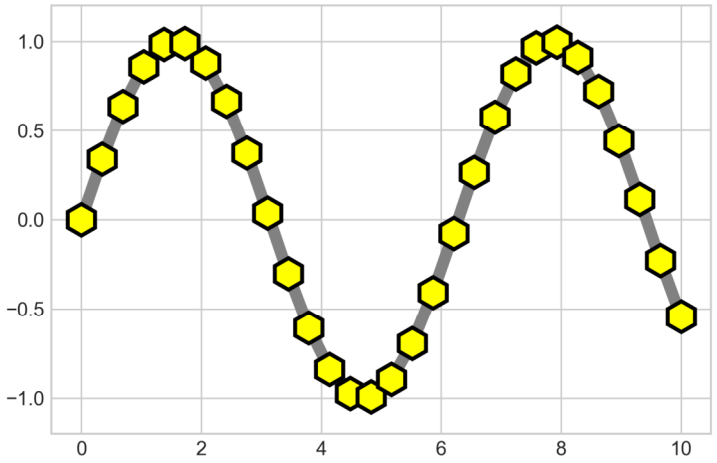


Рисунок 1.15 – Результат виконання коду в `In[19]`

Повний опис доступних опцій дивіться у документації `plt.plot` або у докстрінгу методу.

1.7.2.2. З використанням `scatter`

Другим, більш потужним методом створення графіків розсіювання є функція `plt.scatter / ax.scatter`, яка може бути використана подібною до функції `plot()`:

```
In [20]: plt.scatter(x, y)
```

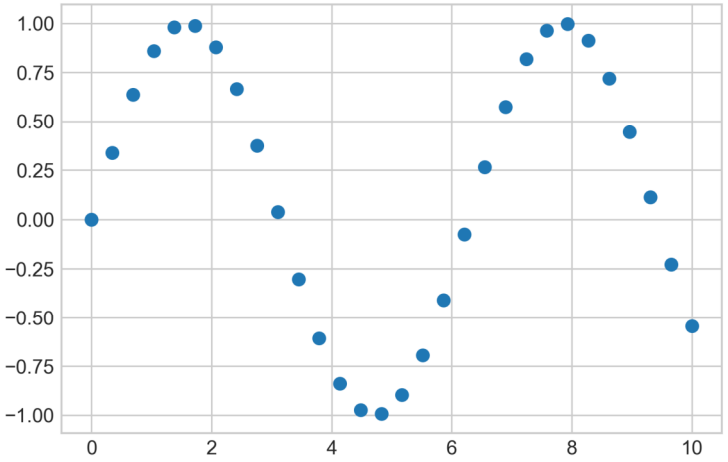



Рисунок 1.16 – Результат виконання коду в `In[20]`

Основна відмінність `scatter` від `plot` полягає в тому, що він може використовуватися для створення графіків, де властивості кожної окремо взятої точки (розмір, колір заповнення, колір контуру тощо) можуть індивідуально контролюватися (так звана бульбашкова діаграми).

Покажемо це, створивши графік точок з випадковими координатами, кольорами та розмірами. Для кращого відображення результатів, що перекриваються, ми також використовуємо ключове слово `alpha` для регулювання рівня прозорості:

```
In [21]: rng = np.random.RandomState(0)
         x = rng.randn(100)
         y = rng.randn(100)
         colors = rng.rand(100)
         sizes = 1000 * rng.rand(100)
         plt.scatter(x, y, c=colors, s=sizes,
                     alpha=0.2, cmap='ocean')
         plt.colorbar();  # показати градацію кольору
```

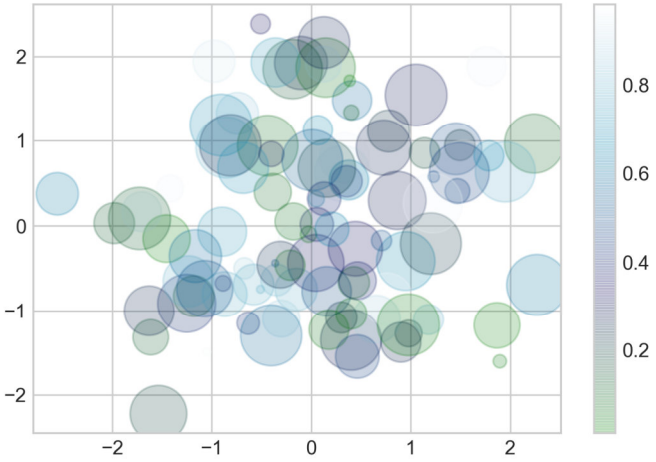


Рисунок 1.17 – Результат виконання коду в `In[21]`

Зверніть увагу, що кольоровий аргумент автоматично відображається на кольорову шкалу (показану тут командою `colorbar`()), а аргумент розміру заданий у пікселях. Таким чином, колір і розмір точок можуть використовуватися для передачі інформації для візуалізації на двовимірному графіку багатовимірних даних.

Наприклад, ми можемо використовувати дані Iris (так звані іріси Фішера) від Scikit-Learn, де кожний примірник – це один із трьох видів квітів, у яких ретельно вимірювали довжину та ширину пелюсток та чашолистків:

```
In [22]: from sklearn.datasets import load_iris
         iris = load_iris()
        features = iris.data.T
        plt.scatter(features[0], features[1],
        alpha=0.3, s=100*features[3], c=iris.target,
        cmap='viridis')
        plt.xlabel(iris.feature_names[0])
        plt.ylabel(iris.feature_names[1])
```

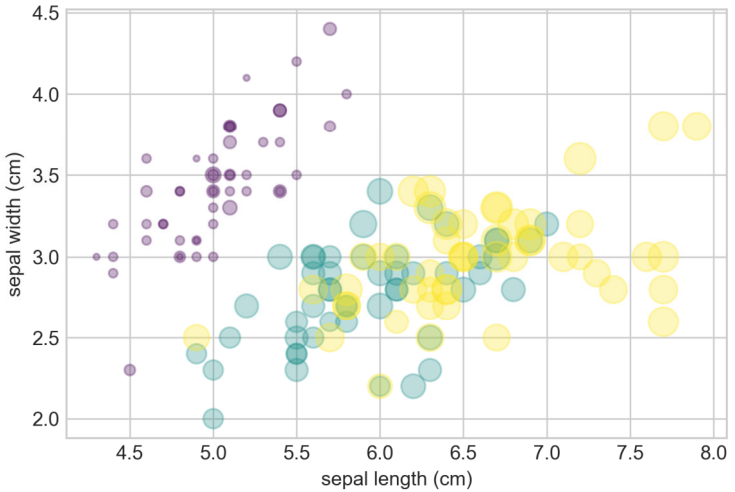



Рисунок 1.18 – Результат виконання коду в `In[22]`

Ми бачимо, що ця бульбашкова діаграма дала нам можливість одночасно досліджувати чотири різні виміри даних:

- розташування кожної точки (x, y) відповідає довжині та ширині чашолистка,
- розмір точки пов'язаний з шириною пелюстки,
- колір – з певним видом квітки.

Подібні різнокольорові та багатофункціональні діаграми можуть бути корисними як для дослідження, так і для подання даних.

1.8. Створення декількох областей рисування на одному рисунку

Для того щоб нарисувати кілька графіків поряд, нам потрібно вказати кількість рядків і стовпчиків. Для цього слід викликати метод subplots і вказати два параметри. Перший – це `nrows` (кількість бажаних рядків), а другий – `ncols` (кількість стовпчиків).

```
In [23]: x = np.linspace(0,10,100)
```

```
# figsize – встановлення розмірів рисунка
fig, axes = plt.subplots(nrows=2, ncols=3,
                         figsize=(10, 5))
for row, row_axes in enumerate(axes):
    for column, ax in enumerate(row_axes):
        ax.plot(np.sin(x*column))
        ax.set_title(f'Графік рядок {row+1}
                     стовпчик {column+1}')
fig.tight_layout()
```

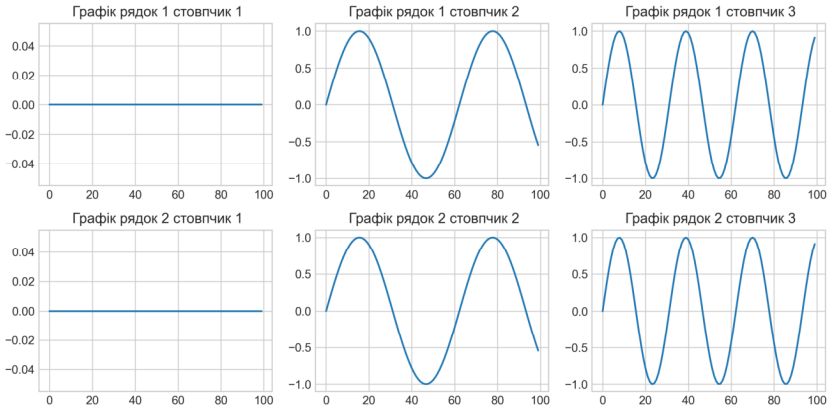


Рисунок 1.19 – Результат виконання коду в `In[23]`

2. ПОСТАНОВА ЗАДАЧІ

1. Відповідно до номера в журналі академічної групи обрати номер індивідуального завдання (таблиця 2.1).
2. Набрати рівняння за допомогою мови LaTeX в клітинці блокнота.
3. Побудувати кожну лінію на окремому графіку, розмістивши їх поруч:

– всі прямі повинні бути різного кольору, але не використовувати системну послідовність кольорів (синій, помаранчевий, зелений…);

– всі прямі повинні мати різний тип ліній (пунктирна, точка тире тощо);

– всі графіки повинні мати сумісну вісь ординат;

– кожен графік повинен мати підпис.

4. Розмістити всі лінії на одному рисунку.

– всі прямі повинні бути різного кольору, але не використовувати системну послідовність кольорів (синій, помаранчевий, зелений…);

– всі прямі повинні мати різний тип ліній (пунктирна, точка тире тощо);

– підібрати масштаб таким чином, щоб всі три точки перетину прямих були в області видимості;

– додати легенду на графік;

– додати сітку, в якій задати колір та тип ліній;

– змінити розмір рисунку (наприклад, 8×16 дюймів) та розподільчу здатність (наприклад, 100 dpi);

– зробити підписи рівнянь прямих вздовж лінії з відповідним нахилом;

– додати підписи осей та назву графіку;

– заповнити кольором область, що утворена перетином всіх прямих (використовуйте метод `fill_between`).

5. За допомогою підмодуля `numpy.linalg` знайти точки перетину всіх пар прямих, відмітити їх та зробити відповідні вказівки на графіку.
6. Зберегти отримані рисунки у форматах .jpg, .png, .svg.
7. Зробити висновки про те, чим відрізняються рисунки в кожному з форматів (розмір, наявність артефактів, вид вмісту файлу).
8. Розмістити створений блокнот на GitHub.

Таблиця 2.1

Завдання до лабораторної роботи

Вар.	Система рівнянь	Вар.	Система рівнянь
1	$4.2\,x_1 + 10\,x_2 = 136$ $-10.1\,x_1 + 13.8\,x_2 = 132$ $18.3\,x_1 - 7.6\,x_2 = 108$	**6**	$-10.3\,x_1 + 10.2\,x_2 = 70$ $4.7\,x_1 + 12.3\,x_2 = 173$ $13.2\,x_1 + 8.8\,x_2 = 282$
2	$6.2\,x_1 + 6.6\,x_2 = 83$ $-9.6\,x_1 + 13.8\,x_2 = 72$ $-13.2\,x_1 + 5.7\,x_2 = 305$	**7**	$3.2\,x_1 + 8.8\,x_2 = 89$ $-14.2\,x_1 + 10.8\,x_2 = 125$ $20.3\,x_1 - 7.2\,x_2 = 142$

3	$8.2\,x_1 + 7.9\,x_2 = 123$ $11.2\,x_1 + 16.3\,x_2 = 201$ $-16.3\,x_1 - 8.3\,x_2 = 169$	**8**	$13.1\,x_1 + 9.2\,x_2 = 173$ $8.3\,x_1 + 21\,x_2 = 271$ $3.2\,x_1 - 8.3\,x_2 = 34$
4	$10.2\,x_1 - 3.2\,x_2 = 149$ $-5.8\,x_1 + 16\,x_2 = 83$ $10.3\,x_1 + 7.3\,x_2 = 234$	**9**	$8.3\,x_1 + 8.2\,x_2 = 134$ $-10.2\,x_1 + 15.2\,x_2 = 102$ $14.5\,x_1 - 7.3\,x_2 = 141$
5	$8.1\,x_1 + 8.5\,x_2 = 72$ $-3.2\,x_1 + 15.2\,x_2 = 42$ $10.2\,x_1 + 8.8\,x_2 = 205$	**10**	$-10.2\,x_1 + 9.3\,x_2 = 73$ $5.3\,x_1 + 15.8\,x_2 = 173$ $17.2\,x_1 + 10.3\,x_2 = 159$

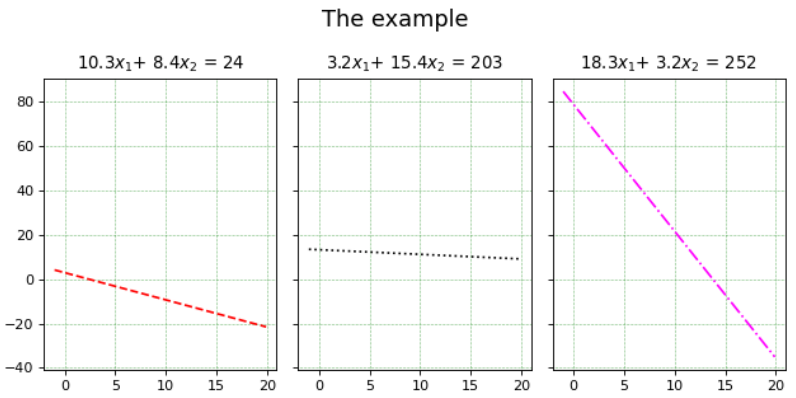
3. ЗМІСТ ЗВІТУ

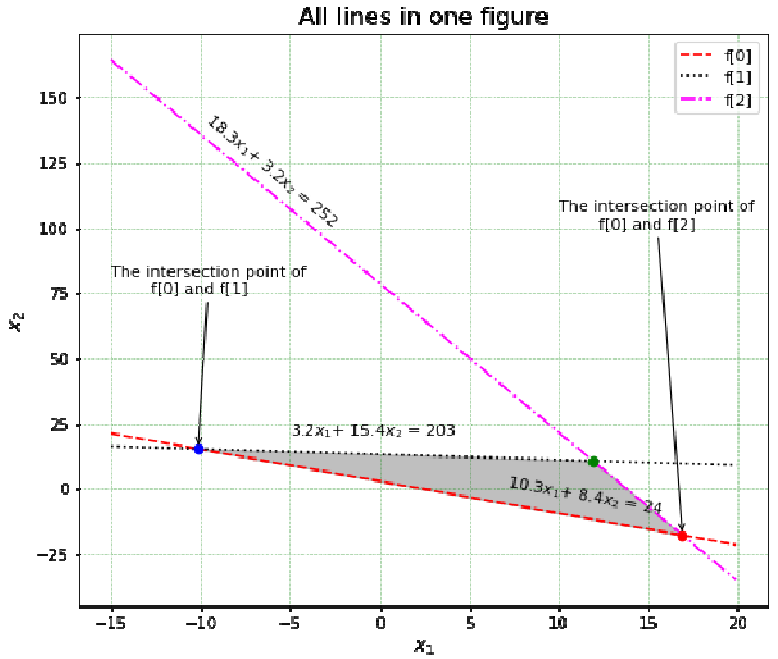
1. Титульний аркуш.
2. Тема і мета роботи.
3. Лістинги комірок.
4. Скріншоти блокноту Jupiter з вхідними та вихідними даними.
5. Посилання на створений блокнот, розміщений на Github та відображений в nbviewer.
6. Висновок.

4. ПРИКЛАД ВИКОНАНОГО ЗАВДАННЯ

Варіант № 11

$$\begin{cases} 10.3x_1 + 8.4x_2 = 24 \\ 3.2x_1 + 15.4x_2 = 203 \\ 18.3x_1 + 3.2x_2 = 252 \end{cases}$$





Контрольні запитання

1. Чи потребує бібліотека Matplotlib окремого встановлення?
2. Назвіть 2–3 стилі відображення Matplotlib, що мають сітку.
4. Охарактеризуйте основні складові об'єкта Matplotlib.
5. Які два типи інтерфейсів має Matplotlib?
6. В чому відмінність команди `%matplotlib notebook` від `%matplotlib inline`?
7. Якими способами можна задати колір лінії методі `plot`?
8. Як можна задати тип лінії в методі `plot`?
9. Чи можна створити точковий графік методом `plot`?
10. Що таке бульбашкова діаграма і які її області використання?
11. Назвіть методи для налаштування міток на рисунку.
12. Як можна розмістити декілька графіків на одному рисунку?
13. Чи є можливість використання структурного підходу для розміщення декількох графіків на одному рисунку?

СПИСОК ЛІТЕРАТУРИ

1. Сторінка завантаження Matplotlib. – Режим доступу: https://matplotlib.org/downloads.html
2. Project Jupyter. https://jupyter.org/
3. Навчіться, як використовувати LaTex в Jupyter Notebook https://towardsdatascience.com/write-markdown-latex-in-the-jupyter-notebook-10985edb91fd
4. Matplotlib Tutorials. – Режим доступу: https://matplotlib.org/stable/tutorials/index.html
5. Matplotlib: Научная графика в Python. – Режим доступу: https://pythonworld.ru/novosti-mira-python/scientific-graphics-in-python.html
6. Построение графиков в Python при помощи Matplotlib. – Режим доступу: https://python-scripts.com/matplotlib

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи

«Основи роботи з бібліотекою Matplotlib»

з курсу «Обробка даних Python»

для студентів спеціальностей 121 Інженерія програмного забезпечення,

122 Комп'ютерні науки, 124 Системний аналіз,

126 Інформаційні системи і технології

Укладачі:

КОВАЛЕНКО Світлана Миколаївна

КОВАЛЕНКО Сергій Володимирович

ШМАТКО Олександр Віталійович

Відповідальний за випуск Годлевський М. Д.

Роботу до видання рекомендував Безменов М. І.

В авторській редакції

План 2021, поз. 272

Підписано до друку 29.10.2021 р. Формат 60 × 84 1/16.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2