Тема. Matplotlib

Мета. Вивчення програмного пакету Matplotlib для візуалізації даних.

Вступ. Matplotlib – програмний пакет на мові програмування Python для візуалізації даних з використанням двовимірної 2D і тривимірної 3D графіки.

План.

1. Вступ
2. Побудова графіків
   2.1. Графіки з однією кривою
   2.2. Графіки з декількома кривими
   2.3. Побудова графіків за даними з файлів
   2.4. Побудова точкових графіків
   2.5. Побудова стовпчастих діаграм
   2.6. Побудова секторних діаграм
   2.7. Побудова триангуляцій
   2.8. Оброблення рядків даних з ключами
3. Налаштування кольору
   3.1. Задання кольору параметром `color`
   3.2. Задання кольору точок параметром `edgecolor`
   3.3. Задання кольору списком значень
   3.4. Використання карти кольорів
4. Налаштування властивостей ліній
   4.1. Властивість linestyle
   4.2. Властивість `linewidth`
5. Заповнення поверхонь щаблонами
6. Заміна точок на маркери
7. Анотації графіків
   7.1. Задання заголовку до графіка
   7.2. Задання заголовку до графіка у LaTex-стилі
   7.3. Задання надписів до кожної осі
   7.4. Добавлення тексту на рисунок
   7.5. Добавлення надпису до кривих
   7.6. Добавлення сітки до графіку
   7.7. Добавлення ліній до графіку
   7.8. Задання розбивки і значень координатних осей
   7.9. Задання розбивки і надписів координатних осей
8. Робота з рисунками
   8.1. Об’єднання рисунків
   8.2. Об’єднання фігур за допомогою команди subplot
   8.3. Однакове масштабування осей
   8.4. Задання діапазону значень осей
   8.5. Задання співвідношення осей
   8.6. Вставлення підрисунків у рисунок
   8.7. Використання логарифмічної шкали
   8.8. Використання полярних координат
9. Двовимірні масиви
   9.1. Візуалізація двовимірного масиву на прикладі множини Мандельброта

9.2. Візуалізація двовимірних скалярних полів
9.3. Візуалізація двовимірних векторних полів
9.4. Візуалізація потокових двовимірних векторних полів
10. Тривимірні поверхні
10.1. Побудова тривимірної поверхні
10.2. Побудова параметризованої тривимірної поверхні
10.3. Вбудовування двовимірного рисунка у тривимірний
10.4. Створення тривимірних стовпчикових діаграм
Висновки
Література

1. Вступ

Matplotlib – програмний пакет на мові програмування Python для візуалізації даних з використанням двовимірної 2D і тривимірної 3D графіки. Існує ряд програмних засобів для створення графіків для доповідей та презентацій. Наприклад, можна відкрити CSV-файл в LibreOffice або Google Docs і побудувати в них графіки. Але що, якщо графіки або діаграми потрібно створювати регулярно, то для цього найкраще підходить Python і його пакет Matplotlib.

Пакет Matplotlib за будовою подібний до NumPy, SciPy і IPython та надає можливості, подібні до пакету MATLAB. На даний час пакет працює з декількома графічними бібліотеками, включаючи wxWindows і PyGTK.

Пакет підтримує наступні види графіків та діаграм:

• графіки (line plot);
• діаграми розсіювання (scatter plot);
• стовпчасті діаграми (bar chart) і гістограми (histogram);
• секторні діаграми (pie chart);
• деревоподібні діаграми (stem plot);
• контурні графіки (contour plot);
• поля градієнтів (quiver);
• спектральні діаграми (spectrogram).

Користувач може вказати вісі координат, сітку, додати підписи і пояснення, використовувати логарифмічну шкалу або полярні координати.

Нескладні тривимірні графіки можна будувати з допомогою набору інструментів (toolkit) mplot3d. Існують і інші набори інструментів: для картографії, для роботи з Excel, утиліти для GTK та інші.

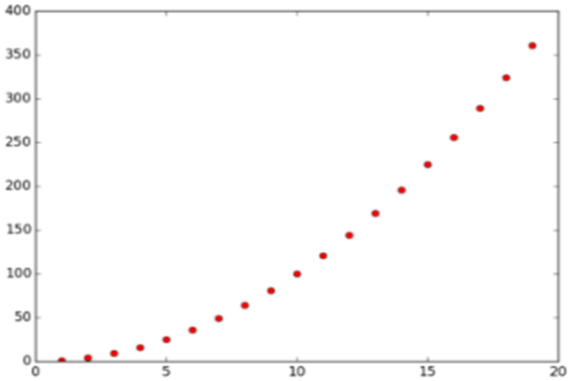
З допомогою Matplotlib можна створювати як звичайні, так і анімовані зображення.

2. Побудова графіків

2.1. Графіки з однією кривою

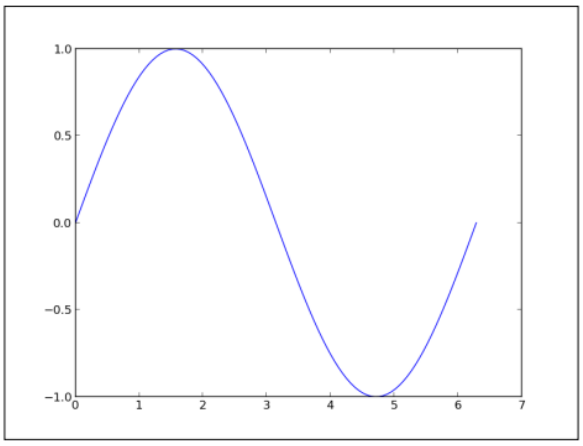
Приклад побудови простого графіка:

```
from pylab import *
plot(range(1, 20),
    [i * i for i in range(1, 20)], 'ro')
savefig('example.png')
show()
```



Приклад побудови графіка `sin(x)` з використанням стандартної бібліотеки Python і з пакету NumPy:

```
# sin_1.py
import math
import matplotlib.pyplot as plt

T = range(100)
X = [(2 * math.pi * t) / len(T) for t in T]
Y = [math.sin(value) for value in X]

plt.plot(X, Y)
plt.show()

# sin_2.py
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(0, 2 * np.pi, 100)
Y = np.sin(X)

plt.plot(X, Y)
plt.show()
```

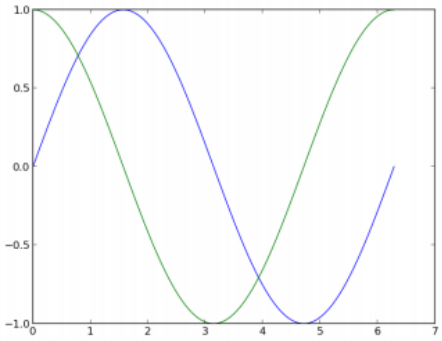


2.2. Графіки з декількома кривими

Сценарій:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
X = np.linspace(0, 2 * np.pi, 100)

Ya = np.sin(X)
Yb = np.cos(X)
plt.plot(X, Ya)
plt.plot(X, Yb)
plt.show()
```



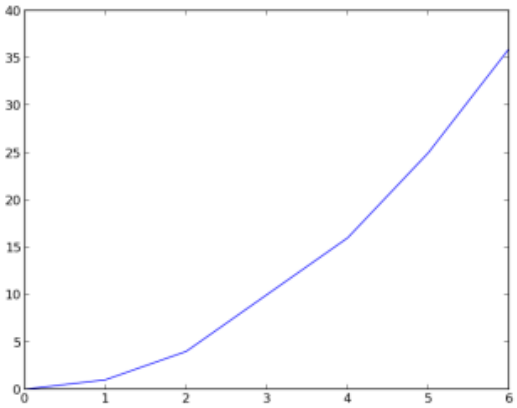
2.3. Побудова графіків за даними з файлів

Нехай наступні дані записані у файл my_data.txt:

```
0  0
1  1
2  4
4 16
5 25
6 36
```

Сценарій:

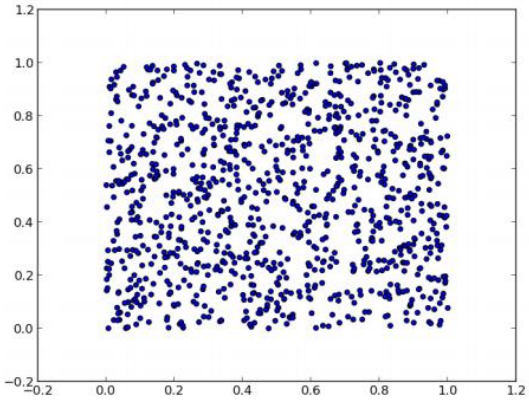
```
import matplotlib.pyplot as plt

with open('my_data.txt', 'r') as f:
  X, Y = zip(*[[float(s) for s in line.split()] for line in f])

plt.plot(X, Y)
plt.show()
```



2.4. Побудова точкових графіків

Для виведення незалежних даних використовуються точкові графіки. Приклад виведення 1024 точок з випадковими координатами:

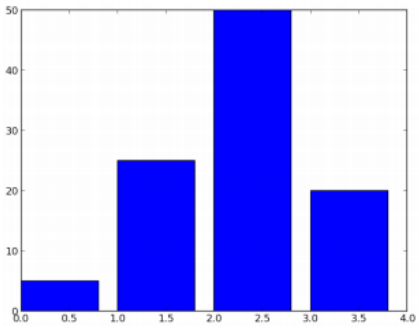
```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.rand(1024, 2)

plt.scatter(data[:,0], data[:,1])
plt.show()
```



2.5. Побудова стовпчастих діаграм

Сценарій:

```
import matplotlib.pyplot as plt

data = [5., 25., 50., 20.]

plt.bar(range(len(data)), data)
plt.show()
```

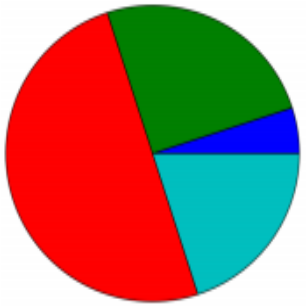


2.6. Побудова секторних діаграм

Сценарій:

```
import matplotlib.pyplot as plt

data = [5, 25, 50, 20]
```

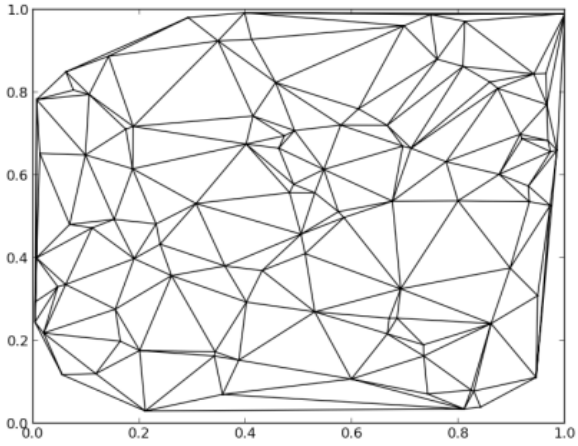
```
plt.pie(data)
plt.show()
```



2.7. Побудова тріангуляцій

Тріангуляція використовується для покриття областей нерегулярними трикутними сітками. Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.tri as tri

data = np.random.rand(100, 2)

triangles = tri.Triangulation(data[:,0], data[:,1])

plt.triplot(triangles)
plt.show()
```



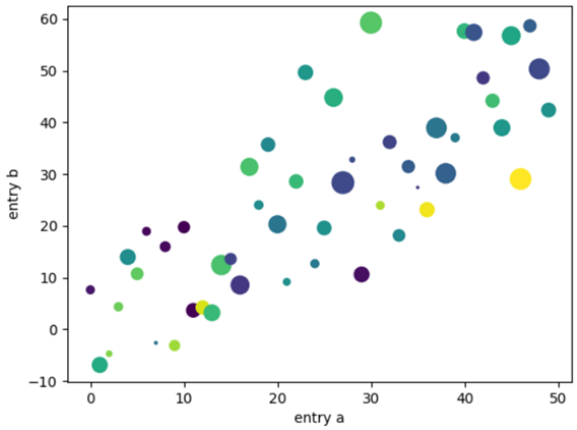
2.8. Оброблення рядків даних з ключами

Існують формати, які підтримують порядковий доступ до змінних, наприклад, `numpy.recarray` або `pandas.DataFrame`.

Matplotlib дозволяє створювати такі об'єкти з використанням ключового слова `data` і генерувати графіки з рядками, що відповідають цим ключам. Приклад:

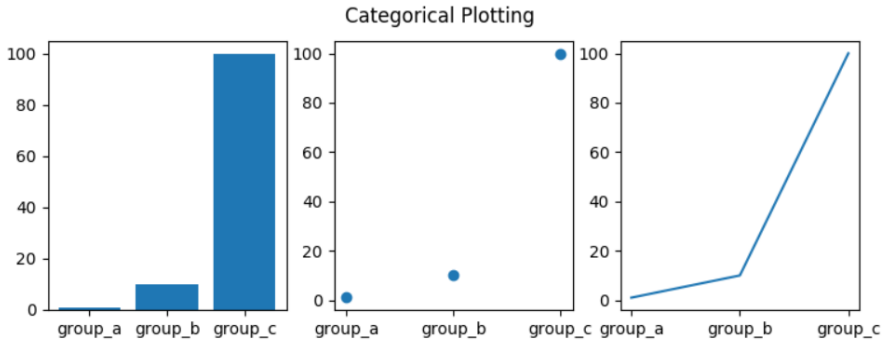
```
data = {'a': np.arange(50),
        'c': np.random.randint(0,50,50),
        'd': np.random.randn(50)}
```

```
data['b'] = data['a'] + 10*np.random.randn(50)
data['d'] = np.abs(data['d'])*100

plt.scatter('a','b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



Також можна створювати декілька підграфіків з використанням змінних категорій. Matplotlib передає змінні категорій безпосередньо у функцій побудови графіків, наприклад:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
plt.figure(figsize=(9,3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.subtitle('Categorucal Plotting')
plt.show()
```



3. Налаштування кольору

Кольори графіків можна задавати наступними способами:

• *Трійками*: три числа задають значення червоного, зеленого і синього (RGB). Числа мають значення в інтервалі [0,1].

• *Четвірками*: три числа мають те саме значення як і у трійок, а четверте число задає прозорість (значення також в інтервалі [0,1]).

• *Зарезервовані імена кольорів*: Matplotlib використовує стандартні імена кольорів HTML. Деякі кольори позначаються одною буквою:

b – синій
g – зелений
r – червоний
c – блакитний
m – пурпуровий
y – жовтий
k – чорний
w – білий

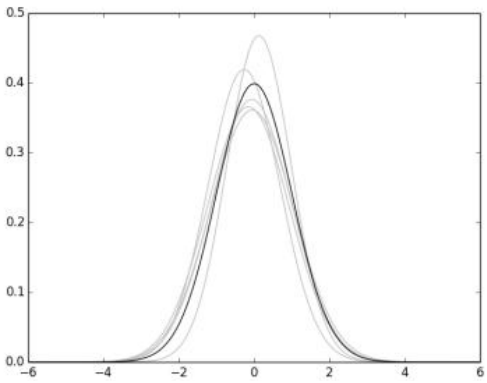
• *HTML стрічки кольорів*: #RRGGBB, де RR, GG, BB 8-бітні значення у шістнадцятковому поданні.

• *стрічки для подання відтінків сірого:* задається число з плаваючою крапкою в інтервалі [0.0- 1.0].

3.1. Задання кольору параметром `color`

Сценарій:

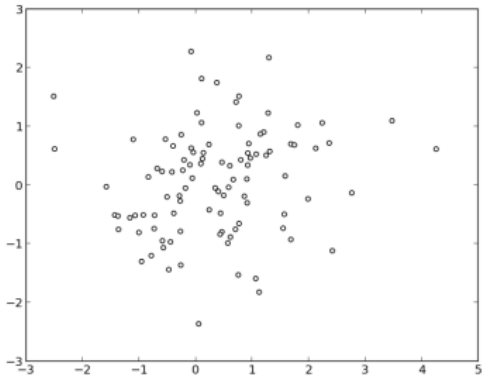
```
import numpy as np
import matplotlib.pyplot as plt

def pdf(X, mu, sigma):
  a = 1. / (sigma * np.sqrt(2. * np.pi))
  b = -1. / (2. * sigma ** 2)
  return a * np.exp(b * (X - mu) ** 2)

X = np.linspace(-6, 6, 1000)

for i in range(5):
  samples = np.random.standard_normal(50)
  mu, sigma = np.mean(samples), np.std(samples)
  plt.plot(X, pdf(X, mu, sigma), color = '.75')

plt.plot(X, pdf(X, 0., 1.), color = 'k')
plt.show()
```



3.2. Задання кольору точок параметром `edgecolor`

Сценарій:

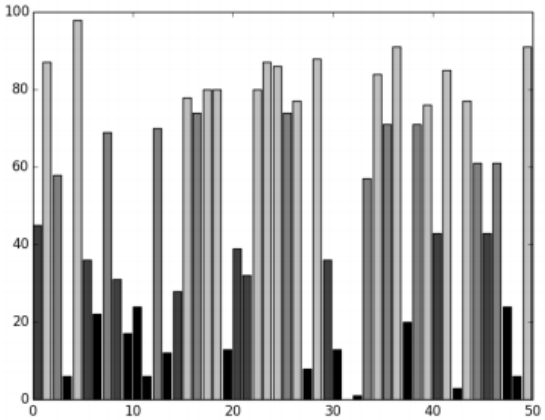
```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.standard_normal((100, 2))

plt.scatter(data[:,0], data[:,1], color = '1.0', edgecolor='0.0')
plt.show()
```



3.3. Задання кольору списком значень

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

values = np.random.random_integers(99, size = 50)

color_set = ('.00', '.25', '.50', '.75')
color_list = [color_set[(len(color_set) * val) // 100] for val in
  values]
plt.bar(np.arange(len(values)), values, color = color_list)
plt.show()
```

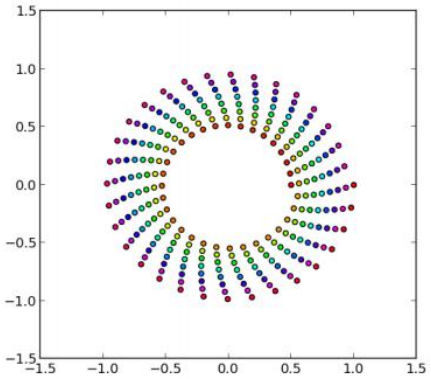


3.4. Використання карти кольорів

Для задання кольорів може використовуватися карта кольорів, яка визначена у модулі `matplotlib.cm`. Кольорова карта визначає кольори як неперервну функцією від одної змінної.

Сценарій:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

N = 256
angle  = np.linspace(0, 8 * 2 * np.pi, N)
radius = np.linspace(.5, 1., N)

X = radius * np.cos(angle)
Y = radius * np.sin(angle)

plt.scatter(X, Y, c = angle, cmap = cm.hsv)
plt.show()
```



4. Налаштування властивостей ліній

Стиль лінії задається властивістю `linestyle`, яка може мати наступні значення:

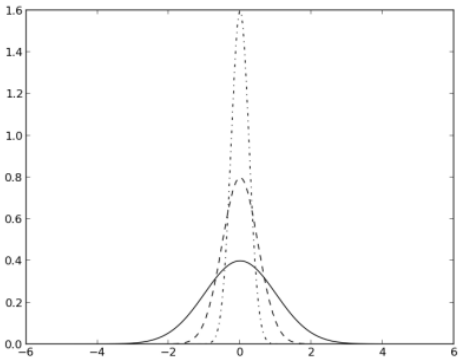
- solid (суцільна);
- dashed (пунктирна);
- dotted (крапкова);
- dashdot (пунктирно-крапкова).

4.1. Властивість linestyle

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

def pdf(X, mu, sigma):
  a = 1. / (sigma * np.sqrt(2. * np.pi))
  b = -1. / (2. * sigma ** 2)
  return a * np.exp(b * (X - mu) ** 2)
```

```
X = np.linspace(-6, 6, 1024)

plt.plot(X, pdf(X, 0., 1.),   color = 'k', linestyle = 'solid')
plt.plot(X, pdf(X, 0.,  .5),  color = 'k', linestyle = 'dashed')
plt.plot(X, pdf(X, 0.,  .25), color = 'k', linestyle = 'dashdot')

plt.show()
```

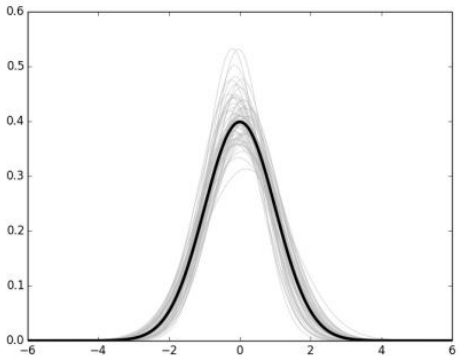


4.2. Властивість `linewidth`

Товщина лінії задається властивістю `linewidth`.

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

def pdf(X, mu, sigma):
  a = 1. / (sigma * np.sqrt(2. * np.pi))
  b = -1. / (2. * sigma ** 2)
  return a * np.exp(b * (X - mu) ** 2)

X = np.linspace(-6, 6, 1024)
for i in range(64):
  samples = np.random.standard_normal(50)
  mu, sigma = np.mean(samples), np.std(samples)
  plt.plot(X, pdf(X, mu, sigma), color = '.75', linewidth = .5)

plt.plot(X, pdf(X, 0., 1.), color = 'y', linewidth = 3.)
plt.show()
```



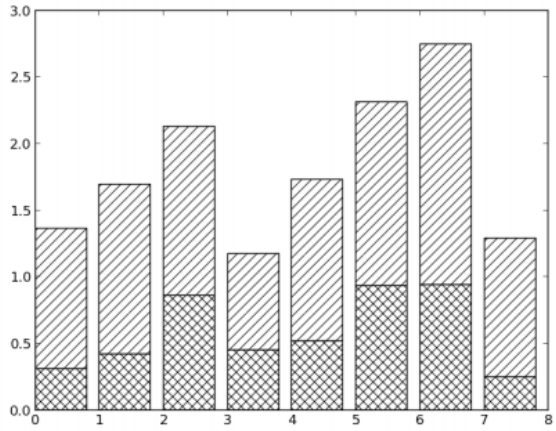
5. Заповнення поверхонь шаблонами

Заповнення поверхонь шаблонами задається властивістю `hatch`, яка може мати наступні значення:

- /
- \
- |
- -
- +
- x
- o
- O
- .
- *

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

N = 8
A = np.random.random(N)
B = np.random.random(N)
X = np.arange(N)

plt.bar(X, A, color = 'w', hatch = 'x')
plt.bar(X, A + B, bottom = A, color = 'w', hatch = '/')

plt.show()
```



6. Заміна точок на маркери

Точки на графіках можна замінити на маркери потрібної форми заданням властивості `marker`. Маркери можна задати наступними способами:

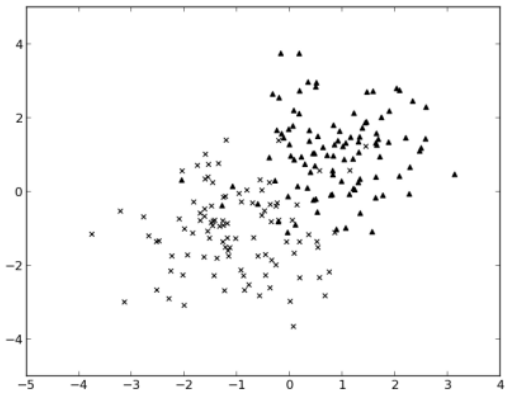
- заданням числа в інтервалі [0-8] для вибору наперед визначених маркерів;
- списком координат вузлів маркера;

• описанням регулярного багатокутника, як трійки (N, 0, α), де N – сторонній багатокутник, α – кут повернення);

• описанням початкового багатокутника, як трійки (N, 1, α), де N – сторонній багатокутник, α – кут повернення).

Сценарій:

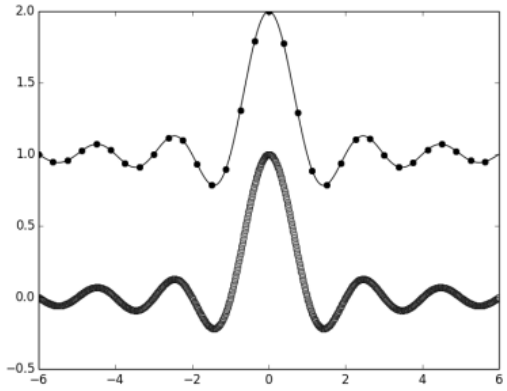
```
import numpy as np
import matplotlib.pyplot as plt

A = np.random.standard_normal((100, 2))
A += np.array((-1, -1))
B = np.random.standard_normal((100, 2))
B += np.array((1, 1))

plt.scatter(A[:,0], A[:,1], color = 'k', marker = 'x')
plt.scatter(B[:,0], B[:,1], color = 'k', marker = '^')
plt.show()
```



Параметр `markevery` дозволяє застосувати один маркер для кожних N точок:

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-6, 6, 1024)
Y1 = np.sinc(X)
Y2 = np.sinc(X) + 1

plt.plot(X, Y1, marker = 'o', color = '.75')
plt.plot(X, Y2, marker = 'o', color = 'k', markevery = 32)

plt.show()
```



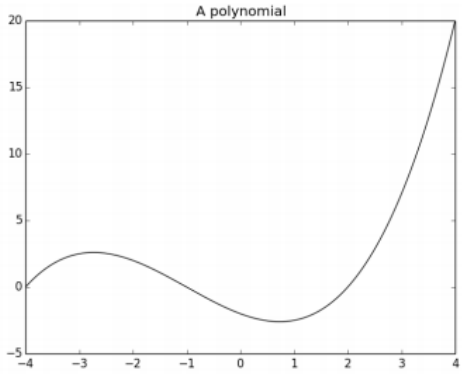
7. Анотація графіків

Анотація графіків дозволяє:
• задати заголовок;
• задати позначки осей;
• задати текст і надписи до кривих;
• задати сітку і лінії;
• задати надписи і розмір розбивки осей;

7.1. Задання заголовку до графіка

Сценарій:

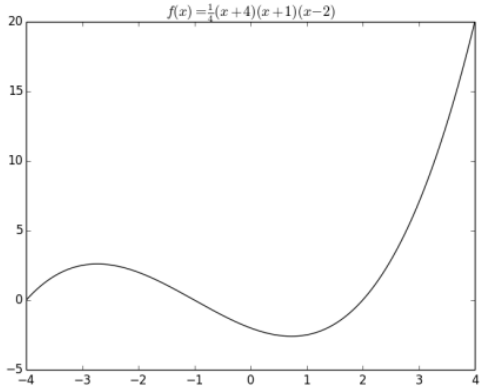
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)

plt.title('A polynomial')
plt.plot(X, Y, c = 'k')

plt.show()
```



7.2. Задання заголовку до графіка у LaTex-стилі

Сценарій:

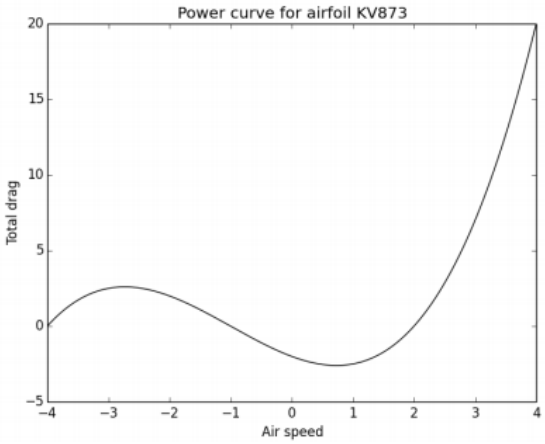
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)

plt.title('$f(x)=\\frac{1}{4}(x+4)(x+1)(x-2)$')
plt.plot(X, Y, c = 'k')
plt.show()
```



7.3. Задання надписів до кожної осі

Сценарій:

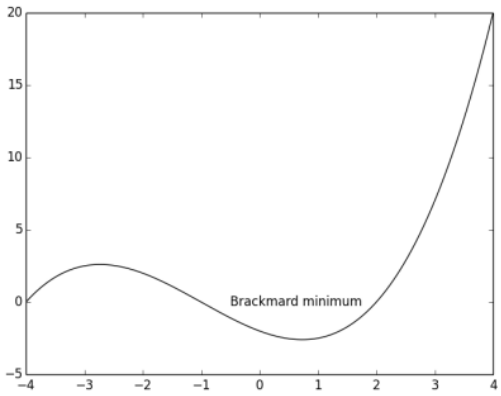
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)

plt.title('Power curve for airfoil KV873')
plt.xlabel('Air speed')
plt.ylabel('Total drag')

plt.plot(X, Y, c = 'k')
plt.show()
```



7.4. Добавлення тексту на рисунок

Сценарій:

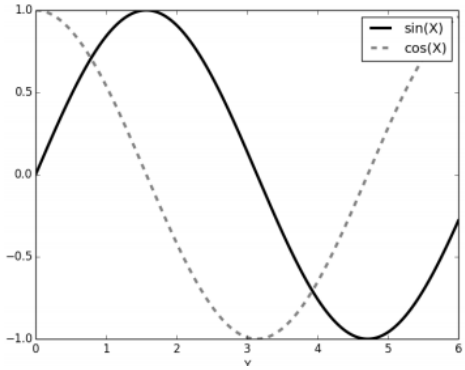
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)

plt.text(-0.5, -0.25, 'Brackmard minimum')

plt.plot(X, Y, c = 'k')
plt.show()
```



7.5. Добавлення надпису до кривих

Сценарій:

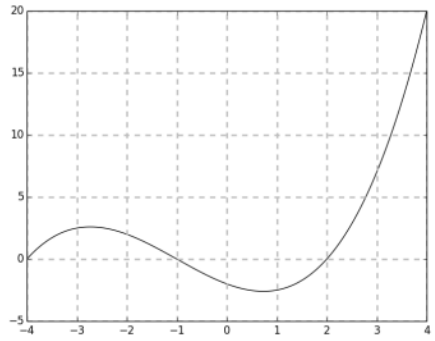
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(0, 6, 1024)
Y1 = np.sin(X)
Y2 = np.cos(X)

plt.xlabel('X')
plt.ylabel('Y')

plt.plot(X, Y1, c = 'k',  lw = 3.,              label = 'sin(X)')
plt.plot(X, Y2, c = '.5', lw = 3., ls = '--', label = 'cos(X)')

plt.legend()
plt.show()
```



7.6. Добавлення сітки до графіку

Сценарій:

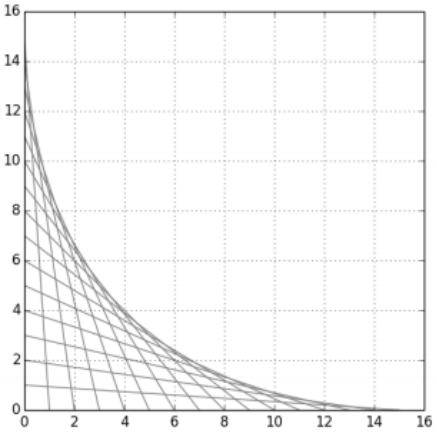
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)

plt.plot(X, Y, c = 'k')
plt.grid(True, lw = 2, ls = '--', c = '.75')
plt.show()
```



7.7. Добавлення ліній до графіку

Сценарій:

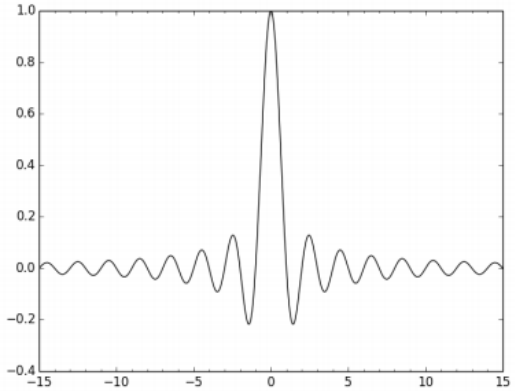
```
import matplotlib.pyplot as plt

N = 16
for i in range(N):
  plt.gca().add_line(plt.Line2D((0, i), (N - i, 0), color = '.75'))

plt.grid(True)
plt.axis('scaled')
plt.show()
```



7.8. Задання розбивки і значень координатних осей

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

X = np.linspace(-15, 15, 1024)
Y = np.sinc(X)

ax = plt.axes()
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))

plt.plot(X, Y, c = 'k')
plt.show()
```

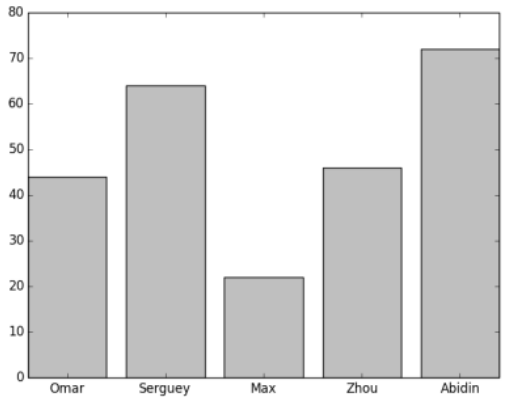


7.9. Задання розбивки і надписів координатних осей

Сценарій:

```
import numpy as np
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt

name_list = ('Omar', 'Serguey', 'Max', 'Zhou', 'Abidin')
```

```
value_list = np.random.randint(0, 99, size = len(name_list))
pos_list = np.arange(len(name_list))

ax = plt.axes()
ax.xaxis.set_major_locator(ticker.FixedLocator((pos_list)))
ax.xaxis.set_major_formatter(ticker.FixedFormatter((name_list)))

plt.bar(pos_list, value_list, color = '.75', align = 'center')
plt.show()
```



8. Робота з рисунками

Робота з рисунками дозволяє:
• об’єднувати декілька рисунків;
• однаково масштабувати осі;
• задавати діапазони осей;
• задавати співвідношення сторін;
• вставляти підрисунки;
• використовувати логарифмічну шкалу;
• використовувати полярні координати.

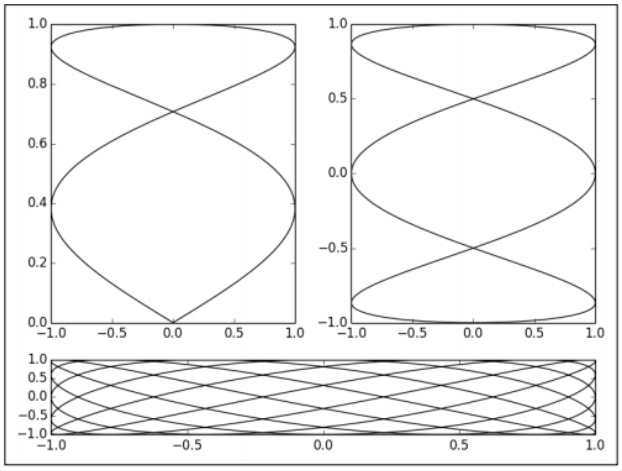
8.1. Об’єднання рисунків

Для об’єднання декількох рисунків на одному рисунку використовується метод `subplot2grid()`, який має чотири параметри:
- кортеж, який задає сітку з R стовпців і C рядків;
- кортеж, який задає координату комірки в сітці з R стовпців і C рядків;
- кількість рядків, які будуть об’єднуватися – `rowspan`;
- кількість стовпців, які будуть об’єднуватися – `colwspan`.

Сценарій:

```
import numpy as np
from matplotlib import pyplot as plt

T = np.linspace(-np.pi, np.pi, 1024)

grid_size = (4, 2)
```

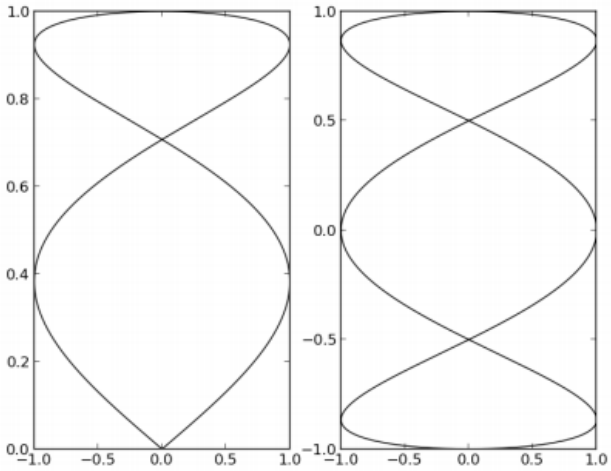
```
plt.subplot2grid(grid_size, (0, 0), rowspan = 3, colspan = 1)
plt.plot(np.sin(2 * T), np.cos(0.5 * T), c = 'k')

plt.subplot2grid(grid_size, (0, 1), rowspan = 3, colspan = 1)
plt.plot(np.cos(3 * T), np.sin(T), c = 'k')

plt.subplot2grid(grid_size, (3, 0), rowspan=1, colspan=3)

plt.plot(np.cos(5 * T), np.sin(7 * T), c= 'k')

plt.tight_layout()
plt.show()
```



8.2. Об’єднання фігур за допомогою команди `subplot`

Сценарій:

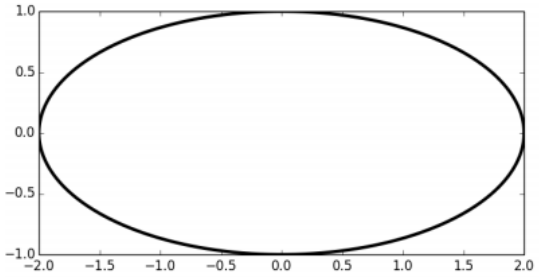
```
import numpy as np
from matplotlib import pyplot as plt

T = np.linspace(-np.pi, np.pi, 1024)

fig, (ax0, ax1) = plt.subplots(ncols =2)
ax0.plot(np.sin(2 * T), np.cos(0.5 * T), c = 'k')
ax1.plot(np.cos(3 * T), np.sin(T), c = 'k')

plt.show()
```



8.3. Однакове масштабування осей

Сценарій:

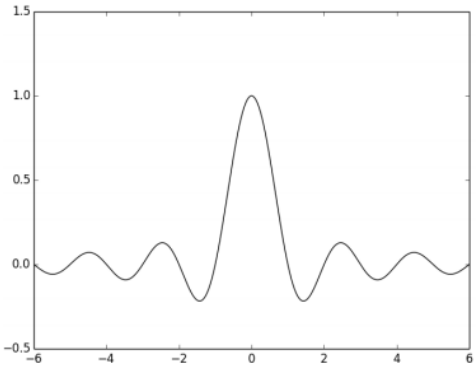
```
import numpy as np
import matplotlib.pyplot as plt

T = np.linspace(0, 2 * np.pi, 1024)

plt.plot(2. * np.cos(T), np.sin(T), c = 'k', lw = 3.)
plt.axes().set_aspect('equal')
plt.show()
```



8.4. Задання діапазону значень осей

Сценарій:

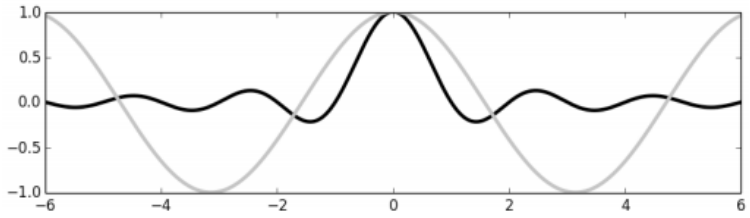
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-6, 6, 1024)

plt.ylim(-.5, 1.5)
plt.plot(X, np.sinc(X), c = 'k')
plt.show()
```



8.5. Задання співвідношення осей

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-6, 6, 1024)
Y1, Y2 = np.sinc(X), np.cos(X)

plt.figure(figsize=(10.24, 2.56))
plt.plot(X, Y1, c='k', lw = 3.)
plt.plot(X, Y2, c='.75', lw = 3.)

plt.show()
```

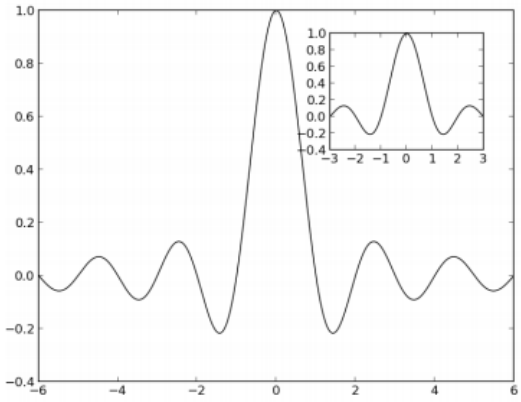


8.6. Вставлення підрисунків у рисунок

Сценарій:

```
import numpy as np
from matplotlib import pyplot as plt

X = np.linspace(-6, 6, 1024)
Y = np.sinc(X)

X_detail = np.linspace(-3, 3, 1024)
Y_detail = np.sinc(X_detail)

plt.plot(X, Y, c = 'k')
sub_axes = plt.axes([.6, .6, .25, .25])
sub_axes.plot(X_detail, Y_detail, c = 'k')
plt.setp(sub_axes)
```

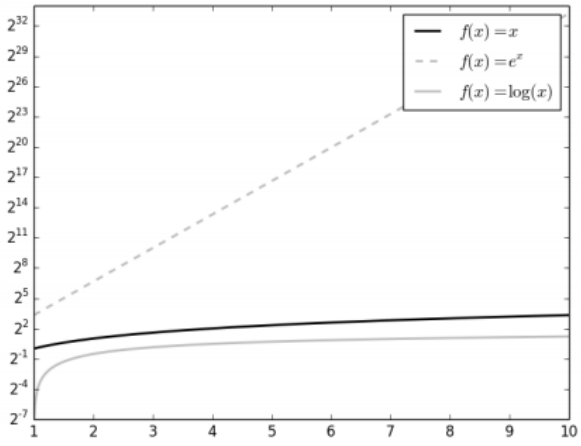
```
plt.show()
```



8.7. Використання логарифмічної шкали

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(1, 10, 1024)

plt.yscale('log')
plt.plot(X, X, c = 'k', lw = 2., label = r'$f(x)=x$')
plt.plot(X, 10 ** X, c = '.75', ls = '--', lw = 2., label =
r'$f(x)=e^x$')
plt.plot(X, np.log(X), c = '.75', lw = 2., label = r'$f(x)=\log(x)$')

plt.legend()
plt.show()
```

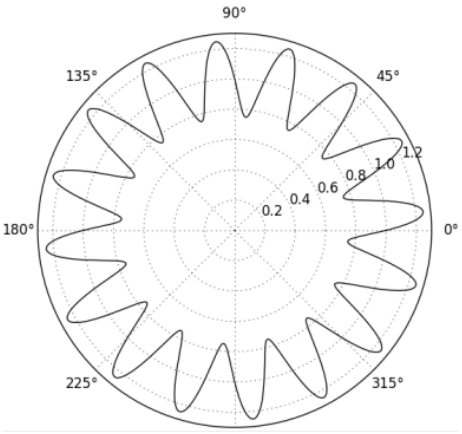


8.8. Використання полярних координат

Сценарій:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
T = np.linspace(0 , 2 * np.pi, 1024)

plt.axes(polar = True)
plt.plot(T, 1. + .25 * np.sin(16 * T), c= 'k')

plt.show()
```

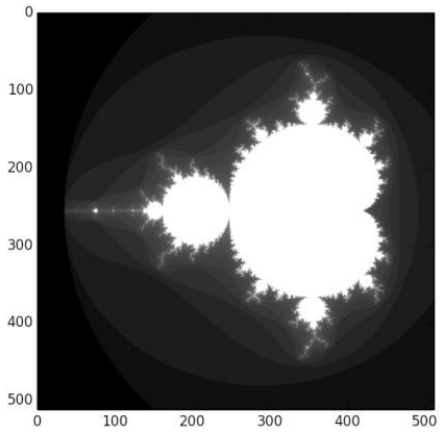


9. Двовимірні масиви

9.1. Візуалізація двовимірного масиву на прикладі множини Мандельброта

Сценарій:

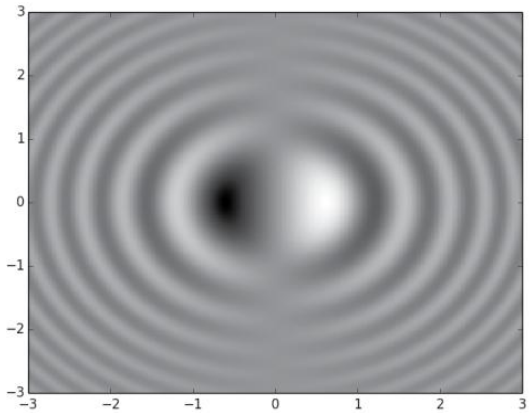
```
import numpy as np
import matplotlib.cm as cm
from matplotlib import pyplot as plt

def iter_count(C, max_iter):
  X = C
  for n in range(max_iter):
    if abs(X) > 2.:
      return n
    X = X ** 2 + C
  return max_iter

N = 512
max_iter = 64
xmin, xmax, ymin, ymax = -2.2, .8, -1.5, 1.5
X = np.linspace(xmin, xmax, N)
Y = np.linspace(ymin, ymax, N)
Z = np.empty((N, N))

for i, y in enumerate(Y):
  for j, x in enumerate(X):
    Z[i, j] = iter_count(complex(x, y), max_iter)

plt.imshow(Z, cmap = cm.gray)
plt.show()
```



9.2. Візуалізація двовимірних скалярних полів

Сценарій:

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.cm as cm

n = 256
x = np.linspace(-3., 3., n)
y = np.linspace(-3., 3., n)
X, Y = np.meshgrid(x, y)

Z = X * np.sinc(X ** 2 + Y ** 2)

plt.pcolormesh(X, Y, Z, cmap = cm.gray)
plt.show()
```

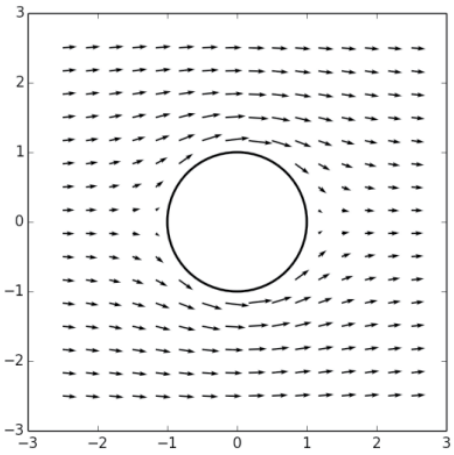


9.3. Візуалізація двовимірних векторних полів

Сценарій:

```
import numpy as np
import sympy
from sympy.abc import x, y
from matplotlib import pyplot as plt
import matplotlib.patches as patches
```

```
def cylinder_stream_function(U = 1, R = 1):
  r = sympy.sqrt(x ** 2 + y ** 2)
  theta = sympy.atan2(y, x)
  return U * (r - R ** 2 / r) * sympy.sin(theta)

def velocity_field(psi):
  u = sympy.lambdify((x, y), psi.diff(y), 'numpy')
  v = sympy.lambdify((x, y), -psi.diff(x), 'numpy')
  return u, v

U_func, V_func = velocity_field(cylinder_stream_function() )

xmin, xmax, ymin, ymax = -2.5, 2.5, -2.5, 2.5
Y, X = np.ogrid[ymin:ymax:16j, xmin:xmax:16j]
U, V = U_func(X, Y), V_func(X, Y)

M = (X ** 2 + Y ** 2) < 1.
U = np.ma.masked_array(U, mask = M)
V = np.ma.masked_array(V, mask = M)

shape = patches.Circle((0, 0), radius = 1., lw = 2., fc = 'w', ec
  = 'k', zorder = 0)
plt.gca().add_patch(shape)

plt.quiver(X, Y, U, V, zorder = 1)

plt.axes().set_aspect('equal')
plt.show()
```

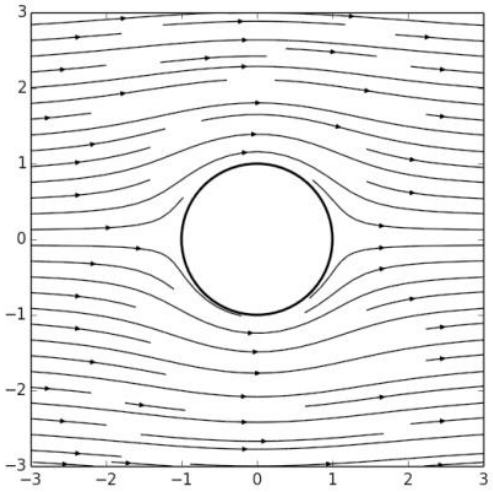


9.4. Візуалізація потокових двовимірних векторних полів

Сценарій:

```
import numpy as np
import sympy
from sympy.abc import x, y
from matplotlib import pyplot as plt
import matplotlib.patches as patches

def cylinder_stream_function(U = 1, R = 1):
  r = sympy.sqrt(x ** 2 + y ** 2)
  theta = sympy.atan2(y, x)
```

```
  return U * (r - R ** 2 / r) * sympy.sin(theta)

def velocity_field(psi):
  u = sympy.lambdify((x, y), psi.diff(y), 'numpy')
  v = sympy.lambdify((x, y), -psi.diff(x), 'numpy')
  return u, v

psi = cylinder_stream_function()
U_func, V_func = velocity_field(psi)

xmin, xmax, ymin, ymax = -3, 3, -3, 3
Y, X = np.ogrid[ymin:ymax:128j, xmin:xmax:128j]
U, V = U_func(X, Y), V_func(X, Y)

M = (X ** 2 + Y ** 2) < 1.
U = np.ma.masked_array(U, mask = M)
V = np.ma.masked_array(V, mask = M)

shape = patches.Circle((0, 0), radius = 1., lw = 2., fc = 'w', ec
  = 'k', zorder = 0)
plt.gca().add_patch(shape)

plt.streamplot(X, Y, U, V, color = 'k')

plt.axes().set_aspect('equal')
plt.show()
```



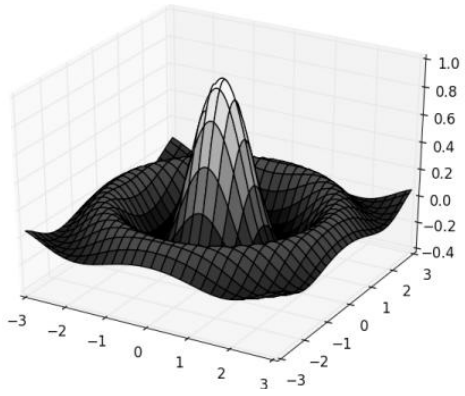
10. Тривимірні поверхні

10.1. Побудова тривимірної поверхні

Сценарій:

```
import numpy as np
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 256)
y = np.linspace(-3, 3, 256)
```

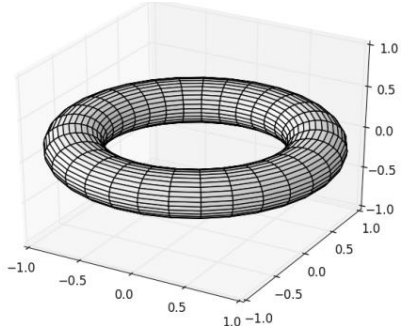
```
X, Y = np.meshgrid(x, y)
Z = np.sinc(np.sqrt(X ** 2 + Y ** 2))

fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.plot_surface(X, Y, Z, cmap=cm.gray)
plt.show()
```



10.2. Побудова параметризованої тривимірної поверхні

Сценарій:

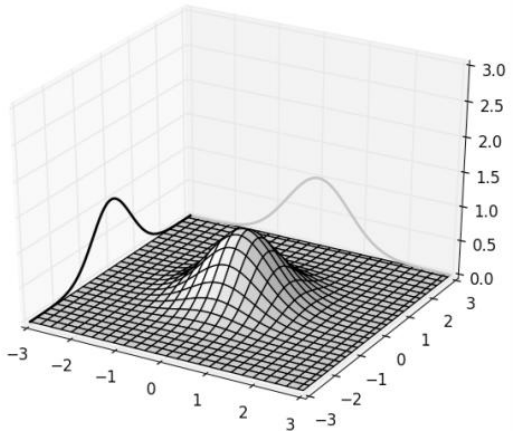
```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# Generate torus mesh
angle = np.linspace(0, 2 * np.pi, 32)
theta, phi = np.meshgrid(angle, angle)
r, R = .25, 1.
X = (R + r * np.cos(phi)) * np.cos(theta)
Y = (R + r * np.cos(phi)) * np.sin(theta)
Z = r * np.sin(phi)

# Display the mesh
fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.set_xlim3d(-1, 1)
ax.set_ylim3d(-1, 1)
ax.set_zlim3d(-1, 1)
ax.plot_surface(X, Y, Z, color = 'w', rstride = 1, cstride = 1)
plt.show()
```



10.3. Вбудовування двовимірного рисунка у тривимірний

Сценарій:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 256)
y = np.linspace(-3, 3, 256)
X, Y = np.meshgrid(x, y)
Z = np.exp(-(X ** 2 + Y ** 2))
u = np.exp(-(x ** 2))

fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.set_zlim3d(0, 3)
ax.plot(x, u, zs=3, zdir='y', lw = 2, color = '.75')
ax.plot(x, u, zs=-3, zdir='x', lw = 2., color = 'k')
ax.plot_surface(X, Y, Z, color = 'w')

plt.show()
```

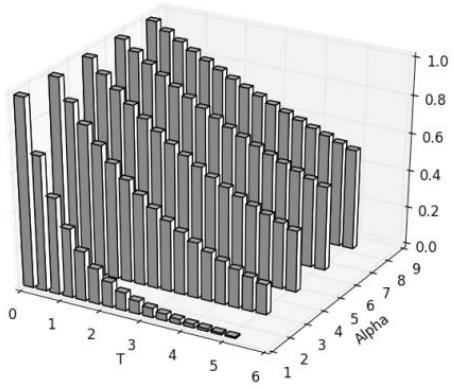


10.4. Створення тривимірних стовпчикових діаграм

Сценарій:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

# Data generation
alpha = np.linspace(1, 8, 5)
t = np.linspace(0, 5, 16)
T, A = np.meshgrid(t, alpha)
data = np.exp(-T * (1. / A))

# Plotting
fig = plt.figure()
```

```
ax = fig.gca(projection = '3d')

Xi = T.flatten()
Yi = A.flatten()
Zi = np.zeros(data.size)

dx = .25 * np.ones(data.size)
dy = .25 * np.ones(data.size)
dz = data.flatten()

ax.set_xlabel('T')
ax.set_ylabel('Alpha')
ax.bar3d(Xi, Yi, Zi, dx, dy, dz, color = 'w')

plt.show()
```



Висновки.

Matplotlib є гнучким, легко конфігуровуваним пакетом, який разом із NumPy, SciPy і IPython надає можливості подібні до MATLAB. Пакет підтримує багато видів графіків і діаграм: графіки, діаграми розсіювання, стовпчасті діаграми, секторні діаграми, діаграми “стовбур-листя”, контурні графіки, поля градієнтів, спектральні діаграми.

Література.

1. Duncan M. McGreggor. Mastering matplotlib. – Packt Publishing, 2015. – 292 p.

2. Alexandre Devert. Matplotlib Plotting Cookbook. – Packt Publishing, 2014. – 222 p.

3. Sandro Tosi. Matplotlib for Python Developers. – Packt Publishing, 2009. – 308 p.

4. Shai Vaingast. Beginning Python Visualization: Crafting Visual Transformation Scripts. – Springer, 2009. — 384 c.

5. https://matplotlib.org

Запитання.

1. Призначення пакету Matplotlib.
2. Побудова графіків з однією і декількома кривими.
3. Побудова точкових графіків.
4. Побудова стовпчастих діаграм.
5. Побудова секторних діаграм.
6. Побудова триангуляцій.
7. Налаштування кольору.
8. Налаштування властивостей ліній.

9. Анотація графіків.
10. Робота з рисунками.
11. Задання логарифмічних і полярних координат.
12. Візуалізація двовимірних масивів.
13. Тривимірні поверхні.