МІНІСТЕРСТВА ОСВІТИ І НАУКІ УКРАЇНИ
«ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

С.Ю. Борю

Вибрані пакети мови програмування python

навчально-методичний посібник для студентів природничих спеціальностей

Затверджено вченою радою ЗНУ Протокол № ___ від _____

Запоріжжя 2025

УДК:

ББК:

Борю С.Ю. Вибрані пакети мови програмування python. Навчально-методичний посібник для студентів природничо-наукових спеціальностей – Запоріжжя; ЗНУ, 2025, – 284 с.

Розглядаються популярні модулі та пакети системи програмування Python. На ведеться велика кількість прикладів та фрагментів програм, знайдених на сайтах, присвячених системі програмування Пітон та адаптованих до версії 3.х. Посібник орієнтований на самостійну роботу студентів природничих спеціальностей.

Рецензент – Кеберле Н.Г.

Відповідальний за випуск – Матвейшена Н.В.

Вибрані пакети мови програмування python

Зміст

Модулі та пакети в Python

Встановлення python-пакетів за допомогою pip

pip - це система керування пакетами, яка використовується для встановлення та керування програмними пакетами, написаними на Python. З Python версії 3.4, pip поставляється разом з інтерпретатором python.

Початок роботи

Спробуємо за допомогою pip встановити якийсь пакет, наприклад, numpy (https://pythonworld.ru/numpy) у консолі (в режимі адміністратора):

```
Linux:
sudo pip3 install numpy

Windows:
pip3 install numpy
```

Може виникнути помилка: "pip3" не є внутрішньою чи зовнішньою командою, яку виконує програма або пакетний файл.

Тоді необхідно вказувати повний шлях до програми, наприклад (якщо Пітон встановлений у C: Python36):

```
C:\Python36\Tools\Scripts\pip3.exe install numpy
```

Або додавати папку C:\Python36\Tools\Scripts\ в PATH вручну.

Або зробивши цю папку з pip3 поточної:

cd “C:\Python36\Tools\Scripts\”

```
pip3 install numpy
```

В останніх версіях python виклик pip зручно виконувати з консолі:

```
python -m pip <command> [options]
```

Що ще вміє робити

Основні команди pip3:

pip help - Допомога по доступним командам.

pip install package_name - Встановлення пакета(ів).

pip uninstall package_name - Видалення пакета(ів).

pip list - Перелік встановлених пакетів.

pip show package_name - Показує інформацію про встановлений пакет.

pip search - Пошук пакетів на ім'я.

pip --proxy user: passwd@proxy.server :port - Використання з проксі.

pip install -U - Оновлення пакета(ів).

pip install --force-reinstall - при оновленні, інсталювати пакет, навіть якщо він останньої версії.

Приклад перевірки, встановлення та переустановки пакета matplotlib

```
"C:\Program Files\Python36-32\Scripts\pip" install -U matplotlib

Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC
v.1900 32 bit (Intel)] on win32
Тип "copyright", "credits" або "license()" for more information.
>>> import matplotlib as mpl
>>> print ('Current version on matplotlib library is',
mpl.__version__)

Current version on matplotlib library is 2.1.1
>>>
# де-факто стандарт виклику pyplot у python
import matplotlib.pyplot as plt
```

Модулі в Python

Система модулів дозволяє логічно організувати ваш код Python. Групування коду в модулі значно полегшує процес написання та розуміння програми. Модуль Python це просто файл, що містить код Python. Кожен модуль Python може містити змінні, оголошення класів і функцій. Крім того, у модулі може знаходитися виконуваний код.

Кожна програма може імпортувати модуль та отримати доступ до його класів, функцій та об'єктів. Потрібно помітити, що модуль може бути написаний не лише на Python, а, наприклад, на C або C++.

Підключення модуля зі стандартної бібліотеки

Підключити модуль можна за допомогою інструкції import. Наприклад, підключимо модуль os для отримання поточної директорії:

```
>>> import os
>>> os.getcwd()
'C:\\Python33'
```

Після ключового слова import вказується назва модуля. Однією інструкцією можна підключити кілька модулів, хоча цього не рекомендується робити, оскільки це знижує читання коду. Імпортуємо модулі time та random.

```
>>>
>>> import time,random
>>> time.time()
1376047104.056417
>>> random.random()
0.9874550833306869
```

Після імпортування модуля його назва стає змінною, через яку можна отримати доступ до атрибутів модуля. Наприклад, можна звернутися до константи e, розташованої в модулі math:

```
>>>
>>> import math
>>> math.e
2.718281828459045
```

Варто зазначити, що якщо зазначений атрибут модуля не буде знайдено, збудеться виняток AttributeError. Якщо ж не вдасться знайти модуль для імпортування, то ImportError.

```
>>>
>>> import notexist
Traceback (most recent call last):
  File "", line 1, in
    importnotexist
ImportError: No module named 'notexist'
>>> import math
>>> math.
Traceback (most recent call last):
  File "", line 1, in
    math.
AttributeError: 'module' object has no attribute 'E'
```

Використання псевдонімів

Якщо назва модуля занадто довга, або вона вам не подобається з якихось інших причин, то для нього можна створити псевдонім за допомогою ключового слова as.

```
>>>
>>> import mathasm
>>> me
2.718281828459045
```

Тепер доступ до всіх атрибутів модуля math здійснюється тільки за допомогою змінної m, а змінної math у цій програмі вже не буде (якщо, звичайно, ви після цього не напишете import math, тоді модуль буде доступний як під ім'ям m, так і під ім'ям math).

Інструкція від

Підключити певні атрибути модуля можна за допомогою інструкції від. Вона має кілька форматів:

```
from<Назва модуля>import<Атрибут 1> [as<Псевдонім 1> ],
[<Атрибут 2> [as<Псевдонім 2> ] ...]

from<Назва модуля>import*
```

Перший формат дозволяє підключити з модуля лише вказані вами атрибути. Для довгих імен можна призначити псевдонім, вказавши його після ключового слова as.

```
>>>
>>> from mathimporte, ceilasc
>>> e
2.718281828459045
>>> c(4.6)
5
```

Атрибути, що імпортуються, можна розмістити на декількох рядках, якщо їх багато (для кращої читання коду):

```
>>>
>>> from mathimport(sin, cos,
...           tan, atan)
```

Другий формат інструкції from дозволяє підключити всі (точніше, майже всі) змінні з модуля. Наприклад імпортуємо всі атрибути з модуля sys:

```
>>>
>>> from sysimport *
>>> version
'3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43)
[MSC v.1600 32 bit (Intel)]'
>>> version_info
sys.version_info(major=3, minor=3, micro=2,
releaselevel='final', serial=0)
```

Слід зазначити, що не всі атрибути імпортуватимуться. Якщо в модулі визначено змінну __all__ (список атрибутів, які можуть бути підключені), то будуть підключені лише атрибути цього списку. Якщо змінна __all__ не визначена, то буде підключено всі атрибути, що не починаються з нижнього підкреслення.

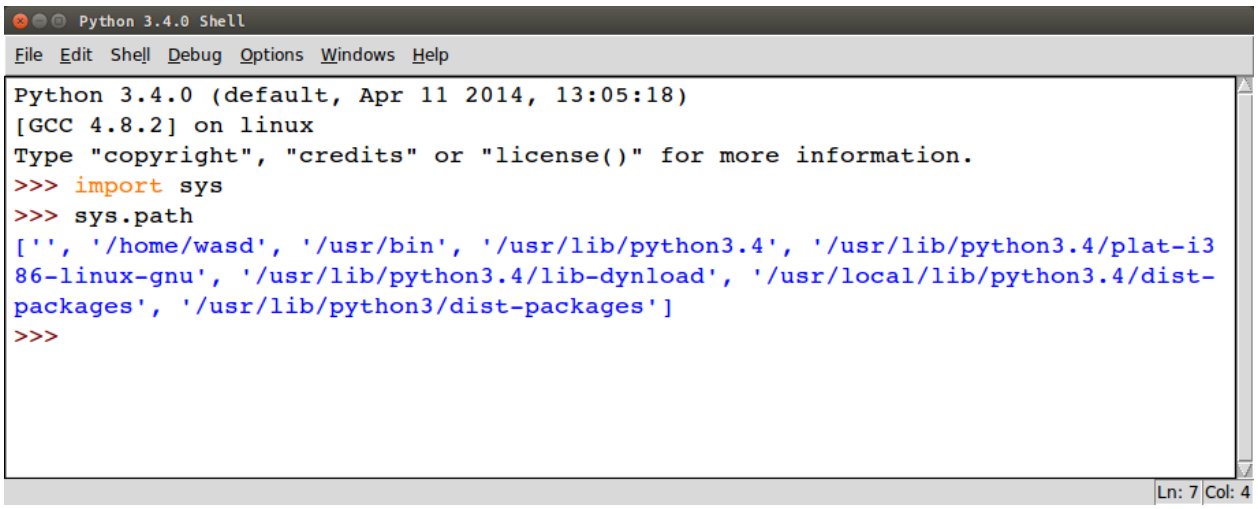
Крім того, необхідно враховувати, що імпортування всіх атрибутів з модуля може порушити простір імен головної програми, оскільки змінні, що мають однакові імена, будуть перезаписані.

Місцезнаходження модулів у Python:

Коли ви імпортуєте модуль, інтерпретатор Python шукає цей модуль у таких місцях:

1. Директорія, в якій знаходиться файл, у якому викликається команда імпорту
2. Якщо модуль не знайдено, Python шукає у кожній директорії, визначеній у консольній змінній PYTHONPATH.
3. Якщо і там модуль не знайдено, Python перевіряє шлях, заданий за замовчуванням.

Шлях пошуку модулів збережено в системному модулі sys змінної path. Змінна sys.path містить усі три вищеописані місця пошуку модулів.

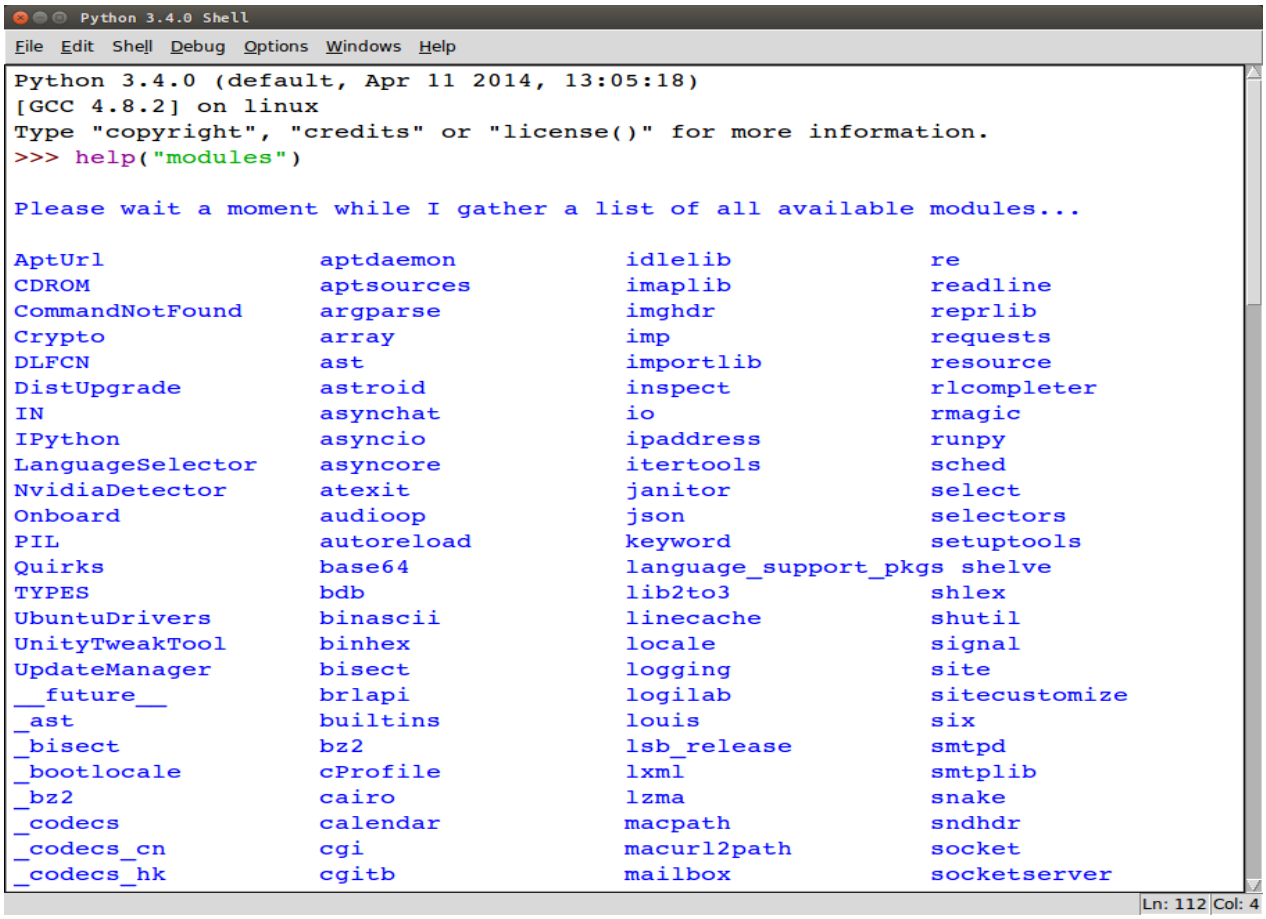


Отримання списку всіх модулів Python, встановлених на комп'ютері

Для того, щоб отримати список усіх модулів, встановлених на вашому комп'ютері, достатньо виконати команду:

```
help("modules")
```

За кілька секунд ви отримаєте список усіх доступних модулів.



Створення свого модуля в Python:

Щоб створити свій модуль у Python, достатньо зберегти ваш скрипт з розширенням .py Тепер він доступний в будь-якому іншому файлі.

Наприклад, створимо два файли: module_1.py та module_2.py і збережемо їх в одній директорії. У першому запишемо:

```
defhello():
    print("Hello from module_1")
```

А в другому викличемо цю функцію:

```
frommodule_1 import hello

hello()
```

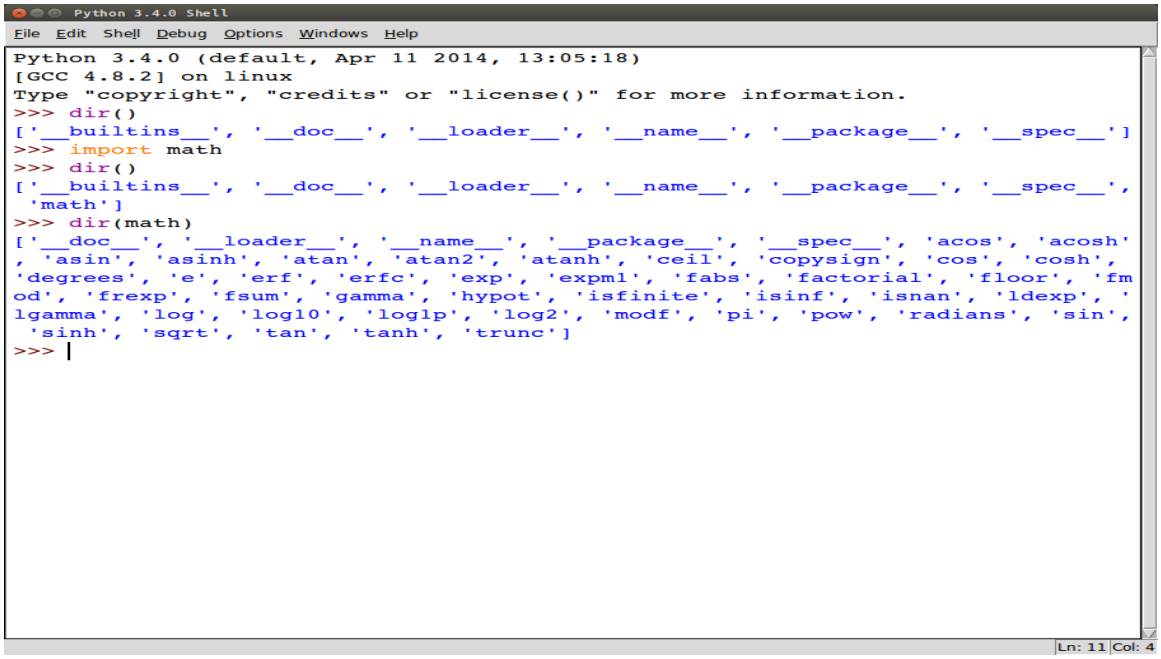
Виконавши код другого файлу отримаємо:

```
Hello from module_1
```

Функція dir():

Вбудована функція dir() повертає відсортований список рядків, що містять усі імена, визначені у модулі.

```
# на даний момент нам доступні лише вбудовані функції
dir()
# Імпортуємо модуль math
importmath
# тепер модуль math у списку доступних імен
dir()
# отримаємо імена, визначені в модулі math
dir(math)
```



Архітектура програми на Python

Код Python може бути організований таким чином:

1. Перший рівень - це звичайні команди на Python.

2. Команди на Python можуть бути зібрані у функції.
3. Функції може бути частиною класу.
4. Класи можна визначити всередині модулів.
5. Нарешті, модулі можуть складатися пакети модулів.

Пакети модулів у Python

Окремі файли-модулі з кодом Python можуть об'єднуватися в пакети модулів. Пакет це директорія (папка), що містить кілька окремих файлів-скриптів.

Наприклад, маємо таку структуру:

```
|_ my_file.py
|_ my_package
         |_ __init__.py
         |_ inside_file.py
```

У файлі inside_file.py визначено певну функцію foo. Тоді, щоб отримати доступ до функції foo, у файлі my_file слід виконати наступний код:

```
frommy_package.inside_file import foo
```

Також зверніть увагу на наявність усередині директорії my_package файлу __init__.py. Це може бути порожній файл, який повідомляє Python, що ця директорія є пакетом модулів. У Python 3.3 і вище включати файл __init__.py в пакет модулів стало необов'язково, проте рекомендується робити це задля підтримки зворотної сумісності.

I. NumPy

NumPy – це розширення мови Python, що додає підтримку великих багатовимірних масивів та матриць, разом із великою бібліотекою високорівневих математичних функцій для операцій із цими масивами.

NumPy початок роботи

NumPy — це бібліотека мови Python, яка додає підтримку великих багатовимірних масивів і матриць, разом із великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій із цими масивами.

Встановлення NumPy

Linux:

```
sudo pip3 install numpy
```

Windows:

```
pip3 install numpy
```

Цікаве посилання https://pyprog.pro/reference_manual.html

Найважливіші атрибути

Основним об'єктом NumPy є однорідний багатовимірний масив (numpy називається numpy.ndarray). Це багатовимірний масив елементів (зазвичай чисел), одного типу.

Найбільш важливі атрибути об'єктів ndarray:

ndarray.ndim - Число вимірювань (частіше їх називають "осі") масиву.

ndarray.shape - Розміри масиву, його форма. Це кортеж натуральних чисел, що показує довжину масиву кожної осі. Для матриці з n рядків та m стовпів, shape буде (n,m). Число елементів кортежу shape дорівнює ndim.

ndarray.size - Кількість елементів масиву. Очевидно, дорівнює добутку всіх елементів атрибуту shape.

ndarray.dtype - Об'єкт, що описує тип елементів масиву. Можна визначити dtype за допомогою стандартних типів даних Python. NumPy тут надає цілий букет можливостей, як вбудованих, наприклад: bool_, character, int8, int16, int32, int64, float8, float16, float32, float64, complex64, object_, і можливість визначити власні типи даних, зокрема і складові.

ndarray.itemsize - Розмір кожного елемента масиву в байтах.

ndarray.data - Буфер, що містить фактичні елементи масиву. Зазвичай не потрібно використовувати цей атрибут, тому що звертатися до елементів масиву найпростіше за допомогою індексів.

Створення масивів

У NumPy є багато способів створити масив.

Один з найпростіших – створити масив зі звичайних списків або кортежів Python. Для цього використовується функція numpy.array() (array - функція, що створює об'єкт типу ndarray):

```
>>>
>>> import numpy as np
>>> a =np.array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> type(a)
<class 'numpy.ndarray'>
```

Функція array() трансформує вкладені послідовності багатомірні масиви. Тип елементів масиву залежить від типу елементів вихідної послідовності (але можна і перевизначити його на момент створення).

```
>>>
>>> b =np.array([[1.5, 2, 3], [4, 5, 6]])
>>> b
array([[ 1.5, 2. , 3. ],
       [4., 5., 6.]])
```

Можна також перевизначити тип у момент створення:

```
>>>
>>> b = np.array([[1.5, 2, 3], [4, 5, 6]],
dtype=np.complex)
>>> b
array([[ 1.5+0.j, 2.0+0.j, 3.0+0.j],
       [ 4.0+0.j, 5.0+0.j, 6.0+0.j]])
```

Функція array() не єдина функція створення масивів. Зазвичай елементи масиву спочатку невідомі, а масив, у якому зберігатимуться, вже потрібний. Тому є кілька функцій для того, щоб створювати масиви з якимось вихідним вмістом (за замовчуванням тип масива, що створюється — float64).

Функція

zeros()створює масив з нулів,

а функція

ones()- Масив з одиниць.

Обидві функції приймають кортеж з розмірами та аргумент dtype:

```
>>>
>>> np.zeros((3, 5))
```

```
array([[ 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [ 0., 0., 0., 0., 0.]])
>>> np.ones((2, 2, 2))
array([[[ 1., 1.],
        [1., 1.]],

       [[ 1., 1.],
        [1., 1.]]])
```

Функція eye() створює одиничну матрицю (двовимірний масив)

```
>>>
>>> np.eye(5)
array([[ 1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [ 0., 0., 0., 0., 1.]])
```

Функція

empty()створює масив без заповнення. Вихідний вміст випадково і залежить від стану пам'яті на момент створення масиву (тобто від сміття, що в ній зберігається):

```
>>>
>>> np.empty((3, 3))
array([[ 6.93920488e-310, 6.93920488e-310, 6.93920149e-310],
       [6.93920058e-310, 6.93920058e-310, 6.93920058e-310],
       [6.93920359e-310, 0.00000000e+000, 6.93920501e-310]])
>>> np.empty((3, 3))
array([[ 6.93920488e-310, 6.93920488e-310, 6.93920147e-310],
       [6.93920149e-310, 6.93920146e-310, 6.93920359e-310],
       [6.93920359e-310, 0.00000000e+000, 3.95252517e-322]])
```

Повний формат виклику функцій:

```
numpy.zeros(shape, dtype = float, order = 'C')
numpy.ones(shape, dtype = float, order = 'C')
numpy.empty(shape, dtype = float, order = 'C')
```

де:

shape - Обов'язковий аргумент, кортеж необхідної розмірності масиву;

dtype - опціональний аргумент, тип елементів масиву, за промовчанням float;

order - опціональний аргумент, рядок, спосіб представлення даних масиву в пам'яті, два можливі значення 'C' і 'F' - "як у C" або "як у фортрані".

Для створення послідовностей чисел, NumPy є функція

arange(),аналогічна до вбудованої в Python range(), тільки замість списків вона повертає масиви, і приймає не тільки цілі значення:

```
>>>
```

```
>>> np.arange(10, 30, 5)
array([10, 15, 20, 25])
>>> np.arange(0, 1, 0.1)
array([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9])
```

Повний формат виклику функції:

numpy.arange([start,] stop[, step,], dtype = None),

де:

start - опціональний аргумент, початкове значення інтервалу, за умовчанням дорівнює 0;

stop - обов'язковий аргумент, кінцеве значення інтервалу, що не входить до самого інтервалу, інтервал замикає значення stop - step;

step - опціональний аргумент, крок ітерації, різницю між кожним наступним та попереднім значеннями інтервалу, за умовчанням дорівнює 1;

dtype - тип елементів масиву, за замовчуванням None, у разі тип елементів визначається за типом переданих функції аргуметів start, stop.

Масив може бути створений за допомогою функції numpy.linspace().

Взагалі, при використанні arange() з аргументами типу float, складно бути впевненим у тому, скільки елементів буде отримано (через обмеження точності чисел із плаваючою комою). Тому в таких випадках зазвичай краще використовувати функцію:

linspace(),яка замість кроку в якості одного з аргументів приймає число, що дорівнює кількості потрібних елементів:

```
>>>
>>> np.linspace(0, 2, 9) #9 чисел від 0 до 2 включно
array([0., 0.25, 0.5, 0.75, 1., 1.25, 1.5, 1.75, 2.])
```

Повний формат виклику функції:

numpy.linspace(start, stop, num = 50, endpoint = True, retstep = False),

де:

start - Обов'язковий аргумент, перший член послідовності елементів масиву;

stop - Обов'язковий аргумент, останній член послідовності елементів масиву;

num - опціональний аргумент, кількість елементів масиву, за умовчанням дорівнює 50;

endpoint - Опціональний аргумент, логічне значення за умовчанням True. Якщо передано True, то stop є останнім елементом масиву. Якщо встановлено False, то послідовність елементів формується від start до stop для num + 1 елементів, при цьому повертається масив останній елемент не входить;

```
>>> d = np.linspace (1.0, 6.0, 5, endpoint = False)
```

```
>>> d
array([ 1., 2., 3., 4., 5.])
```

retstep - Опціональний аргумент, логічне значення за замовчуванням False. Якщо передано True то, функція повертає кортеж із двох членів, перший - масив, послідовність елементів, другий - число, збільшення між елементами послідовності.

```
>>> f = np.linspace(.5, -.5, 5, retstep = True)
>>> f
(array([ 0.5 , 0.25, 0. , -0.25, -0.5 ]), -0.25)
```

fromfunction():застосовує функцію до всіх комбінацій індексів

```
>>>
>>> def f1(i, j):
... return 3 * i + j
...
>>> np.fromfunction(f1, (3, 4))
array([[ 0., 1., 2., 3.],
       [3., 4., 5., 6.],
       [6., 7., 8., 9.]])

>>> np.fromfunction(f1, (3, 3))
array([[ 0., 1., 2.],
       [3., 4., 5.],
       [6., 7., 8.]])
```

Ну і останній з цих способів за допомогою функцій numpy.zeros_like(), numpy.ones_like(), numpy.empty_like().

Обов'язковий аргумент, що приймається функціями - масив, функції повертають масив такої ж структури, що містить, відповідно, нулі, одиниці та те, що виявилося в пам'яті на момент створення масиву.

```
>>> d
array([ 1., 2., 3., 4., 5.])

>>> dz = np.zeros_like(d)
>>> dz
array([0., 0., 0., 0., 0.])

>>> do = np.ones_like(d)
>>> do
array([ 1., 1., 1., 1., 1.])

>>> de = np.empty_like(d)
>>> de
```

```
array([ 2.24122267e+201, 6.32526950e-317,
                                    0.00000000e+000,
         2.24686637e+201, 6.34874355e-321])
```

Повний формат виклику функцій:

numpy.zeros_like(a)
numpy.ones_like(a)
numpy.empty_like(a)

де:
a - Обов'язковий аргумент, масив, структуру якого необхідно повторити;

Масиви можна використовувати у різних ітераціях:

```
>>> for i in a:
... print(i)
...
0.1
0.2
0.3
0.4
0.5

>>> for i in a3:
... for j in i:
... print(j)
...
[1 2]
[3 4]
[5 6]
[7 8]
[9, 10]
[11 12]
```

Друк масивів

Якщо масив занадто великий, щоб друкувати, NumPy автоматично приховує центральну частину масиву і виводить тільки його куточки.

```
>>>
>>> print(np.arange(0, 3000, 1))
[ 0 1 2 ..., 2997 2998 2999]
```

Якщо вам справді потрібно побачити весь масив, використовуйте функцію
numpy.set_printoptions:

```
np.set_printoptions(threshold=np.nan)
```

І взагалі, за допомогою цієї функції можна налаштувати друк масивів "під себе".

Функція numpy.set_printoptions приймає кілька аргументів:

precision : кількість цифр, що відображаються після коми (за замовчуванням 8).

threshold : кількість елементів масиву, що викликає обрізання елементів (за замовчуванням 1000).

edgeitems : кількість елементів на початку та наприкінці кожної розмірності масиву (за замовчуванням 3).

linewidth : кількість символів у рядку, після якого здійснюється перенесення (за умовчанням 75).

suppress : якщо True, не друкує невеликі значення в non-scientific notation (за замовчуванням False).

nanstr : строкове представлення NaN (за умовчанням 'nan')

infstr : строкове представлення inf (за умовчанням 'inf')

formatter : дозволяє більш тонко керувати печаткою масивів. Тут не розглядаємо - дивіться https://docs.scipy.org/doc/numpy/reference/generated/numpy.set_printoptions.html

Використовуйте офіційну документацію по numpy - https://docs.scipy.org/doc/numpy/reference/.

NumPy, базові операції над масивами

Математичні операції над масивами виконуються поелементно. Створюється новий масив, що заповнюється результатами дії оператора.

```
>>>
>>> import numpy as np
>>> a =np.array([20, 30, 40, 50])
>>> b =np.arange(4)
>>> a+b
array([20, 31, 42, 53])
>>> a-b
array([20, 29, 38, 47])
>>> a*b
array([0, 30, 80, 150])
>>> a/b  # При розподілі на 0 повертається inf (нескінченність)
array([ inf, 30. , 20. , 16.66666667])
<string>:1: RuntimeWarning: divide by zero об'єднаний в true_divide
>>> a**b
array([1, 30, 1600, 125000])
>>> a%b  # При взятті залишку від поділу на 0 повертається 0
<string>:1: RuntimeWarning: divide by zero об'єднаний в remainder
array([0, 0, 0, 2])
```

Для цього, природно, масиви мають бути однакових розмірів.

```
>>>
>>> c =np.array([[1, 2, 3], [4, 5, 6]])
>>> d =np.array([[1, 2], [3, 4], [5, 6]])
>>> c+d
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: Operands could not be broadcast together
with shapes (2,3) (3,2)
```

Також можна проводити математичні операції між масивом та числом. У цьому випадку до кожного елемента додається (або що ви там робите) це число.

```
>>>
>>> a + 1
array([21, 31, 41, 51])
>>> a ** 3
array([8000, 27000, 64000, 125000])
>>> a < 35  # І фільтрацію можна проводити
array([True, True, False, False], dtype=bool)
```

NumPy також надає безліч математичних операцій для обробки масивів:

```
>>>
>>> np.cos(a)
array([0.40808206, 0.15425145, -0.66693806,
0.96496603])
>>> np.arctan(a)
array([1.52083793, 1.53747533, 1.54580153, 1.55079899])
>>> np.sinh(a)
array([ 2.42582598e+08, 5.34323729e+12, 1.17692633e+17,
        2.59235276e+21])
```

Повний список можна переглянути https://docs.scipy.org/doc/numpy/reference/routines.math.html

Список корисних математичних функцій пакета NumPy

У всіх визначеннях нижче масив або скаляр.

numpy.abs(a) - абсолютне значення a;

numpy.around(a, decimals = 0, out = None) - Округлює a до заданої кількості десяткових розрядів, за замовчуванням до цілого. Дотримується

наступного правила заокруглення. Якщо значення a знаходиться точно посередині між двома цілими, округлення проводиться до найближчого парного цілого. Так, якщо a дорівнює 1.5 або 2.5 буде повернуто 2, якщо a дорівнює -0.5 або 0.5 буде повернуто 0.0. Аргумент decimals - ціле, десятковий розряд після коми, до якого проводиться округлення, якщо decimals негативне, розряд відраховується ліворуч від коми.

```
Print(np.around)(np.array([0.5, 1.8, 2.5, 3.5])))
[ 0. 2. 2. 4.]

print(np.around(np.array ([1, 5, 15, 45]), decimals =
1))
[ 1 5 15 45]

Print(np.around)(np.array ([1, 5, 15, 45]), decimals =
-1))
[ 0 0 20 40]
```

Аргумент out - масив, в який буде передано результат, структура масиву out, повинна збігатися зі структурою масиву, що повертається. Якщо None (за замовчуванням) буде створено новий масив.

numpy.fix(a, out = None) - Відкидає дробову частину a;

numpy.ceil(a, out = None) - Округлює a до меншого з цілих великих або рівних a;

```
>>> print (np.ceil(np.array([-2.7, -1.2, -0.5, 1.8,
2.4, 3.6])))
[-2. -1. -0. 2. 3. 4.]
```

numpy.floor(a, out = None) - Округлює a до більшого з цілих менших або рівних a;

```
>>> print np.floor(np.array([-2.7, -1.2, -0.5, 1.8,
2.4, 3.6])))
[-3. -2. -1. 1. 2. 3.]
```

numpy.sign(a) - Повертає -1 якщо a < 0, 0 якщо a == 0, 1 якщо a > 0;

numpy.degrees(a) - конвертує a з радіан у градуси;

numpy.radians(a) - конвертує a із градусів у радіани;

numpy.cos(a)numpy.sin(a), numpy.tan(a) - повертає косинус, синус, тангенс a. a у радіанах;

numpy.cosh(a), numpy.sinh(a), numpy.tanh(a) - повертає гіперболічний косинус, синус, тангенс a. a у радіанах;

numpy.arccos(a), numpy.arcsin(a), numpy.arctan(a) - повертає арккосинус, арксинус, арктангенс а в радіанах. Для арккосинусу в діапазоні [0, nump.pi], для арксинусу [-numpy.pi/2, numpy.pi/2], для арктангенса [-numpy.pi/2, numpy.pi/2];

numpy.arccosh(a), numpy.arcsinh(a), numpy.arctanh(a) - повертає гіперболічний косинус, синус, тангенс а. а у радіанах;

numpy.exp(a) - повертає основу натурального логарифму (число e) у ступені а;

numpy.log(a), numpy.log10(a), numpy.log2(a) - повертає натуральний логарифм а, логарифм а на підставі 10, логарифм а на підставі 2;

numpy.log1p(a) - Повертає натуральний логарифм а + 1;

numpy.sqrt(a) - Повертає позитивний квадратний корінь а;

numpy.square(a) - Повертає квадрат а.

У вирази можна включати кілька масивів. Якщо атрибути ndarray.shape цих масивів збігаються, результат очевидний - дії над масивами будуть проводитися поелементно:

```
>>> a1 = np.array([ [1.0, 2.0], [3.0, 4.0] ])
>>> a2 = np.array([ [5.0, 6.0], [7.0, 8.0] ])
>>> a3 = np.array([ [9.0, 10.0], [11.0, 12.0] ])

>>> print (a1 + a2 + a3)
[[ 15. 18.]
 [21. 24.]]

>>> print (a1 * a2)
[[ 5. 12.]
 [21. 32.]]

>>> print a3 / a1
[[ 9. 5. ]
 [ 3.66666667 3. ]]

>>> print (a3 - a1) * a2
[[ 40. 48.]
 [56. 64.]]
```

Якщо атрибути ndarray.shape не збігаються – дії над масивами виконуються відповідно до концепції транслювання (broadcasting).

Операція транслювання – розширення одного або обох масивів операндів до масивів з рівною розмірністю.

Для початку кілька прикладів:

```
>>> a2 = np.array([1, 2])
>>> print a2
[1 2]
>>> print a2.shape
(2,)

>>> a22 = np.array([ [1, 2], [3, 4] ])
>>> print a22
[[1 2]
 [3 4]]
>>> print a22.shape
(2, 2)

>>> a32 = np.array([ [1, 2], [3, 4], [5, 6] ])
>>> print a32
[[1 2]
 [3 4]
 [5 6]]
>>> print a32.shape
(3, 2)

>>> a222 = np.array([ [ [1, 2], [3, 4] ], [ [5, 6], [7,
8] ] ])
>>> print a222
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
>>> print a222.shape
(2, 2, 2)

>>> print a22 + a2
[[2 4]
 [4 6]]

>>> print a32 + a2
[[2 4]
 [4 6]
 [6 8]]

>>> print a222 + a2
[[[ 2 4]
  [4 6]]
```

```
 [[ 6 8]
  [8, 10]]]

>>> print a32 + a22
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shape mismatch: objects cannot be broadcast
to a single shape

>>> print a222 + a22
[[[ 2 4]
  [6, 8]]

 [[ 6 8]
  [10 12]]]

>>> print a222 + a32
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shape mismatch: objects cannot be broadcast
to a single shape
```

І це правило. Якщо довжини осей масивів, починаючи з замикаючої, попарно рівні або в кожній з порівнюваних пар довжин хоча б одна дорівнює одиниці, то до таких масивів може бути застосована операція транслювання.

Довжина замикаючої осі - довжина вкладеного масиву найбільш глибоко.

```
>>> a = np.ones((7, 3, 4, 9, 8))
>>> b = np.ones((4, 9, 8))
>>> c = np.ones((4, 3, 8))

>>> print (a + b).shape
(7, 3, 4, 9, 8)

>>> print (a + c).shape
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shape mismatch: objects cannot be broadcast
to a single shape

>>> d = np.ones((9, 7, 1, 6, 1))
>>> e = np.ones((7, 2, 1, 4))
>>> print (d + e).shape
(9, 7, 2, 6, 4)
```

Якщо в масивах присутні осі одиничної довжини, можуть розширюватися обидва масиви.

```
>>> f = np.array([1, 2])
>>> print f
[1 2]
>>> print f.shape
(2,)

>>> g = np.array ([[3], [4], [5]])
>>> print g
[[3]
 [4]
 [5]]
>>> print g.shape
(3, 1)

>>> h = f + g
>>> print h
[[4 5]
 [5 6]
 [6 7]]
>>> print h.shape
(3, 2)
```

Трансляція масивів відбувається при виклику деяких функцій.

Наприклад, функція numpy.power(a1, a2), де a1, a2 масив або скаляр, повертає a1 до ступеня a2:

```
>>> print np.power(np.array([-2.0, -1.0, 0.0, 2.0, 3.0,
4.0]), 4)
[16. 1. 0. 16. 81. 256.]

>>> print np.power(np.array([-2.0, -1.0, 0.0, 2.0, 3.0,
4.0]), -4)
[0.0625 1. Inf 0.0625 0.01234568 0.00390625]
```

якщо у переданих масивів не збігається структура, проводить трансляцію.

```
>>> print np.power(np.array([3, 7]), np.array([ [1],
[2], [3] ]))
[[ 3 7]
 [949]
 [27343]]
```

Багато унарних операцій, такі як, наприклад, обчислення суми всіх елементів масиву, представлені також і у вигляді методів класу ndarray.

```
>>>
>>> a =np.array([[1, 2, 3], [4, 5, 6]])
```

```
>>> np.sum(a)
21
>>> a.sum()
21
>>> a.min()
1
>>> a.max()
6
```

За замовчуванням, ці операції застосовуються до масиву, ніби він був списком чисел, незалежно від його форми. Однак, вказавши параметр axis, можна застосувати операцію зазначеної осі масиву:

```
>>>
>>> a.min(axis=0)
                    # Найменша кількість у кожному
стовпці
array([1, 2, 3])
>>> a.min(axis=1)
                    # Найменша кількість у кожному рядку
array([1, 4])
```

Індекси, зрізи, ітерації

Одномірні масиви здійснюють операції індексування, зрізів та ітерацій дуже схожим чином із звичайними списками та іншими послідовностями Python (хіба що видаляти за допомогою зрізів не можна).

```
>>>
>>> a =np.arange(10) ** 3
>>> a
array([0, 1, 8, 27, 64, 125, 216, 343, 512, 729])
>>> a[1]
1
>>> a[3:7]
array([ 27, 64, 125,216])
>>> a[3:7] = 8
>>> a
array([ 0, 1, 8, 8, 8, 8, 8, 343, 512, 729])
>>> a[::-1]
array([729, 512, 343, 8, 8, 8, 8, 8, 1, 0])
>>> del a[4:6]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: cannot delete array elements
>>> for iina:
...     print(i ** (1/3))
...
0.0
```

```
1.0
2.0
2.0
2.0
2.0
2.0
7.0
8.0
9.0
```

Багатомірні масиви на кожну вісь мають один індекс. Індекси передаються у вигляді послідовності чисел, розділених комами (тобто кортежами):

```
>>>
>>> b =np.array([[ 0, 1, 2, 3],
...              [10, 11, 12, 13],
...              [20, 21, 22, 23],
...              [30, 31, 32, 33],
...              [40, 41, 42, 43]])
...
>>> b[2,3]  # Другий рядок, третій стовпець
23
>>> b[(2,3)]
23
>>> b[2][3]  # Можна і так
23
>>> b[:,2]  # Третій стовпець
array([2, 12, 22, 32, 42])
>>> b[:2]  # Перші два рядки
array([[ 0, 1, 2, 3],
       [10, 11, 12, 13]])
>>> b[1:3, : : ]  # Другий і третій рядки
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Коли індексів менше, ніж осей, відсутні індекси передбачаються доповненими за допомогою зрізів:

```
>>>
>>> b[-1]  # Останній рядок. Еквівалентно b[-1,:]

array([40, 41, 42, 43])
```

b[i] можна читати як b[i, <стільки символів ':', скільки потрібно>].
У NumPy це може бути записано з допомогою точок, як b[i, ...].

Наприклад, якщо x має ранг 5 (тобто має 5 осей), тоді

x[1, 2, ...] еквівалентно x[1, 2, :, :, :],
x[... , 3] те саме, що x[:, :, :, :, 3] і
x[4, ... , 5, :] це x[4, :, :, 5, :].

```
>>>
>>> a =np.array(([[0, 1, 2], [10, 12, 13]], [[100, 101, 102],
[110, 112, 113]]))
>>> a.shape
(2, 2, 3)
>>> a[1, ...]  # те саме, що a[1, : , :] або a[1]

array([[100, 101, 102],
       [110, 112, 113]])

>>> c[... ,2]  # те, що a[: , : ,2]
array([[ 2, 13],
       [102, 113]])
```

Ітерування багатовимірних масивів починається з першої осі:

```
>>>
>>> for rowina:
...     print(row)
...
[[ 0 1 2]
 [10 12 13]]
[[100 101 102]
 [110112113]]
```

Однак, якщо потрібно перебрати поелементно весь масив, ніби він був одномірним, для цього можна використовувати атрибут flat:

```
>>>
>>> for elina.flat:
...     print(el)
...
0
1
2
10
12
13
100
101
102
110
112
113
```

Маніпуляції із формою

Як мовилося раніше, масив має форму (shape), яка визначається числом елементів уздовж кожної осі:

```
>>>
>>> a
array([[[ 0, 1, 2],
        [10, 12, 13]],

       [[100, 101, 102],
        [110, 112, 113]]])
>>> a.shape
(2, 2, 3)
```

Форма масиву може бути змінена за допомогою різних команд:

```
>>>
>>> a.ravel()  # Робить масив плоским
array([ 0, 1, 2, 10, 12, 13, 100, 101, 102, 110, 112, 113])

>>> a.shape = (6, 2)  # Зміна форми
>>> a
array([[ 0, 1],
       [2, 10],
       [12, 13],
       [100, 101],
       [102, 110],
       [112, 113]])

>>> a.transpose()  # Транспонування
array([[ 0, 2, 12, 100, 102, 112],
       [1, 10, 13, 101, 110, 113]])
>>> a.reshape((3, 4))  # Зміна форми
array([[ 0, 1, 2, 10],
       [12, 13, 100, 101],
       [102, 110, 112, 113]])
```

Порядок елементів у масиві в результаті функції ravel() відповідає звичайному "C-стилю", тобто чим правіше індекс, тим він "швидше змінюється": за елементом a[0,0] слідує a[0,1].

Якщо одна форма масиву була змінена на іншу, масив також переформовується в "C-стилі".

Функції ravel() і reshape() також можуть працювати (при використанні додаткового аргументу) у FORTRAN-стилі, в якому швидше змінюється лівіший індекс.

```
>>>
>>> a
array([[ 0, 1],
```

```
       [2, 10],
       [12, 13],
       [100, 101],
       [102, 110],
       [112, 113]])
>>> a.reshape((3, 4),order='F')
array([[ 0, 100, 1, 101],
       [2, 102, 10, 110],
       [12, 112, 13, 113]])
```

Метод reshape() повертає її аргумент із зміненою формою, тоді як метод resize() змінює сам масив:

```
>>>
>>> a.resize((2, 6))
>>> a
array([[ 0, 1, 2, 10, 12, 13],
       [100, 101, 102, 110, 112, 113]])
```

Якщо при операції такої перебудови один із аргументів задається як -1, то він автоматично розраховується відповідно до інших заданих:

```
>>>
>>> a.reshape((3, -1))
array([[ 0, 1, 2, 10],
       [12, 13, 100, 101],
       [102, 110, 112, 113]])
```

Об'єднання масивів

Декілька масивів можуть бути об'єднані вздовж різних осей за допомогою функцій hstack і vstack.

hstack()об'єднує масиви по перших осях,

vstack()- За останніми:

```
>>>
>>> a =np.array([[1, 2], [3, 4]])
>>> b =np.array([[5, 6], [7, 8]])
>>> np.vstack((a, b))
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
>>> np.hstack((a, b))
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

Функція column_stack() поєднує одновимірні масиви як стовпці двовимірного масиву:

```
>>>
>>> np.column_stack((a, b))
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

Аналогічно для рядків є функція

row_stack().

```
>>>
>>> np.row_stack((a, b))
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

Функція concatenate() з'єднує масиви вздовж зазначеної осі.

np.concatenate((a1, a2, ..., aN), axis=0)

Параметри: a1, a2, ..., aN - послідовність подібних до масиву об'єктів. Будь-які об'єкти, які можуть бути перетворені в масиви NumPy. Дані об'єкти повинні мати однакову форму (кількість осей), але не розмір вказаної осі

axis – ціле число (необов'язковий). Визначає вісь уздовж якої з'єднуються масиви. За замовчуванням axis=0, що відповідає першій осі.

Повертає: ndarray - масив NumPy - масив, що складається з зазначених масивів, з'єднаних уздовж зазначеної осі.

Розбиття масиву

Використовуючи hsplit() ви можете розбити масив уздовж горизонтальної осі, вказавши або кількість масивів, що повертаються, однакової форми, або номери стовпців, після яких масив розрізається "ножицями":

```
>>>
>>> a =np.arange(12).reshape((2, 6))
>>> a
array([[ 0, 1, 2, 3, 4, 5],
       [6, 7, 8, 9, 10, 11]])
>>> np.hsplit(a, 3)  # Розбити на 3 частини
[array([[0, 1], [6, 7]]),
array([[2, 3], [8, 9]]),
array([[ 4, 5], [10, 11]])]
>>> np.hsplit(a, (3, 4))  # Розрізати a після третього
                                  та четвертого стовпця
[array([[0, 1, 2], [6, 7, 8]]),
array([[3], [9]]),
array([[ 4, 5], [10, 11]])]
```

Функція

vsplit()розбиває масив уздовж вертикальної осі, а
array_split()дозволяє вказати осі, вздовж яких станеться розбиття.

Копії та уявлення

При роботі з масивами їх дані іноді необхідно копіювати в інший масив, а іноді ні. Це часто є джерелом плутанини. Можливо 3 випадки:

Взагалі жодних копій

Просте присвоєння не створює копії масиву, ні копії його даних:

```
>>>
>>> a =np.arange(12)
>>> b = a  # Нового об'єкта створено не було
>>> b is a  # a і b це два імені для одного і того ж
об'єкта ndarray
True
>>> b.shape = (3,4)  # змінить форму a
>>> a.shape
(3, 4)
```

Python передає об'єкти, що змінюються, як посилання, тому виклики функцій також не створюють копій.

Подання або поверхнева копія

Різні об'єкти масивів можуть використовувати ті самі дані. Метод view() створює новий об'єкт масиву, що є уявленням тих самих даних.

```
>>>
>>> c = a.view()
>>> cisa
False
>>> c.baseisa  # c це представлення даних, що належать
a
True
>>> c.flags.owndata
False
>>>
>>> c.shape = (2,6)  # форма a не зміниться
>>> a.shape
(3, 4)
>>> c[0,4] = 1234  # дані a зміняться
>>> a
array([[ 0, 1, 2, 3],
       [1234, 5, 6, 7],
       [8, 9, 10, 11]])
```

Зріз масиву це уявлення:

```
>>>
>>> s = a[:,1:3]
>>> s[:] = 10
>>> a
array([[ 0, 10, 10, 3],
       [1234, 10, 10, 7],
       [8, 10, 10, 11]])
```

Глибока копія

Метод copy() створить справжню копію масиву та його даних:

```
>>>
>>> d = a.copy()  # створюється новий об'єкт масиву з
                                          новими даними
>>> d is a
False
>>> d.base is a  # d не має нічого спільного з
False
>>> d[0, 0] = 9999
>>> a
array([[ 0, 10, 10, 3],
       [1234, 10, 10, 7],
       [8, 10, 10, 11]])
```

NumPy випадкові числа

Розглянемо, як створювати масиви з випадкових елементів і як працювати з випадковими елементами NumPy.

Списки до масивів

Створювати списки, використовуючи вбудований модуль random, а потім перетворювати їх на numpy.array:

```
>>>
>>> import numpyasnp
>>> import random

>>> np.array([random.random()foriinrange(10)])

array([0.99538667, 0.16860511, 0.78952804,
                    0.09676316, 0.86110208,
       0.89674666, 0.56401347, 0.63431468,
                          0.51110935, 0.64944844])
```

Але є спосіб кращий.

Модуль numpy.random

Для створення масивів із випадковими елементами служить модуль numpy.random.

```
>>>
>>> import numpy as np # Імпортувати numpy
                                  та писати np.random
>>> np.random
<module 'numpy.random' from <module 'numpy.random' from
'C:\Program Files\Python36\lib\site-
packages\numpy\random\__init__.py'>
>>> import numpy.random as rand # Можна і присвоїти
                               окреме ім'я. Питання смаку
>>> rand
<module 'numpy.random' from 'C:\Program Files\Python36\lib\site-
packages\numpy\random\__init__.py'>
```

Створення масивів із випадковими елементами

Найпростіший спосіб задати масив з випадковими елементами - використовувати функцію sample (або random, або random_sample, або ranf - це все та сама функція).

```
>>>
>>> np.random.sample()
0.6336371838734877
>>> np.random.sample(3)
array([0.53478558, 0.1441317, 0.15711313])
>>> np.random.sample((2, 3))
array([[ 0.12915769, 0.09448946, 0.58778985],
       [0.45488207, 0.19335243, 0.22129977]])
```

Без аргументів повертає просто число у проміжку [0, 1),

- з одним цілим числом – одномірний масив,
- з кортежем - масив із розмірами, зазначеними у кортежі (всі числа - із проміжку [0, 1)).

За допомогою функції randint або random_integers можна створити масив цілих чисел.

Аргументи: low, high, size: від якого, до якого числа (randint не включає це число, а random_integers включає), і size - розміри масиву.

```
>>>
>>> np.random.randint(0,3, 10)
array([0, 2, 0, 1, 1, 0, 2, 2, 2, 0])
>>> np.random.random_integers(0,3, 10)
```

```
array([2, 2, 3, 3, 1, 1, 0, 2, 3, 2])
>>> np.random.randint(0,3, (2, 10))
array([[0, 1, 2, 0, 0, 0, 1, 1, 1, 2],
       [0, 0, 2, 2, 2, 0, 1, 2, 2, 1]])
```

Також можна генерувати числа згідно з різними розподілами (Гаусса, Парето та інші). Найчастіше потрібен рівномірний розподіл, який можна отримати за допомогою функції uniform.

```
>>>
>>> np.random.uniform(2, 8, (2, 10))
array([[ 3.1517914 , 3.10313483, 2.84007134,
                        3.21556436, 4.64531786,
         2.99232714, 7.03064897, 4.38691765,
                        5.27488548, 2.63472454],
       [6.39470358, 5.63084131, 4.69996748,
                        7.07260546, 7.44340813,
         4.10722203, 7.52956646, 4.8596943,
                       3.97923973, 5.64505363]])
```

Вибір та перемішування

Перемішати масив NumPy можна за допомогою функції shuffle:

```
>>>
>>> a =np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.random.shuffle(a)
>>> a
array([2, 8, 7, 3, 5, 0, 4, 9, 1, 6])
```

Також можна перемішати масив за допомогою функції permutation (вона, на відміну shuffle, повертає перемішаний масив). Також вона, викликана одним аргументом (цілим числом), повертає перемішану послідовність від 0 до N.

```
>>>
>>> np.random.permutation(10)
array([1, 2, 3, 8, 7, 9, 4, 6, 5, 0])
```

Зробити довільну вибірку з масиву можна за допомогою функції choice.

```
numpy.random.choice(a, size=None, replace=True, p=None)
```

- a: одновимірний масив або число. Якщо масив, проводитиметься вибірка з нього. Якщо число, то вибірка буде з np.arange(a).
- size: розмірності масиву. Якщо None повертається одне значення.
- replace : якщо True, одне значення може вибиратися більше разу.
- p: ймовірності. Це означає, що елементи можна вибирати з нерівними можливостями. Якщо не задано, використовується рівномірний розподіл.

```
>>>
>>> a =np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.random.choice(a, 10, p=[0.5, 0.25, 0.25, 0, 0,
0, 0, 0, 0, 0])
array([0, 0, 0, 0, 1, 2, 0, 0, 1, 1])
```

Ініціалізація генератора випадкових чисел

seed(число)- Ініціалізація генератора.

```
>>>
>>> np.random.seed(1000)
>>> np.random.random(10)
array([0.65358959, 0.11500694, 0.95028286,
                   0.4821914, 0.87247454,
       0.21233268, 0.04070962, 0.39719446,
                             0.2331322, 0.84174072])
>>> np.random.seed(1000)
>>> np.random.random(10)
array([0.65358959, 0.11500694, 0.95028286,
                              0.4821914, 0.87247454,
       0.21233268, 0.04070962, 0.39719446,
                             0.2331322, 0.84174072])
```

get_statei set_state - повертають та встановлюють стан генератора.

```
>>>
>>> np.random.seed(1000)
>>> state =np.random.get_state()
>>> np.random.random(10)
array([0.65358959, 0.11500694, 0.95028286,
                                0.4821914, 0.87247454,
       0.21233268, 0.04070962, 0.39719446,
                              0.2331322, 0.84174072])
>>> np.random.set_state(state)
>>> np.random.random(10)
array([0.65358959, 0.11500694, 0.95028286,
                             0.4821914, 0.87247454,
       0.21233268, 0.04070962, 0.39719446,
                           0.2331322, 0.84174072])
```

Деякі корисні функції

numpy.min(a, axis = None, out = None),
numpy.max(a, axis = None, out = None)

повертає мінімальне, максимальне значення елементів масиву відповідно:

```
>>> import numpy as np
>>> print np.min(np.array([ [1.0, -0.5, 3.0], [4.0,
3.0, -0.5] ]))
-0.5
```

axis - опціональний аргумент, індекс осі (вимірювання) масиву яким проводиться пошук мінімального, максимального значення. Під індексом осі розуміється індекс у кортежі ndarray.shape.

```
>>> a = np.array([ [ [1.0, 2.0, 3.0], [4.0, 5.0,
6.0] ], [ [7.0, 8.0, 9.0], [11, 12, 13] ] ]))
>>> print a
[[[ 1. 2. 3.]
  [4. 5. 6.]]

 [[ 7. 8. 9.]
  [11. 12. 13.]]]
>>> print a.shape
(2, 2, 3)

>>> print np.min(a, axis = 0)
[[ 1. 2. 3.]
 [4. 5. 6.]]

>>> print np.min(a, axis = 1)
[[ 1. 2. 3.]
 [7. 8. 9.]]

>>> print np.min(a, axis = 2)
[[ 1. 4.]
 [7. 11.]]
```

out- опціональний аргумент, масив, до якого буде поміщений результат. Структура масиву out має відповідати структурі масиву результату. Якщо out = None (за замовчуванням) буде створено новий масив.

numpy.argmin(a, axis = None),
numpy.argmax(a, axis = None) -
повертає індекс мінімального, максимального значення елементів масиву відповідно:

```
>>> print np.argmin(np.array([ [1.0, -0.5, 3.0], [4.0,
3.0, -0.5] ]))
1

>>> print np.argmin(a, axis = 0)
```

```
[[0 0 0]
 [0 0 0]]

>>> print np.argmin(a, axis = 1)
[[0 0 0]
 [0 0 0]]

>>> print np.argmin(a, axis = 2)
[[0 0]
 [0 0]]
```

numpy.sum(a, axis = None, dtype = None, out = None),
numpy.prod(a, axis = None, dtype = None, out = None)

повертає суму, добуток елементів масиву відповідно:

```
>>> print np.sum(np.array([ [1.0, -0.5, 3.0], [4.0,
3.0, -0.5] ]))
10.0
>>> print np.sum(a, axis = 0)
[[ 8. 10. 12.]
 [15. 17. 19.]]
>>> print np.sum(a, axis = 1)
[[ 5. 7. 9.]
 [18. 20. 22.]]
>>> print np.sum(a, axis = 2)
[[ 6. 15.]
 [24. 36.]]
```

NumPy linalg деякі операції лінійної алгебри

Розглянемо модуль numpy.linalg, що дозволяє робити багато операцій із лінійної алгебри.

Зведення у ступінь

linalg.matrix_power(M, n)- Зводить матрицю в ступінь n.

Розкладання

linalg.cholesky(a) – розкладання Холецького.
linalg.qr(a [, mode]) - QR розкладання.
linalg.svd(a[, full_matrices, compute_uv]) - сингулярне розкладання.

Деякі характеристики матриць

linalg.eig(a) - власні значення та власні вектори.
linalg.norm(x[, ord, axis]) - норма вектора чи оператора.
linalg.cond(x[, p]) – число обумовленості.
linalg.det(a) – визначник.

linalg.slogdet(a) - знак і логарифм визначника (для уникнення переповнення, якщо сам визначник дуже маленький).

Системи рівнянь

linalg.solve(a, b) – вирішує систему лінійних рівнянь Ax = b.

linalg.tensorsolve(a, b[, axes]) – вирішує тензорну систему лінійних рівнянь Ax = b.

linalg.lstsq(a, b [, rcond]) – метод найменших квадратів.

linalg.inv(a) – зворотна матриця.

Зауваження:

- **linalg.LinAlgError** - Виняток, що викликається цими функціями у разі невдачі (наприклад, при спробі взяти зворотну матрицю від виродженої).
- Детальна документація: https://docs.scipy.org/doc/numpy/reference/routines.linalg.html
- Масиви більшої розмірності у більшості функцій linalg інтерпретуються як набір кількох масивів потрібної розмірності. Таким чином, можна одним викликом функції виконувати операції над кількома об'єктами.

```
>>>
>>> a =np.arange(18).reshape((2,3,3))
>>> a
array([[[ 0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])

>>> np.linalg.det(a)
array([0., 0.])
```

Винятки numpy.linalg.LinAlgError

numpy.linalg.LinAlgError
exception numpy.linalg.LinAlgError

Об'єкт linalg.LinAlgError генерує винятки Python, викликані функціями модуля linalg.

Цей об'єкт утворений класом винятків загального призначення Python і є похідним від нього. Цей клас викликається, тільки якщо подальша робота будь-якої функції модуля linalg неможлива.

Приклади

```
>>> import numpy as np
>>> від numpy import linalg as LA
>>>
>>> a = [[0, 0], [0, 0]]
>>>
>>> try:
... LA.inv(a)
... except LA.LinAlgError:
... print('Матриця "a" є або виродженою або прямокутною')
...
```

Матриця "a" є або виродженою або прямокутною

Приклади

Добуток одновимірних масивів є скалярним добутком векторів:

```
>>> a = np.array([1,2])
>>> b = np.array([3,4])
>>>
>>> np.dot(a,b)
11
```

Добуток двовимірних масивів за правилами лінійної алгебри також можливий:

```
>>> a = np.arange(2,6).reshape(2,2)
>>> a
array([[2,3],
       [4, 5]])
>>>
>>> b = np.arange(6,10).reshape(2,2)
>>> b
array([[6,7],
       [8, 9]])
>>>
>>>np.dot(a,b)
array([[36,41],
       [64, 73]])
```

У цьому розміри матриць (масивів) мали бути зацікавленими або рівні, а самі матриці квадратними, або узгодженими, тобто. якщо розміри матриці A дорівнюють [m,k], то розміри матриці повинні бути рівні [k,n]:

```
>>> a = np.arange(2,8).reshape(2,3)
>>> a
array([[2,3, 4],
       [5, 6, 7]])
>>>
>>> b = np.arange(4,10).reshape(3,2)
>>> b
array([[4,5],
       [6,7],
       [8, 9]])
>>>
>>>np.dot(a,b)
array([[58, 67],
       [112, 130]])
```

Також за правилами множення матриць, ми можемо помножити матрицю на вектор (одномірний масив). При цьому в такому множенні вектор стовпець повинен бути праворуч, а вектор рядок зліва:

```
>>> a = np.array([1,2,3])
>>> a
array([1, 2, 3])
>>>
>>> b = np.arange(4,10).reshape(3,2)
>>> b
array([[4,5],
       [6,7],
       [8, 9]])
>>>
>>>np.dot(a,b)
array([40, 46])
>>>
>>> a = np.arange(1,3).reshape(2,1)
>>> a
array([[1],
       [2]]))
>>>
>>> b = np.arange(4,10).reshape(3,2)
>>> b
array([[4,5],
       [6,7],
       [8, 9]])
>>>
>>>np.dot(b,a)
array([[14],
       [20],
       [26]]))
```

Квадратні матриці можна зводити до ступеня n тобто. множити самі на себе n разів:

```
>>> a = np.arange(1,5).reshape(2,2)
>>> a
array([[1,2],
       [3, 4]])
>>>
>>>np.dot(a,a) # Рівносильно a**2
array([[7, 10],
       [15, 22]])
>>>
>>>np.linalg.matrix_power(a,2)
array([[7, 10],
       [15, 22]])
>>>
>>>np.linalg.matrix_power(a,5)
array([[1069,1558],
       [2337, 3406]])
>>>
>>>np.linalg.matrix_power(a,0)
array([[1,0],
       [0, 1]])
```

Досить часто доводиться обчислювати ранг матриць:

```
>>> a = np.arange(1,10).reshape(3,3)
>>> a
array([[1,2, 3],
       [4,5, 6],
       [7, 8, 9]])
>>>
>>> np.linalg.matrix_rank(a)
2
>>>
>>> b = np.arange(1,24,2).reshape(3,4)
>>> b
array([[1, 3, 5, 7],
       [9, 11, 13, 15],
       [17, 19, 21, 23]])
>>>
>>> np.linalg.matrix_rank(b)
2
```

Ще частіше доводиться обчислювати визначник матриць, хоча результат вас може трохи здивувати:

```
>>> a = np.array([[1,3],[4,3]])
>>> a
array([[1,3],
       [4, 3]])
>>>
>>> np.linalg.det(a)
-8.99999999999999982
>>>
>>> 1*3 - 3*4 # Результат має бути цілим числом
-9
```

У даному випадку, через двійкову арифметику, результат не ціле число і округляти до найближчого цілого доведеться вручну. Це з тим, що алгоритм обчислення визначника використовуєLU-розкладання- це набагато швидше ніж звичайний алгоритм, але за швидкість все ж таки доводиться трохи заплатити ручним округленням (звичайно, якщо таке потрібно):

```
>>> np.linalg.det(a)
-8.99999999999999982
>>> round(np.linalg.det(a))
-9.0
>>>
>>> b = np.arange(1,48,3).reshape(4,4)
>>> np.linalg.det(b)
-1.0223637331664275e-27
>>> round(np.linalg.det(b))
-0.0
```

Транспонування матриць:

```
>>> a
matrix([[1,3],
        [4, 3]])
>>>
>>>aT
matrix([[1,4],
        [3, 3]])
>>>
>>> b
```

```
array([[1, 4, 7, 10],
       [13,16, 19, 22],
       [25,28, 31, 34],
       [37, 40, 43, 46]])
>>>
>>>bT
array([[1, 13, 25, 37],
       [4, 16, 28, 40],
       [7, 19, 31, 43],
       [10, 22, 34, 46]])
```

Обчислення зворотних матриць:

```
>>> a = np.array([[1,3],[4,3]])
>>> a
array([[1,3],
       [4, 3]])
>>>
>>> b = np.linalg.inv(a)
>>> b
array([[-0.33333333,  0.33333333],
       [0.44444444, -0.11111111]])
>>>
>>>np.dot(a,b)
array([[1., 0.],
       [0, 1.]])
```

Розв'язання систем лінійних рівнянь:

```
... # система з двох лінійних рівнянь:
...
... # 1*x1 + 5*x2 = 11
... # 2*x1 + 3*x2 = 8
...
>>> a = np.array([[1,5],[2,3]])
>>> b = np.array([11,8])
>>>
>>> x = np.linalg.solve(a,b)
>>> x
array([ 1., 2.])
>>>
>>>np.dot(a,x)
array ([11., 8.])
```

NumPy Перетворення фур'є

По-простому, перетворення Фур'є - розкладання деякого сигналу на гармонійні (синуси або косинуси) коливання (спектр).

Перетворення Фур'є є інтегральним перетворенням. Якщо йдеться про дискретний сигнал, то інтеграл перетворюється на суму (і стає дискретним перетворенням Фур'є, ДПФ). Щоб порахувати таку суму N елементів, треба здійснити N^2 операцій із комплексними числами. Але досить давно (Cooley, Tukey, 1965 р, а ще сам Гаус в 1805 р.) придумав алгоритм, що обчислює ДПФ N елементів у N * log (N) операцій (більшість з яких над дійсними числами), що суттєво економить обчислювальний час. Такий алгоритм часто

називають "швидке перетворення Фур'є, ШПФ (Fast Fourier Transform, FFT)". Саме так реалізовано ДПФ у сучасних комп'ютерних програмах.

У бібліотеці NumPy міститься все, що потрібно для дискретного перетворення Фур'є. Все це лежить у модулі numpy.fft.

Ось ці функції.

Загальний випадок: сигнал може бути як із дійсних чисел, так і з комплексних.

fft(a, n=None, axis=-1) - Пряме одновимірне ДПФ.

ifft(a, n=None, axis=-1) - Зворотне одномірне ДПФ.

Тут:

a – "сигнал", вхідний масив (масив numpy array або навіть пітонівський список або кортеж, якщо в ньому лише числа).

Масив може бути і багатовимірним, тоді обчислюватиметься багато одномірних ПФ за рядками (за замовчуванням) або стовпцями, залежно від параметра axis.

Наприклад, a - двомірний, a [n] [m]:

при axis=1 чи -1 буде таке (під fourier(a...) розуміється результат дії ПФ на a):

```
[fourier(a[0][j]), fourier(a[1][j]), ...
                                    foirier(a[n][j])]
```

При axis=0 таке:

```
[fourier(a[i][0]), fourier(a[i][1]),...
                                      foirier(a[i][m])]
```

n- Скільки елементів масиву брати. Якщо менше довжини масиву, то обрізати, якщо більше, доповнити нулями, за замовчуванням len(a).

fft2(a, s=None, axes=(-2, -1)) - Пряме двовимірне ПФ.

ifft2(a, s=None, axes=(-2, -1)) - Зворотне двовимірне ПФ.

fftn(a, s=None, axes=None) - Пряме багатовимірне ПФ.

ifftn(a, s=None, axes=None)- Зворотне багатовимірне ПФ.

Так само, як і для одномірних, але s і axes тепер кортежі для кожної розмірності. Про розмірність fftn, ifftn здогадаються за розмірністю вхідних масивів або s і axes.

Коли сигнал дійсний (real) (мабуть, найпоширеніший випадок):

rfft(a, n=None, axis=-1) - Пряме одновимірне ДПФ (для дійсних чисел).

irfft(a, n=None, axis=-1) - Зворотне одномірне ДПФ.

rfft2(a, s=None, axes=(-2, -1)) - Пряме двовимірне ДПФ.

irfft2(a, s=None, axes=(-2, -1)) - Зворотне двомірне ДПФ.

rfftn(a, s=None, axes=None) - Пряме багатовимірне ДПФ.

irfftn(a, s=None, axes=None) - Зворотне багатовимірне ДПФ.

Так само, як і для загального випадку.

Всі ці функції повертають масив відповідної розмірності, в якому записано результат ДПФ.

Різниця така. Якщо довжина вхідного масиву (або будь-якої його розмірності) N, то загальному випадку (з комплексним сигналом) довжина вихідного масиву N.

Там містяться спочатку позитивні частоти від нуля до частоти Котельникова (Найквіста), потім негативні у порядку зростання.

У разі дійсного сигналу негативні частоти повністю симетричні позитивним, і тоді немає потреби їх записувати: довжина вихідного масиву N/2+1, частоти від нуля до частоти Котельникова.

Якщо спектр сигналу дійсний (а сигнал має "ермітову симетрію": його половини симетричні щодо центру за модулем і є комплексно пов'язаними один одному), то можна застосувати такі функції:

`hfft(a, n=None, axis=-1)` - Пряме одновимірне ДПФ.

`ihfft(a, n=None, axis=-1)` - Зворотне одномірне ДПФ.

Довжина вхідного масиву N, а вихідного 2*N+1.

Крім того, є допоміжні функції (буде зрозумілішим з прикладу):

`fftfreq(n, d=1.0)` - Повертає частоти для вихідних масивів функцій fft*.

`rfftfreq(n, d=1.0)` - Повертає частоти для вихідних масивів функцій rfft*.

Тут – n – довжина вхідного масиву, d – період дискретизації (зворотна частота дискретизації).

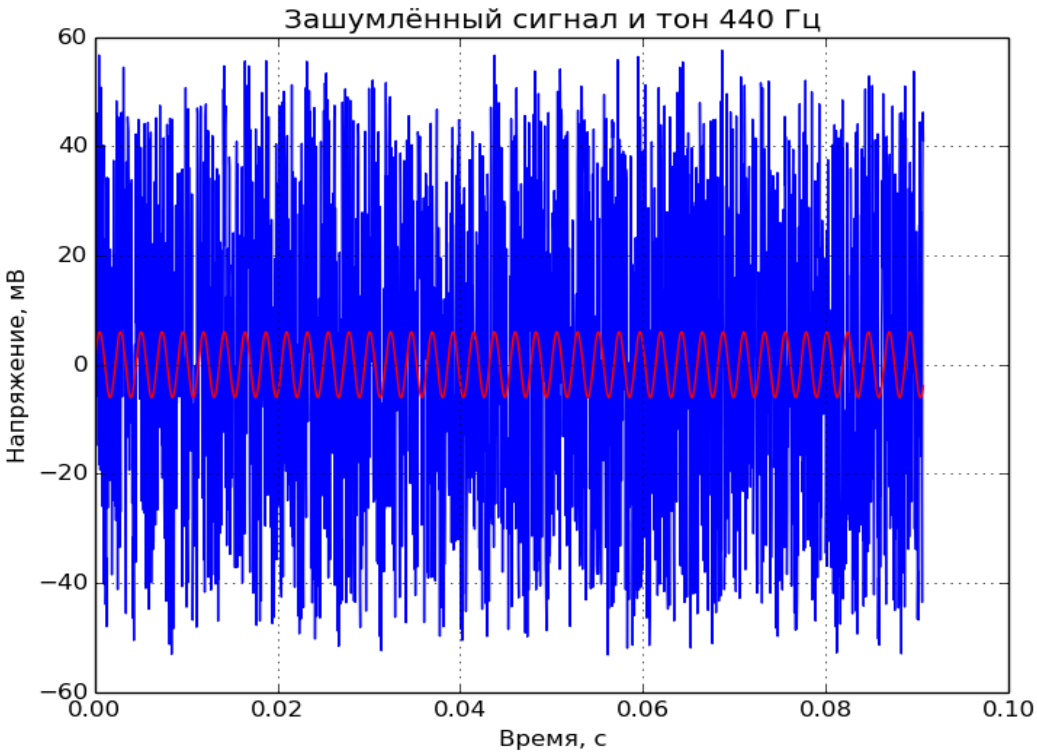
`fftshift(x, axes=None)` - Перетворює масив (з результатом ДПФ, від функцій fft *) так, щоб нульова частота була в центрі.

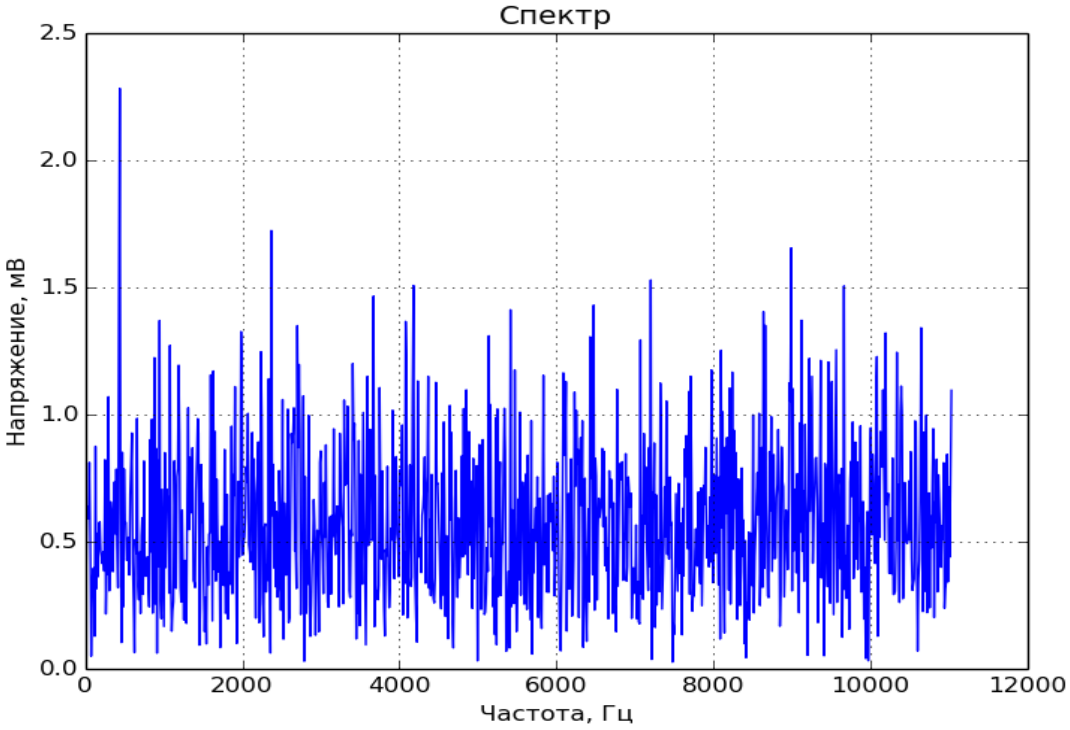
`ifftshift(x, axes=None)` - Здійснює зворотну операцію.

Наведемо такий приклад. Припустимо, записали ми мікрофоном якийсь шум, і треба визначити, чи є там якийсь тон.

```
fromnumpyimportarray, arange, abs as np_abs
fromnumpy.fftimportrfft, rfftfreq
fromnumpy.randomimportuniform
frommathimportsin, pi
importmatplotlib.pyplot as plt
# а можна імпортувати numpy та писати: numpy.fft.rfft
FD = 22050 частота дискретизації, відліків за секунду
а це означає, що в дискретному сигналі представлені
частоти від нуля до 11025 Гц (це і є теорема
Котельникова)
N = 2000  # Довжина вхідного масиву, 0.091 секунд при
          такій частоті дискретизації
# згенеруємо сигнал із частотою 440 Гц довжиною N
```

```
pure_sig =
   array([6.*sin(2.*pi*440.0*t/FD)fort in range(N)])
# згенеруємо шум, теж довжиною N (це важливо!)
noise = uniform(-50.,50., N)
підсумовуємо їх і додамо постійну складову 2 мВ
(припустимо, не дуже хороший мікрофон попався. Або
звукова карта або АЦП)
sig = pure_sig + noise + 2.0
# у numpy так перевантажена функція додавання
# Обчислюємо перетворення Фур'є. # Сигнал дійсний, тому
треба # використовувати rfft, це швидше, ніж fft
spectrum = rfft(sig)

# намалюємо все це, використовуючи matplotlib
# Спочатку сигнал зашумлений і тон окремо
plt.plot(arange(N)/float(FD), sig) # по осі часу #
секунди!
plt.plot(arange(N)/float(FD), pure_sig, 'r') # Чистий #
сигнал буде намальований червоним
plt.xlabel(u'Час, с')
plt.ylabel(u'Напруга, мВ')
plt.title(u'Зашумлений сигнал і тон 440 Гц')
plt.grid(True)
plt.show()
# коли закриється цей графік, відкриється наступний
# Потім спектр
plt.plot(rfftfreq(N, 1./FD), np_abs(spectrum)/N)
# rfftfreq зробить всю роботу з перетворення номерів
елементів масиву в герці
# нас цікавить тільки спектр амплітуд, тому
використовуємо abs з numpy # (діє на масиви
поелементно)

ділимо на число елементів, щоб амплітуди були в
мілівольтах, а не в сумах Фур'є. # Перевірити просто -
постійні складові повинні # збігатися в згенерованому
сигналі та в спектрі

plt.xlabel(u'Частота, Гц')
plt.ylabel(u'Напруга, мВ')
plt.title(u'Спектр')
plt.grid(True)
plt.show()
```


Зашумлённый сигнал и тон 440 Гц
Напряжение, мВ
Время, с
60
40
20
0
−20
−40
−60
0.00
0.02
0.04
0.06
0.08
0.10


Спектр
Напряжение, мВ
Частота, Гц
2.5
2.0
1.5
1.0
0.5
0.0
0
2000
4000
6000
8000
10000
12000

II. Matplotlib

Вступ

matplotlib - Набір додаткових модулів (бібліотек) мови Python. Надає засоби для побудови найрізноманітніших 2D графіків та діаграм даних. Переваги цієї бібліотеки – простота використання. Для побудови дуже складних і барвисто оформлених діаграм достатньо кількох рядків коду. При цьому якість зображень більш ніж достатньо для їх опублікування. Робота цього модуля зазвичай має на увазі також використання модуля NumPy.

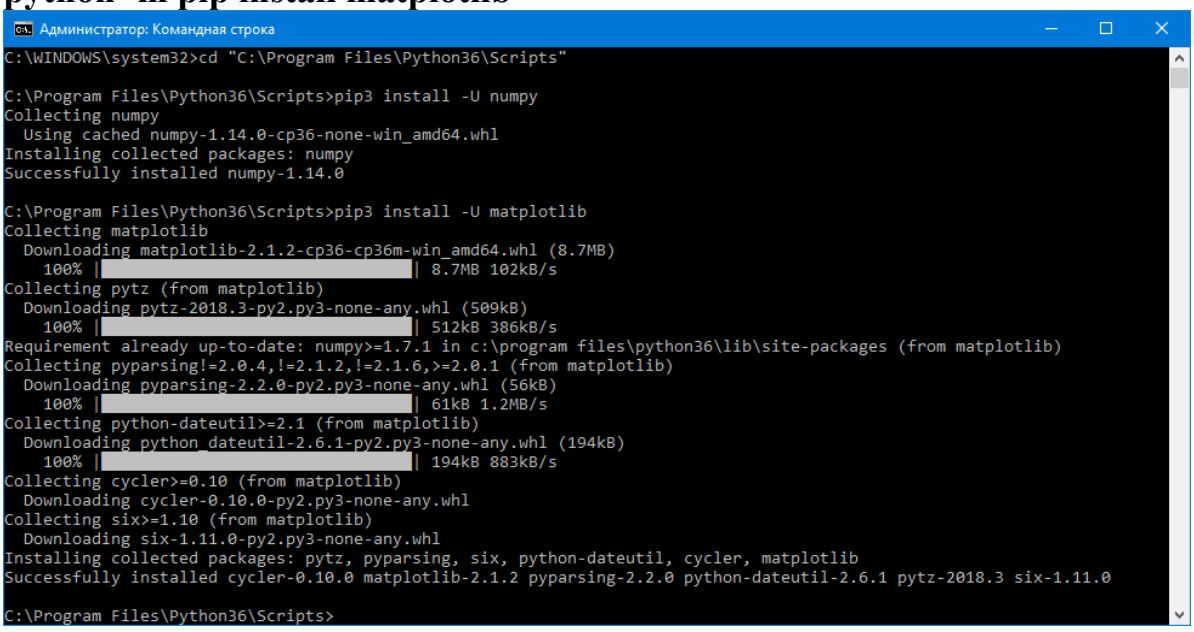
Мережевий ресурсhttps://github.com/whitehorn/Scientific_graphics_in_python містить загальнодоступний підручник. Багато інформації з цієї теми розміщено наhttp://jenyay.net/Programming/Python3dihttps://pyprog.pro/.

Встановлення

У Windows установка і matplotlib і NumPy не повинні викликати жодних проблем. Використовуємо pip3 - систему керування пакетами, яка використовується для встановлення та керування програмними пакетами, написаними на Python.

Якщо Python, наприклад, встановлений у C:\Program Files\Python36\, то в командному рядку адміністратора (це важливо) необхідно виконати такі команди:

cd “:\Program Files\Python36\Scripts”
pip3 install numpy
pip3 install matplotlib
чи так
python -m pip install numpy
python -m pip install matplotlib



Протокол роботи:

```
C:\WINDOWS\system32>cd "C:\Program Files\Python36\Scripts"

C:\Program Files\Python36\Scripts>pip3 install -U numpy
Collecting numpy
  Using cached numpy-1.14.0-cp36-none-win_amd64.whl
Installing collected packages: numpy
Successfully installed numpy-1.14.0

C:\Program Files\Python36\Scripts>pip3 install -U matplotlib
Collecting matplotlib
  Downloading matplotlib-2.1.2-cp36-cp36m-win_amd64.whl (8.7MB)
    100% |████████████████████████████████| 8.7MB 102kB/s
Collecting pytz (від matplotlib)
  Downloading pytz-2018.3-py2.py3-none-any.whl (509kB)
    100% |████████████████████████████████| 512kB 386kB/s
Requirement already up-to-date: numpy>=1.7.1 in c:\program
files\python36\lib\site-packages (from matplotlib)
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib)
  Downloading pyparsing-2.2.0-py2.py3-none-any.whl (56kB)
    100% |████████████████████████████████| 61kB 1.2MB/s
Collecting python-dateutil>=2.1 (from matplotlib)
  Downloading python_dateutil-2.6.1-py2.py3-none-any.whl (194kB)
    100% |████████████████████████████████| 194kB 883kB/s
Collecting cycler>=0.10 (від matplotlib)
  Downloading cycler-0.10.0-py2.py3-none-any.whl
Collecting six>=1.10 (from matplotlib)
  Downloading six-1.11.0-py2.py3-none-any.whl
Installing collected packages: pytz, pyparsing, six, python-dateutil,
cycler, matplotlib
Successfully installed cycler-0.10.0 matplotlib-2.1.2 pyparsing-2.2.0
python-dateutil-2.6.1 pytz-2018.3 six-1.11.0

C:\Program Files\Python36\Scripts>
```

Примітка-У старших версіях MatPlotLib можливі проблеми при інсталяції. У цьому випадку слід встановлювати Python та необхідні модулі для одного користувача (вказується в діалозі встановлення Python).

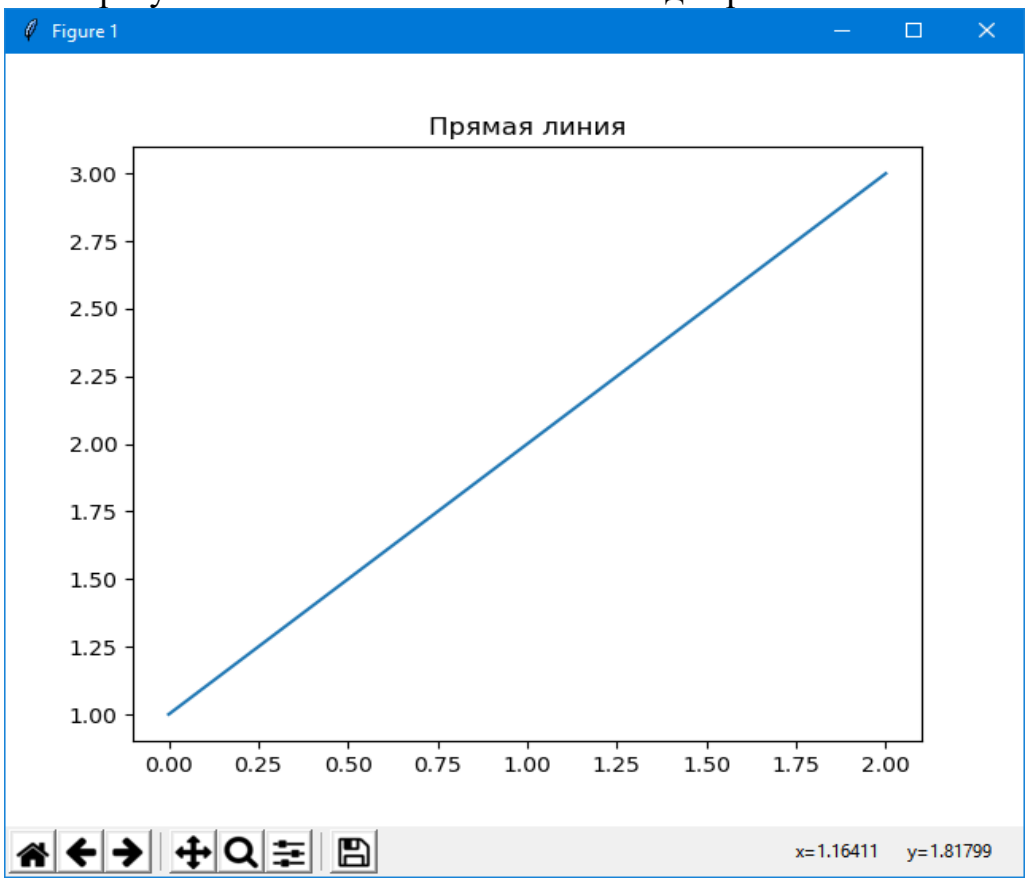
Перевірка

Після встановлення перевіряємо працездатність. Запускаємо консоль Python, вводимо:

```
>>> import matplotlib as mpl
>>> print ('Current version on matplotlib library is',
mpl.__version__)
Current version on matplotlib library is 2.1.2

# де-факто стандарт виклику pyplot у python
import matplotlib.pyplot as plt
plt.plot([1,2,3])
[<matplotlib.lines.Line2D object at 0x0301E810>]
plt.title('Пряма лінія')
<matplotlib.text.Text object at 0x03062430>
```

```
plt.show()
```

В результаті має з'явитися таке вікно діаграми:



Що було зроблено?

З пакету matplotlib імпортований модуль pyplot під назвою plt.

Модуль pyplot містить функції (схожі на команди) створення діаграм та зміни властивостей їх елементів.

Функція plot() будує прямокутні двовимірні діаграми (графіки) координатах X - Y.

Якщо функції plot передано один аргумент (у нашому випадку - список [1,2,3]), вона розглядає його як сукупність значень відкладаються по осі Y, тоді осі X йому буде відповідати автоматично згенерований набір чисел 0,1,2 ... N - 1, де N - число елементів в переданому списку.

Функція title() визначає заголовок діаграми, а функція show() виводить інтерактивне вікно діаграми. Загалом, все дуже просто

Налаштування

Matplotlib можна легко налаштувати через конфігураційний файл matplotlibrc. Якщо Python встановлений в папку C:\Program Files\Python36\, то настроювальний файл розташовується в C:\Program Files\Python36\Lib\site-packages\matplotlib\mpl-data\.

Вносити зміни можна прямо тут, але краще скопіювати файл до папки користувача: C:\Documents and Settings\UserName\.matplotlib\, інакше при

перевстановленні пакета конфігураційний файл буде перезаписано. Параметри пакета задаються у файлі як пар властивість : значення, символ # відокремлює коментар.

Властивість font.family визначає тип активного шрифту за умовчанням, може набувати п'ять значень: serif, sans-serif, cursive, fantasy, monospace.

У свою чергу властивості font.serif, font.sans-serif, font.cursive, font.fantasy, font.monospace містять списки імен шрифтів (через кому, в порядку зменшення пріоритету), що відповідають кожному типу.

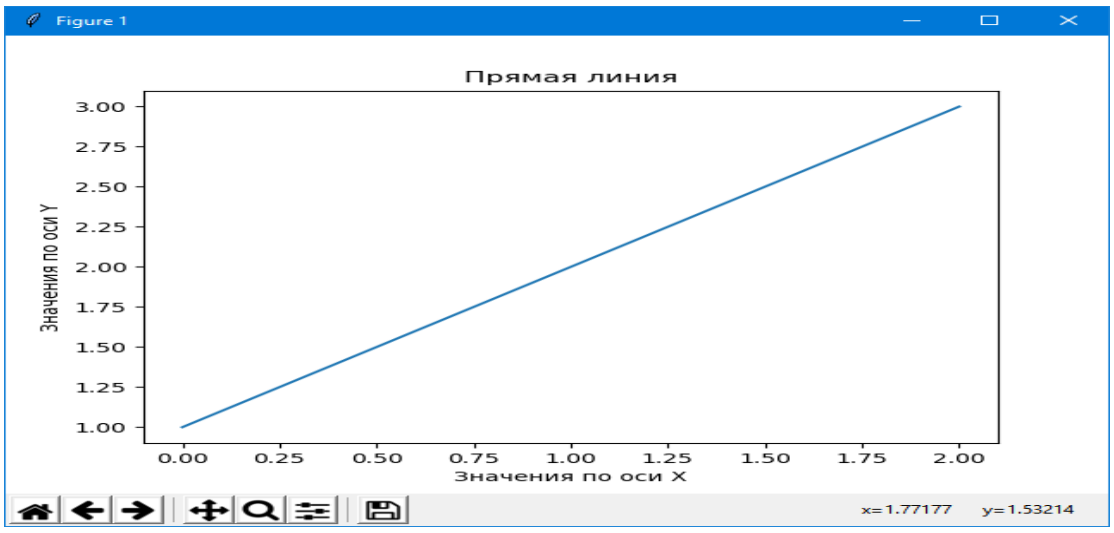
Наприклад, для "русифікації" matplotlib, можливо, доведеться змінити імена шрифтів у списку, на імена доступних в системі шрифтів, що містять кириличні символи. Наприклад так:

```
font.serif : Verdana, Arial
font.sans-serif : Tahoma, Arial
font.cursive: Courier New, Arial
font.fantasy: Comic Sans MS, Arial
font.monospace : Arial
```

Зауважимо, що останні версії пакета встановлюються з коректним настроювальним файлом.

Розглянемо приклад:

```
>>> plt.plot([1,2,3])
[<matplotlib.lines.Line2D object at 0x000001BB6DFCA588>]
>>> plt.title('Пряма лінія')
Text(0.5,1,'Пряма лінія')
>>> plt.xlabel(u'Значення по осі X')
Text(0.5,0,'Значення по осі X')
>>> plt.ylabel(u'Значення по осі Y')
Text(0,0.5,'Значення по осі Y')
>>> plt.show()
```



Функції pyplot.xlabel(), pyplot.ylabel(), як легко здогадатися за назвою, встановлюють підписи до осей X та Y відповідно.

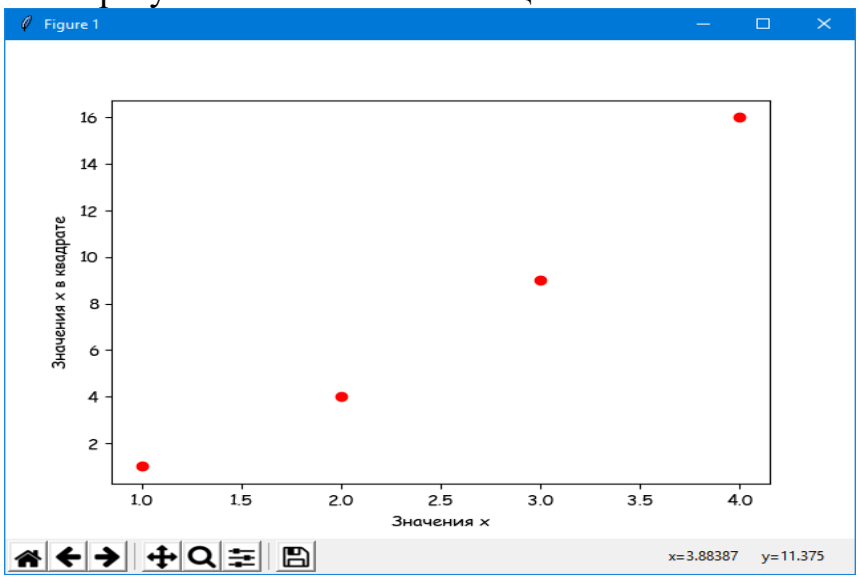
Пакет matplotlib можна налаштовувати динамічно (під час виконання програми). У пакеті визначено структуру matplotlib.rcParams, з якою

працюють зі словником. Ключові слова в даному випадку - імена властивостей файлу matplotlibrc.

Налаштування шрифтів через matplotlib.rcParams можна змінити, наприклад:

```
>>>import matplotlib as mpl
>>>import matplotlib.pyplot as plt
>>> mpl.rcParams['font.family'] = 'fantasy'
>>> mpl.rcParams['font.fantasy'] = 'Comic Sans MS, Arial'
>>> plt.plot([1,2,3,4], [1,4,9,16], 'ro')
[<matplotlib.lines.Line2D object at 0x000001BB6E03B470>]
>>> plt.xlabel('Значення x')
Text(0.5,0,'Значення x')
>>> plt.ylabel('Значення x у квадраті')
Text(0,0.5,'Значення x у квадраті')
>>> plt.show()
```

В результаті має вийти ось що:



Функції pyplot.plot() можна передати довільну кількість пар аргументів типів list або array (тип array визначений у модулі numpy).

У прикладі вище передано одну пару: [1,2,3,4], [1,4,9,16]. Перший елемент кожної пари функція розглядає як значення, що відкладаються по осі X, другий елемент як значення осі Y. Відповідно для кожної пари елементів будуватися своя крива на діаграмі.

Після кожної пари даних може бути переданий додатковий аргумент типу string – рядок форматування, який визначає зовнішній вигляд кривої.

Формат рядка запозичений із системи Matlab. Рядок включає три підрядки.

- перший підрядок задає колір графічного елемента,
- друга стиль маркера,
- третій стиль лінії.

У прикладі 'ro' = 'r' + 'o', де 'r' - червоний, 'o' - кружок. За замовчуванням 'b-' – синя безперервна лінія.

Архітектура matplotlib

Одне з основних завдань, що виконує matplotlib - надання набору функцій та інструментів для представлення та управління Figure (так називається основний об'єкт) разом із усіма внутрішніми об'єктами, з якого він складається. Але в matplotlib є інструменти для обробки подій і, наприклад, анімації. Завдяки їм ця бібліотека здатна створювати інтерактивні графіки на основі подій натискання кнопки або руху миші.

Архітектура matplotlib логічно розділена на три шари, розташовані на трьох рівнях. Комунікація непряма - кожен шар може взаємодіяти тільки з тим, що розташований під ним, але не над.

Ось ці шари:

- Шар сценарію
- Художній шар
- Шар бекенда

Шар бекенда

Шар Backend є нижнім на діаграмі з архітектурою усієї бібліотеки. Він містить усі API та набір класів, які відповідають за реалізацію графічних елементів на низькому рівні.

- `FigureCanvas`- Це об'єкт, що втілює область малювання.
- `Renderer`- Об'єкт, який малює по FigureCanvas.
- `Event`- об'єкт, що обробляє введення від користувача (події з клавіатури та миші)

Художній шар

Середнім шаром є художній (artist). Усі елементи, що становлять графік, такі як назва, мітки осей, маркери тощо, є екземплярами цього об'єкта. Кожен їх грає свою роль ієрархічної структурі.

Є два мистецькі класи: примітивний та складовий.

- Примітивний - це об'єкти, які є базовими елементами для формування графічного представлення графіка, наприклад, Line2D, або геометричні фігури, такі як прямокутник коло або навіть текст.
- Складові - об'єкти, що складаються з кількох базових (примітивних). Це осі, шкали та діаграми.

На цьому рівні часто доводиться мати справу з об'єктами, що займають високе становище в ієрархії: графік, система координат, осі. Тому важливо повністю розуміти, яку роль вони відіграють. Нижче наведено три основні художні (складові об'єкти), які часто використовуються на цьому рівні.

- `Figure`- Об'єкт, що займає верхню позицію в ієрархії. Він відповідає всьому графічному уявленню і може містити багато систем координат.
- `Axes`— це цей графік. Кожна система координат належить лише одному об'єкту Figure і має два об'єкти Axis (або три, якщо йдеться про тривимірний графік). Інші об'єкти, такі як назва, мітки x та y, належать окремо осям.
- `Axis`враховує числові значення в системі координат, визначає межі та керує позначеннями на осях, а також відповідним кожному з них текстом. Положення шкал визначається об'єктом Locator, а зовнішній вигляд – Formatter.

Шар сценарію (pyplot)

Художні класи та пов'язані з ними функції (API matplotlib) підходять усім розробникам, особливо тим, хто працює із серверами веб-додатків або розробляє графічні інтерфейси. Але для обчислень, зокрема дляаналізу та візуалізації данихнайкраще підходить шар сценарію. Він включає інтерфейс pyplot.

pylab і pyplot

З погляду користувача існують дві бібліотеки: pylab та pyplot. Pylab – це модуль, що встановлюється разом з matplotlib, а pyplot – внутрішній модуль matplotlib. Обидва часто посилаються в скриптах:

```
від pylab import *

# і

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

Pylab поєднує функціональність pyplot із можливостями NumPy в одному просторі імен, тому окремо імпортувати NumPy не потрібно. Більше того, при імпорті pylab функції з pyplot та NumPy можна викликати без посилання на модуль (простір імен).

```
plot(x,y)
array([1,2,3,4])
# замість
plt.plot()
np.array([1,2,3,4]
```

Пакет pyplot пропонує класичнийінтерфейс Pythonдля програмування, має власний простір має та потребує окремого імпорту NumPy. У наступних матеріалах використається цей підхід. Його ж застосовує більшість програмістів на Python.

Призначення клавіш інтерактивного вікна діаграми.

Кнопка 
(pan/zoom) призначена для прокручування/масштабування діаграми.

Натискаємо на кнопку, покажчик миші набуває вигляду перехрещених стрілок.

Якщо рухати мишу і утримувати ліву кнопку - діаграма прокручуватиметься у напрямку руху покажчика.

Якщо рухати мишу і утримувати праву кнопку – масштаб діаграми буде змінюватися, при русі вгору/вправо – зменшуватись, вниз/вліво – збільшуватись.

При натиснутих клавішах "x" та "y" переміщення/масштабування діаграми відбуватиметься по осях X та Y відповідно.

При натиснутій кнопці Ctrl діаграма буде змінюватися так, щоб збереглися її пропорції (aspect ratio).

Кнопка 
при натисканні включає режим прямокутного масштабування.

Якщо натиснути кнопку вказівник миші набуде вигляду хреста.

При натиснутій лівій кнопці миші на діаграмі можна виділити прямокутну область. Після звільнення кнопки масштаб діаграми буде змінено так, щоб виділена область зайняла якомога більшу площу діаграми.

Кнопки 
дозволяють переміщатися з історії зміни діаграми. Всі зміни, що вносяться в діаграму користувачем (масштабування, прокручування), запам'ятовуються.

Кнопки зі стрілками дозволяють переміщатися вперед/назад за варіантами діаграми.

Кнопка з будиночком повертає діаграму до початкового стану.

Кнопка 
викликає стандартний системи діалог збереження файла.

Дозволяє зберегти діаграму у файлі. Можна вибрати такі формати діаграми: png, pdf, svg, eps, ps.

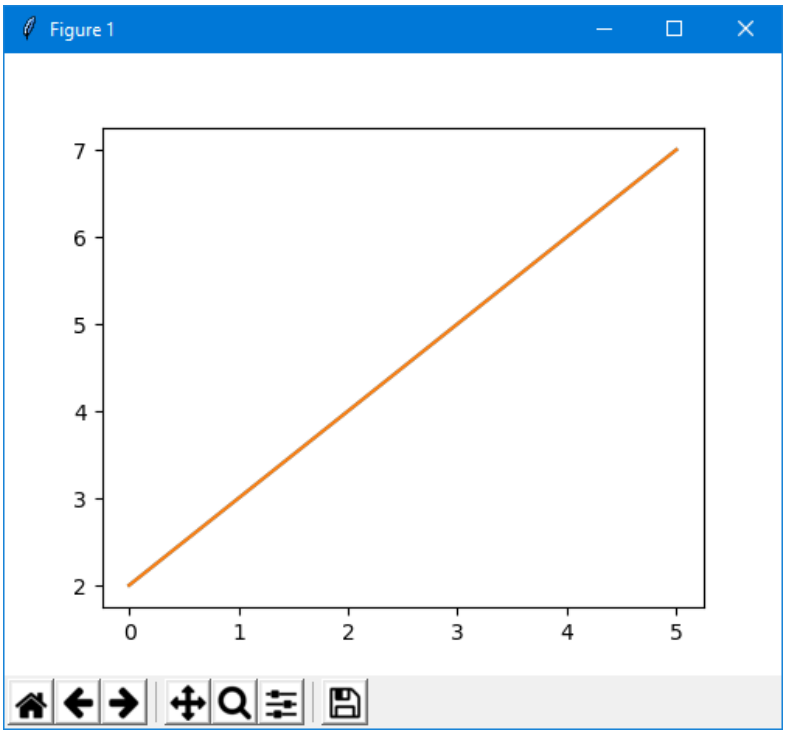
У всіх прикладах імпортуємо модулі:

```
import matplotlib as mpl
# де-факто стандарт виклику pyplot у python
import matplotlib.pyplot as plt
```

П1. Список координат, x координати числа 0, 1, 2, 3...

Список y[2,3,4,5,6,7] координат, x координати утворюють послідовність 0, 1, 2, ...:

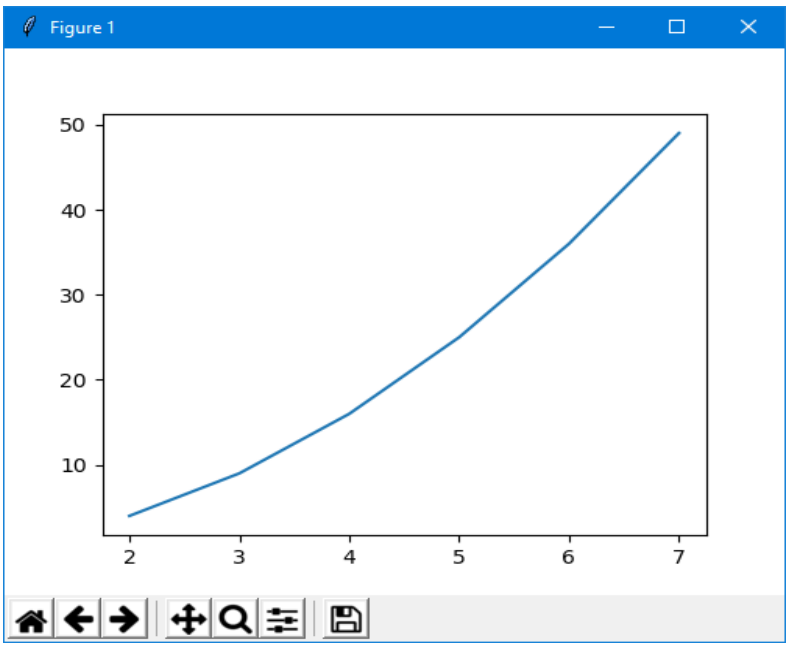
```
plt.plot([2,3,4,5,6,7]);plt.show()
```



П2. Списоки x та y-координат

Список x та y координат - [2,3,4,5,6,7], [4,9,16,25,36,49]);

```
plt.plot([2,3,4,5,6,7],[4,9,16,25,36,49]);plt.show()
```

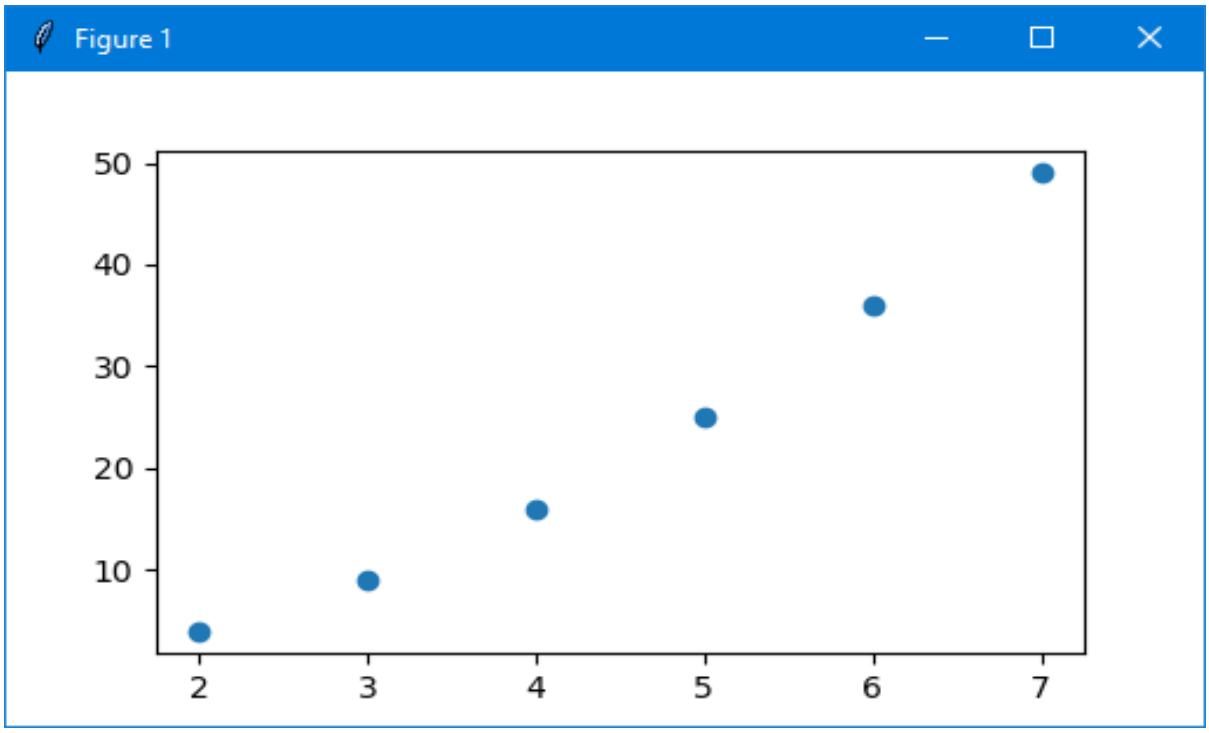


ПЗ. Зображаємо крапки

Список x та y координат - [2,3,4,5,6,7], [4,9,16,25,36,49]);

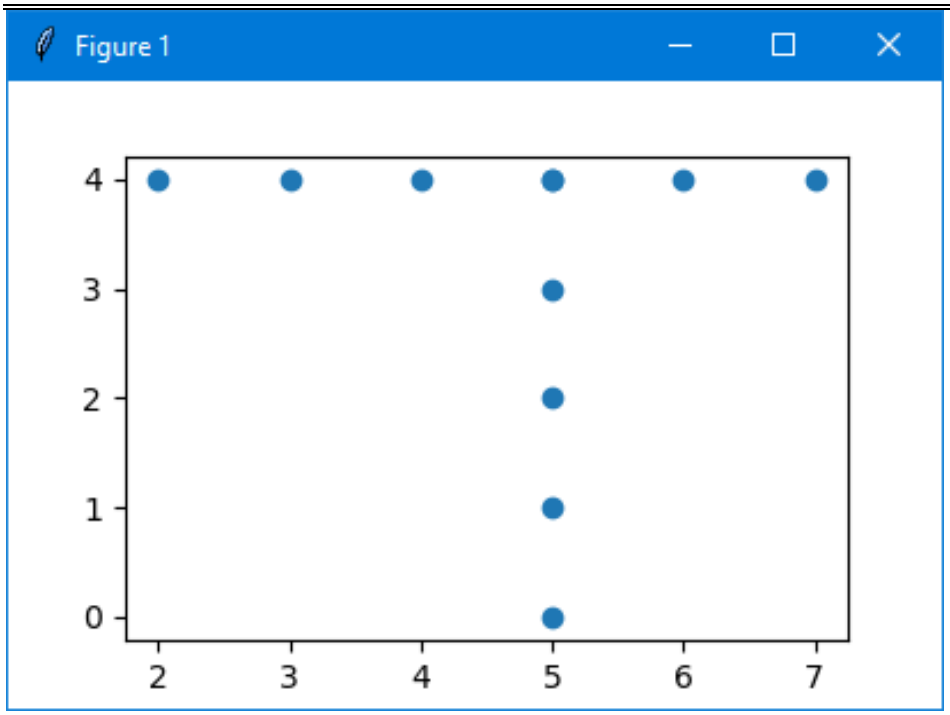
Зображаємо лише крапки

```
plt.scatter([2,3,4,5,6,7],[4,9,16,25,36,49]);plt.show()
```



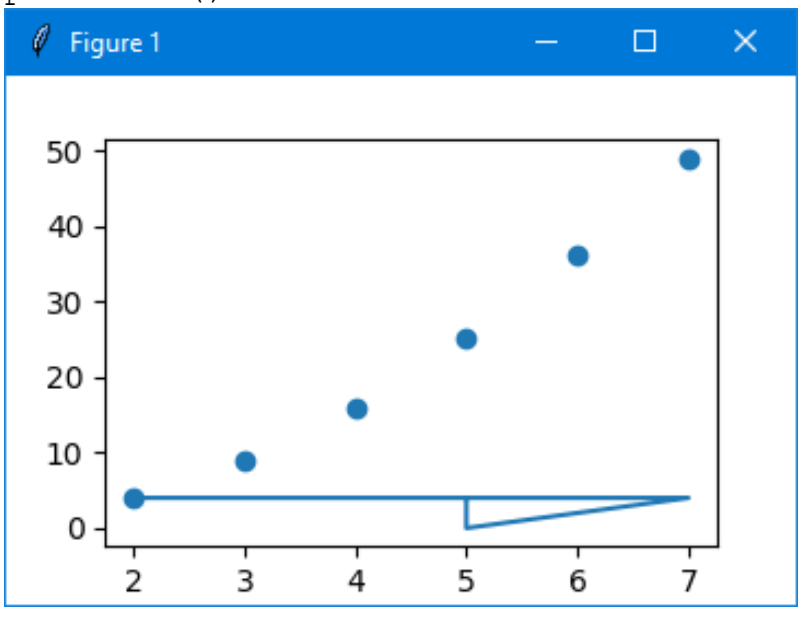
П4. «Довільні» координати

plt.scatter([2,3,4,5,6,7,5,5,5,5,5],[4,4,4,4,4,4,4,0,1,2,3,4,]);plt.show()



П5. Декілька графіків на одному аркуші

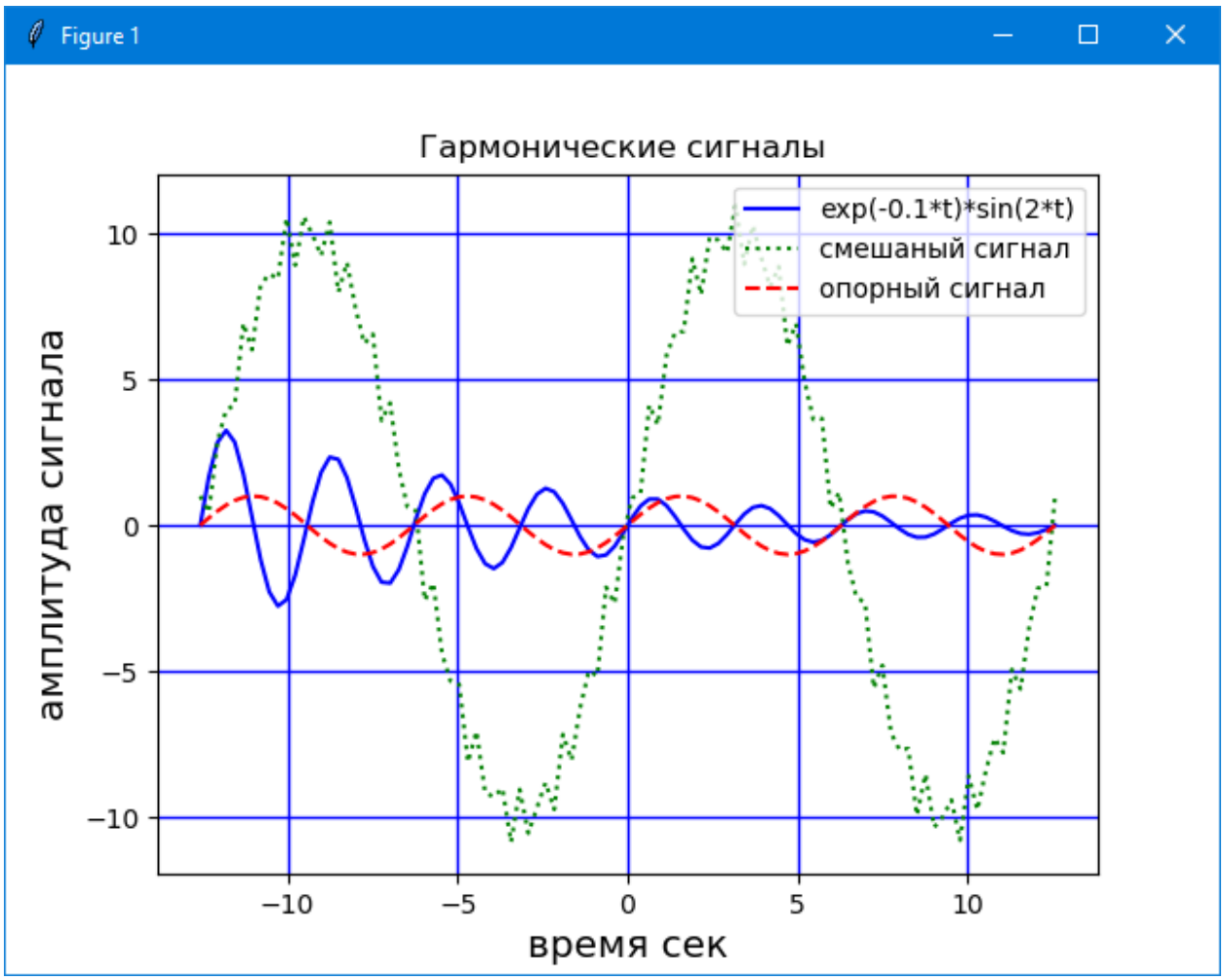
```
plt.scatter([2,3,4,5,6,7],[4,9,16,25,36,49])
plt.plot([2,3,4,5,6,7,5,5,5,5,5],
[4,4,4,4,4,4,0,1,2,3,4,])

plt.show()
```



П5. «Прикрашання» та багато графіків на діаграмі

Малюємо та зберігаємо у файлі три графіки на одній діаграмі в одному масштабі. Дивіться коментарі скрипту.

```
importmatplotlib.pyplot as plt
importnumpy as np
# незалежна змінна
x = np.linspace (-4 * np.pi, 4 * np.pi, 100)
назва осей
xlab='час сек'
ylab='амплітуда сигналу'
перший графік
y1= np.sin(2*x)*np.exp(-0.1*x)
# Легенда
leg1='exp(-0.1*t)*sin(2*t)'
# другий графік
y2=10*np.sin(0.5*x)+np.cos(10*x)
leg2='змішані сигнал'
# третій графік
y3=np.sin(x)
leg3='опорний сигнал'
# Включаємо сітку по осі X та осі Y.
# Задаємо колір товщину сітки
plt.grid(color = 'b',linewidth = 1)
# Задаємо підписи до осей X та осі Y та розмір шрифту
plt.xlabel(xlab, fontsize = 'x-large')
plt.ylabel(ylab, fontsize = 'x-large')
# Задаємо заголовок діаграми
plt.title('Гармонічні сигнали')
# будуємо графіки
plt.plot(x,y1,'b-',label=leg1)
plt.plot(x,y2,'g:',label=leg2)
plt.plot(x,y3,'r--',label=leg3)

# задаємо висновок легенди та її розташування
plt.legend(loc='best')
# Включаємо сітку
plt.grid(True)
# Зберігаємо побудовану діаграму у файл
# Задаємо ім'я файлу та його тип
plt.savefig('signal3.png', format = 'png')
візуалізуємо графіки
plt.show()
```



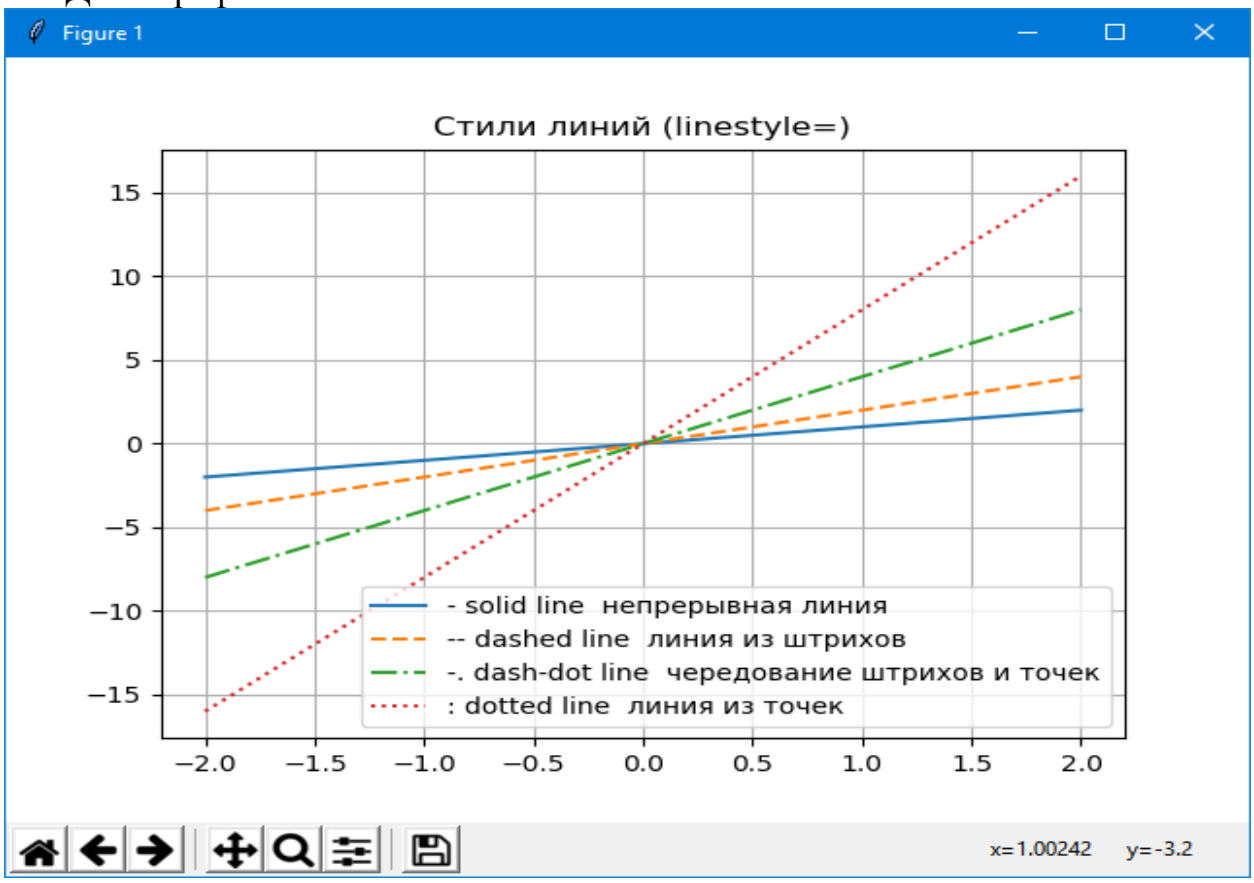
Як задаються стилі ліній, маркери та кольори ліній дивіться у документації або скористайтеся «демо скриптом»:

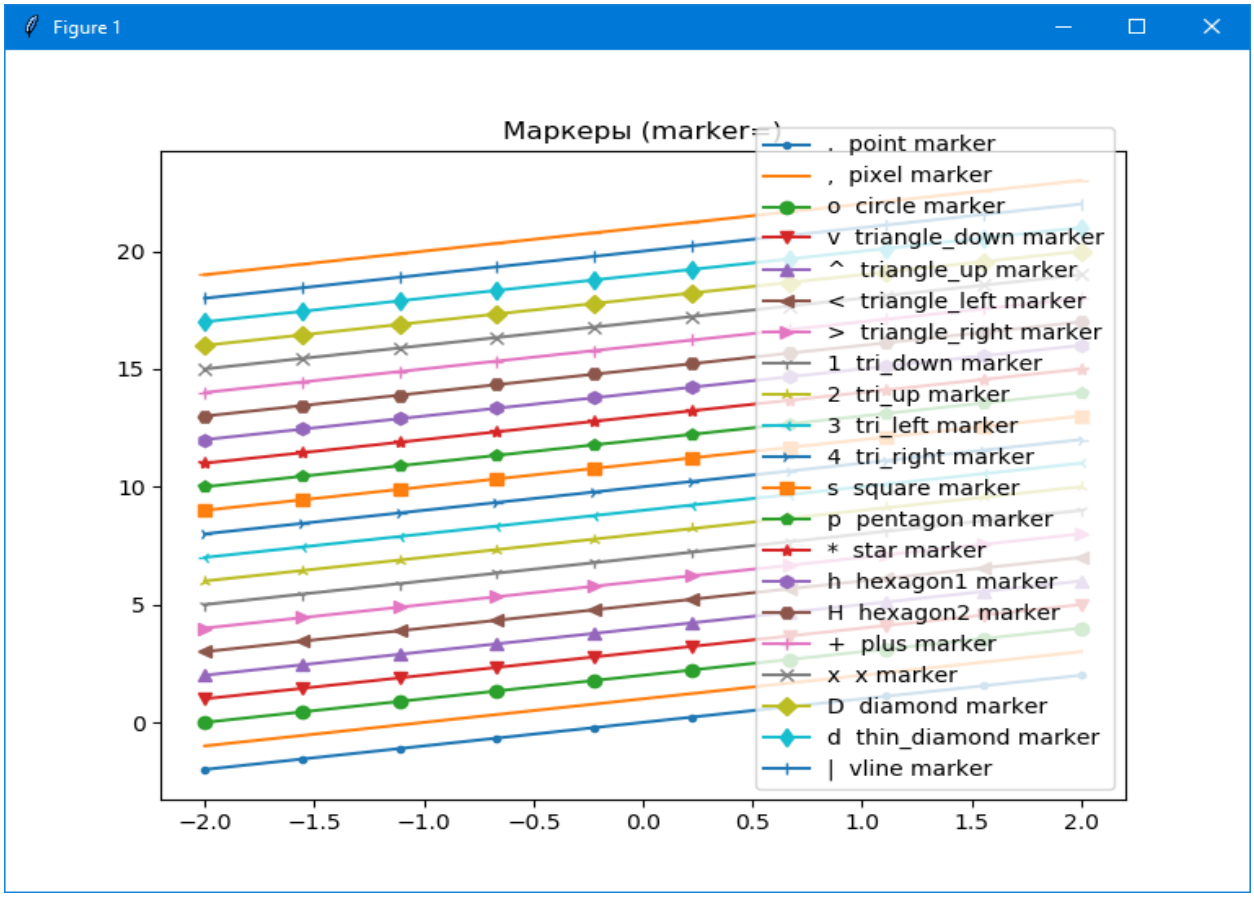
```
importsys
importmath
importos
importmatplotlib.pyplot as plt
importnumpy as np
##sys.exit(0)
importmatplotlib as mpl
# Виведення на екран поточної версії бібліотеки
matplotlib
print('Current version on matplotlib library is',
mpl.__version__)

x = np.linspace(-2, 2, 10)
y1=1*x
y2=2*x
y3=4*x
y4=8*x

s1= ['-', 'solid line', 'безперервна лінія']
```

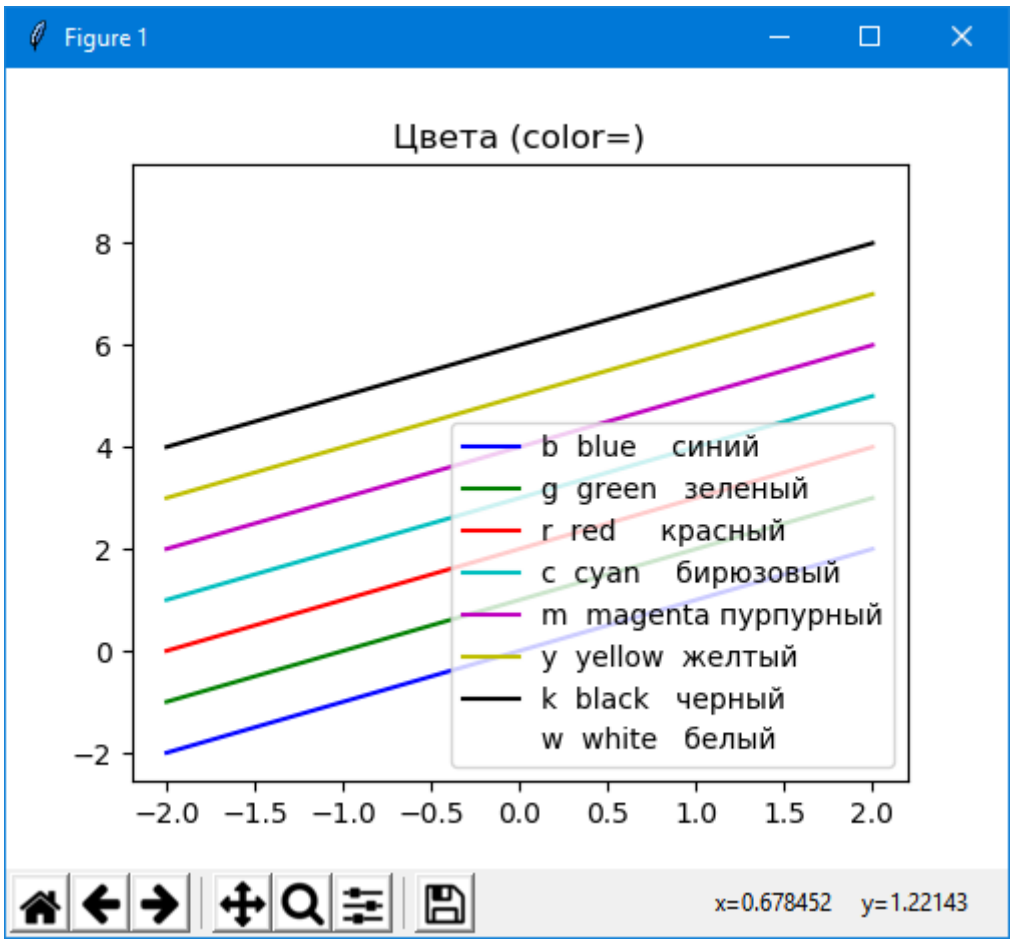
```
s2= ['--', 'dashed line ', 'лінія зі штрихів']
s3= ['-.', 'dash-dot line ', 'чергування штрихів і
точок']
s4= [':', 'dotted line', 'лінія з точок']

leg1=s1[0]+' '+s1[1]+' '+s1[2]
leg2=s2[0]+' '+s2[1]+' '+s2[2]
leg3=s3[0]+' '+s3[1]+' '+s3[2]
leg4=s4[0]+' '+s4[1]+' '+s4[2]

ax=plt.subplot(111)
# можна поміняти розміри вікна графіка у граф. вікні
box=ax.get_position()
ax.set_position([box.x0, box.y0, box.width*1.0 ,
box.height*1.0])

p1 = plt.plot (x, y1, linestyle = s1 [0], label = leg1)
p2=plt.plot(x,y2,linestyle=s2[0], label=leg2)
p3 = plt.plot (x, y3, linestyle = s3 [0], label = leg3)
p4 = plt.plot (x, y4, linestyle = s4 [0], label = leg4)

plt.title('Стилі ліній (linestyle=)')
#ax.legend(loc=(1.0,0.5), mode='expand' )
#ax.legend(mode='expand', bbox_to_anchor=(1, 0.05),loc
# ='upper center' ) #left best
ax.legend(loc='lower right')
# Включаємо сітку
plt.grid()
plt.show()

#sys.exit(0)
# marker
plt.close()
mar=[
'.', 'point marker',
',', 'pixel marker',
'o', 'circle marker',
'v', 'triangle_down marker',
'^', 'triangle_up marker',
'<', 'triangle_left marker',
'>', 'triangle_right marker',
'1', 'tri_down marker',
'2', 'tri_up marker',
'3', 'tri_left marker',
'4', 'tri_right marker',
```

```
's', 'square marker',
'p', 'pentagon marker',
'*', 'star marker',
'h', 'hexagon1 marker',
'H', 'hexagon2 marker',
'+', 'plus marker',
'x', 'x marker',
'D', 'diamond marker',
'd', 'thin_diamond marker',
'|', 'vline marker',
'_', 'hline marker'
]
leg=[]
for i in range(0,len(mar),2):
 leg.append(mar[i]+' '+mar[i+1])
mpl.rcParams['figure.figsize'] = (8.0, 6.0)
kk = int (len (mar) / 2)
for i in range(kk):
    plt.plot(x, x+i, marker=leg[i][0], label=leg[i])
plt.legend(loc='lower right')
plt.title('Маркери (marker=)')
plt.show()

#color
col=[
'b', 'blue синій',
'g', 'green зелений',
'r', 'red червоний',
'c', 'cyan бірюзовий',
'm', 'magenta пурпуровий',
'y', 'yellow жовтий',
'k', 'black чорний',
'w', 'white білий'
]

leg=[]
for i in range(0,len(col),2):
 leg.append(col[i]+' '+col[i+1])
mpl.rcParams['figure.figsize'] = (5.0, 4.0)
kk = int (len (col) / 2)
for i in range(kk):
    plt.plot(x, x+i, color=leg[i][0], label=leg[i])
plt.legend(loc='lower right')
plt.title('Кольори (color=)')
plt.show()
```

Демо графіки:







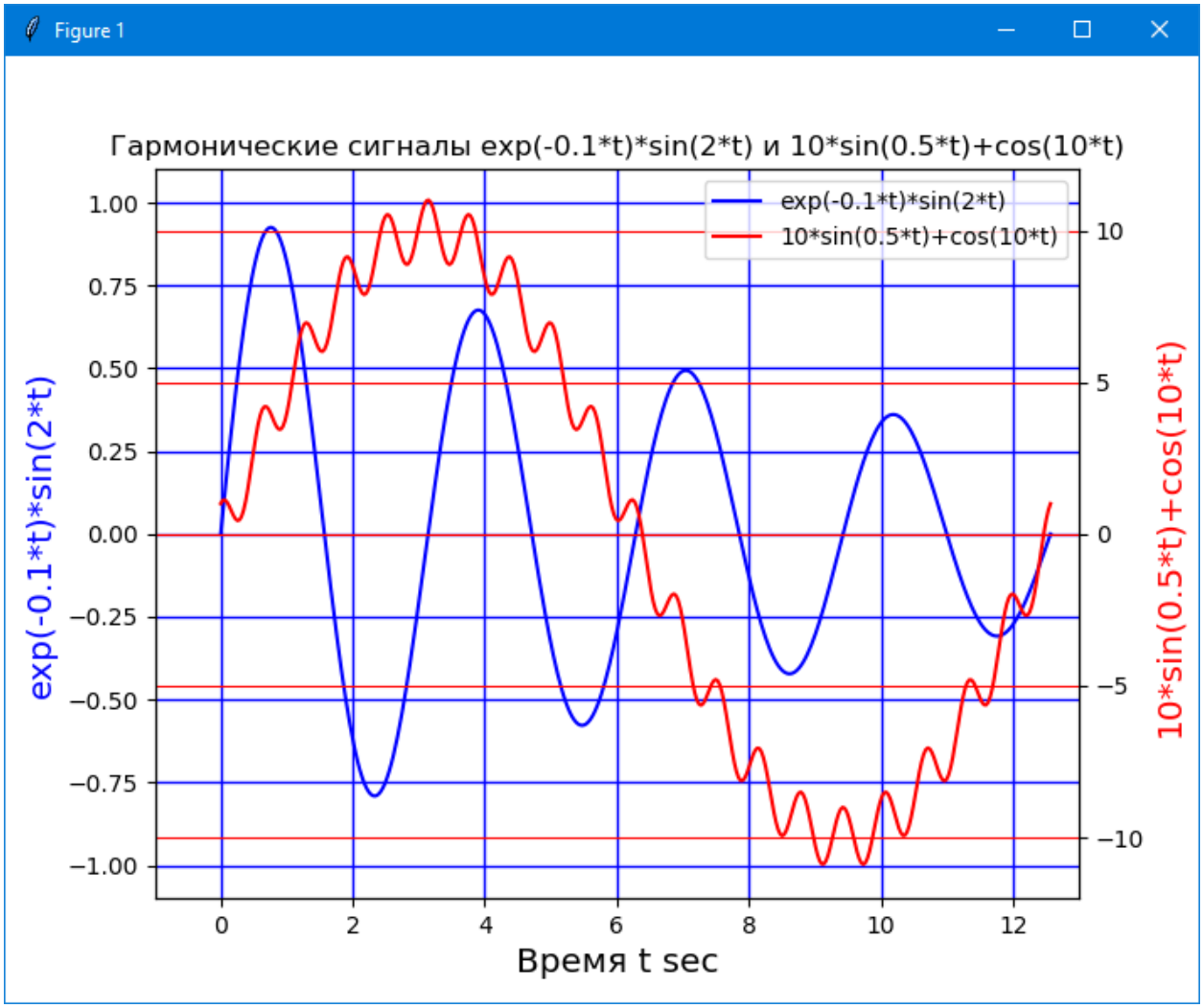
П6. Два графіки на діаграмі в індивідуальних масштабах

Зверніть увагу на «кому» після імені змінної
line_01,= ax_01.plot(X, Y_01, 'b-')

```
# -*- coding: UTF-8 -*-
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
# Значення по осі X час, t
X = np.linspace (0, 4 * np.pi, 400)
# Значення по осях Y_01 & Y_02
leg_01='exp(-0.1*t)*sin(2*t)'
Y_01 = np.sin(2*X)*np.exp(-0.1*X)
# "Модульований" сигнал
leg_02="10*sin(0.5*t)+cos(10*t)"
Y_02=10*np.sin(0.5*X)+np.cos(10*X)
# Задамо розміри зображення діаграми
mpl.rcParams['figure.figsize'] = (8.0, 6.0)
# Будуємо діаграму
```

```
# Отримуємо посилання на об'єкт типу
matplotlib.axes.AxesSubplot, поточну діаграму
ax_01 = plt.axes()
# Задаємо вихідні дані для першої лінії діаграми #
(лінії ступеня розкладання), зовнішній вигляд лінії та
# маркера.
Функція plot() повертає посилання на list,
перший елемент, якого є об'єкт класу #
matplotlib.lines.Line2D
line_01 = ax_01.plot(X, Y_01, 'b-') #, label=leg_01)
# якщо потрібно надалі використовувати об'єкт класу #
matplotlib.lines.Line2D

# то застосовують ДВА варіанти
# 1) line_01 = ax_01.plot(X, Y_01, 'b-') # прийом
першого елемента списку

# 2) line_01 (без коми) = … використовувати line_01 [0]
# Задаємо інтервали значень по осях X та # основної осі
Y
ax_01.axis([-1, 13, -1.1, 1.1])
# Включаємо сітку по осі X та основної осі Y. # Задаємо
колір сітки
ax_01.grid(color = 'b',linewidth = 1)
# Задаємо підписи до осей X та основної осі Y # та
розмір фонту
ax_01.set_xlabel('Час t sec', fontsize = 'x-large')
ax_01.set_ylabel(leg_01, color = 'b',
                                fontsize = 'x-large')
# Задаємо заголовок діаграми
ax_01.set_title('Гармонічні сигнали '+ leg_01+" та "+
leg_02)
#ax_01.legend( loc='upper right')
# Включаємо додаткову вісь Y
ax_02 = ax_01.twinx()
# Задаємо вихідні дані для другої лінії діаграми,
зовнішній вигляд лінії і маркера.
# Функція plot() повертає посилання на об'єкт класу #
matplotlib.lines.Line2D (див. вище)
line_02, = ax_02.plot(X, Y_02, 'r-') #, label=leg_02)
# Задаємо інтервали значень по осях X# та додаткової
осі Y
ax_02.axis([-1, 13, -12, 12])
# Задаємо підпис до додаткової осі Y
```

```
ax_02.set_ylabel(leg_02,color='r', fontsize = 'x-
large')

# Задаємо вихідні дані для легенди та місце її
розміщення
ax_02.legend((line_01, line_02), (leg_01, leg_02), loc
= 'best')
#ax_02.legend(loc='lower left')
# Включаємо сітку додаткової осі Y. # Задаємо колір
сітки
ax_02.grid(color = 'r')
# Зберігаємо побудовану діаграму у файл
# Задаємо ім'я файлу та його тип
#plt.savefig('trigo.png', format = 'png')
візуалізуємо
plt.show()
```



П7 Кілька графіків в одному та в різних графічних вікнах

Робота з matplotlib заснована на використанні графічних вікон та осей (осі дозволяють задати деяку графічну область). Усі побудови застосовуються до поточних осей. Це дозволяє зображувати декілька графіків в одному графічному вікні.

За замовчуванням створюється одне графічне вікно figure(1) та одна графічна область subplot(111) у цьому вікні.

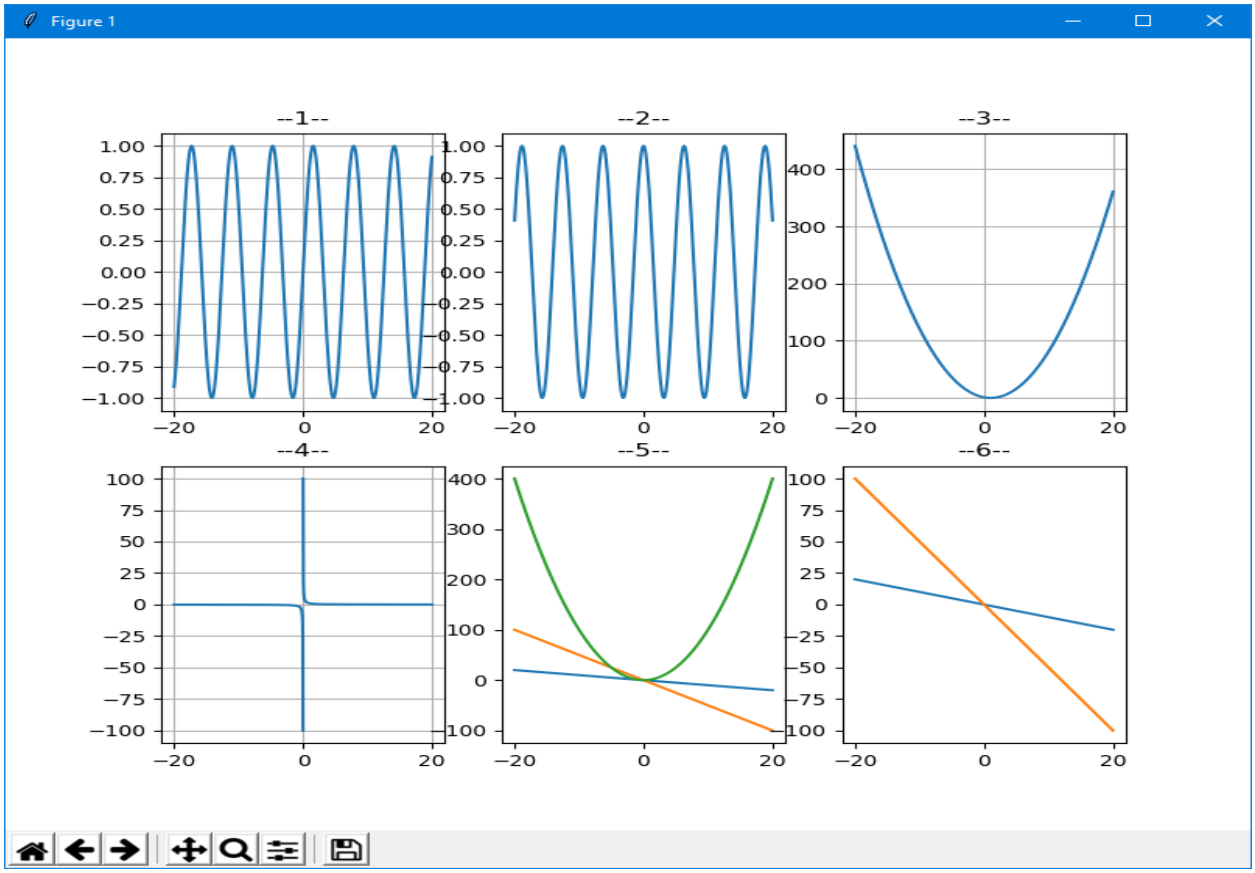
Команда subplot дозволяє розбити графічне вікно на кілька областей. Вона має три параметри: nr; nc; np:

- параметри nr та nc визначають кількість рядків та стовпців на які розбивається графічна область
- параметр np визначає номер поточної області (np набуває значення від 1 до nr*nc).

Якщо nr*nc<10, передавати параметри nr,nc,np можна без використання коми. Наприклад, допустимі форми subplot(2,2,1) та subplot(221).

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

# Імпортуємо пакет із допоміжними функціями
import matplotlib.mlab as ml

# Малюватимемо графік цієї функції
def func (x):
    """
    sinc(x)
    """
    if x == 0:
        return 1.0
    return math.sin(x)/x

# Інтервал зміни змінної по осі X
xmin = -20.0
xmax = 20.0
# Крок між точками
dx = 0.01
# Створимо список координат по осі X на відрізку [-
xmin; xmax], включаючи кінці
xlist = ml. frange (xmin, xmax, dx)
# Обчислимо значення функції у заданих точках
ylist = [func(x) for x in xlist]
```

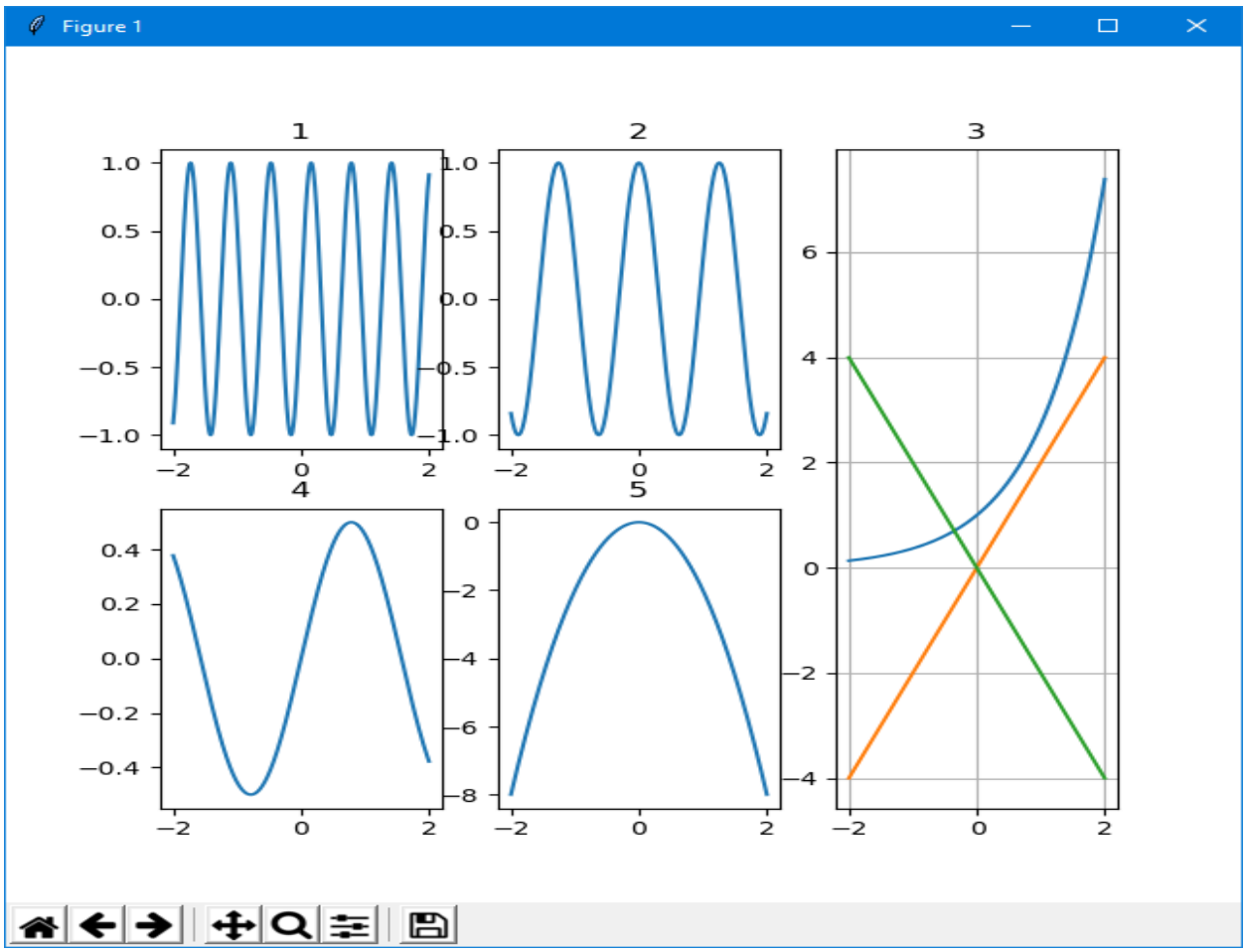
```
# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 1
plt.subplot (2, 3, 1)
plt.plot (xlist, np.sin(xlist) ) #ylist)
plt.grid(True)
plt.title ("--1--")

# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 2
plt.subplot (2, 3, 2)
plt.plot (xlist, np.cos(xlist))
plt.title ("--2--")

# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 3
plt.subplot (2, 3, 3)
plt.plot (xlist, (xlist-1)*(xlist-1))
plt.grid(True)
plt.title ("--3--")

# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 4
plt.subplot (2, 3, 4)
plt.plot (xlist, 1/xlist)
plt.grid(True)
plt.title ("--4--")

# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 5
plt.subplot (2, 3, 5)
plt.plot (xlist, -xlist)
plt.plot (xlist, -5*xlist)
plt.plot (xlist, xlist * xlist)
plt.title ("--5--")

# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 6
plt.subplot (2, 3, 6)
plt.plot (xlist, -xlist)
plt.plot (xlist, -5*xlist)
plt.title ("--6--")

# Покажемо вікно з намальованим графіком
plt.show()
```



Всі ці осередки суто умовні, тому для кожного виклику subplot() розбиття може бути своє, що дозволяє зробити, наприклад, наступне розташування графіків (зверніть увагу на нумерацію осередків):

```
importmath
importnumpy as np
importmatplotlib.pyplot as plt
# Імпортуємо пакет із допоміжними функціями
importmatplotlib.mlab as ml
# Інтервал зміни змінної по осі X
xmin = -2.0
xmax = 2.0
# Крок між точками
dx = 0.001
# Створимо список координат по осі X на відрізку [-
xmin; xmax], включаючи кінці
x = ml. frange (xmin, xmax, dx)
# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 1
plt.subplot (2, 3, 1)
plt.plot (x, np.sin(10*x))
plt.title ("1")
# !!! Два рядки, три стовпці.
```

```
# !!! Поточний осередок - 2
plt.subplot (2, 3, 2)
plt.plot (x, np.cos(5*x) )
plt.title ("2")
# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 4
plt.subplot (2, 3, 4)
plt.plot (x, np.sin(x)*np.cos(x))
plt.title ("4")
# !!! Два рядки, три стовпці.
# !!! Поточний осередок - 5
plt.subplot (2, 3, 5)
plt.plot (x, -2*x*x)
plt.title ("5")
# !!! Один рядок, три стовпці.
# !!! Поточний осередок - 3
plt.subplot (1, 3, 3)
plt.plot (x, np.exp(x))
plt.plot (x, 2*x)
plt.plot (x, -2*x)
plt.grid(True)
plt.title ("3")
# Покажемо вікно з намальованим графіком
plt.show()
```



figure() та axes()

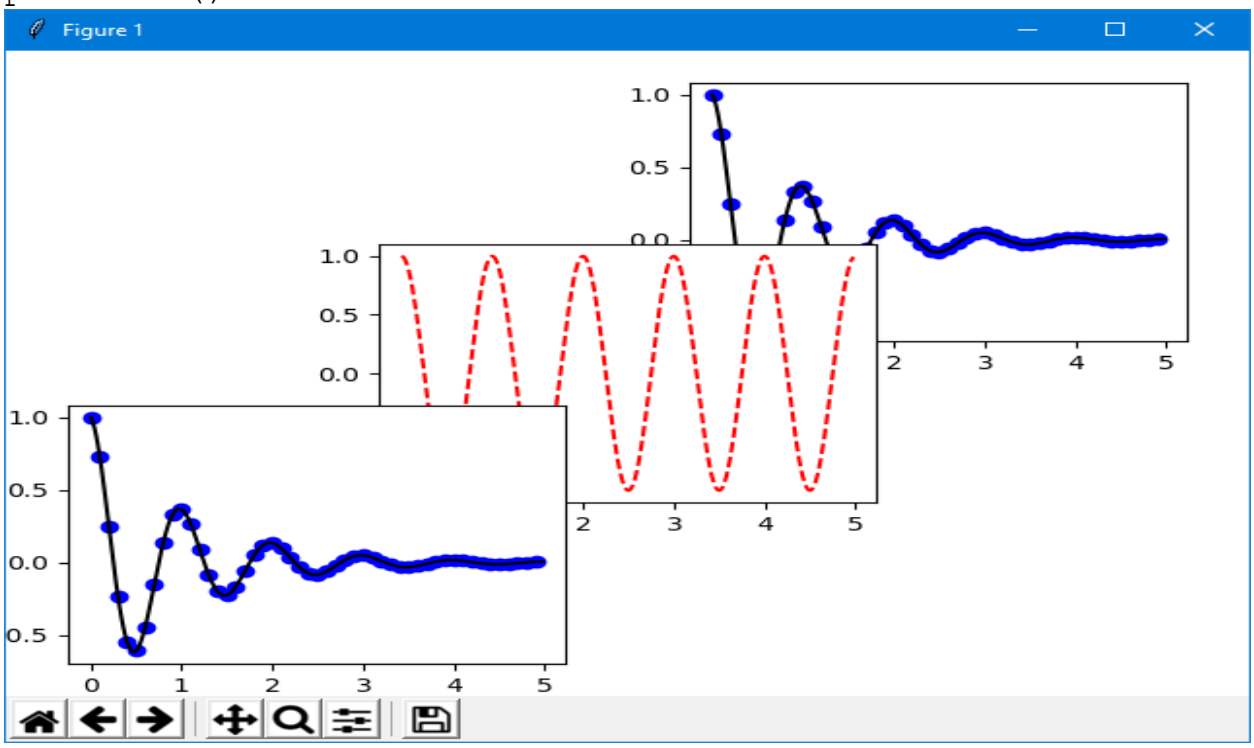
Є можливість розмістити осі (підвіконня) вручну, наприклад не на прямокутній сітці, а як-небудь ялинкою, використовуйте команду axes(), яка дозволяє визначити положення осей як

axes([Left, bottom, width, height]),

де всі значення змінюються від 0 до 1. Виглядати це може так:

```
importmath
importnumpy as np
importmatplotlib.pyplot as plt
# Імпортуємо пакет із допоміжними функціями
importmatplotlib.mlab as ml
importnumpy as np
importmatplotlib.pyplot as plt
deff(t):
    returnnp.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.figure(1)
plt.axes([0.55, 0.55, 0.4, 0.4])
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.axes([0.3, 0.3, 0.4, 0.4])
```

```
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.axes([0.05, 0.05, 0.4, 0.4])
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
# Покажемо вікно з намальованим графіком
plt.show()
```

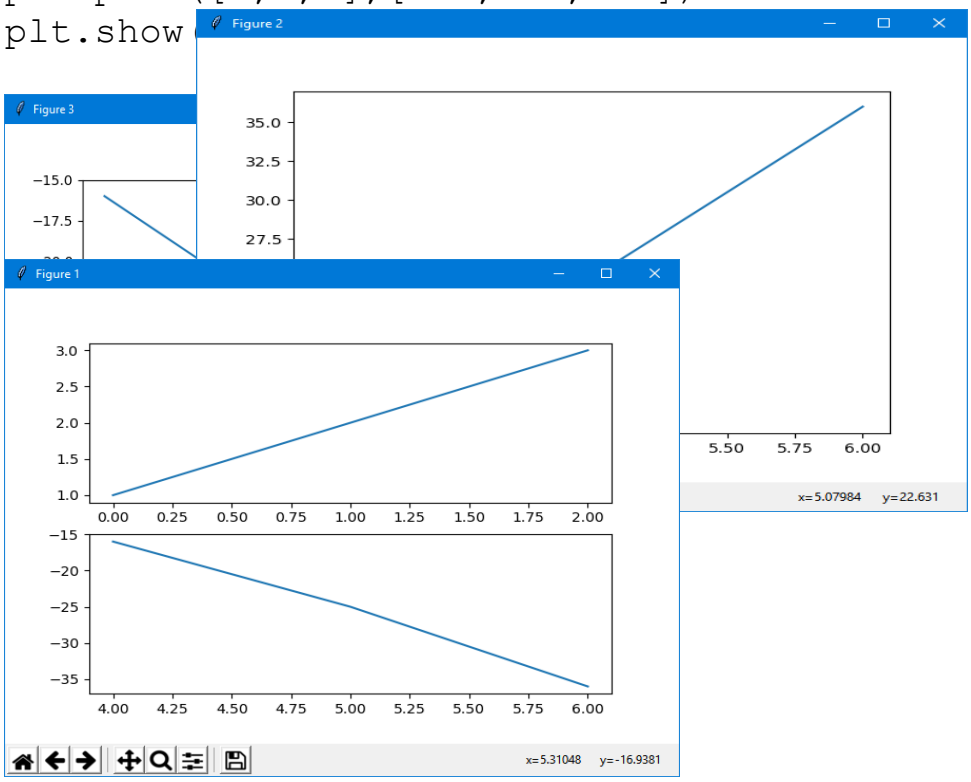


Є можливість створити кілька вікон, викликаючи figure() з номером вікна, що збільшується. Звичайно, кожне вікно може містити кілька осей та підвіконь.

Очистити активне вікно можна clf() , а активні осі за допомогою cla().

```
importmath
importnumpy as np
importmatplotlib.pyplot as plt
# Імпортуємо пакет із допоміжними функціями
importmatplotlib.mlab as ml
importnumpy as np
importmatplotlib.pyplot as plt

plt.figure(1) # the first figure
plt.subplot(211) # 1
plt.plot([1,2,3])
plt.subplot(212) # second subplot in first figure
plt.plot([4,5,6],[-16,-25,-36])
plt.figure(2) # a second figure
plt.plot([4,5,6],[16,25,36]) # creates a subplot(111)
by default
plt.figure(3)
```

```
plt.plot([4,5,6],[-16,-25,-36])
plt.show
```



Встановити свій діапазон осей

```
plt.xlim(right=xmax) #xmax is your value
plt.xlim(left=xmin) #xmin is your value
plt.ylim(top=ymax) #ymax is your value
plt.ylim(bottom=ymin) #ymin is your value
```

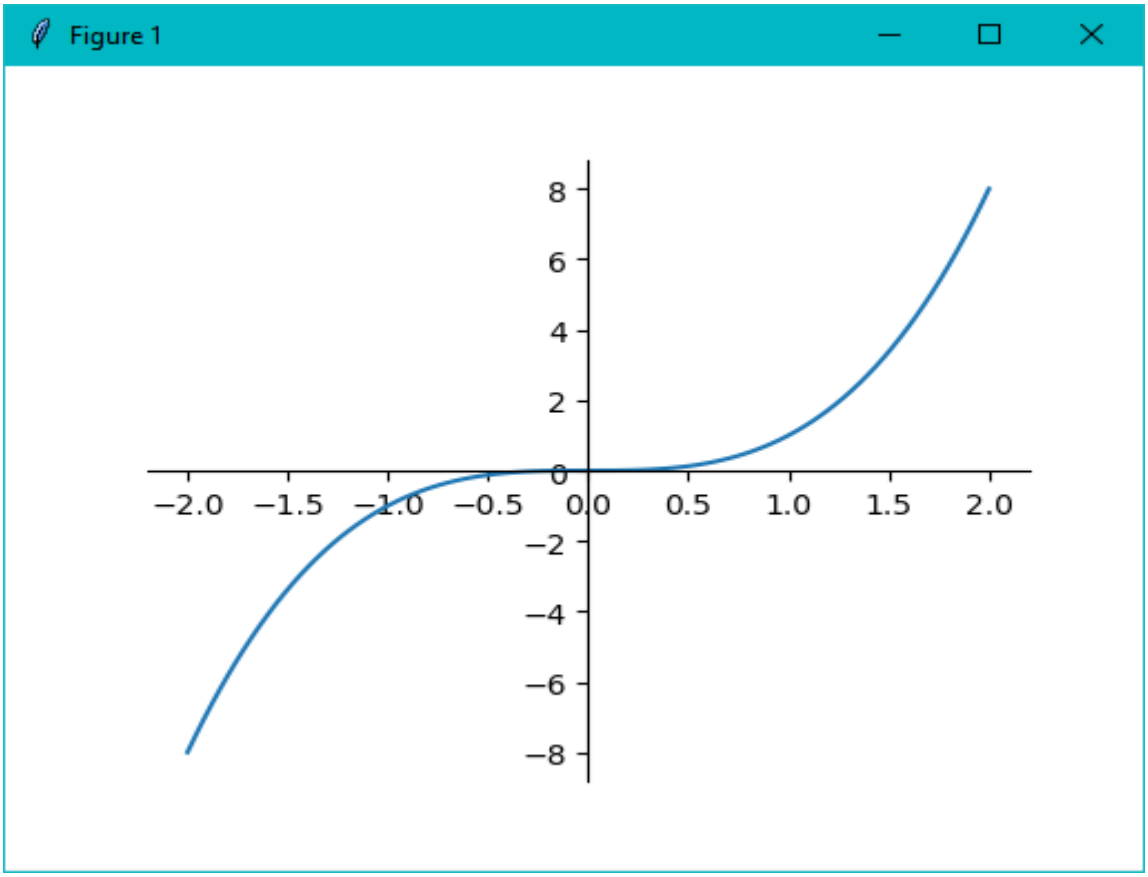
Або так:

```
plt.ylim(ymin, ymax)
plt.xlim(xmin, xmax)
```

Перенесення координатних осей до центру графіка

Якщо необхідно перенести координатні осі в центр малюнка, можна вчинити так:

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(-2,2,100)
Y = X**3
plt.plot(X, Y)
ax = plt.gca()
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```

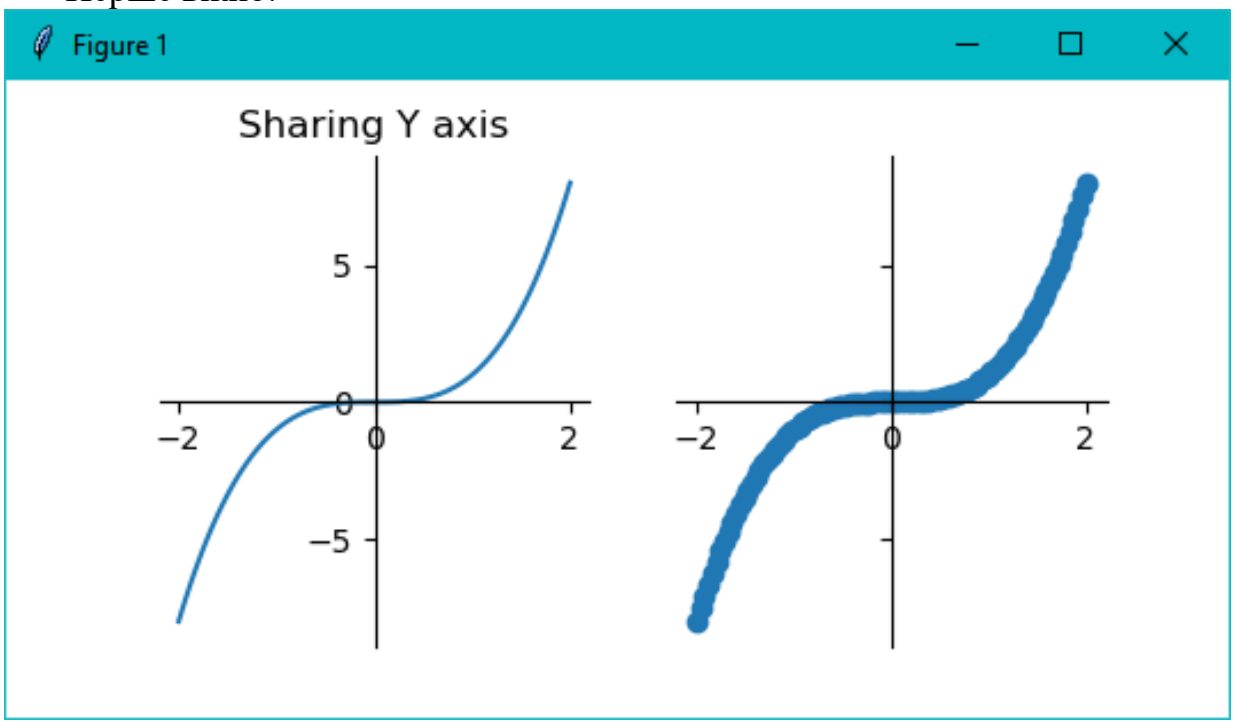


Приклад перенесення осей для subplot:

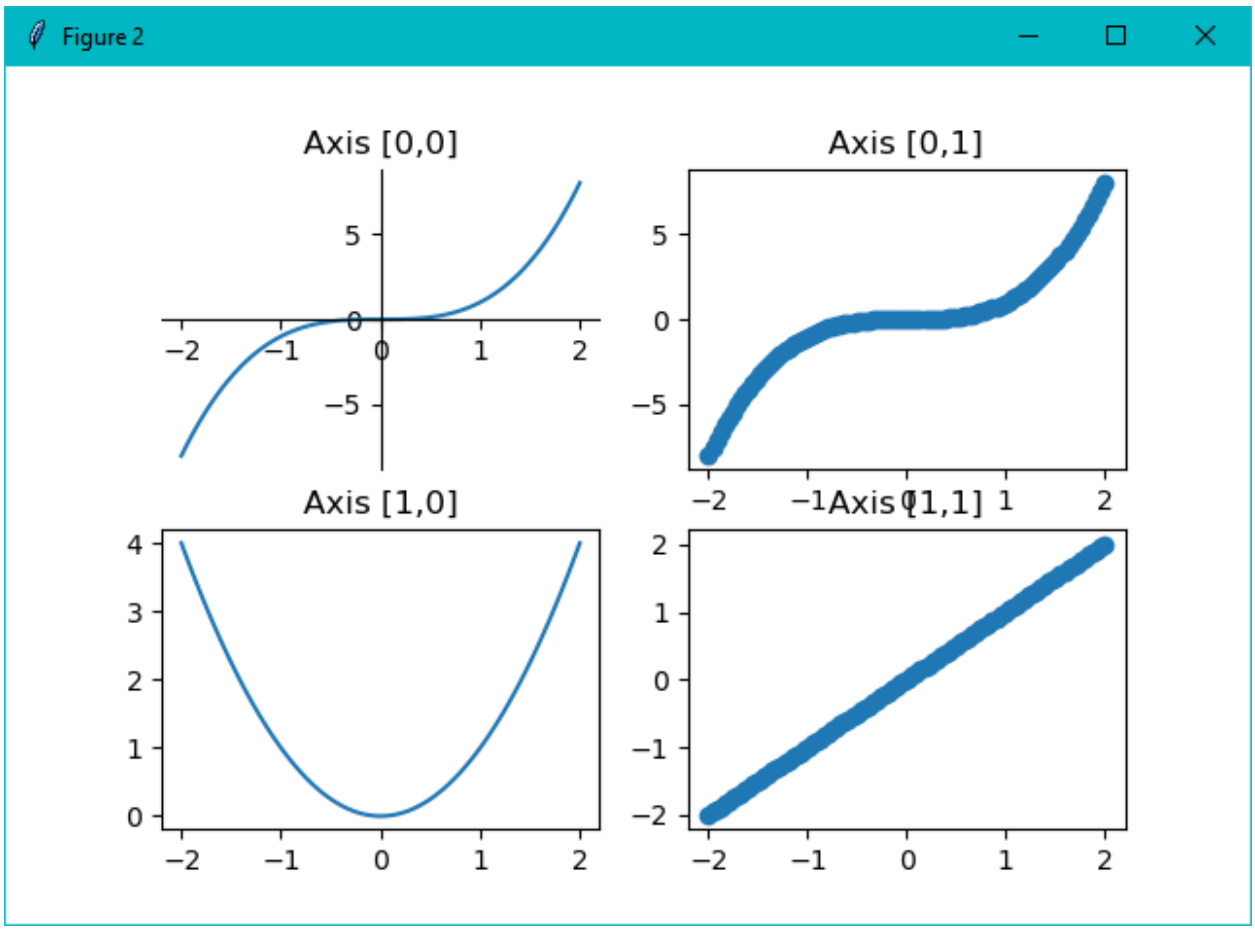
```
#Приклад отримання осей для subplots:
# Two subplots, unpack the axes array immediately
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(X, Y)
ax1.set_title('Sharing Y axis')
ax1.spines['left'].set_position('center')
ax1.spines['bottom'].set_position('center')
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)

ax2.scatter(X, Y)
ax2.spines['left'].set_position('center')
ax2.spines['bottom'].set_position('center')
ax2.spines['top'].set_visible(False)
ax2.spines['right'].set_visible(False)

f, axarr = plt.subplots(2, 2) # Four axes, returned as
a 2-d array


axarr[0, 0].plot(X, Y)
axarr[0, 0].spines['left'].set_position('center')
```

```
axarr[0, 0].spines['bottom'].set_position('center')
axarr[0, 0].spines['top'].set_visible(False)
axarr[0, 0].spines['right'].set_visible(False)
axarr[0, 0].set_title('Axis [0,0]')

axarr[0, 1].scatter(X, Y)
axarr[0, 1].set_title('Axis [0,1]')
axarr[1, 0].plot(X, X ** 2)
axarr[1, 0].set_title('Axis [1,0]')
axarr[1, 1].scatter(X, X ** 1)
axarr[1, 1].set_title('Axis [1,1]')

plt.show()
```

Перше вікно:



Друге вікно:

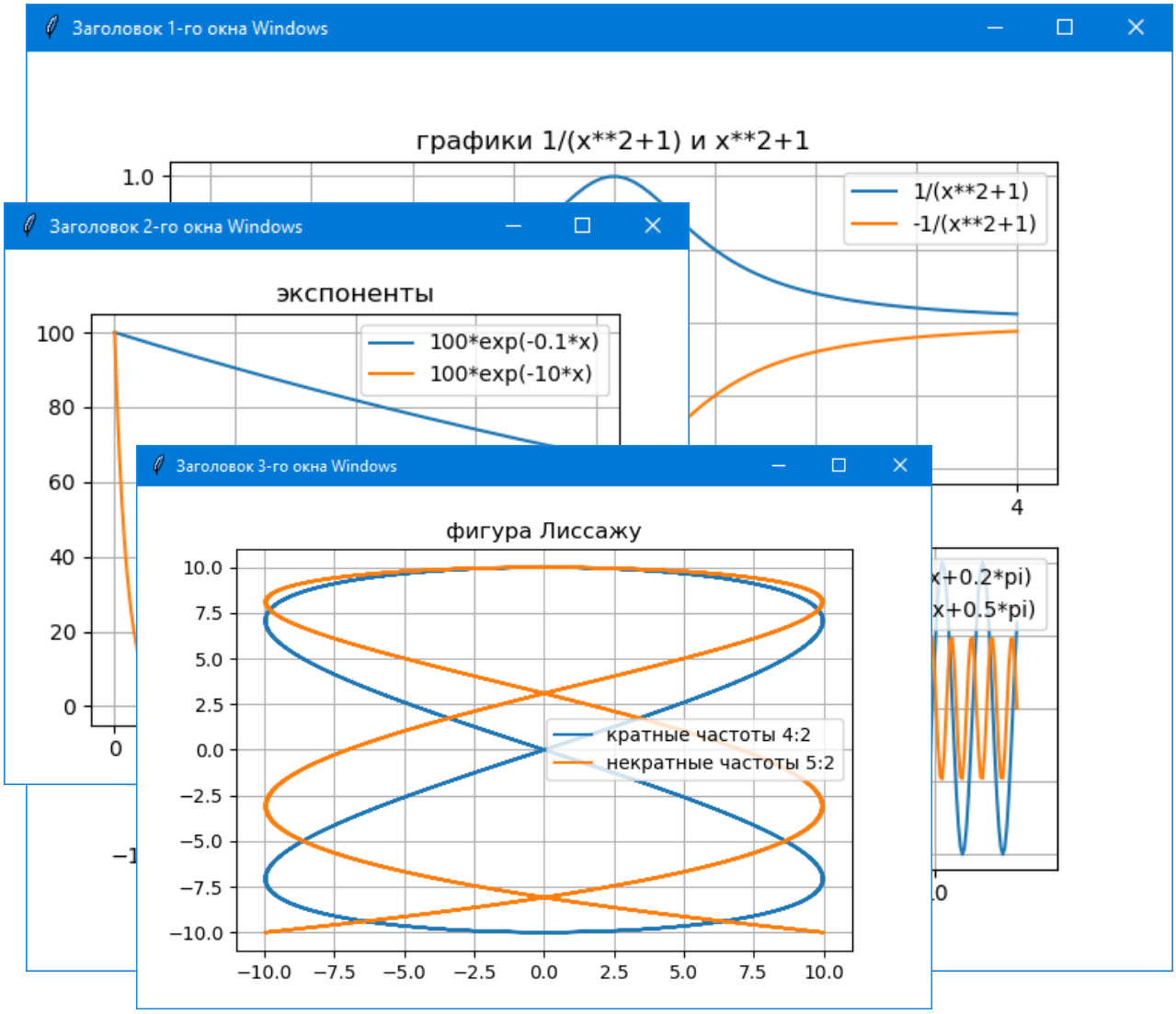


П8 Написи на вікнах діаграм і вікнах Windows, зміна розміру вікна

Наведемо приклад, аналогічний розглянутому вище. Для всіх вікон та діаграм задані титульні (тульбарні) написи:

```
importmath
importnumpy as np
importmatplotlib.pyplot as plt
# Імпортуємо пакет із допоміжними функціями
importmatplotlib as mpl
importnumpy as np
importmatplotlib.pyplot as plt

x1=np.linspace(-4,4,100)
y1=1/(x1**2+1)
y11=-1/(x1**2+1)
# Задамо розміри зображення діаграми
mpl.rcParams['figure.figsize'] = (8.0, 6.0)
plt.figure("Заголовок 1-го вікна Windows") # перше #
вікно windows

plt.subplot(211) # перші графіки у цьому вікні
```

```
plt.plot(x1,y1,label='1/(x**2+1)')
plt.plot(x1,y11,label='-1/(x**2+1)')
plt.title("графіки 1/(x**2+1) та x**2+1" )
plt.grid(True)
plt.legend()
plt.subplot(212) # другі графіки у цьому вікні
x2=np.linspace(-4*np.pi,4*np.pi,400)
plt.plot(x2,10*np.sin(5*x2+0.2*np.pi),label='10*sin(5*x
+0.2*pi)')
plt.plot(x2,5*np.cos(10*x2+0.5*np.pi),label="5*cos(10*x
+0.5*pi)")
plt.title("тригонометричні функції")
plt.grid(True)
plt.legend(loc='best')

plt.figure("Заголовок 2-го вікна Windows") # a second
figure
# або так
#fig = plt.figure()
#fig.canvas.set_window_title("Заголовок 2-го вікна
Windows")
x3=np.linspace(0,4,400)
plt.plot(x3, 100*np.exp(-0.1*x3), label='100*exp(-
0.1*x)')
plt.plot(x3, 100*np.exp(-10*x3), label='100*exp(-
10*x)')
plt.title("експоненти")
plt.grid(True)
plt.legend(loc='best')
fig3=plt.figure()
fig3.canvas.set_window_title("Заголовок 3-го вікна
Windows")
plt.grid(True)
t=np.linspace(0,8*np.pi,500)
plt.plot(10*np.sin(4*t),10*np.cos(2*t), label="кратні
частоти 4:2")

plt.plot(10*np.sin(5*t),10*np.cos(2*t),
                        label="неразові частоти 5:2")
plt.title("фігура Лісажу")
plt.grid(True)
plt.grid(True)
plt.legend(loc='best')

plt.show()
```



П9. Параметричний графік

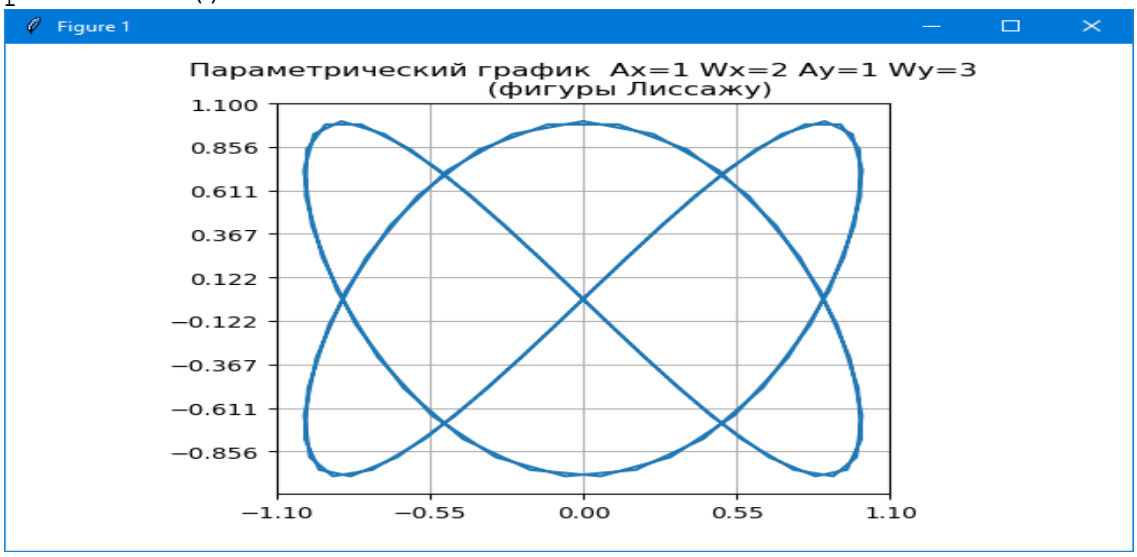
Масив x повинен бути монотонно зростаючим. Можна будувати будь-яку параметричну лінію x = x (t), y = y (t).

Роздільні можливості монітора по різних осях можуть бути різними. Тому щоб кола виглядали як кола, а не як еліпси (а квадрати як квадрати, а не як прямокутники), потрібно встановити параметр - aspect ratio (зазвичай рівний 1).

Побудуємо x = Ax * cos (Wx * t), y = Ay * sin (Wy * t).

```
importmatplotlib.pyplot as plt
importnumpy as np
Ax=1; Ay=1; Wx = 2; Wy=3
# незалежна змінна
```

```
t=np.linspace(0,4*np.pi,100)
# Встановимо параметр - aspect (ratio)
axx=plt.subplot(111)
axx.set_aspect(1)
так можна змінити крок сітки на графіку
plt.xticks([i for i in \ np.linspace(-1.1*Ax, 1.1*Ax,
5)])
plt.yticks([i for i in \ np.linspace(-1.1*Ay, 1.1*Ay,
10)]))
# Задаємо заголовок діаграми
plt.title("Параметричний графік" \
          " Ax={0} Wx={1} Ay={2} Wy={3}\n" \
          "(Фігури Лісажу)" \
               .format(Ax, Wx, Ay, Wy))
plt.plot(Ax*np.sin(Wx*t),Ay*np.cos(Wy*t))
plt.grid(True)
візуалізуємо графіки
plt.show()
```



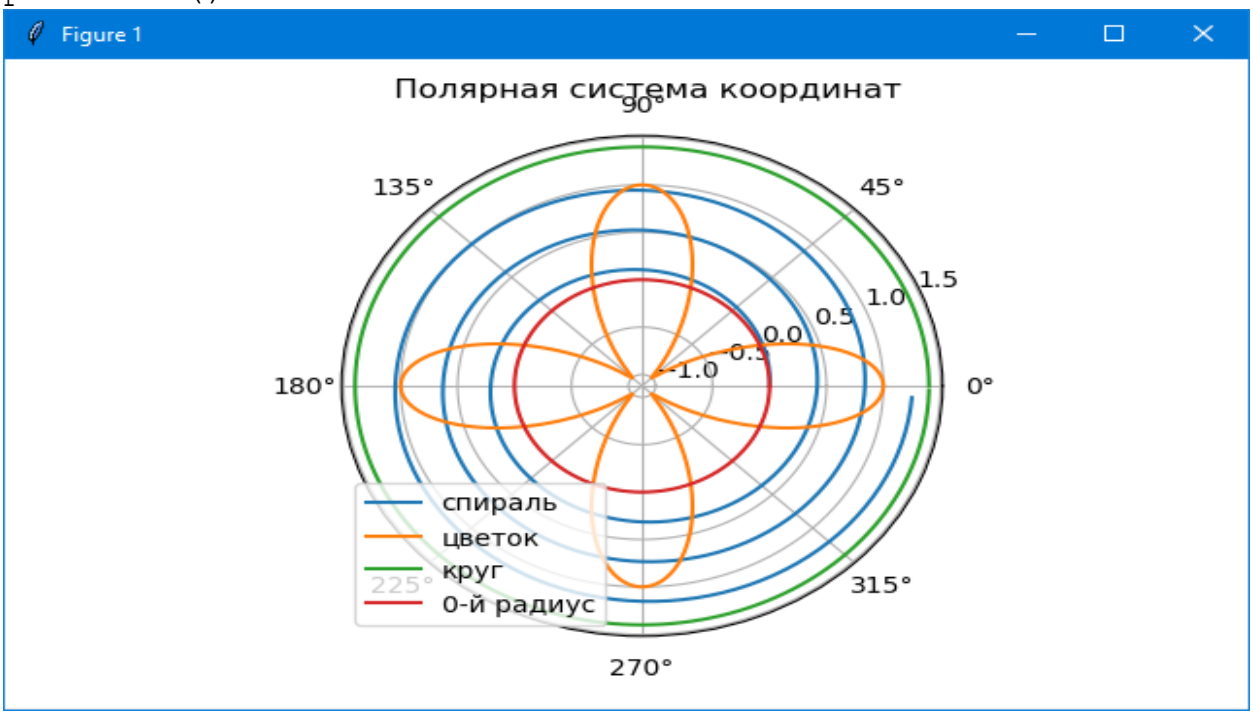
П10. Полярні координати

Крім часто використовуваної декартової системи координат, досить широко застосовується і полярна система координат, зручна в різних радіальних задачах, координати точок в ній задається за допомогою радіус-вектора ρ, що йде з початку координат і кута θ. Кут може бути заданий у радіанах або градусах, matplotlib використовує градуси.

Чотири графіки в полярних координатах:

```
import matplotlib.pyplot as plt
import numpy as np
theta = np.arange(0., 2., 1./180.) * np.pi
#plt.title('Поляна система координат')
plt.polar(3*theta, theta/5, label="спіраль");
plt.polar(theta, 0+np.cos(4*theta), label="квітка");
```

```
plt.polar(theta, [1.4] * len (theta), label = "коло");
plt.polar(theta, 0*theta, label="0-й радіус");
plt.title("Полярна система координат")
lt.grid(True)
plt.legend(loc='lower left')
візуалізуємо графіки
plt.show()
```



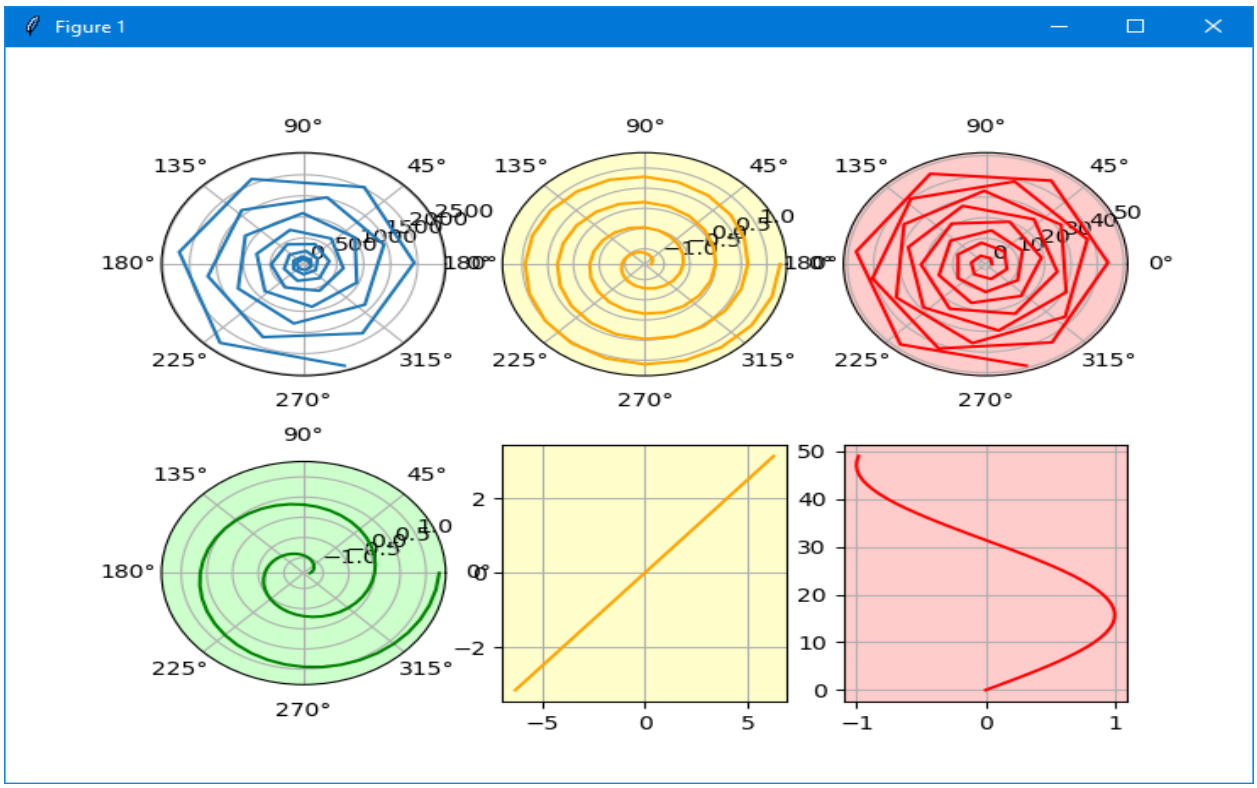
Для побудови в полярних координатах використовується функція polar(). У аргументах першими йдуть кути, потім радіуси. У прикладі використовується той самий масив для кутів і радіус-векторів і малюється чотири різні криві.

Перша утворює спіраль, оскільки з кожним новим кутом змінюється і радіус, друга малює квітку відповідно до рівняння троянди, третя і четверта - кола через незмінність радіусу для всіх кутів (в даному випадку він дорівнює 1.4 і 0).

Ще один приклад:

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

fig = plt.figure (figsize = (8., 6.))
x = np.arange(50)
y = x**2 + 2.5

lag = 0.05 * np.pi
r = np.arange(0.0, 5.0, 0.1)
# phi = r * np.pi
phi = np.arange (-2 * np.pi, 2 * np.pi + lag, lag)
r=phi*0.2
# x та y
```

```
ax1 = fig.add_subplot(231, projection='polar')
ax1.plot(x, y)
ax1.grid(True)

ax2 = fig.add_subplot(232, projection='polar',
facecolor ='#FFFFCC')
ax2.plot(phi*2., r, 'orange') # facecolor='#FFFFCC') #
axisbg
ax2.grid(True)

ax3 = fig.add_subplot(233, projection='polar',
facecolor='#FFCCCC')
ax3.plot(x, x, 'r')
ax3.grid(True)

ax4 = fig.add_subplot(234, projection='polar',
facecolor='#CCFFCC')
ax4.plot(phi, 0.2*phi, 'g') #(-1.5*phi, r, 'g')
ax4.grid(True)

ax5 = fig.add_subplot(235, facecolor='#FFFFCC')
ax5.plot(phi, phi*0.5, 'orange')
ax5.grid(True)

ax6 = fig.add_subplot(236, facecolor='#FFCCCC')
ax6.plot(np.sin(0.1*x), x, 'r')
ax6.grid(True)
###plt.legend(loc='lower left')
plt.show()
```



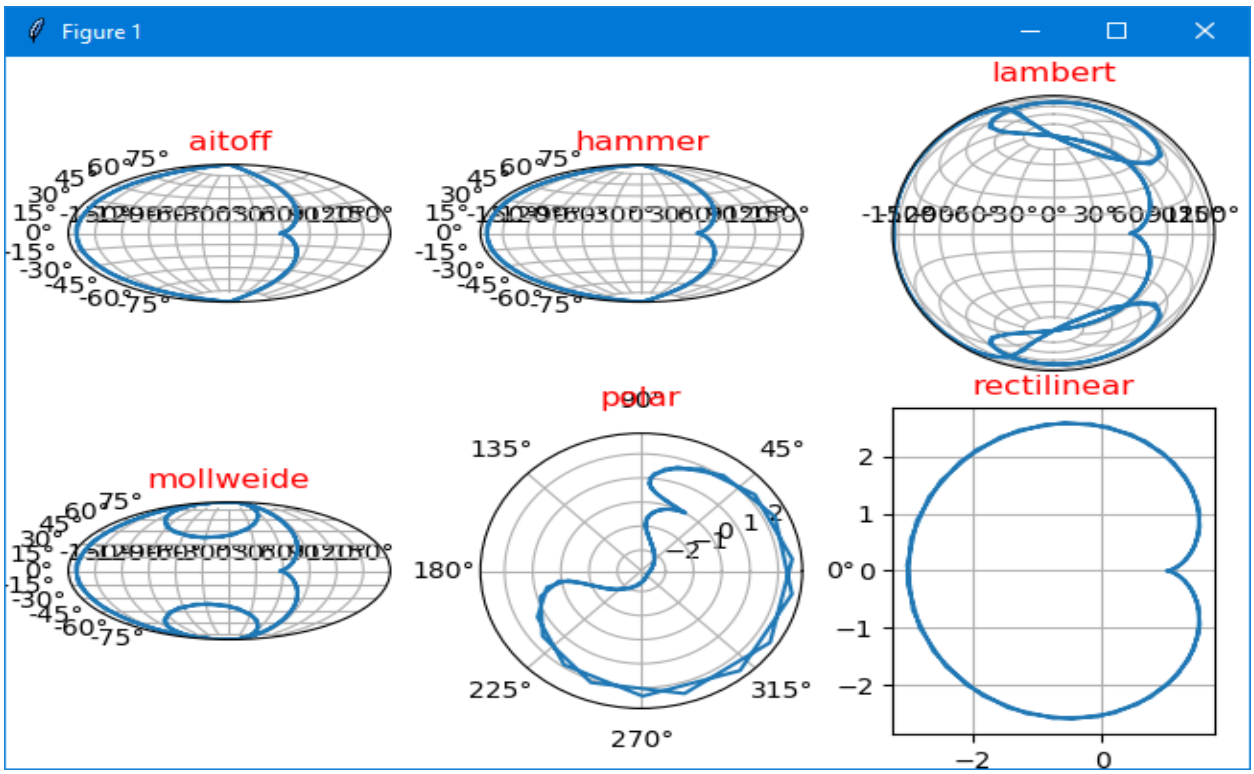
matplotlib дозволяє малювати у полярної системі координат, а й у інших. За це відповідає параметр projection, який у разі рівності значенню 'polar' аналогічний значенню параметра polar=True.

У matplotlib підтримуються такі проекції:

'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear'.

Найчастіше використовуються два останні варіанти, решта є екзотичними для рутинних завдань у науковій графіці.

```
import matplotlib.pyplot as plt
import numpy as np
a = 1.
x = np.arange(-2*np.pi, 2*np.pi, 0.2)
y = np.sin(x) * np.cos(x)
# Рівняння кардіойди
xz = a * (2 * np.cos (x) - np.cos (2 * x))
yz = a*(2*np.sin(x) - np.sin(2*x))
label = ['aitoff', 'hammer', 'lambert', 'mollweide',
'polar', 'rectilinear']
fig = plt.figure(figsize=(10,8))

for i in range(len(label)):
    ax = fig.add_subplot(231+i, projection=label[i])
    ax.plot(xz, yz)
    ax.set_title(label[i], color='r')
    ax.grid(True)
plt.tight_layout()
plt.show()
```



Полярна система координат має спеціальні функції, які дозволяють налаштовувати зовнішній вигляд малюнка.

Для радіусу R:

1. `ax.set_rlabel_position(phi)`- переміщає вісь ординат (радіус) по колу на кут phi (у ГРАДУСАХ) від положення нуля;
2. `ax.set_rmax(R)`- дозволяє обмежити область зміни радіуса R малюнку;
3. `ax.set_rmin(R)`- дозволяє обмежити область зміни радіуса R малюнку;
4. `ax.set_rlim()`- дозволяє обмежити область зміни радіуса R малюнку;
5. `ax.set_rscale()`- дозволяє зробити шкалу радіусів логарифмічної;
6. `ax.set_rgrid()`- дозволяє налаштувати для осі радіусу допоміжну сітку, положення поділів, формат підписів до них тощо.

Для кута phi (в matplotlib він називається theta):

1. `ax.set_theta_zero_location(loc)`- Переміщає положення нуля на певне положення. Loc приймає значення 'N', 'NW', 'W', 'SW', 'S', 'SE', 'E' або 'NE'. За умовчанням положення нуля знаходиться у положенні "схід" або "3 години";
2. `ax.set_theta_offset(phi)`- Переміщає положення нуля на кут phi (у радіанах). За умовчанням положення нуля знаходиться у положенні "схід" або "3 години";

3. `ax.set_theta_direction(loc)`- Визначає напрямок обходу. Loc може бути -1 або за годинниковою стрілкою і 1 або проти годинникової стрілки;
4. `ax.set_thetagrids()`- дозволяє налаштувати для осі кутів допоміжну сітку, положення поділів, формат підписів до них тощо.

П11. Графікрозсіювання

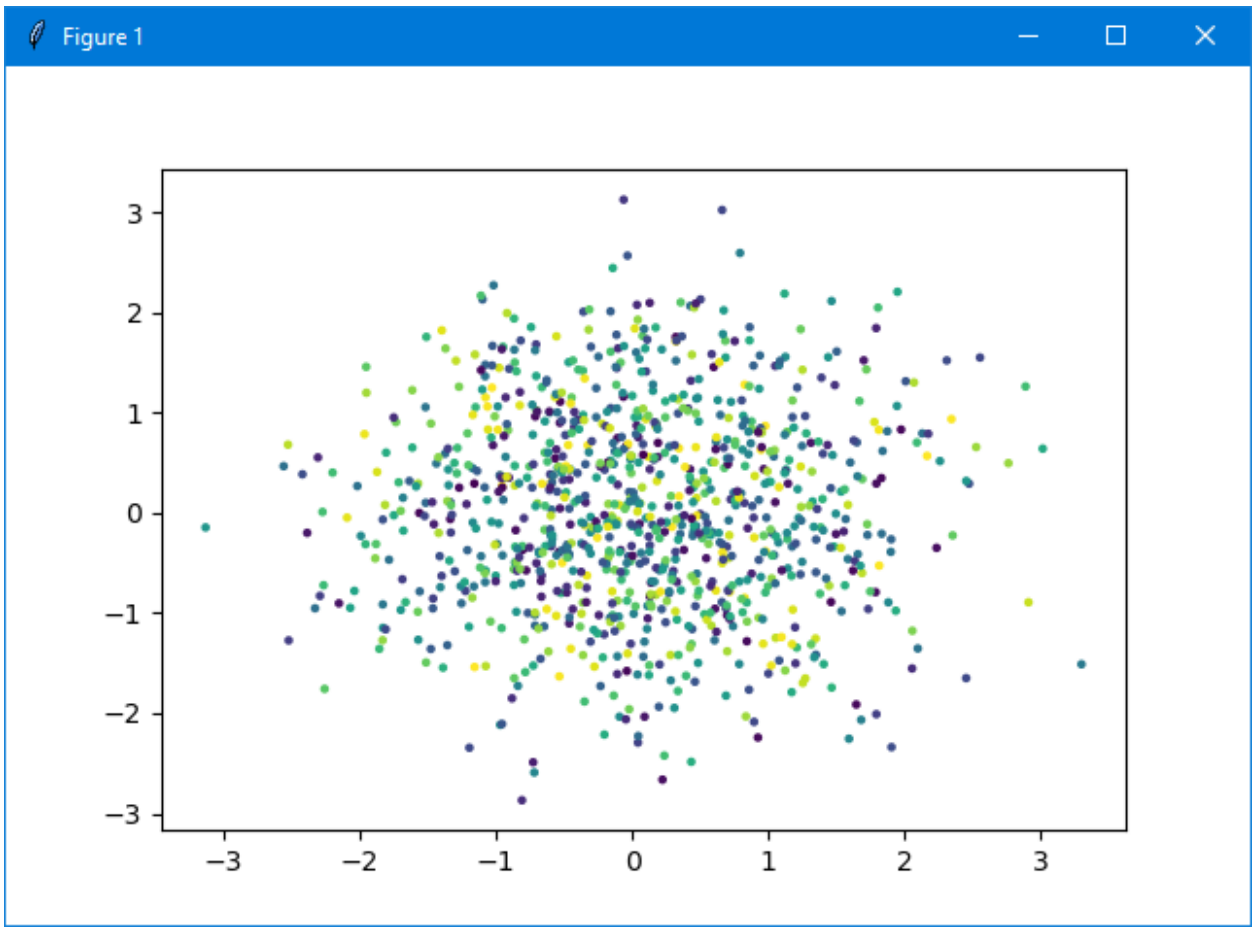
Такий тип графіків дозволяє зображати одночасно дві множини даних, які не утворюють кривої, а саме двовимірне безліч точок. Кожна точка має дві координати. Графік розсіювання часто використовується визначення зв'язку між двома величинами і дозволяє визначити більш точні межі вимірів.

У модулі matplotlib.pyplot є своя функція, для графіка розсіювання (scatter plot) це функція scatter().

Вона приймає дві послідовності та зображує їх на площині у вигляді маркерів, за умовчанням вони круглі та сині. Але природно, з ними можна попрацювати за допомогою keywords тієї ж функції:

- **s**задає розмір маркерів і може бути одним числом для всіх, так і представляти масив значень
- **c**задає колір маркерів, або один для всіх, або безліч
- **marker**визначає тип маркеру.

```
importmatplotlib.pyplot as plt
importnumpy as np

x = np.random.randn(1000)
y = np.random.randn(1000)

size = 5
colors = np.random.rand(1000)

plt.scatter(x, y, s=size, c=colors)
plt.show()
```



П12. Налаштування в стилі LATEX

Ось приклад налаштування майже всього, що можна налаштувати.

Можна задати послідовність засічок на осі x (і y) та підписи до них (у них, як і в інших текстах, можна використовувати LATEX-івські позначення).

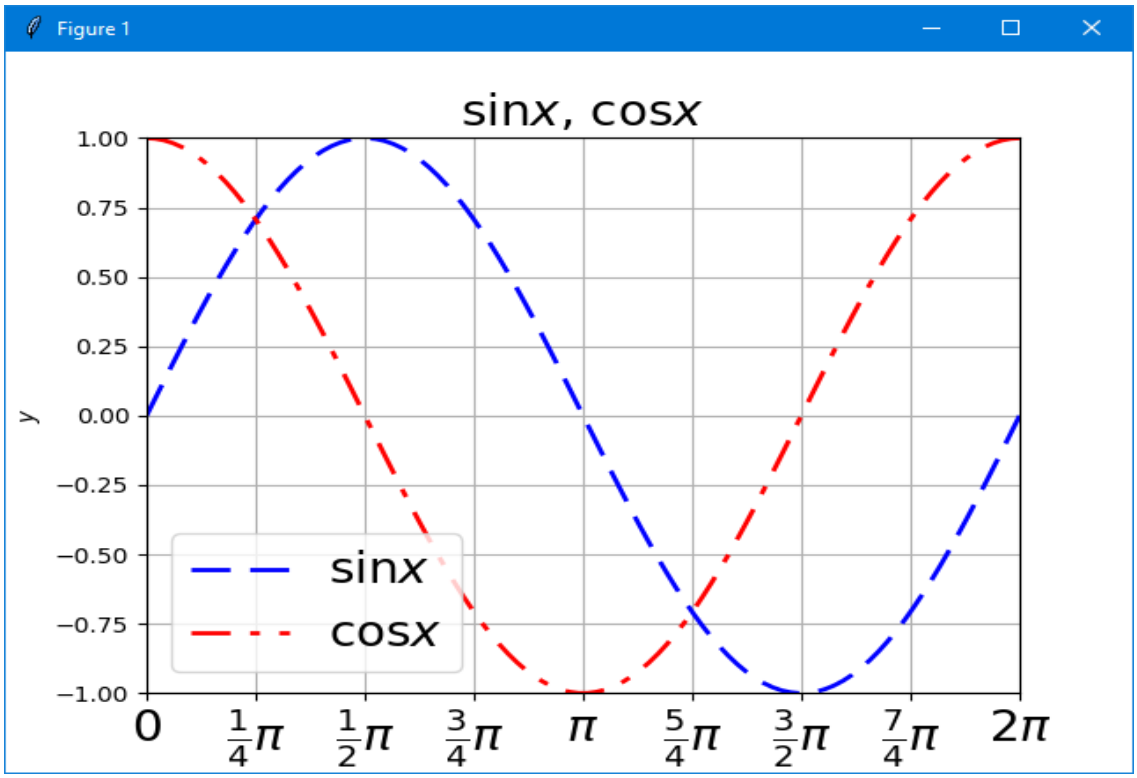
Задати підписи осей x та y та заголовок графіка.

У всіх текстових елементах можна встановити розмір шрифту. Можна задати товщину ліній і штрихи (так, на графіці косинуса малюється штрих довжини 8, потім ділянка довжини 4 не малюється, потім ділянка довжини 2 малюється, потім ділянка довжини 4 знову не малюється, і так за циклом; оскільки товщина лінії дорівнює 2, ці короткі штрихи довжини

Можна поставити підписи до кривих (legend); де розмістити ці підписи також можна регулювати

```
importmatplotlib.pyplot as plt
importnumpy as np
plt.axis([0,2*np.pi,-1,1])

plt.xticks(np.linspace(0,2*np.pi,9),
       ('0',r'$\frac{1}{4}\pi$',r'$\frac{1}{2}\pi$',

r'$\frac{3}{4}\pi$',r'$\pi$',r'$\frac{5}{4}\pi$',
```

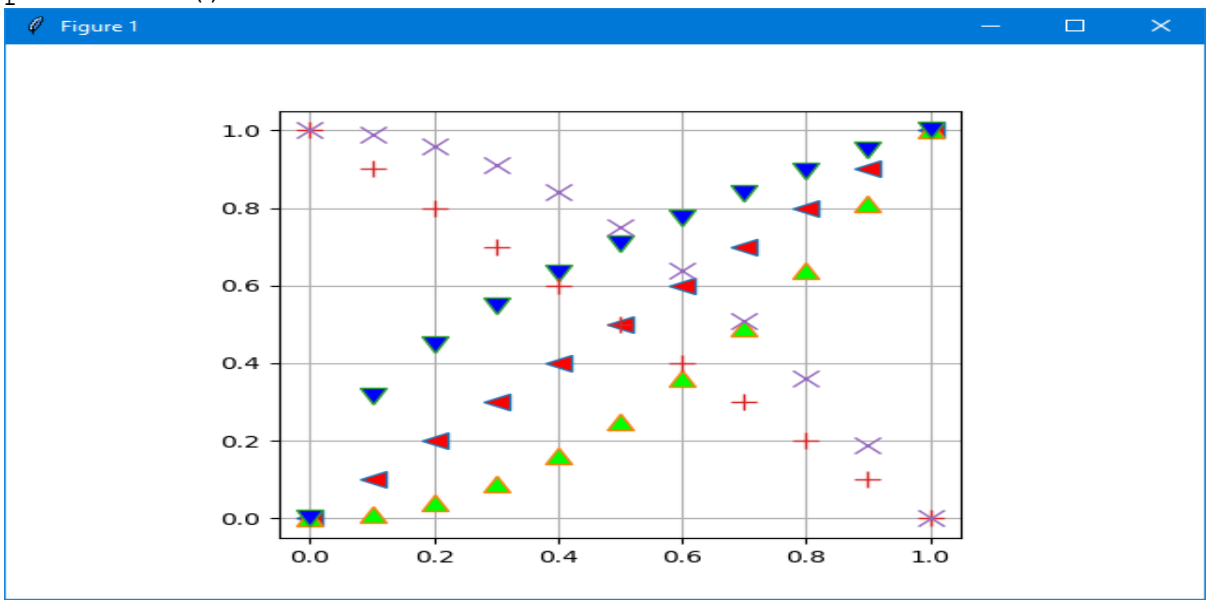
```
r'$\frac{3}{2}\pi$',r'$\frac{7}{4}\pi$',r'$2\pi$'),
      fontsize=20)

plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'$\sin x$, $\cos x$',fontsize=20)

x=np.linspace(0,2*np.pi,100)
plt.plot(x,np.sin(x),linewidth=2,
color='b',dashes=[8,4], label=r'$\sin x$')

plt.plot(x,np.cos(x),linewidth=2,
color='r',dashes=[8,4,2,4], label=r'$\cos x$')

plt.legend(fontsize=20)
plt.grid(True)
візуалізуємо графіки
plt.show()
```



П13. Модифіковані маркери

Якщо linestyle='' то точки не з'єднуються лініями. Самі крапки малюються маркерами різних типів. Тип визначається рядком з одного символу, який чимось нагадує потрібний маркер. На додаток до стандартних маркерів, можна визначити саморобні.

```
importmatplotlib.pyplot as plt
importnumpy as np
x=np.linspace(0,1,11)
# Встановимо параметр - aspect (ratio)
axx=plt.subplot(111)
axx.set_aspect(1)
plt.axis([-0.05,1.05,-0.05,1.05])
plt.plot(x,x,linestyle='',marker='<',markersize=10,
        markerfacecolor='#FF0000')
plt.plot(x,x**2,linestyle='',marker='^',markersize=10,
        markerfacecolor='#00FF00')
plt.plot(x,x**(1/2),linestyle='',marker='v',markersize=
10,
        markerfacecolor='#0000FF')
plt.plot(x,1-x,linestyle='',marker='+',markersize=10,
        markerfacecolor='#0F0F00')
plt.plot(x,1-
x**2,linestyle='',marker='x',markersize=10,
     markerfacecolor='#0F000F')
plt.grid(True)
візуалізуємо графіки
plt.show()
```

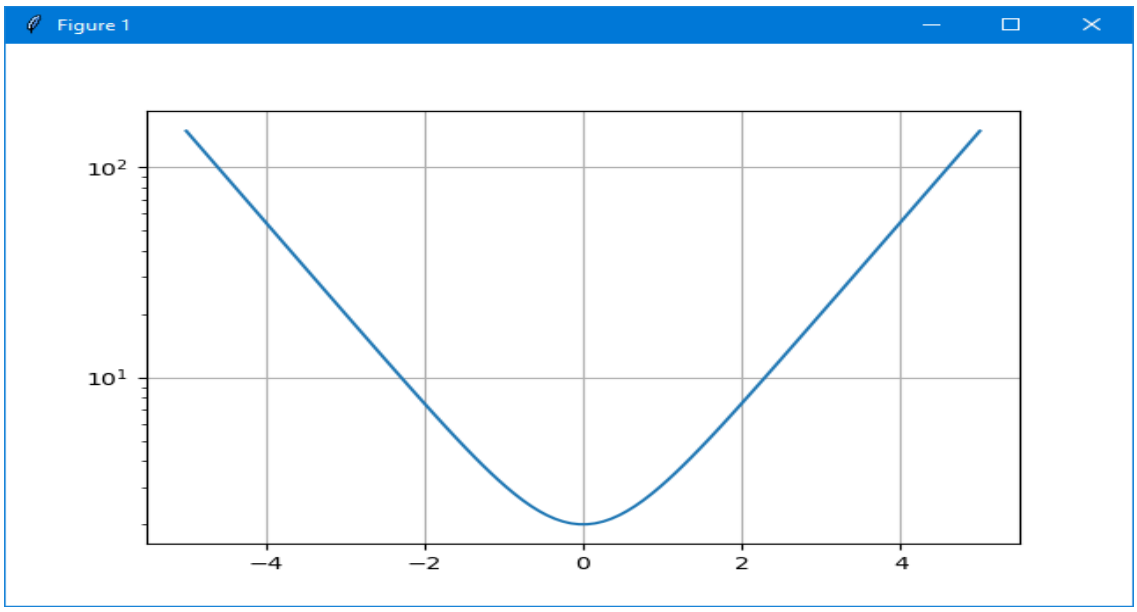


П14. Логарифмічний масштаб

Якщо у змінюється на багато порядків, то зручно використовувати логарифмічний масштаб по у

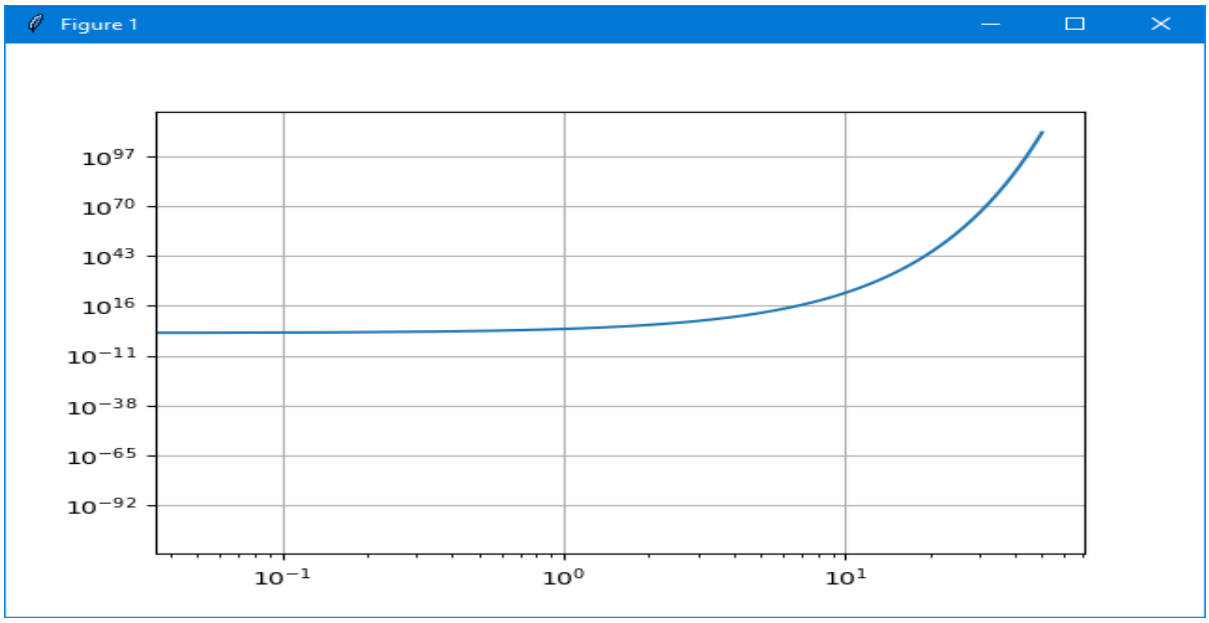
```
importmatplotlib.pyplot as plt
importnumpy as np
x=np.linspace(-5,5,100)
plt.yscale('log')
plt.plot(x,np.exp(x)+np.exp(-x))
```

```
plt.grid(True)
візуалізуємо графіки
plt.show()
```



Можна встановити логарифмічний масштаб по обох осях.

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-50,50,1000)
plt.xscale('log')
plt.yscale('log')
plt.plot(x,np.exp(5*x+3))
plt.grid(True)
візуалізуємо графіки
plt.show()
```



П13. Екпериментальні дані

Допустимо, є теоретична крива (резонанс без фону).

```
xt=np.linspace(-4,4,101)
yt=1/(xt**2+1)
```

Оскільки реальних експериментальних даних немає, ми їх згенеруємо. Нехай вони узгоджуються з теорією і всі статистичні помилки дорівнюють 0.1.

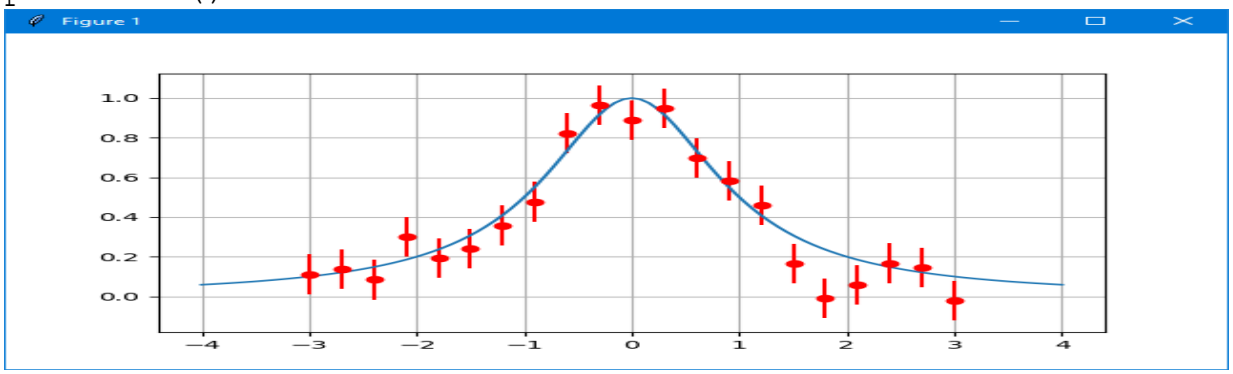
```
xe=np.linspace(-3,3,21)
yerr=0.1*np.ones(21)
ye=1/(xe**2+1)+yerr*np.random.normal(size=21)
```

Експериментальні точки з вусами і теоретична крива однією графіку.

```
plt.plot(xt,yt)
plt.errorbar(xe,ye,fmt='ro',yerr=yerr)
```

Весь скрипт:

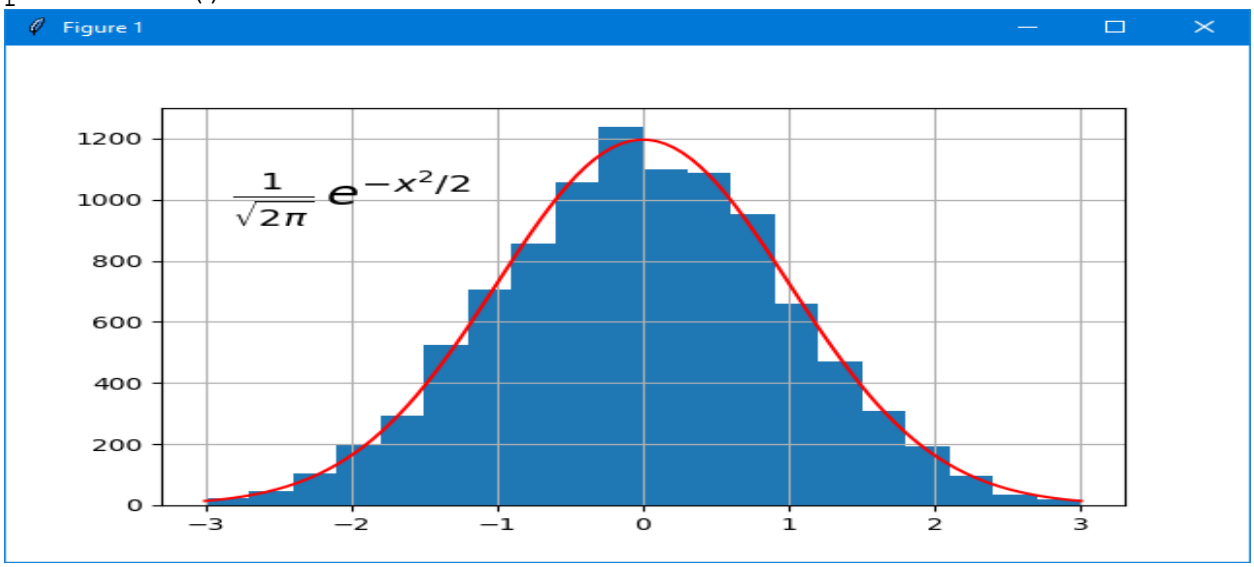
```
importmath
importnumpy as np
importmatplotlib.pyplot as plt
xt=np.linspace(-4,4,101)
yt=1/(xt**2+1)
xe=np.linspace(-3,3,21)
yerr=0.1*np.ones(21)
ye=1/(xe**2+1)+yerr*np.random.normal(size=21)
plt.plot(xt,yt)
plt.errorbar(xe,ye,fmt='ro',yerr=yerr)
plt.grid(True)
візуалізуємо графіки
plt.show()
```



П15. Гістограма

Згенеруємо N випадкових чисел з нормальним (гаусовим) розподілом (середнє 0, середньоквадратичне відхилення 1), і розкидаємо їх по 20 бін від -3 до 3 (точки за межами цього інтервалу відкидаються). Для порівняння, разом із гістограмою намалюємо Гауссову криву у тому самому масштабі. І навіть напишемо формулу Гауса.

```
importnumpy as np
importmatplotlib.pyplot as plt
importmatplotlib.mlab as ml
N=10000
r=np.random.normal(size=N)
n, bins, patches = plt.hist (r, range = (-3,3), bins =
20)
x=np.linspace(-3,3,100)
plt.plot(x,N/np.sqrt(2*np.pi)*0.3* np.exp(-
0.5*x**2),'r')

plt.text(-2,1000,r'$\frac{1}{\sqrt{2\pi}} \,e^{-
x^2/2}$',

     fontsize=20,horizontalalignment='центр',
     verticalalignment='center')
plt.grid(True)
візуалізуємо графіки
plt.show()
```

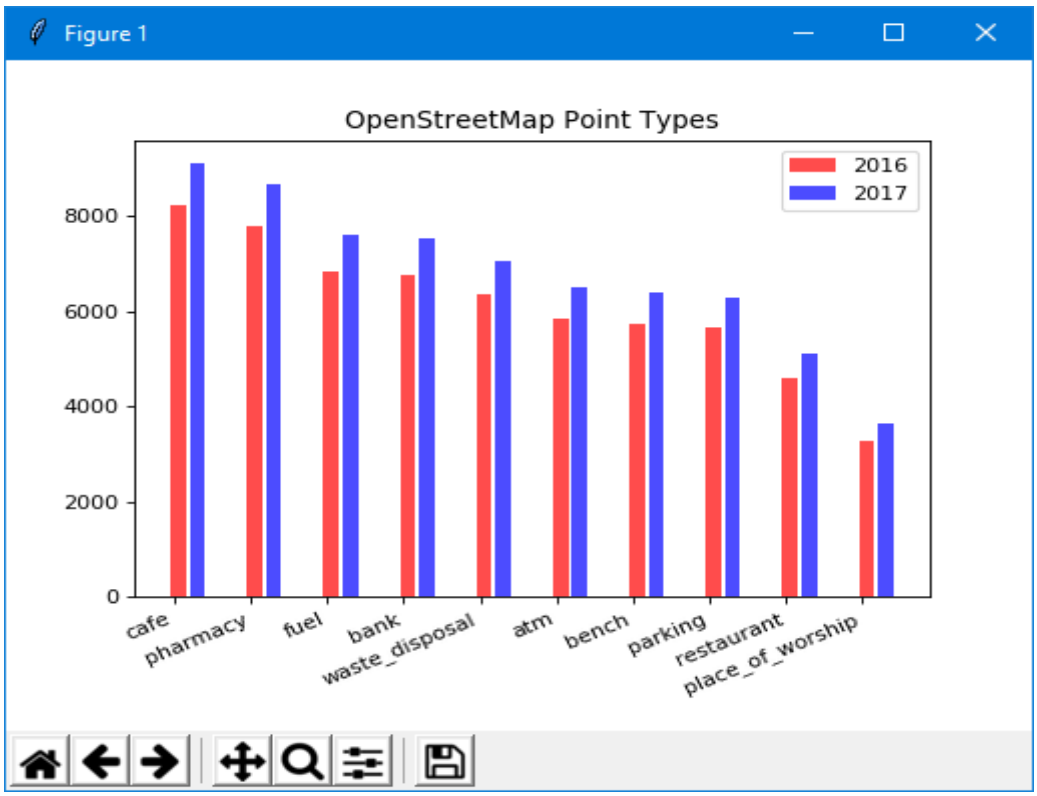


Розглянемо кілька прикладів (https://eax.me/python-matplotlib/) побудови спеціальних діаграм.

Як приклад побудуємо діаграму, що відображає, скільки точок на карті якого типу (заправка, кафе тощо) ставляться. Щоб було трохи цікавіше, вдамо, що торік точок кожного виду було на 10% менше, і спробуємо відобразити цю зміну:

```
importmatplotlib as mpl
importmatplotlib.pyplot as plt
importmatplotlib.dates як mdates
importdatetime as dt
importcsv
```

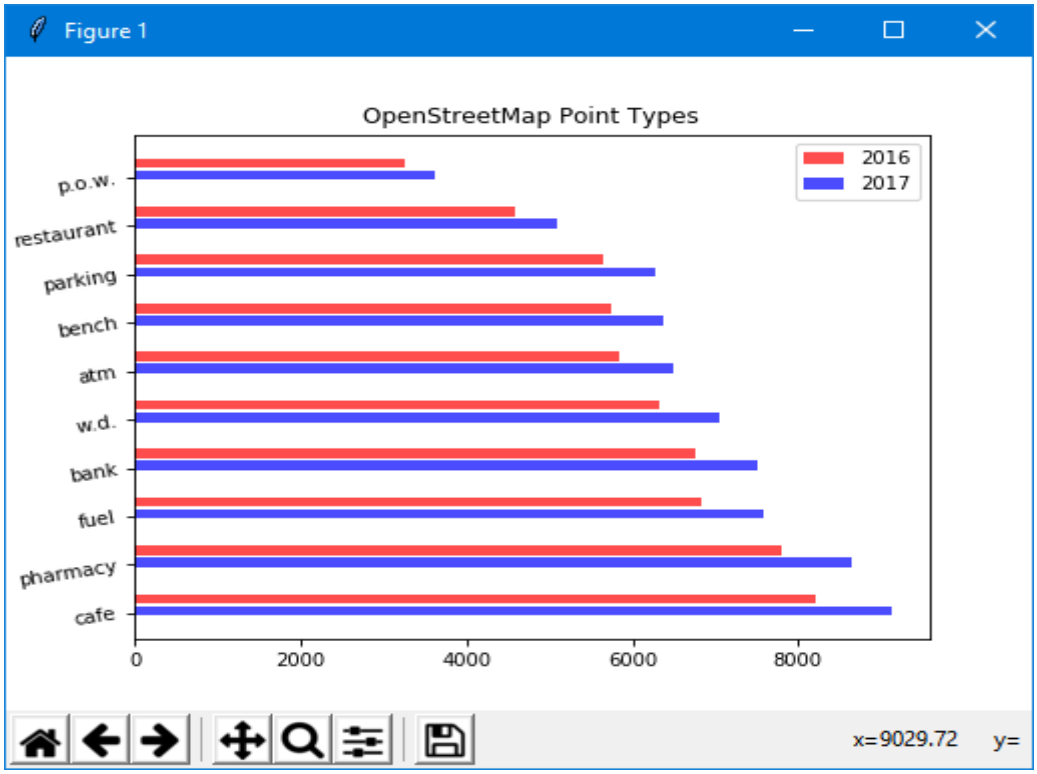
```
data_names = ['cafe', 'pharmacy', 'fuel', 'bank',
'waste_disposal',
              'atm', 'bench', 'parking', 'restaurant',
              'place_of_worship']
data_values = [9124, 8652, 7592, 7515, 7041, 6487,
6374, 6277,
               5092, 3629]
dpi = 80
fig = plt.figure(dpi = dpi, figsize = (512 / dpi, 384 /
dpi) )
mpl.rcParams.update({'font.size': 10})
plt.title('OpenStreetMap Point Types')
#ax = plt.axes()
#ax.yaxis.grid(True, zorder = 1)
xs = range(len(data_names))
plt.bar([x + 0.05 for x in xs], [d * 0.9 for d in
                                        data_values],
        width = 0.2, color = 'red', alpha = 0.7, label
= '2016',
        zorder = 2)
plt.bar([x + 0.3 for x in xs], data_values,
        width = 0.2, color = 'blue', alpha = 0.7, label
= '2017',
        zorder = 2)
plt.xticks(xs, data_names)
fig.autofmt_xdate(rotation = 25)
plt.legend(loc='upper right')
#fig.savefig('bars.png')
plt.show()
```



Ті ж дані можна відобразити, розташувавши стовпчики горизонтально:

```
importmatplotlib as mpl
importmatplotlib.pyplot as plt
importmatplotlib.dates як mdates
importdatetime as dt
importcsv

data_names = ['cafe', 'farmaceu', 'fuel', 'bank', 'wd',
'atm', 'bench', 'parking', 'restaurant', 'pow']
data_values = [9124, 8652, 7592, 7515, 7041, 6487,
6374, 6277, 5092, 3629]

dpi = 80
fig = plt.figure(dpi = dpi, figsize =
                                (512 / dpi, 384 / dpi))
mpl.rcParams.update({'font.size': 9})
plt.title('OpenStreetMap Point Types')
#ax = plt.axes()
#ax.xaxis.grid(True, zorder = 1)
xs = range(len(data_names))
plt.barh([x + 0.3 for x in xs], [ d * 0.9 for d in
data_values],
         height = 0.2, color = 'red', alpha = 0.7,
label = '2016',
         zorder = 2)
plt.barh([x + 0.05 for x in xs], data_values,
```

```
         height = 0.2, color = 'blue', alpha = 0.7,
label = '2017',
         zorder = 2)
plt.yticks(xs, data_names, rotation = 10)

plt.legend(loc='upper right')
plt.show()
```



П16. Кругова діаграма

Наприклад візуалізуємо розподіл кафе по різних містах (приклад взятий з https://eax.me/python-matplotlib/):

```
importmatplotlib as mpl
importmatplotlib.pyplot as plt
importmatplotlib.dates як mdates
importdatetime as dt
importcsv

data_names = ['Москва', 'Санкт-Петербург', 'Сочі',
'Архангельськ',
              'Володимир', 'Краснодар', 'Курськ',
'Вороніж',
              'Ставрополь', 'Мурманськ']
data_values = [1076, 979, 222, 189, 137, 134, 124, 124,
91, 79]
```

```
dpi = 80
fig = plt.figure(dpi = dpi, figsize = (512 / dpi, 384 /
dpi) )
mpl.rcParams.update({'font.size': 9})

plt.title('Розподіл кафе по містах Росії (%)')

xs = range(len(data_names))

plt.pie(
    data_values, autopct='%.1f', radius = 1.1,
    explode = [0.15] + [0 for _ in
range(len(data_names) - 1)] )
plt.legend(
    bbox_to_anchor = (-0.16, 0.45, 0.25, 0.25),
    loc = 'low left', labels = data_names )
plt.show()
```

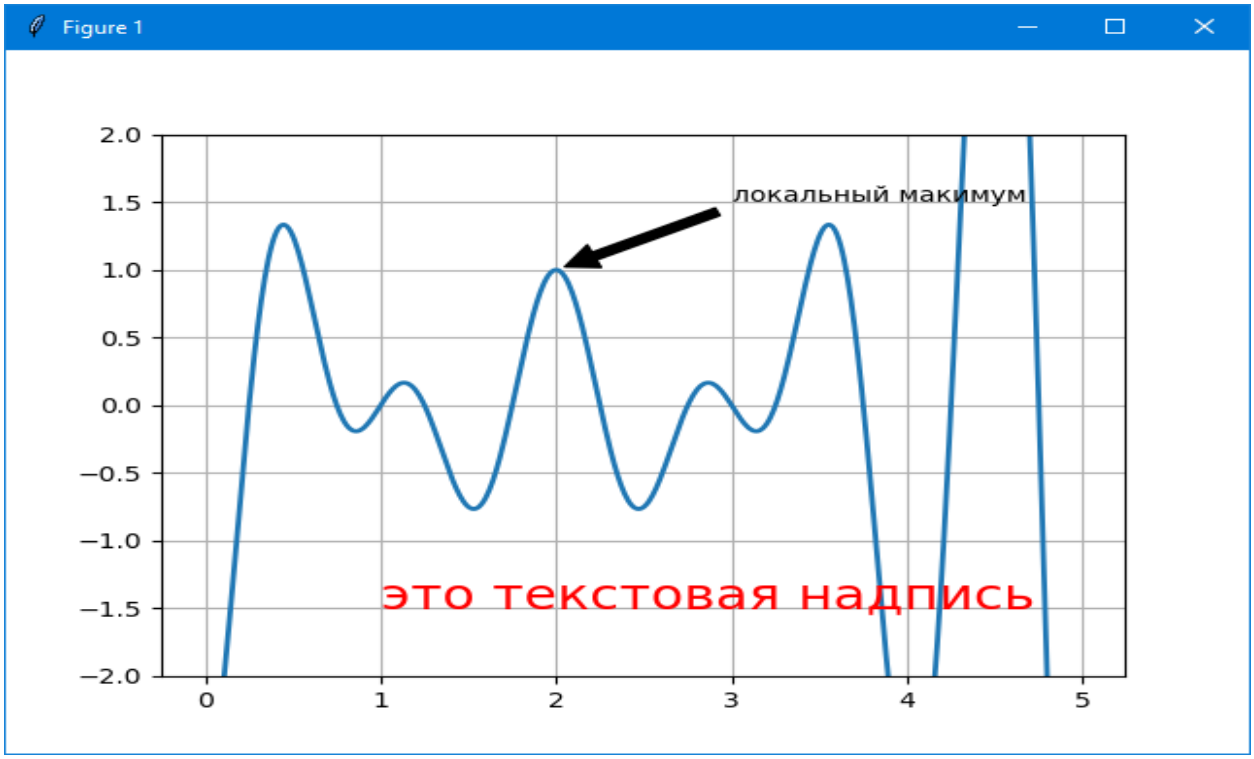
Велику кількість прикладів можна знайти на https://matplotlib.org/gallery.html

П17. Текст та написи

Текст та додаткові написи (аннатації) розміщуються на діаграмі:

```
importnumpy as np
importmatplotlib as mpl
```

```
importmatplotlib.pyplot as plt
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = (1-(t-2)*(t-2)) * np.cos(2*np.pi*t)
line, = plt.plot (t, s, lw = 2)
plt.text(1,-1.5,'це текстовий напис', size=20,
color='r')
plt.grid(True)
plt.annotate('локальний макимум', xy=(2, 1), xytext=(3,
1.5), arrowprops= dict(facecolor='black',
shrink=0.05), )
plt.ylim(-2,2)
plt.show()
```

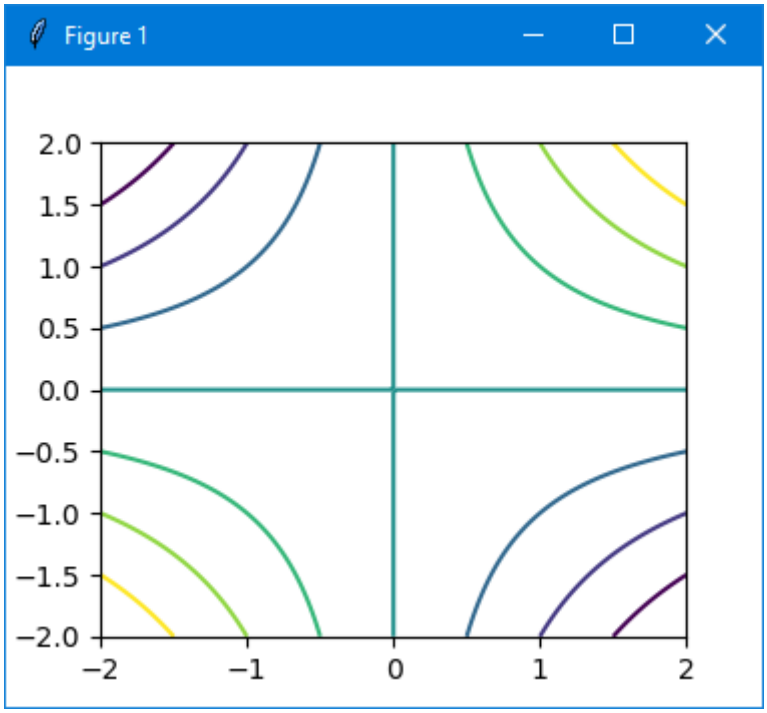


П18. Контурні графіки

Нехай потрібно вивчити поверхню z = xy. Побудуємо її «горизонталі» (взяти з http://www.inp.nsk.su/~grozin/python/python6.html):

```
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
x=np.linspace(-1,1,50)
y=x
z = np.outer (x, y)
plt.contour(x,y,z)
```

```
plt.axes().set_aspect(1)
plt.show()
```

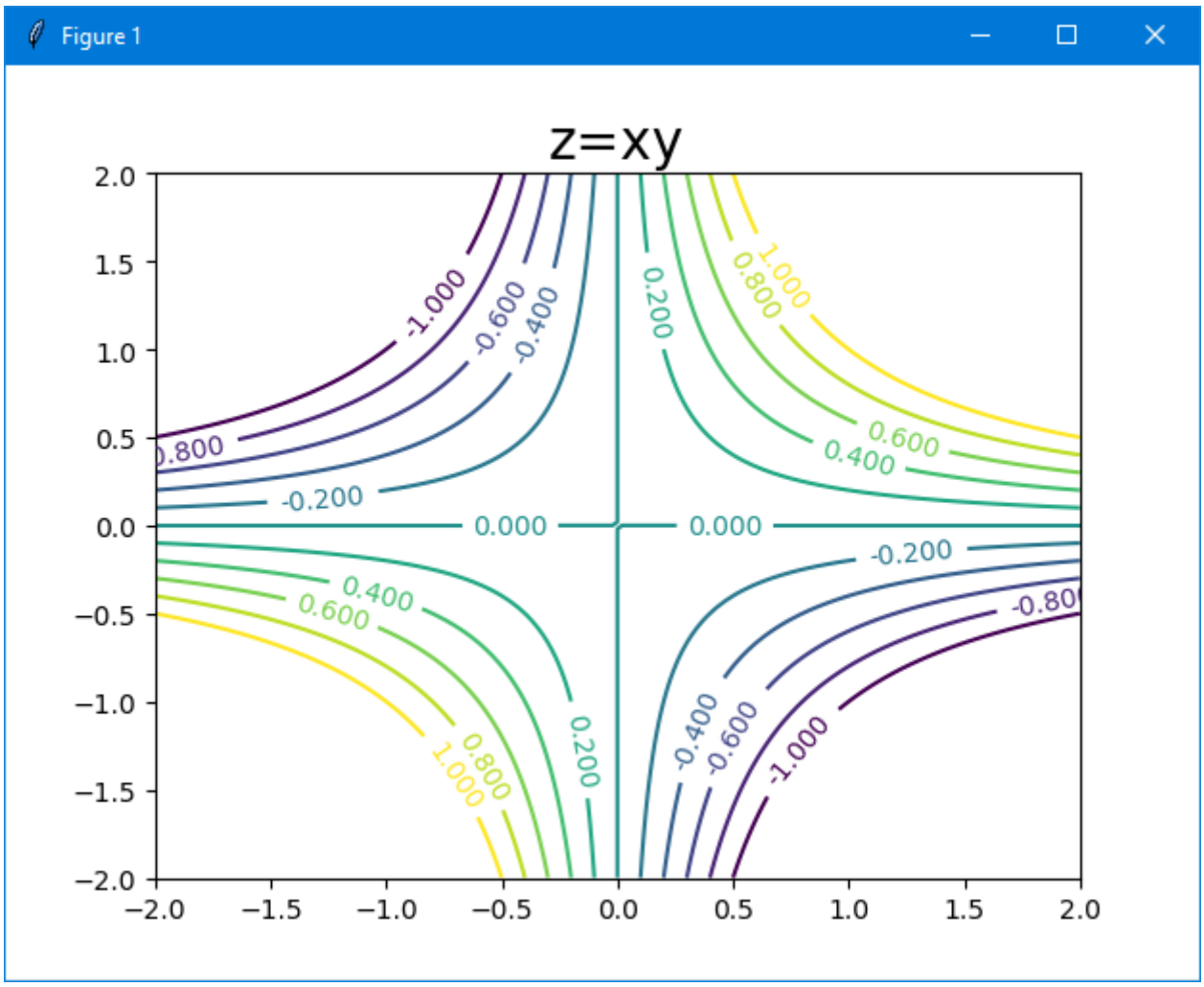


Роль функції np.outer(x,y) пакета numpy очевидна з прикладу:

```
>>> x=np.array([1,2,3])
>>> y=np.array([10,20,30])
>>> x*y
array([10, 40, 90])
>>> np.outer(x,y)
array([[10, 20, 30],
       [20, 40, 60],
       [30, 60, 90]])
```

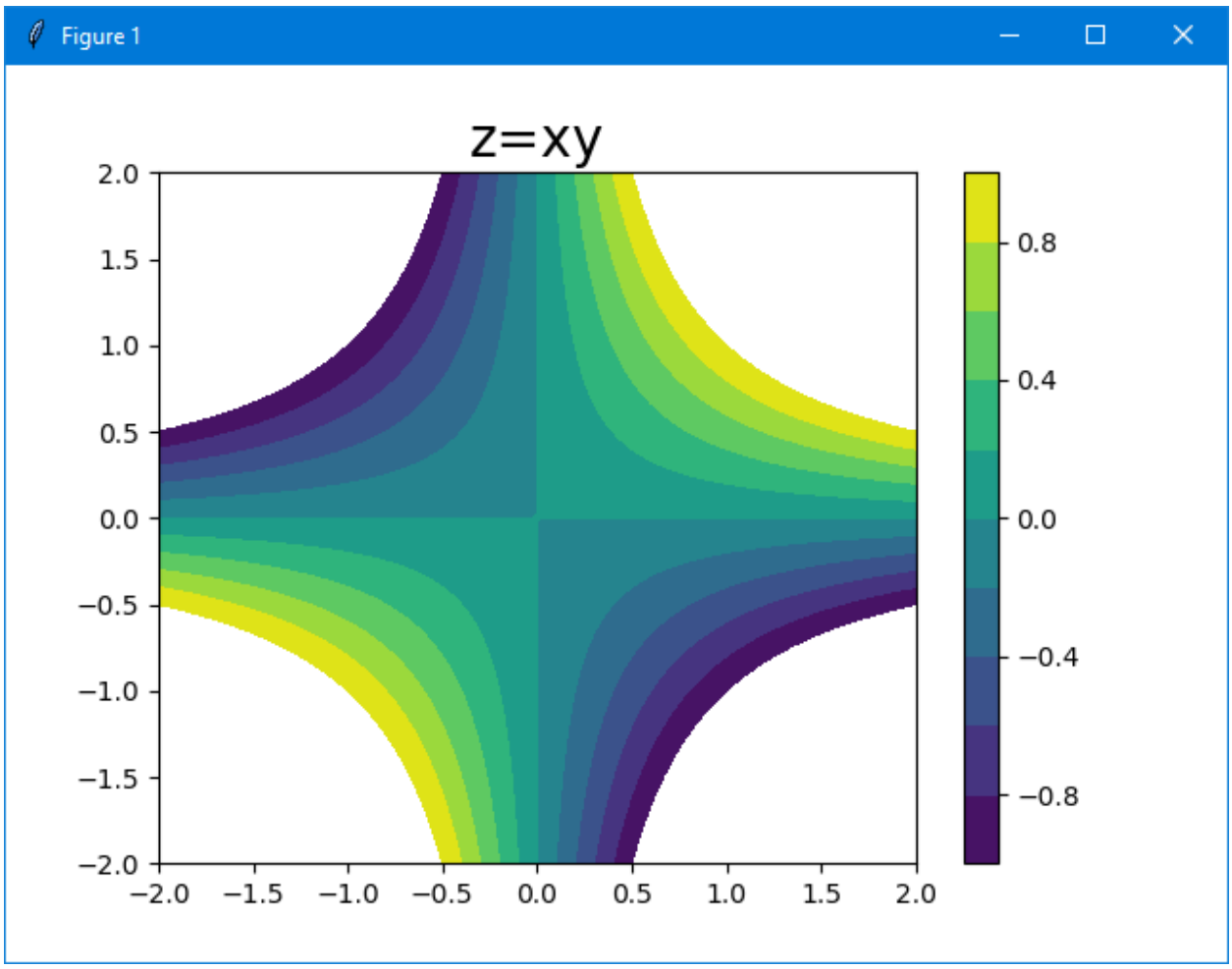
Горизонталі можна розфарбувати і підписати, а також збільшити їх кількість:

```
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y=x
z = np.outer (x, y)
plt.title('z=xy',fontsize=20)
curves=plt.contour(x,y,z,np.linspace(-1,1,11))
plt.clabel(curves)
plt.show()
```



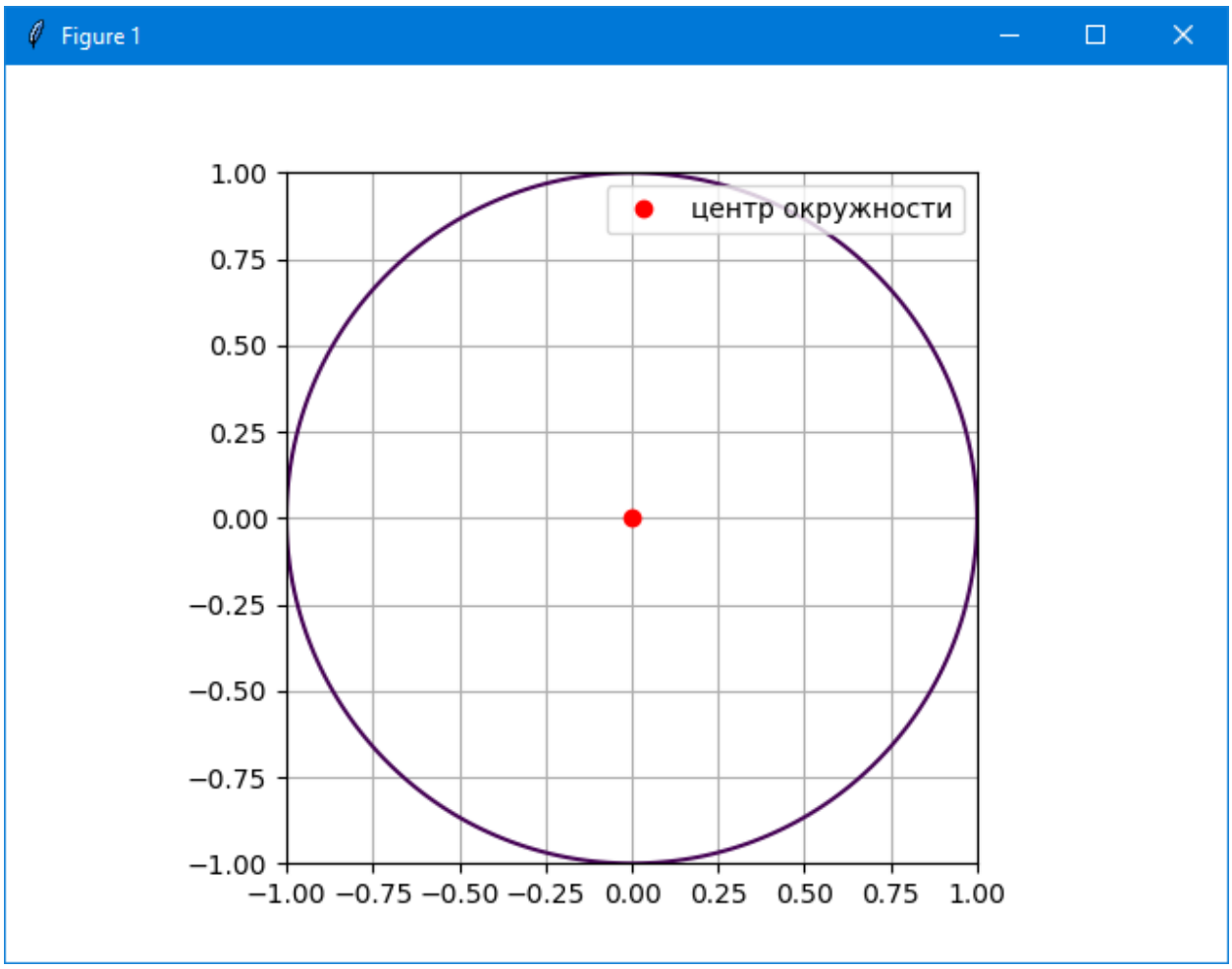
Висоту (значення Z) можна встановити кольором, як на фізичних географічних картах. colorbar показує відповідність кольорів та значень z.

```
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y=x
z = np.outer (x, y)
plt.title('z=xy',fontsize=20)
plt.contourf(x,y,z,np.linspace(-1,1,11))
plt.colorbar()
#plt.axes().set_aspect(1)
plt.show()
```



Так можна «малювати» графіки функцій на площині, задані «неявно» - F(x, y)=0:

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

x = np.linspace(-1.0, 1.0, 100)
y = np.linspace(-1.0, 1.0, 100)
X, Y = np.meshgrid(x,y)
F = X**2 + Y**2 - 1  #0.6
plt.contour(X,Y,F,[0])
plt.plot([0],[0],'ro', label="центр кола")
plt.gca().set_aspect('equal') #, щоб малюнок виглядав кругом

# Включаємо сітку
plt.grid(True)
plt.legend(loc='best')
plt.show()
```

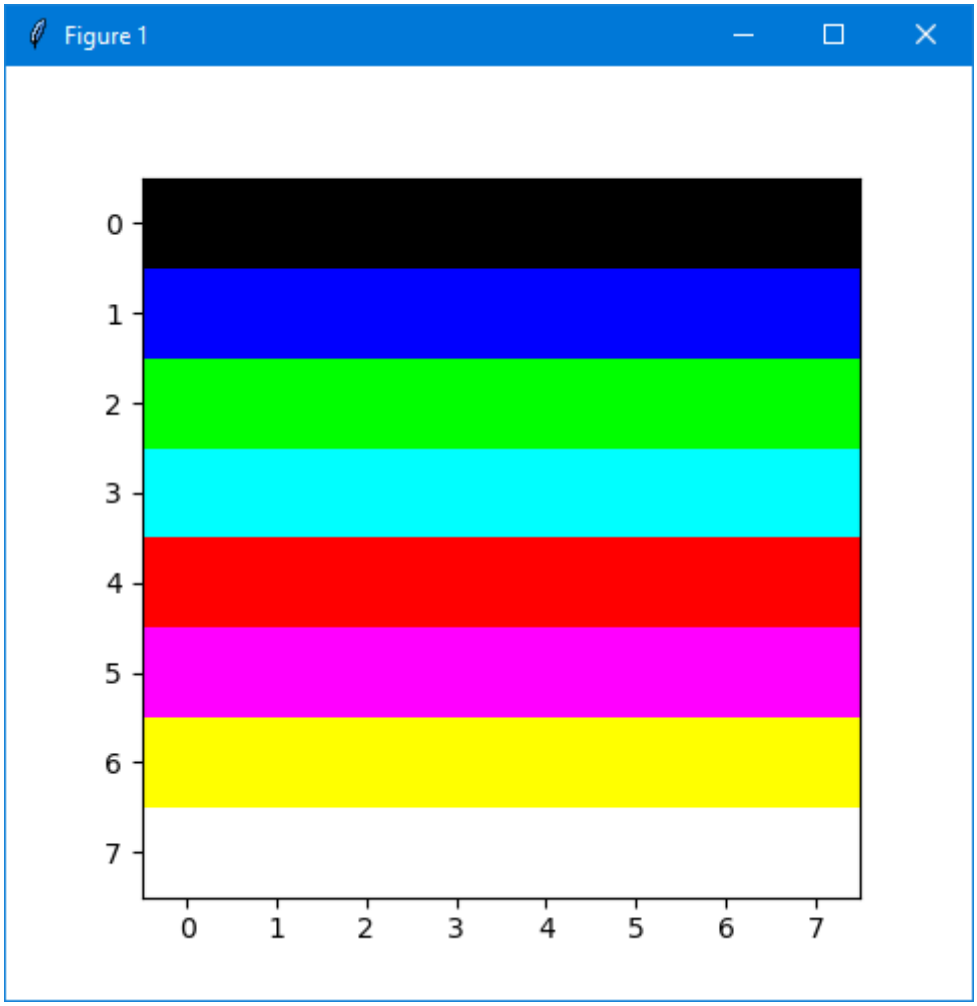


П19. Images (піксельні картинки)

Картинка задається масивом z: z[i,j] - це колір пікселя i,j, масив із 3 елементів (rgb, числа від 0 до 1). Для наочності у прикладі формуються «натуральні» кольори до списку col:

```
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt

n=8
u=np.linspace(0,1,n)
x,y=np.meshgrid(u,u)
z=np.zeros((n,n,3))

col=[]
forr in range(0,2):
    forg in range(0,2):
        forb in range(0,2):
            print(r, g, b)
            col.append([r, g, b])
#print (col)
```

```
for i in range(n):
    for j in range(n): # RGB
        z[i,j]=col[i]

plt.imshow(z)
plt.show()
```



П20. Тривимірний графік

Усі класи для роботи з тривимірними графіками знаходяться у пакеті mpl_toolkits.mplot3d, з якого потрібно їх імпортувати.

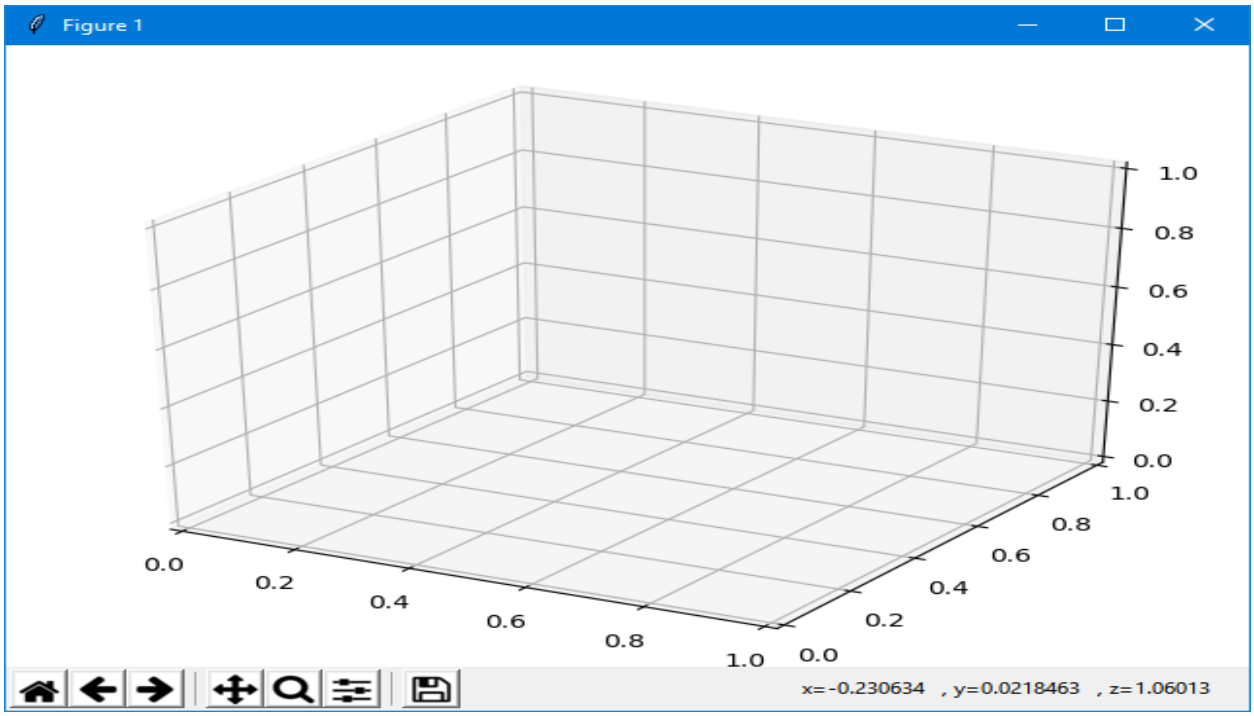
Для того щоб намалювати тривимірний графік, в першу чергу треба створити тривимірні осі.

Щоб створити їх, потрібно створити екземпляр класу mpl_toolkits.mplot3d.Axes3D. Його конструктор чекає, як мінімум, один параметр – екземпляр класу matplotlib.figure.Figure. Цей об'єкт створюється pylab.figure(). Конструктор класу matplotlib.figure.Figure має ще й інші необов'язкові параметри (за матеріалами сайту http://jenyay.net/Programming/Python3d).

Намалюємо порожні осі:

```
importnumpy as np
importpylab
frommpl_toolkits.mplot3d import Axes3D

fig = pylab.figure()
Axes3D(fig)
pylab.show()
```

В результаті побачимо наступне вікно:



Отримані осі можна обертати мишкою.

П21. Тривимірна лінія

Лінія в просторі задається параметрично: x = x (t), y = y (t), z = z (t). Наприклад так:

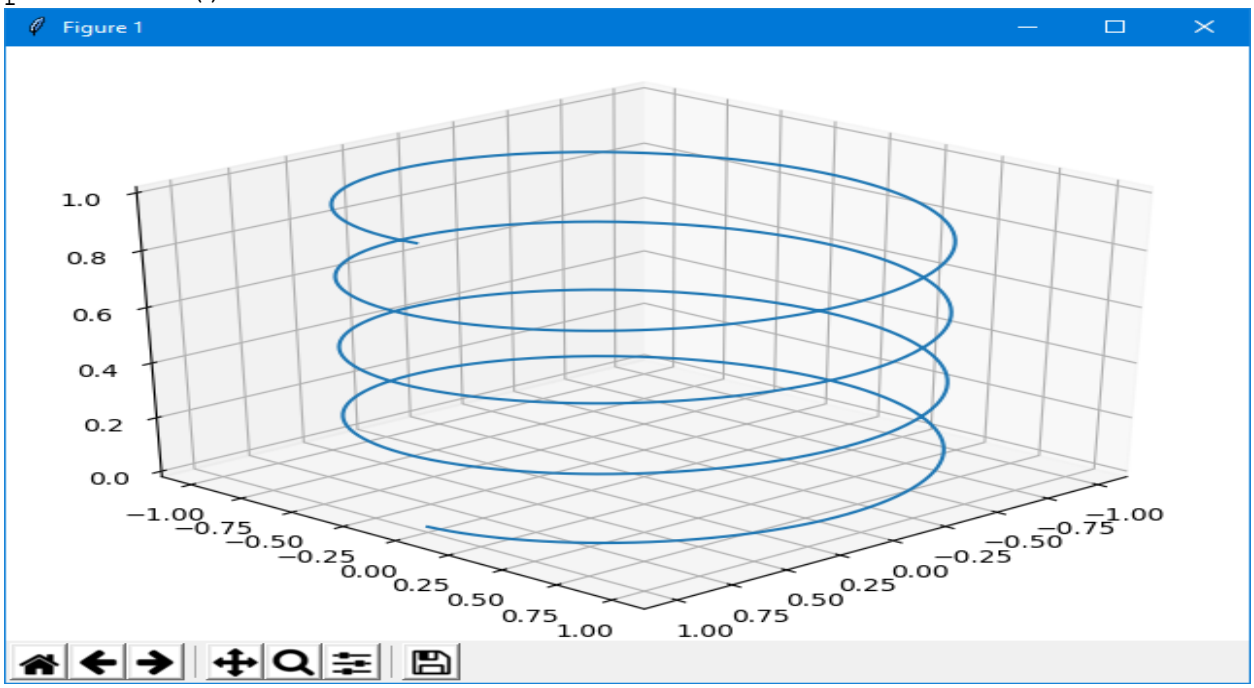
```
t=np.linspace(0, 4*np.pi,1000)
x=np.cos(2*t)
y=np.sin(2*t)
z = t / (4 * np.pi)
```

Для побудови та візуалізації використовується об'єкт класу Axes3D з пакету mpl_toolkits.mplot3d:

```
importpylab
importmpl_toolkits.mplot3d as A3D
```

figure() - це поточний малюнок, створюємо в ньому об'єкт ax, потім використовуємо його методи для візуалізації

```
importpylab
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
##from mpl_toolkits.mplot3d import Axes3D
importmpl_toolkits.mplot3d as A3D

t=np.linspace(0, 4*np.pi,1000)
x=np.cos(2*t)
y=np.sin(2*t)
z = t / (4 * np.pi)

#Тут потрібен об'єкт класу Axes3D з пакету
mpl_toolkits.mplot3d.
# figure() - це поточний малюнок, створюємо в ньому
об'єкт ax,
потім використовуємо його методи.
fig=pylab.figure()
ax=A3D.Axes3D(fig)
ax.elev,ax.azim=30,45 # задати, з якого боку дивимося.

ax.plot3D(x,y,z)
plt.show()
```



Побудуємо три прямі по осях координат:

```
від mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
```

```
import numpy as np

fig = plt.figure()
axes = axes3d.Axes3D(fig)

t = np.arange(-1, 1, 0.01)
x, y = np.meshgrid(t, t)

axes.plot3D(t, 0*t, 0*t, color='k' )
axes.plot3D(0*t, t, 0*t, color='k' )
axes.plot3D(0*t, 0*t, t, color='k' )

plt.show()
```

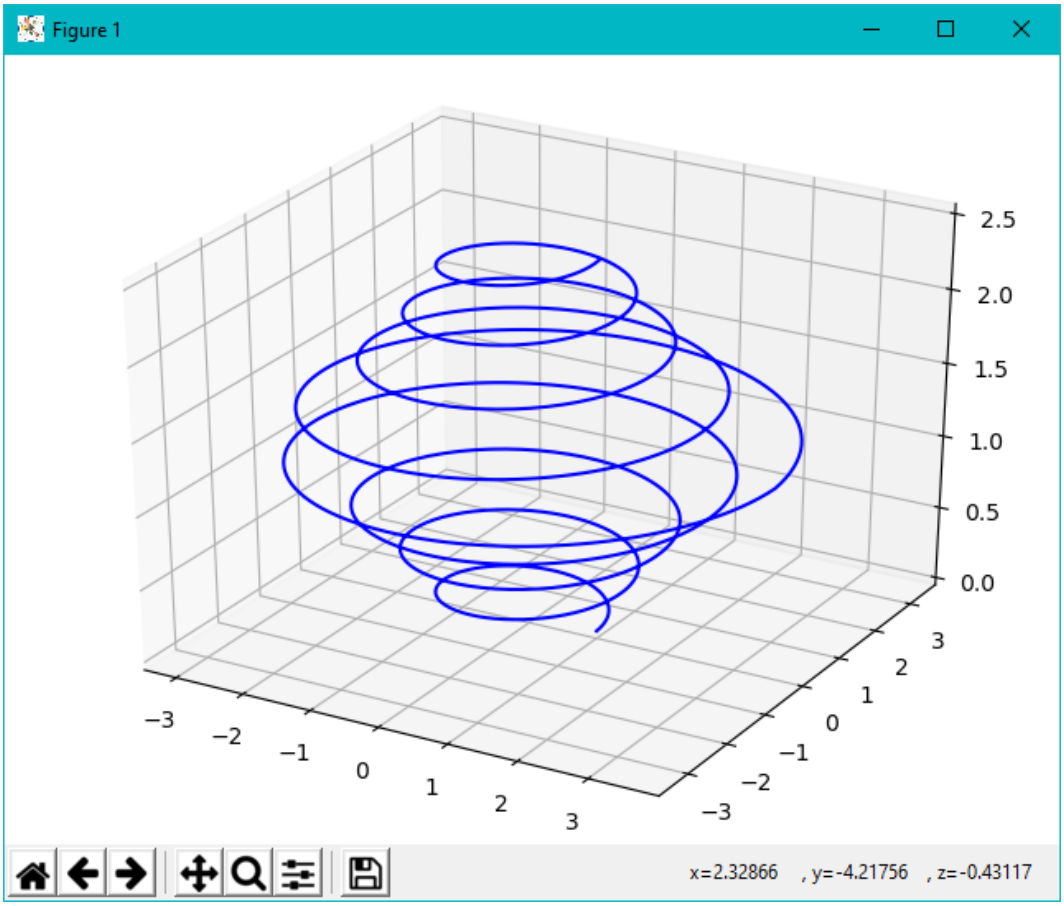
Побудуємо лінію, задану рівняннями:

$x(t) = \cos(2 \cdot t) \cdot e^{\frac{a \cdot t}{10}}$ $y(t) = \sin(2 \cdot t) \cdot e^{\frac{a \cdot t}{10}}$ $z(t) = \frac{t}{10}$

$t \in [0, 8 \cdot \pi]$ $a = \{-1, \text{при } t \in [0, 4 \cdot \pi)|$

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as A3D

tt1 = 0; tt2 = 4 * np.pi; tt3 = 8 * np.pi; n1 = 500;
n2=500
#-------
t1 = np.linspace (tt1, tt2, n1)
x1=np.cos(2*t1) *np.exp(0.1*t1)
y1=np.sin(2*t1) *np.exp(0.1*t1)
z1 = 0.1 * t1
#------
ttt=t1[500-1]
t2 = np.linspace (tt2, tt3, n2)
x2=np.cos(2*(t2)) *np.exp(-0.1*(t2-ttt)+0.1*ttt) #+
x1[500-2]
y2=np.sin(2*(t2)) *np.exp(-0.1*(t2-ttt)+0.1*ttt) #+
y1[500-2]
z2 = 0.1 * t2
#-----
#print("x=", x1[500-1], x2[0], t1[500-1], t2[0])
#print("y=", y1[500-1], y2[0], t1[500-1], t2[0])
#-----
x=np.concatenate((x1[:-1],x2))
y=np.concatenate((y1[:-1],y2))
z=np.concatenate((z1[:-1],z2))
```

```
#----
fig=plt.figure()
ax=A3D.Axes3D(fig)
ax.plot3D(x,y,z,'b-')
plt.show()
```



П22. Поверхні

Усі поверхні задаються параметричні: x = x (u, v), y = y (u, v), z = z (u, v).

Якщо хочемо задати поверхню «явно» z=z(x,y), то зручно створити масиви x=u і y=v функцією meshgrid.

Для всіх прикладів будемо використовувати наступну функцію від двох координат.

$$f(x,y) = \frac{\sin(x)\sin(y)}{xy}$$

Спочатку потрібно підготувати дані для малювання. Нам знадобляться три двомірні матриці:

- матриці X і Y зберігатимуть координати сітки точок, в яких обчислюватиметься наведена вище функція,
- матриця Z зберігатиме значення цієї функції у відповідній точці.

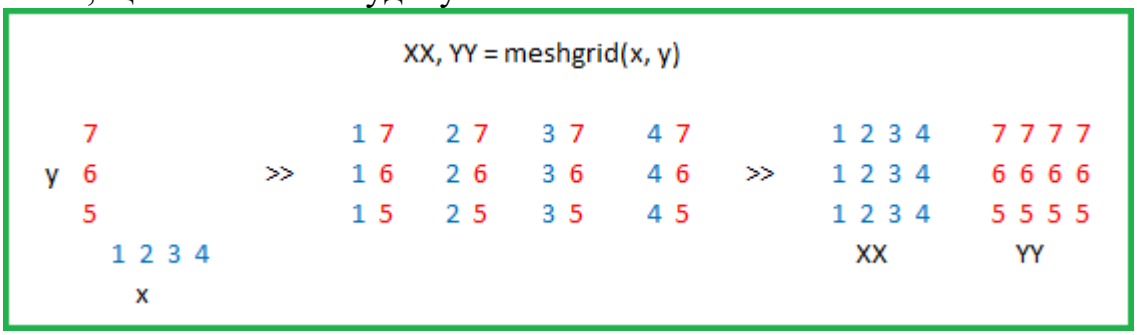
Якщо ми хочемо намалювати тривимірний графік на еквідистантній сітці (на сітці, яка має відстань між точками однакова), то для створення матриць, які зберігатимуть координати, будемо використовувати функцію meshgrid() з бібліотеки numpy.

Ця функція створює двовимірні матриці сіток за одновимірними масивами. Робота цієї функції дуже наочно показана нижче:

```
>>> X, Y = numpy.meshgrid([1,2,3], [4,5,6,7])
>>> Xarray([[1, 2, 3], [1, 2, 3], [1, 2, 3], [1, 2,
3]])

>>> Yarray([[4, 4, 4], [5, 5, 5], [6, 6, 6], [7, 7,
7]])
```

Тепер індексу вузла сітки можна дізнатися реальні координати: X[0][0] = 1, Y[0][0] = 4 тощо.

Фактично функція [X, Y] = meshgrid(x, y) задає сітку на площині xy у вигляді двовимірних масивів X, Y, які визначаються одновимірними масивами x та y. Рядки масиву X є копіями вектора x, а стовпці - копіями вектора y. Формування таких масивів полегшує обчислення функцій двох змінних, дозволяючи застосовувати операції над масивами. Нижче наведемо малюнок, що пояснює побудову масивів X і Y:



Щоб відокремити підготовку даних від самого малювання, створення сітки та розрахунок функції виділимо в окрему функцію:

```
defmakeData (): # Будуємо сітку в інтервалі від -10 до
10 # з кроком 0.1 по обох координатах
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    # Створюємо двомірну матрицю-сітку
    xgrid, ygrid = numpy.meshgrid(x, y)
    # У вузлах розраховуємо значення функції
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                        (xgrid * ygrid)
    returnxgrid, ygrid, zgrid
```

Ця функція повертає три двовимірні матриці: x, y, z. Координати x та y лежать в інтервалі від -10 до 10 з кроком 0.1.

Тепер повертаємось безпосередньо до малювання. Щоб відобразити наші дані, достатньо викликати метод plot_surface() екземпляра класу Axes3D, який передамо отримані за допомогою функції makeData() двомірні матриці.

Тепер наш приклад виглядає так:

```
importpylabfrom mpl_toolkits.mplot3d import
Axes3Dimport numpydef makeData(): x = numpy.arange (-
10, 10, 0.1)

   y = numpy.arange (-10, 10, 0.1)
   xgrid, ygrid = numpy.meshgrid(x, y)
   zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                        (xgrid * ygrid)
   returnxgrid, ygrid, zgrid

x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z)
pylab.show()
```

У нових версіях matplotlib замість

```
axes = Axes3D(fig)
```

рекомендується кодувати:

```
fig = pylab.figure()
axes = Axes3D(fig,auto_add_to_figure=False)
fig.add_axes(axes)
```

Так, наприклад, можна "підписати" осі координат:

```
axes.set_xlabel("-- x -->")
axes.set_ylabel("-- y -->")
axes.set_zlabel("-- z -->")
```

Якщо ми запустимо цей скрипт, то з'явиться вікно:

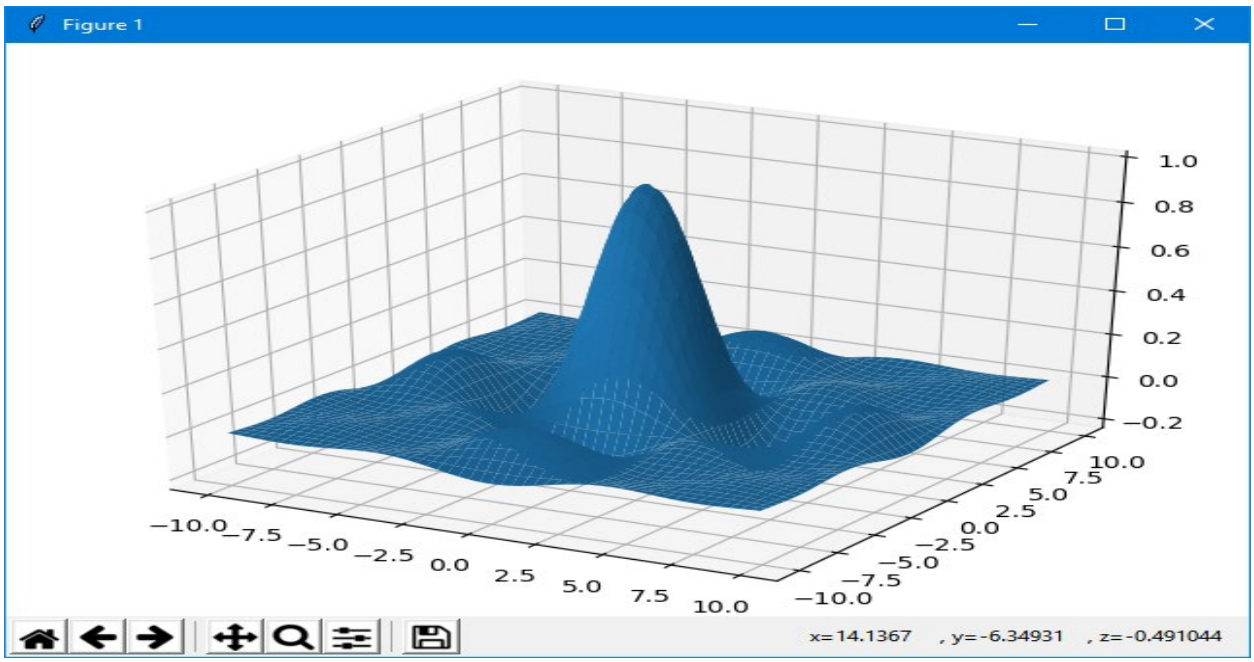


Рисунок можна обертати за допомогою мишки. Matplotlib не використовує графічний прискорювач, тому обертання відбувається досить повільно, хоча швидкість залежить кількості точок на поверхні.

Розглянемо інші параметри методу Axes3D.plot_surface(X, Y, Z, args, kwargs), їх не так багато:

- *X*, Y та Z - ці параметри задають сітку та значення функції у вузлах.
- *rstride* та cstride задають крок виведення графіка. Чим менший крок, тим точніше відображається функція, але довше відбувається малювання.
- *color* - задає колір графіка
- *cmap* - задає градієнт кольорів, щоб колір комірки графіка залежав від значення функції у цій галузі.

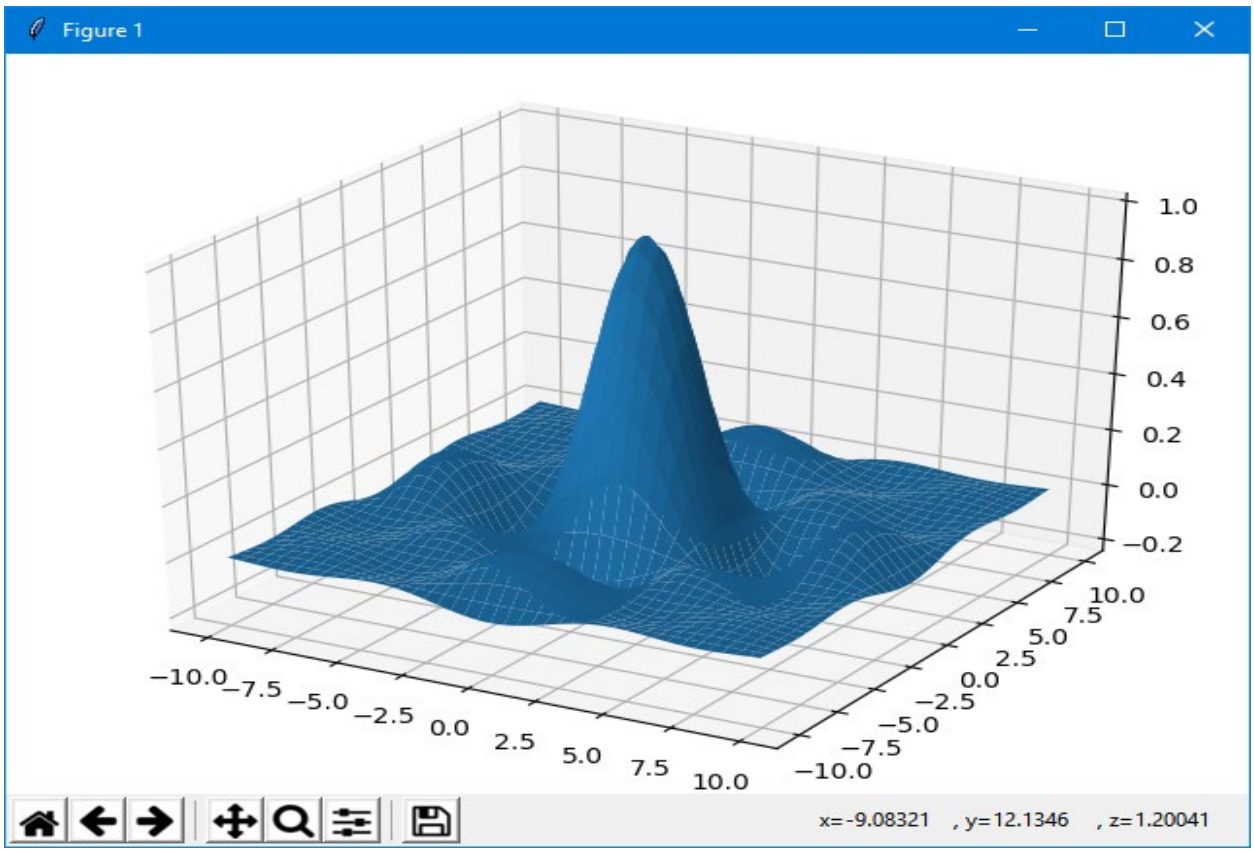
Розглянемо ці параметри.

Крок сітки

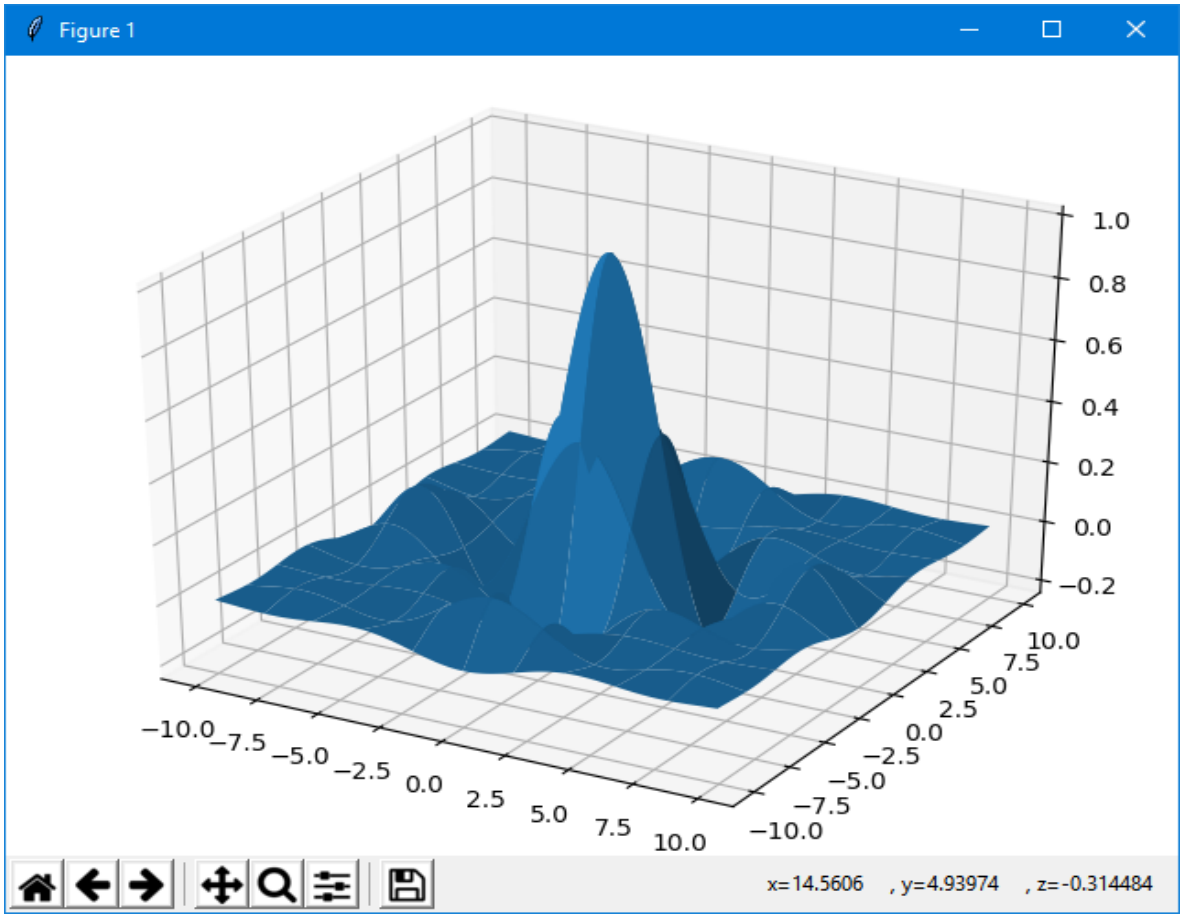
Спочатку подивимося як впливають зовнішній вигляд параметри rstride і cstride. Змінимо у попередньому прикладі рядок з використанням методу plot_surface() наступним чином:

```
axes.plot_surface(x, y, z, rstride=5, cstride=5)
```

В результаті ми отримаємо дрібнішу сітку на графіку:



Якщо так само встановити значення цих параметрів в 20, то сітка буде навпаки більша:



Зміна кольору

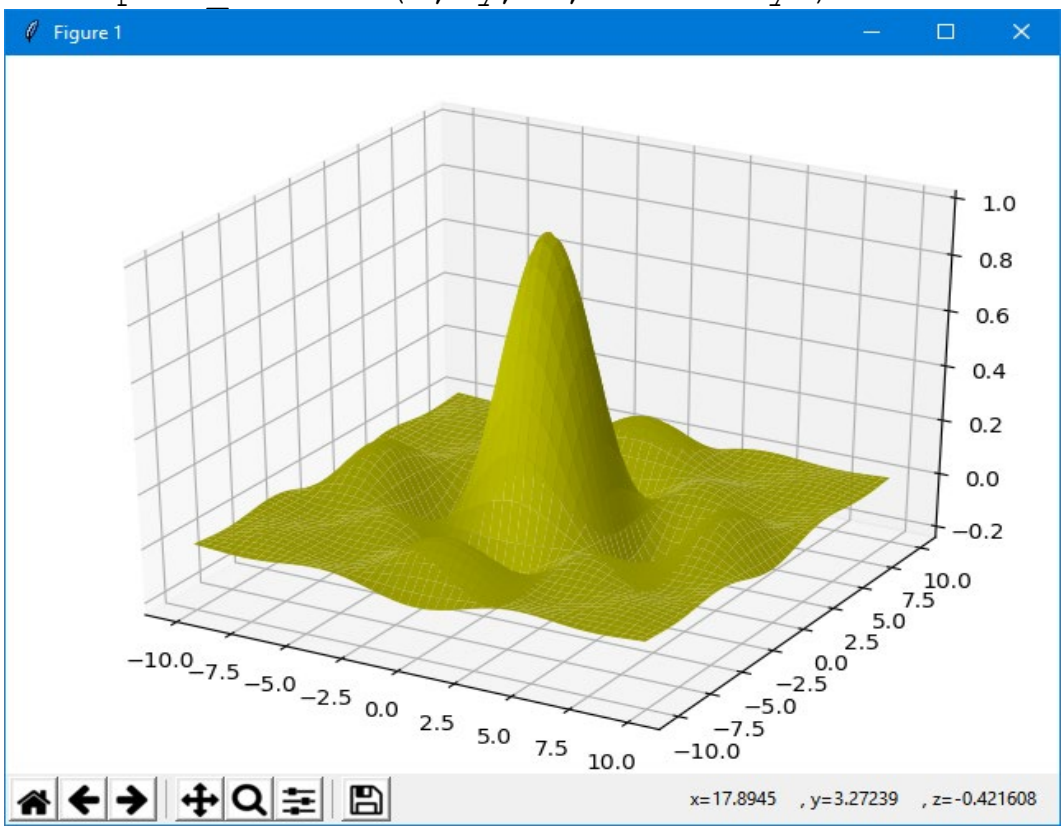
Тепер змінимо колір поверхні за допомогою параметра color. Цей параметр є рядком, який описує колір. Рядок кольору може задаватися різними способами:

Колір можна визначити англійським словом для відповідного кольору або однією літерою. Таких квітів небагато:

- 'b' або 'blue'
- 'g' або 'green'
- 'r' або 'red'
- 'c' або 'cyan'
- 'm' або 'magenta'
- 'y' або 'yellow'
- 'k' або 'black'
- 'w' або 'white'

Для прикладу зробимо поверхню жовтою:

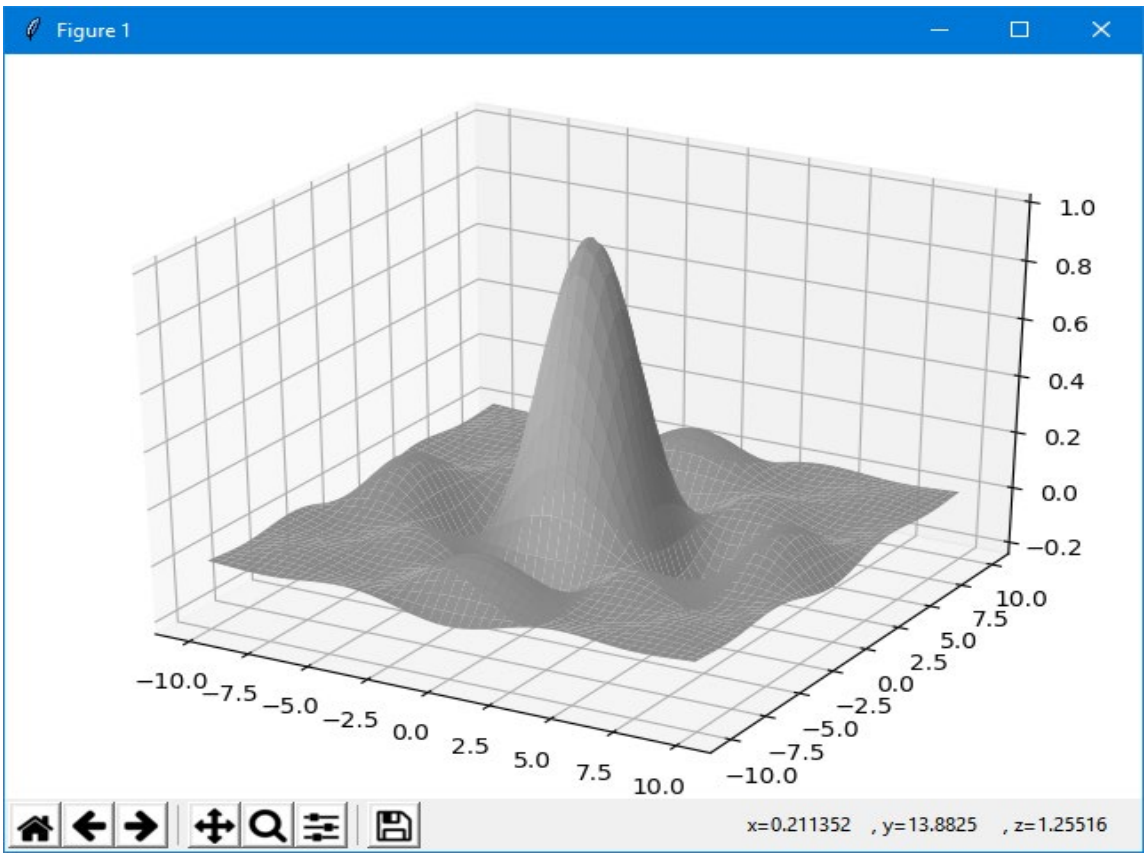
```
axes.plot_surface(x, y, z, color='yellow')
   або
axes.plot_surface(x, y, z, color='y')
```



Якщо нам потрібен сірий колір, його яскравість можемо задати за допомогою рядка, що містить число в інтервалі від 0.0 до 1.0 (0 - білий, 1 - чорний). Наприклад, можна написати наступний рядок:

```
axes.plot_surface(x, y, z, color='0.7')
```

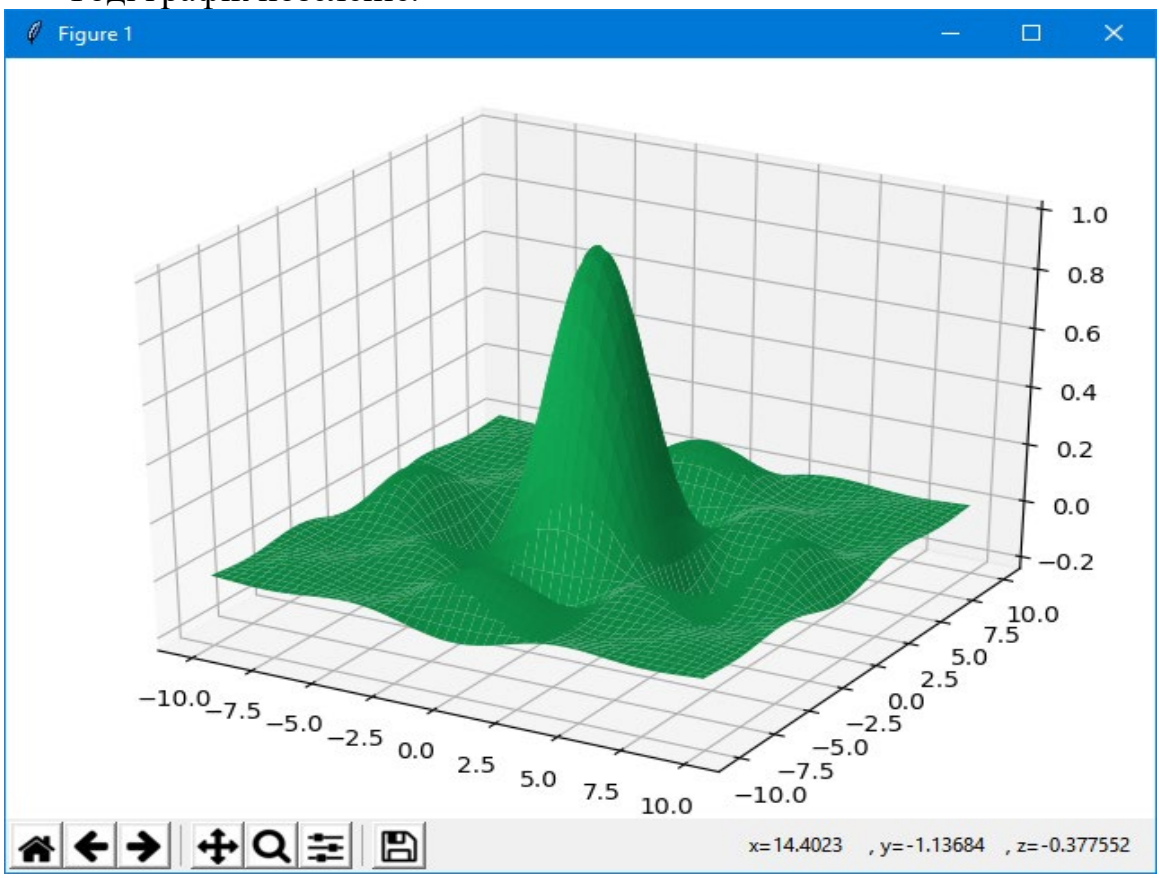
У цьому випадку ми побачимо такий ось сірий графік:



Крім того, ми можемо задавати колір, оскільки це прийнято в HTML після символу решітки ('#'). Наприклад, можемо задати колір так:

```
axes.plot_surface(x, y, z, color='#11aa55')
```

Тоді графік позеленіє:



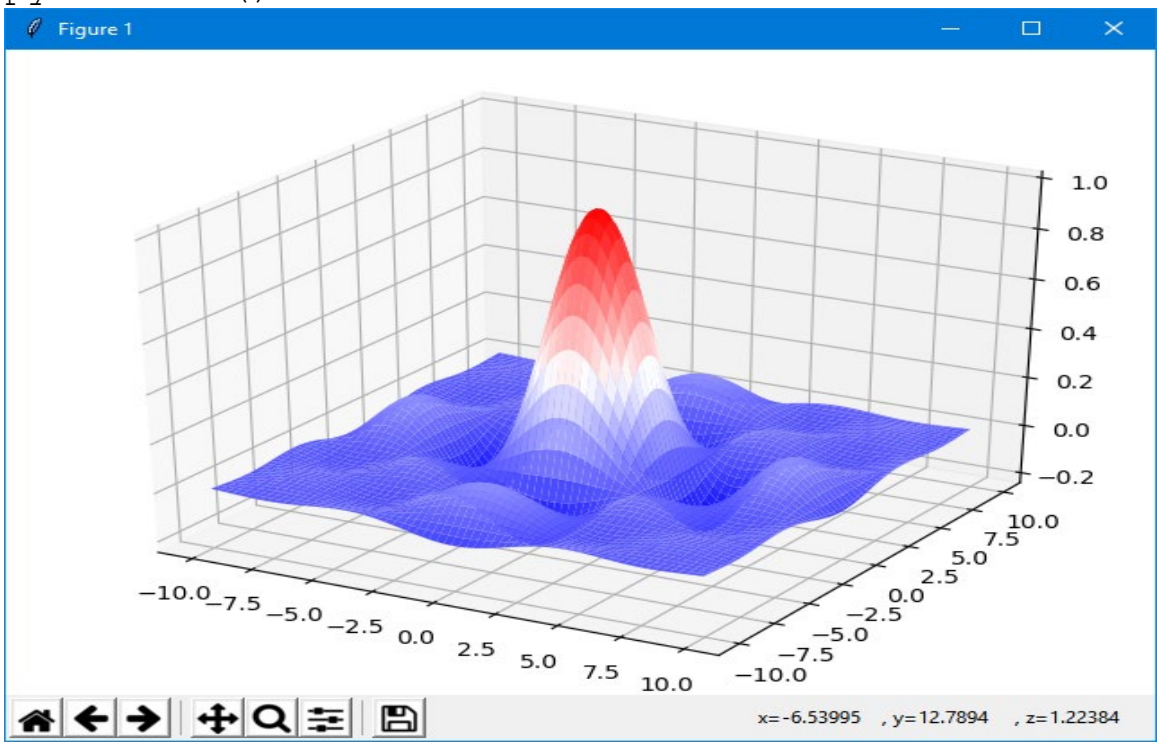
Використання колірних карток (colormap)

Кольорові карти використовуються, якщо потрібно вказати в які кольори мають фарбуватись ділянки тривимірної поверхні залежно від значення Z у цій галузі (завдання кольорового градієнта).

Щоб при виведенні графіка використовувався градієнт, як значення параметра cmap (від слова colormap, колірна карта) потрібно передати екземпляр класу matplotlib.colors.Colormap або похідного від нього.

Наступний приклад використовує клас LinearSegmentedColormap, похідний від Colormap, щоб створити градієнт переходу від синього кольору до червоного через білий.

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LinearSegmentedColormap
import numpy
def makeData():
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
            (xgrid * ygrid)
    return xgrid, ygrid, zgrid
x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=3, cstride=3, \
cmap = LinearSegmentedColormap.from_list ("red_blue",
                                  [ 'b', 'w', 'r'], 256))
pylab.show()
```

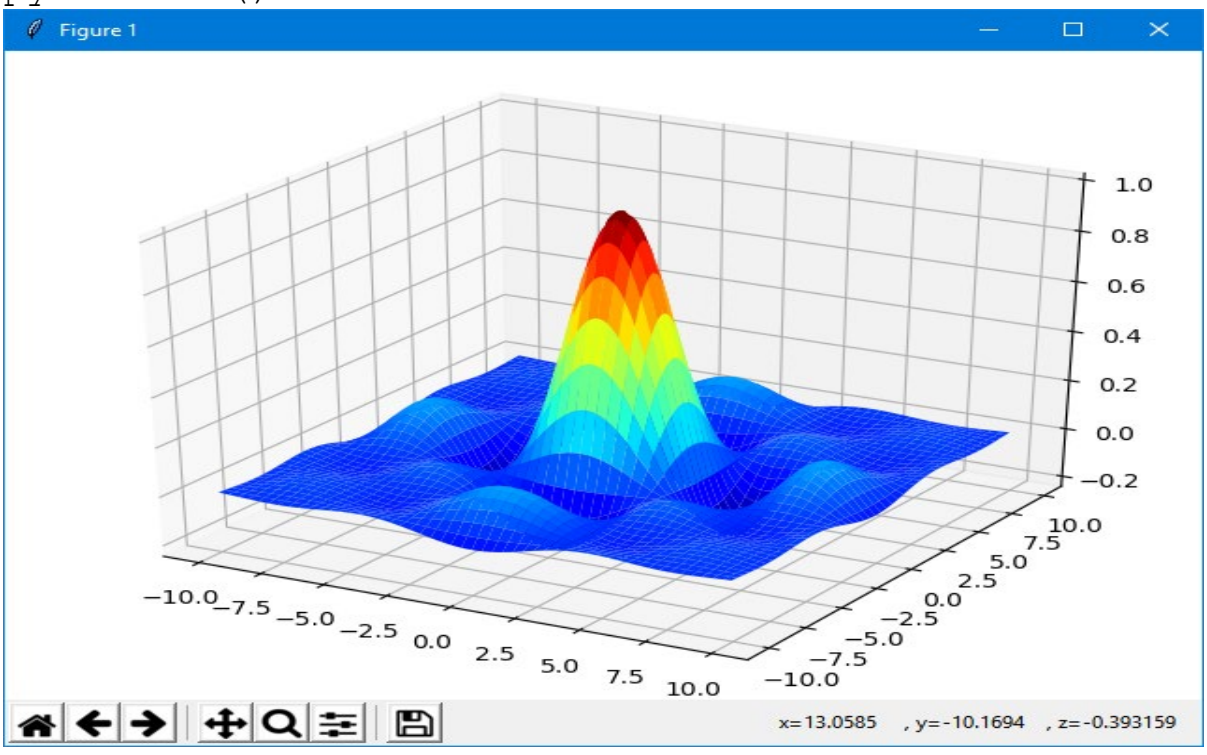


Тут використовується статичний метод from_list(), який приймає три параметри:

- Ім'я створюваної картки
- Список кольорів, починаючи з кольору для мінімального значення на графіці (блакитний – 'b'), через проміжні кольори (у нас це білий – 'w') до кольору для максимального значення функції (червоний – 'r').
- Кількість перехідів кольорів. Чим це число більше, тим паче плавний градієнт, але тим більше пам'яті він займає.

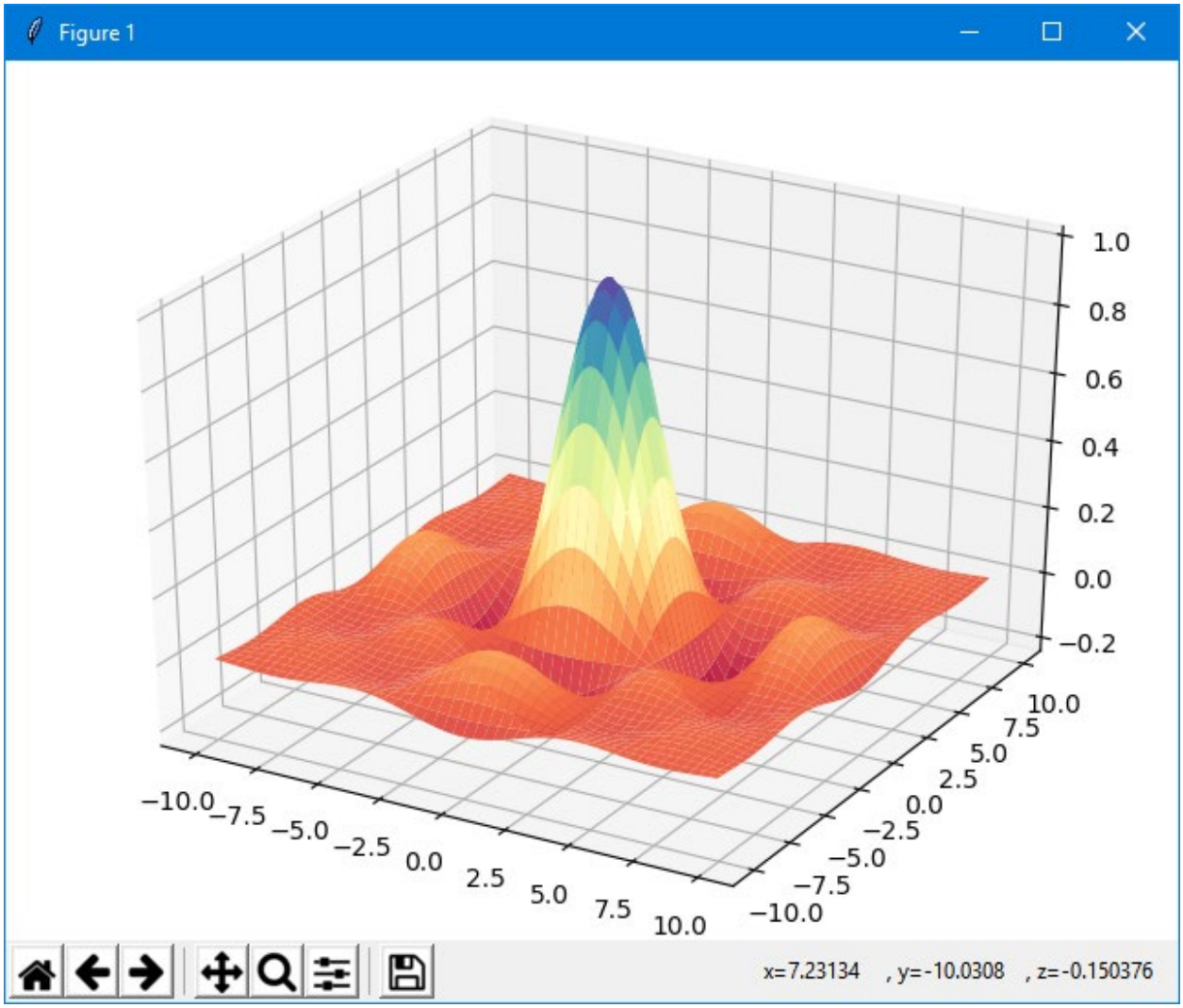
Карта cm.jet - це, напевно, найпоширеніша карта в прикладах.

```
importpylab
frommpl_toolkits.mplot3d import Axes3D
frommatplotlib.colors import LinearSegmentedColormap
frommatplotlib import cm
importnumpy
defmakeData():
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
(xgrid * ygrid)
    return xgrid, ygrid, zgrid
x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=4, cstride=4,
cmap=cm.jet)
pylab.show()
```



Зауважте, що як аргумент cmap ми передаємо не рядок, а ім'я змінної з модуля cm, причому це вже створений екземпляр класу, а не ім'я класу. Так, наприклад, cm.jet - це екземпляр класу matplotlib.colors.LinearSegmentedColormap.

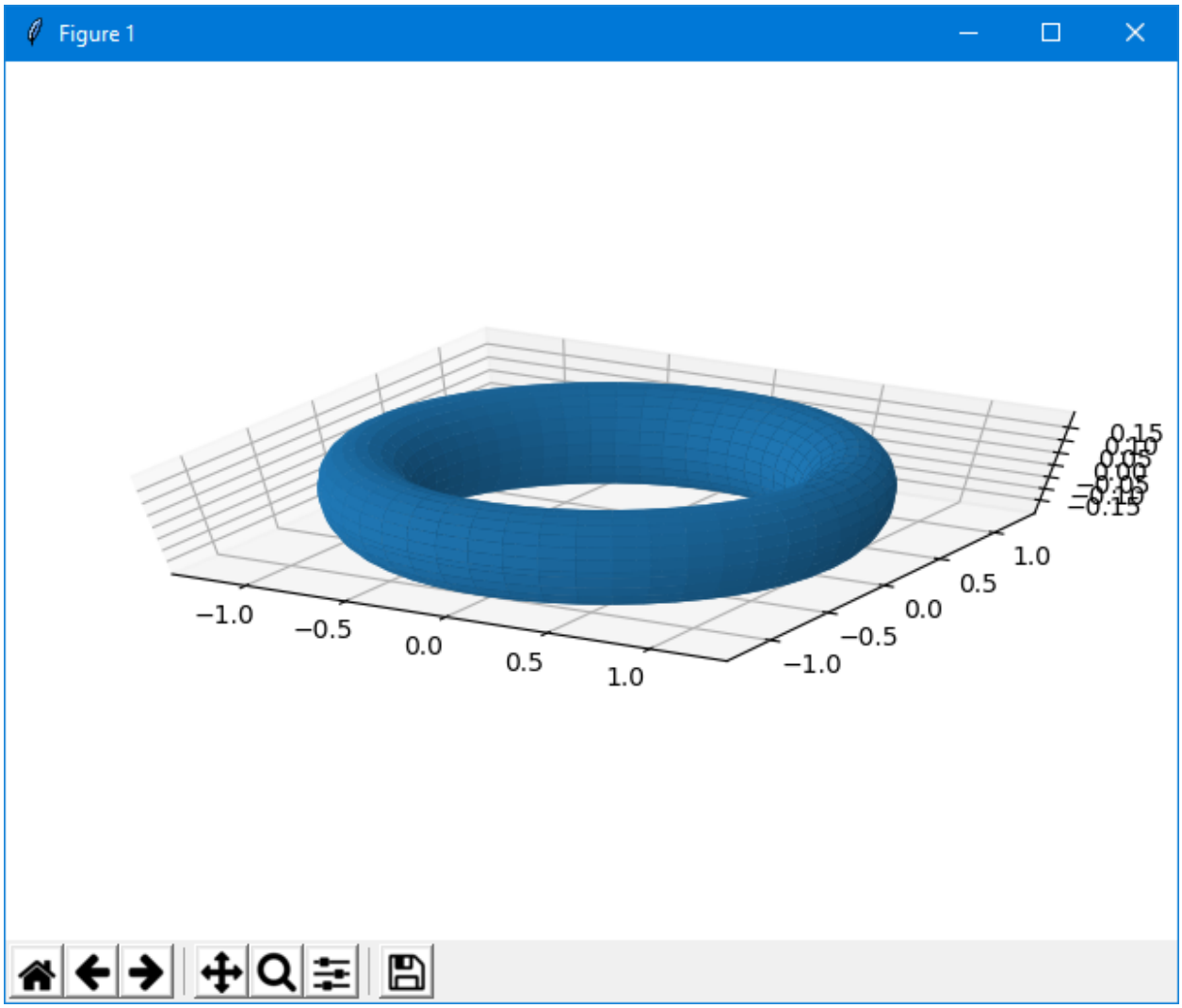
Для прикладу колірна карта cm.Spectral виглядає так:



П23. Параметричні поверхні з параметрами ϑ φ

```
importpylab
frommpl_toolkits.mplot3d import Axes3D
frommatplotlib.colors import LinearSegmentedColormap
frommatplotlib import cm
importnumpy as np

t=np.linspace(0,2*np.pi,50)
th,ph=np.meshgrid(t,t)
r=0.2
```

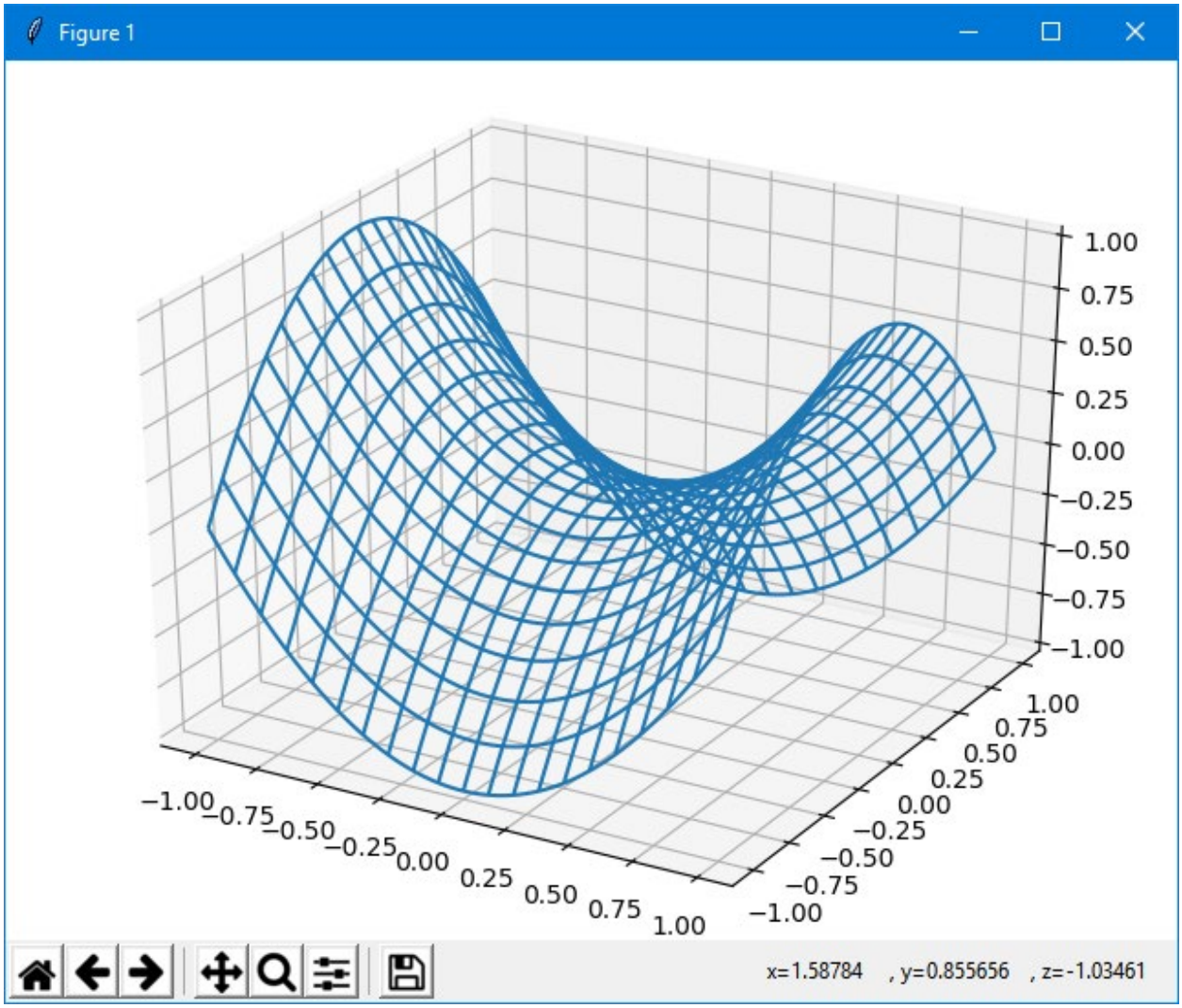
```
x,y,z=(1+r*np.cos(ph))*np.cos(th),(1+r*np.cos(ph))*np.s
in(th),r*np.sin(ph)
fig=pylab.figure()
ax=Axes3D(fig)
ax.elev=60
ax.set_aspect(0.3)
ax.plot_surface(x,y,z,rstride=2,cstride=1)
pylab.show()
```



П24. Побудова графіка «сітки» функції двох змінних

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
ax = axes3d.Axes3D(plt.figure())
i = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(i, i)
Z = X**2-Y**2
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
```

```
plt.show()
```



Для прикладу збудує три площини паралельні координатним площинам.

```
від mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
axes = axes3d.Axes3D(fig)
t = np.arange(-1, 1, 0.01)x, y = np.meshgrid(t, t)

axes.plot_wireframe(0*x, y, x, rstride=5, cstride=5,
color='r')

axes.plot_wireframe(x, 0*y, y, rstride=5, cstride=5,
color='b' )

axes.plot_wireframe(x, y, 0*x, rstride=5, cstride=5,
color='g')

plt.show()
```

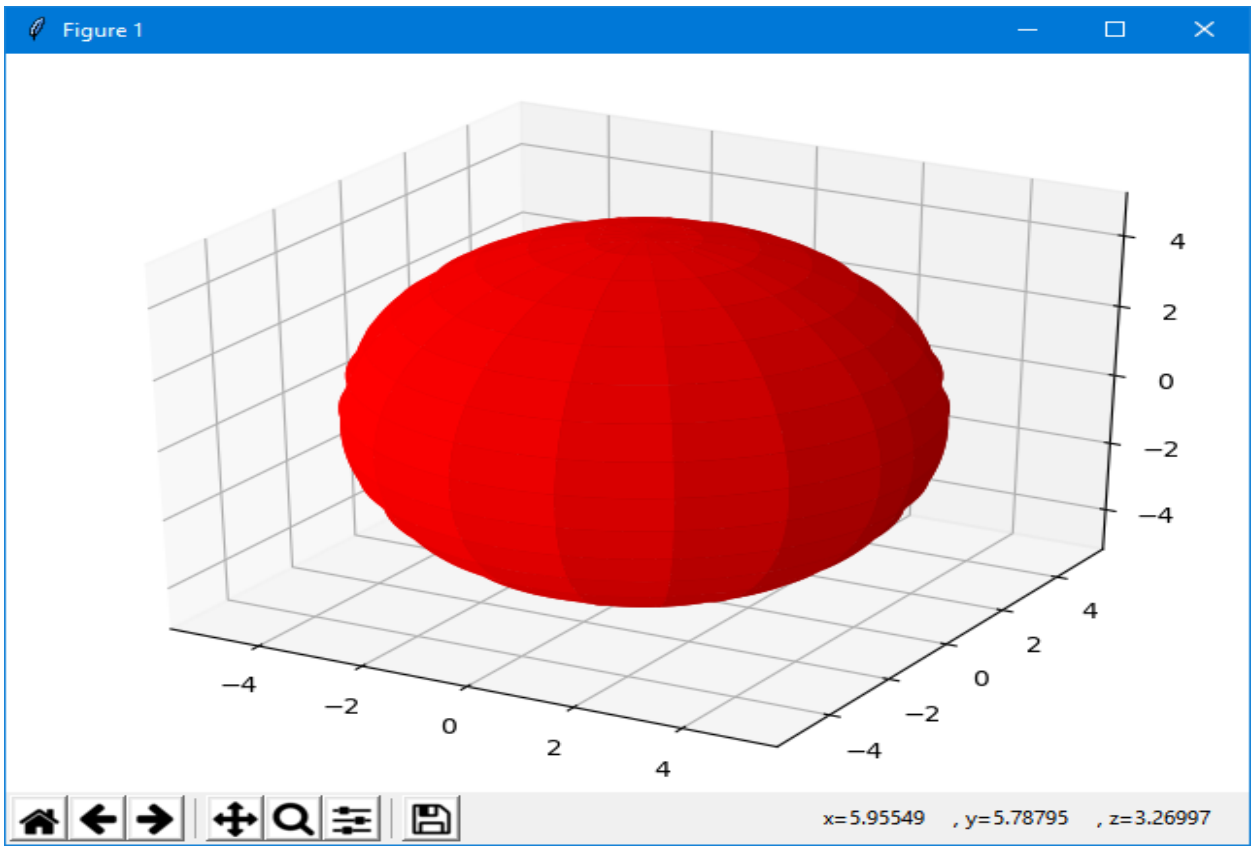
П25 Побудова графіка функції двох змінних x=x(u,v), y=y(u,v), z=z(u,v).

Поверхня визначається параметрично: x=x(u,v), y=y(u,v), z=z(u,v).

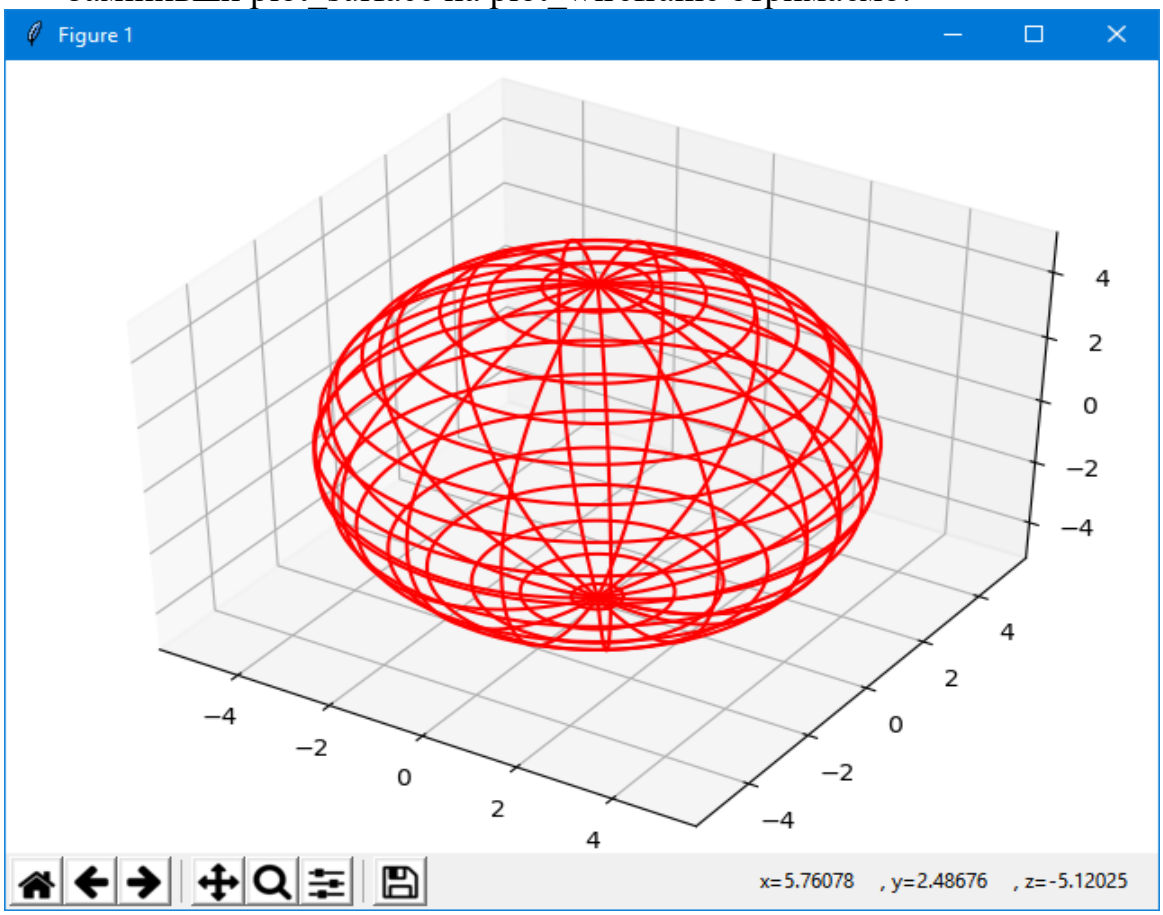
Класичне рівняння сфери, задане параметрично:

$x(u,v) = \cos(u) \cdot \cos(v)$
$y(u,v) = \sin(u) \cdot \cos(v)$
$z(u,v) = \sin(v)$
$u \in [-\pi, +\pi]$
$v \in [-2\pi, +2\pi]$

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
ax = axes3d.Axes3D(plt.figure())
u = np.linspace (0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 5*np.outer(np.cos(u), np.sin(v))
y =5*np.outer(np.sin(u), np.sin(v))
z =5*np.outer(np.ones(np.size(u)), np.cos(v))
ax.plot_surface(x, y, z, rstride=6, cstride=6,
color='r')
plt.show()
```



Замінивши plot_surface на plot_wireframe отримаємо:



У програмі використовувалася функція модуля numpy.outer, яка обчислює зовнішній (прямий або тензорний) добуток двох векторів:

numpy.outer(a, b, out=None)

Для двох векторів a довгою N та b довгою M зовнішній твір визначається наступним правилом:

```
[[a0*b0, a0*b1, ..., a0*bN],
[a1*b0, a1*b1, ..., a1*bN],
...
[aM*b1, aM*b1, ..., aM*bN]]
```

Це правило може бути узагальнено на масиви з більшою розмірністю. Але ця функція стискає всі багатовимірні масиви до однієї осі.

Параметри: a, b - числа, масиви NumPy чи подібні до масивів об'єкти. Одновимірні масиви, необов'язково однакової довжини. Двовимірні та багатовимірні масиви стискуються до однієї осі. out – масив NumPy, необов'язковий параметр.

Масив, в який можна помістити результат функції. Даний масив повинен відповідати формі та типу даних результуючого масиву функції, а також обов'язково бути C-суміжним, тобто. зберігати дані у рядковому З стилі. Вказівка даного параметра дозволяє уникнути зайвої операції присвоювання тим самим трохи прискорюючи роботу вашого коду. Корисний параметр якщо ви часто звертаєтеся до функції в циклі.

Повертає: результат - двовимірний масив NumPy, що є зовнішнім твором двох векторів.

Функція ones() повертає новий масив зазначеної форми та типу, заповнений одиницями. А np.ones(np.size(u)) - повертає новий масив, розміру і так само заповнений одиницями.

У наведеному вище прикладі будувалася так звана «щільна» сітка 3-мірного координатного простору на значеннях сітки координат x, y, z відповідно. Для цього використовувалася функція прямого або тензорного добутку двох векторів значень u та v.

Можна надійти «інше» - побудувати масив щільних координатних сіток 3-мірного координатного простору для зазначених у вигляді діапазонів одновимірних масивів координатних векторів u та v. Потім «просто» застосувати формули, що задають необхідні залежності x = x (u, v), y = y (u, v), z = z (u, v).

Для побудови масиву щільних координатних сіток використовуємо функцію numpy.mgrid:

```
numpy.mgrid[index object] =
<numpy.lib.index_tricks.nd_grid object>
```

Функція mgrid() повертає масив щільних координатних сіток N-вимірного координатного простору для зазначених у вигляді діапазонів одновимірних масивів координатних векторів.

Параметри: index object - об'єкт індексації

Під об'єктом індексації розуміється список із двох або трьох елементів, наприклад [0:5] або [0:5:10j].

Якщо елементів у списку всього два, це інтерпретується як напіввідкритий інтервал [start, ... , stop), у якому всі елементи відрізняються на 1, а значення stop в сам інтервал не входить.

Як третій елемент вказується уявна частина комплексного числа, яке вказує на кількість рівномірно рознесених елементів усередині закритого інтервалу [start, ... , stop], при цьому значення stop потрапляє в інтервал.

Повертає: результат - масив NumPy, що є масивом щільних координатних сіток N-мірного координатного простору, кількість та розміри яких залежать від зазначених діапазонів.

Побудову виконаємо наступним скриптом:

```
від mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
axes = axes3d.Axes3D(fig)
u, v = np.mgrid[0:2 * np.pi: 100j, 0:2 * np.pi: 100j]
x = 5*np.cos(u)*np.sin(v)
y = 5*np.sin(u)*np.sin(v)
z = 5 * np.cos (v)
#axes.plot_surface(x, y, z,rstride=6, cstride=6,
color='r')

axes.plot_wireframe(x, y, z, rstride=5, cstride=5,
color='r')

plt.show()
```

Для отримання масиву щільних координатних сіток можна також скористатися (розглянутою вище) функцією np.meshgrid:

```
u1=np.linspace(0,2*np.pi,100)
v1 = np.linspace (0,2 * np.pi, 100)
u, v = np.meshgrid(u1,v1)
```

За потреби можна весь малюнок «розтягнути» в kg разів по горизонталі та в kv разів по вертикалі. Для цього вкажемо параметр методу plt.figure():

```
kg=2; kv=1
fig = plt.figure(figsize=plt.figaspect(1/kg)*kv)
```

П26. "Скролінг" по осі X

Matplotlib включає дуже невелику кількість елементів управління, які в термінах Matplotlib називаються віджетами, які розташовуються пакеті matplotlib.widgets. Цей пакет містить також віджети для взаємодії з графіками (віджети для виділення областей) – докладніше див., наприклад, http://jenyay.net/Matplotlib/Widgets.

Використовуючи віджет Slider можна «розтягнути» вісь X:

```
from matplotlib.widgets import Slider
import math
import matplotlib.pyplot as plt
# докладніше див.
# http://jenyay.net/Matplotlib/Widgets
fig, ax = plt.subplots()
plt.subplots_adjust(left=0.25, bottom=0.25)
t = [i / 100. for i in range(0, int(math.pi) * 100, 1)]
s = [math.sin(i * 20) for i in t]
l, = plt.plot (t, s, lw = 2, color = 'red')

plt.axis([0, 1, -10, 10])
plt.grid(True)

axcolor = 'lightgoldenrodyellow' #'gray'
ax_x_pos = plt.axes([0.25, 0.1, 0.65, 0.03],
facecolor=axcolor)
wsize = 10
x_pos = Slider(ax_x_pos, 'Position', 0, l en(t) - wsize
- 1, valfmt='%d', \

                                valinit=0, color="g")

ax.set_xlim(t[0], t[wsize])
ax.set_ylim(-1.1, 1.1)

def update(val):
    #print(x_pos.val)
    pos = int(x_pos.val)
    ax.set_xlim(t[pos], t[pos + wsize])
    fig.canvas.draw_idle()

x_pos.on_changed(update)

plt.show()
```

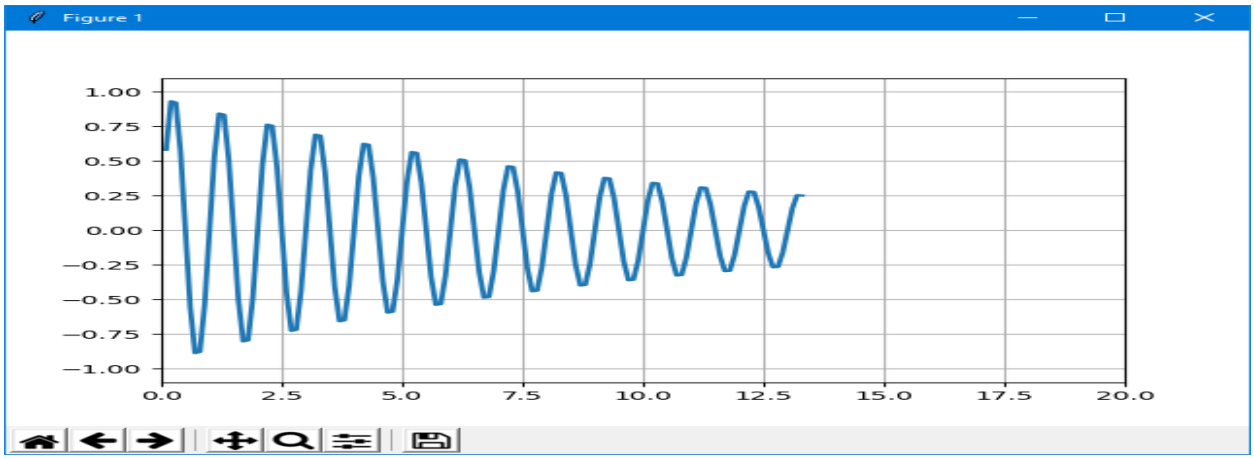
Результат роботи скрипту:

П27. «Динамічні» графіки

Без коментарів наведемо приклад «динамічної» побудови графіка см https://matplotlib.org/2.0.0/examples/animation/animate_decay.html:

```
'''
Цей випадок показуєвипадки виняткового позбавлення
animation.
'''
importnumpy as np
importmatplotlib.pyplot as plt
importmatplotlib.animation as animation

defdata_gen(t=0):
    cnt = 0
    while cnt < 1000:
        cnt += 1
        t += 0.1
        yield t, np.sin(2*np.pi*t) * np.exp(-t/10.)

definit():
    ax.set_ylim(-1.1, 1.1)
    ax.set_xlim(0, 10)
    del xdata[:]
    del ydata[:]
    line.set_data(xdata, ydata)
    return line,
```

```
fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.grid()
xdata, ydata = [], []

defrun(data):
    # update the data
    t, y = data
    xdata.append(t)
    ydata.append(y)
    xmin, xmax = ax.get_xlim()

    if t >= xmax:
        ax.set_xlim(xmin, 2*xmax)
        ax.figure.canvas.draw()
    line.set_data(xdata, ydata)
    return line,

ani = animation.FuncAnimation(fig, run, data_gen,
blit=False, interval=10,

                    repeat = False, init_func = init)
plt.show()
```



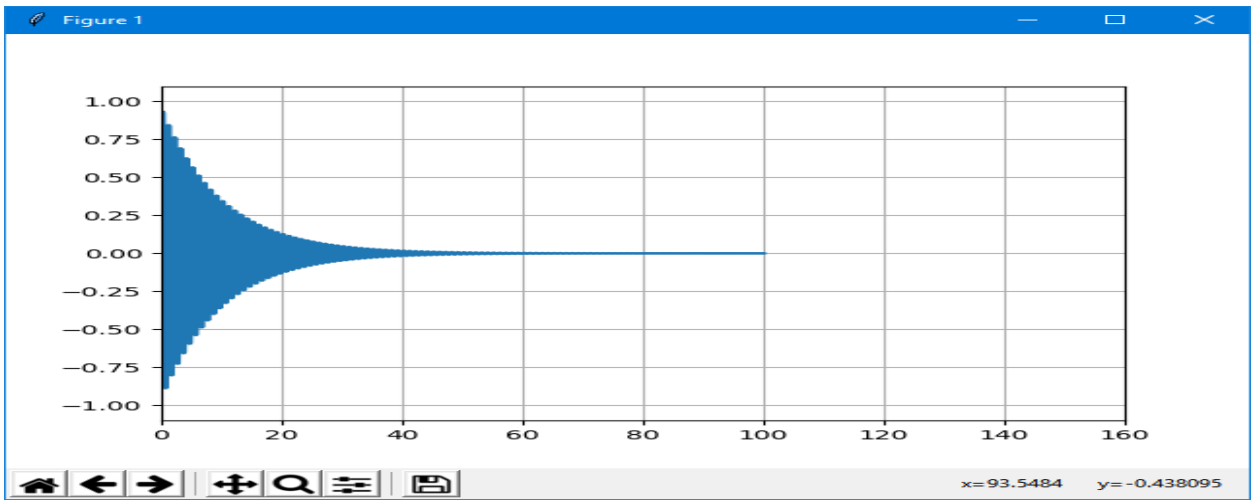


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120
140
160
x=93.5484
y=-0.438095

III. Модуль math

Вбудований модуль math у Python надає набір функцій для виконання математичних, тригонометричних та логарифмічних операцій. Деякі з основних функцій модуля:

- **pow(num, power)**: зведення числа num у ступінь power
- **sqrt(num)**: квадратний корінь числа num
- **ceil(num)**: округлення числа до найближчого найбільшого цілого
- **floor(num)**: округлення числа до найближчого найменшого цілого
- **factorial(num)**: факторіал числа
- **degrees(rad)**: переведення з радіан у градуси
- **radians(grad)**: переведення з градусів у радіани
- **cos(rad)**: косинус кута в радіанах.
- **sin(rad)**: синус кута в радіанах
- **tan(rad)**: тангенс кута в радіанах
- **acos(rad)**: арккосинус кута в радіанах
- **asin(rad)**: арксинус кута в радіанах
- **atan(rad)**: арктангенс кута в радіанах
- **log(n, base)**: логарифм числа n на основі base
- **log10(n)**: десятковий логарифм числа n

Приклад застосування деяких функцій:

```
import math

# Зведення числа 2 у ступінь 3
n1 = math.pow(2, 3)
print(n1) # 8

ту ж саму операцію можна виконати так
n2 = 2**3
print(n2)

# Зведення в квадрат
print(math.sqrt(9)) # 3

# найближче найбільше ціле число
print(math.ceil(4.56)) # 5

найближче найменше ціле число
print(math.floor(4.56)) # 4

# Переведення з радіан в градуси
```

```
print(math.degrees(3.14159)) # 180

# Переведення з градусів в радіани
print(math.radians(180)) # 3.1415.....
# косинус
print(math.cos(math.radians(60))) # 0.5
# Синус
print(math.sin(math.radians(90))) # 1.0
# тангенс
print(math.tan(math.radians(0))) # 0.0

print(math.log(8,2)) # 3.0
print(math.log10(100)) # 2.0
```

Також модуль math надає ряд вбудованих констант, такі як, PI та E:

```
import math
radius = 30
площа кола з радіусом 30
area = math.pi * math.pow (radius, 2)
print(area)

# натуральний логарифм числа 10
number = math.log(10, math.e)
print(number)
```

IV. Модуль fractions - раціональні числа (звичайні дроби)

Модуль fractions дозволяє виконувати арифметичні дії над раціональними числами, що у деяких ситуаціях буває надзвичайно зручно.

Створення звичайних дробів

Способів створення екземплярів раціональних чисел досить багато тому ми розділимо їх на групи.

Найпростіший спосіб створити звичайний дріб – вказати чисельник (numerator) та знаменник (denominator):

```
>>> з fractions import Fraction
>>>
>>> Fraction() # за замовчуванням numerator=0,
denominator=1
Fraction(0, 1)
>>>
>>> Fraction(numerator=1, denominator=2) # рівносильно
Fraction(1, 2)
```

```
Fraction(1, 2)
>>>
>>> Fraction(1, 2)
Fraction(1, 2)
```

Якщо вказані чисельник і знаменник мають спільні дільники, перед створенням раціонального числа вони скорочені:

```
>>> Fraction(8, 16), Fraction(15, 30)
(Fraction(1, 2), Fraction(1, 2))
```

Як чисельник і (або) знаменник можуть бути зазначені інші дроби:

```
>>> Fraction(3, Fraction(1, 2))
Fraction(6, 1)
>>>
>>> Fraction(Fraction(1, 2), 3)
Fraction(1, 6)
>>>
>>> Fraction(Fraction(1, 2), Fraction(3, 7))
Fraction(7, 6)
```

Ціле і речове число, так само можна перетворити на звичайний дріб:

```
>>> Fraction(10)
Fraction(10, 1)
>>>
>>> Fraction(11.11)
Fraction(781796747813847, 70368744177664)
>>>
>>> Fraction(1.25)
Fraction(5, 4)
>>>
>>> Fraction(2e-10)
Fraction(7737125245533627, 38685626227668133590597632)
```

А ось комплексні числа призведуть до помилки:

```
>>> Fraction(1j, 1 + 2j)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: both arguments повинні бути Rational
instances
```

Але уявною і дійсною частиною комплексного числа можуть бути вказані раціональні числа:

```
>>> complex(Fraction(3, 5), Fraction(4, 7))
(0.6+0.5714285714285714j)
```

Так само будь-який десятковий дріб модуля Decimal може бути перетворений на звичайний дріб:

```
>>> з import decimal Decimal
>>> Fraction(Decimal('7.7'))
Fraction(77, 10)
```

Будь-який рядок, який може бути перетворений на будь-яке допустиме число, також може бути використаний для отримання звичайних дробів:

```
>>> Fraction('111')
Fraction(111, 1)
>>>
>>> Fraction('1.11')
Fraction(111, 100)
>>>
>>> Fraction('1.11e+10')
Fraction(11100000000, 1)
>>>
>>> Fraction('-17/63')
Fraction(-17, 63)
>>>
>>> Fraction('-0.115')
Fraction(-23, 200)
>>>
>>> Fraction('-.115')
Fraction(-23, 200)
>>>
>>> Fraction('1.11 \t\n')
Fraction(111, 100)
>>> Fraction('\t\n 1.11 \t\n')
Fraction(111, 100)
>>> Fraction('\t\n 1.11')
Fraction(111, 100)
>>>
>>> Fraction('\t\n -13/37')
Fraction(-13, 37)
```

Математичні операції над раціональними числами

Усі арифметичні оператори підтримують обчислення з раціональними числами:

```
>>> x = Fraction(2, 5)
>>> y = Fraction(3, 7)
>>>
```

```
>>> x + y, y - x
(Fraction(29, 35), Fraction(1, 35))
>>>
>>> x*y, x/y
(Fraction(6, 35), Fraction(14, 15))
>>>
>>> x**0.5, 2**0.5/5**0.5
(0.6324555320336759, 0.6324555320336759)
>>>
>>> x**y, (x**3)**(1/7)
(0.6752339686501552, 0.6752339686501552)
>>>
>>> y//x
1
>>> x%y
Fraction(2, 5)
```

Однак арифметичні оператори не здатні до дій над типами 'Fraction' та 'decimal.Decimal' в одному виразі:

```
>>> x + 1.1
1.5
>>>
>>> x + Decimal('1.1') # призведе до помилки
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'Fraction' and 'decimal.Decimal'
```

Функції модуля math здатні приймати раціональні числа як аргументи, так як останні, по суті, можуть бути просто перетворені до речовинних чисел:

```
>>> import math
>>>
>>> x = Fraction(4, 11)
>>>
>>> math.exp(x), math.exp(4/11)
(1.438551009577678, 1.438551009577678)
```

Fraction - атрибути та методи

x.numerator

повертає чисельник дробу:

```
>>> з fractions import Fraction
>>>
>>> x = Fraction(3, 8)
```

```
>>> x
Fraction(3, 8)
>>>
>>> x.numerator
3
```

x.denominator

повертає знаменник дробу:

```
>>> x.denominator
8
```

Fraction.from_float(flt)

приймає flt – число типу float і повертає звичайний дріб ставлення чисельника до знаменника якого максимально наближається до значення flt:

```
>>> з fractions import Fraction
>>>
>>> Fraction.from_float(0.5)
Fraction(1, 2)
>>>
>>> Fraction.from_float(0.6)
Fraction(5404319552844595, 9007199254740992)
```

Як можна помітити, через зберігання чисел з плаваючою точкою в двійковому поданні Fraction.from_float(0.6) зовсім не одно Fraction(6, 10):

```
>>> Fraction.from_float(0.6) == Fraction(6, 10)
False
```

Fraction.from_decimal(dec)

створює звичайний дріб, який є точним уявленням десяткового дробу зазначеного в dec, де dec – це екземпляр класу decimal.

```
>>> з import decimal Decimal
>>>
>>> Fraction.from_decimal(Decimal('0.7'))
Fraction(7, 10)
>>>
>>> Fraction.from_decimal(Decimal('0.77'))
Fraction(77, 100)
>>>
>>> Fraction.from_decimal(Decimal('0.125'))
Fraction(1, 8)
```

Fraction.limit_denominator(max_denominator=1000000)

повертає найближче раціональне уявлення вказаного числа, знаменник якого не перевищує значення max_denominator.

```
>>> import math
```

```
>>> math.pi
3.141592653589793
>>>
>>> Fraction(math.pi).limit_denominator(10)
Fraction(22, 7)
>>>
>>> Fraction(math.pi).limit_denominator(1000)
Fraction(355, 113)
>>>
>>> Fraction(math.pi).limit_denominator(100000)
Fraction(312689, 99532)
```

Даний метод дозволяє отримувати як раціональні наближення, так і відновлювати раціональні значення за їх неточним поданням у вигляді чисел з плаваючою точкою:

```
>>> math.cos(math.pi/3) # точне значення 0.5
0.5000000000000001
>>>
>>> Fraction(math.cos(math.pi/3)).limit_denominator()
Fraction(1, 2)
>>>
>>>
>>> 2**(1/3)
1.2599210498948732
>>>
>>> Fraction(2**(1/3)).limit_denominator()
Fraction(1054215, 836731)
>>>
>>> 1054215/836731
1.2599210498953666
```

x.__floor__()

повертає значення типу int, яке є найбільшим серед усіх int <= x:

```
>>> Fraction(234, 47).__floor__()
4
```

Та й з огляду на те, що всі функції модуля math можуть обробляти прості дроби, то скористатися даним методом можна і так:

```
>>> math.floor(Fraction(234, 47))
4
```

x.__ceil__()

повертає значення типу int, яке є найменшим серед усіх int >= x:

```
>>> Fraction(234, 47).__ceil__()
5
```

```
>>>
>>> import math
>>>
>>> math.ceil(Fraction(234, 47))
5
```

x.__round__(ndigits=None)

Округлює до найближчого парного числа.

Якщо ndigits=None, округлення виконується до найближчого парного цілого числа:

```
>>> Fraction('1/2').__round__()
0
>>>
>>> Fraction('3/2').__round__()
2
```

Якщо ndigits не дорівнює None, то округлення виконується до найближчого числа кратного дробу Fraction(1, 10**ndigits), при цьому якщо остання цифра 5 то округлення буде виконано до найближчої парної цифри:

```
>>> Fraction('7/3').__round__(2)
Fraction(233, 100)
>>>
>>> Fraction('7/3').__round__(3)
Fraction(2333, 1000)
>>>
>>>
>>> Fraction('555/1000').__round__(1)
Fraction(3, 5)
>>>
>>> Fraction('555/1000').__round__(2)
Fraction(14, 25)
```

Якщо ndigits не дорівнює None, але менше 0, то округлення виконується до розряду на який вказує abs(ndigits), при цьому якщо остання цифра 5 то округлення буде виконано до найближчої парної цифри:

```
>>> Fraction('-5555555/10').__round__(-1)
Fraction(-555560, 1)
>>> Fraction('-5555555/10').__round__(-3)
Fraction(-556000, 1)
>>>
>>> Fraction('77777/10').__round__(-3)
Fraction(8000, 1)
```

fractions.gcd(a, b)

повертає найбільший загальний дільник чисел a та b.

Повертає НОД(a, b). Знак результату залежить від знака b, якщо b не дорівнює 0, інакше знак результату дорівнює знаку a. Повертає 0, тільки якщо a та b обидва рівні 0:

```
>>> fractions.gcd(135, 15)
15
>>> fractions.gcd(135, -15)
-15
>>> fractions.gcd(-135, 15)
15
>>> fractions.gcd(-135, 0)
-135
>>> fractions.gcd(0, 0)
0
```

З версії 3.5. вважається застарілим, і краще використовувати команду math.gcd().

Приклад:

Обчислити $2:\frac{3}{5}+\frac{3}{5}:2+1\frac{1}{2}:6+6:1\frac{1}{2}$

```
від fractions import Fraction as dr
rez=dr(2,1)/dr(3,5)+dr(3,10)+dr(3,2)/6+6/dr(3,2)
print(rez, '=',end="")
a = int (rez.numerator / rez.denominator)
b = rez - a
print(a,"+", b, sep="")

>>> 473/60 = 7+53/60
```

V. Модуль cmath

Модуль cmath – надає функції до роботи з комплексними числами.

cmath.phase(x) – повертає фазу комплексного числа (її ще називають аргументом). Еквівалентно math.atan2(x.imag, x.real). Результат лежить у проміжку [-π, π].

Отримати модуль комплексного числа можна за допомогою вбудованої функції abs().

cmath.polar(x) – перетворення до полярних координат. Повертає пару (r, phi).

cmath.rect(r, phi) – перетворення з полярних координат.

cmath.exp(x) - ex.

cmath.log(x[, base]) - логарифм x на основі base. Якщо base не вказано, повертається натуральний логарифм.

cmath.log10(x) – десятковий логарифм.

cmath.sqrt(x) - квадратний корінь із x.

cmath.acos(x) – арккосинус x.
cmath.asin(x) – арксинус x.
cmath.atan(x) – арктангенс x.
cmath.cos(x) – косинус x.
cmath.sin(x) – синус x.
cmath.tan(x) – тангенс x.
cmath.acosh(x) – гіперболічний арккосинус x.
cmath.asinh(x) – гіперболічний арксинус x.
cmath.atanh(x) – гіперболічний арктангенс x.
cmath.cosh(x) – гіперболічний косинус x.
cmath.sinh(x) – гіперболічний синус x.
cmath.tanh(x) – гіперболічний тангенс x.
cmath.isfinite(x) - True, якщо дійсна та уявна частини кінцеві.
cmath.isinf(x) - True, якщо дійсна, або уявна частина нескінченна.
cmath.isnan(x) - True, якщо або дійсна, або уявна частина NaN.
cmath.pi- π.
cmath.e- e.

VI. Модуль struct -упаковка даних у бінарний файл

У деяких додатках доводиться мати справу з упакованими двійковими даними, які створюються, наприклад, програмами на мові C. Для їх збереження та відновлення можна скористатися модулем struct із стандартної бібліотеки Python.

Приклад запису у файл та читання з файлу:

```
import struct

myfile = open("data.bin", "wb")
# упаковуємо дані
bytes = struct.pack(b'>i4sh', 7, b'spam', 8)
myfile.write(bytes)
myfile.close()

myfile = open("data.bin", "rb")
data = myfile.read()
# Вилучаємо дані
values = struct.unpack(b'>i4sh', data)
print(values)
```

Методи

struct.unpack(format, data)
Розпаковує дані зі структури

Приклади:

```
два цілі 4х байтові числа прямого порядку
struct.unpack('>LL', b'x00x00x00x9ax00x00x00x8d')
# (154, 141)

два цілі 4х байтові числа прямого порядку
struct.unpack('>2L', b'x00x00x00x9ax00x00x00x8d')
# (154, 141)

два цілі 4х байтові числа прямого порядку, пропустивши
1 і 2 байти по краях
struct.unpack('>1x2L2x',
     b'x00x00x00x00x9ax00x00x00x8dx00x00')
# (154, 141)
```

struct.pack(format, data)

Пакує дані до структури

```
struct.pack('>L', 154)
# b'x00x00x00x9a'

struct.pack('>L', 141)
# b'x00x00x00x8d'
```

Специфікатори формату

>- Прямий порядок
< -зворотний порядок
x- пропустити один байт
b -знаковий один байт
B -беззнаковий байт
h -знакове коротке ціле число, 2 байти
H -беззнакове коротке ціле число, 2 байти
i -знакове ціле число, 4 байти
I -беззнакове ціле число, 4 байти
l -знакове довге ціле число, 4 байти
L -беззнакове довге ціле число, 4 байти
Q -беззнакове дуже довге ціле число, 8 байт
f -число з плаваючою точкою, 4 байти
d -число з плаваючою точкою подвійної точності, 8 байт
p -лічильник та символи, 1 + count байт
s -символи, count символів

Приклад запису та читання речових даних у бінаному файлі

```
importsys
importstruct
myfile = open("data.bin", "wb")
# упаковуємо дані
fori in range(100):
    x1=10.0*float(i+0)
    x2=10.0*float(i+1)
    x3=10.0*float(i+2)
    x4=10.0*float(i+3)
    x5=10.0*float(i+4)
    print(x1, x2, x3, x4, x5)
    b=struct.pack(b'>fffff', x1,x2,x3,x4,x5)
    myfile.write(b)
myfile.close()
#sys.exit(0)
myfile = open("data.bin", "rb")
while(True):
    data = myfile.read(4*5)
    if data==b'':
        break
    # Вилучаємо дані
    values = struct.unpack(b'>fffff', data)
    y1=values[0]
    y2=values[1]
    y3=values[2]
    y4=values[3]
    y5=values[4]
    print(y1, y2, y3, y4, y5)
myfile.close()
```

Приклад запису файлу на FORTRAN та читання на PYTHON речових даних у бінаному файлі

Формування файлу:

```
! компілятор ming gfortran
  implicit none
  integer :: j, i
  real(4), DIMENSION(5) :: buf
  character(40) :: fname='d:\float5.bin'
  open(unit=50, file=trim(fname), status='UNKNOWN',
form='UNFORMATTED',err=100)
!
! запис у файлі містить у "початку та "в кінці" запису
```

```
! "дискриптор" Чотири байти - кількість байт
! у запису крім дискрипторів
!
  do i=1,50
     buf(1)=float(i)
     buf(2)=float(i+1)
     buf(3)=float(i+2)
     buf(4)=float(i+3)
     buf(5)=float(i+4)
     write(50) (buf(j), j=1,5)
     write(*, '( 5(g12.5,1x) )' ) (buf(j), j=1,5)
  enddo
  close(50)
  stop 0
100 write(*,*) '* error open file ', trim(fname), '*'
    stop 16
    end
```

Читання файлу:

```
importsys
importstruct
myfile = open(r"d:\float5.bin", "rb")
##kbyteB=myfile.read(4) # дискриптор запису
nzap=0
while(True):
    kbyteB=myfile.read(4) # дискриптор запису -
                          # кількість байт у цьому
записі
    if kbyteB==b'':
        break
    nzap+=1
    kbyte = struct.unpack(b'<L', kbyteB)[0]
    print('запис № ',nzap,' містить ',kbyte,' байт')
    data = myfile.read(4*5) #читання запису
    if data==b'':
        break
    # Вилучаємо дані
    values = struct.unpack(b'<fffff', data)
    y1=values[0]
    y2=values[1]
    y3=values[2]
    y4=values[3]
    y5=values[4]
    print(y1, y2, y3, y4, y5) #values)
```

```
    kbyteB=myfile.read(4) # дискриптор кінця запису - #
кількість байт у цьому записі
myfile.close()
```

VII. Файли CSV

Програмісти часто стикаються із завданням обробки великих обсягів структурованих даних. Python має вбудовану бібліотеку CSV, за допомогою якої програміст може працювати із спеціальними CSV файлами. Це своєрідні електронні таблиці.

Кожен рядок у файлі csv представляє окремий запис або рядок, який складається з окремих стовпців, розділених комами. Саме тому формат і називається Comma Separated Values. Але хоча формат CSV - це формат текстових файлів, Python для спрощення роботи з ним надає спеціальний вбудований модуль CSV

Що таке файли CSV

CSV - це особливий вид файлу, який дозволяє структурувати великі обсяги даних.

По суті він є звичайним текстовим файлом, однак кожен новий елемент відокремлений від попереднього комою або іншим роздільником. Зазвичай кожен запис починається з нового рядка. Дані CSV можна легко експортувати до електронних таблиць або баз даних. Програміст може розширювати файл CSV, додаючи нові рядки.

Приклад CSV файлу, де як роздільник використовується кома:

```
Ім'я,Професія,Рік народження
Віктор, Токар, 1995
Сергій, Зварювальник, 1983
```

Як видно з прикладу, у першому рядку зазвичай вказується, яка інформація буде знаходитись у кожному стовпці. Крім того, після останнього елемента рядка кома не ставиться, інтерпретатор визначає кінець рядка за символом перенесення.

Замість коми можна використовувати будь-який інший роздільник, тому при читанні файлу CSV потрібно заздалегідь знати, який символ використовується.

Важливо пам'ятати, що CSV - це звичайний текстовий файл, який не підтримує символи в кодуваннях, які відрізняються від ASCII або Unicode.

Бібліотека CSV

Це основна бібліотека для роботи з файлами CSV в Python.

Бібліотека csv є вбудованою, тому її не потрібно завантажувати, достатньо використовувати звичайний імпорт:

```
import csv
```

Читання з файлів (парсинг)

Для того, щоб прочитати дані з файлу, програміст повинен створити об'єкт reader:

```
reader_object = csv.reader(file, delimiter = ",")
```

readerмає метод __next__(), тобто є об'єктом, що ітерується, тому читання з файлу відбувається наступним чином:

```
import csv
with open("classmates.csv", encoding='utf-8') as r_file:
    # Створюємо об'єкт reader, вказуємо символ-розділювач ","
    file_reader = csv.reader(r_file, delimiter = ",")
    # Лічильник для підрахунку кількості рядків та виведення
заголовків стовпців
    count = 0
    # Зчитування даних із CSV файлу
    for row in file_reader:
        if count == 0:
            # Виведення рядка, що містить заголовки для стовпців
            print(f'Файл містить стовпці: {", ".join(row)}')
        else:
            # Виведення рядків
            print(f'{row[0]} - {row[1]} і він народився в
{row[2]} році.')

        count += 1
    print(f'Всього у файлі {count} рядків.')
```

Припустимо, що ми маємо CSV файл, який містить таку інформацію:

```
Ім'я,Успішність,Рік народження
Саша, відмінник, 200
Маша,хорошистка,1999
Петя, трієчник, 2000
```

Тоді, якщо відкрити цей файл у нашій програмі, будуть отримані наступні результати:

```
Файл містить стовпці: Ім'я, Успішність, Рік народження
    Сашко - відмінник і він народився 200 року.
    Маша - хорошистка і він народився 1999 року.
    Петро - трієчник і він народився 2000 року.
Усього у файлі 4 рядків.
```

Використання конструкції with…as дозволяє програмісту бути впевненим, що файл буде закритий, навіть якщо під час виконання коду станеться якась помилка.

Зверніть увагу, що при відкритті потрібно вказати правильне кодування, в якому зберігаються дані. В даному випадку encoding = 'utf-8'. Якщо не вказувати, то використовуватиметься кодування за замовчуванням. Для Windows це CP1251.

Бібліотека CSV дозволяє працювати з файлами, як із словниками, для цього потрібно створити об'єкт DictReader. Звертатися до елементів можна на ім'я стовпців, а не за допомогою індексів. Щоб вихідна програма робила аналогічний висновок, її слід змінити так:

```
import csv
```

```
with open("classmates.csv", encoding='utf-8') as r_file:
    #Створюємо об'єкт DictReader, вказуємо символ-розділювач ","
    file_reader = csv.DictReader(r_file, delimiter = ",")
    # Лічильник для підрахунку кількості рядків та виведення
заголовків стовпців

    count = 0
    # Зчитування даних із CSV файлу
    for row in file_reader:
        if count == 0:
            # Виведення рядка, що містить заголовки для стовпців
            print(f'Файл містить стовпці: {", ".join(row)}')
        # Виведення рядків
        print(f' {row["Ім'я"]} - {row["Успішність"]}', end='')
        print(f' і він народився в {row["Рік народження"]}
року.')
        count += 1
    print(f'Всього у файлі {count + 1} рядків.')
```

Звертатися до елементів за назвою стовпця зручніше, крім того, це спрощує розуміння коду.

Зверніть увагу, що цикл for при першій ітерації буде записаний в row не шапка таблиці, а перший її рядок. Тому за виведення кількості рядків змінну count збільшили на 1.

Додаткові параметри об'єкта DictReader

DictReaderмає параметри:

- `dialect`— Набір параметрів форматування інформації. Докладніше про них нижче.
- `line_num`— Встановлює кількість рядків, які можна прочитати.
- `fieldnames`— Визначає заголовки для стовпців, якщо не визначити атрибут, в нього запишуться елементи з першого прочитаного рядка файлу. Заголовки потрібні для того, щоб легко було зрозуміти, яка інформація міститься або повинна міститись у стовпці.

Наприклад, якби в classmates.csv не було першого рядка із заголовками, то можна було б його відкрити наступним чином:

```
fieldnames = ['Ім'я', 'Успішність', 'Рік народження']
file_reader = csv.DictReader(r_file, fieldnames =
fieldnames)
```

Також можна використовувати метод __next__() для отримання наступного рядка. Цей метод робить об'єкт reader ітерованим. Тобто він викликається за кожної ітерації і повертає наступний рядок. Цей метод і використовується при кожній ітерації в циклі для отримання чергового рядка.

Запис до файлів

Для запису інформації в CSV файл необхідно створити об'єкт writer:

```
file_writer = csv.writer(w_file, delimiter = "\t")
```

Для запису файл даних використовується метод writerow(), який має наступний синтаксис:

```
writerow(["Ім'я", "Прізвище", "По батькові"])
```

Код програми для запису в CSV файл виглядає так:

```
import csv
with open("classmates.csv", mode="w", encoding='utf-8')
as w_file:
    file_writer = csv.writer(w_file, delimiter = ",",
lineterminator="\r")

    file_writer.writerow(["Ім'я", "Клас", "Вік"])
    file_writer.writerow(["Женя", "3", "10"]))
    file_writer.writerow(["Саша", "5", "12"]))
    file_writer.writerow(["Маша", "11", "18"])
```

Зверніть увагу, що при записі використовувався lineterminator="r". Це роздільник між рядками таблиці, за умовчанням він \r\n.

Після виконання програми у файлі CSV буде наступний текст:

```
Ім'я,Клас,Вік
Женя,3,10
Саша, 5,12
Маша, 11,18
```

Як параметр метод writerow() приймає список, елементи якого будуть записані в рядок через символ-розділювач.

Запис у файл також можна здійснити за допомогою об'єкта DictWriter.

Важливо пам'ятати, що він потребує явної вказівки параметра fieldnames. Як аргумент методу writerow використовується словник.

Код програми виглядає так:

```
import csv
with open("classmates.csv", mode="w", encoding='utf-8') as
w_file:

    names = ["Ім'я", "Вік"]
    file_writer = csv.DictWriter(w_file, delimiter = ",",
                                 lineterminator="\r",
                                     fieldnames=names)
    file_writer.writeheader()
    file_writer.writerow({"Ім'я": "Саша", "Вік": "6"})
    file_writer.writerow({"Ім'я": "Маша", "Вік": "15"})
    file_writer.writerow({"Ім'я": "Вова", "Вік": "14"})
```

Виведення у файл буде наступним:

```
Ім'я,Вік
Саша,6
Маша,15
Вова,14
```

Додаткові параметри DictWriter

Об'єкт writer також має атрибут dialect, який визначає, як форматуватимуться дані під час запису файл, про нього буде описано нижче.

Крім того, writer має методи:

- **writerows(rows)**- Записує всі елементи рядків.
- **writeheader()**-Виводить заголовки для стовпців. Заголовки повинні бути передані об'єкту writer у вигляді списку, як атрибут fieldnames. writeheader був використаний у попередньому прикладі.

Розглянемо застосування writerows:

```
file_writer.writerows([{"Ім'я": "Саша", "Вік": "6"},
                       {"Ім'я": "Маша", "Вік": "15"},
                      {"Ім'я": "Вова", "Вік": "14"}])
```

Діалекти

Щоб не вказувати формат вхідних і вихідних даних, певні параметри форматування згруповані в діалекти (dialect).

При створенні об'єкта reader або writer програміст може вказати потрібний діалект, крім того, деякі параметри діалекту можна перевизначити вручну, також вказавши їх при створенні об'єкта.

Для створення діалекту використовується команда:

```
register_dialect("ім'я", delimiter = "\t", ...)
```

Клас Dialect дозволяє визначити наступні атрибути форматування:

Атрибут	Значення
`delimiter`	Встановлює символ, за допомогою якого елементи розділяються у файлі. За замовчуванням використовується кома.
`doublequote`	Якщо True, то символ quotechar подвоюється, якщо False, то до символу qutechar додається ecsapechar як префікс.
`escapechar`	Рядок з одного символу, який використовується для екранування символу-розділювача.
`lineterminator`	Визначає роздільник для рядків, за замовчуванням використовується \r\n
`quotechar`	Визначає символ, який використовується для оточення символу-розділювача. За замовчуванням використовуються подвійні лапки, тобто quotechar = '"'.
`quoting`	Визначає символ, який використовується для екранування роздільного символу (якщо не використовуються лапки).
`skipinitialspace`	Якщо встановити значення цього параметра в True, всі прогалини після символу-розділювача будуть ігноруватися.

Атрибут	Значення
strict	Якщо встановити в True, то при неправильному введенні CSV збуджуватиметься виняток Error.

Приклад використання:

```
import csv
csv.register_dialect('my_dialect', delimiter=':',
lineterminator="\r")

with open("classmates.csv", mode="w", encoding='utf-8')
as w_file:
    file_writer = csv.writer(w_file, 'my_dialect')
    file_writer.writerow(["Ім'я", "Клас", "Вік"])
    file_writer.writerow(["Женя", "3", "10"]))
    file_writer.writerow(["Саша", "5", "12"]))
    file_writer.writerow(["Маша", "11", "18"])
```

В результаті отримаємо:

```
Ім'я:Клас:Вік
Женя:3:10
Саша:5:12
Маша:11:18
```

приклад

```
import csv
FILENAME = "users.csv"
users = [
    [1, "математика", 75],
    [2, "програмування", 85],
    [3, "мат аналіЗ", 34]
]

with open(FILENAME, "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(users)

with open(FILENAME, "a", newline="") as file:
    user = [4, "веб прог", 80]
    writer = csv.writer(file)
    writer.writerow(user)

d={}
with open(FILENAME, "r", newline="") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row[0], "-", row[1], "-", row[2])
        d[row[1]]=int(row[2])
print("d=",d)

#сформуємо список D1 зі словника та d
```

```
# та його сортуємо за ключом словника key=lambda x:x[0]
# якщо хочемо за значенням, пишіть key=lambda x:x[1]
# якщо хочемо у зворотному порядку, то пишіть key=lambda x:-x[1]
D1=sorted(d.items(),key=lambda x:-x[1]) # за ключами

print("D1=",D1) # друк списку

# відсортований список у словник D2 та його друк
D2={D1[i][0] : D1[i][1] for i in range(len(D1)) }
#друк таблицею D2
for kk, vv in D2.items():
    print(vv,'\t',kk)

ll=list(D2.items())
kkl = len (ll)-1
print("maximum score of", ll[0][1], "in the subject", ll[0][0])
print("minimum score of", ll[kkl][1], "in the subject",
ll[kkl][0])

user1=[]
nn=0
for kk, vv in D2.items():
    user1.append([nn,kk,vv])
    nn+=1

namerow=["№пп", "предмет", "оцінка"]
with open(FILENAME, "w", newline="") as file: #encoding='utf-8'
    writer=csv.writer(file, delimiter=",", lineterminator="\n")
    writer.writerow(namerow)
    writer.writerows(user1)
```