# 5. ЧИСЕЛЬНІ АЛГОРИТМИ. МАТРИЧНІ ОБЧИСЛЕННЯ.

## 5.1. Модуль Numpy.

*Numpy* — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами.

Оскільки Python — інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм який може бути виражений в основному як послідовність операцій над масивами і матрицями працює також швидко як еквівалентний код написаний на C.

Для того, щоб встановити NumPy, слід:

а) Linux:

***sudo pip3 install numpy***

б) Windows:

***pip3 install numpy***

## 5.2. Створення масиву.

Основним об'єктом *NumPy* є однорідний багатовимірний масив (в *numpy* називається *numpy.ndarray*). Це багатомірний 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 по кожній осі. Для матриці з n рядків і m стовпців, *shape* буде (n, m). Число елементів кортежу *shape* дорівнює *ndim*.

*ndarray.size* - кількість елементів масиву. Очевидно, дорівнює добутку всіх елементів атрибута *shape*.

*ndarray.dtype* - об'єкт, що описує тип елементів масиву.

*ndarray.itemsize* - розмір кожного елемента масиву в байтах.

*ndarray.data* - буфер, який містить фактичні елементи масиву.

У NumPy існує багато способів створити масив. Один з найбільш простих - створити масив із звичайних списків або кортежів Python, використовуючи функцію *numpy.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 в багатовимірні масиви. Тип елементів масиву залежить від типу елементів початкової послідовності (але також можна і перевизначити його в момент створення).

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

Також можна змінити тип в 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 для створення масивів. Зазвичай елементи масиву спочатку невідомі, а масив, в якому вони будуть зберігатися, вже потрібен. Тому є кілька функцій для того, щоб створювати масиви з якимось вихідним вмістом (за замовчуванням тип створюваного масиву - *float64*).

Функція *zeros()* створює масив з нулів, а функція *ones()* - масив з одиниць. Обидві функції приймають кортеж з розмірами, і 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 (двомірний 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 його заповнення. Початковий вміст формується випадково і залежить від стану пам'яті на момент створення масиву (тобто від того 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]])
```

Для створення послідовностей чисел, в NumPy є функція *arange()*, аналогічна 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])
```

Взагалі, при використанні *arange()* з аргументами типу *float*, складно бути впевненим в тому, скільки елементів буде отримано (через обмеження точності чисел з 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. ])
```

*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.]])
```

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

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

Якщо вам дійсно потрібно вивести весь масив, використовуйте функцію *numpy.set_printoptions*:

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

### 5.3. Функції для роботи з масивами.

Математичні операції над масивами виконуються поелементно. Створюється новий масив, який 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
array([       inf,  30.       ,  20.       ,  16.66666667])
<string>:1: RuntimeWarning: divide by zero encountered in true_divide
>>> a ** b
array([     1,     30,   1600, 125000])
>>> a % b
<string>:1: RuntimeWarning: divide by zero encountered in remainder
array([0, 0, 0, 2])
```

Для цього, звісно, масиви повинні 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 робити математичні операції між масивом і числом. В цьому випадку між заданим числом та кожним елементом проводиться задана маніпуляція.

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

NumPy також надає безліч математичних операцій для обробки масивів (Див.

https://docs.scipy.org/doc/numpy/reference/routines.math.html):

```
>>> 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])
```

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, дані операції застосовуються до масиву так, якби він був списком чисел, незалежно від його форми. Однак, вказавши параметр *axis*, можна застосувати операцію для зазначеної осі масиву:

```
>>> a.min(axis=0)  # Найменше число в кожному стовпці
array([1, 2, 3])
>>> a.min(axis=1)  # Найменше число в кожному рядку
array([1, 4])
```

### 5.4.Індекси, зрізи, ітерації.

Одномірні масиви здійснюють операції індексування, зрізу та ітерацій, що дуже схожі на звичайні операції зі списками і іншими послідовностями Python (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 i in a:
...     print(i ** (1/3))
...
0.0
1.0
2.0
2.0
2.0
2.0
2.0
7.0
8.0
9.0
```

У багатовимірних масивів на кожну вісь припадає 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]  # Друга строка, третій стовпець
23
>>> b[(2,3)]
23
>>> b[2][3]  # або так
23
>>> b[:,2]  # третій стовпець
array([ 2, 12, 22, 32, 42])
>>> b[:2]  # Перші дві строки
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13]])
>>> b[1:3, : : ]  # Друга та третя строки
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

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

```
>>> b[-1]  # Остання строка. Еквівалентно b[-1,:]
array([40, 41, 42, 43])
```

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

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

***x [1, 2, ...] еквівалентно x [1, 2,:,:,:],***
***x [..., 3] те ж 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, ...]  # теж саме, що і a[1, : , :] або a[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[... ,2]  # теж саме, що і a[: , : ,2]
array([[  2,  13],
       [102, 113]])
```

Ітерування багатомірних масивів починається з першої осі:

```
>>> for row in a:
...     print(row)
...
[[ 0  1  2]
 [10 12 13]]
[[100 101 102]
 [110 112 113]]
```

Проте, якщо потрібно перебрати поелементно весь масив так, якщо б він був одновимірним, для цього можна використовувати атрибут *flat*:

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

## 5.5. Функції модуля Numpy.

5.5.1. Маніпуляції з формою

Кожен масив має форму (*shape*), яка визначається числом елементів вздовж 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 в масиві в результаті функції *ravel()* відповідає звичайному "C-стилю", тобто, чим правіше індекс, тим він "швидше змінюється": за елементом a [0,0] буде йти a [0,1].

```
>>> 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 зі зміненою формою, в той час 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 з аргументів задається як -1, то він автоматично розраховується відповідно з іншими заданими:

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

5.5.2. Об'єднання масивів

Кілька масивів можуть бути об'єднані разом вздовж різних вісей за допомогою функцій *hstack* і *vstack*.

*hstack()* об'єднує масиви за першими вісями, *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()* об'єднує одномірні масиви в якості стовпців 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]])
```

5.5.3. Розбиття масиву

Використовуючи *hsplit()* можна розбити масив вздовж горизонтальної осі, вказавши або число повернутих масивів однакової форми, або номери стовпців, після яких масив розрізається:

```
>>> 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)  # Разбити 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 і четвертого стовпця
[array([[0, 1, 2], [6, 7, 8]]),
 array([[3], [9]]),
 array([[ 4,  5], [10, 11]])]
```

Функція *vsplit()* розбиває масив вздовж вертикальної осі, а *array_split()* дозволяє вказати осі, вздовж яких відбудеться розбиття.

# ПРАКТИЧНА РОБОТА

1) Створити вектор розміром 10, заповнений нулями, але п'ятий елемент дорівнює 1

```
import numpy as np
Z = np.zeros(10)
Z[4] = 1
print(Z)
```

2) Створити вектор зі значеннями від 10 до 49

```
import numpy as np
Z = np.arange(10,50)
print(Z)
```

3) Розвернути вектор (перший елемент стає останнім)

```
import numpy as np
Z = np.arange(50)
Z = Z[::-1]
print(Z)
```

4) Створити матрицю (двомірний масив) 3x3 зі значеннями від 0 до 8

```
import numpy as np
Z = np.arange(9).reshape(3,3)
print(Z)
```

5) Створити масив 10x10 з випадковими значеннями, знайти мінімум і максимум

```
import numpy as np
Z = np.random.random((10,10))
Zmin, Zmax = Z.min(), Z.max()
print(Zmin, Zmax)
```

6) Створити випадковий вектор розміром 30 і знайти середнє значення всіх елементів

```
import numpy as np
Z = np.random.random(30)
m = Z.mean()
print(m)
```

7) Створити 8x8 матрицю і заповнити її в шаховому порядку

```
import numpy as np
Z = np.zeros((8,8), dtype=int)
Z[1::2,::2] = 1
Z[::2,1::2] = 1
print(Z)
```

8) Дано масив 10x2 (точки в декартовій системі координат), перетворити в полярну

```
import numpy as np
Z = np.random.random((10,2))
X,Y = Z[:,0], Z[:,1]
R = np.hypot(X, Y)
T = np.arctan2(Y,X)
print(R)
print(T)
```

9) Замінити максимальний елемент на нуль

```
import numpy as np
Z = np.random.random(10)
Z[Z.argmax()] = 0
print(Z)
```

10) Знайти найближче до заданого значення число в заданому масиві

```
import numpy as np
Z = np.arange(100)
v = np.random.uniform(0,100)
index = (np.abs(Z-v)).argmin()
print(Z[index])
```