Тема 2. Пакет 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 a 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 a. a у радіанах;

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

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

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

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

numpy.square(a) - Повертає квадрат a.

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 "С-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 "С-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]])
```

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).

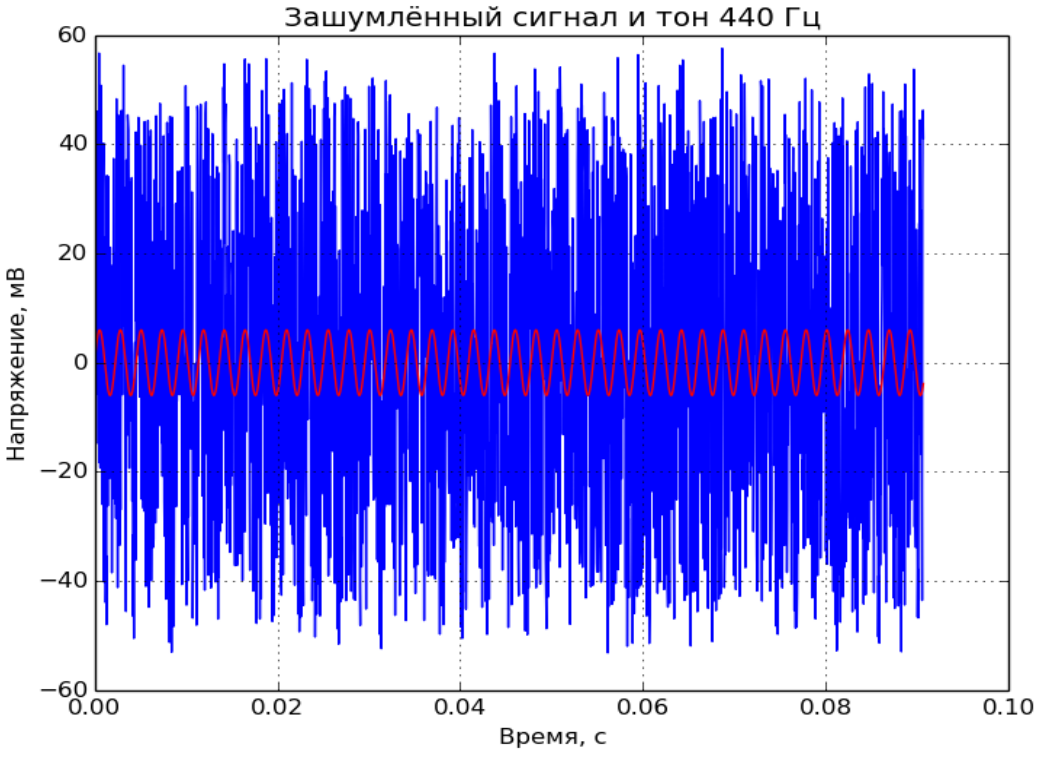
fftshift(x, axes=None) - Перетворює масив (з результатом ДПФ, від функцій fft *) так, щоб нульова частота була в центрі.

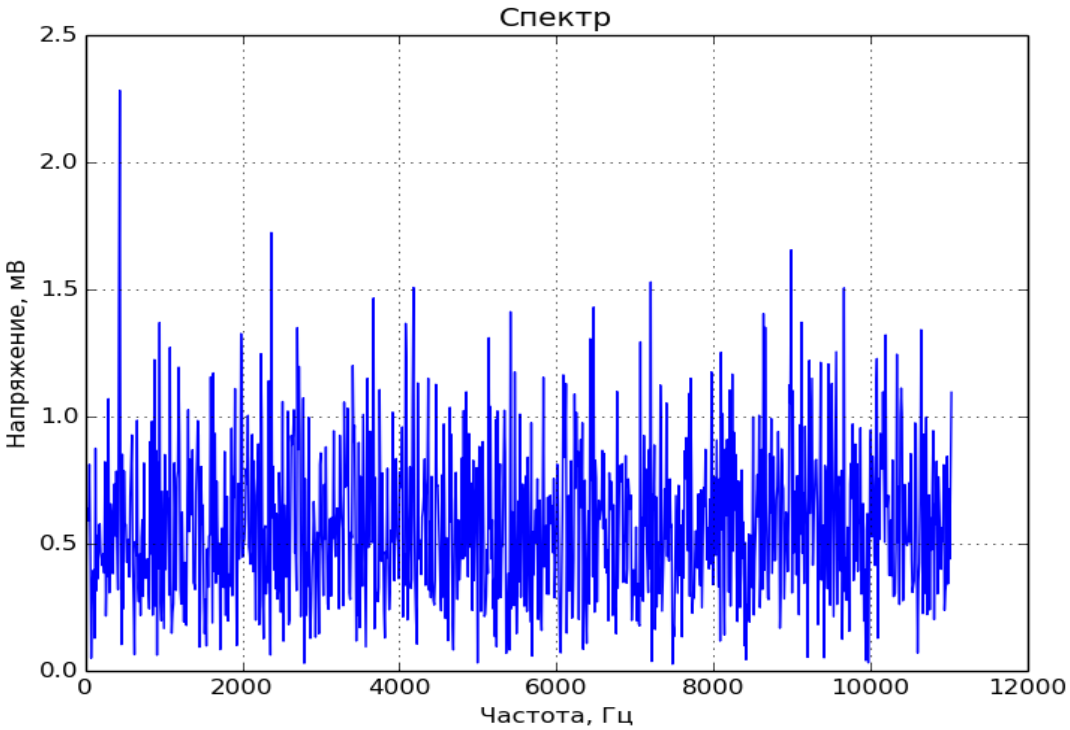
ifftshift(x, axes=None) - Здійснює зворотну операцію.

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

```
from numpy import array, arange, abs as np_abs
from numpy.fft import rfft, rfftfreq
from numpy.random import uniform
from math import sin, pi
import matplotlib.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)for t 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()
```




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