**NumPy**

NumPy - библиотека для научных вычислений (обработки набора чисел). Работает с многомерными массивами (матрицами).

**import numpy**

Для работы с функциями numpy нужно импортировать библиотеку

import numpy as np

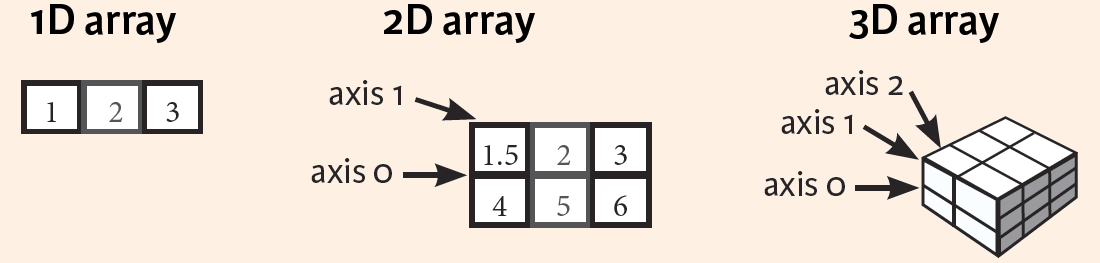
**Все примеры ниже приводятся в интерпретаторе питона. Он запускается из командной строки ipython**

DEA! Не забывайте в начале работы писать import numpy as np

**Термины**

numpy работает с однородными многомерными массивами (матрицами). Это матрица из элементов одинакового типа (обычно чисел).

В numpy размерность называют осями (**axes**). Количество осей - **ранг**.



Например, координаты точки в 3D пространстве [1, 2, 1] это м ранга 1, так как 1 ось. Это ось длины 3.

Массив ниже имеет ранг 2 (он двумерный). Первая ось длины 2 (в нем две строки), вторая ось длины 3 (в нем 3 столбца)

In [1]: import numpy as np

In [2]: a = np.array([

...: [1, 2, 3],

...: [4, 5, 6]

...: ])

In [3]: a

Out[3]:

array([[1, 2, 3],

[4, 5, 6]])

**ndarray**

Массив numpy реализован в классе **np.ndarray**

Характеристики массива a = np.array([[1, 2, 3], [4, 5, 6]]):

|  |  |  |
| --- | --- | --- |
| [**переменная**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=1&up=0#sorted_table) | [**значит**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=1&up=0#sorted_table) | [**равна для массива a**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=2&table=1&up=0#sorted_table) |
| a.shape | Размерность массива | (2, 3) |
| len(a) len(a[0]) | длина (первой оси) массива длина (второй оси) массива | 2 3 |
| a.ndim | количество осей | 2 |
| a.size | количество элементов (произведение всех чисел shape) | 6 |
| a.dtype | тип каждого элемента массива (см. ниже) | dtype('int32') |
| a.dtype.name | название типа 1 элемента массива | 'int32' |
| a.itemsize | размер 1 элемента в байтах (зависит от типа) (=32/8) | 4 |
| a.nbytes | размер всего массива в байтах (=4\*6) | 24 |
| af = a.astype(float) | преобразовать тип каждого элемента массива к float, получится новый массив, тип элементов старого массива не изменится |  |

In [1]: import numpy as np

In [2]: a = np.array([

...: [1, 2, 3],

...: [4, 5, 6]

...: ])

In [3]: a

Out[3]:

array([[1, 2, 3],

[4, 5, 6]])

In [4]: a.shape

Out[4]: (2, 3)

In [5]: a.ndim

Out[5]: 2

In [6]: a.dtype.name

Out[6]: 'int32'

In [7]: a.itemsize

Out[7]: 4

In [8]: a.size

Out[8]: 6

In [9]: type(a)

Out[9]: numpy.ndarray

In [10]: a.dtype

Out[10]: dtype('int32')

In [11]: a.nbytes

Out[11]: 24

In [12]: len(a)

Out[12]: 2

In [13]: len(a[0])

Out[13]: 3

**типы данных**

a.**astype**(*тип*) - сделать новый массив с типом элементов *тип*

|  |  |
| --- | --- |
| [**Тип**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=2&up=0#sorted_table) | [**Значит**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=2&up=0#sorted_table) |
| np.int64 | 64-битный целый тип со знаком |
| np.float32 | стандартное дробное число двойной точности |
| np.complex | комплексное число, представленное 128 floats |
| np.bool | Булевское (логическое) значение. Может быть True или False. |
| np.object | объект python (базовый тип в иерархии классов питона) |
| np.string\_ | строка фиксированной длины |
| np.unicode\_ | юникод-строка фиксированной длины |

Основные численные типы данных numpy (т.е. в таблице np.int и np.int32):

|  |  |  |
| --- | --- | --- |
| [**dtype**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=3&up=0#sorted_table) | [**варианты**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=3&up=0#sorted_table) | [**значит**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=2&table=3&up=0#sorted_table) |
| int | int8, int16, int32, int64 | целое число со знаком |
| uint | uint8, uint16, uint32, uint64 | целое число без знака (только положительные) |
| bool | bool | булевское (True или False) |
| float | float16, float32, float64, float128 | дробное |
| complex | complex64, complex128, complex256 | комлексное из дробных чисел |

In [16]: a.dtype

Out[16]: dtype('int32')

In [17]: b = a.astype(float)

In [18]: a.dtype

Out[18]: dtype('int32')

In [19]: b.dtype

Out[19]: dtype('float64')

**Создание массива**

Массив можно сделать:

* из другой структуры python (список, кортеж и других)
* специальными функциями создания (arange, ones, zeros и другие)
* из файла (специальный формат)
* из байт (из строки или буфера)
* специальными функциями (random)

**Создаем массив из объектов python функцией array**

Можно использовать list [], tuple () и их комбинации.

x = np.array([2,3,1,0])

x = np.array([2, 3, 1, 0])

x = np.array([[1,2.0],[0,0],(1+1j,3.)]) # tuple и list вместе

x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j], [ 1.+1.j, 3.+0.j]]) # комплексные числа

**Mesh-grid Arrays**

Эти массивы вы использовали для построения 3D графиков.

In [54]: x = np.array([-1, 0, 1]) # x это 1D массив

In [55]: y = np.array([-2, 0, 2]) # у это 1D массив

In [56]: X, Y = np.meshgrid(x, y)

In [57]: X

Out[57]: array([[-1, 0, 1], # 2D массив

[-1, 0, 1],

[-1, 0, 1]])

In [58]: Y # 2D массив

Out[58]: array([[-2, -2, -2],

[ 0, 0, 0],

[ 2, 2, 2]])

In [59]: Z = (X + Y) \*\* 2 # получим 2D массив z координат точек, таких что z = (x+y)\*\*2

In [60]: Z

Out[60]: array([[9, 4, 1],

[1, 0, 1],

[1, 4, 9]])

**Создание характерных массивов**

В любой функции можно еще указать dtype.

|  |  |  |
| --- | --- | --- |
| [**функция**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=4&up=0#sorted_table) | [**перечислением**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=4&up=0#sorted_table) | [**значит**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=2&table=4&up=0#sorted_table) |
| np.zeros((2, 3)) | array([[ 0., 0., 0.], [ 0., 0., 0.]]) | все 0 |
| np.ones((2,3,4)) | array([[ 1, 1, 1], [ 1, 1, 1]]) | все 1 |
| np.full((2,3), 7) | array([[ 7., 7., 7.], [ 7., 7., 7.]]) | матрица 2х3, везде 7 |
| np.identity(2) | array([[ 1., 0.], [ 0., 1.]]) | единичная матрица размера 2 |
| np.eye(2) | array([[ 1., 0.], [ 0., 1.]]) | единичная матрица размера 2 (единицы только на одной диагонали) |
| np.eye(2, 3, k=1) | array([[ 0., 1., 0.], [ 0., 0., 1.]]) | матрица 2х3, все 0, кроме 1 на диагонали номер k=1 |
| np.empty((2,3)) |  | матрица 2х3, значения не инициализированы (могут быть 0, а могут быть любыми |
| np.arange(10) | array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) | [0, 10) с шагом 1 |
| np.arange(2, 10, dtype=np.float) | array([ 2., 3., 4., 5., 6., 7., 8., 9.]) | [2, 10) с шагом 1 |
| np.arange(2, 3, 0.1) | array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9]) | [2, 3) с шагом 0.1 |
| np.linspace(1., 4., 6) | array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ]) | [1, 4] из 6 точек |
| np.logspace(0, 2, 5) | array([ 1. , 3.16227766, 10. , 31.6227766 , 100.]) | массив из 5 точек от 10\*\*0=1 до 10\*\*2=100 |

In [12]: f = np.indices((3,4))

In [13]: f

Out[13]:

array([[[0, 0, 0, 0],

[1, 1, 1, 1],

[2, 2, 2, 2]],

[[0, 1, 2, 3],

[0, 1, 2, 3],

[0, 1, 2, 3]]])

**Массив случайных чисел**

**равномерное распределение, float**

**np.random.random\_sample**(*shape*) возвращает **равномерно** распределенные случайные числа на [0, 1)

>>> np.random.random\_sample() # одно число

0.47108547995356098

>>> type(np.random.random\_sample()) # оно типа float

<type 'float'>

>>> np.random.random\_sample((5,)) # вектор длины 5 случайных чисел

array([ 0.30220482, 0.86820401, 0.1654503 , 0.11659149, 0.54323428])

Для равномерного распределения на [a, b) где a < b есть формула

(b - a) \* random\_sample() + a

Для массива 2х3 с числами, равномерно распределенными на [-5, 7)

r = (7 - (-5)) \* np.random.random\_sample((2, 3)) + (-5)

array([[ 1.94852693, 6.99283723, 1.8778662 ],

[ 0.18915957, -3.69433527, 0.20130357]])

**нормальное распределение, float**

**numpy.random.standard\_normal**(size=None)  
Массив чисел с нормальным распределением (mean=0, stdev=1).

s = np.random.standard\_normal((2, 3))

array([[ 0.23295507, -0.43122137, 0.00879516],

[-0.05973499, 1.32533893, -0.375821 ]])

1 число

s = np.random.standard\_normal()

-0.8945336142780018

**равномерное распределение, int**

**numpy.random.randint**(low, high=None, size=None, dtype='l')  
Равномерно распределенные случайные целые числа

* на [low, hight), если оба определены
* на [0, low) если hight не определено
* если size нет, то вернет 1 число

np.random.randint(15) # 1 целое число из [0, 15)

14

np.random.randint(2, size=10) # массив из 10 целых чисел из [0, 2)

array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])

np.random.randint(5, 15, size=10) # массив из 10 целых чисел из [5, 15)

array([13, 11, 8, 5, 5, 11, 8, 9, 12, 8])

np.random.randint(5, size=(2, 4)) # массив 2х4 из целых чисел из [0, 5)

array([[4, 0, 2, 1],

[3, 2, 2, 0]])

**Сделаем массив по формуле**

In [76]: def f(n,m):

res = n + 10 \* m

return res

In [77]: A = np.fromfunction(f, (6, 6), dtype=int)

In [78]: A

Out[78]: array([[ 0, 1, 2, 3, 4, 5],

[10, 11, 12, 13, 14, 15],

[20, 21, 22, 23, 24, 25],

[30, 31, 32, 33, 34, 35],

[40, 41, 42, 43, 44, 45],

[50, 51, 52, 53, 54, 55]])

**Комплексные числа**

Из массива комплексных чисел можно получить ссылку на массив действительных **real** и мнимых **imag** частей.

DEA! При изменении массива действительной или мнимой части, изменяется массив исходных чисел.

In [28]: d = np.array([1, 2, 3], dtype=complex)

In [29]: d

Out[29]: array([ 1.+0.j, 2.+0.j, 3.+0.j])

In [30]: re = d.real

In [31]: re

Out[31]: array([ 1., 2., 3.])

In [32]: im = d.imag

In [32]: im

Out[32]: array([ 0., 0., 0.])

In [33]: re[0] = 10 # изменим элементы массивов re и im

In [34]: im[1] = -13

In [35]: d # посмотрим массив комплексных чисел

Out[35]: array([ 10.+0.j, 2.-13.j, 3.+0.j]) # она тоже изменилась

**Чтение из файла и запись в файл**

**во внутреннем формате numpy**

np.save('my\_array', a)

np.savez('array.npz', a, b)

np.load('my\_array.npy')

**Из текстового или csv файла**

np.loadtxt("myfile.txt")

np.genfromtxt("my\_file.csv", delimiter=',')

np.savetxt("myarray.txt", a, delimiter=" ")

**Доступ к элементам и срезы**

**Одномерный массив**

|  |  |
| --- | --- |
| [**Выражение**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=6&up=0#sorted_table) | [**Значит**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=6&up=0#sorted_table) |
| a[m] | элемент с номером m, где m целое (нумерация начинается с 0). |
| a[-m] | элемент с номером m с конца. Последний элемент a[-1] |
| a[m:n] | элементы с номерами от m (включая) до n (НЕ включая) |
| a[:] или a[0:-1] | все элементы по данной оси |
| a[:n] | элементы с номерами от 0 до n-1 (НЕ включая n) |
| a[m:] или a[m:-1] | элементы с номерами от m до конца |
| a[m:n:p] | элементы с номерами от m (включая) до n (НЕ включая) с шагом p. |
| a[::-1] | все элементы в обратном порядке |

In [66]: a = np.arange(0, 11)

In [67]: a

Out[67]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

In [68]: a[0] # первый элемент

Out[68]: 0

In [69]: a[-1] # последний элемент

Out[69]: 10

In [70]: a[4] # пятый элемент (с номером 4)

Out[70]: 4

In [71]: a[1:-1]

Out[71]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [72]: a[1:-1:2]

Out[72]: array([1, 3, 5, 7, 9])

In [73]: a[:5]

Out[73]: array([0, 1, 2, 3, 4])

In [74]: a[-5:]

Out[74]: array([6, 7, 8, 9, 10])

In [75]: a[::-2] # с конца через 1

Out[75]: array([10, 8, 6, 4, 2, 0])

**Многомерные массивы**

К каждой оси применимы правила для одномерных массивов.

In [76]: def f(n,m):

res = n + 10 \* m

return res

In [77]: A = np.fromfunction(f, (6, 6), dtype=int)

In [78]: A

Out[78]: array([[ 0, 1, 2, 3, 4, 5],

[10, 11, 12, 13, 14, 15],

[20, 21, 22, 23, 24, 25],

[30, 31, 32, 33, 34, 35],

[40, 41, 42, 43, 44, 45],

[50, 51, 52, 53, 54, 55]])

# выделим колонки или строки, используя срезы и индексы

In [79]: A[:, 1] # вторая колонка (колонка с индексом 1)

Out[79]: array([ 1, 11, 21, 31, 41, 51])

In [80]: A[1, :] # вторая строка (строка с индексом 1)

Out[80]: array([10, 11, 12, 13, 14, 15])

# срезы в строках и столбцах

In [81]: A[:3, :3] # верхняя левая часть матрицы

Out[81]: array([[ 0, 1, 2],

[10, 11, 12],

[20, 21, 22]])

In [82]: A[3:, :3] # нижняя левая часть матрицы

Out[82]: array([[30, 31, 32],

[40, 41, 42],

[50, 51, 52]])

# используем шаг

In [83]: A[::2, ::2] # каждый второй элемент, начиная с 0,0

Out[83]: array([[ 0, 2, 4],

[20, 22, 24],

[40, 42, 44]])

In [84]: A[1::2, 1::3] # каждая вторая строка, начиная с 1; каждый третий столбец, начиная с 1

Out[84]: array([[11, 14],

[31, 34],

[51, 54]])

**copy**

Срез выбирает часть массива. Можно **изменить** часть массива через срез.

In [76]: def f(n,m):

res = n + 10 \* m

return res

In [77]: A = np.fromfunction(f, (6, 6), dtype=int)

In [78]: A

Out[78]: array([[ 0, 1, 2, 3, 4, 5],

[10, 11, 12, 13, 14, 15],

[20, 21, 22, 23, 24, 25],

[30, 31, 32, 33, 34, 35],

[40, 41, 42, 43, 44, 45],

[50, 51, 52, 53, 54, 55]])

In [79]: A[0, 2:5] = 100

In [80]: A

Out[80]: array([[ 0, 1, 100, 100, 100, 100], # стали 100

[10, 11, 12, 13, 14, 15],

[20, 21, 22, 23, 24, 25],

[30, 31, 32, 33, 34, 35],

[40, 41, 42, 43, 44, 45],

[50, 51, 52, 53, 54, 55]])

In [85]: B = A[1:5, 1:5] # B это часть А

In [86]: B

Out[86]: array([[11, 12, 13, 14],

[21, 22, 23, 24],

[31, 32, 33, 34],

[41, 42, 43, 44]])

In [87]: B[:, :] = 0 # меняем В. Изменилась А.

In [88]: A

Out[88]: array([[ 0, 1, 100, 100, 100, 100],

[10, 0, 0, 0, 0, 15],

[20, 0, 0, 0, 0, 25],

[30, 0, 0, 0, 0, 35],

[40, 0, 0, 0, 0, 45],

[50, 51, 52, 53, 54, 55]])

Из массива А сделали массив В. В элементы массива В записали новые значения. Но это изменило сам массив А. То есть оба массива расположены в той же памяти компьютера.

Массив В смотрит на те же данные, что и массив А.

Если нужно, чтобы в массиве В была копия данных, нужно использовать функцию **copy**

In [89]: C = B[1:3, 1:3].copy()

In [90]: C

Out[90]: array([[0, 0],

[0, 0]])

In [91]: C[:, :] = 1 # не влияет на В или А, так как С только копия B[1:3, 1:3]

In [92]: C

Out[92]: array([[1, 1], # C изменился

[1, 1]])

In [93]: B # B НЕ изменился

Out[93]: array([[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]])

**Необычное индексирование и True и False в виде индексов**

Индексы можно перечислить через список или другой numpy массив

In [94]: A = np.linspace(0, 1, 11)

Out[94]: array([ 0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])

In [95]: A[np.array([0, 2, 4])] # индексы заданы в массиве

Out[95]: array([ 0. , 0.2, 0.4])

In [96]: A[[0, 2, 4]] # индексы заданы через список

Out[96]: array([ 0. , 0.2, 0.4])

Можно узнать, какие элементы подходят и сделать из них срез.

# какие элементы массива А имеют ЗНАЧЕНИЕ больше 0.5

In [97]: A > 0.5 # получили новый массив с элементами True (да) или False (нет)

Out[97]: array([False, False, False, False, False, False, True, True, True, True, True], dtype=bool)

In [98]: A[A > 0.5] # сделали срез массива по условию "значение элемента массива больше 0.5"

Out[98]: array([ 0.6, 0.7, 0.8, 0.9, 1. ])

**Изменяем количество строк матрицы. Данные не меняются**

**ravel, reshape, T**

Будем изменять этот массив:

In [1]: import numpy as np

In [2]: a = np.array([

...: [1, 2, 3],

...: [4, 5, 6]

...: ])

In [3]: a

Out[3]:

array([[1, 2, 3],

[4, 5, 6]])

In [4]: a.shape

Out[4]: (2, 3)

In [8]: a.ravel() # в 1 строку

Out[8]: array([1, 2, 3, 4, 5, 6])

In [11]: a.reshape(3,2) # из 2х3 получили 3х2 (вытянули в 1 строку и порезали снова)

Out[11]:

array([[1, 2],

[3, 4],

[5, 6]])

In [12]: a.reshape(3, -1) # -1 значит "столбцов на 1 меньше"

Out[12]:

array([[1, 2],

[3, 4],

[5, 6]])

In [13]: a.T # транспонируем (поменяем строки и столбцы местами) матрицу

Out[13]:

array([[1, 4],

[2, 5],

[3, 6]])

In [14]: np.transpose(a) # транспонируем матрицу другой функцией

Out[14]:

array([[1, 4],

[2, 5],

[3, 6]])

TODO: np.expand\_dims + np.newaxis

**vstack, hstack, вектор строкой и столбцом, column\_stack**

In [20]: b = np.array([1, 2, 3, 4]) # вектор в строку

Out[20]: array([1, 2, 3, 4])

In [21]: v = b[:, np.newaxis] # вектор в столбец

Out[21]:

array([[1],

[2],

[3],

[4]])

In [22]: bm = np.vstack((b, b, b)) # поставили массивы один на другой, вертикально

Out[22]:

array([[1, 2, 3, 4],

[1, 2, 3, 4],

[1, 2, 3, 4]])

In [24]: bs = np.hstack((b, b, b)) # поставили массивы рядом, горизонтально

Out[24]: array([1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4])

In [25]: np.column\_stack((b, b)) # сделаем из b вектор столбец и поставим рядом столбцы

Out[25]:

array([[1, 1],

[2, 2],

[3, 3],

[4, 4]])

In [39]: a1 = np.array([[1, 2], [3, 4]])

Out[39]:

array([[1, 2],

[3, 4]])

In [41]: a2 = np.array([[5, 6]])

Out[41]: array([[5, 6]])

In [44]: np.concatenate((a1, a2), axis= 0) # рядом по оси 0 (строки)

Out[44]:

array([[1, 2],

[3, 4],

[5, 6]])

In [45]: a2.T # a2 = [[5, 6]] - матрица из строки, транспонированная - матрица из столбца

Out[45]:

array([[5],

[6]])

In [46]: np.concatenate((a1, a2.T), axis= 1) # рядом по оси 1 (столбцы)

Out[46]:

array([[1, 2, 5],

[3, 4, 6]])

**split**

Функция **np.hsplit** - матрицы стоят рядом

Функция **np.vsplit** - матрицы стоят одна на другой

**делим НА 3 части**

Функция **np.hsplit(массив, n)** - разделить массив на 3 массива

In [62]: a = np.arange(1, 25)

Out[62]:

array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,

18, 19, 20, 21, 22, 23, 24])

In [64]: a1 = a.reshape((2, 12))

Out[64]:

array([[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])

In [75]: np.hsplit(a1, 3) # делим матрицу НА 3 части, матрицы рядом по горизонтали

Out[75]:

[array([[ 1, 2, 3, 4],

[13, 14, 15, 16]]), array([[ 5, 6, 7, 8],

[17, 18, 19, 20]]), array([[ 9, 10, 11, 12],

[21, 22, 23, 24]])]

In [76]: s1 = np.hsplit(a1, 3) # получаем список массивов

In [77]: s1[0] # первый массив

Out[77]:

array([[ 1, 2, 3, 4],

[13, 14, 15, 16]])

In [78]: s1[1] # следующий массив

Out[78]:

array([[ 5, 6, 7, 8],

[17, 18, 19, 20]])

In [79]: s1[2] # последний массив

Out[79]:

array([[ 9, 10, 11, 12],

[21, 22, 23, 24]])

**делим ПОСЛЕ столбцов номер (n1, n2, n3 и далее)**

Функция **np.hsplit(массив, (3, 4))** - разделить массив ПОСЛЕ столбца 3 и ПОСЛЕ столбца 4

In [62]: a = np.arange(1, 25)

Out[62]:

array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,

18, 19, 20, 21, 22, 23, 24])

In [64]: a1 = a.reshape((2, 12))

Out[64]:

array([[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])

In [80]: w1 = np.hsplit(a1, (3, 4))

In [81]: w1

Out[81]:

[array([[ 1, 2, 3],

[13, 14, 15]]), array([[ 4],

[16]]), array([[ 5, 6, 7, 8, 9, 10, 11, 12],

[17, 18, 19, 20, 21, 22, 23, 24]])]

In [82]: w1[0]

Out[82]:

array([[ 1, 2, 3],

[13, 14, 15]])

In [83]: w1[1]

Out[83]:

array([[ 4],

[16]])

In [84]: w1[2]

Out[84]:

array([[ 5, 6, 7, 8, 9, 10, 11, 12],

[17, 18, 19, 20, 21, 22, 23, 24]])

**Арифметические операции**

Операция - создаем новый массив.

Короткая форма - изменяем массив а.

|  |  |
| --- | --- |
| [**Операция**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=7&up=0#sorted_table) | [**Короткая форма**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=7&up=0#sorted_table) |
| a+b | a += b |
| a-b | a -= b |
| a\*2 | a \*= 2 |
| a\*b | a \*= b |
| a/2 | a /= 2 |
| a/b | a /= b |
| a\*\*2 | a \*\*= 2 |
| 2\*\*a | нет краткой формы |

In [85]: a = np.array([[1, 2, 3], [4, 5, 6]])

In [86]: a

Out[86]:

array([[1, 2, 3],

[4, 5, 6]])

In [87]: a\*2

Out[87]:

array([[ 2, 4, 6],

[ 8, 10, 12]])

In [88]: b = np.array([[11, 22, 33], [44, 55, 66]])

In [89]: b

Out[89]:

array([[11, 22, 33],

[44, 55, 66]])

In [90]: a+b

Out[90]:

array([[12, 24, 36],

[48, 60, 72]])

In [91]: b-a

Out[91]:

array([[10, 20, 30],

[40, 50, 60]])

In [92]: a\*b

Out[92]:

array([[ 11, 44, 99],

[176, 275, 396]])

In [93]: 2\*a

Out[93]:

array([[ 2, 4, 6],

[ 8, 10, 12]])

In [94]: 2\*\*a

Out[94]:

array([[ 2, 4, 8],

[16, 32, 64]], dtype=int32)

In [95]: b/a

Out[95]:

array([[ 11., 11., 11.],

[ 11., 11., 11.]])

In [96]: b/10

Out[96]:

array([[ 1.1, 2.2, 3.3],

[ 4.4, 5.5, 6.6]])

**Введение в Numpy**

Numpy (numeric python) - библиотека работы с числами.

В ней задаются наборы точек для построения графиков.

**Задаем набор точек**

x = np.arange([1, 2, 3, 5, 7, 10, 15])

x = np.arange(-10, 10.01, 0.01) # от -10 до 10.01 с шагом 0.01 (>1000 точек)

x = np.linspace(-5, 2, 100) # от -5 до 2 сделать 100 точек

**Константы и функции numpy**

|  |  |
| --- | --- |
| [**numpy**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=8&up=0#sorted_table) | [**Математика**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=8&up=0#sorted_table) |
| np.pi | Число pi |
| np.e | Число e |
| np.[cos](http://docs.scipy.org/doc/numpy/reference/generated/numpy.cos.html) | Косинус |
| np.[sin](http://docs.scipy.org/doc/numpy/reference/generated/numpy.sin.html) | Синус |
| np.[tan](http://docs.scipy.org/doc/numpy/reference/generated/numpy.tan.html) | Тангенс |
| np.[acos](http://docs.scipy.org/doc/numpy/reference/generated/numpy.acos.html) | Арккосинус |
| np.[asin](http://docs.scipy.org/doc/numpy/reference/generated/numpy.asin.html) | Арксинус |
| np.[atan](http://docs.scipy.org/doc/numpy/reference/generated/numpy.atan.html) | Арктангенс |
| np.[exp](http://docs.scipy.org/doc/numpy/reference/generated/numpy.exp.html) | Экспонента |
| np.[log](http://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html) | Логарифм натуральный |
| np.[log2](http://docs.scipy.org/doc/numpy/reference/generated/numpy.log2.html) | Логарифм по основанию 2 |
| np.[log10](http://docs.scipy.org/doc/numpy/reference/generated/numpy.log10.html) | Логарифм десятичный |
| np.[sqrt](http://docs.scipy.org/doc/numpy/reference/generated/numpy.sqrt.html) | x1/2 |

|  |  |
| --- | --- |
| [**функции numpy**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=9&up=0#sorted_table) | [**Описание**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=9&up=0#sorted_table) |
| np.add, np.subtract, np.multiply, np.divide | +, -, \*, / |
| np.power(a, b) | a[i,j]\*\*b[i,j] |
| np.remainder(a, b) | остаток от деления a[i,j]%b[i,j] |
| np.reciprocal(a) | 1/a[i,j] |
| np.real, np.imag, np.conj | действительная часть; мнимая часть; a+bj заменяется на a-bj |
| np.sign | знак, 1, -1 или 0 |
| np.abs | модуль |
| np.floor, np.ceil, np.rint | преобразуем к целым числам |
| np.round | округление с указанной точностью |

**Агрегации**

|  |  |
| --- | --- |
| [**функции numpy**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=10&up=0#sorted_table) | [**Описание**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=10&up=0#sorted_table) |
| np.mean | среднее |
| np.std | стандартное отклонение |
| np.var | дисперсия |
| np.sum | сумма всех элементов |
| np.prod | произведение всех элементов |
| np.cumsum | сумма всех элементов по указанной оси |
| np.cumprod | произведение всех элементов по указанной оси |
| np.min, np.max | минимальное и максимальное число в массиве |
| np.argmin, np.argmax | индекс минимального и максимального числа в массиве |
| np.all | True если все элементы в массиве НЕ 0 |
| np.any | True если хоть один элемент в массиве не 0 |

In [180]: data = np.arange(1,10).reshape(3,3)

In [181]: data

Out[181]: array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

In [182]: data.sum() # сумма по всем элементам массива

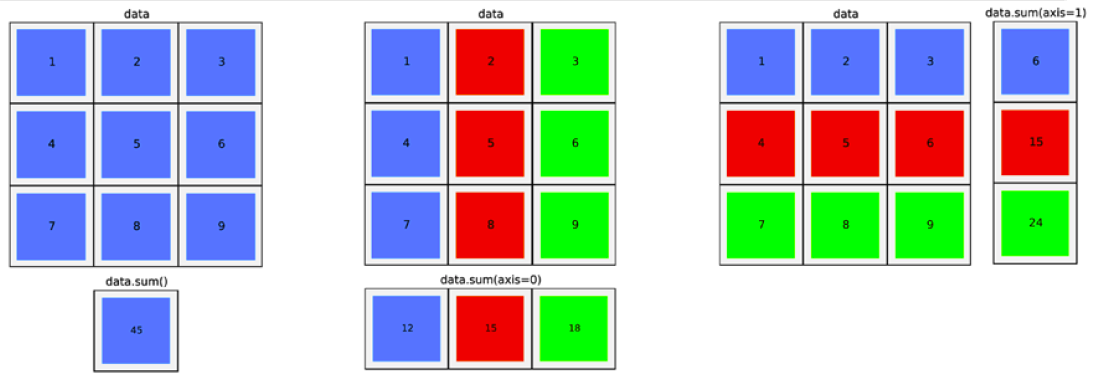
Out[182]: 45

In [183]: data.sum(axis=0) # сумма по столбцам

Out[183]: array([12, 15, 18])

In [184]: data.sum(axis=1) # сумма по строкам

Out[184]: array([ 6, 15, 24])



**sort - сортировка массива**

>>> a = np.array([[1,4],[3,1]])

>>> np.sort(a) # сортируем по последней оси (самые внутренние массивы)

array([[1, 4],

[1, 3]])

>>> np.sort(a, axis=None) # сортируем массив, вытянутый в 1 строку

array([1, 1, 3, 4])

>>> np.sort(a, axis=0) # сортируем по оси 0

array([[1, 1],

[3, 4]])

**Сравнение массивов**

>>> a == b # сравниваем по элементам два массива

array([[False, True, True],

[False, False, False]], dtype=bool)

>>> a < 2 # сравниваем каждый элемент массива с 2

array([True, False, False], dtype=bool)

>>> np.array\_equal(a, b) # сравниваем два массива, возвращает True или False

Если нужно сравнивать по элементам:

In [185]: a = np.array([1, 2, 3, 4])

In [186]: b = np.array([4, 3, 2, 1])

In [187]: a < b

Out[187]: array([ True, True, False, False], dtype=bool)

In [188]: np.all(a < b)

Out[188]: False

In [189]: np.any(a < b)

Out[189]: True

In [190]: if np.all(a < b):

...: print("All elements in a are smaller than their corresponding element in b")

...: elif np.any(a < b):

...: print("Some elements in a are smaller than their corresponding elemment in b")

...: else:

...: print("All elements in b are smaller than their corresponding element in a")

Some elements in a are smaller than their corresponding elemment in b

**Как заставить свою функцию работать с векторами**

In [165]: def heaviside(x): # обычная функция, работает с числами

...: return 1 if x > 0 else 0

In [166]: heaviside(-1)

Out[166]: 0

In [167]: heaviside(1.5)

Out[167]: 1

In [168]: x = np.linspace(-5, 5, 11) # массив numpy

In [169]: heaviside(x) # передадим в эту функцию массив

...

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any()

or a.all()

# Как сделать так, чтобы эта функция применялась к каждому элементу массива?

In [170]: vheaviside = np.vectorize(heaviside) # сделаем новую функцию vheaviside

In [171]: vheaviside(x) # vheaviside работает с массивами

Out[171]: array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

**Операции над матрицами и векторами. Линейная алгебра**

Numpy используют для представления векторов, матриц и тензоров. Для них определены специальные функции. Эти функции вы учили в линейной алгебре.

**Умножение по элементам \* (уже знаем)**

Пусть а и b - 2D массивы, обращаемся их элементам a[i,j] и b[i,j]

c = a \* b

Для массивов а и b получаем массив с, где c[i.j] = a[i,j] \* b[i,j]

In [85]: a = np.array([[1, 2, 3], [4, 5, 6]])

Out[85]:

array([[1, 2, 3],

[4, 5, 6]])

In [88]: b = np.array([[11, 22, 33], [44, 55, 66]])

Out[88]:

array([[11, 22, 33],

[44, 55, 66]])

In [90]: a+b

Out[90]:

array([[12, 24, 36],

[48, 60, 72]])

In [91]: b-a

Out[91]:

array([[10, 20, 30],

[40, 50, 60]])

In [92]: a\*b

Out[92]:

array([[ 11, 44, 99],

[176, 275, 396]])

**Произведение матриц np.dot**

Произведением матрицы A с элементами a[i,j] размерности mxn на матрицу B с элементами b[i,j] размерности nxk называется матрица С с элементам c[i,j] размера mxk, такое что c[i,j] = a[i,1]\*b[1,j] + a[i,2]\*b[2,j]+..+a[i,n]\*b[n,j]

DEA! Будем дальше обозначать матричное произведение в тексте как C = A.B (как пишут математики, а не программисты)

В numpy для матричного произведения используется функция **np.dot**

In [223]: A = np.arange(1, 7).reshape(2, 3)

In [224]: A

Out[224]: array([[1, 2, 3],

[4, 5, 6]])

In [225]: B = np.arange(1, 7).reshape(3, 2)

In [226]: B

Out[226]: array([[1, 2],

[3, 4],

[5, 6]])

In [227]: np.dot(A, B) # A.B

Out[227]: array([[22, 28],

[49, 64]])

In [228]: np.dot(B, A) # B.A

Out[228]: array([[ 9, 12, 15],

[19, 26, 33],

[29, 40, 51]])

**vdot (скалярное произведение)**

Для 1D массивов (векторов) получаем их скалярное произведение.

In [97]: a = np.array([1, 2, 3])

In [98]: b = np.array([4, 5, 6])

In [99]: np.vdot(a,b)

Out[99]: 32

In [100]: 1\*4 + 2\*5 + 3\*6

Out[100]: 32

DEA! Для многомерных массивов, их сначала вытягивают в строку (делают из них 1D векторы). Потом эти векторы умножают скалярно.

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

>>> b = np.array([[4, 1], [2, 2]])

>>> np.vdot(a, b)

30

>>> np.vdot(b, a)

30

>>> 1\*4 + 4\*1 + 5\*2 + 6\*2

30

**Транспонированная матрица**

Для массива a транспонированная матрица вычисляется как **a.T**

**Единичная матрица**

В единичной матрице на главной диагонали 1, а на остальных местах 0.

Создаем единичную матрицу размера 2х2 функцией e = **np.eye(2)**

**Обратная матрица**

Определение (математика): E - единичная матрица. Квадратная матрица В называется обратной по отношению к матрице А того же размера, если A.B = B.A = E

Обратная матрица для матрицы А в линейной алгебре обозначается как А-1

**numpy.linalg - библиотека функций линейной алгебры из библиотеки numpy**

Для того, чтобы функции inv, det, solve из библиотеки numpy.linalg работали в вашей программе, нужно сделать import

import numpy as np

a = np.array([[1, 2], [3,4]])

print(a)

ai = np.linalg.inv(a) # пишем полное имя функции с указанием библиотек, где она находится

print(ai)

Короче:

import numpy as np

import numpy.linalg as lg # запоминаем библиотеку numpy.linalg под коротким именем lg

a = np.array([[1, 2], [3,4]])

print(a)

ai = lg.inv(a) # обращаемся к функции по короткому имени библиотеки

print(ai)

Совсем коротко: перечисляем имена функций, которые будем использовать из этого пакета

import numpy as np

from numpy.linalg import inv, det, solve

a = np.array([[1, 2], [3,4]])

print(a)

ai = inv(a)

print(ai)

**Обратные матрицы в решении линейных уравнений**

Пусть у нас есть уравнение A.X = B

Домножим с ОДИНАКОВОЙ стороны правую и левую сторону на A-1

A-1.A.X = A-1.B

E.X = A-1.B

X = A-1.B

**Решение линейных уравнений linalg.solve(a,b)**

Решим систему *линейных* уравнений 3 \* x0 + x1 = 9 и x0 + 2 \* x1 = 8

>>> a = np.array([[3,1], [1,2]])

>>> b = np.array([9,8])

>>> x = np.linalg.solve(a, b)

>>> x

array([ 2., 3.])

Проверим решение:

>>> np.allclose(np.dot(a, x), b)

True

**Детерминант матрицы**

Для вычисления детерминанта матрицы используют функцию **numpy.linalg.det**

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

>>> np.linalg.det(a)

-2.0

**Ранг матрицы (в смысле линейной алгебры)**

Ранг матрицы - количество линейно независимых столбцов. (Их нельзя получить умножая на числа и складывая другие столбцы).

По теореме о базисном миноре, ранг равен и количеству линейно независимых строк.

В numpy ранг матрицы вычисляется функцией **numpy.linalg.matrix\_rank**(a)

In [103]: a = ([[1, 2, 3], [4, 5, 6]])

In [104]: np.linalg.matrix\_rank(a)

Out[104]: 2

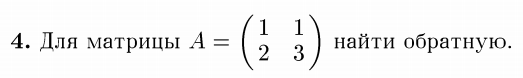
In [105]: b = ([[1, 2, 3], [4, 5, 6], [6, 9, 12]]) # последняя строка = первая \* 2 + вторая строка (линейно зависимая)

In [106]: np.linalg.matrix\_rank(b)

Out[106]: 2

**Решение задачи в отдельном файле**

DEA! Решения оформлять 1 задача - 1 файл.



import numpy as np

from numpy.linalg import inv, det, solve

a = np.array([[1, 1], [2, 3]])

print(a)

ai = inv(a)

print(ai)

**Задачи**

* Определить единичную матрицу 5х5
* Сохранить этот текст в файл. Прочитать матрицу из файла.
  + найдите для этой матрицы сумму всех элементов, максимальный и минимальный элемент (число)

3,4,17,-3

5,11,-1,6

0,2,-5,8

* [Учебник по линейной алгебре](http://judge.mipt.ru/tasks/book/Butuzov-VF-Krutitskaya-NCh-Shishkin-AA-Lineinaya-algebra-v-voprosah-i-zadachah.pdf)
  + страница 11, номера:
    - 5 (а, б, г, д); в задаче 5а матрицу В создать через функцию zeros и присвоение потом 1 нужному элементу.
    - 9 (а, б, г, д)
    - 8 (a, в, г, д, е), - решите с помощью обратных матриц и умножения; если решения не существует, вычислите ранг и детерминант матриц
  + страница 21, номера 17 и 19 - по вариантам

|  |  |  |  |
| --- | --- | --- | --- |
| [**вариант**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=0&table=11&up=0#sorted_table) | [**17a**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=1&table=11&up=0#sorted_table) | [**17б**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=2&table=11&up=0#sorted_table) | [**17в**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpy?sortcol=3&table=11&up=0#sorted_table) |
| **19a** | 1 | 2 | 3 |
| **19б** | 4 | 5 | 6 |
| **19в** | 7 | 8 | 9 |
| **19г** | 10 | 11 | 12 |

* [ответы для самопроверки](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonNumpySolutions)

**Взаимодействие с кодом на С или Fortran**

Можно использовать функции, написанные на других языках.

**Хранение массива в памяти**

Хранить многомерные массивы одним куском можно в стиле С (строка за строкой) или Fortran (столбец за столбцом). Неправильный порядок перебора элементов массива может тормозить. Порядок хранения можно задать при создании, указав функции **np.array** аргумент либо **order=C** (как в С), либо **order=F** (как в Fortran).

**Документация и учебники**

* <https://docs.scipy.org/doc/numpy/user/index.html> - Numpy (+tutorial)
* <https://docs.scipy.org/doc/numpy/reference/index.html#reference> - Numpy документация
* <http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html> - Numpy tutorial by Nicolas P. Rougier
* <http://www.labri.fr/perso/nrougier/teaching/numpy.100/index.html> - 100 упражнений по numpy
* [Numpy\_Python\_Cheat\_Sheet.pdf](http://acm.mipt.ru/twiki/pub/Cintro/PythonNumpy/Numpy_Python_Cheat_Sheet.pdf): шпаргалка на 1 лист А4
* [matrix-multiply-a.svg](http://acm.mipt.ru/twiki/pub/Cintro/PythonNumpy/matrix-multiply-a.svg):
* task\_description.png: