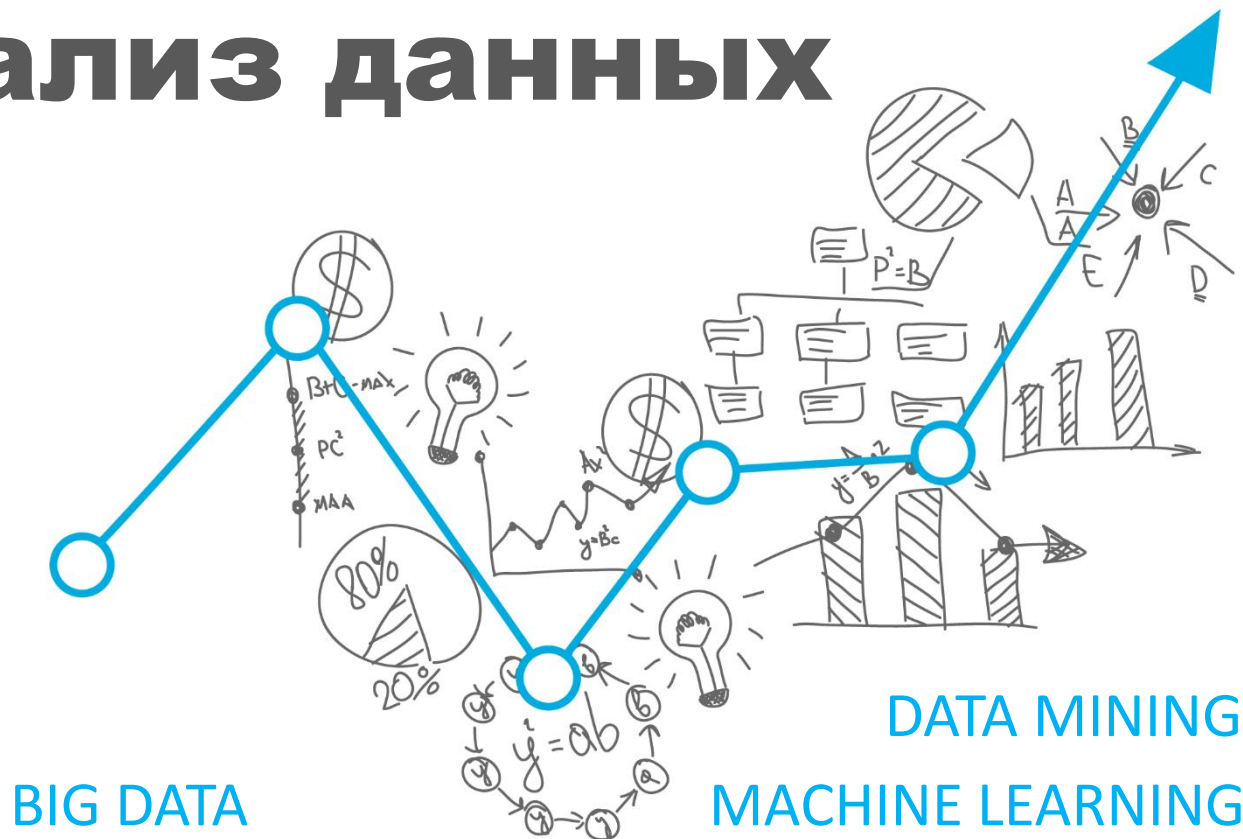


# Интеллектуальный анализ данных



## Лекция 1. Основы линейной алгебры

- Основы линейной алгебры в языке Python
- Задачи машинного обучения (регрессия, классификация, кластеризация), предсказательная модель
- Методы машинного обучения, машина опорных векторов, нейронные сети и другие
- Методика решения задач машинного обучения и анализа данных, диагностика, метрики, регуляризация
- Обнаружение аномалий
- Обработка больших данных

Матрица  $m \times n$

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad \left. \vphantom{\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}} \right\} m \quad \mathbb{R}^{m \times n}$$

Элемент матрицы  $A_{ij}$   $n$

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 147 & 1437 \end{bmatrix}$$

$$A_{11} = 1402$$

$$A_{12} = 191$$

$$A_{21} = 1371$$

$$A_{32} = 1437$$

$$A_{23} = \text{undefined}$$

Вектор – матрица  
размерами  $n \times 1$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\mathbb{R}^n$$

Индексация на базе

нуля

единицы

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Сложение матриц

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1+4 & 0+0.5 \\ 2+2 & 5+5 \\ 3+0 & 1+1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 9 \\ 1 & 0.7 \end{bmatrix} = \text{error}$$

Операция сложения матриц является *коммутативной*:  $A + B = B + A$

Умножения матрицы  
на число

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 9 \end{bmatrix} = \begin{bmatrix} 3 \times 1 & 3 \times 0 \\ 3 \times 2 & 3 \times 5 \\ 3 \times 3 & 3 \times 9 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 27 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 9 \end{bmatrix} \times 3$$

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} / 4 = \frac{1}{4} \times \begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1.5 & 3/4 \end{bmatrix}$$

Комбинирование операций  
(приоритеты как в арифметике)

$$3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix} - \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} / 2 = \begin{bmatrix} 3 \\ 12 \\ 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 11 \\ 7 \end{bmatrix}$$

Умножение матрицы на вектор

$$A^{m \times n} \times x^n = y^m$$

$$y_i = \sum_{j=1}^n A_{ij} \cdot x_j, \quad i = 1, \dots, m$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 3 \cdot 5 \\ 4 \cdot 1 + 0 \cdot 5 \\ 2 \cdot 1 + 1 \cdot 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -3 & -2 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 3 + 1 \cdot 2 + 5 \cdot 1 \\ 0 \cdot 1 + 3 \cdot 3 + 0 \cdot 2 + 4 \cdot 1 \\ -1 \cdot 1 + (-2) \cdot 3 + 0 \cdot 2 + 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix}$$

Умножение матрицы на матрицу

$$A^{m \times n} \times B^{n \times o} = C^{m \times o}$$

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}, \quad i = 1, \dots, m, \quad j = 1, \dots, o.$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 3 \cdot 0 + 2 \cdot 5 & 1 \cdot 3 + 3 \cdot 1 + 2 \cdot 2 \\ 4 \cdot 1 + 0 \cdot 0 + 1 \cdot 5 & 4 \cdot 3 + 0 \cdot 1 + 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix} \begin{matrix} 10 \\ 14 \end{matrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

Умножение матрицы на матрицу **не является коммутативным!**

$$A \times B \neq B \times A$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \\ 2 & 7 \\ 8 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \end{bmatrix} = \text{error}$$

Умножение матрицы на матрицу **является ассоциативным**

$$A \times (B \times C) = (A \times B) \times C.$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = ?$$

Вариант 1:  $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}; \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 0 \end{bmatrix}$

Вариант 2:  $\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 4 & 2 \end{bmatrix}; \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 0 \end{bmatrix}$



**Единичная матрица** является такой матрицей, умножение которой на любую другую матрицу дает в результате вторую матрицу без изменений:

$$I \times A = A \times I = A$$

Аналог среди действительных чисел

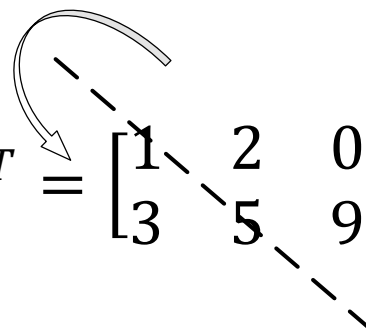
$$1 \cdot z = z \cdot 1 = z$$

$$I^{m \times m} \quad \begin{aligned} I^{2 \times 2} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ I^{3 \times 3} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ I^{1 \times 1} &= [1] \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}^{4 \times 4}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 1 \\ 3 \cdot 1 + 4 \cdot 0 & 3 \cdot 0 + 4 \cdot 1 \\ 5 \cdot 1 + 6 \cdot 0 & 5 \cdot 0 + 6 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Операция транспонирования матрицы  $A^T$

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$$


**Обратная матрица** – это такая матрица, при умножении на которую в результате получается единичная матрица:

$$A \times A^{-1} = A^{-1} \times A = I$$

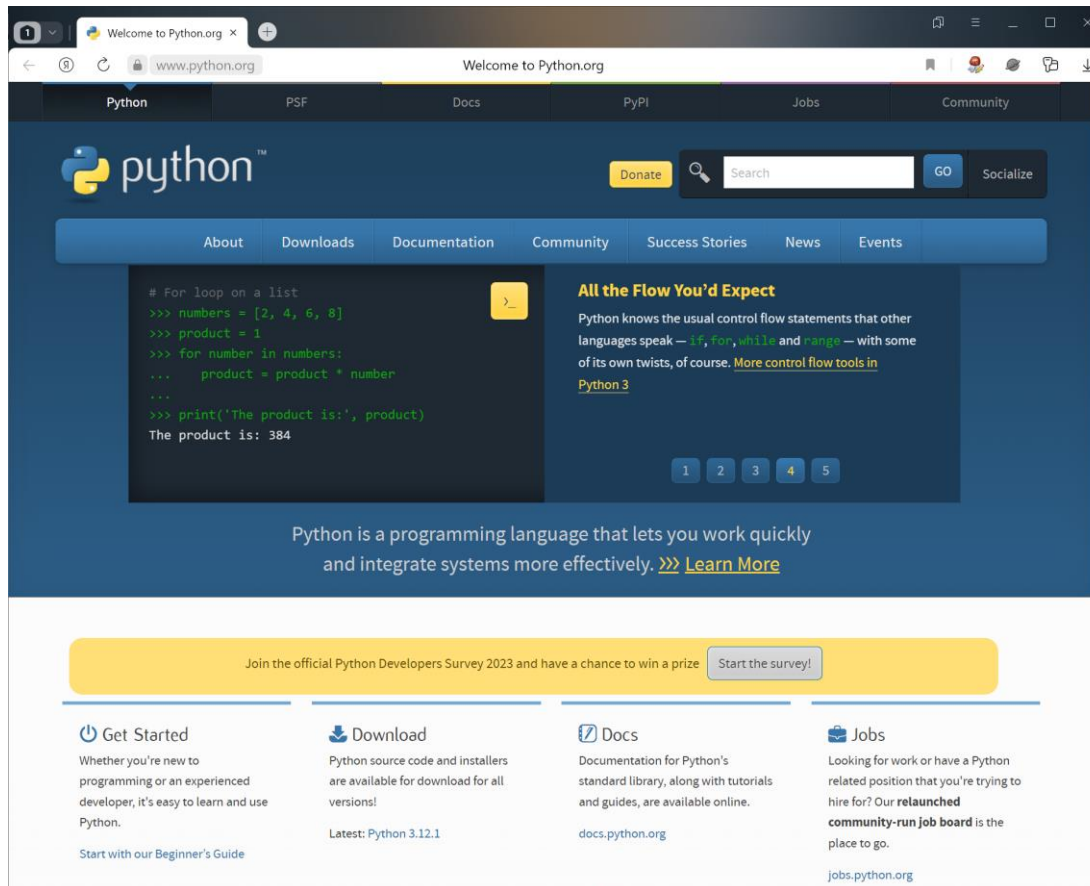
$$A = \begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \quad A^{-1} = \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix}$$

$$AA^{-1} = \begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

*Вырожденные матрицы не имеют обратной! Пример:*

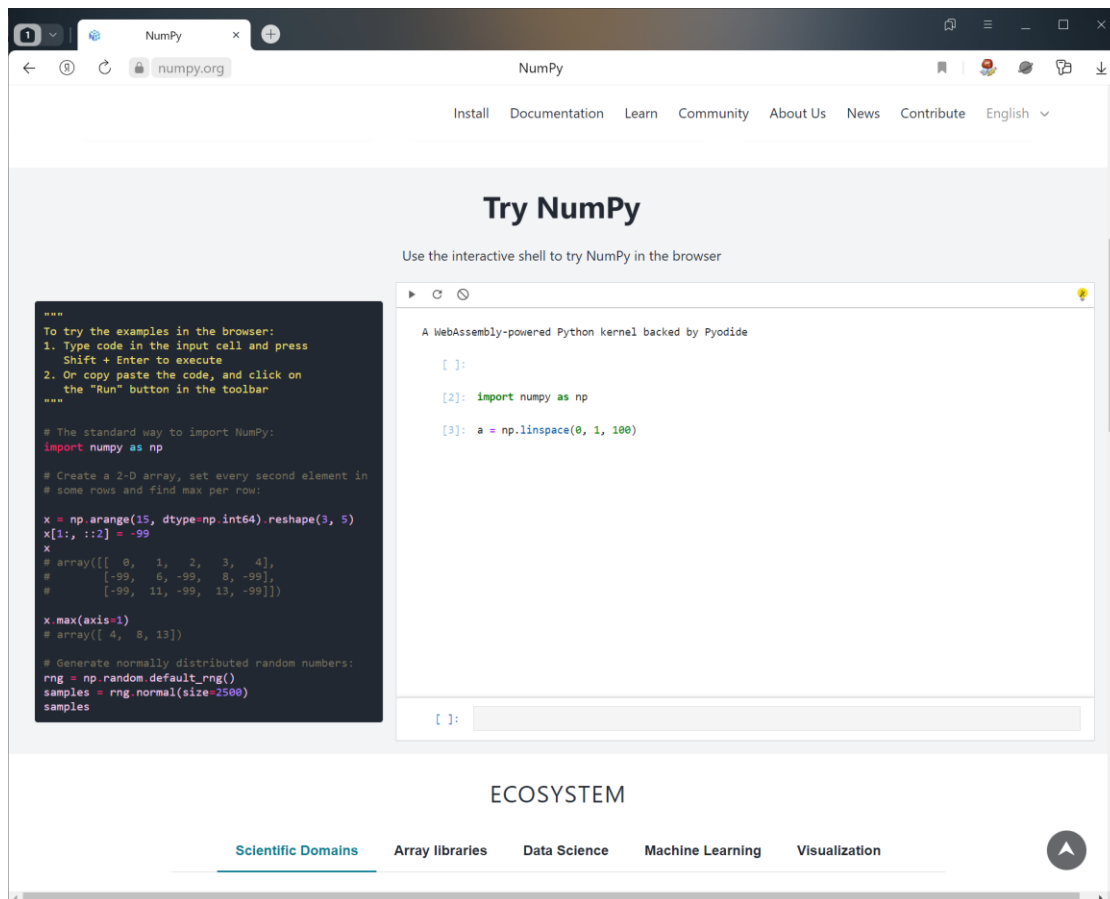
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Python – интерпретируемый язык программирования с богатым набором библиотек для инженерных и научных расчётов



<https://www.python.org/>

NumPy – библиотека численных алгоритмов и матричных операций для языка Python



<https://numpy.org/>

```
pip install numpy
```

```
import numpy as np
```

## Создание массива (вектора) из списка

```
list1d = [1, 0.5, 4, 2]  
arr1d = np.array(list1d)  
print(arr1d)
```

```
[1.  0.5 4.  2. ]
```

## Создание матрицы из списка

```
list2d = [[1, 2, 3], [4, 5, 6]]  
arr2d = np.array(list2d)  
print(arr2d)
```

```
[[1 2 3]  
 [4 5 6]]
```

```
arr1d = np.array([1, 0.5, 4, 2])  
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
```

## Указание типа данных в массиве

```
arr1d = np.array([1, 0.5, 4, 2], dtype=float)  
print(arr1d)
```

```
[1.  0.5 4.  2. ]
```

```
arr1d = np.array([1, 0.5, 4, 2], int)  
print(arr1d)
```

```
[1 0 4 2]
```

```
arr1d = np.array([2, 0.5, 0, -7], bool)  
print(arr1d)
```

```
[ True  True False  True]
```

## Нулевые массивы

```
z1 = np.zeros(5, float)  
print(z1)
```

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

```
z2 = np.zeros((2, 3), int)
```

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

```
z3 = np.zeros((2, 3, 4), int)
```



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

## Массивы единиц

```
e1 = np.ones(3, int)
```

```
[1 1 1]
```

```
e2 = np.ones((3, 2))
```

```
[[1. 1.]  
 [1. 1.]  
 [1. 1.]]
```

## Массивы с одним значением

```
f1 = np.full((2, 3), 5.1, float)
```

```
[[5.1 5.1 5.1]  
 [5.1 5.1 5.1]]
```



## Единичные матрицы

```
np.eye(3)
```

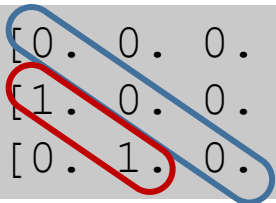
```
[[1.  0.  0.]  
 [0.  1.  0.]  
 [0.  0.  1.]]
```

```
np.eye(3, 4)
```

```
[[1.  0.  0.  0.]  
 [0.  1.  0.  0.]  
 [0.  0.  1.  0.]]
```

```
np.eye(3, 4, -1)
```

```
[[0.  0.  0.  0.]  
 [1.  0.  0.  0.]  
 [0.  1.  0.  0.]]
```



```
np.eye(3, dtype=int)
```

```
[[1 0 0]  
 [0 1 0]  
 [0 0 1]]
```

## Генерация диапазона

```
arange([start,] stop, [step,] [dtype])
```

start	начальное значение диапазона (если не указано, то равно 0)
stop	конечное значение диапазона (обязательный параметр), не входит в результат
step	шаг (если не задан, то равен 1)
dtype	тип данных (если не задан, то определяется по типу первых параметров)

```
np.arange(1, 9, 2, float)  
np.arange(start=1, stop=9, step=2, dtype=float)
```

```
[1.  3.  5.  7.]  
[1.  3.  5.  7.]
```

```
np.arange(1, step=0.2)
```

```
[0.  0.2  0.4  0.6  0.8]
```

## Генерация диапазона

```
linspace(start, stop, num=50, endpoint=True, dtype=None)
```

start	начальное значение диапазона (обязательный параметр)
-------	--

stop	конечное значение диапазона (обязательный параметр)
------	---

num	число элементов (если не указано, 50)
-----	---------------------------------------

endpoint	признак включения значения stop в последовательность (по умолчанию, True)
----------	---

dtype	тип данных
-------	------------

```
np.linspace(1, 2, 5)
```

```
[1.    1.25 1.5   1.75 2.   ]
```

```
np.linspace(1, 2, 5, endpoint=False)
```

```
[1.  1.2 1.4 1.6 1.8]
```

## Генерация случайного массива целых чисел

```
randint(low, high=None, size=1)
```

low	если задан только этот параметр, то указывает максимальное значение для случайных чисел (но само значение не входит в диапазон)
-----	---

high	если заданы оба (low и high), то они задают диапазон генерации чисел [low, high)
------	--

size	число элементов (если не указано, 1)
------	--------------------------------------

```
np.random.randint(5, size=5)
```

```
[0 0 2 1 4]
```

```
np.random.randint(10, 20, size=5)
```

```
[16 17 16 18 18]
```

Генерация случайного массива действительных чисел  $[0, 1)$

```
random.rand(d0, d1, ...)
```

d0          размер по первому измерению (оси) массива

d1          размер по второму измерению (оси) массива

...        и т.д., сколько необходимо измерений

```
np.random.rand(4)
```

```
[0.1836634  0.46449058 0.88689603 0.32688258]
```

```
np.random.rand(2, 3)
```

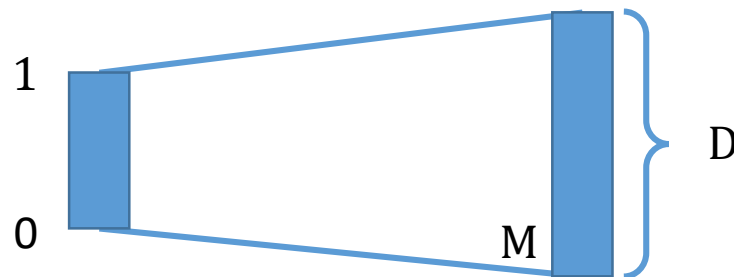
```
[[0.77549814 0.83479098 0.71086936]  
 [0.04089972 0.83923191 0.33830922]]
```

Генерация случайного массива действительных чисел  
в произвольном диапазоне

$$D * \text{rand}() + M$$

D	ширина диапазона случайных чисел
---	----------------------------------

M	нижнее значение диапазона случайных чисел
---	---



```
2 * np.random.rand(2, 3) - 1
```

```
[[ 0.35214644 -0.32754867 -0.28123389]
 [ 0.97920488 -0.81045682 -0.77036573]]
```

## Основные типы данных

<code>int</code>	целочисленный
<code>float</code>	с плавающей точкой
<code>bool</code>	булевый
<code>complex</code>	комплексный
<code>str</code>	символьный
и другие	

## Определение типа данных массива

```
d1 = np.array([1, 2, 3])  
print(d1.dtype)
```

```
int32
```

## Изменение типа данных массива

```
d1 = np.array([1, 0, 3])
print('Тип исходный:', d1.dtype)
print('Значение: ', d1)
d2 = d1.astype(bool)
print('Тип после преобразования:', d2.dtype)
print('Значение: ', d2)
```

```
Тип исходный: int32
Значение:  [1 0 3]
Тип после преобразования: bool
Значение:  [ True False  True]
```



## Свойства

<code>ndim</code>	Количество измерений в массиве
<code>shape</code>	Размеры массива по всем измерениям: (d1, d2, ...)

```
d1 = np.ones((4, 3))  
print('Размерность:', d1.ndim, 'Размер:', d1.shape)
```

```
Размерность: 2 Размер: (4, 3)
```

```
print('Строк:', d1.shape[0], 'Столбцов:', d1.shape[1])
```

```
Строк: 4 Столбцов: 3
```

## Изменение размеров массива

```
np.reshape(arr, (d0, d1, ...))  
arr.reshape((d0, d1, ...))
```

`arr`      исходный массив

`d0`      размер по первому измерению (оси) массива

`d1`      размер по второму измерению (оси) массива

`...`      и т.д., сколько необходимо измерений

```
d1 = np.ones((4, 3))  
d2 = np.reshape(d1, (2, 6))
```

```
d1 = np.ones((4, 3))  
d2 = d1.reshape((2, 6))
```

```
[[1.  1.  1.  1.  1.  1.]  
 [1.  1.  1.  1.  1.  1.]
```

Количество элементов должно совпадать!

Обращение к элементам массива (начинается с нуля)

<code>d1[0]</code>	Обращение к первому элементу
<code>d1[i]</code>	Обращение к $i$ -му элементу
<code>d1[1, 2]</code>	Обращение элементу матрицы второй строки третьего столбца

Обращаться за  
границы массива  
запрещено!

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(d1[1, 2])
```

6

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(d1[1])
```

[4 5 6]

## Присваивание массивов не копирует их содержимое

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(d1)  
d2 = d1  
d2[0, 1] = 9  
print(d1)
```



```
[[1 2 3]  
 [4 5 6]]  
[[1 9 3]  
 [4 5 6]]
```

## Создание независимой копии массива

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(d1)  
d2 = d1.copy()  
d2[0, 1] = 9  
print(d1)
```



```
[[1 2 3]  
 [4 5 6]]  
[[1 2 3]  
 [4 5 6]]
```

Срез – представление произвольной части массива с другими размерами

start : stop

start    начальный индекс среза, если не указан, то с начала массива

stop     конечный индекс (в диапазон не входит), если не указан, то до конца

```
d1 = np.arange(10)
print(d1)
d2 = d1[2:5]
print(d2)
```

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4]
```

```
d2 = d1[:]
print(d2)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

### Срез по нескольким измерениям

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = d1[:, 1:3]  
print(d2)
```

```
[[2 3]  
 [5 6]]
```

### Получение столбца из матрицы

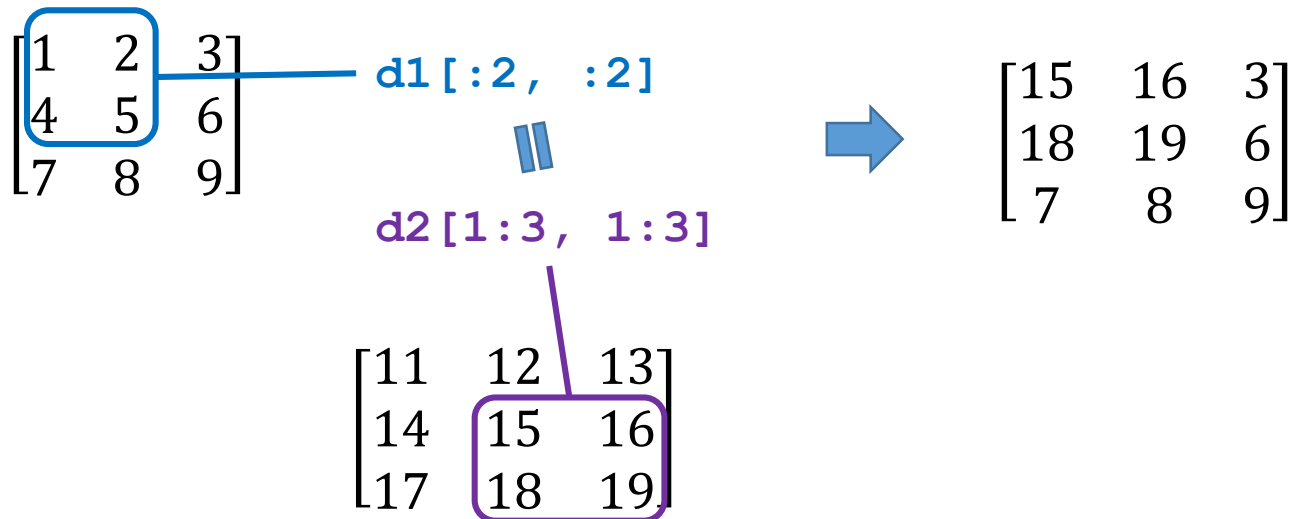
```
d1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
d2 = d1[:, 1]  
print(d2)
```

```
[2 5 8]
```

Пример: заменить часть матрицы частью другой матрицы

```
d1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
d2 = np.array([[11, 12, 13], [14, 15, 16], [17, 18, 19]])  
d1[:2, :2] = d2[1:3, 1:3]  
print(d1)
```

```
[[15 16  3]  
 [18 19  6]  
 [ 7  8  9]]
```



Срез является ссылкой, а не копией массива

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = d1[:, 1:3]  
print(d2)
```

```
[[2 3]  
 [5 6]]
```

Можно сделать копию среза и поместить в отдельный массив

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = d1[:2, 1:3].copy()  
d2[0, 0] = 9  
print(d1)  
print(d2)
```

```
[[1 2 3]  
 [4 5 6]]  
[[9 3]  
 [5 6]]
```



## Объединение массивов

```
append(arr, values, axis=None)
```

`arr`      исходный массив

`values`   размер по второму измерению (оси) массива

`axis`      вдоль какой оси складывать (если не указана, то оба массива будут преобразованы в одномерные)

```
d1 = np.array([[1, 2], [3, 4]])  
d2 = np.array([[11], [12]])  
d3 = np.append(d1, d2, axis=1)  
print(d3)
```

```
[[ 1  2 11]  
 [ 3  4 12]]
```

## Объединение массивов

`vstack(tup, dtype=None)` – соединить по вертикали  
`hstack(tup, dtype=None)` – соединить по горизонтали

`tup` список объединяемых массивов: `(a1, a2, ...)`

`dtype` тип данных нового массива (если не указан, то определяется сам)

```
d1 = np.array([1, 2, 3])
d2 = np.array([4, 5, 6])
d3 = np.array([7, 8, 9])
d4 = np.vstack((d1, d2, d3))
print(d4)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
d1 = np.array([[1], [2], [3]])
d2 = np.array([[4], [5], [6]])
d3 = np.array([[7], [8], [9]])
d4 = np.hstack((d1, d2, d3))
print(d4)
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Логическое условие формирует булевый массив

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = (d1 == 3)  
print(d2)
```

```
[[False False  True]  
 [False False False]]
```

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = (d1 >= 2) & (d1 < 5)  
print(d2)
```

```
[[False  True  True]  
 [ True False False]]
```

Булевый массив можно использовать для выбора элементов из другого массива (в виде одномерного массива)

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = (d1 >= 2) & (d1 < 5)  
d3 = d1[d2]  
print(d3)
```

```
[2 3 4]
```

Условие можно писать в скобках для индексации

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = d1[(d1 >= 2) & (d1 < 5)]  
print(d2)
```

```
[2 3 4]
```

Тернарный оператор для массивов (поэлементно)

```
where(condition, x, y)
```

condition	логическое условие или булевый массив (размера как x)
-----------	---

x	первый массив
---	---------------

y	второй массив (размера как x)
---	-------------------------------

Если условие выполняется, берется элемент из **x**, иначе из **y**

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([[11, 12, 13], [14, 15, 16]])  
d3 = np.where(d1 % 2 == 0, d1, d2)  
print(d3)
```

```
[[11  2 13]  
 [ 4 15  6]]
```

Функции могут вычисляться для каждого элемента массива

```
d1 = np.array([1, 2, 3, 4, 5])  
print(np.log(d1))
```

```
[0.          0.69314718  1.09861229  1.38629436  1.60943791]
```

```
np.sqrt([4, 9, 16, 25])
```

```
[2.  3.  4.  5.]
```

Функции могут вычисляться для всего массива, строки, столбца

```
np.max([4, 9, 16, 25])
```

```
25
```

Режим работы функции может зависеть от параметров

```
m = np.random.rand(3, 3)
np.mean(m)
np.std(m)
```

```
0.617
0.245
```

```
m = [[0.614 0.609 0.058]
      [0.807 0.550 0.977]
      [0.450 0.725 0.757]]
```

```
np.mean(m, axis=0)
np.std(m, axis=0)
```

```
[0.624 0.628 0.597]
[0.146 0.073 0.392]
```

```
np.mean(m, axis=1)
np.std(m, axis=1)
```

```
[0.427 0.778 0.644]
[0.260 0.175 0.138]
```

## Примеры некоторых функций

abs	Вычисляет абсолютные значения каждого элементов массива
sqrt	Вычисляет квадратный корень из каждого элемента массива (эквивалентно $\text{arr} ** 0.5$ )
exp	Вычисляет экспоненту ( $e^x$ ) от каждого элемента массива
log, log10, log2, log1p	Вычисляет натуральный, десятичный логарифмы, логарифм по основанию 2 и $\log(1 + x)$
ceil	Вычисляет наименьшее целое число большее либо равное каждого элемента массива
floor	Вычисляет наибольшее целое число меньшее либо равное каждого элемента массива
cos, cosh, sin, sinh, tan, tanh	Обычные и гиперболические тригонометрические функции
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Обратные тригонометрические функции



```
np.sort(arr)
```

```
arr
```

массив для сортировки

Возвращает упорядоченный по возрастанию значений элементов массив (есть другие параметры: порядок сортировки, оси и т.д.)

```
d1 = np.random.randint(100, size=10)
print(d1)
d2 = np.sort(d1)
print(d2)
```

```
[11 85 79 35  0 15 60 73 10 70]
[ 0 10 11 15 35 60 70 73 79 85]
```

**Множество** в NumPy – это специальный массив, в котором каждый элемент встречается только один раз

Сформировать множество из массива

```
d1 = np.array([[1, 2, 2], [3, 3, 4]])  
d2 = np.unique(d1)  
print(d2)
```

```
[1 2 3 4]
```

Проверить вхождение элементов одного массива в другом

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([1, 3, 5])  
d3 = np.in1d(d1, d2)  
print(d3)
```

```
[ True False  True False  True False]
```

## Основные функции работы со множествами

<code>unique(x)</code>	Возвращает отсортированные единственные элементы из <code>x</code>
<code>intersect1d(x, y)</code>	Возвращает общие элементы массивов <code>x</code> и <code>y</code>
<code>union1d(x, y)</code>	Возвращает объединение элементов массивов <code>x</code> и <code>y</code>
<code>in1d(x, y)</code>	Возвращает булев массив, указывающий содержится ли каждый элемент массива <code>x</code> в <code>y</code>
<code>setdiff1d(x, y)</code>	Возвращает элементы массива <code>x</code> , которых нет в <code>y</code>
<code>setxor1d(x, y)</code>	Возвращает элементы, которые есть либо в <code>x</code> , либо в <code>y</code> , но не в обоих массивах

Библиотека **numpy** и ее подмножество **numpy.linalg** содержит набор функций для выполнения операций над матрицами и векторами

Транспонирование матрицы

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = d1.T  
print(d2)
```

```
[[1 4]  
 [2 5]  
 [3 6]]
```

Альтернатива: использование функции `transpose`

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.transpose(d1)  
print(d2)
```

## Сложение и вычитание матриц

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([[11, 12, 13], [14, 15, 16]])  
print(d1 + d2)  
print(d2 - d1)
```

```
[[12 14 16]  
 [18 20 22]]  
[[10 10 10]  
 [10 10 10]]
```

## Поэлементное умножение и деление

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([[2, 1, 2], [1, 2, 1]])  
print(d1 * d2)  
print(d1 / d2)
```

```
[[ 2  2  6]  
 [ 4 10  6]]  
[[0.5 2.  1.5]  
 [4.  2.5 6.  ]]
```

Арифметические операции между матрицами и скалярами (+ - \* / и т.д.)

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(1 - d1)  
print(d1 / 2)
```

```
[[ 0 -1 -2]  
 [-3 -4 -5]]  
[[0.5 1.  1.5]  
 [2.  2.5 3.  ]]
```

Возведение всех элементов матрицы в степень

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
print(d1 ** 2)
```

```
[[ 1  4  9]  
 [16 25 36]]
```

## Произведение матрицы на вектор

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([1, 2, 3])  
print(np.dot(d1, d2))
```

```
[14 32]
```

```
print(d1 @ d2)
```

– альтернативный синтаксис

## Произведение матрицы на матрицу

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
d2 = np.array([[11, 12], [13, 14], [15, 16]])  
print(np.dot(d1, d2))
```

```
[[ 82  88]  
 [199 214]]
```

```
print(d1 @ d2)
```

– альтернативный синтаксис

Нахождение определителя (det), обратной матрицы (inv) ,  
псевдообратной матрицы (pinv)

```
d1 = np.random.rand(3, 3)
print(np.linalg.det(d1))
print(np.linalg.inv(d1))
print(np.linalg.pinv(d1))
```

```
0.3457249105553521
```

```
[[-1.17032681  2.06019789 -0.15938304]
 [-0.46795666 -0.09955335  1.2684454 ]
 [ 1.55842401 -0.29567334 -0.64258227]]
```

```
[[-1.17032681  2.06019789 -0.15938304]
 [-0.46795666 -0.09955335  1.2684454 ]
 [ 1.55842401 -0.29567334 -0.64258227]]
```



Сохранить переменную в файл

```
d1 = np.array([[1, 2, 3], [4, 5, 6]])  
np.save('ndarr-d1', d1)
```

Прочитать из файла

```
d2 = np.load('ndarr-d1.npy')  
print(d2)
```

```
[[1 2 3]  
 [4 5 6]]
```

Если расширение файла не указано, то принимается \*.npy

Сохранить можно несколько массивов в один файл

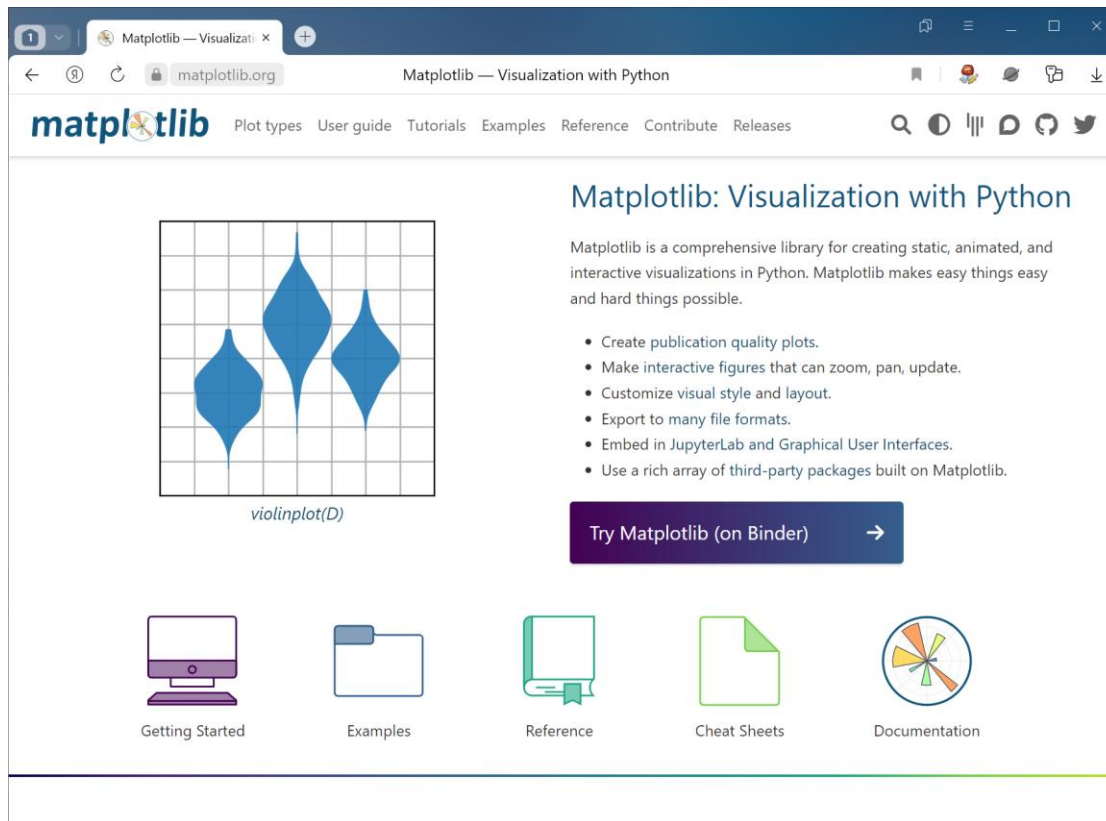
```
d1 = np.array([[1, 2, 3], [4, 5, 6]])
d2 = np.array([[11, 12, 13], [14, 15, 16]])
with open('ndarr-2.npy', 'wb') as f:
    np.save(f, d1)
    np.save(f, d2)
```

Прочитать из файла в том же порядке, что и были записаны

```
with open('ndarr-2.npy', 'rb') as f:
    d1l = np.load(f)
    d2l = np.load(f)
print(d1l)
print(d2l)
```

```
[[1 2 3]
 [4 5 6]]
[[11 12 13]
 [14 15 16]]
```

**matplotlib.pyplot** – библиотека для рисование различных видов графиков (двумерные, трехмерные, диаграммы, гистограммы и т.д.)



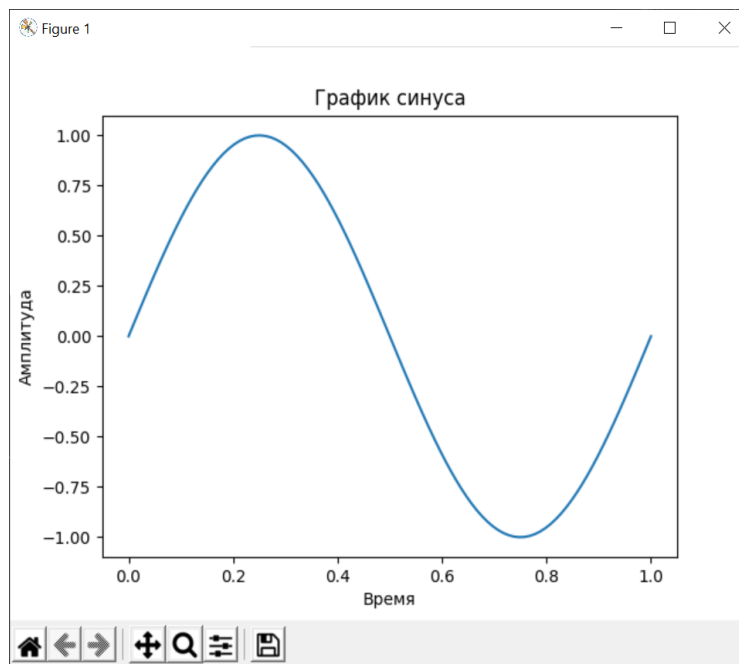
<https://matplotlib.org/>

```
pip install matplotlib
```

```
import matplotlib.pyplot as plt
```

## Отображение одного графика

```
t = np.linspace(0, 1, 100)
y = np.sin(2 * np.pi * t)
plt.plot(t, y)
plt.title('График синуса')
plt.xlabel('Время')
plt.ylabel('Амплитуда')
plt.show()
```



## Отображение нескольких графиков

```
t = np.linspace(0, 1, 100)
y1 = np.sin(2 * np.pi * t)
y2 = np.sin(4 * np.pi * t)
plt.plot(t, y1, 'r', label='sin 2*pi*t')
plt.plot(t, y2, 'g', label='sin 4*pi*t')
plt.title('Графики синусов')
plt.xlabel('Время')
plt.ylabel('Амплитуда')
plt.legend()
plt.show()
```

