



Введение в машинное обучение

Библиотека языка Питон

NumPy

Дьяконов А.Г.

**Московский государственный университет
имени М.В. Ломоносова (Москва, Россия)**

NumPy

NumPy – пакет для языка программирования Python

- **N-мерные массивы**
- **функции линейной алгебры, преобразования Фурье, псевдослучайные генераторы**
- **средства для интеграции C/C++ и Fortran**

По сути, копия MatLab

Для начала работы:

```
import numpy as np
```

Для графики:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Векторы

```
a = np.array([0, 1, 2, 3])  
b = np.array(range(4))  
c = np.arange(4)
```

Создание вектора

array – матрица

a, b, c

```
(array([0, 1, 2, 3]),  
 array([0, 1, 2, 3]),  
 array([0, 1, 2, 3]))
```

arange – вектор из
последовательности

Зачем нужен пакет

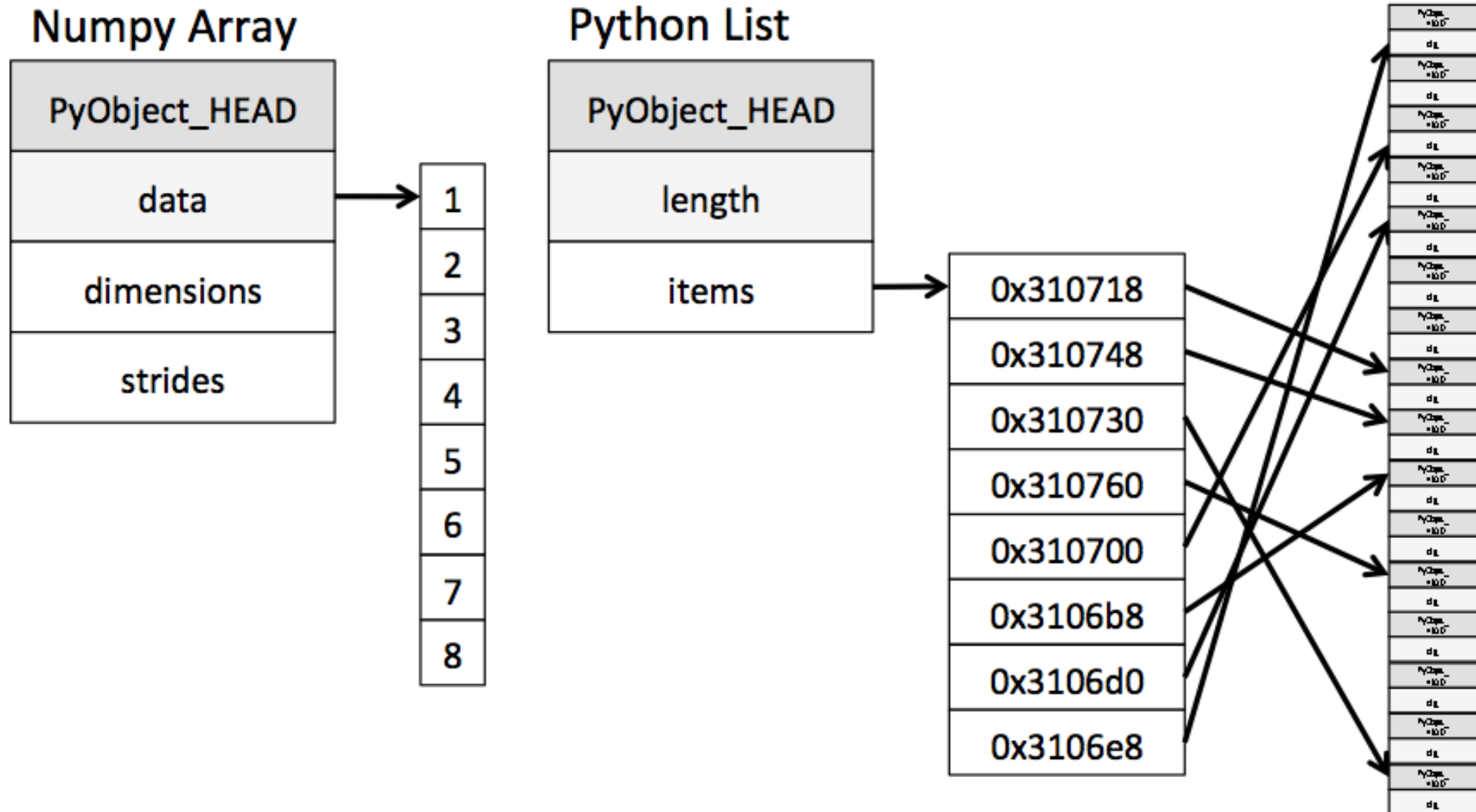
```
%timeit z = [x ** 2 for x in range(1000)]
```

206 μ s \pm 1.98 μ s per loop

```
%timeit z = np.arange(1000) ** 2
```

2.97 μ s \pm 31.7 ns per loop

Типы в Python



<https://jakevdp.github.io/PythonDataScienceHandbook/02.01-understanding-data-types.html>

Создание векторов

```
np.full(5, 3.14) # из одного элемента  
[3.14 3.14 3.14 3.14 3.14]
```

```
np.arange(1, 10, 3) # от, до, шаг  
[1 4 7]
```

```
np.linspace(0, 1, 5) # от, до, сколько  
[0. 0.25 0.5 0.75 1. ]
```

```
np.ones(3) # из единиц  
[1. 1. 1.]
```

```
np.ones(3, dtype=int) # из единиц  
[1 1 1]
```

```
np.zeros(3) # из нулей
```

```
np.empty(4) # просто выделить память  
[0.77132064 0.02075195 0.63364823 0.74880388]
```

Создание векторов

```
np.random.seed(10) # фиксируем сид
```

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

```
[0.77132064 0.02075195 0.63364823 0.74880388]
```

```
np.random.randn(4)
```

```
[-1.54540029 -0.00838385 0.62133597 -0.72008556]
```

```
np.random.randint(0, 5, 10)
```

```
[0 2 0 4 3 0 4 3 0 3]
```

```
np.random.choice([2, 3, 4], 5, replace=True)
```

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

```
np.tile([1, 2], 3) # повторить
```

```
[1 2 1 2 1 2]
```

Создание векторов

```
np.choose([0,1,0,1,1], [[1,1,1,1,1], [2,2,2,2,2]]) # откуда брать  
[1 2 1 2 2]
```

аналогично с двумя векторами **new**

```
np.where(np.arange(5) % 2, np.ones(5), np.zeros(5))  
array([0., 1., 0., 1., 0.])
```

выбор элементов с указанными индексами, но можно выбирать из списка

```
np.take([1, 2, 3, 4, 5], [0, 2, 3])  
array([1, 3, 4])
```

Добавление элементов

```
x = np.arange(3)
```

new

```
x1 = np.append(x, 3)
```

```
array([0, 1, 2, 3])
```

```
x2 = np.insert(x, 1, 3)
```

```
array([0, 3, 1, 2])
```

```
x3 = np.delete(x, 1)
```

```
array([0, 2])
```


Динамическая типизация

```
x = np.array([1, 2])  
print (x.dtype)  
int64
```

```
x = x + 0.0 # x += 0.0 конкретно  
здесь не работает  
print (x.dtype)  
float64
```

```
x = x + 2j  
print (x.dtype)  
complex128
```

```
x = x > 0  
print (x.dtype)  
bool
```

**Тип можно указывать при
создании массива.**

**Разные типы – разная
потребляемая память**

Типы

```
x = np.array([-0.6, 0.4, 1.4, 1.5, 1.6])
```

```
print (np.round(x))  
[-1.  0.  1.  2.  2.]
```

```
print (x.astype(int))  
[0 0 1 1 1]
```

```
print (np.round(x))  
[-1.  0.  1.  2.  2.]
```

```
print (np.round(x).astype(int))  
[-1  0  1  2  2]
```

uint8	8 bits	float16	16 bits
uint16	16 bits	float32	32 bits
uint32	32 bits	float64	64 bits (same as float)
uint64	64 bits	float96	96 bits, platform-dependent (same as np.longdouble)
		float128	128 bits, platform-dependent (same as np.longdouble)

Индексация как для списков

```
x = np.arange(10)
```

```
x[0]
```

```
0
```

```
x[:2]
```

```
[0 1]
```

```
x[-2:]
```

```
[8 9]
```

```
x[::-2]
```

```
[9 7 5 3 1]
```

```
x[1:8:3])
```

```
[1 4 7]
```

```
x = np.arange(10)
```

```
x[1:8:3] = -1
```

```
x
```

```
[ 0 -1  2  3 -1  5  6 -1  8  9]
```

С 0 (как в C, Matlab и Fortran – с 1)

**Индексировать можно и при
присваивании.**

Тонкие моменты

Присваивание никогда не меняет тип

```
x = np.arange(5, dtype='int')
x[1] = 3.14 # округление
x
array([0, 3, 2, 3, 4])
```

Общая память

не происходит копирования!

```
x = np.ones(3)
x2 = x[1:] # надо .copy()
x2[0] = 0
x
array([1., 0., 1.])
```

При выделении подматрицы – она не копируется!

```
np.may_share_memory(x, y)
True
```

Операции – поэлементные

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

```
y = np.array([2, 1, 0])
```

```
x + y  
[3 3 3]
```

```
x - y  
[-1  1  3]
```

```
x * y  
[2 2 0]
```

```
x / y  
[0.5 2.  inf]
```

```
x ** y  
[1 2 1]
```

```
x + 1  
[2 3 4]
```

```
2 ** x  
[2 4 8]
```

```
np.sin(x)  
[0.84147098 0.90929743 0.14112001]
```

```
np.exp(x)  
[ 2.71828183  7.3890561 20.08553692]
```

```
np.log(x).round(2)  
[0.  0.69 1.1 ]
```

Функции **new**

`np.cos, np.sin, np.tan`

`np.arccos, np.arcsin, np.arctan`

`np.cosh, np.sinh, np.tanh`

`np.arccosh, np.arcsinh, np.arctanh`

`np.sqrt`

`np.exp`

`np.log, np.log2, np.log10`

`np.sign, np.abs`

`np.floor, np.ceil, np rint`

`np.round`

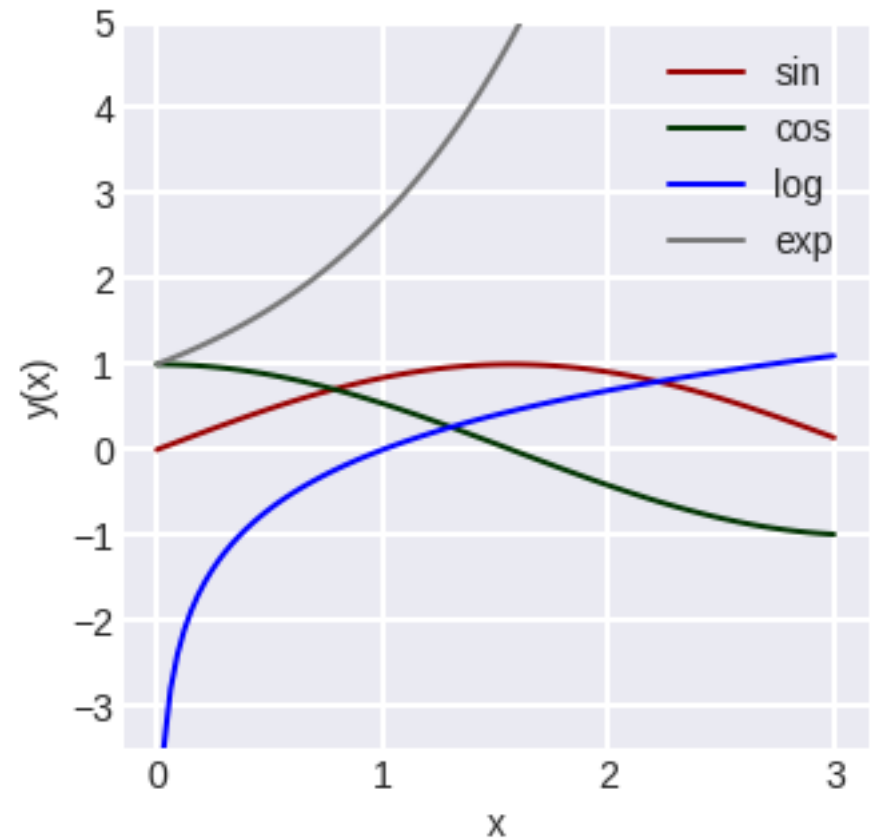
Пример использования векторов – графики

```
x = np.linspace(0, 3, 100)

plt.figure(figsize=(5, 5))

plt.plot(x, np.sin(x), lw=2,
c='#990000', label='sin')
plt.plot(x, np.cos(x), lw=2,
c='#003300', label='cos')
plt.plot(x, np.log(x), lw=2,
c='#0000FF', label='log')
plt.plot(x, np.exp(x), lw=2,
c='#777777', label='exp')

plt.ylim([-3.5, 5])
plt.grid(lw=2)
plt.legend()
plt.xlabel('x')
plt.ylabel('y(x)')
plt.show()
```



Логические массивы (маски)

```
x = np.round(np.random.rand(6), 2)
[ 0.44  0.03  0.55  0.44  0.42  0.33]
y = np.round(np.random.rand(6), 2)
[ 0.2   0.62  0.3   0.27  0.62  0.53]
```

round – округление

```
x[y > 0.5]
[ 0.03  0.42  0.33]
```

**индексация логическим
вектором**

```
xi = x > 0.5
[False False  True False False False]
yi = y > 0.5
[False  True False False  True  True]
```

**Дизъюнкция,
конъюнкция и
отрицание.**

```
xi | yi
[False  True  True False  True  True]
```

**Можно использовать
logical_or, np.logical_and**

```
xi & yi
[False False False False False False]
```

```
~xi
[ True  True False  True  True  True]
```


Логические массивы (маски)

```
xi.any()
```

```
True
```

any – хотя бы один эл-т True

all – все элементы True

```
xi.all()
```

```
False
```

nonzero – индексы

ненулевых эл-ов

```
yi.nonzero()
```

```
array([1, 4, 5])
```

```
x = np.array([0.1, 0.5, 0.2, 0.6])
```

Замена элементов

```
x[x<0.3] = 0.3
```

```
array([ 0.3,  0.5,  0.3,  0.6])
```

другой способ

```
np.maximum(x, 0.3)
```

```
array([ 0.3,  0.5,  0.3,  0.6])
```

Логические массивы (маски)

```
a = np.array([1, 3, 1, 4])  
b = np.array([2, 1, 1, 4, 4])  
np.in1d(a, b)  
array([ True, False,  True,  True])
```

Сравнения

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

```
y = np.array([2, 1, 2])
```

```
print (x > y)
```

```
[False  True False]
```

```
print (x == y)
```

```
[False False  True]
```

```
print (np.array_equal(x, y))
```

```
False
```

```
print (np.array_equal(x, x))
```

```
True
```

Разные максимумы

```
x = np.array([0.1, 0.5, 0.2, 0.6])
```

```
y = np.array([0.2, 0.4, 0.3, 0.1])
```

```
x.min(), x.max()
```

```
0.1 0.6
```

```
np.min(x), np.max(x)
```

```
0.1 0.6
```

```
np.minimum(x, y)
```

```
[ 0.1  0.4  0.2  0.1]
```

```
np.maximum(x, y)
```

```
[ 0.2  0.5  0.3  0.6]
```

```
np.argmax(x)
```

```
3
```

minimum, maximum –
поэлементные минимум и
максимум

argmax, argmin – индекс
максимального /
минимального эл-та

Кумулятивные функции

```
x = np.array([2, 1, 2, 0, 0, 4])
```

```
x.sum(), np.sum(x)  
(9, 9)
```

```
x.cumsum()  
[2 3 5 5 5 9]
```

```
x.prod(), np.prod(x)  
(0, 0)
```

```
x.cumprod()  
[2 2 4 0 0 0]
```

```
np.unique(x)  
[0 1 2 4]
```

```
u, i = np.unique(x,  
return_counts=True)  
[0 1 2 4] [2 1 2 1]
```

cumsum – кумулятивная сумма,

cumprod – кумулятивное
произведение (можно делать по
отдельным размерностям в
матрице)

unique – уникальные элементы
(устранение повторов)

Опреации с «нанами»

```
x = np.array([2, 1, 2, np.nan, np.nan, 4])
```

```
np.nansum(x)
```

```
9.0
```

```
np.nanprod(x)
```

```
16.0
```

Сортировка

```
x = np.random.randint(0, 6, 10)
```

```
[0 1 3 1 3 5 3 5 2 2]
```

```
np.sort(x)
```

```
[0 1 1 2 2 3 3 3 5 5]
```

```
np.argsort(x)
```

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

```
np.partition(x, 3) # <3 слева, >3 справа
```

```
[0 1 1 2 2 5 3 5 3 3]
```

Множества **new**

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

```
np.union1d(a, b)  
[1 2 3 4]
```

```
np.intersect1d(a, b)  
[1 4]
```

```
np.unique(a)  
[1 3 4]
```


Матрицы

```
b = np.array([[0, 1, 2], [3, 4, 5]])
```

	0	1	2
0	0	1	2
1	3	4	5

```
# такая же матрица!
```

```
a = np.arange(6).reshape(2, 3)
```

```
b.shape, b.ndim, len(b), a.shape,  
a.ndim, len(a)
```

```
((2, 3), 2, 2, (2, 3), 2, 2)
```

```
x = np.arange(6).reshape(2, 3)
```

```
x.T.ravel()
```

```
array([0, 3, 1, 4, 2, 5])
```

Создание матрицы

reshape – изменение размеров
(можно какую-нибудь
размерность пометить -1)

T – транспозиция

ravel – векторизация (просто вид)

flatten – *= (копия)
(flattening)

Матрицы

```
np.ones((1,2,4,1,3,1)).squeeze().shape  
(2, 4, 3)
```

убираем 1-размерности
new

```
np.expand_dims(np.ones((2,3)),axis=1).shape  
(2, 1, 3)
```

добавляем 1-размерности
new

np.newaxis играет ту же
роль

```
np.eye(3)  
[[ 1.  0.  0.]  
 [ 0.  1.  0.]  
 [ 0.  0.  1.]]
```

Матрицы

Диагонализация бывает двух видов:

матрица → вектор,

вектор → матрица

```
x = np.diag([1, 3, 2])
```

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

Матрицы

```
np.triu(np.ones((5, 5)))
```

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

```
np.tril(np.ones((5, 5)), k=-2)
```

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

```
np.triu(np.ones((5, 5)).T, k=1)
```

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

**Транспозиция – просто способ
просмотра**

Индексация

```
x = np.arange(12).reshape(3, 4)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
x[2, 3]
```

```
11
```

```
x[:, 0]
```

```
[0 4 8]
```

```
x[1, :]
```

```
[4 5 6 7]
```

```
x[-2:, -2:]
```

```
[[ 6  7]
 [10 11]]
```

```
x[:, :2, ::2])
```

```
[[ 0  2]
 [ 8 10]]
```

Изменение размеров матрицы

```
x = np.zeros(12)  
x.reshape(2, 6)
```

```
array([[0., 0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0., 0.]])
```

```
x.reshape(4, -1)
```

```
array([[0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.]])
```

Разные размеры матриц

```
x = np.arange(3)
```

```
y = np.arange(3)[:, np.newaxis]
```

```
z = np.arange(3)[np.newaxis,:]
```

```
print (x.shape, y.shape, z.shape)
```

```
(3,) (3, 1) (1, 3)
```

```
x + x, x + y, x + z
```

```
(array([0, 2, 4]),
```

```
array([[0, 1, 2],  
       [1, 2, 3],  
       [2, 3, 4]]),
```

```
array([[0, 2, 4]]))
```

Разные максимумы

```
x = np.array([[1, 4], [3, 5]])
```

```
x.min()
```

```
1
```

```
x.min(axis=0)
```

```
[1 4]
```

```
x.min(axis=1)
```

```
[1 3]
```

```
x.min(axis=0, keepdims=True)
```

```
[[1 4]]
```

```
x.min(axis=1, keepdims=True)
```

```
[[1]
```

```
[3]]
```

Аналогично с функциями **max**, **sum** (сумма), **prod** (произведение), **mean** (среднее), **median** (медиана), **std** (стандартное отклонение), **argmax**, **argmin**.

Можно выполнять по отдельным размерностям, сохраняя размерность матрицы.

Функции имеют направление

```
x = np.arange(12).reshape(3, 4)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
np.max(x)
```

```
11
```

```
np.max(x, axis=0)
```

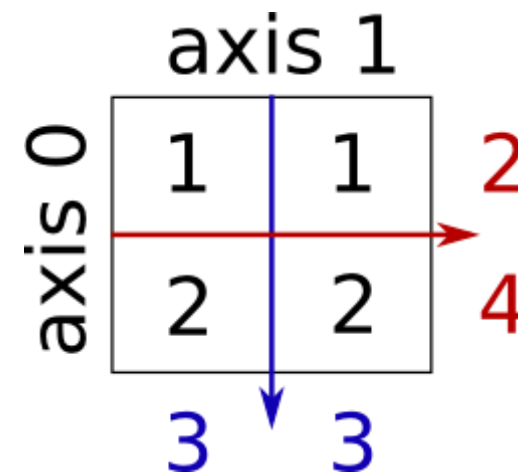
```
[ 8  9 10 11]
```

```
np.max(x, axis=1)
```

```
[ 3  7 11]
```

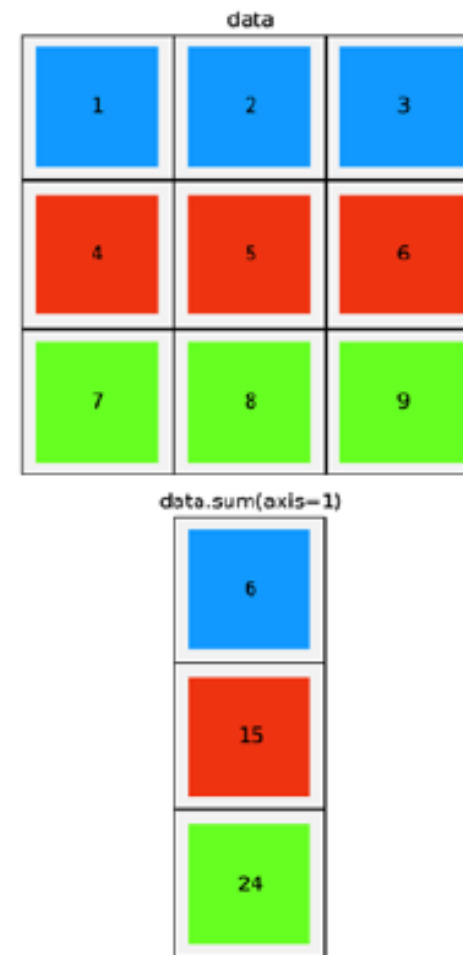
```
np.maximum(x, 5)
```

```
[[ 5  5  5  5]
 [ 5  5  6  7]
 [ 8  9 10 11]]
```



<http://www.scipy-lectures.org/intro/numpy/operations.html>

Функции имеют направление



кн. Numerical Python

Операции над матрицами аналогично операциям над векторами

```
np.ones((2, 2)) + np.eye(2)
```

```
[[ 2.  1.]  
 [ 1.  2.]]
```

```
np.ones((2, 2)) ** 2
```

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

```
np.ones((2, 2)) * np.ones((2, 2))
```

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

```
np.dot(np.ones((2, 2)), np.ones((2, 2)))
```

```
[[ 2.  2.]  
 [ 2.  2.]]
```

**Операция `*` –
поэлементное умножение,
чтобы сделать матричное,
используйте `dot`.**

Broadcasting – операции с матрицами разных размеров

```
np.ones((3, 3)) + np.arange(3)
array([[ 1.,  2.,  3.],
       [ 1.,  2.,  3.],
       [ 1.,  2.,  3.]])
```

```
x = np.ones((3, 3))
```

```
x[0] = 0
```

```
x
array([[ 0.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

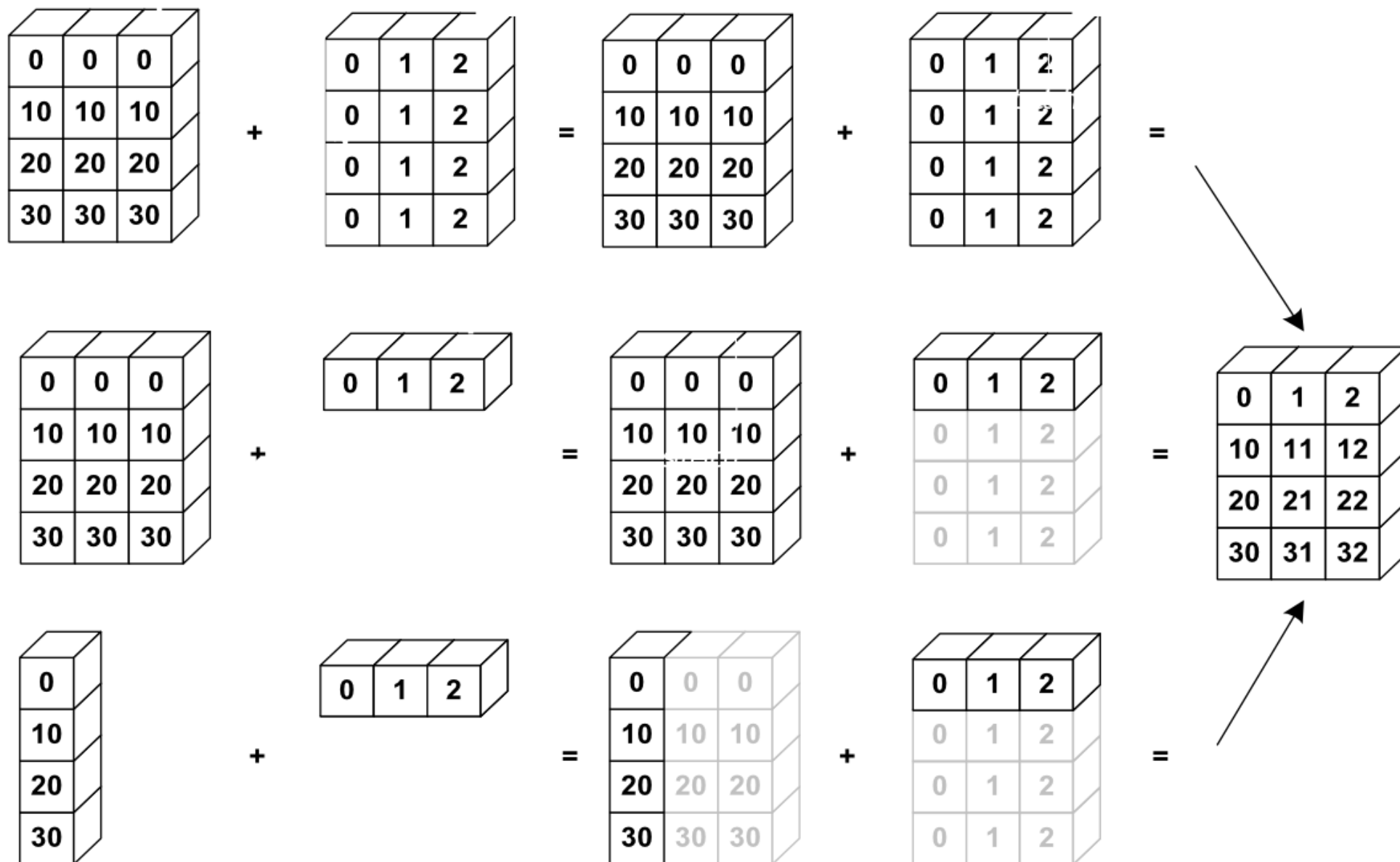
```
x = np.array([1, 5, 0, 2])
```

```
np.abs(x - x[:, np.newaxis])
array([[0, 4, 1, 1],
       [4, 0, 5, 3],
       [1, 5, 0, 2],
       [1, 3, 2, 0]])
```

Ещё мы встречали np.newaxis

Матрица попарных расстояний

Broadcasting



Конкатенация

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

```
np.concatenate([x, x+1])
```

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

```
np.concatenate([x, x+1], axis=1)
```

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

```
np.vstack([x, x + 1])
```

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

```
np.hstack([x, x + 1])
```

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

**dstack – в третьем
направлении**

Расщепление

```
x = np.arange(10).reshape(2, 5)
a, b, c = np.hsplit(x, [1, 4])
# есть split, vsplit
```

```
print (a)
```

```
[[0]
 [5]]
```

```
print (b)
```

```
[[1 2 3]
 [6 7 8]]
```

```
print (c)
```

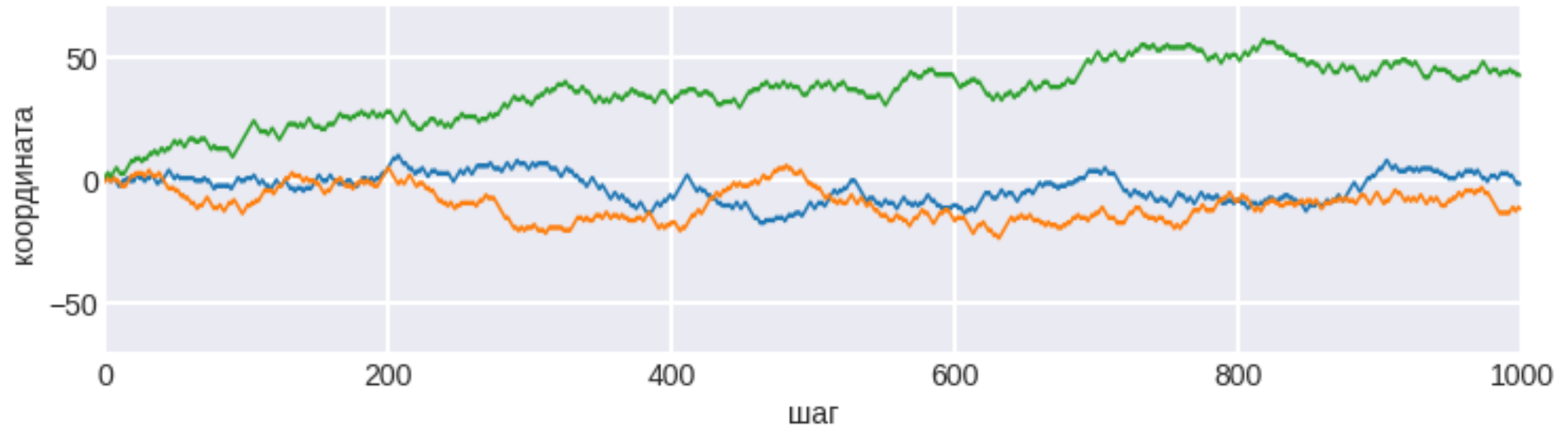
```
[[4]
 [9]]
```

Многомерные матрицы

```
a = np.arange(4*3*2).reshape(4, 3, 2)
print(a.shape)
(4, 3, 2)
```

```
a.reshape(6, -1)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```


Модель броуновского движения



```
def random_walk(n=1000, p=0.5):  
    x = 2*(np.random.rand(n) < p) - 1  
    return (x.cumsum())
```

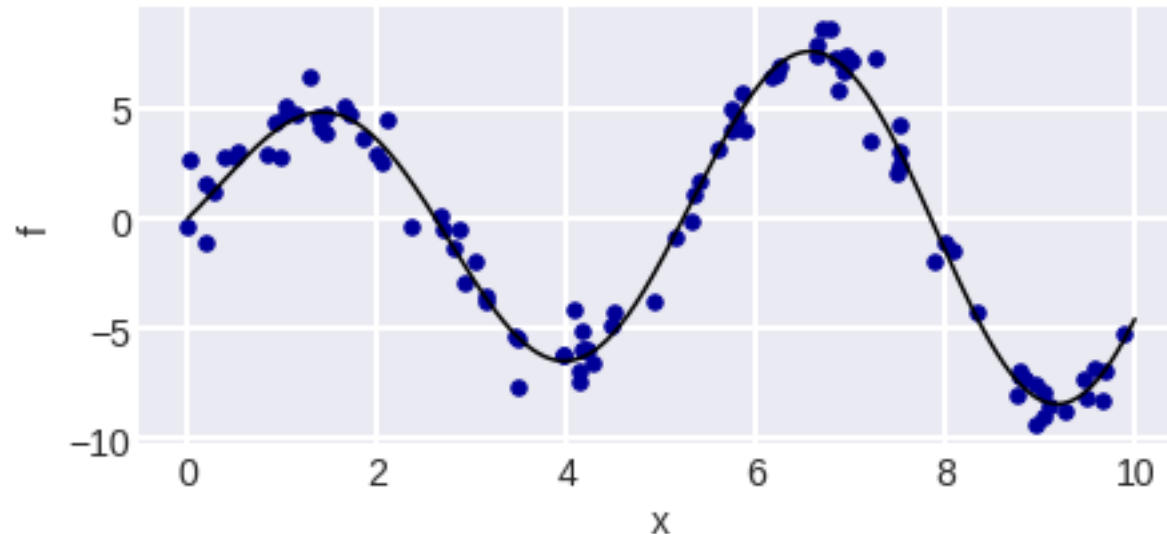
```
plt.figure(figsize=(12, 3))  
plt.plot(random_walk())  
plt.plot(random_walk())  
plt.plot(random_walk())  
plt.xlim([0, 1000])  
plt.ylim([-70, 70])  
plt.xlabel('шаг')  
plt.ylabel('координата')  
plt.grid(lw=2)
```

SciPy

– модуль для научных вычислений

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

Пример: интерполяция



```
from scipy import optimize
```

```
xx = np.linspace(0, 10, num=1001)
```

```
np.random.seed(1)
```

```
x = 10 * np.random.rand(100)
```

```
y = 3.5 * np.sin(1.2 * x) * np.log1p(1.3 + x) + np.random.normal(size=100)
```

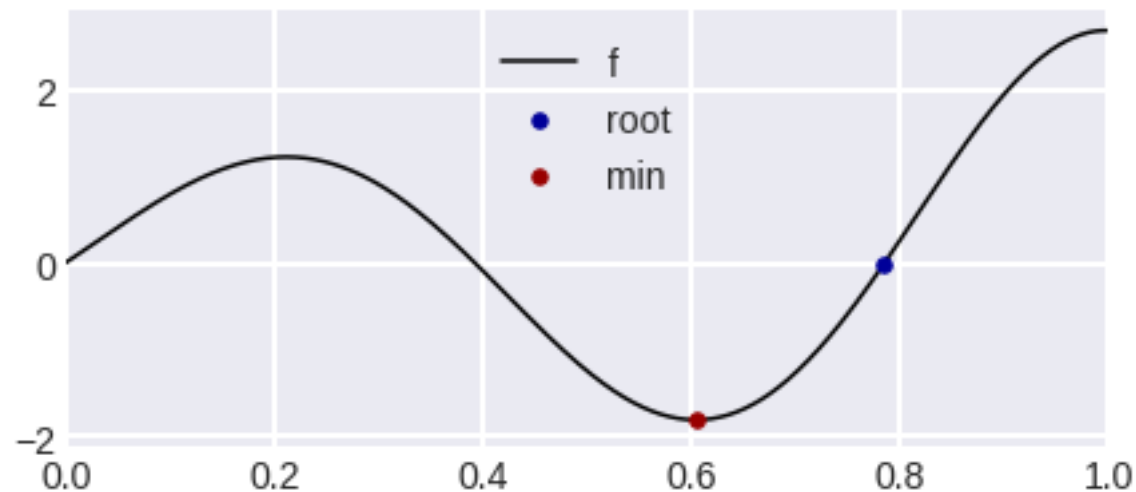
```
def f(x, a, b, c):
```

```
    return a * np.sin(b * x) * np.log1p(c + x)
```

```
params, params_covariance = optimize.curve_fit(f, x, y, p0=[1, 1, 1])
```

```
print(params)
```

Пример: корни и минимум



```
from scipy import optimize
```

```
def f(x):  
    return (np.sin(8 * x) * np.exp(x))
```

```
x = np.linspace(0, 1, 1001)  
y = f(x)
```

```
root = optimize.root(f, x0=0.8)  
minf = optimize.minimize(f, x0=0.8, bounds=((0.2, 1), ))
```

Как выводится графика

```
plt.figure(figsize=(7, 3))
plt.plot(x, y, c='black', label='f', zorder=1)
plt.scatter([root.x], [f(root.x)], 30, label='root', zorder=2,
            color='#000099')
plt.scatter([minf.x], [f(minf.x)], 30, label='min', zorder=2,
            color='#990000')
plt.legend()
plt.xlim([0, 1])
plt.grid(lw=2)
```

Эффективное программирование¶

1) векторизация (Vectorizing for loops)

```
x = np.zeros(1000000)
```

```
%%time  
for i in range(1000000):  
    x[i] += 1
```

CPU times: user 368 ms

```
%%time  
x += 1
```

CPU times: user 20 ms

**Можно делать векторизованные
версии функции**

```
def f(x):  
    if x > 0.5:  
        x += 1  
    else:  
        x -= 1  
    return (x)
```

```
f_v = np.vectorize(f)
```

```
%%time  
x = f_v(x)
```

CPU times: user 188 ms

Можно проще и быстрее;)

Эффективное программирование¶

2) использовать Broadcasting

3) встроенные операции

```
a = np.zeros(1000000)
```

```
%timeit global a; a = 0*a
```

1.75 ms \pm 178 μ s per loop (mean \pm std. dev. of 7 runs, 100 loops each)

```
%timeit global a; a *= 0
```

242 μ s \pm 112 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

4) не копировать без необходимости по умолчанию питон экономит память

Эффективное программирование¶

5) обращение к близким объектам быстрее (кэширование)

```
A = np.zeros((4000, 4000))
```

```
%timeit A.sum(axis=1)
```

5.52 ms \pm 67.2 μ s per loop (mean \pm std. dev. of 7 runs, 100 loops each)

```
%timeit A.sum(axis=0)
```

8.45 ms \pm 43.8 μ s per loop (mean \pm std. dev. of 7 runs, 100 loops each)

6) используйте скомпилированный код

например, Cython

The NumPy array: a structure for efficient numerical computation

<https://hal.inria.fr/inria-00564007/en>

Разреженные матрицы

```
values = [1, 2, 3, 4]
rows = [0, 1, 2, 3]
cols = [1, 3, 2, 0]
A = sp.coo_matrix((values, (rows, cols)), shape=[4, 4])
<4x4 sparse matrix of type '<class 'numpy.int64'>'
  with 4 stored elements in COOrdinate format>
```

```
A.shape, A.ndim, A.nnz, A.data
((4, 4), 2, 4, array([1, 2, 3, 4]))
```

```
A.toarray()
```

```
array([[0, 1, 0, 0],
       [0, 0, 0, 2],
       [0, 0, 3, 0],
       [4, 0, 0, 0]])
```

```
A.tocsr()
```

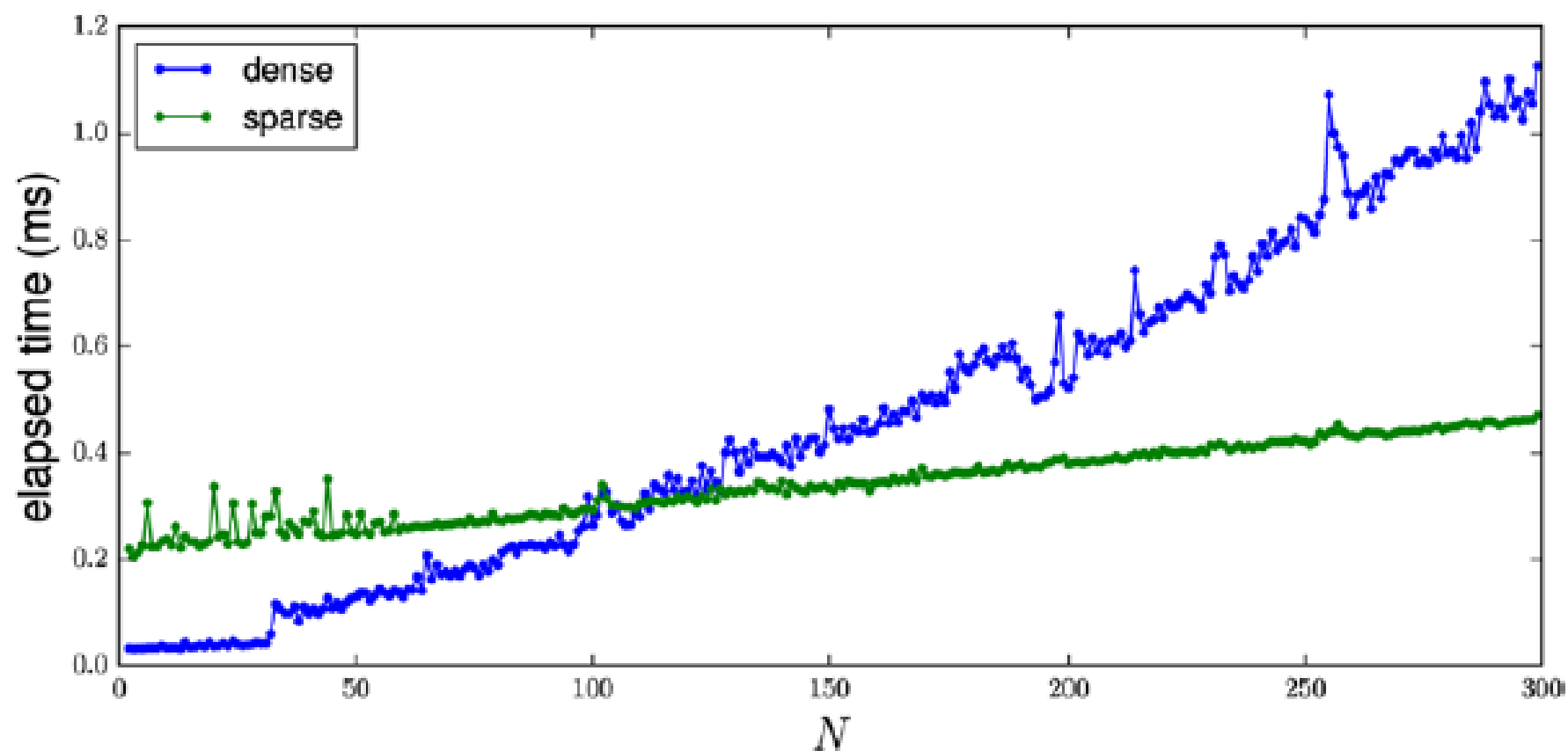
```
<4x4 sparse matrix of type '<class 'numpy.int64'>'
  with 4 stored elements in Compressed Sparse Row format>
```

Разреженные матрицы

```
n=5
A = sp.eye(n, k=1) - 2 * sp.eye(n) + sp.eye(n, k=-1)
A.toarray()
array([[ -2.,   1.,   0.,   0.,   0.],
       [  1.,  -2.,   1.,   0.,   0.],
       [  0.,   1.,  -2.,   1.,   0.],
       [  0.,   0.,   1.,  -2.,   1.],
       [  0.,   0.,   0.,   1.,  -2.]])

b = np.arange(5)
x = sp.linalg.spsolve(A, b)
x.round(2)
array([-3.33, -6.67, -9.   , -9.33, -6.67])
```

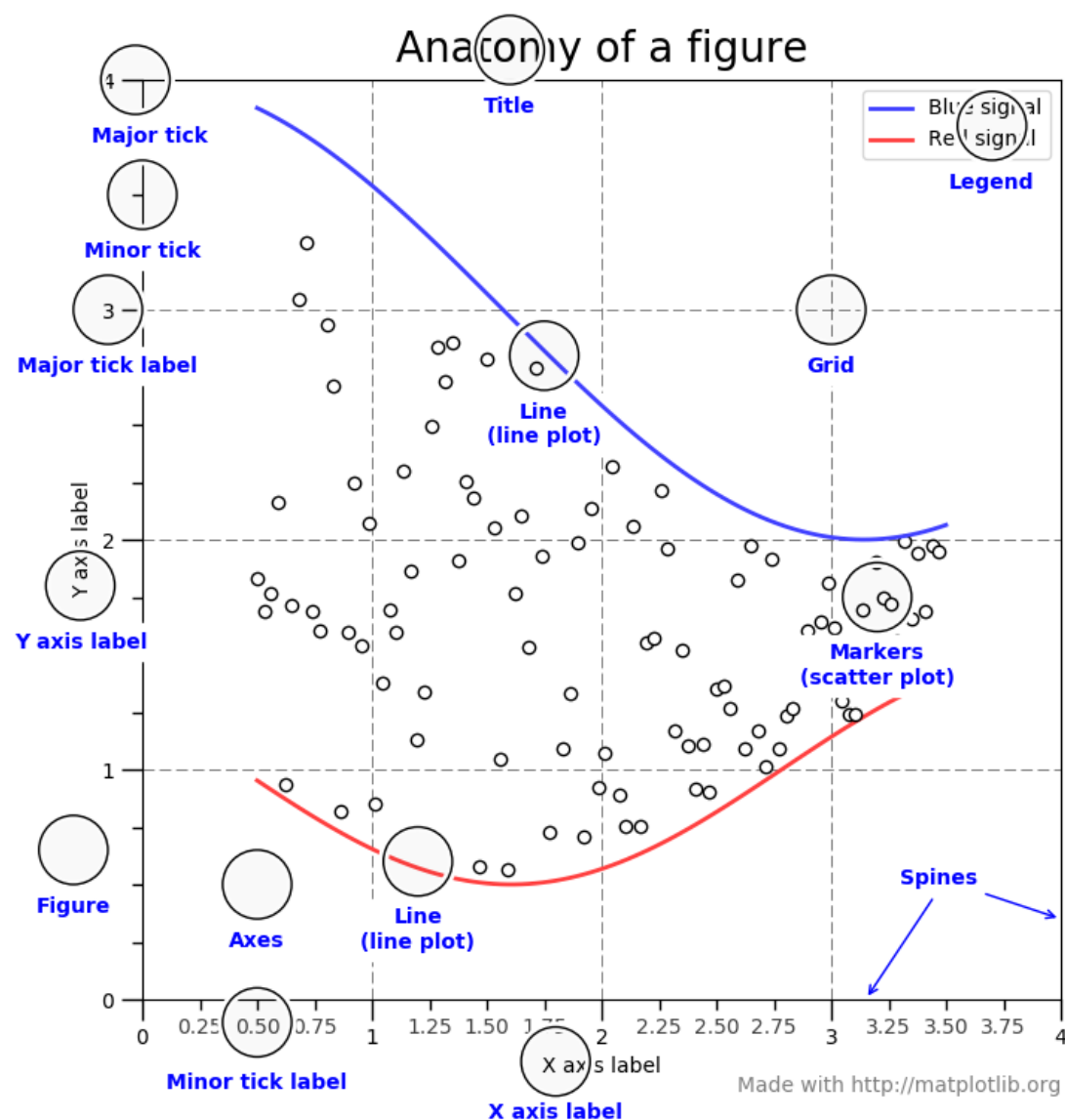
Разреженные матрицы



как время зависит от размеров

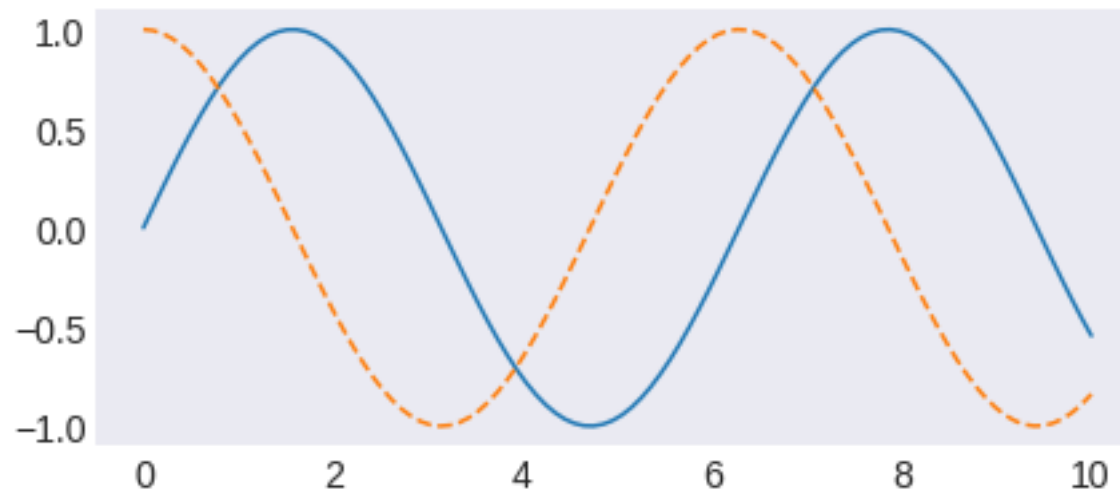
- Разреженные матрицы
- Индексация по элементам

Графика Matplotlib



<https://matplotlib.org/examples/showcase/anatomy.html>

Простой график

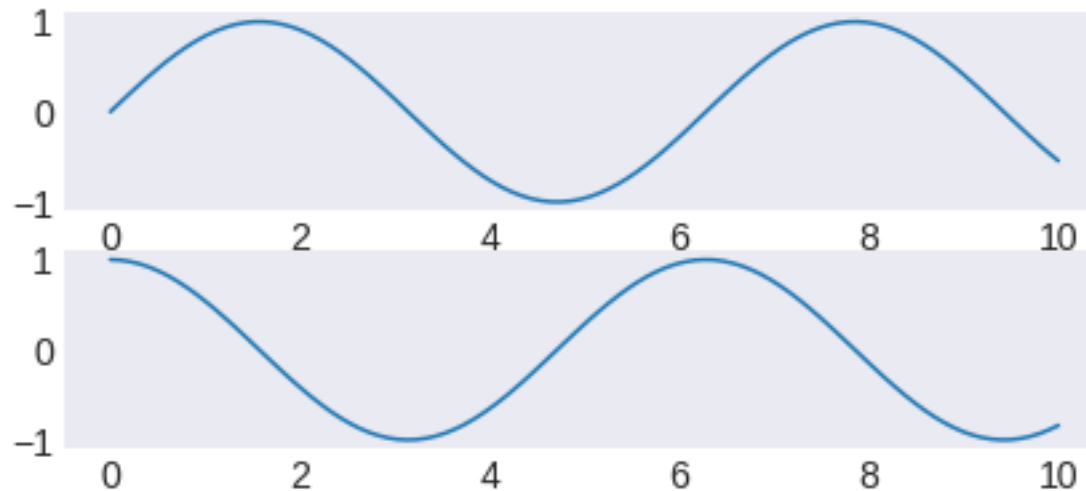


```
import numpy as np
x = np.linspace(0, 10, 100)

fig = plt.figure(figsize=(7, 3))
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
plt.show() # если в юпитере один график, то можно не писать
```

```
fig.savefig('my_figure.png') # сохранение
```

Несколько графиков



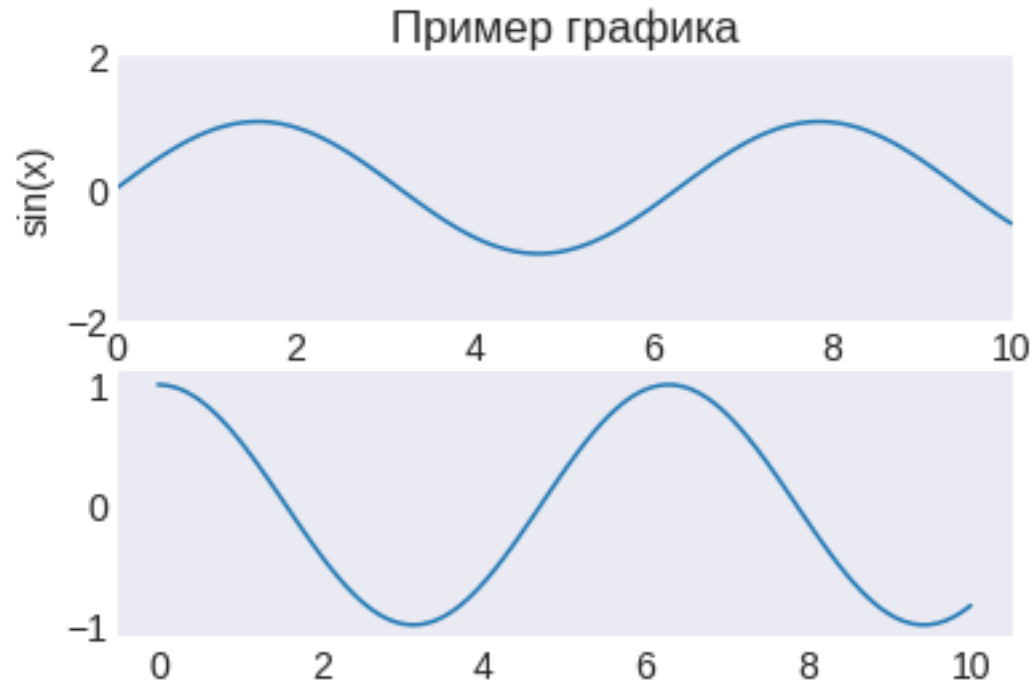
несколько графиков - первый способ

```
plt.figure(figsize=(7, 3)) # create a plot figure
```

```
plt.subplot(2, 1, 1) # (строк, столбцов, текущий)  
plt.plot(x, np.sin(x))
```

```
plt.subplot(2, 1, 2)  
plt.plot(x, np.cos(x));
```

Несколько графиков



второй способ

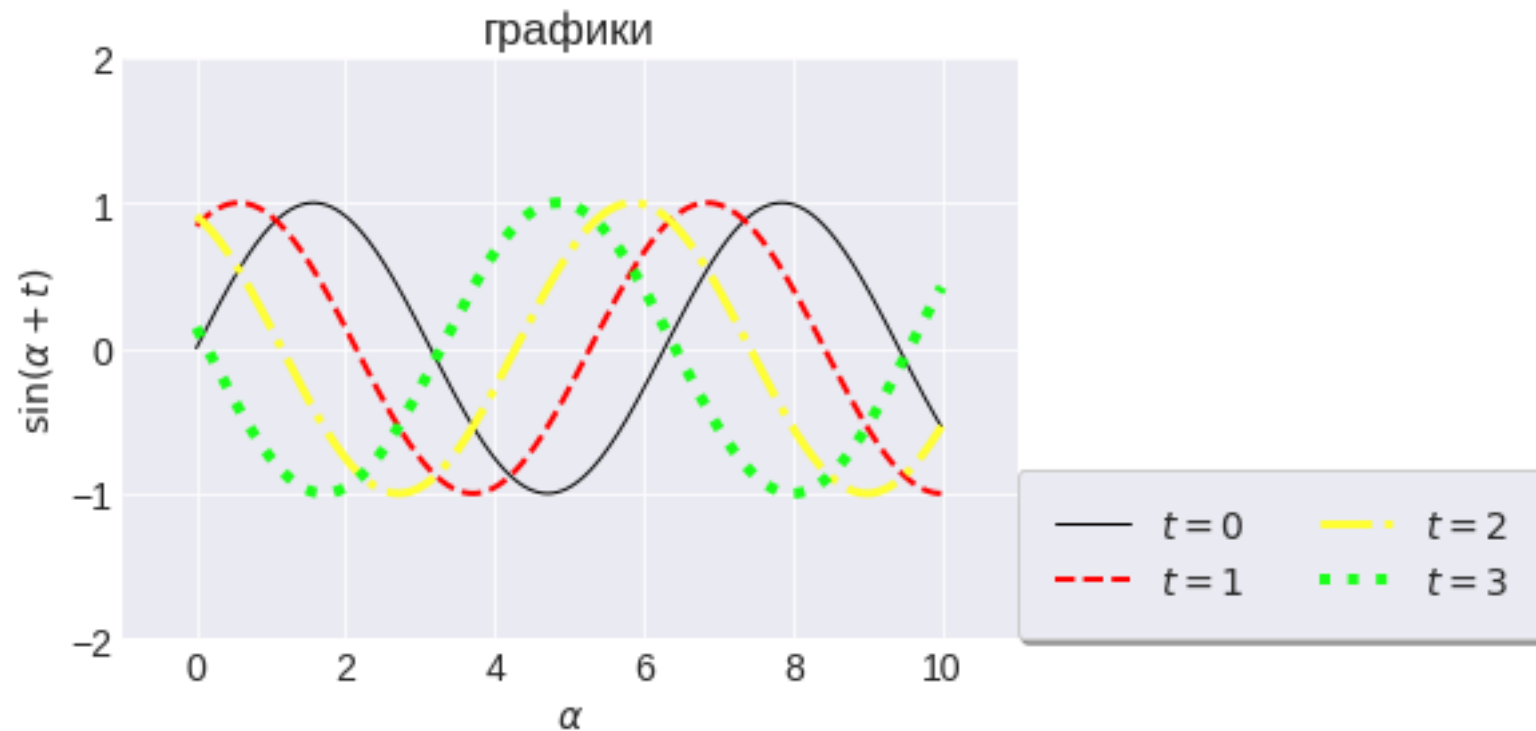
```
fig, ax = plt.subplots(2)
```

```
ax[0].plot(x, np.sin(x))
```

```
ax[1].plot(x, np.cos(x));
```

```
ax[0].set(xlim=(0, 10), ylim=(-2, 2),  
          xlabel='x', ylabel='sin(x)',  
          title='Пример графика');
```


Сложный график



Сложный график

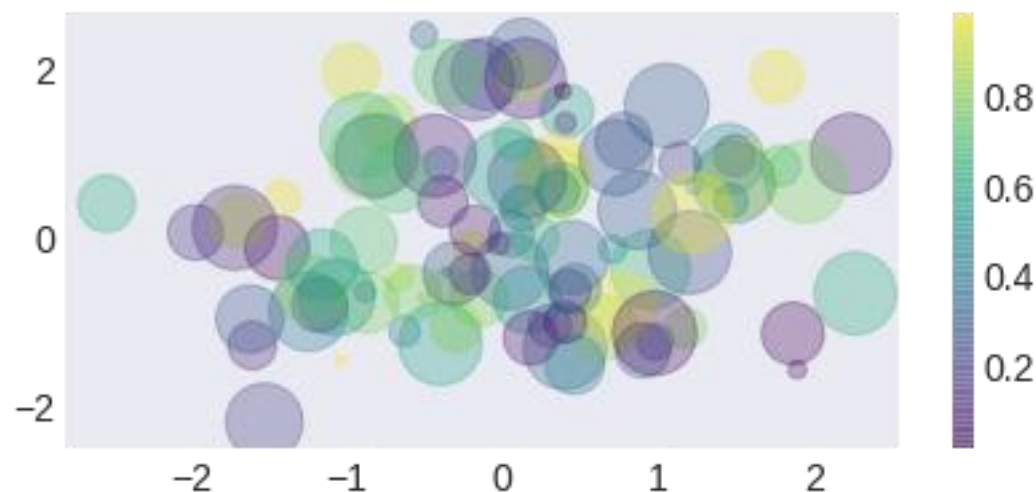
```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x), c='black', linestyle='-',
        lw=1, label='$t=0$');
ax.plot(x, np.sin(x + 1), c='r', linestyle='--',
        lw=2, label='$t=1$');
ax.plot(x, np.sin(x + 2), c='#FFFF33', linestyle='-.',
        lw=3, label='$t=2$');
ax.plot(x, np.sin(x + 3), c=(0.1, 1.0, 0.1), linestyle=':',
        lw=4, label='$t=3$');

plt.xlim([-1, 11])
plt.ylim([-2, 2])

plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\sin(\alpha + t)$');
plt.title('графики');
plt.legend(loc=(1, 0), frameon=True, ncol=2,
          fancybox=True, framealpha=1, shadow=True, borderpad=1)
plt.grid()
```

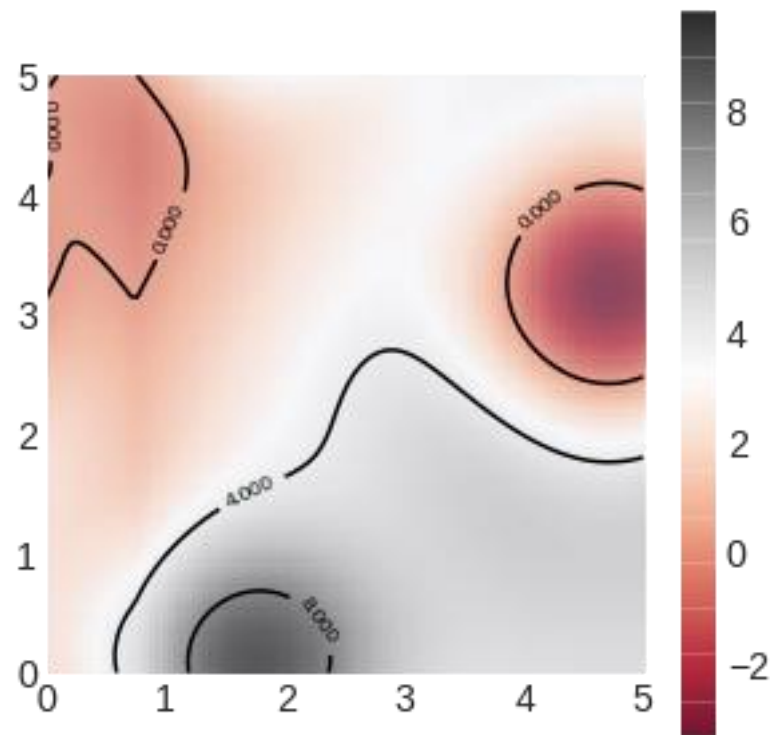
Диаграмма рассеивания



```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.figure(figsize=(7, 3))
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();
```

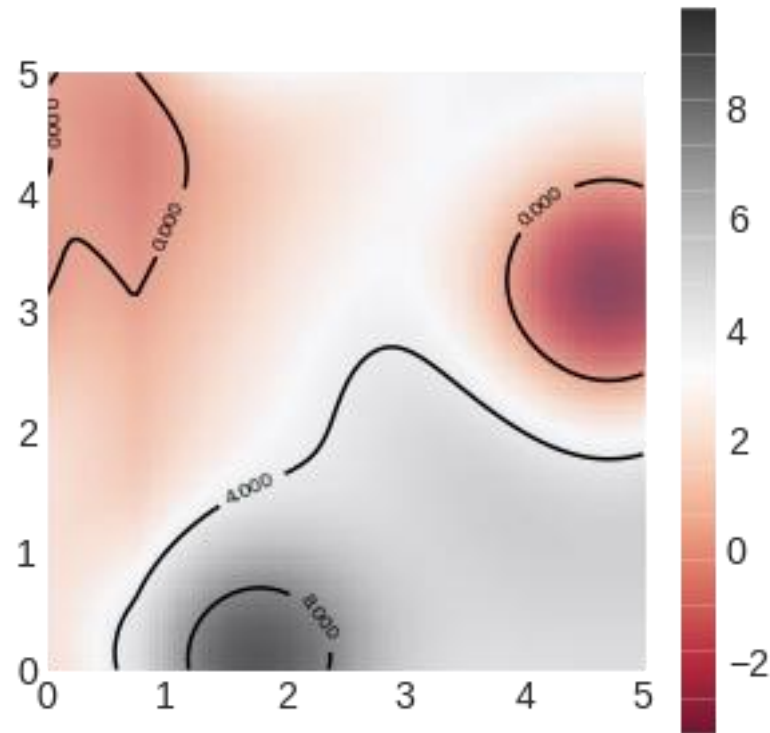
Линии уровня функции



```
def f(x, y):  
    z = (np.sin(x) + np.cos(y)) ** 3 + np.abs(np.cos(x) - x) + (np.sin(y))  
    return z
```

```
x = np.linspace(0, 5, 100)  
y = np.linspace(0, 5, 100)  
X, Y = np.meshgrid(x, y)  
Z = f(X, Y)
```

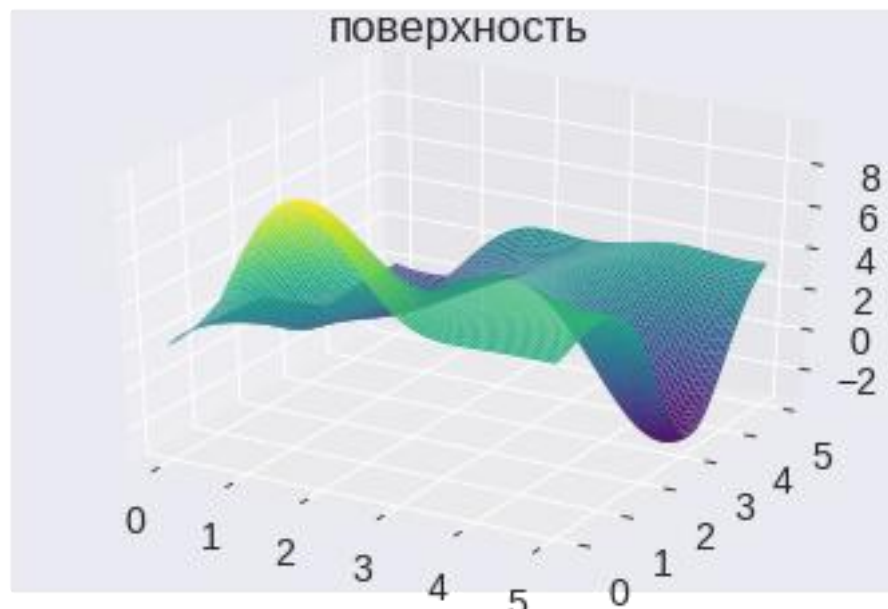
Линии уровня функции



```
plt.figure(figsize=(5, 5))
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)

plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
           cmap='RdGy', alpha=0.7)
plt.colorbar();
```

Поверхность



```
from mpl_toolkits import mplot3d # import mplot3d
# mplot3d.enable_notebook()

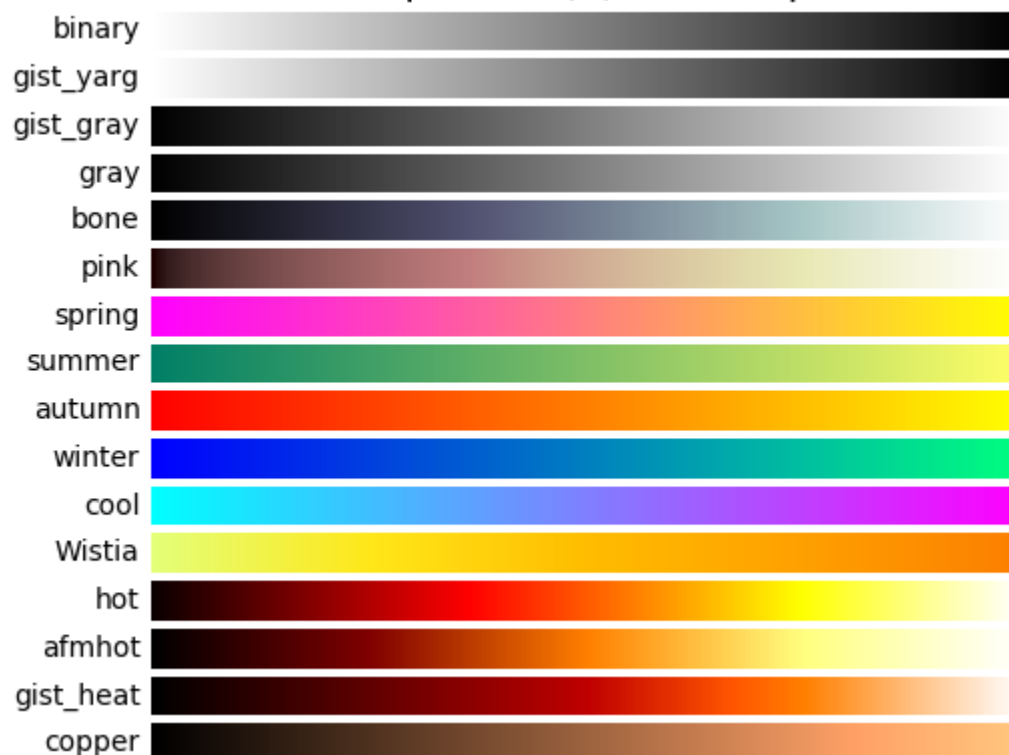
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
               cmap='viridis', edgecolor='none')
ax.set_title('поверхность');
```

CMAP

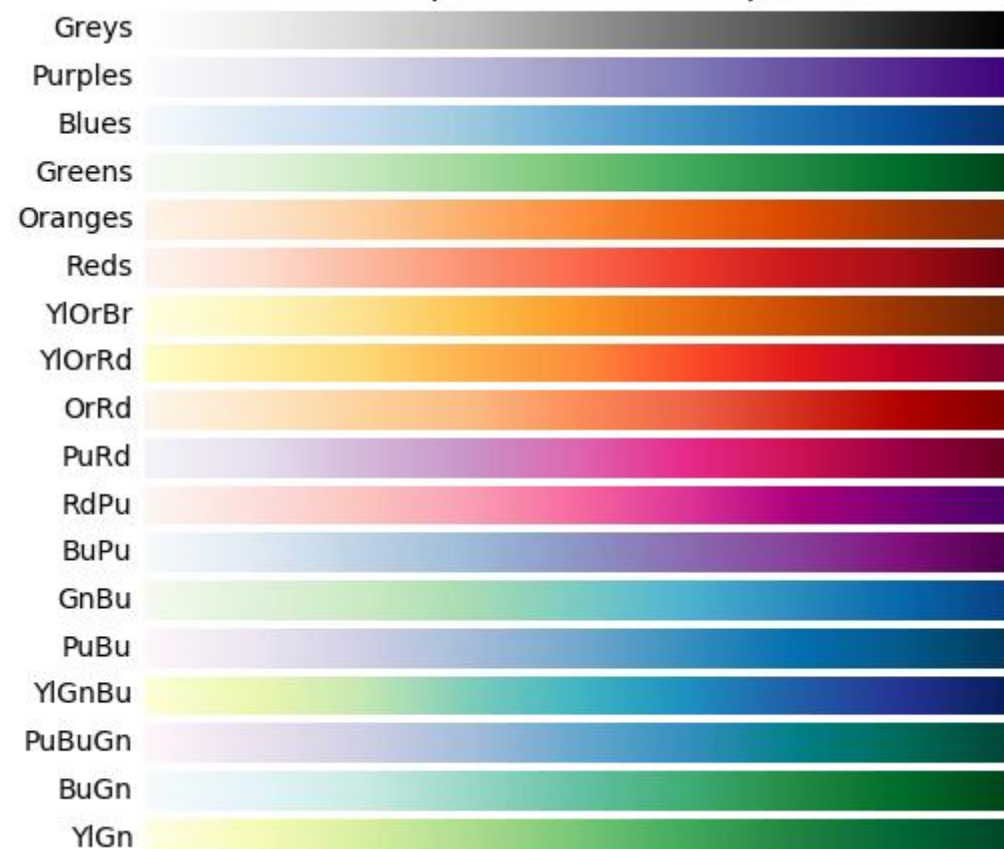
Perceptually Uniform Sequential colormaps



Sequential (2) colormaps

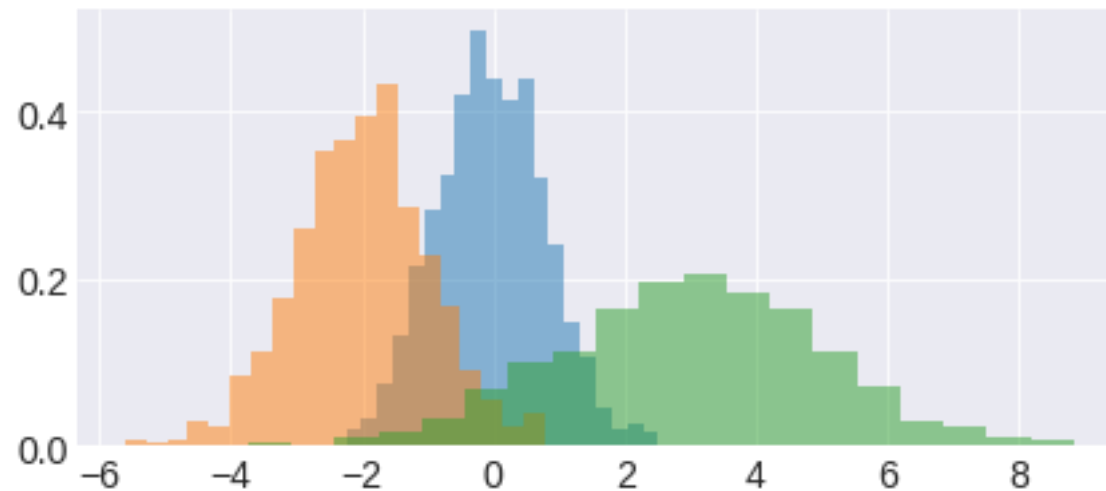


Sequential colormaps



https://matplotlib.org/examples/color/colormaps_reference.html

Плотность распределения

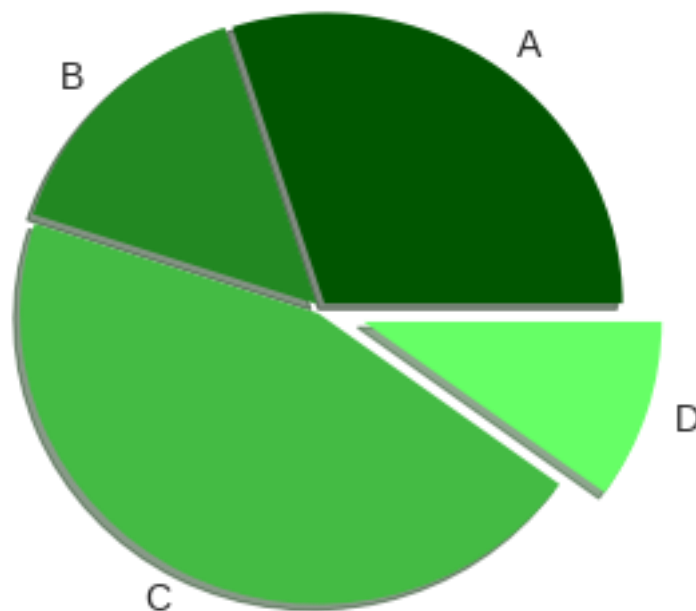


```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
```

```
kwargs = dict(bins=20, histtype='stepfilled', alpha=0.5, normed=True)
```

```
plt.figure(figsize=(7, 3))
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
plt.grid()
```

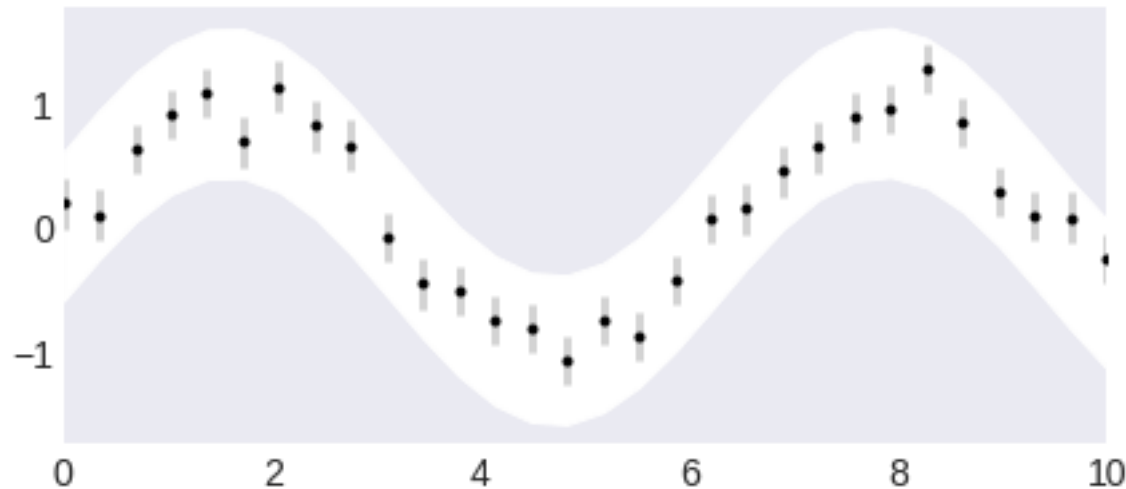

Пирог



```
fracs = [30, 15, 45, 10]
colors = ['#005500', '#228822', '#44BB44', '#66FF66']

fig, ax = plt.subplots(figsize=(5, 5))
pie = ax.pie(frac, colors=colors,
             explode=(0.02, 0.02, 0.02, 0.15), shadow=True,
             labels=['A', 'B', 'C', 'D'])
```

Заливка и ошибки



```
x = np.linspace(0, 10, 30)
dy = 0.2
y = np.random.normal(np.sin(x), dy)
y1 = np.sin(x) + 3*dy
y2 = np.sin(x) - 3*dy

plt.figure(figsize=(7, 3))
plt.fill_between(x, y1, y2, color='white')
plt.errorbar(x, y, dy, fmt='.k', ecolor='lightgray', elinewidth=3)
plt.xlim([0, 10])
```

Литература

- **Scipy Lecture Notes**

Хороший курс – пройти

<http://www.scipy-lectures.org/index.html>

- **Официальный сайт**

<http://www.numpy.org/>

<https://docs.scipy.org/doc/>

- **Хорошие репозитории**

<https://github.com/jrjohansson>

- **Быстрый курс**

<https://github.com/rpmuller/PythonCrashCourse>

<https://github.com/ipython-books/minibook-2nd-code>

- **П. Шабанов «Научная графика в Питон»**

https://github.com/whitehorn/Scientific_graphics_in_python