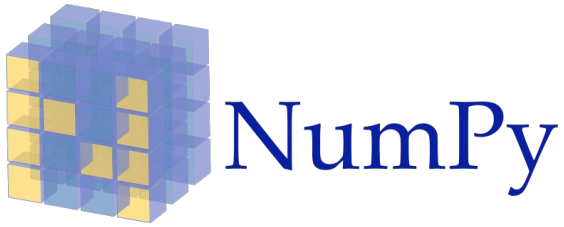
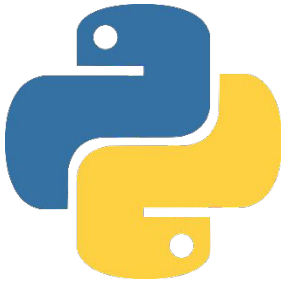
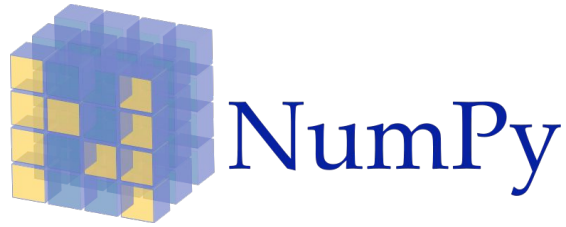




Section XIII: Библиотека NumPy

Section XIII: Библиотека NumPy



Что такое NumPy?

- NumPy – библиотека для «научных» вычислений на Python.
- Основной (core) объект – `ndarray object` – инкапсуляция многомерного массива однородных данных.

- NumPy массивы имеют фиксированный размер
- Элементы массива однотипны – занимают одинаковое место в памяти
- Продвинутые математические операции на больших объемах данных
- NumPy – стандарт хранения данных во многих Python модулях

https://numpy.org/doc/stable/

Section XIII: Библиотека NumPy

А что под капотом? Кончено же C!

```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```

```
for (i = 0; i < rows; i++): {
  c[i] = a[i]*b[i];
}
```

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

```
c = a * b
```

Что дает векторизованный код?

- Лаконичность и удобство чтения
- Меньше строк – меньше багов
- Стандартная математическая нотация (broadcast – главная фишка)
- Pythonic code

```
python -m pip install --user numpy
```

Section XIII: Библиотека NumPy

Основы

```
[1, 2, 1]
```
1 axis, 3 elements

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```
2 axes

`ndarray` – класс – многомерный массив

a.k.a. просто `array`

`numpy.array` – не то же самое, что и стандартный в Python `array.array`!

Наиболее важные атрибуты `ndarray`:

- `ndarray.ndim`
- `ndarray.shape`
- `ndarray.size`
- `ndarray.dtype`
- `ndarray.itemsize`
- `ndarray.data`

Section XIII: Библиотека NumPy
Примеры

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
```

```
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

Section XIII: Библиотека NumPy

Создание массива

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> a = np.array(1,2,3,4)    # WRONG
>>> a = np.array([1,2,3,4])  # RIGHT
```

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

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Section XIII: Библиотека NumPy

Создание шаблона (initial placeholder content)

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                # dtype can also be specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> np.empty( (2,3) )                                 # uninitialized, output may vary
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

Section XIII: Библиотека NumPy

Заполнение последовательностями

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )                 # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

```
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )                 # 9 numbers from 0 to 2
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
>>> x = np.linspace( 0, 2*pi, 100 )        # useful to evaluate function at lots of points
>>> f = np.sin(x)
```

See also:
array, zeros, zeros_like, ones, ones_like, empty, empty_like, arange, linspace, numpy.random.RandomState.rand, numpy.random.RandomState.randn, fromfunction, fromfile

Section XIII: Библиотека NumPy

Печать массивов

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = np.arange(24).reshape(2,3,4)         # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Layout:

- Последняя ось – слева направо
- Со второй до последней – сверху вниз
- Первая – сверху вниз на отдельных строках

Section XIII: Библиотека NumPy

Печать массивов

```
>>> print(np.arange(10000))
[   0    1    2 ..., 9997 9998 9999]
>>>
>>> print(np.arange(10000).reshape(100,100))
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

```
>>> np.set_printoptions(threshold=sys.maxsize)       # sys module should be imported
```

Section XIII: Библиотека NumPy

Базовые операции

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

```
>>> A = np.array( [[1,1],
...             [0,1]] )
>>> B = np.array( [[2,0],
...             [3,4]] )
>>> A * B                        # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B                        # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                     # another matrix product
array([[5, 4],
       [3, 4]])
```

Операции применяются поэлементно (elementwise). Результат всегда сохраняется в новый массив.

Section XIII: Библиотека NumPy

In place замена (не обновление)

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.417022  ,  3.72032449,  3.00011437],
       [ 3.30233257,  3.14675589,  3.09233859]])
>>> a += b                  # b is not automatically converted to integer type
Traceback (most recent call last):
  ...
TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same_kin
d'
```

Section XIII: Библиотека NumPy

Upcasting

```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = np.exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
       -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

Section XIII: Библиотека NumPy

Операции типа reduce

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.1862602113776709
>>> a.max()
0.6852195003967595
```

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                    # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                    # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                 # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

Section XIII: Библиотека NumPy

Универсальные функции

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```

See also:
all, any, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cov, cross, cumprod, cumsum, diff, dot, floor, inner, *inv*, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sort, std, sum, trace, transpose, var, vdot, vectorize, where

Section XIII: Библиотека NumPy

Indexing, Slicing and Iterating (1-dim)

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; from start to position 6, exclusive, set every 2nd
element to -1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,   729])
>>> a[ : :-1]                                 # reversed a
array([  729,   512,   343,   216,   125, -1000,    27, -1000,     1, -1000])
>>> for i in a:
...     print(i**(1/3.))
...
nan
1.0
nan
3.0
...
```

Одномерные массивы – аналогичны листам и другим Python последовательностям

Section XIII: Библиотека NumPy

Indexing, Slicing and Iterating (n-dim)

```
>>> def f(x,y):
...     return 10*x+y
...
>>> b = np.fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]                       # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[ : ,1]                        # equivalent to the previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]                      # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Section XIII: Библиотека NumPy

Indexing, Slicing and Iterating (n-dim)

```
>>> b[-1]                                  # the last row. Equivalent to b[-1,:]
array([40, 41, 42, 43])
```

```
>>> c = np.array( [[[  0,  1,  2],               # a 3D array (two stacked 2D arrays)
...                 [ 10, 12, 13]],
...                [[100,101,102],
...                 [110,112,113]]])
>>> c.shape
(2, 2, 3)
>>> c[1,...]                                   # same as c[1,:,:] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[...,2]                                   # same as c[:,:,2]
array([[  2,  13],
       [102, 113]])
```

Section XIII: Библиотека NumPy

Indexing, Slicing and Iterating (n-dim)

```
>>> for row in b:
...     print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

```
>>> for element in b.flat:
...     print(element)
...
0
1
2
3
10
11
...
```

See also:
newaxis, ndenumerate, indices

Section XIII: Библиотека NumPy

Изменение формы массивов

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.shape
(3, 4)
```

```
>>> a.ravel()  # returns the array, flattened
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])
>>> a.reshape(6,2)  # returns the array with a modified shape
array([[ 2.,  8.],
       [ 0.,  6.],
       [ 4.,  5.],
       [ 1.,  1.],
       [ 8.,  9.],
       [ 3.,  6.]])
>>> a.T  # returns the array, transposed
array([[ 2.,  4.,  8.],
       [ 8.,  5.,  9.],
       [ 0.,  1.,  3.],
       [ 6.,  1.,  6.]])
>>> a.T.shape
(4, 3)
>>> a.shape
(3, 4)
```

Section XIII: Библиотека NumPy

Изменение формы массивов. `reshape` и `ndarray.resize`

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.resize((2,6))
>>> a
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

```
>>> a.reshape(3,-1)
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
```

See also:
ndarray.shape, reshape, resize, ravel

Section XIII: Библиотека NumPy

Stacking массивов

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 8.,  8.],
       [ 0.,  0.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

Section XIII: Библиотека NumPy

Stacking массивов

```
>>> from numpy import newaxis
>>> np.column_stack((a,b))     # with 2D arrays
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
>>> a = np.array([4.,2.])
>>> b = np.array([3.,8.])
>>> np.column_stack((a,b))     # returns a 2D array
array([[ 4., 3.],
       [ 2., 8.]])
>>> np.hstack((a,b))           # the result is different
array([ 4., 2., 3., 8.])
>>> a[:,newaxis]               # this allows to have a 2D columns vector
array([[ 4.],
       [ 2.]])
>>> np.column_stack((a[:,newaxis],b[:,newaxis]))
array([[ 4.,  3.],
       [ 2.,  8.]])
>>> np.hstack((a[:,newaxis],b[:,newaxis]))   # the result is the same
array([[ 4.,  3.],
       [ 2.,  8.]])
```

Section XIII: Библиотека NumPy

Конкатенация массивов

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6]])
>>> np.concatenate((a, b), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> np.concatenate((a, b.T), axis=1)
array([[1, 2, 5],
       [3, 4, 6]])
>>> np.concatenate((a, b), axis=None)
array([1, 2, 3, 4, 5, 6])
```

Для массивов размерности 2+:

- `vstack` – стекинг вдоль 1х осей
- `hstack` – стекинг вдоль 2х осей
- `concatenate` – конкатенация вдоль указанной оси

Section XIII: Библиотека NumPy

Разбиение массивов

```
>>> a = np.floor(10*np.random.random((2,12)))
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])
>>> np.hsplit(a,3)   # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
       [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
       [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
       [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))   # Split a after the third and the fourth column
[array([[ 9.,  5.,  6.],
       [ 1.,  4.,  9.]]), array([[ 3.],
       [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])]
```

Section XIII: Библиотека NumPy

Копировать или не копировать?

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
>>> b is a           # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

Not Copy at All

```
>>> def f(x):
...     print(id(x))
...
>>> id(a)                           # id is a unique identifier of an object
148293216
>>> f(a)
148293216
```

Python передает изменяемые (~~mutable~~) объекты как ссылки, поэтому вызов функции не производит копирование.

Section XIII: Библиотека NumPy

Копировать или не копировать?

```
>>> c = a.view()
>>> c is a
False
>>> c.base is a                        # c is a view of the data owned by a
True
>>> c.flags.owndata
False
>>>
>>> c.shape = 2,6                      # a's shape doesn't change
>>> a.shape
(3, 4)
>>> c[0,4] = 1234                      # a's data changes
>>> a
array([[   0,    1,    2,    3],
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```

View или Shallow Copy

Слайсы возвращают view

```
>>> s = a[ : , 1:3]     # spaces added for clarity; could also be written "s = a[:,1:3]"
>>> s[:] = 10           # s[:] is a view of s. Note the difference between s=10 and s[:]=10
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

Section XIII: Библиотека NumPy

Копировать или не копировать?

```
>>> d = a.copy()                          # a new array object with new data is created
>>> d is a
False
>>> d.base is a                           # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

Deep Copy

```
>>> a = np.arange(int(1e8))
>>> b = a[:100].copy()
>>> del a  # the memory of ``a`` can be released.
```

Без `copy()` удалить `a` не удастся

Section XIII: Библиотека NumPy

Обзор методов https://numpy.org/doc/stable/reference/routines.html#routines

Array Creation

arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, *r*, zeros, zeros_like

Conversions

ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat

Manipulations

array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack

Questions

all, any, nonzero, where

Ordering

argmax, argmin, argsort, max, min, ptp, searchsorted, sort

Operations

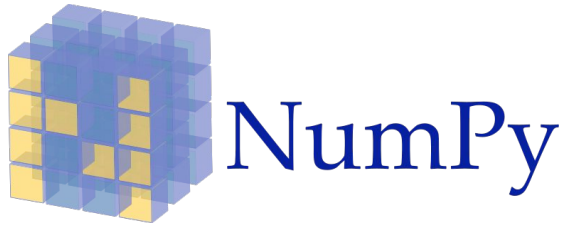
choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum

Basic Statistics

cov, mean, std, var

Basic Linear Algebra

cross, dot, outer, linalg.svd, vdot



Section XIII: Библиотека NumPy

Чем дальше в лес, тем круче код – индексация массивом индексов

```
>>> a = np.arange(12)**2                        # the first 12 square numbers
>>> i = np.array( [ 1,1,3,8,5 ] )               # an array of indices
>>> a[i]                                        # the elements of a at the positions i
array([ 1,  1,  9, 64, 25])
>>>
>>> j = np.array( [ [ 3, 4], [ 9, 7 ] ] )       # a bidimensional array of indices
>>> a[j]                                        # the same shape as j
array([[ 9, 16],
       [81, 49]])
```

Section XIII: Библиотека NumPy

Чем дальше в лес, тем круче код – индексация массивом

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

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

```
>>> a = np.arange(5)
>>> a[[0,0,2]]+=1
>>> a
array([1, 1, 3, 3, 4])
```

Section XIII: Библиотека NumPy

Чем дальше в лес, тем круче код – индексация массивом индексов

```
>>> time = np.linspace(20, 145, 5)                 # time scale
>>> data = np.sin(np.arange(20)).reshape(5,4)      # 4 time-dependent series
>>> time
array([  20.  ,   51.25,   82.5 ,  113.75,  145.  ])
>>> data
array([[ 0.        ,  0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ,  0.6569866 ],
       [ 0.98935825,  0.41211849, -0.54402111, -0.99999021],
       [-0.53657292,  0.42016704,  0.99060736,  0.65028784],
       [-0.28790332, -0.96139749, -0.75098725,  0.14987721]])
>>>
>>> ind = data.argmax(axis=0)                 # index of the maxima
>>> ind
array([2, 0, 3, 1])
>>>
```

Поиск максимумов временного ряда

```
>>> time_max = time[ind]                        # times corresponding to the maxima
>>>
>>> data_max = data[ind, range(data.shape[1])] # => data[ind[0],0], data[ind[1],1]...
>>>
>>> time_max
array([  82.5 ,   20.  ,  113.75,   51.25])
>>> data_max
array([ 0.98935825,  0.84147098,  0.99060736,  0.6569866 ])
>>>
>>> np.all(data_max == data.max(axis=0))
True
```

Section XIII: Библиотека NumPy

Чем дальше в лес, тем круче код – индексация Boolean Arr

```
>>> a = np.arange(12).reshape(3,4)
>>> b = a > 4
>>> b                                          # b is a boolean with a's shape
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]                                       # 1d array with the selected elements
array([ 5,  6,  7,  8,  9, 10, 11])
```

```
>>> a[b] = 0                                   # All elements of 'a' higher than 4 become 0
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
```

Section XIII: Библиотека NumPy

Чем дальше в лес, тем круче код – индексация Boolean Arra

```
>>> a = np.arange(12).reshape(3,4)
>>> b1 = np.array([False,True,True])            # first dim selection
>>> b2 = np.array([True,False,True,False])      # second dim selection
>>>
>>> a[b1,:]                                   # selecting rows
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> a[b1]                                     # same thing
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> a[:,b2]                                   # selecting columns
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
>>>
>>> a[b1,b2]                                  # a weird thing to do
array([ 4, 10])
```

Section XIII: Библиотека NumPy

Линейная алгебра

```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]
```

```
>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])
```

```
>>> y = np.array([[5.], [7.]])
>>> np.linalg.solve(a, y)
array([[-3.],
       [ 4.]])
```

```
>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

```
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j        # matrix product
array([[-1.,  0.],
       [ 0., -1.]])
```

```
>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
```

```
>>> np.trace(u)  # trace
2.0
```

```
>>> np.linalg.eig(j)
(array([ 0.+1.j,  0.-1.j]), array([[ 0.70710678+0.j        ,  0.70710678-0.j        ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
```

Section XIII: Библиотека NumPy

Автоматический решейпин

```
>>> a = np.arange(30)
>>> a.shape = 2,-1,3  # -1 means "whatever is needed"
>>> a.shape
(2, 5, 3)
>>> a
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],
        [24, 25, 26],
        [27, 28, 29]]])
```

Q&A

https://numpy.org/doc/stable/