# NumPy

## Основные возможности

Ф.Я.Халили

МГУ, физический факультет

19 апреля 2008 г.

1. Введение
2. Создание массивов
3. Разрезание и склеивание
4. Покомпонентные операции над массивами
5. Редукция
6. Максимумы, минимумы, сортировка
7. Матричный режим
8. Файловый ввод/вывод

Пакет NumPy обеспечивает:

- новый набор типов данных — многомерные массивы чисел (объект `array`) и
- т.н. *универсальные фунции* (`ufunc`), позволяющие выполять операции над этими массивами в один присест (как в Matlab'е)

Для использования этих возможностей необходимо в начале программы указать:

```
from numpy import *
```

# Типы данных, из которых может состоять массив

| Тип | Размер, байт | Описание |
|---|---|---|
| bool | 1 | Логический тип |
| uint8 | 1 | Целые без знака |
| uint16 | 2 | |
| uint32=uint | 4 | |
| uint64 | 8 | |
| int8 | 1 | Целые со знаком |
| int16 | 2 | |
| int32=int | 4 | |
| int64 | 8 | |
| float32 | 4 | Действительные |
| float64=float | 8 | |
| float96 | 12 | |
| complex64 | 8 | Комплексные |
| complex128=complex | 16 | |
| complex192 | 24 | |

# Явное задание компонентов

```
array(<список компонентов>,[<тип данных>])
```

```
>>> a=array([1,pi,15])
>>> a
array([  1.        ,   3.14159265,  15.        ])
>>> b=array([[1],[2],[15]],complex)
>>> b
array([[  1.+0.j],
       [  2.+0.j],
       [ 15.+0.j]])
>>> c=array([[1,2,3,4],[11,12,13,14],[21,22,23,24]])
>>> c
array([[ 1,  2,  3,  4],
       [11, 12, 13, 14],
       [21, 22, 23, 24]])
```

# Специальные массивы

`zeros(<размерность>,[<тип данных>])`

`ones(<размерность>,[<тип данных>])`

`identity(<размер>,[<тип данных>])`

```
>>> zeros((5))
array([ 0.,  0.,  0.,  0.,  0.])
>>> zeros((3,1),complex)
array([[ 0.+0.j],
       [ 0.+0.j],
       [ 0.+0.j]])
>>> ones((2,4),int)
array([[1, 1, 1, 1],
       [1, 1, 1, 1]])
>>> identity(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
```

# Диапазоны

```
arange(<от>,<до>,<шаг>,[<тип данных>])
linspace(<первый>,<последний>,<размер>)
logspace(<первый>,<последний>,<размер>)
```

```
>>> arange(0,1,0.2)
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
>>> arange(-5,5,2)
array([-5, -3, -1,  1,  3])
>>> arange(-5,5,2,float)
array([-5., -3., -1.,  1.,  3.])
>>> linspace(-1,1,6)
array([-1. , -0.6, -0.2,  0.2,  0.6,  1. ])
>>> logspace(0,1,3)
array([  1.        ,   3.16227766,  10.        ])
```

# Просто индексация

```
>>> m=b+c
>>> m
array([[11, 12, 13, 14],
       [21, 22, 23, 24],
       [31, 32, 33, 34]])
m[1,1]
22
>>> m[1]
array([21, 22, 23, 24])
>>> m[:,2]
array([13, 23, 33])
```

# Сечения

Общий синтаксис индексатора:

`<от>:<до>` или `<от>:<до>:<шаг>`

```
>>> m
array([[11, 12, 13, 14],
       [21, 22, 23, 24],
       [31, 32, 33, 34]])
>>> m[0:2]
array([[11, 12, 13, 14],
       [21, 22, 23, 24]])
>>> m[:,1:-1]
array([[12, 13],
       [22, 23],
       [32, 33]])
>>> m[0:2:,0:4:2]
array([[11, 13],
       [21, 23]])
```

# Размер и форма

`.size` `.shape` `.transpose()`

```
>>> a=array([[11,12,13],[21,22,23]])
>>> a.size
6
>>> a.shape
(2, 3)
>>> a.shape=(3,2)
>>> a
array([[11, 12],
       [13, 21],
       [22, 23]])
>>> transpose(a)
array([[11, 13, 22],
       [12, 21, 23]]
```

# Размер и форма

`.ravel()`

```
>>> a
array([[11, 12, 13],
       [21, 22, 23]])
>>> a.ravel()
array([11, 12, 13, 21, 22, 23])
>>> a
array([[11, 12, 13],
       [21, 22, 23]])
>>> a.shape=a.size
>>> a
array([11, 12, 13, 21, 22, 23])
```

# Изменение размера

`resize(<массив>,<размерность>)`

```
>>> a=array([[11,12],[21,22]])
>>> a
array([[11, 12],
       [21, 22]])
>>> resize(a,5)
array([11, 12, 21, 22, 11])
>>> resize(a,(4,5))
array([[11, 12, 21, 22, 11],
       [12, 21, 22, 11, 12],
       [21, 22, 11, 12, 21],
       [22, 11, 12, 21, 22]])
>>> a
array([[11, 12],
       [21, 22]])
```

# Склеивание

`hstack(<tuple>)` `vstack(<tuple>)`

```
>>> a=array([[1,2,3],[4,5,6]])
>>> b=array([[10,20,30],[40,50,60]])
>>> c=array([[100,200],[400,500]])
>>> vstack((a,b))
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [10, 20, 30],
       [40, 50, 60]])
>>> hstack((a,b,c))
array([[  1,   2,   3,  10,  20,  30, 100, 200],
       [  4,   5,   6,  40,  50,  60, 400, 500]])
```

# Размножение

`tile(<массив>,<размерность>)`

```
>>> tile(pi,(2,3))
array([[ 3.14159265,  3.14159265,  3.14159265],
       [ 3.14159265,  3.14159265,  3.14159265]])
>>> tile([1,2],4)
array([1, 2, 1, 2, 1, 2, 1, 2])
>>> a=array([[11,12],[21,22]])
>>> tile(a,(2,3))
array([[11, 12, 11, 12, 11, 12],
       [21, 22, 21, 22, 21, 22],
       [11, 12, 11, 12, 11, 12],
       [21, 22, 21, 22, 21, 22]])
```

# Массивы одного размера

```
>>> a=array([[1,2,3],[4,5,6]])
>>> b=array([[1,1,1],[2,2,2]])
>>> a+b
array([[2, 3, 4],
       [6, 7, 8]])
>>> a**b
array([[ 1,  2,  3],
       [16, 25, 36]])
>>> a*=b
>>> a
array([[ 1,  2,  3],
       [ 8, 10, 12]])
```

# Если размерностей не хватает

то они добавляются:

```
>>> a=zeros((3,4),int)
a+15
array([[15, 15, 15, 15],
       [15, 15, 15, 15],
       [15, 15, 15, 15]])
>>> b=array([1,2,3,4])
>>> a+b
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
>>> c=array([[1],[2],[3]])
>>> a+c
array([[1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])
```

# Если размерностей не хватает

Но:

```
>>> a=zeros((3,4),int)
>>> a
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
>>> d=array([[1,2],[2,3],[3,4]])
array([[1, 2],
       [2, 3],
       [3, 4]])
>>> a+d
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shape mismatch: objects cannot be
broadcast to a single shape
```

# Если размерностей не хватает

А так можно:

```
>>> b=array([1,2,3,4])
>>> b
array([1, 2, 3, 4])
>>> c=array([[10],[20],[30]])
>>> b
array([1, 2, 3, 4])
>>> b+c
array([[11, 12, 13, 14],
       [21, 22, 23, 24],
       [31, 32, 33, 34]])
>>> b*c
array([[ 10,  20,  30,  40],
       [ 20,  40,  60,  80],
       [ 30,  60,  90, 120]])
```

# Покомпонентное сравнение массивов

```
>>> x=arange(-2,3)
>>> x
array([-2, -1,  0,  1,  2])
>>> y=-x
>>> maximum(x,y)
array([2, 1, 0, 1, 2])
>>> minimum(x,y)
array([-2, -1,  0, -1, -2])
>>> maximum(x,0)
array([0, 0, 0, 1, 2])
>>> x>y
array([False, False, False,  True,  True], dtype=bool)
>>> x==y
array([False, False,  True, False, False], dtype=bool)
>>> x>0
array([False, False, False,  True,  True], dtype=bool)
```

# Универсальные функции

В NumPy определены следующие покомпонентные функции от одного аргумента над массивами:

```
cos   arccos   cosh    arccosh
sin   arcsin   sinh    arcsinh
tan   arctan   tanh    arctanh
exp    log    log10     sqrt
abs   floor    ceil   conjugate
```

Для специальных функций существует специальный пакет scipy.special

# Векторизация скалярных функций

Никто не мешает использовать и свои функции:

```
>>> def cube(x):
...     return x**3
...

>>> x=arange(-2,3)
>>> x
array([-2, -1,  0,  1,  2])
>>> cube(x)
array([-8, -1,  0,  1,  8])
```

# Векторизация скалярных функций

Но есть нюанс:

```
>>> def f(x):
...   if x>0: return x**2
...   else: return 0
...
>>> a=array([[-1,0,1],[-10,0,10]])
>>> f(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ValueError: The truth value of an array with more
than one element is ambiguous. Use a.any()
or a.all()
```

Сравнение массива с нулем дает логический массив, а нужен скаляр.

# Векторизация скалярных функций

Правильно так:

```
>>> def f(x):
...   if x>0: return x**2
...   else: return 0
...
>>> vf=vectorize(f)
>>> a=array([[-1,0,1],[-10,0,10]])
>>> vf(a)
array([[  0,   0,   1],
       [  0,   0, 100]])
```

# “Длинная” форма

```
xxx.reduce(<массив>)
xxx.accumulate(<массив>)
```

где xxx – одна из следующих бинарных операций:

```
    add        subtract     multiply     divide
 bitwise_or  bitwise_xor  bitwise_and  remainder
```

```
>>> a=arange(1,10)
>>> multiply.reduce(a)
362880
>>> add.accumulate(a)
array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])
```

# “Длинная” форма

`xxx.reduce(<массив>,<ось>=0)`

`xxx.accumulate(<массив>,<ось>=0)`

Для многомерных массивов следует указать ось редукции (по умолчанию 0):

```
>>> b=array([[1,2,3],[10,20,30]])
>>> b
array([[ 1,  2,  3],
       [10, 20, 30]])
>>> add.reduce(b)
array([11, 22, 33])
>>> add.reduce(b,0)
array([11, 22, 33])
>>> add.reduce(b,1)
array([ 6, 60])
```

# Краткая форма

```
sum(<массив>,<ось>)
cumsum(<массив>,<ось>)
prod(<массив>,<ось>)
cumprod(<массив>,<ось>)
```

— короткие синонимы для

add.reduce(<массив>,<ось>)
add.accumulate(<массив>,<ось>)
multiply.reduce(<массив>,<ось>)
multiply.accumulate(<массив>,<ось>)

# Еще более краткая форма

```
sum(<массив>)
cumsum(<массив>)
prod(<массив>)
cumprod(<массив>)
```

— синонимы для

add.reduce(<массив>.ravel())
add.accumulate(<массив>.ravel())
multiply.reduce(<массив>.ravel())
multiply.accumulate(<массив>.ravel())

# Максимумы и минимумы

| | |
|---|---|
| `.max()` | `.max([<ось>])` |
| `.min()` | `.min([<ось>])` |

```
>>> a = array([[10,50,30],[60,20,40]])
>>> a
array([[10, 50, 30],
       [60, 20, 40]])
>>> a.max()
60
>>> a.min()
10
>>> a.max(0)
array([60, 50, 40])
>>> a.min(1)
array([10, 20])
```

# Сортировка

`argsort([<ось>=-1])` `sort([<ось>=-1])`

```
>>> a = array([[10,50,30],[60,20,40]])
>>> argsort(a,0)
array([[0, 1, 0],
       [1, 0, 1]])
>>> argsort(a,1)
array([[0, 2, 1],
       [1, 2, 0]])
>>> sort(a,0)
array([[10, 20, 30],
       [60, 50, 40]])
>>> sort(a,1)
array([[10, 30, 50],
       [20, 40, 60]])
```

# Сортировка “на месте”

```
.sort([<ось>=-1])
```

```
>>> a
array([[10, 50, 30],
       [60, 20, 40]])
>>> a.sort(0)
>>> a
array([[10, 20, 30],
       [60, 50, 40]])
>>> a.sort(1)
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
```

# Матрицы и массивы

`matrix(<данные>)` `array(<данные>)`

```
>>> a=array([[11,12,13],[21,22,23]])
>>> b=matrix([10,100])
>>> b*matrix(a)
matrix([[2210, 2320, 2430]])
>>> c=matrix([1,10,100]).transpose()
>>> c
matrix([[  1],
        [ 10],
        [100]])
>>> a*c
matrix([[1431],
        [2541]])
array(a*c)
array([[1431],
       [2541]])
```

# Простые матричные вычисления

```
>>> q=matrix([[1.0,1.0],[0,1.0]])
>>> p=q**(-1)
>>> p
matrix([[ 1., -1.],
        [ 0.,  1.]])
>>> p*q
matrix([[ 1.,  0.],
        [ 0.,  1.]])
>>> r=matrix([[1],[15]])
>>> x=p**(-1)*r
>>> x
matrix([[ 16.],
        [ 15.]])
>>> p*x
matrix([[  1.],
        [ 15.]])
```

# Короткие синонимы

`.T -- транспонирование` `.H -- эрмитово сопряжение`

`.I -- обратная матрица`

```
>>> m=matrix([[11+1j,12+2j],[21+3j,22+4j]])
>>> m
matrix([[ 11.+1.j,  12.+2.j],
        [ 21.+3.j,  22.+4.j]])
>>> m.T
matrix([[ 11.+1.j,  21.+3.j],
        [ 12.+2.j,  22.+4.j]])
>>> m.H
matrix([[ 11.-1.j,  21.-3.j],
        [ 12.-2.j,  22.-4.j]])
>>> m.I
matrix([[-1.07692308+1.11538462j,  0.57692308-0.615384
        [ 0.98076923-1.09615385j, -0.48076923+0.596153
```

# Короткие синонимы

`.A -- режим массива`

```
>>> m=matrix([[11+1j,12+2j],[21+3j,22+4j]])
>>> m
matrix([[ 11.+1.j,  12.+2.j],
        [ 21.+3.j,  22.+4.j]])
>>> m**2
matrix([[ 366.+100.j,  386.+126.j],
        [ 678.+204.j,  714.+254.j]])
>>> m.A
array([[ 11.+1.j,  12.+2.j],
       [ 21.+3.j,  22.+4.j]])
>>> m.A**2
array([[ 120. +22.j,  140. +48.j],
       [ 432.+126.j,  468.+176.j]])
```

# НЕпростые матричные вычисления

Куда более широкие возможности дает пакет

scipy.linalg

!

# Бинарные файлы

`.tofile(<имя файла>)`

`fromfile(<имя файла>,<тип>=float)`

```
>>> a=array([[1,2],[3,4]])
>>> a.tofile('a.bin')
>>> b=fromfile('a.bin')
>>> b
array([  4.24399158e-314,   8.48798317e-314])
>>> b=fromfile('a.bin',int)
>>> b
array([1, 2, 3, 4])
>>> b.shape=2,2
>>> b
array([[1, 2],
       [3, 4]])
```

Ни тип, ни форма массива в файле не сохраняются!

# Текстовые файлы

```
.tofile(<имя файла>,sep=<разделитель>)
fromfile(<имя файла>,<тип>=float,sep=<разделитель>)
```

```
>>> b
array([[1, 2],
       [3, 4]])
>>> b.tofile('b.txt',sep=' ')
>>> c=fromfile('b.txt',int,sep=' ')
>>> c
array([1, 2, 3, 4])
```