# Numpy
# Python for Ecologists

Tao Hong, Tom Purucker , Chance Pascale

Ecological Society of America Workshop
Portland, OR
hongtao510@gmail.com
August 1st, 2012

1

## Overview of files

- Install Numpy
- Array
- Indexing
- Matrix
- Reference

2

## Install Numpy

- Windows
  - 1. for 32 bit machine, download from: http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/
  - 2. for 64 bit system, download from: http://www.lfd.uci.edu/~gohlke/pythonlibs/
  - **Portable Python** http://www.portablepython.com/wiki/PortablePython2.7.3.1

3

## Create an array

# create an array from a list

>>>import numpy as np

**>>>a=np.array([10, 20, 30, 40])**

**>>>a**

>>>> [10 20 30 40]

**>>> a.shape**

>>>(4,)

4

## np.array (cont'd)

**>>> b = np. array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]])**
**>>> b**
>>> [[ 1 2 3 4]
[ 4 5 6 7]
[ 7 8 9 10]]
**>>> b.shape**
>>>(2,2)
**>>> b.reshape(4,3)**
>>> [[ 1 2 3]
[ 4 4 5]
[ 6 7 7]
[ 8 9 10]]

5

## np.array (cont'd)

**>>> b.shape = 6,-1**
Python will automatically calculate the length of second axis 12/6=2
**>>>b**
>>> [[ 1 2]
[ 3 4]
[ 4 5]
[ 6 7]
[ 7 8]
[ 9 10]]

6

## np.array (cont'd)

**>>>b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]],dtype=float)**

>>>b.dtype

>>> float64

>>>b

>>> [[ 1. 2. 3. 4.]
[ 4. 5. 6. 7.]
[ 7. 8. 9. 10.]]

7

## np.arange

np.array approach is not efficient, let's try np.arange

np.arange(*start*, *stop*, *step*, *dtype=None*)

>>>np.arange(0,4,1) #Is '4' included?

>>>[0 1 2 3] (does not include '4')

>>> np.arange(4)

>>>[0 1 2 3]

8

## np.linespace

np.linspace(start, stop, num, endpoint=True, retstep=False)

>>>a = np.linspace(2.0, 3.0, num=5, endpoint= True, retstep=True)

>>>(array([ 2. , 2.25, 2.5 , 2.75, 3. ]), 0.25)

>>>b = np.linspace(2.0, 3.0, num=5, endpoint= False, retstep=True)

>>>(array([ 2. , 2.2, 2.4, 2.6, 2.8]), 0.2)

9

## Structured (record) Arrays

- allows access to its data using named fields.

>>>persontype = np.dtype({'names':['name', 'age', 'weight'],'formats':['S32','i', 'f']})

>>>a = np.array([('Name A',32,75.5),('Name B',24,65.5)], dtype=persontype)

>>>a

>>>[('Name A', 32, 75.5) ('Name B', 24, 65.5)]

10

>>a[0]

>>>('Name A', 32, 75.5)

>>>a['name']

>>>['Name A' 'Name B']

>>>a['age'][0]

>>>32

>> a['name'][1]='tao' #modify

>>>a

>>> [('Name A', 32, 75.5) ('tao', 24, 65.5)]

11

## fromfunction

- Construct an array by executing a function over each coordinate

>>>def func(i):

>>> return i%4+1

>>> np.fromfunction(func, (5,))

>>> [ 1. 2. 3. 4. 1.]

12

## Some propertites

```
>>>a = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]],dtype=float)
>>>[[ 0 10 20]
     [30 40 50]]

>>>a.shape          #shape
>>>(2, 3)

>>>a.ndim           #number of dimensions
>>>2

>>>a.dtype          #data type
>>> int32

>>>a.size           #number of elements
>>>6

>>> k= a.flat       #return a flat iterator over an array
>>>for i in k:
 >>>  print i
```

13

## Indexing

>>> a = np.arange(10)

Integer index

>>> a[5]

| 0 | 1 | 2 | 3 | 4 | **5** | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

>>5

A range (starts at the 3th and ends before 5th)

>>> a[3:5]

| 0 | 1 | 2 | **3** | **4** | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

>>>[3, 4]

14

The first five elements

>>> a[:5]

| **0** | **1** | **2** | **3** | **4** | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

>>>[0, 1, 2, 3, 4]

Counting backwards

>>> a[:-1]

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | 9 |
|---|---|---|---|---|---|---|---|---|---|

array([0, 1, 2, 3, 4, 5, 6, 7, 8])

15

Reverse the array

>>> a[::-1]

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|

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

```
>>> a[::-1][::2]      #every other one
>>>[ 9, 7, 5, 3, 1]

Two dimensions
>>b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]])
>>c= b[:,::-1]
b= [[ 1  2  3  4]                c= [[ 4  3  2  1]
    [ 4  5  6  7]                    [ 7  6  5  4]
    [ 7  8  9 10]]                   [10  9  8  7]]
```

16

Modify content

>>> a[2:4] = 100,101

>>> a

| 0 | 1 | **100** | **101** | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

>>> [ 0, 1, 100, 101, 4, 5, 6, 7, 8, 9]

a[ i:j:k] *i* is the starting index, *j* is the stopping index, and *k* is the step.

>>> a[1:-1:2]

| 0 | **1** | 100 | **101** | 4 | **5** | 6 | **7** | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

>>>[ 1, 101, 5, 7]

17

```
>>>x = np.arange(10)
>>> [0 1 2 3 4 5 6 7 8 9]
>>>x[[3, 3, -3, 8]]   #list
>> [3 3 7 8]
```

| 0 | 1 | 2 | **3** | 4 | 5 | 6 | **7** | **8** | 9 |
|---|---|---|---|---|---|---|---|---|---|

```
>> x[np.array([3,3,-3,8])] #array
>> [3 3 7 8]

>> x[np.array([True, False, True, False, False])]
>> [0 2]        #the missing ones considered as 'False'
```

18

## Multidimensions

a = np.arange(0, 60, 10).reshape(-1, 1)+np.arange(0, 6)

| | | | | | |
|---|---|---|---|---|---|
| 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 |

19

```
>>>a[:,2]
>>> [ 2 12 22 32 42 52]

>>>a[0,3:5]
>>>[3,4]

>>>a[4:,4:]  #select a 'block'
>>> [[44 45]
     [54 55]]
```


```
>>>a[2::2,::2]      #start from 3rd row, step_row=2, start from 1st col, step_col=2
>>>[[20 22 24]
     [40 42 44]]

>>>x[2::2,::-1]?
>>> [[25 24 23 22 21 20]
     [45 44 43 42 41 40]]
```

20

## np.indices

- Return an array representing the indices of a grid
  x = np.arange(20).reshape(5, 4)
  x=[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]
  [12 13 14 15]
  [16 17 18 19]]
  row, col = np.indices((2, 3))
  x[row, col]=[[0 1 2]
  [4 5 6]]
- Extract the required elements directly with ``x[:2, :3]``.

21

## Broadcasting

- Deal with inputs that do not have exactly the same shape.
- Rule 1: if arrays do not have the same number of dimensions, then a "1" will be repeatedly added to the shapes of the smaller arrays
- Rule 2: arrays with a size of 1 along a particular dimension act as if they had the size of the array with the largest shape along that dimension.

22

```
>>> a = np.arange(0, 60, 10).reshape(-1, 1)    # a.shape=(6,1)
>>> b = np.arange(0, 5)                        #b.shape=(5,)
>>> c = a+b
How does array c looks like?

>>> a
>>>[[ 0]
     [10]
     [20]
     [30]
     [40]
     [50]]
```

Rule 1

```
>>> a = a.repeat(5, axis=1)
        >>>a
>>> [[ 0  0  0  0  0]
     [10 10 10 10 10]
     [20 20 20 20 20]
     [30 30 30 30 30]
     [40 40 40 40 40]
     [50 50 50 50 50]]
```

23

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

Rule 1

```
>>> b.shape=1,5
>>> b
>>>[[0, 1, 2, 3, 4]]
```

Rule 2

```
>>> b = b.repeat(6,axis=0)
>>> b
>>> [[0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4]]
```

24

```
>>> c = a + b
>>> c
>>> [[ 0  1  2  3  4]
     [10 11 12 13 14]
     [20 21 22 23 24]
     [30 31 32 33 34]
     [40 41 42 43 44]
     [50 51 52 53 54]]
```

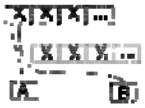
25

## Deep copy shallow copy

- A shallow copies collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Shallow copy

```
>>>a = np.arange(0, 60, 10)
>>>b=a
>>>a
>>>[ 0 10 20 30 40 50]
>>>b
>>>[ 0 10 20 30 40 50]
>>>b[0]=100
>>>b
>>>[100  10  20  30  40  50]
>>>a
>>>[100  10  20  30  40  50]
```

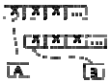



26

- Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

Deep copy

```
>>>a = np.arange(0, 60, 10)
>>> b= copy.deepcopy(a)
>>>a
>>>b[0]=100
>>>b
>>>[100  10  20  30  40  50]
>>>a
>>>[0  10  20  30  40  50]
```




27

## np.array VS np.mat

```
>>> a=np.mat('4 3; 2 1')
>>>a
>>> [[4 3]
     [2 1]]
>>> b=np.mat('1 2; 3 4')
>>>b
>>> [[1 2]
     [3 4]]
>>>c=a*b
>>>c
>>> [[13 20]
     [ 5  8]]
```

```
>>> a= np.array([[4, 3], [2, 1]])
>>>a
>>> [[4 3]
     [2 1]]
>>> b= np.array([[1, 2], [3, 4]])
>>>b
>>> [[1 2]
     [3 4]]
>>>c=a*b
>>>c
>>> [[4 6]
     [6 4]]
>>>d=np.dot(a,b)
>>> [[13 20]
     [ 5  8]]
```

28

```
>>>A = matrix( [[1,2,3],[11,12,13],[21,22,23]]) # Creates a matrix.
>>> [[ 1  2  3]
     [11 12 13]
     [21 22 23]]

>>>print A.T            # Transpose of A
>>> [[ 1 11 21]
     [ 2 12 22]
     [ 3 13 23]]

>>>print A.I            # Inverse of A.
>>> [[  3.00239975e+14  -6.00479950e+14   3.00239975e+14]
     [ -6.00479950e+14   1.20095990e+15  -6.00479950e+14]
     [  3.00239975e+14  -6.00479950e+14   3.00239975e+14]]
```

29

## Solve a linear system

$$[a]_{3\times3}[x]_{3\times1} = [b]_{3\times1}$$

```
>>> a= np.random.rand(3,3)
>>> [[ 0.26835516  0.1812329   0.07554446]
     [ 0.2915491   0.27213494  0.05657924]
     [ 0.89496488  0.35577792  0.88181086]]
>>> b= np.random.rand(3).reshape(3,-1)
>>> [[ 0.80181235]
 [ 0.13312618]
 [ 0.5599297 ]]
>>>x=np.linalg.solve(a,b)
>>> [[ 0.83270995]
     [ 0.43698752]
     [-0.144376 ]]
```

30

## More numpy matrix functions

- **NumPy for MATLAB Users**
http://www.scipy.org/NumPy_for_Matlab_Users

- **NumPy for R (and S-Plus) users**
http://mathesaurus.sourceforge.net/r-numpy.html

31

## Reference

- Official document
NumPy User Guide (not the reference guide)
http://docs.scipy.org/doc/numpy/numpy-user.pdf
- Guide to NumPy by Travis E. Oliphant
http://www.tramy.us/numpybook.pdf
- http://stackoverflow.com/

32