# Numpy Python for Ecologists

### Tao Hong, Tom Purucker, Jonathan Flaishans, Marcia Snyder

Ecological Society of America Workshop Minneapolis, MN hongtao510@gmail.com

August 1, 2013

### Overview

- Install Numpy
- Array
- Index
- Random number
- Basic operations
- Reference

### **Install Numpy**

### Windows

- Official http://sourceforge.net/projects/numpy/?source=dlp
- Unofficial (Windows binaries)
  http://www.lfd.uci.edu/~gohlke/pythonlibs/
- Portable Python
  http://portablepython.com/wiki/PortablePython2.7.5.1/

### **Arrays**

Create an array

```
a = np.array([10, 20, 30, 40], float)
[ 10. 20. 30. 40.]
b = np. array([10, 20, 30, 40])
[10 20 30 40]
c = np. array([[1, 2],[4, 5]])
[[1 \ 2]
 [4 5]]
d = np. array([[1, 2],[4, 5.0]])
[[ 1. 2.]
 [ 4. 5.]]
```

4

### np.arange

- np.array is not efficient, try np.arange
- np.arange(start, stop, step, dtype=None)

```
f = np.arange(0,4,1) #does not include '4'
[0 1 2 3]
g = np.arange(4)
[0 1 2 3]
```

### np.linespace

- Compare to np.arange
- np.linspace(start, stop, num, endpoint=True, retstep=False)

6

### Other ways to create arrays

np.ones

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

np.zeros

```
x = np.zeros((2,3), float)
[[ 0. 0. 0.]
[ 0. 0. 0.]]
```

np.identity

```
x = np.identity(3, float)
[[ 1.  0.  0.]
  [ 0.  1.  0.]
  [ 0.  0.  1.]]
```

### Array properties (1)

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
a.shape #shape
(2, 3)
a.reshape(1,6)
[[ 1. 2. 3. 4. 5. 6.]]
a.ndim #number of dimensions
2
a.dtype #data type
float64
a. size #number of elements
6
```

8

### Array properties (2)

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
k = a.flatten()
[ 1.  2.  3.  4.  5.  6.]

I = a.tolist() #array to list
[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]
type(I)
<type 'list'>
```

# Array properties (3) m = a.transpose() [[ 1. 4.] [ 2. 5.] [ 3. 6.]] p = np.array([[1, 2], [3, 4]]) g = np.array([[5, 6], [7,8]])

```
q = np.array([[5, 6], [7,8]])

r = np.concatenate((p,q), axis)
```

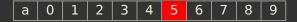
```
r = np.concatenate((p,q), axis=0) #Join arrays together
[[1 2]
  [3 4]
  [5 6]
  [7 8]]
```

s = np.concatenate((p,q), axis=1) [[1 2 5 6] [3 4 7 8]]

### Index (1)

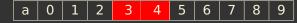
Integer index

a = np.arange(10)



a[5] 5

A range (starts at the 4th and ends before 6th)



a[3:5]

[3, 4]

### Index (2)

The first three elements



a[:3] [0,1,2]

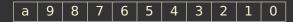
Counting backwards



a[:-7] [0,1,2]

### Index (3)

Reverse the array

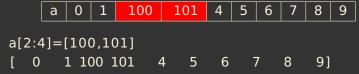


How to get?



### Index (4)

Modify content



a[i:j:k]

i-first, j-last (not included), k-step



```
a[1:-1:2]
[ 1 101 5 7]
```

Try

### Index (5)

Find index of an array

```
x = np.arange(9.).reshape(3, 3)
x

[[ 0.  1.  2.]
  [ 3.  4.  5.]
  [ 6.  7.  8.]]

np.where( x > 4.5 )
(array([1, 2, 2, 2]), array([2, 0, 1, 2]))
```

### Random number (1)

rand(dim) Uniform distribution over [0, 1)

```
a = np.random.rand(2, 3)
[[ 0.42811767  0.43032497  0.19511638]
[ 0.19985235  0.09149539  0.42384995]]
```

randn(dim) standard normal

### Random number (2)

- log-normal lognormal(mean, sigma, dim)
- Possion poisson(mean, dim)
- Beta beta(a, b, dim)
- Fix a seed seed(number)
- more distritions are available
  http://docs.scipy.org/doc/numpy/reference/
  routines.random.html

### Basic operations (1)

sum a = np.array([[1, 2, 3], [4, 5, 6]], float)[[ 1. 2. 3.] [4.5.6.]] a.sum() 21.0 a.sum(axis=0) #col sum [5.7.9.] mean a.mean() 3.5 variance a.var() 2.91666666667

## Basic operations (2)

- min a = np.array([[1, 2, 2], [4, 5, 4]], float)[[ 1. 2. 2.] [4.5.4.]] a.min() 1.0
  - index lookup a.argmin() 0
    - find unique elements np.unique(a)
    - <u>[ 1.</u> 2. 4. 5.] diagnoal
- a.diagonal() [ 1. 5.]

### Basic operations (3)

inverse

```
a = np.array([[1, 2], [4, 5]], float)
[[ 1. 2.]
 [ 4. 5.]]
b=np.linalg.inv(a)
[[-1.666666667]
 [1.3333333333333333333333333]
determinant
np.linalg.det(a)
-3
```

### Basic operations (4)

matrix multiply

```
np.dot(a,b)
[[ 1. 0.]
[ 0. 1.]]
```

element-wise multiply

```
a*b
[[-1.66666667 1.33333333]
[ 5.33333333 -1.66666667]]
```

solve a linear system
a x c=h

```
a x c=b
c=np.linalg.solve(a,b)
[[ 3.666666667 -1.33333333]
[-2.666666667 1. ]]
```

### Shallow copy

arrays share the same elements

```
a = np.arange(0, 60, 10)
b = a
а
[ 0 10 20 30 40 50]
b
[ 0 10 20 30 40 50]
a[0]=100
а
[100 10 20 30
                 40
                      501
b
[100 10 20 30
                 40
                     501
```

### Deep copy

each array has its own elements

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

### Reference

- Official document http://docs.scipy.org/doc/
- 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
- Stackoverflow
  http://stackoverflow.com/