Numpy

Python for Ecologists

Tao Hong, Chance Pascale, Tom Purucker

Ecological Society of America Workshop
Minneapolis, MN

hongtao510@gmail.com

July 31, 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.]]
```

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)

```
h = np.linspace(2.0, 3.0, num=5)
[ 2.    2.25  2.5   2.75  3.  ]

i = np.linspace(2.0, 3.0, num=5, endpoint=False)
[ 2.   2.2  2.4  2.6  2.8]

j = np.linspace(2.0, 3.0, num=5, retstep=True)
(array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)
```

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
```

Array properties (2)

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

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

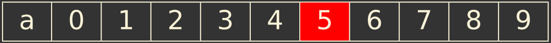
Array properties (3)

```
m = a.transpose()
[[ 1.  4.]
 [ 2.  5.]
 [ 3.  6.]]

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

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

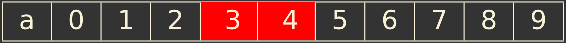
```
a = np.arange(10)
```



a	0	1	2	3	4	**5**	6	7	8	9

```
a[5]
5
```

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

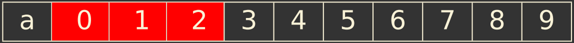


a	0	1	2	**3**	**4**	5	6	7	8	9

```
a[3:5]
[3, 4]
```

Index (2)

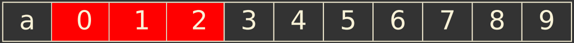
- The first three elements



a	0	1	2	3	4	5	6	7	8	9

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

- Counting backwards



a	0	1	2	3	4	5	6	7	8	9

```
a[:−7]
[0,1,2]
```

Index (3)

- Reverse the array

a	9	8	7	6	5	4	3	2	1	0

```
a[::−1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

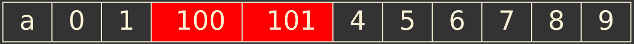
- How to get?

a	9	8	7	6	5	4	3	2	1	0

```
a[::−3]
[9, 6, 3, 0]
```

Index (4)

- Modify content

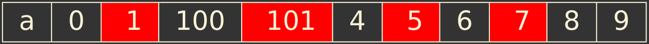


a	0	1	100	101	4	5	6	7	8	9

```
a[2:4]=[100,101]
[   0    1  100  101    4    5    6    7    8    9]
```

- a[i:j:k]

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



a	0	1	100	101	4	5	6	7	8	9

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

- Try

```
a[1::2]
[   1  101    5    7    9]
```

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

```
a = np.random.randn(2, 3)
[[ 0.41391683  1.31774009 −1.10235464]
 [ 0.32073693  0.32847825 −0.49657114]]
```

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)
[ 1.  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)
[[−1.66666667   0.66666667]
 [ 1.33333333  −0.33333333]]
```

- 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=b

```
c=np.linalg.solve(a,b)
[[ 3.66666667 -1.33333333]
 [-2.66666667  1.        ]]
```

Shallow copy

- arrays share the same elements

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

a[0]=100
a
[100  10  20  30  40  50]
b
[100  10  20  30  40  50]
```

Deep copy

- each array has its own elements

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

a[0]=100
a
[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/`