NumPy Basics: Arrays and Vectorized Computation

Numerical Python 的简称，是Python科学计算的基础包。

NumPy的主要对象是**同类型**元素的多维数组。

其所有的元素都是一种类型、通过一个正整数元组索引的元素表格。 本节的内容包括array数据的：

- 生成
- 选取
- 数据处理
- 运算等

- 导入numpy, 设置数据显示的格式，设置在notebook中显示图形

In [1]:
```
from numpy.random import randn
import numpy as np

# 设置NumPy对象的显示的格式,
np.set_printoptions(precision=4, suppress=True)

%matplotlib inline
```

In [8]:
```
a=np.array([34.23534,0.1**30])
a
```

Out[8]:
```
array([34.2353,  0.    ])
```

NumPy 多维数值

在NumPy中维度(dimensions)叫做轴(axes)。

- [1, 2, 3] 是1×3的数组

- [[1., 0., 0.], [0., 1., 2.]]是2×3的数组， 它有两个维度，第一个维度长度为2,第二个维度长度为3.

- 下面的例子抽取一个2×3的数组，并进行计算。 randn函数从标准正态分布抽取随机数。

In [4]:
```
data = randn(2, 3)
#data
```

In [5]:
```
print(data * 10,end="\n\n")
print(data + data)
```

```
[[ -0.5263   7.9418 -10.2008]
 [ -4.0823  11.3989   3.5736]]

[[-0.1053  1.5884 -2.0402]
 [-0.8165  2.2798  0.7147]]
```

In [6]:

```
data.shape
```

Out[6]:
```
(2, 3)
```

In [7]:
```
data.dtype
```

Out[7]:
```
dtype('float64')
```

产生数组变量

- 整数数组序列

In [8]:
```
print(np.arange(15),end="\n\n")
print(np.arange(2,15),end="\n\n")
print(np.arange(2,15,2),end="\n\n")
print(np.arange(2,15,1.5),end="\n\n")
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

[ 2  3  4  5  6  7  8  9 10 11 12 13 14]

[ 2  4  6  8 10 12 14]

[ 2.   3.5  5.   6.5  8.   9.5 11.  12.5 14. ]
```

- 用list产生

In [9]:
```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

Out[9]:
```
array([6. , 7.5, 8. , 0. , 1. ])
```

In [22]:
```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
```

Out[22]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [11]:
```
arr2.shape
```

Out[11]:
```
(2, 4)
```

In [135…
```
# reshape可以改变维度
arr2.reshape((1, 8)).reshape(4,2)# reshape((1, 8))和reshape(1, 8)都可以
```

Out[135…
```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

- 一些常用的矩阵和向量

In [136…
```
np.zeros(10,dtype="int")
#np.empty(10)
```

Out[136…
```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

In [13]:
```
np.ones((2,3))
```

Out[13]:
```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

In [137…
```
np.identity(2)
np.eye(2,3)
```

Out[137…
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.]])
```

多维数组的类型

- np.array的数据类型可以指定，如果没有指定，会自动推断

In [15]:
```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
arr = np.array([1, 2, 3, 4, 5])
arr1.dtype,arr2.dtype,arr.dtype
```

Out[15]:
```
(dtype('float64'), dtype('int32'), dtype('int32'))
```

- 转换数据类型

In [16]:
```
# 整数转换为浮点
float_arr = arr.astype(np.float64)

float_arr.dtype,float_arr
```

Out[16]:
```
(dtype('float64'), array([ 1.,  2.,  3.,  4.,  5.]))
```

In [17]:
```
# 浮点转换为整数
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr.astype(np.int32)
```

Out[17]:
```
array([ 3, -1, -2,  0, 12, 10])
```

In [18]:
```
# 字符转浮点
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=str)
numeric_strings.astype(float)
```

Out[18]:
```
array([  1.25,  -9.6 ,  42.  ])
```

数组和标量的运算

数组的+，-，*,/为对应位置的点对点运算

In [19]:
```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
```

```
In [19]:
arr * arr
```

```
Out[19]:
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])
```

```
In [20]:
arr - arr
```

```
Out[20]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
In [21]:
1 / arr
```

```
Out[21]:
array([[ 1.    ,  0.5   ,  0.3333],
       [ 0.25  ,  0.2   ,  0.1667]])
```

```
In [22]:
arr ** 0.5
```

```
Out[22]:
array([[ 1.    ,  1.4142,  1.7321],
       [ 2.    ,  2.2361,  2.4495]])
```

索引和切片

```
In [3]:
arr = np.arange(10)
arr[5]
```

```
Out[3]:
5
```

```
In [24]:
arr[5:8]
```

```
Out[24]:
array([5, 6, 7])
```

- 5:8表示[5,6,7]
- 而list,array等python对象位置索引从0开始的，其实是引用第6，7，8个元素。

```
In [25]:
arr[5:8] = 12
arr
```

```
Out[25]:
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

- 切片传地址;
- 注意这里和list有区别,一般数列list的切片拷贝生成新的对象

```
In [24]:
arr=np.arange(10)
arr_slice = arr[5:8]
arr_slice[1] = 12345
arr
```

```
Out[24]:
array([    0,     1,     2,     3,     4,     5, 12345,     7,     8,
           9])
```

```
In [25]:
arr_slice[:] = 64
arr
```

Out[25]:
```
array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

In [26]:
```
arr_slice=6400
arr
```

Out[26]:
```
array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

- list 切片拷贝生成新的对象

In [1]:
```
a_list=list(range(10))
list_slice = a_list[5:8]
list_slice[1] = 12345
a_list
```

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

In [3]:
```
a=[1,2,3,[1,2]]
b=a[3]
b[1]=1000
```

In [5]:
```
a
```

Out[5]:
```
[1, 2, 3, [1, 1000]]
```

- 二维数组的引用和切片

In [30]:
```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d)
"几种不同的引用方式:",arr2d[2],arr2d[:][1],arr2d[:,1],arr2d[0][2],arr2d[0, 2]
#比较 arr2d[:][1] 和 arr2d[:,1]
```

Out[30]:
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
('几种不同的引用方式:', array([7, 8, 9]), array([4, 5, 6]), array([2, 5, 8]), 3, 3)
```

In [31]:
```
# 二维数组切片也传地址
b=arr2d[2]
b1=arr2d[0][:2]
b2=arr2d[0][:1]

b1[:]=1000
b2[:]=999
b[:]=30
arr2d
```

Out[31]:
```
array([[ 999, 1000,    3],
       [   4,    5,    6],
       [  30,   30,   30]])
```

- 3维数组的引用和切片
- 三层嵌套[],每层一个维度

In [33]:
```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
arr3d
```

Out[33]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

In [34]:
```
arr3d.shape
```

Out[34]:
```
(2, 2, 3)
```

In [35]:
```
"第一维：",arr3d[0],"第二维：",arr3d[0][0],"第三维：",arr3d[0][0][0]
```

Out[35]:
```
('第一维：', array([[1, 2, 3],
        [4, 5, 6]]), '第二维：', array([1, 2, 3]), '第三维：', 1)
```

- 数组拷贝
- 要生成新的对象(传递数值)，而不是传递地址，需要用到copy函数

In [36]:
```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d)
old_values=arr2d.copy()
old_values
```

Out[36]:
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [40]:
```
arr2d[0][:] =999
print(arr2d)
print("copy:",old_values)
```

```
[[999 999 999]
 [  4   5   6]
 [  7   8   9]]
copy: [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

- 利用切片索引

In [45]:
```
arr[1:6]
```

Out[45]:
```
array([ 1,  2,  3,  4, 64])
```

In [46]:
```
arr2d
arr2d[:2]
```

Out[46]:
```
array([[0, 0, 0],
       [4, 5, 6]])
```

In [48]:
```
#arr2d[1, :2]
```

```
arr2d
```

Out[48]:
```
array([[0, 0, 0],
       [4, 5, 6],
       [7, 8, 9]])
```

In [49]:
```
arr2d[:2, 1:] = 0
arr2d
```

Out[49]:
```
array([[0, 0, 0],
       [4, 0, 0],
       [7, 8, 9]])
```

利用布尔值索引

In [50]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
data = randn(7, 4)
print(names,end="\n\n")
data
```

```
['Bob' 'Joe' 'Will' 'Bob' 'Will' 'Joe' 'Joe']
```

Out[50]:
```
array([[ 0.4284,  0.5831,  0.8291,  1.5017],
       [ 2.282 ,  0.2246, -0.0124,  0.7796],
       [-0.1227, -0.1779, -1.566 , -0.4094],
       [-0.253 , -1.3919,  2.0042,  0.0999],
       [-0.2978, -1.1485,  0.529 , -0.3879],
       [ 0.3699, -0.021 , -0.6435, -0.866 ],
       [ 1.3693, -0.3835,  0.2603,  0.2608]])
```

In [51]:
```
names == 'Bob'
```

Out[51]:
```
array([ True, False, False,  True, False, False, False], dtype=bool)
```

In [52]:
```
data[names == 'Bob']
```

Out[52]:
```
array([[ 0.4284,  0.5831,  0.8291,  1.5017],
       [-0.253 , -1.3919,  2.0042,  0.0999]])
```

In [53]:
```
data[names == 'Bob', 2:]
```

Out[53]:
```
array([[ 0.8291,  1.5017],
       [ 2.0042,  0.0999]])
```

In [54]:
```
data[names == 'Bob', 3]
```

Out[54]:
```
array([ 1.5017,  0.0999])
```

In [55]:
```
names != 'Bob'
data[names != 'Bob']
```

Out[55]:
```
array([[ 2.282 ,  0.2246, -0.0124,  0.7796],
       [-0.1227, -0.1779, -1.566 , -0.4094],
       [-0.2978, -1.1485,  0.529 , -0.3879],
       [ 0.3699, -0.021 , -0.6435, -0.866 ],
       [ 1.3693, -0.3835,  0.2603,  0.2608]])
```

In [56]:

```
mask = (names == 'Bob') | (names == 'Will')
mask
```

Out[56]:
```
array([ True, False,  True,  True,  True, False, False], dtype=bool)
```

In [57]:
```
data[mask]
```

Out[57]:
```
array([[ 0.4284,  0.5831,  0.8291,  1.5017],
       [-0.1227, -0.1779, -1.566 , -0.4094],
       [-0.253 , -1.3919,  2.0042,  0.0999],
       [-0.2978, -1.1485,  0.529 , -0.3879]])
```

In [58]:
```
data
```

Out[58]:
```
array([[ 0.4284,  0.5831,  0.8291,  1.5017],
       [ 2.282 ,  0.2246, -0.0124,  0.7796],
       [-0.1227, -0.1779, -1.566 , -0.4094],
       [-0.253 , -1.3919,  2.0042,  0.0999],
       [-0.2978, -1.1485,  0.529 , -0.3879],
       [ 0.3699, -0.021 , -0.6435, -0.866 ],
       [ 1.3693, -0.3835,  0.2603,  0.2608]])
```

In [59]:
```
data<0
```

Out[59]:
```
array([[False, False, False, False],
       [False, False,  True, False],
       [ True,  True,  True,  True],
       [ True,  True, False, False],
       [ True,  True, False,  True],
       [False,  True,  True,  True],
       [False,  True, False, False]], dtype=bool)
```

In [60]:
```
data[data < 0] = 0
#data
data
```

Out[60]:
```
array([[ 0.4284,  0.5831,  0.8291,  1.5017],
       [ 2.282 ,  0.2246,  0.    ,  0.7796],
       [ 0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  2.0042,  0.0999],
       [ 0.    ,  0.    ,  0.529 ,  0.    ],
       [ 0.3699,  0.    ,  0.    ,  0.    ],
       [ 1.3693,  0.    ,  0.2603,  0.2608]])
```

In [61]:
```
data[names != 'Joe'] = 7
data
```

Out[61]:
```
array([[ 7.    ,  7.    ,  7.    ,  7.    ],
       [ 2.282 ,  0.2246,  0.    ,  0.7796],
       [ 7.    ,  7.    ,  7.    ,  7.    ],
       [ 7.    ,  7.    ,  7.    ,  7.    ],
       [ 7.    ,  7.    ,  7.    ,  7.    ],
       [ 0.3699,  0.    ,  0.    ,  0.    ],
       [ 1.3693,  0.    ,  0.2603,  0.2608]])
```

花式索引

利用整数数组进行索引，该引用拷贝数据。

In [62]:
```
 np.empty((8, 4))
```

Out[62]:
```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

In [63]:
```
arr = np.empty((8, 4))
for i in range(8):
    arr[i] = i
arr
```

Out[63]:
```
array([[ 0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.],
       [ 7.,  7.,  7.,  7.]])
```

In [66]:
```
arr[[4, 3, 0, 6]]
```

Out[66]:
```
array([[ 4.,  4.,  4.,  4.],
       [ 3.,  3.,  3.,  3.],
       [ 0.,  0.,  0.,  0.],
       [ 5.,  5.,  5.,  5.]])
```

In [67]:
```
arr[[-3, -5, -7]]
#- 表示倒序
```

Out[67]:
```
array([[ 5.,  5.,  5.,  5.],
       [ 3.,  3.,  3.,  3.],
       [ 1.,  1.,  1.,  1.]])
```

In [68]:
```
arr = np.arange(32).reshape((8, 4))
arr
```

Out[68]:
```
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, 30, 31]])
```

In [69]:
```
arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]]
```

Out[69]:
```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

In [70]:
```
arr[[1, 5, 7, 2], [0, 3, 1, 2]]
```

Out[70]:
```
array([ 4, 23, 29, 10])
```

转置

In [71]:
```
arr = np.arange(15).reshape((3, 5))
arr
arr.T
```

Out[71]:
```
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

In [72]:
```
arr = np.random.randn(6, 3)
np.dot(arr.T, arr)
```

Out[72]:
```
array([[ 10.2805,  -3.794 ,  -1.4193],
       [ -3.794 ,   7.043 ,  -0.4982],
       [ -1.4193,  -0.4982,   6.6694]])
```

In [90]:
```
arr =np.arange(16).reshape((2, 2, 4))
arr
```

Out[90]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
```

通用函数，快速的元素级数组函数

In [95]:
```
arr = np.arange(10)
np.sqrt(arr)
np.exp(arr)
```

Out[95]:
```
array([    1.    ,     2.7183,     7.3891,    20.0855,    54.5982,
        148.4132,   403.4288,  1096.6332,  2980.958 ,  8103.0839])
```

In [139…
```
x = randn(8)
y = randn(8)
print(x)
print(y)
np.maximum(x, y) #  x,y生成一个2为元组，对应位置取x,y的最大值
```

```
[-1.1185  0.0937  0.2493 -0.4965  0.5028 -0.3574  0.6705  1.884 ]
[-0.8791 -1.6034 -0.7306 -0.8252 -1.1263 -0.2182  0.8934  0.3222]
```

Out[139…
```
array([-0.8791,  0.0937,  0.2493, -0.4965,  0.5028, -0.2182,  0.8934,
        1.884 ])
```

In [97]:
```
arr = randn(7) * 5
print(arr)
np.modf(arr)
```

```
[-3.8957 -2.157  -2.9302 -1.6644  3.0934 -2.1484  4.5131]
```

Out[97]:
```
(array([-0.8957, -0.157 , -0.9302, -0.6644,  0.0934, -0.1484,  0.5131]),
 array([-3., -2., -2., -1.,  3., -2.,  4.]))
```

使用数组处理数据

In [98]:
```
#meshgrid生成网格
points = np.arange(-5, 5, 0.01) # 1000 equally spaced points
xs, ys = np.meshgrid(points, points)

ys
```

Out[98]:
```
array([[-5.  , -5.  , -5.  , ..., -5.  , -5.  , -5.  ],
       [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
       [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

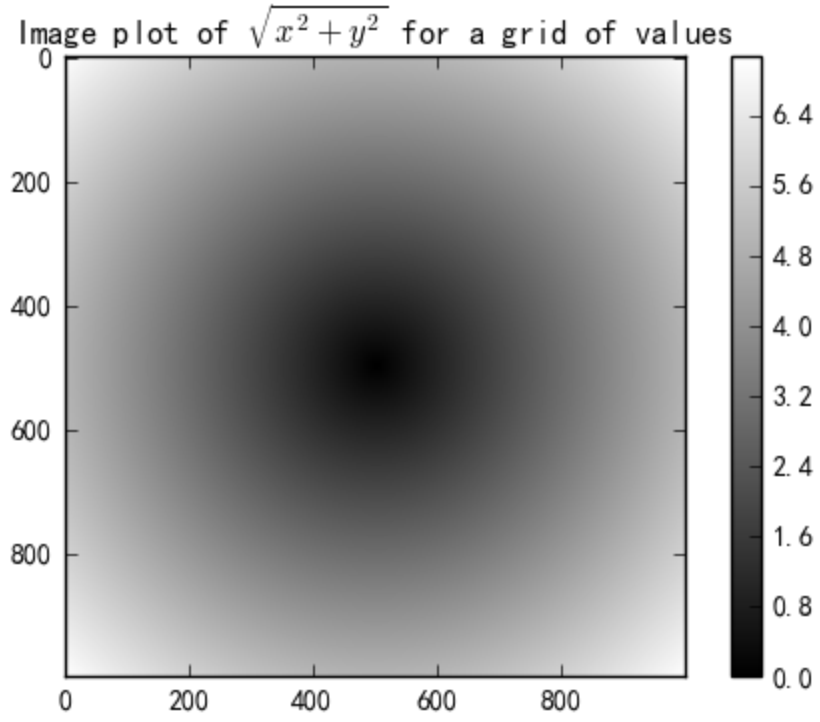
In [99]:
```
xs
```

Out[99]:
```
array([[-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       ...,
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99]])
```

In [100…
```
from matplotlib.pyplot import imshow, title
```

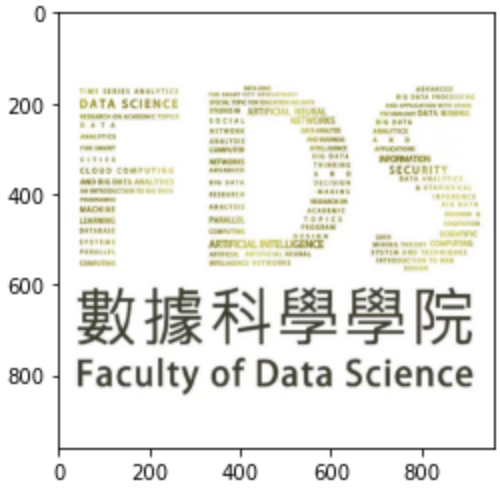
In [101…
```
import matplotlib.pyplot as plt
z = np.sqrt(xs ** 2 + ys ** 2)
z
plt.imshow(z, cmap=plt.cm.gray); plt.colorbar()
plt.title("Image plot of $\sqrt{x^2 + y^2}$ for a grid of values")
```

Out[101…
```
<matplotlib.text.Text at 0x1f4b67500f0>
```



In [3]:
```
#import skimage
import matplotlib.pyplot as plt
from skimage import io
p=io.imread("./fdslogo.jpg")
plt.imshow(p)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x11cc05fa0>
```



In [4]:
```
p[0,0,:]
```

Out[4]:
```
array([255, 255, 255], dtype=uint8)
```

In [5]:
```
p.shape
```

Out[5]:
```
(960, 960, 3)
```

In [6]:
```
p[1,1,:]
```

Out[6]:
```
array([255, 255, 255], dtype=uint8)
```

条件表达式,np.where

In [106…
```
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])
```

In [107…
```
result = np.where(cond, xarr, yarr)  ## if cond return xarr else return yarr
result
```

Out[107…
```
array([ 1.1,  2.2,  1.3,  1.4,  2.5])
```

In [108…
```
arr = randn(4, 4)
arr
np.where(arr > 0, 2, -2)
np.where(arr > 0, 2, arr) # set only positive values to 2#
```

Out[108…
```
array([[ 2.    , -0.8001,  2.    ,  2.    ],
       [ 2.    , -1.1511, -0.458 ,  2.    ],
       [-0.169 ,  2.    , -0.1923, -0.443 ],
       [-0.0794,  2.    , -0.2228, -1.6255]])
```

数学及统计函数

包括函数 sum, mean, std,var, min, max, argmin,argmax,cumsum,cumprod

In [109…
```
arr = np.random.randn(5, 4) # normally-distributed data
arr.mean()
np.mean(arr)
arr.sum()
```

Out[109…
```
-7.1066868542055159
```

In [110…
```
print(arr)
print(arr.mean())
print(arr.mean(axis=0))
arr.sum(0)
```

Out[110…
```
[[ 0.8082  0.8331 -1.5653 -0.997 ]
 [-4.3703 -1.0217 -0.4008 -0.2977]
 [ 0.4127 -0.0656  0.4016  0.8074]
 [-0.7379  1.6584 -1.9362 -0.968 ]
 [ 1.2921 -0.6088 -0.5618  0.2105]]
-0.35533434271
[-0.519   0.1591 -0.8125 -0.249 ]
array([-2.595 ,  0.7955, -4.0624, -1.2448])
```

In [140…
```
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
arr.cumsum(1)
arr.cumprod(1)
```

Out[140…
```
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336]], dtype=int32)
```

Methods for boolean arrays

In [154…
```
arr = randn(100)
(arr > 0).sum() # Number of positive values
```

Out[154…
```
54
```

In [113…
```
bools = np.array([False, False, True, False])
bools.any()
bools.all()
```

Out[113…
```
False
```

Sorting

In [114…
```
arr = randn(8)
arr
arr.sort()
arr
```

Out[114…
```
array([-1.9452, -0.4968,  0.216 ,  0.6045,  0.6046,  0.8408,  1.2012,
        2.4475])
```

In [27]:
```
arr = randn(5, 3)
print(arr)
arr.sort(0) ## 0每列排序，1，每行
arr
```

```
[[-0.6949  1.2335  2.3124]
 [-1.2293  1.3576 -0.2191]
 [ 0.0377  2.1066  1.0451]
 [-1.538  -0.9314  0.8158]
 [ 0.659   0.196  -1.4613]]
```

Out[27]:
```
array([[-1.538 , -0.9314, -1.4613],
       [-1.2293,  0.196 , -0.2191],
       [-0.6949,  1.2335,  0.8158],
       [ 0.0377,  1.3576,  1.0451],
       [ 0.659 ,  2.1066,  2.3124]])
```

In [29]:
```
arr = randn(5, 3)
print(arr)
arr.sort(1) ## 0每列排序，1，每行排序
arr
```

```
[[ 0.418   0.5854 -0.271 ]
 [ 1.2828  0.2964 -0.3823]
 [-0.2308 -0.509   0.1124]
 [-1.0308 -0.3335 -0.453 ]
 [ 0.8622 -1.1119 -1.2908]]
```

Out[29]:
```
array([[-0.271 ,  0.418 ,  0.5854],
       [-0.3823,  0.2964,  1.2828],
       [-0.509 , -0.2308,  0.1124],
       [-1.0308, -0.453 , -0.3335],
       [-1.2908, -1.1119,  0.8622]])
```

Unique and other set logic

函数包括：unique(x),intersect1d(x,y),union1d(x,y),in1d(x,y),setdiff1d(x,y),setxor1d(x,y)

In [117…
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
print(np.unique(names))
ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
np.unique(ints)
```

```
['Bob' 'Joe' 'Will']
```

Out[117…
```
array([1, 2, 3, 4])
```

In [118…
```
sorted(set(names))
```

Out[118…
```
['Bob', 'Joe', 'Will']
```

In [119…
```
values = np.array([6, 0, 0, 3, 2, 5, 6])
np.in1d(values, [2, 3, 6])
```

Out[119…
```
array([ True, False, False,  True,  True, False,  True], dtype=bool)
```

数组的输入和输出

In [120…
```
arr = np.arange(10)
```

```
np.save('some_array', arr)
```

In [121…
```
a=np.load('some_array.npy')
```

In [122…
```
a
```

Out[122…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [30]:
```
#np.savez,多个数组
arch = np.load('array_archive.npz')
arch['b']
```

Out[30]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Saving and loading text files

In [124…
```
!type data\array_ex.txt
```

```
-0.951266864671,-0.64723096535,1.09667101587,0.400586955639
-1.40858316447,0.643109396866,-1.55032379004,-0.463958033067
0.496607504203,-1.79963950946,0.622557557274,1.30738800961
```

In [125…
```
arr = np.loadtxt('data/array_ex.txt', delimiter=',')
arr
```

Out[125…
```
array([[-0.9513, -0.6472,  1.0967,  0.4006],
       [-1.4086,  0.6431, -1.5503, -0.464 ],
       [ 0.4966, -1.7996,  0.6226,  1.3074]])
```

Linear algebra

函数包括：diag,dot,trace,det,eig,inv,pinv,qr,svd,solve,lstsq

In [126…
```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
y = np.array([[6., 23.], [-1, 7], [8, 9]])
print(x)
print(y)
x.dot(y)  # equivalently np.dot(x, y)
```

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
[[  6.  23.]
 [ -1.   7.]
 [  8.   9.]]
```

Out[126…
```
array([[  28.,   64.],
       [  67.,  181.]])
```

In [127…
```
print(np.ones(3))
np.dot(x, np.ones(3))
```

```
[ 1.  1.  1.]
```

Out[127…
```
array([  6.,  15.])
```

numpy.linalg中有更多矩阵函数。

In [128…
```
from numpy.linalg import inv, qr
X = randn(5, 5)
mat = X.T.dot(X)
inv(mat)
mat.dot(inv(mat))
q, r = qr(mat)
r
```

Out[128…
```
array([[-10.6008,   4.2297,   9.0275,  -1.2605,   4.9558],
       [  0.    ,  -3.952 ,  -4.228 ,  -4.5512,   2.3925],
       [  0.    ,   0.    ,  -3.9131,  -0.2853,  -6.2165],
       [  0.    ,   0.    ,   0.    ,  -1.8914,   0.36  ],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.536 ]])
```

Random number generation

seed,permutation,shuffle,rand,randit,randn,vinomial,normal,beta,chisquare,gamma,uniform等

In [129…
```
samples = np.random.normal(size=(4, 4))
samples
```

Out[129…
```
array([[-1.2618,  1.1544, -0.6608,  0.2324],
       [-0.5089,  0.6056,  0.755 , -1.4125],
       [ 1.3507,  0.6583,  0.7353,  1.5432],
       [-1.2186, -0.7663,  0.2028, -0.1586]])
```

作业

1. 给定一个二维数组，每行是一个向量。找出所有不同的行。 比如：

`x = np.array([[1., 2., 3.], [4., 5., 6.],[1., 2., 3.], [4., 5., 6.]])`,则`x[:2]`，即x的第0,1行，`array([[ 1.,  2.,  3.],[ 4.,  5.,  6.]])`就是要找到行。

备注：请不要直接调用numpy.unique

1. a).产生正态数据数据向量 Z,长度为T， 建立一个二维数组，使其第一行为 (Z[0],Z[1],Z[2])，然后每一行都后移一位 （因此最后一行为 (Z[T-3],Z[T-2],Z[T-1]);b). 计算每列数据的样本均值，方差，各列之间的协方差。

2. a). 随机生成一个1000行50列二维数组，每个元素等概率取值0，1，2，并将该数组按第1列递增和第2列递减对每行排序，即先按第一列递增排序，然后在第一列每组中， 按第二列递减排序。 b). 编一个函数sortbycols(data,cols,descending)，实现行排序。给定参数 data是二维数值，cols是数或list,给出需要排序的列，descending是布尔值或布尔向量，如果是一个布尔值，则所有列都按该顺序排，如果descending是和cols等长的list，则各列按descending给定的顺序排。

3. 熟悉学习numpy中的现金流函数, np.pv,np.fv,np.nper,np.pmt,np.rate, 然后编制一个函数，计算给定现金流(array)，利率(array),任意时刻(t)的现金价值。