numpy 入门

In [1]:
```
import numpy as np
```

基于 NumPy 比纯 Python 处理的速度更快, 占用的内存更少

In [2]:
```
my_arr = np.arange(1000000)
my_list = list(range(1000000))
%time my_arr2 = my_arr ** 2
%time my_list2 = [x ** 2 for x in my_list]
```

```
Wall time: 2.99 ms
Wall time: 339 ms
```

ndarray : 一种多维数组对象

In [3]:
```
data = np.random.randn(2,3)  # 随机生成一个2x3的服从正态分布的随机矩阵
data, data * 10, data + data # 所有元素都x10，所有元素各自相加
```

Out[3]:
```
(array([[ 0.61755716,  0.27939505, -0.13490957],
        [ 0.8987287 , -0.50354707,  0.92604458]]),
 array([[ 6.17557158,  2.79395053, -1.34909571],
        [ 8.987287  , -5.03547071,  9.26044575]]),
 array([[ 1.23511432,  0.55879011, -0.26981914],
        [ 1.7974574 , -1.00709414,  1.85208915]]))
```

每个ndarray数组都有一个 *shape* 和一个 *dtype* 属性

> ***shape*** : 表示各维度大小的元组
> ***dtype*** : 用于说明数组数据类型的对象
> ***ndim*** : 用于说明数组的维数
> ***arr.shape, arr.dtype, arr.ndim***

In [4]:
```
data.shape, data.dtype, data.ndim
```

Out[4]:
```
((2, 3), dtype('float64'), 2)
```

ndarray 的创建

np.array(object), np.asarray(object)

In [5]:
```
data1 = [6, .5, 8]
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr1 = np.array(data1)
arr2 = np.array(data2)
(arr1, arr1.ndim, arr1.shape), (arr2, arr2.ndim, arr2.shape)
```

Out[5]:
```
((array([6. , 0.5, 8. ]), 1, (3,)),
 (array([[1, 2, 3, 4],
         [5, 6, 7, 8]]), 2, (2, 4)))
```

np.array(object), np.asarray(object) 区别 :

> 数据源非ndarray : 两者都开辟新的内存复制数据
> 数据源为ndarray : array 占用新内存就行复制, asarray 不进行复制直接引用源内存

In [6]:
```
data = [1,2,3]
arr1 = np.array(data)
arr2 = np.asarray(data)
data[1] = 4
arr1, arr2
```

Out[6]:
```
(array([1, 2, 3]), array([1, 2, 3]))
```

In [7]:
```
data = np.zeros(3)
arr1 = np.array(data)
arr2 = np.asarray(data)
data[1] = 999
data, arr1, arr2
```

Out[7]:
```
(array([  0., 999.,   0.]), array([0., 0., 0.]), array([  0., 999.,   0.]))
```

np.ones(shape), np.zeros(shape), np.empty(shape), np.full(shape, fill_value)

> *np.empty*: 给变量分配一个没有任何值的数组空间，可以定义这个空间的shape

In [8]:
```
shape = (3, 3)
arr0 = np.zeros(shape = shape)
arr1 = np.ones(shape = shape)
arre = np.empty(shape = shape)
arrf = np.full(shape = shape, fill_value= 999)
arr0, arr1, arre, arrf
```

Out[8]:
```
(array([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]),
 array([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]),
 array([[0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
        [0.00000000e+000, 0.00000000e+000, 6.58095440e-321],
        [6.23058028e-307, 1.69115935e-306, 1.89142822e-307]]),
 array([[999, 999, 999],
        [999, 999, 999],
        [999, 999, 999]]))
```

np.ones_like(arr), np.zeros_like(arr), np.empty_like(arr), np.full_like(arr, fill_value)

In [9]:
```
data = [[1,2,3], [4,5,6], [7,8,9]]
arr1 = np.ones_like(data)
arrf = np.full_like(data, fill_value=999)
arr1, arrf
```

Out[9]:
```
(array([[1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]]),
 array([[999, 999, 999],
        [999, 999, 999],
        [999, 999, 999]]))
```

np.eye(N), np.identity(N)

In [10]:
```
arre = np.eye(3)
arri = np.identity(3)
arre, arri, arre == arri
```

Out[10]:
```
(array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[ True,  True,  True],
        [ True,  True,  True],
        [ True,  True,  True]]))
```

np.arange(start, stop, step)

In [11]:
```
start, stop, step = 1, 10, 2
arrr = np.arange(start = start, stop = stop, step = step)
arrr
```

Out[11]:
```
array([1, 3, 5, 7, 9])
```

np.random.randn(d0, d1, d2, ...)

In [12]:
```
np.random.randn(2,3,4)
```

Out[12]:
```
array([[[ 1.86999   ,  0.0390554 , -0.50994802, -1.78571159],
        [-1.16586811, -1.80043925,  0.12937635, -0.51073928],
        [ 0.90820582,  0.16733293,  1.37816101, -1.99030047]],

       [[-0.37128402,  1.01049711,  1.16644033, -0.42575843],
        [ 1.04713973,  0.30232925,  1.05983582,  1.48025048],
        [ 0.65822568,  1.00390684,  1.51106941,  1.22224557]]])
```

ndarray 的数据类型

np.array(object, dtype), arr.astype(dtype)

In [13]:
```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
arr1.dtype, arr2.dtype
```

Out[13]:
```
(dtype('float64'), dtype('int32'))
```

In [14]:
```
arr = np.array([1, 2, 3, 4, 5])
float_arr = arr.astype(np.float64)
arr.dtype, float_arr.dtype
```

Out[14]:
```
(dtype('int32'), dtype('float64'))
```

类型	类型代码	说明
int8, uint8	i1, u1	有符号和无符号的8位（1个字节）整型
int16, uint16	i2, u2	有符号和无符号的16位（2个字节）整型
int32, uint32	i4, u4	有符号和无符号的32位（4个字节）整型
int64, uint64	i8, u8	有符号和无符号的64位（8个字节）整型
float16	f2	半精度浮点数
float32	f4 or f	单精度浮点数
float64	f8 or d	双精度浮点数
float128	f16 or g	扩展精度浮点数
complex64, complex128, complex256	c8, c16, c32	分别用两个32位、64位和128位浮点数表示的复数

类型	类型代码	说明
bool	?	存储 True 和 False 值得布尔类型
objec	O	Python 的对象类型
string_	S	固定长度的字符串类型
unicode_	U	固定长度的 unicode 类型

类型转换

float → int : 截断取整

In [15]:
```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr, arr.astype(np.int32) #截断取整
```

Out[15]:
```
(array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1]), array([ 3, -1, -2,  0, 12, 10]))
```

string_ → float : 截取

> NumPy 的字符串数据是**大小固定**的，使用 **np.string_** 类型时会发生**截取**

In [16]:
```
numeric_strings = np.array(['1.2555555555', '-9.6', '42'], dtype=np.string_)
numeric_strings.astype(float), numeric_strings.dtype
```

Out[16]:
```
(array([ 1.25555556, -9.6       , 42.        ]), dtype('S12'))
```

ndarray 的运算

*arr1+arr2, arr1-arr2, arr1*arr2, arr1/arr2*

> 矢量化运算：大小相等的数组之间的任何算术运算都会将运算应用到元素级

In [17]:
```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
arr, arr * arr, arr - arr,  arr / arr
```

Out[17]:
```
(array([[1., 2., 3.],
        [4., 5., 6.]]),
 array([[ 1.,  4.,  9.],
        [16., 25., 36.]]),
 array([[0., 0., 0.],
        [0., 0., 0.]]),
 array([[1., 1., 1.],
        [1., 1., 1.]]))
```

*arr+num, arr-num, arrnum, arr/num, num/arr, arr*num*

> 数组与标量的算术运算：会将标量值传播到数组的各个元素

In [18]:
```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
arr, arr + 1, arr / 0.5, 1 / arr, arr**2
```

Out[18]:
```
(array([[1., 2., 3.],
        [4., 5., 6.]]),
 array([[2., 3., 4.],
        [5., 6., 7.]]),
 array([[ 2.,  4.,  6.],
        [ 8., 10., 12.]]),
 array([[1.        , 0.5       , 0.33333333],
        [0.25      , 0.2       , 0.16666667]]),
 array([[ 1.,  4.,  9.],
        [16., 25., 36.]]))
```

arr2>arr1

> 布尔运算：大小相同的数组之间的比较会生成布尔值数组

In [19]:
```
arr1 = np.array([[1., 2., 3.], [4., 5., 6.]])
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
arr2 > arr
```

Out[19]:
```
array([[False,  True, False],
       [ True, False,  True]])
```

切片与索引

基本的切片和索引

> *arr[d1, d2, ...], arr[d1][d2]...*

In [20]:
```
arr = np.arange(10)
arr, arr[5], arr[5:8], arr[-5:-1], arr[-5:10]
```

Out[20]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 5,
 array([5, 6, 7]),
 array([5, 6, 7, 8]),
 array([5, 6, 7, 8, 9]))
```

切片的赋值（广播）

In [21]:
```
arr[5:8] = 12
arr
```

Out[21]:
```
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

基本切片的赋值：**直接引用内存（原地操作）**

In [22]:
```
arr = np.arange(10)
print(arr)
arr_slice = arr[5:]
arr_slice[:] = 0
print(arr)
```

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

如果想得到切片的副本而不是引用，要明确进行复制操作：***arr.copy()***

In [23]:
```
arr = np.arange(10)
print(arr)
arr_copy = arr[5:].copy()
arr_copy[:] = 0
print(arr)
print(arr_copy)
```

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

对多维数组的切片

In [24]:
```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
arr2d, arr2d[1:], arr2d[1:][1:], arr2d[1:, 1:]
```

Out[24]:
```
(array([[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]]),
 array([[ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]]),
 array([[ 7,  8,  9],
        [10, 11, 12]]),
 array([[ 5,  6],
        [ 8,  9],
        [11, 12]]))
```

In [25]:
```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
arr3d, arr3d[0, :, :], arr3d[:, 0, :], arr3d[:, :, 0]
```

Out[25]:
```
(array([[[ 1,  2,  3],
         [ 4,  5,  6]],

        [[ 7,  8,  9],
         [10, 11, 12]]]),
 array([[1, 2, 3],
        [4, 5, 6]]),
 array([[1, 2, 3],
        [7, 8, 9]]),
 array([[ 1,  4],
        [ 7, 10]]))
```

在多维数组中, 如果省略了后面的索引, 则返回一个维度低一点的数组 : ***arr[d1].ndim = arr.ndim - 1***

In [26]:
```
arr3d[0], arr3d[0, :, :], arr3d[0][0], arr3d[0, 0, :]
```

Out[26]:
```
(array([[1, 2, 3],
        [4, 5, 6]]),
 array([[1, 2, 3],
        [4, 5, 6]]),
 array([1, 2, 3]),
 array([1, 2, 3]))
```

高维数组切片的赋值

In [27]:
```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d)
arr2d[:2, 1:] = 0
print(arr2d)
```

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

布尔型索引

> ***arr[condition]***

In [28]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
courses = np.array(['Maths', 'English', 'Physics', 'Python'])
data = np.random.randint(90, 100, (4, 7))
names, data
```

Out[28]:
```
(array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4'),
 array([[93, 95, 94, 94, 91, 90, 92],
        [90, 96, 95, 92, 99, 99, 90],
        [97, 99, 95, 94, 99, 93, 95],
        [90, 95, 95, 90, 98, 91, 95]]))
```

布尔型数组

In [29]:
```
names == 'Bob', courses == 'Python'
```

Out[29]:
```
(array([ True, False, False,  True, False, False, False]),
 array([False, False, False,  True]))
```

布尔型数组的长度必须跟被索引的轴长度一致

In [30]:
```
data[:, names == 'Bob'], data[courses == 'Python', :], \
data[courses == 'Python', names == 'Bob']
```

Out[30]:
```
(array([[93, 94],
        [90, 92],
        [97, 94],
        [90, 90]]),
 array([[90, 95, 95, 90, 98, 91, 95]]),
 array([90, 90]))
```

操作符 '~'

In [31]:
```
data[~(courses == 'Python')]
```

Out[31]:
```
array([[93, 95, 94, 94, 91, 90, 92],
       [90, 96, 95, 92, 99, 99, 90],
       [97, 99, 95, 94, 99, 93, 95]])
```

操作符 '&', '|'

> Python 关键字 'and' 和 'or' 在布尔型数组中无效, 要使用 '&' 与 '|'

In [32]:
```
mask = (courses == 'Maths') | (courses == 'Physics')
data[mask]
```

Out[32]:
```
array([[93, 95, 94, 94, 91, 90, 92],
       [97, 99, 95, 94, 99, 93, 95]])
```

通过布尔型索引选取数组中的数据，将总是创建数据的副本

> 基本切片：引用内存；布尔索引：创建副本

In [33]:
```
data = np.random.randint(90, 100, (4, 7))
mask = (courses == 'Maths') | (courses == 'Physics')
print(mask)
data_mask = data[mask]
print(data_mask)
data_mask[:] = 0
print(data)
```

```
[ True False  True False]
[[99 97 98 92 92 90 99]
 [94 94 95 94 91 94 98]]
[[99 97 98 92 92 90 99]
 [95 99 92 99 93 99 97]
 [94 94 95 94 91 94 98]
 [91 94 95 94 96 90 99]]
```

In [34]:
```
data_slice = data[0:2, :]
print(data_slice)
data_slice[:] = 0
print(data)
```

```
[[99 97 98 92 92 90 99]
 [95 99 92 99 93 99 97]]
[[ 0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0]
 [94 94 95 94 91 94 98]
 [91 94 95 94 96 90 99]]
```

布尔型切片赋值

In [35]:
```
data = np.random.randint(90, 100, (4, 7))
print(data)
data[data < 95] = 0
print(data)
```

```
[[92 95 92 93 93 90 96]
 [90 98 99 97 91 96 95]
 [94 95 97 97 98 98 96]
 [90 97 96 97 96 93 95]]
[[ 0 95  0  0  0  0 96]
 [ 0 98 99 97  0 96 95]
 [ 0 95 97 97 98 98 96]
 [ 0 97 96 97 96  0 95]]
```

花式索引（Fancy Indexing）

> *arr[list1, list2, ...]*

In [36]:
```
arr = np.empty((8, 4))
for i in range(8):
    for j in range(4):
        arr[i,j] = i+j
arr = arr.astype(np.int32)
arr
```

Out[36]:
```
array([[ 0,  1,  2,  3],
       [ 1,  2,  3,  4],
       [ 2,  3,  4,  5],
       [ 3,  4,  5,  6],
       [ 4,  5,  6,  7],
       [ 5,  6,  7,  8],
       [ 6,  7,  8,  9],
       [ 7,  8,  9, 10]])
```

利用整数数组进行索引

In [37]:
```
arr[[4, 3, 0, 6]], arr[[-3, -5, -7]]
```

Out[37]:
```
(array([[4, 5, 6, 7],
        [3, 4, 5, 6],
        [0, 1, 2, 3],
        [6, 7, 8, 9]]),
 array([[5, 6, 7, 8],
        [3, 4, 5, 6],
        [1, 2, 3, 4]]))
```

一次传入多个索引数组：返回一个一维数组，其中的元素对应各个索引元组

In [38]:
```
arr[[4, 3, 0, 6], [2, 1, 0, 3]]
# 最终选出的是元素 (4, 2), (3, 1), (0, 0), (6, 3)
# 无论数组是多少维的，花式索引总是一维的
```

Out[38]:
```
array([6, 4, 0, 9])
```

In [39]:
```
arr[[4, 3, 0, 6]][:, [2, 1, 0, 3]]
```

Out[39]:
```
array([[6, 5, 4, 7],
       [5, 4, 3, 6],
       [2, 1, 0, 3],
       [8, 7, 6, 9]])
```

通过花式索引选取数组中的数据，将总是创建数据的副本

> 基本切片：引用内存；布尔索引：复制副本；花式索引：复制副本

In [40]:
```
arr_fancy = arr[range(0,4)][:, range(0,4)]
print(arr_fancy)
arr_fancy[:] = 0
print(arr)
```

```
[[0 1 2 3]
 [1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
[[ 0  1  2  3]
 [ 1  2  3  4]
 [ 2  3  4  5]
 [ 3  4  5  6]
 [ 4  5  6  7]
 [ 5  6  7  8]
 [ 6  7  8  9]
 [ 7  8  9 10]]
```

数组转置和轴对换

arr.T：转置

In [41]:
```
arr = np.arange(15).reshape((3, 5))
arr, arr.T
```

Out[41]:
```
(array([[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14]]),
 array([[ 0,  5, 10],
        [ 1,  6, 11],
        [ 2,  7, 12],
        [ 3,  8, 13],
        [ 4,  9, 14]]))
```

np.dot(arr1, arr2)：矩阵内积

In [42]:
```
np.dot(arr, arr.T)  #3x5 5x3 → 3x3
```

Out[42]:
```
array([[ 30,  80, 130],
       [ 80, 255, 430],
       [130, 430, 730]])
```

arr.transpose(axes)：轴对换

> ***axes***：轴编号序列的元组

In [43]:
```
arr = np.arange(15).reshape((3, 5))
arr, arr.transpose((0, 1)), arr.transpose((1, 0)) # 第1轴与第2轴互换位置
```

Out[43]:
```
(array([[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14]]),
 array([[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14]]),
 array([[ 0,  5, 10],
        [ 1,  6, 11],
        [ 2,  7, 12],
        [ 3,  8, 13],
        [ 4,  9, 14]]))
```

In [44]:
```
arr = np.arange(24).reshape((2, 3, 4))
arr, arr.transpose((1,0,2)) # 第1轴与第2轴互换，第3轴不变
```

Out[44]:
```
(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]]]),
 array([[[ 0,  1,  2,  3],
         [12, 13, 14, 15]],

        [[ 4,  5,  6,  7],
         [16, 17, 18, 19]],

        [[ 8,  9, 10, 11],
         [20, 21, 22, 23]]]))
```

arr.swapaxes(axis1, axis2)：两个轴对换

> ***axis***：轴编号

In [45]:
```
arr = np.arange(24).reshape((2, 3, 4))
arr, arr.swapaxes(0, 1) # 第1轴与第2轴互换，第3轴不变
```

Out[45]:
```
(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]]]),
 array([[[ 0,  1,  2,  3],
         [12, 13, 14, 15]],

        [[ 4,  5,  6,  7],
         [16, 17, 18, 19]],

        [[ 8,  9, 10, 11],
         [20, 21, 22, 23]]]))
```

通用的一元函数

In [46]:
```
arr = np.arange(1,10,2)
arr
```

Out[46]:
```
array([1, 3, 5, 7, 9])
```

np.sqrt(arr), np.exp(arr)：平方根，平方

In [47]:
```
np.sqrt(arr), np.square(arr)
```

Out[47]:
```
(array([1.        , 1.73205081, 2.23606798, 2.64575131, 3.        ]),
 array([ 1,  9, 25, 49, 81], dtype=int32))
```

np.exp(arr), np.log(arr), np.log1p(arr)：指数，对数，log(1+x)

In [48]:
```
np.exp(arr), np.log(arr), np.log1p(arr)
```

Out[48]:
```
(array([2.71828183e+00, 2.00855369e+01, 1.48413159e+02, 1.09663316e+03,
        8.10308393e+03]),
 array([0.        , 1.09861229, 1.60943791, 1.94591015, 2.19722458]),
 array([0.69314718, 1.38629436, 1.79175947, 2.07944154, 2.30258509]))
```

np.sign(arr)：符号函数

In [49]:
```
arr = np.random.randn(5) * 5
arr
```

Out[49]:
```
array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986])
```

In [50]:
```
arr, np.sign(arr)
```

Out[50]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([-1.,  1., -1.,  1.,  1.]))
```

np.ceil(arr), np.floor(arr), np.rint(arr)：进一取整，退一取整，四舍五入

In [51]:
```
arr, np.ceil(arr), np.floor(arr), np.rint(arr)
```

Out[51]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([-5.,  5., -1.,  2.,  2.]),
 array([-6.,  4., -2.,  1.,  1.]),
 array([-6.,  4., -1.,  1.,  1.]))
```

np.modf(arr)：返回小数、整数

In [52]:
```
part, main = np.modf(arr)
arr, main, part
```

Out[52]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([-5.,  4., -1.,  1.,  1.]),
 array([-0.72033251,  0.14975286, -0.29811492,  0.15525507,  0.39684986]))
```

np.abs(arr), np.fabs(arr)：绝对值，非复数绝对值

In [53]:
```
arr, np.abs(arr), np.fabs(arr)
```

Out[53]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([5.72033251, 4.14975286, 1.29811492, 1.15525507, 1.39684986]),
 array([5.72033251, 4.14975286, 1.29811492, 1.15525507, 1.39684986]))
```

np.isnan(arr)：判断非数字

In [54]:
```
arr, np.isnan(arr)
```

Out[54]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([False, False, False, False, False]))
```

np.isfinite(arr), np.isinf(arr)：判断有穷，判断无穷

In [55]:
```
arr, np.isfinite(arr), np.isinf(arr)
```

Out[55]:
```
(array([-5.72033251,  4.14975286, -1.29811492,  1.15525507,  1.39684986]),
 array([ True,  True,  True,  True,  True]),
 array([False, False, False, False, False]))
```

np.cos(arr), np.sin(arr), np.tan(arr)：三角函数

np.arccos(arr), np.arcsin(arr), np.arctan(arr)：反三角函数

np.logical_not(arr)：逻辑非 ~

In [56]:
```
arr = np.random.randint(0, 2, (5))
arr, np.logical_not(arr)
```

Out[56]:
```
(array([0, 1, 1, 1, 0]), array([ True, False, False, False,  True]))
```

通用的二元函数

In [57]:
```
arr1 = np.random.randint(0,10,(5))
arr2 = np.random.randint(0,10,(5))
arr1, arr2
```

Out[57]:
```
(array([6, 2, 3, 8, 4]), array([3, 9, 0, 1, 0]))
```

加减乘除：

> *np.add(arr1, arr2), np.subtract(arr1, arr2)*
> *np.multiply(arr1, arr2), np.divide(arr1, arr2)*

In [58]:
```
arr1, arr2, \
np.add(arr1, arr2), np.subtract(arr1, arr2), \
np.multiply(arr1, arr2), np.divide(arr1, arr2)
```

```
<ipython-input-58-7d5cbea6c653>:3: RuntimeWarning: divide by zero encountered in true_divide
  np.multiply(arr1, arr2), np.divide(arr1, arr2)
```

Out[58]:
```
(array([6, 2, 3, 8, 4]),
 array([3, 9, 0, 1, 0]),
 array([ 9, 11,  3,  9,  4]),
 array([ 3, -7,  3,  7,  4]),
 array([18, 18,  0,  8,  0]),
 array([2.        , 0.22222222,        inf, 8.        ,        inf]))
```

除法的取整和取余：

> *np.floor_divide(arr1, arr2), np.mod(arr1, arr2)*：各返回除法整数、除法余数
> *np.divmod(arr1, arr2)*：返回除法整数和余数

In [59]:
```
arr1, arr2, np.floor_divide(arr1, arr2), np.mod(arr1, arr2), np.divmod(arr1, arr2)
```

```
<ipython-input-59-f630586a71d1>:1: RuntimeWarning: divide by zero encountered in floor_divide
  arr1, arr2, np.floor_divide(arr1, arr2), np.mod(arr1, arr2), np.divmod(arr1, arr2)
<ipython-input-59-f630586a71d1>:1: RuntimeWarning: divide by zero encountered in remainder
  arr1, arr2, np.floor_divide(arr1, arr2), np.mod(arr1, arr2), np.divmod(arr1, arr2)
<ipython-input-59-f630586a71d1>:1: RuntimeWarning: divide by zero encountered in divmod
  arr1, arr2, np.floor_divide(arr1, arr2), np.mod(arr1, arr2), np.divmod(arr1, arr2)
```

Out[59]:
```
(array([6, 2, 3, 8, 4]),
 array([3, 9, 0, 1, 0]),
 array([2, 0, 0, 8, 0], dtype=int32),
 array([0, 2, 0, 0, 0], dtype=int32),
 (array([2, 0, 0, 8, 0], dtype=int32), array([0, 2, 0, 0, 0], dtype=int32)))
```

最大值和最小值：

> *np.maximum(arr1, arr2), np.fmax(arr1, arr2)*
> *np.minimum(arr1, arr2), np.fmin(arr1, arr2)*

In [60]:
```
a = np.array([4, 2 , np.nan,3, 1])
b = np.array([2, 3, 4, 5, np.nan])
a, b, np.maximum(a, b), np.fmax(a, b)  #fmax, fmin 将忽略NaN
```

Out[60]:
```
(array([ 4.,  2., nan,  3.,  1.]),
 array([ 2.,  3.,  4.,  5., nan]),
 array([ 4.,  3., nan,  5., nan]),
 array([4., 3., 4., 5., 1.]))
```

np.copysign(arr1, arr2) : copy sign of arr2 to arr1

In [61]:
```
a = np.random.randint(-9, 0, (5))
b = np.random.randint(0, 9, (5))
a, b, np.copysign(b, a)
```

Out[61]:
```
(array([-5, -6, -5, -6, -5]),
 array([3, 4, 1, 2, 6]),
 array([-3., -4., -1., -2., -6.]))
```

np.power(arr1, arr2)：对 arr_1 ， 根据 arr_2 中的元素, 计算对应的 $arr_1^{arr_2}$

In [62]:
```
arr1, arr2, np.power(arr1, arr2)
```

Out[62]:
```
(array([6, 2, 3, 8, 4]),
 array([3, 9, 0, 1, 0]),
 array([216, 512,   1,   8,   1], dtype=int32))
```

比较：

> *np.greater(arr1, arr2), np.greater_equal(arr1, arr2)*
> *np.less(arr1, arr2), np.less_equal(arr1, arr2)*
> *np.equal(arr1, arr2)np.not_equal(arr1, arr2)*

In [63]:
```
arr1, arr2, np.greater_equal(arr1, arr2), np.less_equal(arr1, arr2)
```

Out[63]:
```
(array([6, 2, 3, 8, 4]),
 array([3, 9, 0, 1, 0]),
 array([ True, False,  True,  True,  True]),
 array([False,  True, False, False, False]))
```

逻辑运算：

np.logical_and(arr1, arr2)：逻辑与 &
np.logical_or(arr1, arr2)：逻辑或 |
np.logical_xor(arr1, arr2)：逻辑异或 ^

In [64]:
```
arr1 = np.random.randint(0, 2, (5))
arr2 = np.random.randint(0, 2, (5))
arr1, arr2, \
np.logical_and(arr1, arr2), np.logical_or(arr1, arr2), np.logical_xor(arr1, arr2)
```

Out[64]:
```
(array([0, 0, 1, 1, 0]),
 array([1, 0, 0, 1, 0]),
 array([False, False, False,  True, False]),
 array([ True, False,  True,  True, False]),
 array([ True, False,  True, False, False]))
```

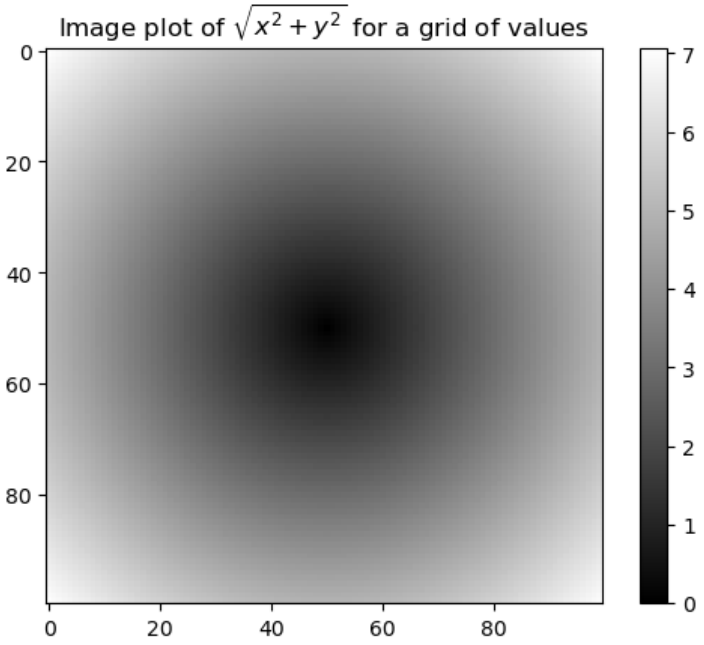
利用数组进行数据处理

np.meshgrid(arr1, arr2)：基于两个一维数组arr1和arr2，返回两个二维数组

In [65]:
```
points = np.arange(-5, 5, 0.1)
x, y = np.meshgrid(points, points)
x, y, x.T == y
```

Out[65]:
```
(array([[-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9],
        [-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9],
        [-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9],
        ...,
        [-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9],
        [-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9],
        [-5. , -4.9, -4.8, ...,  4.7,  4.8,  4.9]]),
 array([[-5. , -5. , -5. , ..., -5. , -5. , -5. ],
        [-4.9, -4.9, -4.9, ..., -4.9, -4.9, -4.9],
        [-4.8, -4.8, -4.8, ..., -4.8, -4.8, -4.8],
        ...,
        [ 4.7,  4.7,  4.7, ...,  4.7,  4.7,  4.7],
        [ 4.8,  4.8,  4.8, ...,  4.8,  4.8,  4.8],
        [ 4.9,  4.9,  4.9, ...,  4.9,  4.9,  4.9]]),
 array([[ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        ...,
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True]]))
```

In [66]:
```
z = np.sqrt(x**2 + y**2)
import matplotlib.pyplot as plt
plt.imshow(z, cmap=plt.cm.gray)
plt.colorbar()
plt.title("Image plot of $\sqrt{x^2 + y^2}$ for a grid of values")
plt.show()
```



np.where(condition, x, y) : np.where(...).shape = condition.shape

$$condition = True \rightarrow x, condition = False \rightarrow y$$

In [67]:
```
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])
np.where(cond, xarr, yarr)
```

Out[67]:
```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

In [68]:
```
arr = np.random.randn(4, 4)
arr, arr > 0, np.where(arr > 0, 2, -2) # 将arr>0的元素替换为2， 否则替换为-2
```

Out[68]:
```
(array([[-1.38574074,  0.36308053,  1.70333481, -1.28078003],
        [ 1.11809371,  0.6485143 ,  0.86350574, -0.0687285 ],
        [-0.12629966,  0.89419753, -0.56168284,  0.73884436],
        [-0.26990796,  0.27744503,  0.6568969 ,  0.02221542]]),
 array([[False,  True,  True, False],
        [ True,  True,  True, False],
        [False,  True, False,  True],
        [False,  True,  True,  True]]),
 array([[-2,  2,  2, -2],
        [ 2,  2,  2, -2],
        [-2,  2, -2,  2],
        [-2,  2,  2,  2]]))
```

In [69]:
```
arr, np.where(arr > 0, 2, arr) # 将arr>0的元素替换为2，arr<0的不变
```

Out[69]:
```
(array([[-1.38574074,  0.36308053,  1.70333481, -1.28078003],
        [ 1.11809371,  0.6485143 ,  0.86350574, -0.0687285 ],
        [-0.12629966,  0.89419753, -0.56168284,  0.73884436],
        [-0.26990796,  0.27744503,  0.6568969 ,  0.02221542]]),
 array([[-1.38574074,  2.        ,  2.        , -1.28078003],
        [ 2.        ,  2.        ,  2.        , -0.0687285 ],
        [-0.12629966,  2.        , -0.56168284,  2.        ],
        [-0.26990796,  2.        ,  2.        ,  2.        ]]))
```

统计方法

可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算

求平均、求和

np.mean(arr, axis), arr.mean(axis) np.sum(arr, axis), arr.sum(axis)

In [70]:
```
arr = np.arange(24).reshape((2, 3, 4))
arr, arr.mean(), np.mean(arr), arr.sum(), np.sum(arr)
```

Out[70]:
```
(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]]]),
 11.5,
 11.5,
 276,
 276)
```

In [71]:
```
arr, np.mean(arr, axis=0), np.sum(arr, axis=0) # 对第1维进行取平均/求和处理
# 0+12   1+13   2+14   3+15
# 4+16   5+17   6+18   7+19
# 8+20   9+21   10+22  11+23
```

Out[71]:
```
(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]]]),
 array([[ 6.,  7.,  8.,  9.],
        [10., 11., 12., 13.],
        [14., 15., 16., 17.]]),
 array([[12, 14, 16, 18],
        [20, 22, 24, 26],
        [28, 30, 32, 34]]))
```

In [72]:
```
arr, np.mean(arr, axis=1), np.sum(arr, axis=1) # 对第2维进行取平均/求和处理
# 0+4+8      1+5+9      2+6+10     3+7+11
# 12+16+20   13+17+21   14+18+22   15+19+23
```

Out[72]:
```
(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]]]),
 array([[ 4.,  5.,  6.,  7.],
        [16., 17., 18., 19.]]),
 array([[12, 15, 18, 21],
        [48, 51, 54, 57]]))
```

In [73]:
```
arr = np.random.randn(100)
(arr > 0).sum() # Number of positive values
```

Out[73]:
```
39
```

累加、累乘

np.cumsum(arr, axis), arr.cumsum(axis)：累加
np.cumprod(arr, axis), arr.cumprod(axis)：累乘

In [74]:
```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
arr, arr.cumsum(), arr.cumprod()
```

Out[74]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 array([ 0,  1,  3,  6, 10, 15, 21, 28], dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0], dtype=int32))
```

In [75]:
```
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
arr, np.cumsum(arr, axis=0), np.cumprod(arr, axis=1)
```

Out[75]:
```
(array([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]]),
 array([[ 0,  1,  2],
        [ 3,  5,  7],
        [ 9, 12, 15]], dtype=int32),
 array([[  0,   0,   0],
        [  3,  12,  60],
        [  6,  42, 336]], dtype=int32))
```

标准差、方差

> ***np.std(arr, axis), arr.std(axis)***：标准差
> ***np.var(arr, axis), arr.var(axis)***：方差

In [76]:
```
arr = np.random.randn(10000)
arr.std(), arr.var()
```

Out[76]:
```
(1.0108654484477226, 1.0218489548654153)
```

In [77]:
```
arr = np.random.randint(10, 100, (3, 4))
arr, np.std(arr, axis=0), np.var(arr, axis=1)
```

Out[77]:
```
(array([[66, 80, 28, 18],
        [45, 55, 57, 40],
        [66, 30, 29, 76]]),
 array([ 9.89949494, 20.41241452, 13.4412301 , 23.9072281 ]),
 array([662.    ,  49.1875, 443.1875]))
```

最值及其索引

> ***np.max(arr, axis), arr.max(axis)***：最大值
> ***np.argmax(arr, axis), arr.argmax(axis)***：最大值索引（从0开始）
>
> ***np.min(arr, axis), arr.min(axis)***：最小值
> ***np.argmin(arr, axis), arr.argmin(axis)***：最小值索引（从0开始）

In [78]:
```
arr = np.random.randint(10, 100, (3, 4))
arr, np.max(arr, axis=0), np.argmax(arr, axis=0)
```

Out[78]:
```
(array([[96, 69, 22, 95],
        [43, 38, 95, 73],
        [80, 13, 23, 90]]),
 array([96, 69, 95, 95]),
 array([0, 0, 1, 0], dtype=int64))
```

用于布尔类型的方法

> ***np.any(arr, axis), arr.any(axis)***：检查数组中是否存在一个或多个True
> ***np.all(arr, axis), arr.all(axis)***：检查数组中所有值是否都是True

In [79]:
```
bools = np.array([False, False, True, False])
bools.any(), bools.all()
```

Out[79]:
```
(True, False)
```

In [80]:
```
arr = (np.random.randint(10, 100, (3, 4))>50)
arr, np.any(arr, axis=0), np.all(arr, axis=0)
```

Out[80]:
```
(array([[ True,  True,  True,  True],
        [ True,  True,  True,  True],
        [False,  True,  True,  True]]),
 array([ True,  True,  True,  True]),
 array([False,  True,  True,  True]))
```

排序

> *np.sort(arr, axis), arr.sort(axis)*：排序

np.sort返回排序副本，arr.sort()修改数组本身

In [81]:
```
arr = np.random.randn(6)
print(arr)
arr.sort()
print(arr)
```

```
[-0.6477684   0.73474585 -0.88816762  1.67601792 -1.00884885 -1.31953076]
[-1.31953076 -1.00884885 -0.88816762 -0.6477684   0.73474585  1.67601792]
```

In [82]:
```
arr = np.random.randint(0, 10, (3, 4))
print(arr)
print(arr.sort(axis=1))
print(arr)
```

```
[[0 3 2 7]
 [1 5 0 3]
 [6 3 0 2]]
None
[[0 2 3 7]
 [0 1 3 5]
 [0 2 3 6]]
```

In [83]:
```
arr = np.random.randint(0, 10, (3, 4))
print(arr)
print(np.sort(arr, axis=1))
print(arr)
```

```
[[8 4 4 3]
 [8 4 9 2]
 [3 9 4 4]]
[[3 4 4 8]
 [2 4 8 9]
 [3 4 4 9]]
[[8 4 4 3]
 [8 4 9 2]
 [3 9 4 4]]
```

集合逻辑

np.unique(arr)：找出arr中的唯一值，返回有序结果

In [84]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
np.unique(names), sorted(set(names)), np.unique(ints), sorted(set(ints))
```

Out[84]:
```
(array(['Bob', 'Joe', 'Will'], dtype='<U4'),
 ['Bob', 'Joe', 'Will'],
 array([1, 2, 3, 4]),
 [1, 2, 3, 4])
```

np.in1d(arr1, arr2)：测试arr1的值是否在arr2中，返回布尔型数组

> arr1.shape = np.in1d(arr1, arr2).shape

In [85]:
```
arr1 = np.array([6, 0, 0, 3, 2, 5, 6])
arr2 = np.array([2, 3, 6, 7])
np.in1d(arr1, arr2)
```

Out[85]:
```
array([ True, False, False,  True,  True, False,  True])
```

np.intersect1d(arr1, arr2)：计算arr1和arr2中的交集，返回有序结果

In [86]:
```
arr1 = np.array([6, 0, 0, 3, 2, 5, 6])
arr2 = np.array([2, 3, 6, 7])
np.intersect1d(arr1, arr2)
```

Out[86]:
```
array([2, 3, 6])
```

np.union1d(arr1, arr2)：计算arr1和arr2中的并集，返回有序结果

In [87]:
```
arr1 = np.array([6, 0, 0, 3, 2, 5, 6])
arr2 = np.array([2, 3, 6, 7])
np.union1d(arr1, arr2)
```

Out[87]:
```
array([0, 2, 3, 5, 6, 7])
```

np.setdiff1d(arr1, arr2)：计算arr1和arr2的差，即元素在arr1中但不在arr2中

In [88]:
```
arr1 = np.array([6, 0, 0, 3, 2, 5, 6])
arr2 = np.array([2, 3, 6, 7])
np.setdiff1d(arr1, arr2)
```

Out[88]:
```
array([0, 5])
```

np.setxor1d(arr1, arr2)：计算arr1和arr2的对称差，即元素当且仅能在其中一个数组

In [89]:
```
arr1 = np.array([6, 0, 0, 3, 2, 5, 6])
arr2 = np.array([2, 3, 6, 7])
np.setxor1d(arr1, arr2)
```

Out[89]:
```
array([0, 5, 7])
```

文件

np.save(file, arr), np.load(file)：写文件，读文件

> ***file : *.npy***

In [90]:
```
arr = np.arange(10)
np.save('some_array', arr)
```

In [91]:
```
np.load('some_array.npy')
```

Out[91]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

np.savez(file, var1 = arr1, var2 = arr2, ...)：将多个数组保存到一个未压缩的文件

中

> *file : *.npz*

np.savez_compressed(file, var1 = arr1, var2 = arr2, ...)：数据压缩

> *file : *.npz*

In [92]:
```
np.savez('array_archive.npz', a=arr, b=arr)
```

In [93]:
```
arch = np.load('array_archive.npz')
arch['b']
```

Out[93]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [94]:
```
np.savez_compressed('arrays_compressed.npz', a=arr, b=arr)
```

In [95]:
```
arch = np.load('arrays_compressed.npz')
arch['b']
```

Out[95]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

线性代数

np.diag(arr, k)：返回方阵的对角线或者将一维数组转换为对角方阵

In [96]:
```
arr1 = np.random.randint(0,10,(3,3))
arr2 = np.arange(3)
arr1, np.diag(arr1), np.diag(arr1, k=1), arr2, np.diag(arr2)
```

Out[96]:
```
(array([[3, 7, 0],
        [7, 9, 4],
        [1, 8, 2]]),
 array([3, 9, 2]),
 array([7, 4]),
 array([0, 1, 2]),
 array([[0, 0, 0],
        [0, 1, 0],
        [0, 0, 2]]))
```

np.dot(arr1, arr2)：矩阵乘法

In [97]:
```
arr1 = np.random.randint(0, 10, (3, 3))
arr2 = np.linalg.inv(arr1)
np.dot(arr1, arr2)
```

Out[97]:
```
array([[ 1.00000000e+00,  8.88178420e-16, -8.88178420e-16],
       [ 1.11022302e-16,  1.00000000e+00, -4.44089210e-16],
       [ 0.00000000e+00,  4.44089210e-16,  1.00000000e+00]])
```

np.trace(arr)：矩阵的迹

In [98]:
```
arr = np.random.randint(0,10,(3,3))
arr, np.trace(arr)
```

Out[98]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 7)
```

np.linalg.det(arr)：矩阵的行列式

In [99]:
```
arr, np.linalg.det(arr)
```

Out[99]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 179.0)
```

np.linalg.eig(arr)：矩阵的特征值和特征向量

In [100]:
```
arr, np.linalg.eig(arr)
```

Out[100]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 (array([14.394287  , -2.58656667, -4.80772032]),
  array([[-0.46692605, -0.5381644 , -0.38398256],
         [-0.7072986 ,  0.74694386, -0.53437623],
         [-0.53076243, -0.39045353,  0.75299365]])))
```

In [101]:
```
lbd, e = np.linalg.eig(arr)
np.dot(arr, e[:, 0]) , lbd[0]*e[:, 0]
```

Out[101]:
```
(array([ -6.72106756, -10.18105905,  -7.6399467 ]),
 array([ -6.72106756, -10.18105905,  -7.6399467 ]))
```

np.linalg.inv(arr)：方阵的逆

np.linalg.pinv(arr)：矩阵的More-Penrose伪逆

In [102]:
```
arr, np.linalg.inv(arr), np.linalg.pinv(arr)
```

Out[102]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 array([[-0.24581006,  0.1452514 ,  0.08379888],
        [ 0.20670391, -0.16759777,  0.13407821],
        [-0.00558659,  0.1396648 , -0.11173184]]),
 array([[-0.24581006,  0.1452514 ,  0.08379888],
        [ 0.20670391, -0.16759777,  0.13407821],
        [-0.00558659,  0.1396648 , -0.11173184]]))
```

In [103]:
```
arr2 = np.arange(1,13).reshape((3,4))
arr2, np.linalg.pinv(arr2)
```

Out[103]:
```
(array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 array([[-0.375     , -0.1       ,  0.175     ],
        [-0.14583333, -0.03333333,  0.07916667],
        [ 0.08333333,  0.03333333, -0.01666667],
        [ 0.3125    ,  0.1       , -0.1125    ]]))
```

np.linalg.qr(arr)：计算QR分解（矩阵等价变换）

> ***arr = QR***
> Q:可逆矩阵，R:上三角矩阵

In [104]:
```
Q, R = np.linalg.qr(arr)
arr, Q, R
```

Out[104]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 array([[ 0.        ,  0.99951255,  0.03121953],
        [-0.62469505,  0.02437835, -0.78048818],
        [-0.78086881, -0.01950268,  0.62439054]]),
 array([[-6.40312424, -7.80868809, -7.18399305],
        [ 0.        ,  5.00243843,  6.17747514],
        [ 0.        ,  0.        , -5.58829534]]))
```

In [105]:
```
arr, np.dot(Q, R)
```

Out[105]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 array([[0., 5., 6.],
        [4., 5., 9.],
        [5., 6., 2.]]))
```

In [106]:
```
np.linalg.det(arr), np.linalg.det(R)
```

Out[106]:
```
(179.0, 179.0)
```

np.linalg.svd(arr) : 奇异值(SVD)分解

In [107]:
```
U, S, V = np.linalg.svd(arr)
arr, U, S, V
```

Out[107]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 array([[-0.49438846,  0.41745486,  0.76243786],
        [-0.73566235,  0.26627093, -0.62281675],
        [-0.46301292, -0.86881024,  0.17546455]]),
 array([14.76104704,  4.89682835,  2.47640109]),
 array([[-0.35618842, -0.6048576 , -0.71223387],
        [-0.66961046, -0.36640706,  0.64604002],
        [-0.65172973,  0.70703122, -0.27450903]]))
```

> ***arr = U diag(S) V***
> U : 正交矩阵; V : 正交矩阵; S : 降序排序的非负数
> S : 奇异值; U : 左奇异向量; V : 右奇异向量

In [108]:
```
U.dot(np.diag(S)).dot(V)
```

Out[108]:
```
array([[-1.54427352e-15,  5.00000000e+00,  6.00000000e+00],
       [ 4.00000000e+00,  5.00000000e+00,  9.00000000e+00],
       [ 5.00000000e+00,  6.00000000e+00,  2.00000000e+00]])
```

In [109]:
```
np.dot(U,U.T), np.dot(V,V.T)
```

Out[109]:
```
(array([[1.00000000e+00, 3.54512330e-16, 1.29342340e-16],
        [3.54512330e-16, 1.00000000e+00, 2.65502607e-16],
        [1.29342340e-16, 2.65502607e-16, 1.00000000e+00]]),
 array([[ 1.00000000e+00,  1.05593218e-16, -2.36545414e-16],
        [ 1.05593218e-16,  1.00000000e+00,  1.07467290e-16],
        [-2.36545414e-16,  1.07467290e-16,  1.00000000e+00]]))
```

np.linalg.solve(A, b) : 解线性方程 $Ax = b$

In [110]:
```
A = arr
b = np.random.randint(0,10,(3))
A, b, np.linalg.solve(A, b), A.dot(np.linalg.solve(A, b))
```

Out[110]:
```
(array([[0, 5, 6],
        [4, 5, 9],
        [5, 6, 2]]),
 array([5, 4, 3]),
 array([-0.39664804,  0.76536313,  0.19553073]),
 array([5., 4., 3.]))
```

np.linalg.lstsq(A, b, rcond=None)：计算 $Ax = b$ 的最小二乘解

In [111]:
```
A = np.random.randint(0,10,(4, 3))
b = np.random.randint(0, 10, (4, 2))
A, b, np.linalg.lstsq(A, b, rcond=None)
```

Out[111]:
```
(array([[1, 5, 3],
        [1, 2, 8],
        [9, 4, 9],
        [7, 4, 2]]),
 array([[9, 3],
        [8, 6],
        [0, 3],
        [2, 4]]),
 (array([[-1.18525308, -0.15774406],
         [ 1.90640595,  0.57450974],
         [ 0.49149608,  0.41539645]]),
  array([ 7.50870761, 10.28111032]),
  3,
  array([17.23239582,  6.21628481,  3.92458118])))
```

返回值: x : 近似解; cost : 损失; n : 维度 ; S : A的奇异值

In [112]:
```
x, cost, n, S = np.linalg.lstsq(A, b, rcond=None)
b, A.dot(x)
```

Out[112]:
```
(array([[9, 3],
        [8, 6],
        [0, 3],
        [2, 4]]),
 array([[9.82126489, 3.96099399],
        [6.55952745, 4.31444705],
        [1.38181078, 4.61691052],
        [0.31184438, 2.02462346]]))
```

In [113]:
```
cost, np.sum((A.dot(x) - b)**2, 0)
```

Out[113]:
```
(array([ 7.50870761, 10.28111032]), array([ 7.50870761, 10.28111032]))
```

In [114]:
```
S, np.linalg.svd(A)[1]
```

Out[114]:
```
(array([17.23239582,  6.21628481,  3.92458118]),
 array([17.23239582,  6.21628481,  3.92458118]))
```

随机数生成

np.random.permutation(x)：产生给定序列的随机排列或者一个随机排列的序列

In [115]:
```
np.random.permutation(range(-5,0))
```

Out[115]:
```
array([-2, -4, -5, -1, -3])
```

In [116]:
```
np.random.permutation(5)
```

Out[116]:
```
array([4, 2, 3, 1, 0])
```

np.random.rand(d0, d1, ...) : 产生均匀分布的随机数

In [117]:
```
np.random.rand(3,3)
```

Out[117]:
```
array([[0.93188729, 0.19373543, 0.03242586],
       [0.0891599 , 0.11993145, 0.41048466],
       [0.31863807, 0.77394962, 0.4442849 ]])
```

np.random.uniform(low, high, size) : 产生给定范围均匀分布的随机数

In [118]:
```
np.random.uniform(0, 10, (3, 3))
```

Out[118]:
```
array([[2.60630253, 8.86000804, 3.7889038 ],
       [1.25386602, 4.55746877, 6.21932379],
       [0.48936224, 9.65439911, 9.45159411]])
```

np.random.randint(low, high, size) : 产生给定范围的整数随机数

In [119]:
```
np.random.randint(0, 10, (3, 3))
```

Out[119]:
```
array([[7, 5, 9],
       [1, 4, 6],
       [5, 6, 2]])
```

np.random.randn(size) : 产生服从标准正态分布的随机数

In [120]:
```
np.random.randn(3,3)
```

Out[120]:
```
array([[ 0.5632786 , -2.02610778, -1.7987696 ],
       [-0.27670206, -2.16489614, -0.70812467],
       [ 1.8628367 ,  0.27024486, -0.80207598]])
```

np.random.normal(loc, scale, size) : 产生服从 $N(loc, scale^2)$ 正态分布的随机数

In [121]:
```
np.random.normal(0, 1, (3,3))
```

Out[121]:
```
array([[ 1.49669768,  0.1519672 ,  0.61085676],
       [ 0.21297848, -0.32231956,  0.7621172 ],
       [-0.11691781, -1.36556747,  0.42578613]])
```

np.random.binomial(n, p, size): 产生服从 $B(n, p)$ 二项分布的随机数

In [122]:
```
np.random.binomial(100, 0.5, (3,3))
```

Out[122]:
```
array([[48, 52, 54],
       [55, 50, 51],
       [47, 44, 40]])
```

np.random.beta(a, b, size) : 产生服从 Beta 分布的随机数

np.random.chisquare(k, size) : 产生服从 $X^2(k)$ 卡方分布的随机数

np.random.gamma(shape, scale, size) : 产生服从 $G(shape, scale)$ Gamma 分布的随机数