科学计算与可视化

NumPy

NumPy简介

- 标准的Python 中用列表(list)保存一组值，可以当作数组使用。但由于列表的元素可以是任何对象，因此列表中保存的是对象的指针。对于数值运算来说，这种结构显然比较浪费内存和CPU计算。

- Python 提供了array 模块，它和列表不同，能直接保存数值，但是由于它不支持多维数组，也没有各种运算函数，因此也不适合做数值运算。

NumPy简介

- NumPy是Numerical Python的简写，是高性能科学计算和数据分析的基础包，是许多高级工具的构建基础
- 多维向量的描述和快速高效计算能力，让数组和矩阵的使用更加自然；大量实用的数学函数

```
import numpy as np
```

数组创建

- ndarray(n-dimensional array object, 简称为数组)是存储单一数据类型的多维数组

```
data = [[1,2,3,4],[5,6,7,8.2]]
arr = np.array(data)

print(arr)
# [[1.   2.   3.   4. ]
   [5.   6.   7.   8.2]]
print(arr.ndim)                    # 2
print(arr.shape)                   # (2, 4)
print(arr.dtype)                   # float64
print(type(arr))                   # <class 'numpy.ndarray'>
```

数组创建

- 显式的指定数据类型

```
arr1 = np.array([1,2,3,4], dtype=np.float64)
# [ 1.   2.   3.   4.]

arr2 = np.array([1,2,3,4], dtype=np.int32)
# [1 2 3 4]

arr3 = arr2.astype(np.float64)
# [ 1.   2.   3.   4.]
```

数组创建

- 改变数组维度大小

```
arr.shape = 4, 2

>>>
[[1. 2.]
 [3. 4.]
 [5. 6.]
 [7. 8.2]]
```

```
arr1 = arr.reshape(2,4)

>>>
[[1. 2. 3. 4.]
 [5. 6. 7. 8.2]]
```

arr和arr1共享数据存储区域

数组创建

- 通过arange函数创建数组

```
arr1 = np.arange(8)
print(arr1)
print(type(arr1))
```

```
>>>
[0 1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```

```
arr2 = np.arange(0,10,2,dtype=float)
print(arr2)
```

```
>>>
[0. 2. 4. 6. 8.]
```

数组创建

- linspace函数通过指定开始值、终值和元素个数来创建一维数组，可以通过endpoint关键字指定是否包括终值

```
print(np.linspace(0, 1, 10))
print(np.linspace(0, 1, 10, endpoint=False))
```

```
>>>
[0.         0.11111111 0.22222222 0.33333333 0.44444444
0.55555556 0.66666667 0.77777778 0.88888889 1.        ]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

数组创建

- logspace函数和linspace类似，不过它创建等比数列

```
print(np.logspace(0, 2, 3))
print(np.logspace(0, 2, 3, base = 2))
```

```
>>>
[  1.  10. 100.]
[1. 2. 4.]
```

存取元素

```
>>> a = np.arange(10)
>>> a[5] # 用整数作为下标可以获取数组中的某个元素
5

>>> a[3:5] # 用范围作为下标获取数组的一个切片
array([3, 4])

>>> a[:5] # 省略开始下标，表示从a[0]开始
array([0, 1, 2, 3, 4])

>>> a[:-1] # 下标可以使用负数，表示从数组后往前数
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> a[2:4] = 100,101 # 下标还可以用来修改元素的值
>>> a
array([ 0, 1, 100, 101, 4, 5, 6, 7, 8, 9])

>>> a[1:-1:2] # 范围中的第三个参数表示步长
array([ 1, 101, 5, 7])

>>> a[::-1] # 省略范围的开始下标和结束下标，步长为-1，整个数组头尾颠倒
array([ 9, 8, 7, 6, 5, 4, 101, 100, 1, 0])

>>> a[5:1:-2] # 步长为负数时，开始下标必须大于结束下标
array([ 5, 101])
```

存取元素

- 和Python的列表序列不同，通过下标范围获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据空间：

```
>>> b = a[3:7] # 通过下标范围产生一个新的数组b，b和a共享同一块数据空间
>>> b
array([101, 4, 5, 6])

>>> b[2] = -10 # 将b的第2个元素修改为-10
>>> b
array([101, 4, -10, 6])

>>> a # a的第5个元素也被修改为-10
array([ 0, 1, 100, 101, 4, -10, 6, 7, 8, 9])
```

存取元素

- 把一个标量值赋值给一个切片时，该值会自动传播给整个所选区域

```
>>> a = np.arange(10)
>>> a[3:5] = 100
>>> a
array([  0,   1,   2, 100, 100,   5,   6,   7,   8,   9])
```

- 如需获取副本，而不是视图，需要用copy函数

```
>>> a = np.arange(10)
>>> b = a[1:5].copy()
>>> b[0] = 999
>>> b
array([999,   2,   3,   4])
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

存取元素

- 使用整数序列存取元素
  - 使用整数序列作为下标获得的数组不和原始数组共享数据空间

```
x = np.arange(10, 1, -1)
print(x)
print(x[[3, 3, 1, 8]])
```

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

```
y = x[[3, 3, -3, 8]]
y[2] = 100
print(y)
print(x)
```

```
>>>
[  7   7 100   2]
[10  9  8  7  6  5  4  3  2]
```

存取元素

```
x[[3,5,1]] = -1, -2, -3
print(x)
```

```
>>>
[10 -3  8 -1  6 -2  4  3  2]
```

存取元素

- 使用布尔数组存取元素，总是创建数组的副本

```
x = np.arange(5, 0, -1)
# array([5, 4, 3, 2, 1])

x[np.array([True, False, True, False, False])]
# array([5, 3])

x[np.array([True, False, True, False, False])] = -1, -2
# array([-1, 4, -2, 2, 1])
```

存取元素

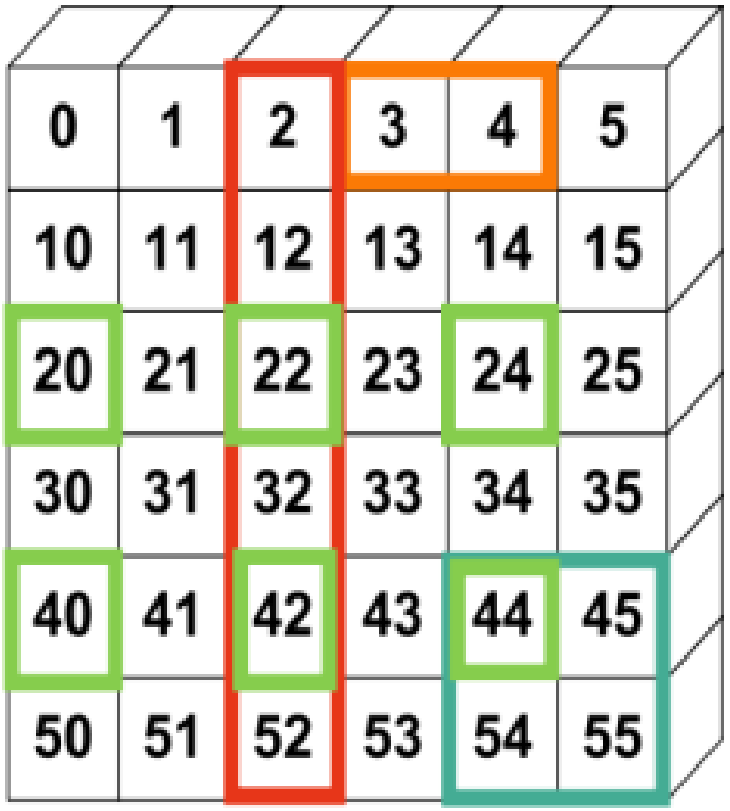
- 布尔数组可以通过使用布尔运算的函数产生

```
names = np.array(['tom','bob','bill','jack','tom'])
arr_bool = names == 'tom'
print(arr_bool)

# [ True False False False  True]
```

```
a = np.arange(5)
a[a > 2]

# array([3, 4])
```

多维数组存取元素

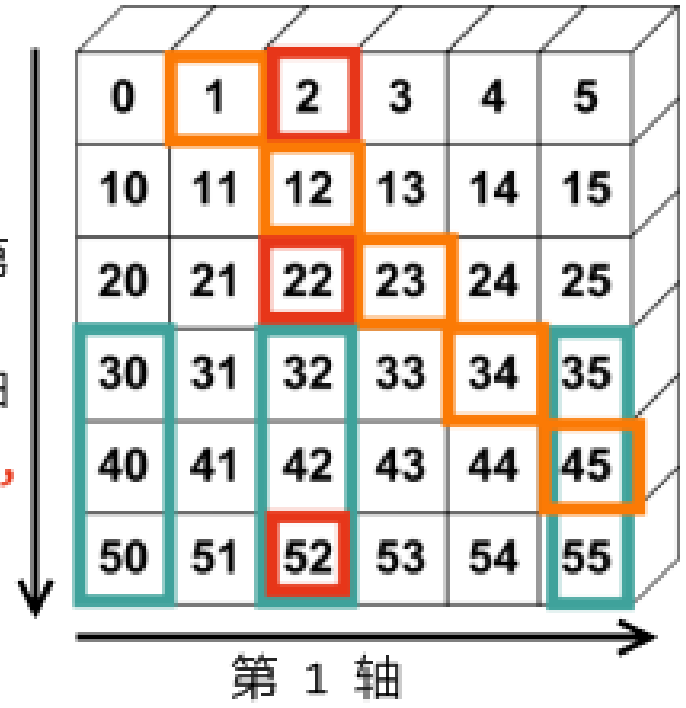
```
>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,::2]
array([[20,22,24],
       [40,42,44]])
```



多维数组存取元素

- 多维数组同样也可以使用整数序列和布尔数组进行存取

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([1,12,23,34,45])
>>> a[3:,[0,2,5]]
array([[30,32,35],
       [40,42,45],
       [50,52,55]])
>>> mask=np.array([1,0,1,0,0,1],
                  dtype=np.bool)
>>> a[mask,2]
array([2,22,52])
```



基本运算

- 元素级别运算

```
arr1 = np.array([1,3,5,4,5])
arr2 = np.array([4,6,1,3,4])
print(np.sqrt(arr1))
print(np.square(arr2))
print(np.multiply(arr1,arr2))
print(np.subtract(arr1,arr2))
```

```
>>>
[1.         1.73205081 2.23606798 2.         2.23606798]
[16 36  1  9 16]
[ 4 18  5 12 20]
[-3 -3  4  1  1]
```

基本运算

- 操作符重载

y = x1 + x2: add(x1, x2 [, y])
y = x1 - x2: subtract(x1, x2 [, y])
y = x1 * x2: multiply (x1, x2 [, y])
y = x1 / x2: divide (x1, x2 [, y]), 如果两个数组的元素为整数，那么用整数除法
y = x1 / x2: true_divide (x1, x2 [, y]), 总是返回精确的商
y = x1 // x2: floor_divide (x1, x2 [, y]), 总是对返回值取整
y = -x: negative(x [,y])
y = x1**x2: power(x1, x2 [, y])
y = x1 % x2: remainder(x1, x2 [, y]), mod(x1, x2, [, y])

基本运算

- 比较和布尔运算
  - 使用“==”、“>”等比较运算符对两个数组进行比较，将返回一个布尔数组，它的每个元素值都是两个数组对应元素的比较结果。例如：

```
np.array([1, 2, 3]) < np.array([3, 2, 1])

# array([ True, False, False])
```

```
(np.array([1, 2, 3, 4, 5]) > 2).sum()

# 3
```

基本运算

- 条件逻辑：np.where

```
arr = np.arange(10).reshape(2, 5)

np.where(arr > 6, -1, 1)

# array([[ 1,  1,  1,  1,  1],
#        [ 1,  1, -1, -1, -1]])

np.where(arr > 6, -1, arr)

# array([[ 0,  1,  2,  3,  4],
#        [ 5,  6, -1, -1, -1]])
```

基本运算

- 各轴的统计运算

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

print(arr.mean())           # 4.0
print(arr.mean(axis=0))     # [3.  4.  5.]
print(arr.mean(axis=1))     # [1.  4.  7.]

print(arr.sum())            # 36
print(arr.sum(axis=0))      # [ 9 12 15]
print(arr.sum(axis=1))      # [ 3 12 21]
```

形状操作

```
arr = np.array([[0, 1, 2, 3], [3, 4, 5, 6], [6, 7, 8, 9]])

arr.ravel()  # flatten the array
# array([0, 1, 2, 3, 3, 4, 5, 6, 6, 7, 8, 9])


arr.transpose()
# array([[0, 3, 6],
#        [1, 4, 7],
#        [2, 5, 8],
#        [3, 6, 9]])
```

形状操作

```
a1 = np.array([[0, 1], [2, 3]])
a2 = np.array([[4, 5], [6, 7]])
```

```
np.vstack((a1, a2))
# array([[0, 1],
#        [2, 3],
#        [4, 5],
#        [6, 7]])
```

```
np.hstack((a1, a2))
# array([[0, 1, 4, 5],
#        [2, 3, 6, 7]])
```

最值和排序

- 用min()和max()可以计算数组的最大值和最小值，而ptp()计算最大值和最小值之间的差。

```
a = np.array([[1,2,3,4],[5,6,7,8]])
print(np.min(a))                    # 1
print(np.max(a))                    # 8
print(np.ptp(a))                    # 7
```

- 用argmax()和argmin()可以求最大值和最小值的下标。如果不指定axis参数，就返回平坦化之后的数组下标，例如：

```
print(np.argmax(a))                 # 7
print(a.ravel()[5])                 # 6
```

最值和排序

- 可以通过unravel_index()将一维下标转换为多维数组中的下标，它的第一个参数为一维下标值，第二个参数是多维数组的形状：

```
idx = np.unravel_index(5, a.shape)
idx                                 # (1,1)
a[idx]                              # 6
```

- 当使用axis参数时，可以沿着指定的轴计算最大值的下标。

```
idx = np.argmax(a, axis = 1)
idx                                 # array([3, 3], dtype=int32)
```

最值和排序

- 数组的sort()方法用于对数组进行排序，它将改变数组的内容。而sort()函数则返回一个新数组，不改变原始数组。它们的axis参数默认值都为-1,即沿着数组的最后一个轴进行排序

```
a = np.array([[3,2,1,4],[6,8,5,7]])
np.sort(a)

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


np.sort(a, axis = 0)

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

最值和排序

- argsort()返回数组的排序下标，axis参数的默认值为-1:

```
idx = np.argsort(a)
# array([[2, 1, 0, 3],
         [2, 0, 3, 1]], dtype=int32)
```

- 用median()可以获得数组的中值

```
np.median(a, axis = 0)
# array([4.5, 5. , 3. , 5.5])
```

多项式函数

- 多项式函数是变量的整数次幂与系数的乘积之和，可以用下面的数学公式表示：

$$f(x)= a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

- 在NumPy中，多项式函数的系数可以用一维数组表示，例如$f(x) = x^3 - 2x + 1$可以用下面的数组表示，其中a[0]是最高次的系数，a[-1]是常数项，注意x^2的系数为0。

```
>>>a= np.array([1.0, 0, -2, 1])
```

多项式函数

- 可以用poly1d()将系数转换为poly1d(一元多项式)对象，此对象可以像函数一样调用，它返回多项式函数的值：

```
>>> p = np.poly1d(a)

>>>type(p)
< numpy.lib.polynomial.poly1d>

>>> p(np.linspace(0,1, 5))
array([ 1., 0.515625,0.125  , -0.078125, 0. ])
```

多项式函数

- 对poly1d对象进行加减乘除运算相当于对相应的多项式函数进行计算。例如：

```
>>> p + [-2, 1] # 和 p + np.poly1d([-2, 1])相同
poly1d([ 1., 0., -4., 2.])

>>> p*p #两个3次多项式相乘得到一个6次多项式
 poly1d([ 1.,  0., -4., 2., 4., -4.,  1.])

>>> p / [1, 1] #除法返回两个多項式，分别为商式和余式
(poly1d([ 1., -1., -1.]), poly1d([ 2.]))
```

多项式函数

- 由于多项式的除法不一定能正好整除，因此它返回除法所得到的商式和余式。上面的例子中，商式为$x^2 - x - 1$,余式为2。因此将商式和被除式相乘后，再加上余式就等于原来的P:

```
>>> p==np.poly1d([ 1., -1., -1.]) * [1,1] + 2
>>> True
```

多项式函数

- 多项式对象的deriv()和integ()方法分别计算多项式函数的微分和积分：

```
>>> p.deriv()
poly1d([ 3., 0., -2.])

>>> p.integ()
poly1d([ 0.25, 0., -1., 1. , 0.])

>>> p.integ().deriv() == p
True
```

多项式函数

- 多项式函数的根可以使用roots()函数计算：

```
>>> r = np.roots(p)
>>> r
array([-1.61803399, 1.        , 0.61803399])

>>> p(r)  #将根带入多项式计算，得到的值近似为0
array([ -4.21884749e-15,   -4.44089210e-16,   -
2.22044605e-16])
```

- 而poly()函数可以将根转换回多项式的系数：

```
>>> np.poly(r)
array([ 1. 00000000e+00, 9.99200722e-16, -2.
000000000e+00 , 1.00000000e+00])
```

多项式函数

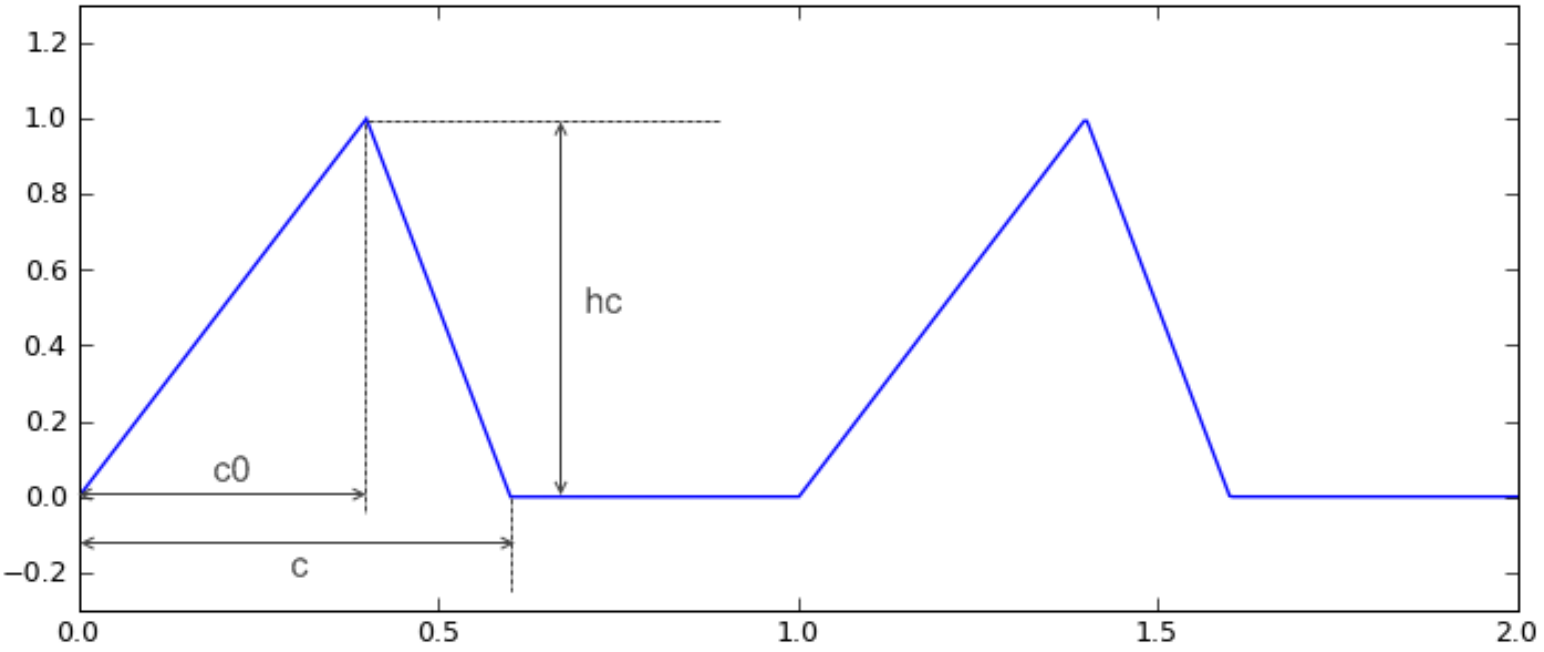
- 计算-pi/2~ pi/2区间与sin(x)函数最接近的多项式的系数：

```
import numpy as np

x = np.linspace(-np.pi/2, np.pi/2, 1000)
y = np.sin(x)
for deg in [3,5,7]:
    a = np.polyfit(x, y, deg)
    error = np.abs(np.polyval(a, x)-y)
    print("poly {}: {}".format(deg, a))
    print("max error of order {}: {}".format(deg, np.max(error)))
```

多项式函数

poly 3: [-1.45021094e-01 -1.01610165e-16 9.88749145e-01
3.34489452e-17]
max error of order 3: 0.008946993767070865
poly 5: [7.57279944e-03 2.48129858e-16 -1.65823793e-01 -
3.82318692e-16
  9.99770071e-01 3.36354605e-17]
max error of order 5: 0.0001574086141691 3
poly 7: [-1.84445514e-04 1.74230649e-16 8.30942467e-03 -
7.51623772e-16
 -1.66651593e-01 5.40261160e-16 9.99997468e-01 -5.84874370e-
17]
max error of order 7: 1.5268255887379567e-06

分段函数



分段函数

numpy.where(condition [, x, y])

```
def triangle_wave(x, c, c0, hc):
    x = x - x.astype(np.int) # 三角波的周期为1，因此只取x坐标的小数部分进行计算
    return np.where(x>=c, 0, np.where(x<c0, x/c0*hc, (c-x)/(c-c0)*hc))
```

分段函数

随着分段函数的分段数量的增加，需要嵌套更多层where(),
但这样做不便于程序的编写 和阅读

numpy.select(condlist, choicelist, default=0)

```
def triangle_wave2(x, c, c0, hc):
    x = x - x.astype(np.int)
    return np.select([x>=c, x<c0, True], [0, x/c0*hc, (c-x)/(c-c0)*hc])
```

分段函数

where()和select()的所有参数都需要在调用它们之前完成计算，因此NumPy会计算下 面4个数组：

x>=c, x<c0, x/c0*hc, (c-x)/(c-c0)*hc

numpy.piecewise(x, condlist, funclist, *args, **kw)

```
def triangle_wave3(x, c, c0, hc):
    x = x - x.astype(np.int)
    return np.piecewise(x, [x >= c, x < c0], [0, lambda x: x / c0 * hc, lambda x: (c - x) / (c - c0) * hc])
```

统计函数

- unique():返回其参数数组中所有不同的值，并按从小到大的顺序排列。它有两个可选参数：
  - return_index : True表示同时返回原始数组中的下标。
  - return_inverse: True表示返回重建原始数组用的下标数组。
- 例子：用randint()创建含有10个元素、值在0到9范围之内的随机整数数组：

```
>>> a =np.random.randint(0,10,10)
>>> a
array([1, 1, 9, 5, 2, 6, 7, 6, 2, 9])
```

统计函数

- 通过unique(a)可以找到数组a中所有的整数，并按照顺序排列：

```
>>> np.unique(a)
array([l, 2, 5, 6, 7, 9])
```

- 如果参数return_index为True,就返回两个数组,第二个数组是第一个数组在原始数组中的下标：

```
>>> x, idx = np.unique(a, return_index=True)
>>> x
array([1, 2, 5, 6, 7, 9])
>>> idx
array([0, 4, 3, 5, 6, 2])
```

统计函数

- 数组idx保存的是数组x中每个元素在数组a中的下标:

```
>>> a[idx]
array([1, 2, 5, 6, 7, 9])
```

- 如果参数return_inverse为True,那么返回的第二个数组是原始数组a中每个元素在数组x 中的下标：

```
>>> x, ridx = np.unique(a, return_inverse=True)
>>> ridx
array([0, 0, 5, 2, 1, 3, 4, 3, 1, 5])
>>>all(x[ridx]==a) #原始数组a和x[ridx]完全相同
True
```

统计函数

- bincount():对整数数组中各个元素出现的次数进行统计，它要求数组中所有元素都是非负的。其返回数组中第i个元素的值表示整数i在参数数组中出现的次数。

```
>>> np.bincount(a)
array([0, 2, 2, 0, 0, 1, 2, 1, 0, 2])
```

统计函数

- 通过weights参数可以指定每个数对应的权值。当指定weights参数时，bincount(x, weights=w)返回数组x中每个整数所对应的w中的权值之和。

```
>>> x =np.array([0 , 1, 2, 2, 1, 1, 0])
>>> w = np.array([0.1, 0.3, 0.2,0.4,0.5,0.8,1.2])
>>> np.bincount(x, w)
array ([ 1.3,1.6,0.6])
```

- 如果要求平均值, 可以用求和结果与次数相除：

```
>>> np.bincount(x,w)/np.bincount(x)
array([ 0.65    , 0.53333333, 0.3     ])
```

统计函数

- histogram() :对一维数组进行直方图统计，其参数列表如下：

```
Histogram(a,bins=10,range=None,normed=False,weights=
None)
```

- histogram()返回两个一维数组--hist和bin_edges,第一个数组是每个区间的统计结果， 第二个数组长度为len(hist)+1,每两个相邻的数值构成一个统计区间。

```
>>> a = np.random.rand(100)
>>> np.histogram(a,bins=5,range=(0,1))
(array([20,26,20,16,18]), array([ 0. , 0.2, 0.4, 0.6, 0.8, 1.])
```

统计函数

- 如果需要统计的区间长度不等，可以将表示区间分隔位置的数组传递给bins参数，例如:

```
>>> np.histogram(a,bins=[0, 0.4, 0.8, 1.0], range=(0,1))
(array([46, 36, 18]), array([ 0. , 0.4, 0.8, 1.]))
```