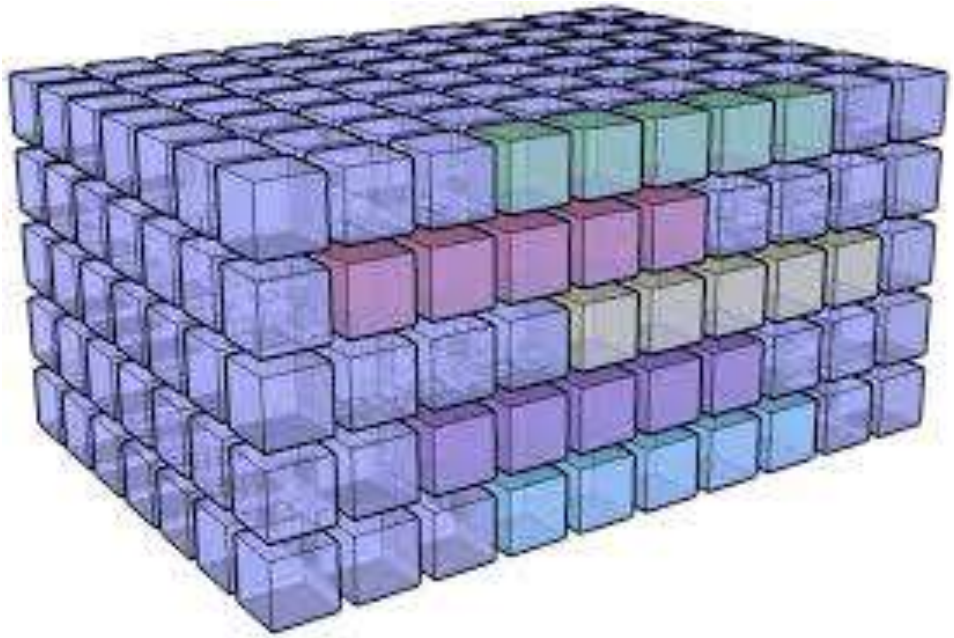
Python数据分析-NumPy

NumPy 是什么？

NumPy是Python中科学计算的核心库，提供了高性能的多维数组对象，以及用于处理这些数组的工具。主要是方便熟悉Matlab的用户，通过Numpy在Python中对数组进行操作。

主要功能有：

· 创建n维数组（矩阵）
· 对数组进行函数运算
· 数值积分
· 线性代数运算
· 傅里叶变换
· 随机数产生
……



NumPy 是什么？

标准的**Python**中用**list**（列表）保存值，可以当做数组使用，但因为列表中的元素可以是任何对象，所以浪费了CPU运算时间和内存。

NumPy诞生为了弥补这些缺陷。它提供了两种基本的对象：
ndarray：全称（n-dimensional array object）是储存单一数据类型的多维数组。
ufunc：全称（universal function object）它是一种能够对数组进行处理的函数。

NumPy的官方文档：
https://docs.scipy.org/doc/numpy/reference/

NumPy 是什么？

numpy库处理的最基础数据类型是由同种元素构成的多维数组（ndarray），简称“数组”。

数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从0开始。ndarray类型的维度(dimensions)叫做轴(axes)，轴的个数叫做秩(rank)。一维数组的秩为1，二维数组的秩为2，二维数组相当于由两个一维数组构成。

ndarray 对象

- ndarray的创建
- ndarray的属性
- ndarray的切片
- 多维数组
- 结构数组

ndarray的创建

由于numpy库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用numpy库：

```
>>>import numpy as np
```

其中，as保留字与import一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单说，在程序的后续部分中，np代替numpy。

ndarray的创建

NumPy中的核心对象是ndarray。
ndarray可以看成数组，类似于Matlab中的矩阵。
NumPy里面所有的函数都是围绕ndarray展开的。

Numpy库常用的创建数组函数

函数	描述
np.array([x,y,z], dtype=int)	从 Python 列表和元组创造数组
np.arange(x,y,i)	创建一个由 x 到 y，以 i 为步长的数组
np.linspace(x,y,n)	创建一个由 x 到 y，等分成 n 个元素的数组
np.indices((m,n))	创建一个 m 行 n 列的矩阵
np.random.rand(m,n)	创建一个 m 行 n 列的随机数组
np.ones((m,n),dtype)	创建一个 m 行 n 列全 1 的数组，dtype 是数据类型
np.empty((m,n),dtype)	创建一个 m 行 n 列全 0 的数组，dtype 是数据类型

ndarray的创建

创建一个简单的数组后，可以查看ndarray类型的一些基本属性

属性	描述
ndarray.ndim	数组轴的个数，也被称作秩
ndarray.shape	数组在每个维度上大小的整数元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型，dtype 类型可以用于创建数组中
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器

ndarray的创建

np.array([x, y, z], dtype)

创建一个数组ndarray

```python
import numpy as np

a = [1,2,3]
b = np.array(a)

print(b)
print(type(b))
```

```
[1 2 3]
<class 'numpy.ndarray'>

进程已结束,退出代码0
```

```python
import numpy as np

a = (1,2,3)
b = np.array(a)

print(b)
print(type(b))
```

```
[1 2 3]
<class 'numpy.ndarray'>

进程已结束,退出代码0
```

分别将Python的列表和元组转换为数组类型

ndarray的创建

np.array([x, y, z], dtype)

dtype的用法

```
1  import numpy as np
2
3  a = (1, 2, 3)
4  b = np.array(a, dtype=float)
5
6  print(b)
7  print(b.dtype)
8  print(type(b))
```

```
[1. 2. 3.]
float64
<class 'numpy.ndarray'>

进程已结束,退出代码0
```

默认数据类型是int，可以通过dtype改变数据类型

ndarray的创建

ndarray对维数没有限制。

数组的轴即数组的维度，分别为第0轴，第1轴，第2轴。

如创建c为三行四列的一个数组，

c的第0轴长度为3，第1轴长度为4。

```
> a = np.array([1, 2, 3, 4])
> b = np.array((5, 6, 7, 8))
> c = np.array([[1, 2, 3, 4],[5, 6, 7, 8], [9, 10, 11, 12]])
```

```
a                b                c
------------     ------------     ------------------
[1, 2, 3, 4]     [5, 6, 7, 8]     [[ 1,  2,  3,  4],
                                   [ 5,  6,  7,  8],
                                   [ 9, 10, 11, 12]]
```

ndarray的创建

NumPy提供了**专门用于生成ndarray的函数**，提高创建ndarray的速度。

```
> a = np.arange(0, 1, 0.1)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
> b = np.linspace(0, 1, 10)
array([ 0. ,  0.11111111,  0.22222222,  0.33333333,
0.44444444,0.55555556,  0.66666667,  0.77777778,
0.88888889,  1. ])
> c = np.linspace(0, 1, 10, endpoint=False)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
> d = np.logspace(0, 2, 5)
array([ 1. , 3.16227766, 10., 31.6227766 , 100. ])
```

ndarray的创建

np.empty((2,3), int)	创建2*3的整形型空矩阵，只分配内存
np.zeros(4, int)	创建长度为4，值为全部为0的矩阵
np.full(4, np.pi)	创建长度为4，值为全部为pi的矩阵

还可以**自定义函数产生ndarray。**

```
> def func(i):
      return i % 4 + 1
> np.fromfunction(func, (10,)) #括号内参数数量与函数形参数量一致，逗号不可省略
array([ 1.,  2.,  3.,  4.,  1.,  2.,  3.,  4.,  1.,  2.])
```

fromfunction第一个参数接收计算函数，第二个参数接收数组的形状。

ndarray的创建

ndarray类的形态操作方法

方法	描述
ndarray.reshape(n,m)	不改变数组 ndarray，返回一个维度为(n,m)的数组
ndarray.resize(new_shape)	与 reshape()作用相同，直接修改数组 ndarray
ndarray.swapaxes(ax1, ax2)	将数组 n 个维度中任意两个维度进行调换
ndarray. flatten()	对数组进行降维，返回一个折叠后的一维数组
ndarray.ravel()	作用同 np.flatten()，但是返回数组的一个视图

ndarray的属性

ndarray的元素具有相同的元素类型。常用的有int（整型），float（浮点型），complex（复数型）。

```
> a = np.array([1, 2, 3, 4], float)
array([ 1.,  2.,  3.,  4.])

> a.dtype
dtype('float64')
```

ndarray的shape属性用来获得它的形状，也可以自己指定。

```
> c = np.array([[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]])
> c.shape
(3, 4)

> a = np.array([1, 2, 3, 4])
> d = a.reshape(2,2)
array([[1, 2],
       [3, 4]])
```

ndarray的属性

ndarray的切片和list是一样的。

方法	描述
x[i]	索引第 i 个元素
x[-i]	从后向前索引第 i 个元素
x[n:m]	默认步长为 1，从前往后索引，不包含 m
x[-m:-n]	默认步长为 1，从后往前索引，结束位置为 n
x[n,m,i]	指定 i 步长的由 n 到 m 的索引

ndarray的切片

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

a[5]    a[3:5]       a[:5]              a[:-1]
----    ------    --------------    --------------------------
5         [3, 4]    [0, 1, 2, 3, 4]        [0, 1, 2, 3, 4, 5, 6, 7, 8]

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

可以通过切片对ndarray中的元素进行**更改**。

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

ndarray的切片

ndarray通过切片产生一个新的数组b，**b和a共享同一块数据存储空间。**

```
> b = a[3:7]
> b[2] = -10
 b                          a
------------------  ----------------------------------------
[101,   4, -10,   6]    [  0,   1, 100, 101,   4, -10,   6,   7,   8,   9]
```

如果想改变这种情况，我们可以**用列表对数组元素切片。**

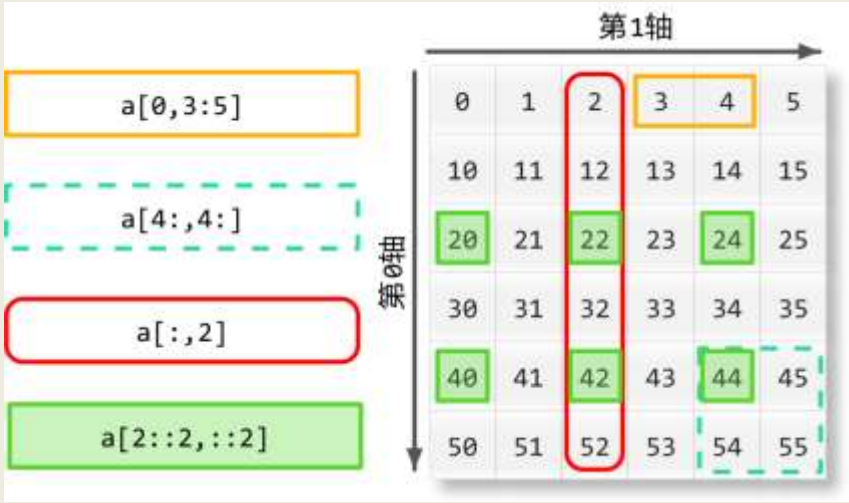
```
> a=array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
> b = a[[3, 3, -3, 8]]
> b
array([3, 3, 7, 8])

> b[2] = 100
b                           a
------------------  ----------------------------------------
[3,   3, 100,   8]      [  0,   1,  2,  3,   4,  5,   6,   7,   8,   9]
```

多维数组

NumPy的多维数组和一维数组类似。多维数组有多个轴。
我们前面已经提到从内到外分别是第0轴，第1轴...

```
> a = np.arange(0, 60, 10).reshape(-1, 1) + np.arange(0, 6)
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
```




```
a[0, 3:5]      a[4:, 4:]      a[2::2, ::2]
---------      ----------     --------------
[3, 4]         [[44, 45],     [[20, 22, 24],
                [54, 55]]      [40, 42, 44]]
```

#上面方法对于数组的切片都是共享原数组的储存空间的。

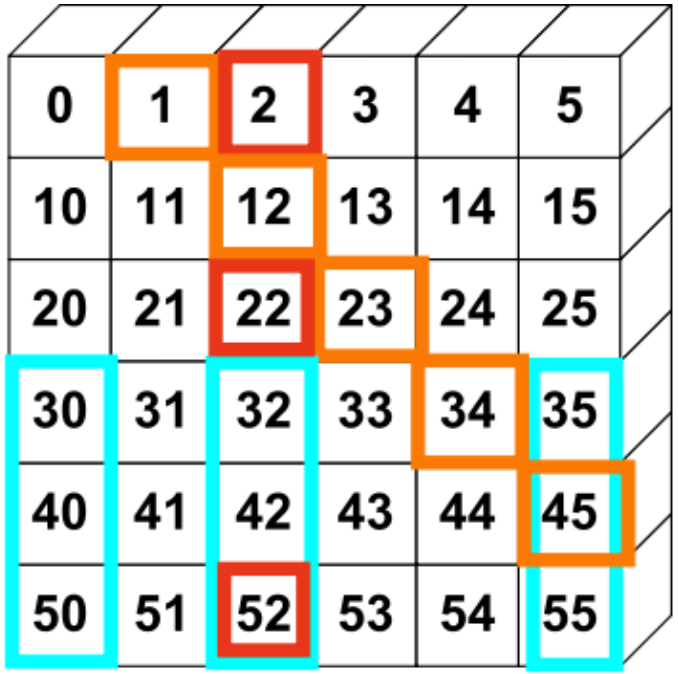
多维数组

如果我们想**创立原数组的副本**，我们可以用**整数元组，列表，整数数组，布尔数组**进行切片。

```
>>> 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 = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



结构数组

C语言中可以通过struct关键字定义结构类型。NumPy中也有类似的**结构数组。**

```
> persontype = np.dtype({'names':['name', 'age', 'weight'],
   'formats':['S30','i', 'f']})

> a = np.array([("Zhang", 32, 75.5), ("Wang", 24, 65.2)],
     dtype=persontype)
```

	name	age	weight
0	zhang	32	75.5
1	wang	24	65.2

我们就创建了一个结构数组，并且可以通过索引得到每一行。

```
> print a[0]
('Zhang', 32, 75.5)
```

ufunc函数

- ufunc简介
- 四则运算
- 比较运算和布尔运算
- 自定义ufunc函数
- 广播(broadcasting)

ufunc简介

ufunc是**universal function**的简称，它是一种能对**数组每个元素进行运算**的函数。**NumPy**的许多**ufunc**函数都是用C语言实现的，因此它们的运算速度非常快。

```
> x = np.linspace(0, 2*np.pi, 10)
> y = np.sin(x)
> y
array([  0.00000000e+00,   6.42787610e-01,
9.84807753e-01,
. . . . . ,
       -2.44929360e-16])
```

值得注意的是，对于**同等长度**的ndarray，np.sin()比math.sin()快

但是对于**单个数值**，math.sin()的速度则更快。

四则运算

NumPy提供了许多**ufunc**函数，它们和相应的运算符运算结果相同。

```
> a = np.arange(0, 4)
> b = np.arange(1, 5)
> np.add(a, b)
array([1, 3, 5, 7])

> a+b
array([1, 3, 5, 7])

> np.subtract(a, b) # 减法
> np.multiply(a, b) # 乘法
> np.divide(a, b) # 如果两个数字都为整数，则为整数除法
> np.power(a, b) # 乘方
```

比较运算和布尔运算

使用==，>对两个数组进行比较，会返回一个**布尔数组**，**每一个元素都是对应元素的比较结果。**

```
> np.array([1, 2, 3]) < np.array([3, 2, 1])
array([ True, False, False], dtype=bool)
```

布尔运算在NumPy中也有对应的ufunc函数。

表达式	**ufunc函数**
y=x1==x2	equal(x1,x2[,y])
y=x1!=x2	not_equal(x1,x2[,y])
y=x1<x2	less(x1,x2[,y])
y=x1<=x2	not_equak(x1,x2[,y])
y=x1>x2	greater(x1,x2[,y])
y=x1>=x2	gerater_equal(x1,x2[,y])

自定义ufunc函数

NumPy提供的标准ufunc函数可以组合出复合的表达式，但是有些情况下，自己编写的则更为方便。我们可以**把自己编写的函数用frompyfunc()转化成ufunc函数。**

```
#不使用frompyfunc()
> def num_judge(x, a): #对于一个数字如果是3或5的倍数就返回0，否则返回a。
      if x%3 == 0:
          r = 0
      elif x%5 == 0:
          r = 0
      else:
          r = a
      return r
> x = np.linspace(0, 10, 11)
> y = np.array([num_judge(t, 2) for t in x])#列表生成式
array([0, 2, 2, 0, 2, 0, 0, 2, 2, 0, 0])
```

自定义ufunc函数

使用frompyfunc()进行转化，调用格式如下：

frompyfunc(func, nin, nout)

func：计算函数
nin：func()输入参数的个数
nout：func()输出参数的个数

```
> numb_judge = np.frompyfunc(num_judge, 2, 1)

> y = numb_judge(x,2)
array([0, 2, 2, 0, 2, 0, 0, 2, 2, 0, 0], dtype=object)
```

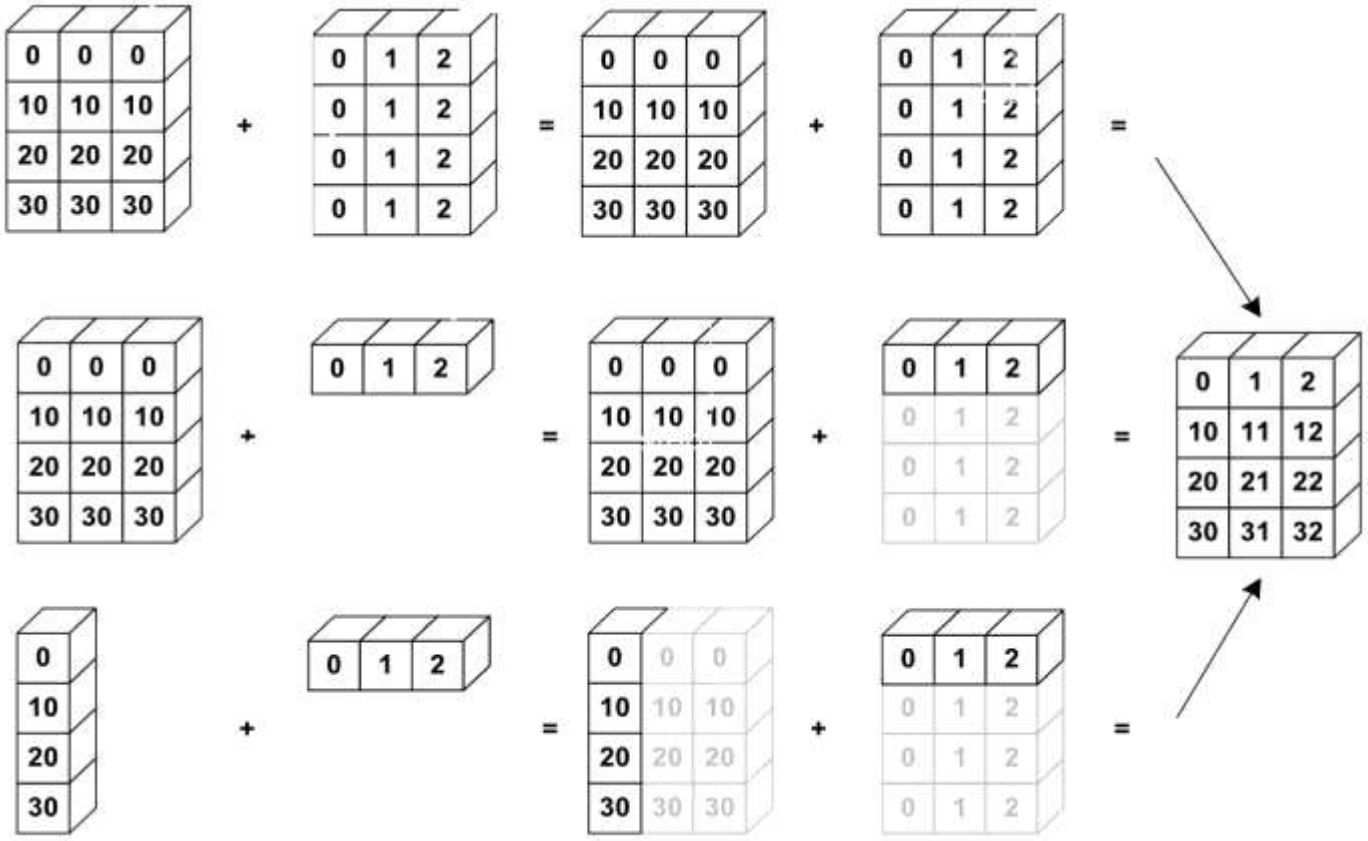
因为最后输出的元素类型是object，所以我们还需要把它转换成整型。

```
y.astype(np.int)
```

广播(broadcasting)

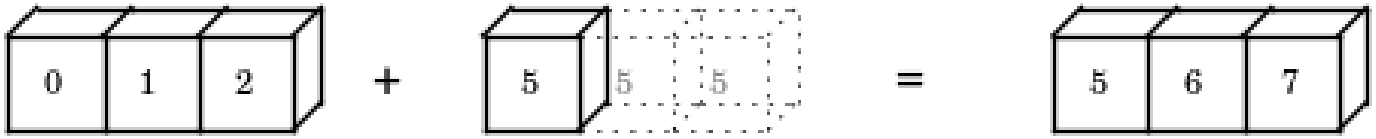
使用ufunc对两个数组进行运算时，ufunc函数会对两个数组的对应元素进行运算。如果数组的形状不相同，就会进行**下广播**处理。

简而言之，就是向**两个数组每一维度上的最大值靠齐**。

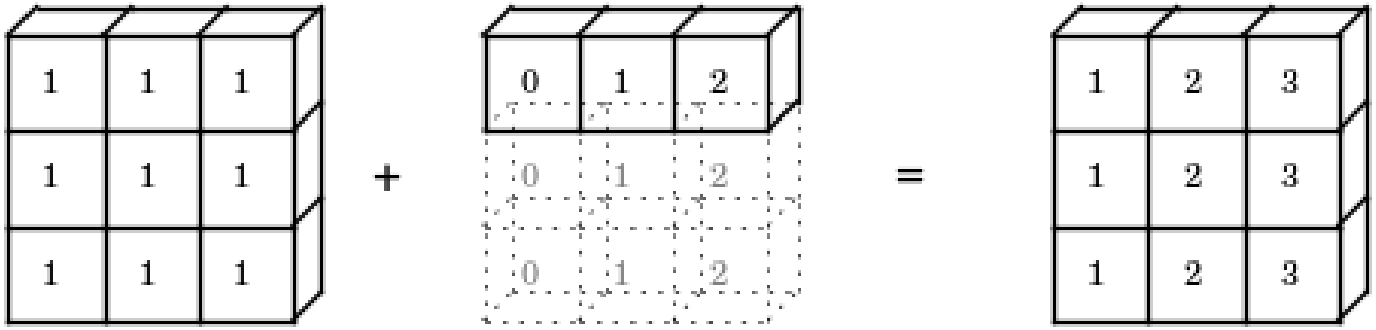


广播(broadcasting)

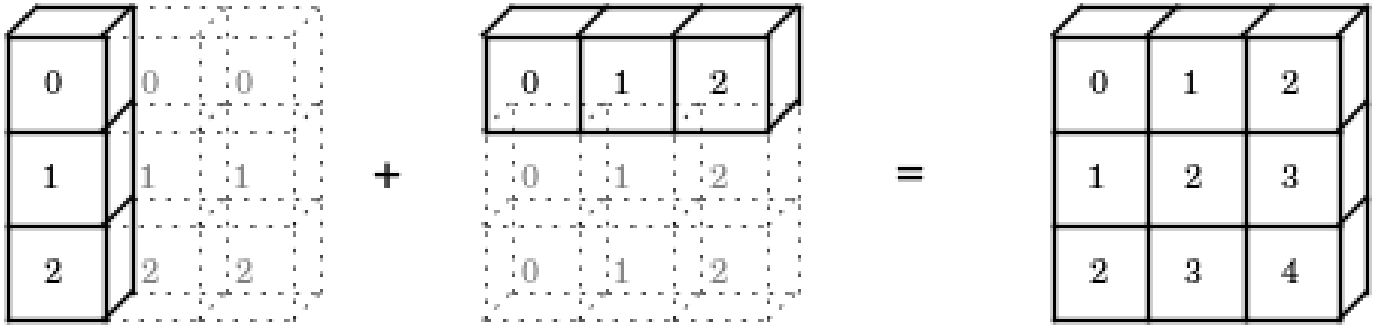
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



广播(broadcasting)

我们看一下具体的例子：

```
> a = np.arange(0, 60, 10).reshape(-1, 1)
> b = np.arange(0, 5)
> c = a+b
```

```
c                                   c.shape
----------------------              -------
[[ 0,  1,  2,  3,  4],                (6, 5)
 [10, 11, 12, 13, 14],
 [20, 21, 22, 23, 24],
 [30, 31, 32, 33, 34],
 [40, 41, 42, 43, 44],
 [50, 51, 52, 53, 54]]
```

广播(broadcasting)

ogrid用来生成广播运算所用的数组。

```
> x, y = np.ogrid[:5, :5]
x             y
-----   -----------------
[[0],      [[0, 1, 2, 3, 4]]
 [1],
 [2],
 [3],
 [4]]
```

广播(broadcasting)

下面操作和a.reshape(1,-1), a.reshape(-1,1)相同。

```
> a = np.arange(4)  # array([0, 1, 2, 3]) (4,)
a[None, :]                    a[:, None]
--------------                  ----------
[[0, 1, 2, 3]] #(1,4)          [[0],[1],[2],[3]] #(4,1)
```

None的作用是在相应的位置上增加了一个维度。

```
假设 x.shape == (a,b)
(a, b) >>> [None, :, :] >>> (1, a, b)
(a, b) >>> [:, None, :] >>> (a, 1, b)
(a, b) >>> [:, :, None] >>> (a, b, 1)
```

NumPy的函数库

- 随机数
- 求和，平均值，方差
- 大小与排序
- 统计函数
- 操作多维数组
- 多项式函数

随机数

除了前面介绍的ufunc()函数之外，NumPy还提供了大量对于数组运算的函数。它们能够简化逻辑，提高运算速度。

我们首先看随机数。NumPy产生随机数的模块在**random里面，其中有大量的分布。**

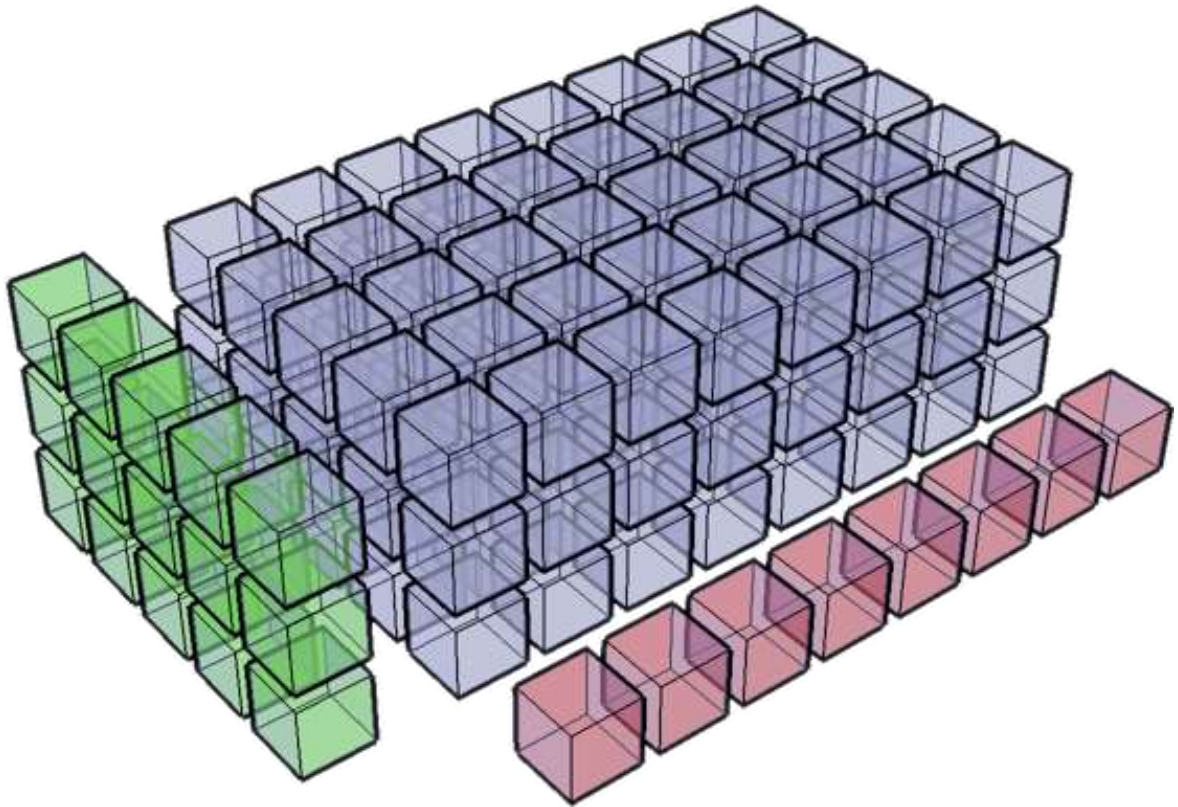
```
> from numpy import random as nr
> np.set_printoptions(precision=2) #显示小数点后两位数

> r1 = nr.rand(4, 3)
[[ 0.87,  0.42,  0.34],
[ 0.25,  0.87,  0.42],
[ 0.49,  0.18,  0.44],
[ 0.53,  0.23,  0.81]]

> r2 = nr.poisson(2.0, (4, 3))
[[3, 1, 5],
[2, 2, 3],
[2, 4, 4],
[2, 2, 3]]
```

随机数

rand	0到1之间的随机数	normal	正态分布的随机数
randint	制定范围内的随机整数	uniform	均匀分布
randn	标准正态的随机数	poisson	泊松分布
choice	随机抽取样本	shuffle	随机打乱顺序



求和，平均值，方差

NumPy在**均值等方面常用的函数**如下：

函数名	功能
sum	求和
average	加权平均数
var	方差
mean	期望
std	标准差
product	连乘积

```
> np.random.seed(42)
> a = np.random.randint(0,10,size=(4,5))

> np.sum(a)
96
```

求和，平均值，方差

```
a                    np.sum(a, axis=1)   np.sum(a, axis=0)
-------------        -----------------    -------------------
[[6, 3, 7, 4, 6],    [26, 28, 24, 18]     [22, 13, 27, 18, 16]
 [9, 2, 6, 7, 4],
 [3, 7, 7, 2, 5],
 [4, 1, 7, 5, 1]]
```

keepdims可以保持原来数组的**维数。**

```
np.sum(a,1,keepdims=True)   np.sum(a,0,keepdims=True)
-------------------------    -------------------------
[[26],                       [[22, 13, 27, 18, 16]]
 [28],
 [24],
 [18]]
```

大小与排序

NumPy在**排序等方面常用的函数**如下：

函数名	功能	函数名	功能
min	最小值	max	最大值
ptp	极差	argmin	最小值的下标
mininum	二元最小值	maxinum	二元最大值
sort	数组排序	argsort	数组排序下标
percentile	分位数	median	中位数

min,max都有axis,out,keepdims等参数，我们来看其他函数。

```
> a = np.array([1, 3, 5, 7])
> b = np.array([2, 4, 6])
> np.maximum(a[None, :], b[:, None])#maxinum返回两组矩阵广播计算后的结果
array([[2, 3, 5, 7],
       [4, 4, 5, 7],
       [6, 6, 6, 7]])
```

大小与排序

sort()对数组进行排序会改变数组的内容，返回一个新的数组。**axis**的默认值都为-1，即按最终轴进行排序。axis=0对每列上的值进行排序。

```
np.sort(a)                np.sort(a, axis=0)          a
----------------          ------------------          --------------
[[3, 4, 6, 6, 7],         [[3, 1, 6, 2, 1],           [[6, 3, 7, 4, 6],
 [2, 4, 6, 7, 9],          [4, 2, 7, 4, 4],            [9, 2, 6, 7, 4],
 [2, 3, 5, 7, 7],          [6, 3, 7, 5, 5],            [3, 7, 7, 2, 5],
 [1, 1, 4, 5, 7]]          [9, 7, 7, 7, 6]]            [4, 1, 7, 5, 1]]
```

percentile计算处于p%上的值。

```
> r = np.abs(np.random.randn(100000)) %标准正态分布
> np.percentile(r, [68.3, 95.4, 99.7])
array([ 1.00029686,  1.99473003,  2.9614485 ])
```

统计函数

NumPy中常用的统计函数有：**unique(), bicount(), histogram()**。我们来一个个介绍。首先看**unique():**

```
> np.random.seed(42)
> a = np.random.randint(0, 8, 10)
> np.unique(a)
```

a	np.unique(a)
[6, 3, 4, 6, 2, 7, 4, 4, 6, 1]	[1, 2, 3, 4, 6, 7]

unique有两个参数,**return_index=True**同时返回原始数组中的下标。

```
> x, index = np.unique(a, return_index=True)]
```

x	index
[1, 2, 3, 4, 6, 7]	[9, 4, 1, 2, 0, 5]

统计函数

return_inverse=True表示原始数据在新数组的下标

```
>x, rindex = np.unique(a, return_inverse=True)
x
------------------
[1, 2, 3, 4, 6, 7]

 a
------------------------------
[6, 3, 4, 6, 2, 7, 4, 4, 6, 1]

rindex
------------------------------
[4, 2, 3, 4, 1, 5, 3, 3, 4, 0]
```

统计函数

bincount()对**非负整数数组**中的各个元素出现的次数进行统计，返回数组中的第i个元素是整数i出现的次数。

```
> a = np.array([6, 3, 4, 6, 2, 7, 4, 4, 6, 1])
> np.bincount(a)
array([0, 1, 1, 1, 3, 0, 3, 1])

> 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])
```

统计函数

histogram()对一维数组进行直方图统计，其参数为：
histogram(a, bins=10, range=None, weights=None)
函数返回两个一维数组，分别是每个区间的统计结果和区间的边界值。

```
> a = np.random.rand(100)
> np.histogram(a, bins=5, range=(0, 1))
(array([28, 18, 17, 19, 18]),
 array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ]))

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

操作多维数组

多维数组可以进行**连接，分段**等多种操作。我们先来看**vstack(),hstack(),column_stack()**函数。

```
> a = np.arange(3)
> b = np.arange(10, 13)

> v = np.vstack((a, b)) # 按第1轴（列）连接数组
> h = np.hstack((a, b)) # 按第0轴（行）连接数组
> c = np.column_stack((a, b)) # 按列连接多个一维数组
```

```
 v                  h                          c
--------------  -------------------------  ----------
[[ 0,  1,  2],     [ 0,  1,  2, 10, 11, 12]     [[ 0, 10],
 [10, 11, 12]]                                   [ 1, 11],
                                                 [ 2, 12]]
```

操作多维数组

split()函数进行分段。

```
> a = np.array([6, 3, 7, 4, 6, 9, 2, 6, 7, 4, 3, 7])
> b = np.array([ 1,  3,  6,  9, 10]
> np.split(a, b) # 按元素位置进行分段
[array([6]),
array([3, 7]),
array([4, 6, 9]),
array([2, 6, 7]),
array([4]),
array([3, 7])]

> np.split(a, 2) # 按数组个数进行分段
[array([6, 3, 7, 4, 6, 9]),
array([2, 6, 7, 4, 3, 7])]
```

多项式函数

多项式函数是整数的次幂与系数的乘积，如：
f(x)= an(x^n)+ an-1(x^(n-1))+...+ a1(x)+ a0
NumPy中多项式函数可以用**一维数组**表示。a[0]为最高次，a[-1]为常数项。

```
> a = np.array([1.0, 0, -2, 1])
> p = np.poly1d(a)
> print type(p)
<class 'numpy.lib.polynomial.poly1d'>

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

多项式函数

多项式函数可以进行**四则运算**，其中**运算的列表自动化成多项式函数。**

```
> p + [-2, 1]
poly1d([ 1.,  0., -4.,  2.])
> p * p
poly1d([ 1.,  0., -4.,  2.,  4., -4.,  1.])
> p / [1, 1] # 分别为商和余
(poly1d([ 1., -1., -1.]), poly1d([ 2.]))
```

多项式也可以进行**积分和求导。**

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

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

多项式函数

Roots可以求多项式的根。

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

polyfit()可以对数据进行多项式拟合。x, y为数据点，deg为多项式最高阶数。

```
> a = np.polyfit(x , y, deg)
```

poly()返回多项式系数构成的数组。

```
> a = np.poly(x )
```

numpy.linalg模块

Numpy.linalg模块包含线性代数的函数。使用这个模块，可以计算逆矩阵、求特征值、解线性方程组以及求解行列式等.

计算逆矩阵

在线性代数中，矩阵A与其逆矩阵A^{-1}相乘后会得到一个单位矩阵I。该定义可以写为$A * A^{-1}=I$。

numpy.linalg模块中的inv函数可以计算逆矩阵。按如下步骤来对矩阵求逆。

numpy.linalg模块

使用numpy.linalg模块中的inv函数计算了逆矩阵，并检查了原矩阵与求得的逆矩阵相乘的结果确为单位矩阵。

```
import numpy as np
A = np.mat("0 1 2;1 0 3;4 -3 8")  #使用mat函数创建示例矩阵
# A=[[0 1 2]
        [1 0 3]
        [4 -3 8]]

inverse = np.linalg.inv(A)  #使用inv函数计算逆矩阵
# inverse =[[-4.5 7. -1.5]
                [-2. 4. -1.]
                [1.5 -2. 0.5]]

np.dot(A, inverse)= [[1. 0. 0.]
                              [0. 1. 0.]
                              [0. 0. 1.]]
```

numpy.linalg模块

求解线性方程组

numpy.linalg中的函数solve可以求解形如 Ax = b 的线性方程组，其中 A 为矩阵，b 为一维或二维的数组，x 是未知变量。

```
import numpy as np

A = np.mat("1 -2 1;0 2 -8;-4 5 9") #创建矩阵
# A=[[1 -2 1]
#    [0 2 -8]
#    [-4 5 9]]

b = np.array([0, 8, -9]) #创建数组    # b=[0 8 -9]

x = np.linalg.solve(A, b)  #使用solve函数求解线性方程     # x=[29. 16. 3.]

np.dot(A , x) #使用dot函数检查解是否正确  #  [[0. 8. -9.]]
```

numpy.linalg模块

特征值和特征向量

特征值（eigenvalue）即方程 Ax = ax 的根，是一个标量。其中，A 是一个二维矩阵（n×n），x 是一个非零n维列向量。特征向量（eigenvector）是关于特征值的向量。在numpy.linalg模块中，eigvals函数可以计算矩阵的特征值，而eig函数可以返回一个包含特征值和对应的特征向量的元组。

numpy.linalg模块

```
import numpy as np

A = np.mat("3 -2;1 0")
# A = [[3 -2]
       [1 0]]

np.linalg.eigvals(A)  #调用eigvals函数求解特征值  # [2. 1.]

eigenvalues, eigenvectors = np.linalg.eig(A)
#调用eig求解特征值和特征向量
#eigenvalues = [2. 1.]
#eigenvectors = [[0.89442719, 0.70710678]
                 [0.4472136 , 0.70710678]]
```

numpy.linalg模块

```
# 调用dot函数验证解是否正确
for i in range(len(eigenvalues)):
   print('Left:',np.dot(A,eigenvectors[:,i]))
   print('Right:',eigenvalues[i]*eigenvectors[:,i])

Left:  [[1.78885438]
 [0.89442719]]
Right: [[1.78885438]
 [0.89442719]]

Left: [[0.70710678]
 [0.70710678]]
Right: [[0.70710678]
 [0.70710678]]
```

numpy.linalg模块

在numpy.linalg模块中的svd函数可以对矩阵进行奇异值分解。该函数返回3个矩阵——U、Sigma和V，其中U和V是正交矩阵，Sigma包含输入矩阵的奇异值。

```
import numpy as np
A = np.mat("4 11 14;8 7 -2")   # A=[[4 11 14]
                                    [8 7 -2]]
U, Sigma, V=np.linalg.svd(A) #使用svd函数分解矩阵

U=[[-0.9486833 -0.31622777]
    [-0.31622777 0.9486833 ]]

Sigma=[18.97366596  9.48683298] #为节省空间，只输出奇异值向量

V = [[-0.33333333, -0.66666667, -0.66666667],
      [ 0.66666667,  0.33333333, -0.66666667],
      [-0.66666667,  0.66666667, -0.33333333]]
```

numpy.linalg模块

广义逆矩阵

摩尔·彭罗斯广义逆矩阵（Moore-Penrose pseudoinverse）可以使用numpy.linalg模块中的pinv函数进行求解。计算广义逆矩阵需要用到奇异值分解。inv函数只接受方阵作为输入矩阵，而pinv函数则没有这个限制。

```
import numpy as np
A = np.mat("4 11 14;8 7 -2")
A=[[4 11 14]
     [8 7 -2]]
pseudoinv = np.linalg.pinv(A)
pseudoinv = [[-0.00555556 0.07222222]
                  [ 0.02222222 0.04444444]
                  [ 0.05555556 -0.05555556]]
```

numpy.linalg模块

行列式

numpy.linalg模块中的det函数可以计算矩阵的行列式。

```
import numpy as np

A = np.mat("3 4;5 6")
A = [[3 4]
     [5 6]]

np.linalg.det(A) = -2.0
```

End