

# 第 10 章 NumPy 库

## 本章主要内容：

(1) NumPy数组对象ndarray

(2) Numpy通用函数及数组之间的运算

(3) NumPy 矩阵运算

(4) NumPy 读写文件

## 10.1 NumPy 数组对象 ndarray

---

NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

**Numpy** 提供了两种基本的对象，**ndarray**( N-dimensional array object) 和**ufunc**( universal function object)。ndarray为多维数组，ufunc为对数组进行处理的函数。

使用时，需要引用：`import numpy as np`

## 10.1.1 创建数组

(1) 数组属性：`ndarray`（数组）是存储**单一数据类型**的多维数组，如表所示。

序号	属性	说明
1	<code>ndim</code>	返回 <code>int</code> ，表示数组的维数
2	<code>shape</code>	返回 <code>tuple</code> ，表示数组的尺寸，对于 <code>n</code> 行 <code>m</code> 列的矩阵，形状为 <code>(n, m)</code> ，例如： 一维数组返回 <code>(n,)</code> ，二维数组返回 <code>(n, m)</code>
3	<code>size</code>	返回 <code>int</code> ，数组的元素总数，等于数组形状的乘积，对于 <code>n</code> 行 <code>m</code> 列的矩阵，为 <code>n*m</code>
4	<code>dtype</code>	返回 <code>data-type</code> ，描述数组中元素的类型
5	<code>itemsize</code>	返回 <code>int</code> ，表示数组的每个元素的大小（以字节为单位）

## 10.1.1 创建数组

(2) 数组创建：创建一维或多维数组，语法如下。

```
numpy.array(object, dtype=None, copy=True)
```

序号	参数	说明
1	<b>object</b>	接收array。表示想要创建的数组。无默认
2	<b>dtype</b>	接收data-type。表示数组所需的数据类型。如果未给定，则选择保存对象所需的最小类型。默认为None
3	<b>ndmin</b>	接收int。指定生成数组应该具有的最小维数。默认为None

## 例10-1 创建数组并查看数组属性。

```
1 import numpy as np           # 导入Numpy库
2 arr1 = np.array([1, 2, 3, 4]) # 创建的一维数组,参数为列表
3 print("数组的尺寸: ",np.shape(arr1)) # (4,) , 一个元素的元组, 表示是一维数组
4
5 arr1 = [ 1  2  3  4 ]
6
7 arr2 = np.array([[1,2,3,4],[4,5,6,7], [7,8,9,10]]) # 创建二维数组
8 print('数组的尺寸: ',np.shape(arr2)) # 2个元素的元组: (3, 4)
9 arr2.shape[0] # 返回二维数组的行数: 3
10 arr2.shape[1] # 返回二维数组的列数: 4
```

arr1 .shape (形状) : (4,)

```
arr2 = [[ 1  2  3  4 ]
         [ 4  5  6  7 ]
         [ 7  8  9 10 ]]
```

arr2 .shape (形状) : (3,4)

## 10.1.1 创建数组

numpy 提供了很多专门用来创建数组的函数（8个函数），如表10-3所示。

序号	函数名	说明	例子
1	<b>arange(a,b,x)</b>	创建含开始值a、不含终值b，步长x的一维数组	<code>np.arange(0,1,0.2) == [0. 0.2 0.4 0.6 0.8]</code>
2	<b>linspace(a,b,n)</b>	创建含开始值a、含终值b和等分个数n的一维数组	<code>np.linspace(0,10,5) == [0., 2.5, 5., 7.5, 10.]</code>
3	<b>logspace(a,b,n)</b>	生成10的a次方到10的b次方的n个元素的等比数列	<code>np.logspace(0,2,5) == [1. 3.162 10. 31.62 100.]</code>
4	<b>zeros(m)</b> <b>zeros((m,n))</b>	<b>zeros(m)</b> 创建元素全为0的一维数组 <b>zeros((m,n))</b> 创建元素全为0的二维数组	<code>np.zeros(3)</code> <code>np.zeros((2,3))</code>
5	<b>ones(m)</b> <b>ones((m,n))</b>	<b>ones(m)</b> : 创建元素全为1的一维数组 <b>ones((m,n))</b> 创建元素全为1的二维数组	<code>np.ones(3)</code> <code>np.ones((2,3))</code>
6	<b>eye(n)</b>	创建n阶单位二维数组（对角线上元素为1）	<code>np.eye(2)</code>
7	<b>diag()</b>	创建对角二维数组	<code>np.diag([2,5,-1])</code>
8	<b>full([x,y],z)</b>	生成x行y列元素全为z的二维数组	<code>np.full([2,3],5)</code>

## 10.1.1 创建数组

(1) **arange(a,b,x)** : 通过指定开始值a、终值b和步长x创建一维数组 (不含终值)

如 `np.arange(0,1,0.1)` # [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]

(2) **linspace(a,b,n)** : 通过指定开始值a、终值b和等分个数n创建一维数组 (含终值)。

如 `np.linspace(0,10,5)` # [0., 2.5, 5., 7.5, 10.]

(3) **logspace(a,b,n)** 函数创建一维数组: 生成 10的a次方到10的b次方的n个元素的等比数列。

## 10.1.1 创建数组

(4) **zeros(m)** 创建元素全为0的一维数组, **zeros((m,n))**创建元素全为0的二维数组: (m行, n列)。

如: `np.zeros(3)`, `np.zeros((2,3))`

```
= [[ 0  0  0]
   [ 0  0  0]]
= [ 0  0  0]
```

(5) **ones(m)** 创建元素全为1的一维数组, **ones((m,n))**创建元素全为1的二维数组: (m行, n列)。

如: `np.ones((2,3))`, `np.ones(3)`

```
= [[ 1  1  1]
   [ 1  1  1]]
= [ 1  1  1]
```

(6) **eye(n)** 函数创建n阶单位二维数组 (对角线上元素全为1)。

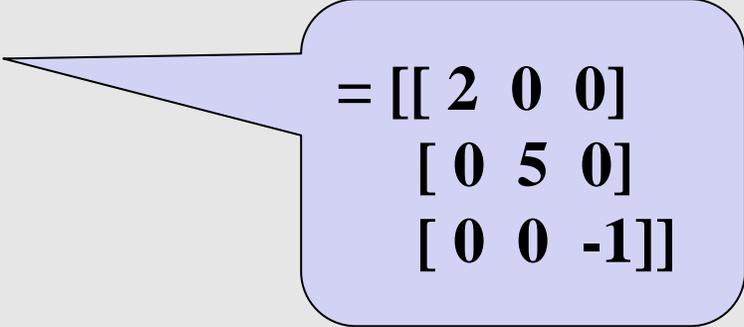
如: `np.eye(2)`

```
= [[ 1  0]
   [ 0  1]]
```

## 10.1.1 创建数组

(7) **diag()**函数创建对角二维数组。

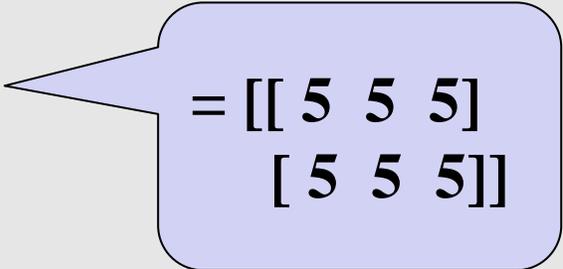
如: `np.diag([2,5,-1])`



```
= [[ 2  0  0]
   [ 0  5  0]
   [ 0  0 -1]]
```

(8) **full([x, y], z)** 生成x行y列元素全为z的二维数组。

如: `np.full([2,3],5)`



```
= [[ 5  5  5]
   [ 5  5  5]]
```

## 10.1.1 创建数组

构造复杂数组:

重复数组: **tile** , 如:

```
a = np.arange(5)
```

```
# a= [0 1 2 3 4]
```

```
b = np.tile(a, 2)
```

```
# 对变量a重复2次: b = [0 1 2 3 4 0 1 2 3 4]
```

```
c = np.tile(a, (3,2))
```

```
# 对a 行重复3次, 列重复2次
```

```
c = [[0 1 2 3 4 0 1 2 3 4]
      [0 1 2 3 4 0 1 2 3 4]
      [0 1 2 3 4 0 1 2 3 4]]
```

重复元素: **repeat** ,如:

```
d = a.repeat(2)
```

```
#对变量a里的元素依次重复2次: d= [0 0 1 1 2 2 3 3 4 4]
```



## 10.1.1 创建数组

(4) Numpy中的特殊值：**nan**和**inf**（注意：**inf**和**nan**都是**float**类型）

**nan** 是不合法数字（**Not A Number**）的缩写。什么时候**numpy**中会出现**nan**：

- 1) 当我们读取本地的文件为**float**的时候，如果有缺失，就会出现**nan**；
- 2) 当做了一个不合适的计算的时候(比如无穷大(**inf**)减去无穷大)。

**inf**是无穷大（**Infinite**）的缩写。**inf(-inf,inf)**：**infinity**，**inf**表示正无穷，**-inf**表示负无穷。什么时候会出现**inf**包括（**-inf**，**+inf**）？比如一个数字除以**0**，（**python**中直接会报错，**numpy**中是一个**inf**或者**-inf**，不会报错）。如：

```
a = np.arange(1,3)          # a = [1 2]
b = a / 0                   # [inf inf]
```

## 10.1.2 生成随机数

在numpy.random子模块中，提供了多个与随机数数相关函数，如表10-4所示。

序号	函数名	说明	例子
1	<b>random(n)</b>	返回n个随机数的一维数组，这些数为[0, 1)之间的浮点数，服从均匀分布	np.random.random(3)
2	<b>rand(x,y)</b>	生成x行y列二维数组，其元素为区间[0, 1)上的均匀分布的随机浮点数	np.random.rand(2,3)
3	<b>rand(m,x,y)</b>	生成三维数组，共有m个x行y列二维数组，其元素为区间[0, 1)上的均匀分布的随机浮点数	np.random.rand(3,2,2)
4	<b>randn(x,y)</b>	生成x行y列二维数组，其元素为标准正态分布的随机浮点数	np.random.randn(3,3)
5	<b>randint(low, high, (shape))</b>	创建一个最小值不低于low、最大值不高于high整数随机数组	np.random.randint(2,10,size = [2,5])

## 10.1.2 生成随机数

在numpy.random子模块中，提供了多个与随机数数相关函数，如表10-4所示。

序号	函数名	说明
6	seed()	确定随机数生成器的种子
7	choice(a)	参数a为列表，或数组，从a中随机取一个元素
8	y=permutation(x)	将序列x随机打乱，返回给y，此时x的值不变。如果x是多维数组，则沿其第一个坐标轴的索引随机排列数组
9	shuffle(x)	直接对序列x进行随机打乱排序。
10	normal(u,v,n)	生成n个，均值为u，方差为v的高斯分布随机浮点数

## 10.1.2 生成随机数

`np.random.seed(0)`作用：使得随机数据可预测，即只要seed的值一样，后续生成的随机数都一样。

当我们设置相同的seed，每次生成的随机数相同。如果不设置seed，则每次会生成不同的随机数。如：

```
1 import numpy as np
2 np.random.seed(0)
3 A = np.random.rand(4)           # 生成区间[0, 1)上的均匀分布的4个随机浮点数的一维数组
4 print(A)                       # [0.5488135 0.71518937 0.60276338 0.54488318]
5 a = [i for i in range(10)]      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
6 b = [np.random.choice(a) for i in range(6)] #每次从 a 中随机返回一个元素，共迭代6次
7 print(b)                       # [3, 0, 3, 5, 0, 2]
```

## 10.1.2 生成随机数

函数`shuffle`与`permutation`都可以打乱数组元素顺序，区别在于：

**shuffle**：直接在原来的数组上进行操作；

**permutation**：不直接在原来的数组上操作，会返回一个新的打乱顺序的数组。

例如：

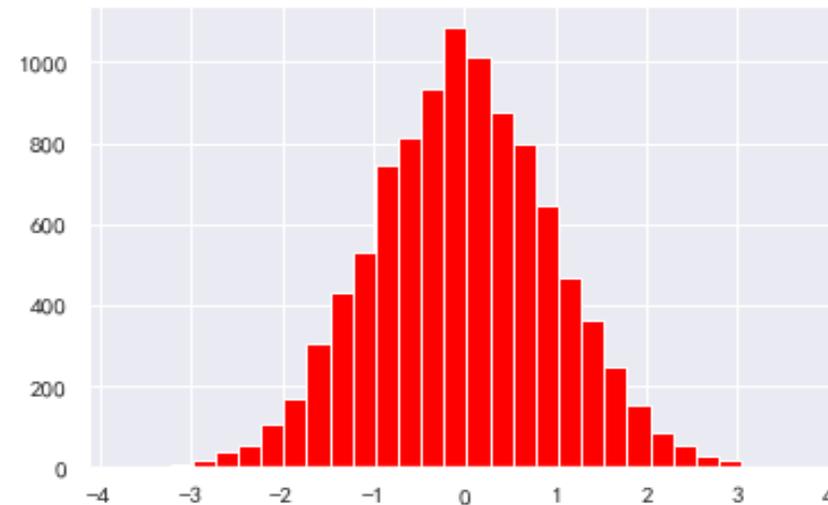
```
1 import numpy as np
2 x = np.array([1,2,3,4,5,6,7,8,9])
3 y = np.random.permutation(x) # x的数据不动，把随机打乱后的数据返给y
4 np.shuffle(x) # 直接随机打乱数组x的数据
5 print(x)
```

## 10.1.2 生成随机数

例10-2 绘制：随机生成10000数据，服从均值为0、方差为1的正态分布的直方图（间隔个数：50）。

**基本思路：** 求出这10000个随机数中的最小数、最大数，将这最小、最大数等分为50个间隔，这10000个数中，落在每个间隔的个数（频数），即为直方图的高度。

```
data = np.random.normal(0,1,10000)
x     = np.linspace(min(data),max(data),50)
```



直方图函数：`n, bins, patches = plt.hist ()` 有3个返回值：

(1) `n` 为每个小区间所含数据的个数（即频数）；

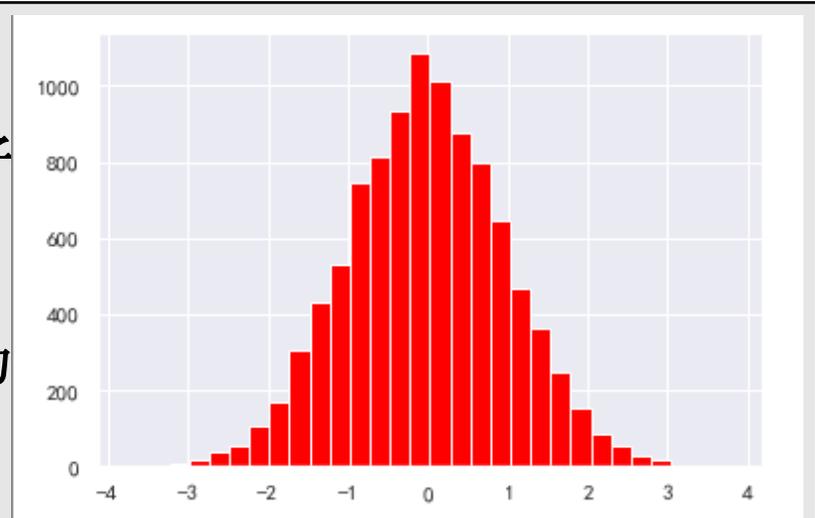
(2) `bins`相当于：`bins = np.linspace(min(data),max(data),50)`，即将10000个数中，

最小数、最大数，等分为50个数（含端点）。

## 10.1.2 生成随机数

例10-2 绘制：随机生成10000数据，服从均值为0，方差为1的正态分布的直方图（间隔个数：50）。

```
1 import numpy as np
2 import matplotlib.pyplot as plt          # 引入matplotlib 下的子
3 np.random.seed(0)
4 data = np.random.normal(0,1,10000)     # 生成10000个，均值为
5 print(min(data),max(data))             # 最小数、最大数
6 n, bins, patches = plt.hist(data,50,facecolor='red',edgecolor='white')    # 直方图函数
7 plt.grid(True)                          # 设置显示网格
8 plt.show()                               # 显示绘图
```



## 10.1.3 通过索引访问数组

(1) 一维数组的索引：与Python中的list的索引方法一致。如：

`arr[n]`: 返回索引号为 `n` 的元素

`arr[n: m]`: 返回索引号为从 `n` 的元素，到索引号为 `m` 的元素（不含 `m`）

```
1 arr = np.arange(10)           # arr = [0 1 2 3 4 5 6 7 8 9]
2 print( arr[5] )               # 首个索引号从0开始，输出：5
3 print( arr[3: 5] )            # 索引号从第3个，到第4个，输出：[3 4]
4 print( arr[: 5] )             # 索引号从第0个，到第4个，输出：[0 1 2 3 4]
5 print(arr[-1] )               # -1表示数组最后一个元素：9
6 print( arr[6:-1:2] )          # 索引号从第6个，到最后一个，2为步长，表示每隔一个元素：[6 8]
7 print( arr[5:1:-2] )          # 步长为负数时，开始索引号必须大于结束索引号，输出：[5 3]
```

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

`arr[5:1:-2]`

## 10.1.3 通过索引访问数组

(2) 多维数组的索引：多维数组的每一个维度都有一个索引，各个维度的索引之间用逗号隔开（**行、列的索引号，从0开始**）。如：

```
arr = np.array([[1, 2, 3, 4, 5],[4, 5, 6, 7, 8], [7, 8, 9, 10, 11]])
```

```
[[ 1  2  3  4  5 ]  
 [ 4  5  6  7  8 ]  
 [ 7  8  9 10 11 ]
```

在二维数组中，第1个索引为行，第2个索引为列。

```
arr[2,3] # 索引号第2行第3列的元素，输出：10  
# 等同于：arr[2][3]
```

```
[[ 1  2  3  4  5 ]  
 [ 4  5  6  7  8 ]  
 [ 7  8  9 10 11 ]
```

```
arr[0,3:5] # 索引号第0行中第3和4列的元素：[4 5]
```

```
[[ 1  2  3  4  5 ]  
 [ 4  5  6  7  8 ]  
 [ 7  8  9 10 11 ]
```

```
arr[1:,2:] #索引号第1行、第2列后面所有元素
```

```
[[ 1  2  3  4  5 ]  
 [ 4  5  6  7  8 ]  
 [ 7  8  9 10 11 ]
```

## 10.1.3 通过索引访问数组

`arr[1:,2:]`

#索引号第1行、第2列后面所有元素

```
[[1 2 3 4 5]
 [4 5 6 7 8]
 [7 8 9 10 11]]
```

`arr[2:]`

#索引号第2行所有列: [ 7 8 9 10 11 ]

```
[[1 2 3 4 5]
 [4 5 6 7 8]
 [7 8 9 10 11]]
```

`arr[:,2]`

#索引号第2列所有行: [ 3 6 9 ]

```
[[1 2 3 4 5]
 [4 5 6 7 8]
 [7 8 9 10 11]]
```

## 10.1.3 通过索引访问数组

(3) 数组的迭代：数组是一种可迭代对象。

1) 一维数组的迭代：与列表的迭代机制一致，即迭代数组中的每一个元素。如：

```
1 import numpy as np
2 a = np.arange(2,8,2)
3 for i in a:
4     print( i, end=', ' )      # 2, 4, 6
5 for i in enumerate(a):
6     print( i, i[0], i[1] )
```

`a = [ 2, 4, 6 ]`

`enumerate(a) : [ (0, 2), (1, 4), (2, 6) ]`

输出：

	i	i[0]	i[1]
	(0, 2)	0	2
	(1, 4)	1	4
	(2, 6)	2	6

## 10.1.3 通过索引访问数组

---

附：在Spyder 查询帮助：

法（1）：要查看模块的作用说明、简介，可以直接在交互区直接输入：

**print(模块名.\_\_doc\_\_)**，如：

```
import numpy as np
```

```
print(np.random.__doc__)
```

法（2）：查看某个函数的用法，**help(函数名)**，如：

```
help(np.random.permutation)
```

## 10.2 NumPy 通用函数及数组之间的运算

---

**全称通用函数（universal function），是一种能够对数组中所有元素进行操作的函数。ufunc函数是针对数组进行操作的，且使用ufunc函数比使用 math 库中的函数效率要高很多。**

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(1) 数组间的四则运算（包括一维与一维、二维与二维之间的四则运算）表示对每个数组中的对应元素分别进行四则运算，所以形状必须相同。如：

```
1 x = np.array([1,2,3]); y = np.array([4,5,6]); z = np.array("1,2,3")
2 x + y # 数组元素对应相加，结果为：[5 7 9]
3 x - y # 数组元素对应相减，结果为：[-3 -3 -3]
4 x * y # 数组元素对应相乘，结果为：[4 10 18]
5 x / y # 数组元素对应相除，结果为：[0.25 0.4 0.5]
6 x ** y # 数组元素对应幂运算，结果为：[1 32 729]
7 注意：
8 print(z.dtype); print(z.ndim); # z 的数据类型：<U5，z 的维数：0
9 x + z # 报错，数据形状不一样，不能进行运算
```

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(2) 一个数与数组的四则运算：表示这个数与数组的所有元素进行运算。如：

```
1 a = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]]) # np.shape(a): (3,4)
2 b = 2 * a
3 c = b.reshape(4,3) # 将数组b的形状改为(4,3), b不动
```

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

```
b = [[ 2  4  6  8]
      [ 8 10 12 14]
      [14 16 18 20]]
```

```
c = [[ 2  4  6]
      [ 8  8 10]
      [12 14 14]
      [16 18 20]]
```

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(3) 比较运算：>、<、==、>=、<=、!=。比较运算返回的结果是一个布尔数组，每个元素为每个数组对应元素的比较结果。如：

```
1 x = np.array([1,3,5])
2 y = np.array([2,3,4])
3 print(x<y)           # 输出: [ True False False]
4 print(x>=y)          # 输出: [False True True]
5 print(x==y)          # 输出: [False True False]
6 print(x!=y)          # 输出: [ True False True]
```

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(4) 逻辑运算：np.any函数表示逻辑“or”，np.all函数表示逻辑“and”。运算结果返回布尔值。如：

```
1 x = np.array([1,3,5])
2 y = np.array([2,3,4])
3 print(np.all(x==y))           # 输出: False
4 print(np.all(x!=y))          # 输出: False
5 print(np.any(x!=y))          # 输出: True
```

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(5) 数组的转置：一维数组的转置还是它自己；二维数组的转置，行列互换。如：

```
1 a = np.array([1,2,3,4]) # np.shape(a) = (4,) 表示一维数组
2 b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]]) # (3,4) 表示二维数组
3 c = np.transpose(a) # 一维数组 a 的转置: np.shape(c) = (4,)
4 d = np.transpose(b) # 二维数组 b 的转置: np.shape(d) = (4,3)
```

## 10.2.1 四则运算：加 (+)、减 (-)、乘 (\*)、除 (/)、幂 (\*\*)

(6) 数组的点积运算：其运算机制相当于两个矩阵的乘法运算。第1个数组的列数必需等于第2个数组的行数。如：

```
1 a = np.array([1,2,3,4]) # (4,) 表示一维数组
2 b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]]) # (3,4) 表示二维数组
3 e = np.dot(b,a) # (3, 4) 与 (4,) 点积运算
4 print(np.shape(e),e) # (3,) [30 60 90]
5 e = np.dot(a,b) # Error: shapes (4,) and (3,4)
```

第5行报错：**ValueError: shapes (4,) and (3,4) not aligned: 4 (dim 0) != 3 (dim 0)**

## 10.2.2 ufunc 函数的广播机制（慎用）

---

广播（broadcasting）是指不同形状的数组之间执行算术运算的方式。当使用 ufunc 函数进行数组计算时，ufunc 函数会对两个数组的对应元素进行计算。进行这种计算的前提是两个数组的 shape 一致。若两个数组的 shape 不一致，则 NumPy 会进行广播机制。NumPy 会进行广播机制并不容易理解，特别是在进行高维数组计算的时候。

## 10.2.2 ufunc 函数的广播机制（慎用）

---

为了更好地理解广播机制，需要遵循4个原则：

- (1) 让所有输入数组都向其中shape最长的数组看齐，shape中不足的部分都通过在前面加1补齐；
- (2) 输出数组的shape是输入数组shape的各个轴上的最大值；
- (3) 如果输入数组的某个轴和输出数组的对应轴的长度相同或者其长度为1时，这个数组能够用来计算，否则出错；
- (4) 当输入数组的某个轴的长度为1时,沿着此轴运算时都用此轴上的第一组值。

## 10.2.2 ufunc 函数的广播机制（慎用）

一维数组的广播机制，如：

```
arr1 = np.array([[0,0,0],[1,1,1],[2,2,2],[3,3,3]]) # arr1.shape : (4,3)
arr2 = np.array([1,2,3]) # arr2.shape : (3,)
print(arr1+arr2)
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} + [1 \ 2 \ 3] \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix}$$

## 10.2.2 ufunc 函数的广播机制（慎用）

二维数组的广播机制，如：

```
arr1 = np.array([[0,0,0],[1,1,1],[2,2,2],[3,3,3]]) # arr1.shape : (4,3)
```

```
arr2 = np.array([1,2,3,4]).reshape((4,1)) # arr2.shape : (4,1)
```

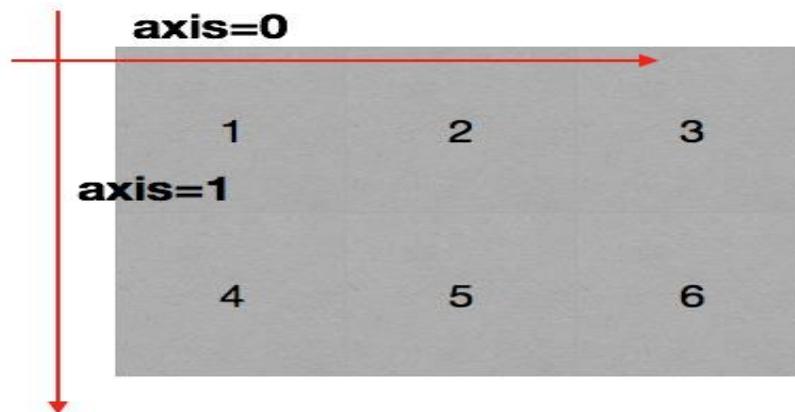
$$\begin{array}{ccc} \begin{array}{c} [0\ 0\ 0] \\ [1\ 1\ 1] \\ [2\ 2\ 2] \\ [3\ 3\ 3] \end{array} & + & \begin{array}{c} [1] \\ [2] \\ [3] \\ [4] \end{array} \\ \rightarrow & & \begin{array}{c} [0\ 0\ 0] \\ [1\ 1\ 1] \\ [2\ 2\ 2] \\ [3\ 3\ 3] \end{array} + \begin{array}{c} [1\ 1\ 1] \\ [2\ 2\ 2] \\ [3\ 3\ 3] \\ [4\ 4\ 4] \end{array} = \begin{array}{c} [1\ 1\ 1] \\ [3\ 3\ 3] \\ [5\ 5\ 5] \\ [7\ 7\ 7] \end{array} \end{array}$$

## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

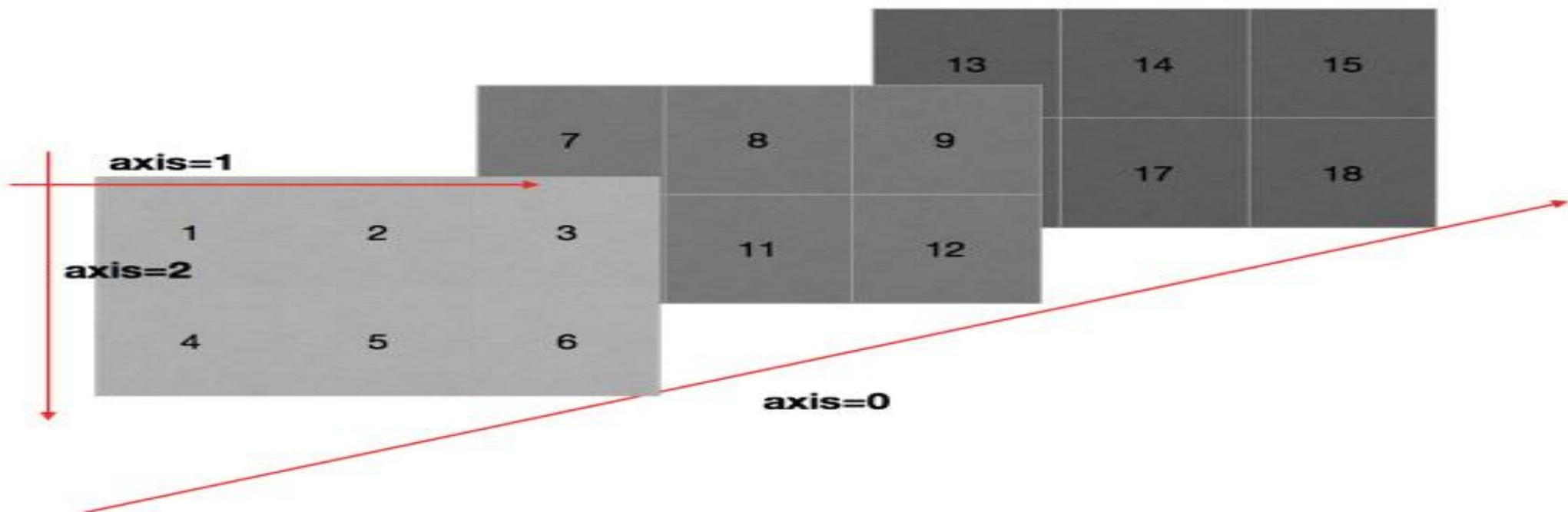
轴：二维数组的轴：

axis = 0 表示对行操作，

axis = 1 表示对列操作。



三维数组的轴：



## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

二维数组，可以通过参数axis指定操作的维度，axis = 0表示对行操作，axis = 1表示对列操作。如果没有给出参数axis，表示对数组所有元素操作，一维数组不需要指定参数axis。如：

```
arr = np.arange(20).reshape(4,5)
```

```
arr = [[ 0  1  2  3  4]
        [ 5  6  7  8  9]
        [10 11 12 13 14]
        [15 16 17 18 19]]
```

- |                     |                                  |
|---------------------|----------------------------------|
| (1) arr.sum()       | # 计算数组的所有元素的和,输出： 190            |
| (2) arr.sum(axis=0) | # 按行求和（固定列）,输出： [30 34 38 42 46] |
| (3) arr.sum(axis=1) | # 按列求和（固定行）,输出： [10 35 60 85]    |
| (4) arr.mean()      | # 计算数组的均值,输出： 9.5                |
| (5) arr.std()       | # 计算数组的标准差,输出： 5.766             |
| (6) arr.var()       | # 计算数组的方差,输出： 33.25              |

## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

二维数组，可以通过参数axis指定操作的维度，axis = 0表示对行操作，axis = 1表示对列操作。如果没有给出参数axis，表示对数组所有元素操作，一维数组不需要指定参数axis。如：

```
arr = np.arange(20).reshape(4,5)
```

```
arr = [[ 0  1  2  3  4]
       [ 5  6  7  8  9]
       [10 11 12 13 14]
       [15 16 17 18 19]]
```

(7) arr.min()

# 计算数组的最小值,输出：0

(8) arr.max()

# 计算数组的最大值,输出：19

(9) arr.argmin()

#返回数组最小元素的索引,输出：0

(10) arr.argmax()

#返回数组最大元素的索引,输出：19

(11) arr.argmin(axis=0)

#按行返回数组最小元素的索引,输出：[0 0 0 0 0]

(12) arr.argmax(axis=1)

#按列返回数组最大元素的索引,输出：[4 4 4 4]

## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

二维数组，可以通过参数axis指定操作的维度，axis = 0表示对行操作，axis = 1表示对列操作。如果没有给出参数axis，表示对数组所有元素操作，一维数组不需要指定参数axis。如：

```
arr = np.arange(20).reshape(4,5)
```

```
arr = [[ 0  1  2  3  4]
        [ 5  6  7  8  9]
        [10 11 12 13 14]
        [15 16 17 18 19]]
```

(13) arr.cumsum()

#计算所有元素的累计和,输出：

(14) arr.cumprod()

#计算所有元素的累计积,输出：

(15) np.prod(arr)

# 求所有元素的积

(16) np.ptp(arr,axis=0)

# 按行求最大元素与最小元素的差

## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

NumPy可以对数组中的元素，进行直接排序：

(1) sort函数是最常用的排序方法。arr.sort()，默认axis=1

(2) sort函数也可以指定一个axis参数，使得sort函数可以沿着指定轴对数据集进行排序。axis=1为沿横轴(x轴)排序；axis=0为沿纵轴(y轴)排序。

```
import numpy as np
arr = np.array([[8, -2, 3, 1],[2, 5, 0, 7], [7, 4, 9, 6]])
arr.sort() # 默认 axis=1 (x轴)
arr.sort(axis=0)
```

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

arr.sort(axis=0)



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

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

arr.sort()

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

## 10.2.3 利用 NumPy 进行统计分析（非常实用的常见统计函数）

NumPy可以对数组中的元素，进行间接排序：

argsort函数返回值为重新排序值的下标，用法为：`arr.argsort()`。

```
import numpy as np
arr = np.array([[8, -2, 3, 1],[2, 5, 0, 7], [7, 4, 9, 6]])
A = arr.argsort() # 排序后，元素原来的下标
```

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

`arr.sort()`

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

`arr.argsort()`

```
A = [[1 3 2 0]
     [2 0 1 3]
     [1 3 0 2]]
```

## 10.2.4 在数组中插入一行（或一列）

(1) 在数组a\_array的第row\_m行前，插入一行，其值为b\_array

语法: `np.insert ( a_array , row_m , values = b_array , axis=0 )`

```
1 a_array = np.array([[1,2,3],[4,5,6],[7,8,9]])
2 n , m = np.shape(a_array)
3 b_array = np.ones(3)
4 c1 = np.insert(a_array,0,values=b_array,axis=0) # 在第0行前插入一行
5 c2 = np.insert(a_array,1,values=b_array,axis=0) # 在第1行前插入一行
6 c3 = np.insert(a_array,n,values=b_array,axis=0) # 在最后一行后插入一行
```

```
a_array= [[1 2 3]
          [4 5 6]
          [7 8 9]]
b_array = [1. 1. 1.]
```

```
c1= [[1 1 1]
     [1 2 3]
     [4 5 6]
     [7 8 9]]
```

```
c2 = [[1 2 3]
      [1 1 1]
      [4 5 6]
      [7 8 9]]
```

```
c3= [[1 2 3]
     [4 5 6]
     [7 8 9]
     [1 1 1]]
```

## 10.2.4 在数组中插入一行（或一列）

(2) 在数组a\_array的第col\_m列前，插入一列,值为b\_array

语法: `np.insert ( a_array , col_m , values = b_array , axis=1 )`

```
1 a_array = np.array([[1,2,3],[4,5,6],[7,8,9]])
2 n , m = np.shape(a_array)
3 b_array = np.ones(3)
4 c4 = np.insert(a_array,0,values=b_array,axis=1) # 在第0列前插入一列
5 c5 = np.insert(a_array,1,values=b_array,axis=1) # 在第1列前插入一列
6 c6 = np.insert(a_array,n,values=b_array,axis=1) # 在最后一前后插入一前
```

```
a_array= [[1 2 3]
          [4 5 6]
          [7 8 9]]
b_array = [1. 1. 1.]
```

```
c4= [[1 1 2 3]
     [1 4 5 6]
     [1 7 8 9]]
```

```
c5= [[1 1 2 3]
     [4 1 5 6]
     [7 1 8 9]]
```

```
c6= [[1 2 3 1]
     [4 5 6 1]
     [7 8 9 1]]
```

## 10.3 NumPy 矩阵运算

NumPy 对于多维数组的运算，默认情况下并不进行矩阵运算。如果要对数组进行矩阵运算，则可调用相应的函数。

在NumPy中，数组和矩阵有着重要的区别。NumPy提供了两个基本的对象：一个 N 为数组对象 `ndarray` 和一个通用函数对象。其它对象都是在它们之上构建的。矩阵是继承自 NumPy 数组对象的二维数组对象，是 `ndarray` 的子类。本节讲解用 `mat`、`matrix`、`bmat` 函数来创建矩阵。

## 10.3.1 创建NumPy矩阵

调用mat函数、或matrix函数创建矩阵是等价的。如：

(1) 使用mat函数创建矩阵：

```
matr1 = np.mat('1 2 3;4 5 6;7 8 9') 或
```

```
matr1 = np.mat([[1,2,3],[4,5,6],[7,8,9]])
```

(2) 使用matrix函数创建矩阵：

```
matr2 = np.matrix([[1,2,3],[4,5,6],[7,8,9]]) 或
```

```
matr2 = np.matrix('1 2 3;4 5 6;7 8 9')
```

(3) 使用bmat函数合成矩阵：

```
np.bmat('matr1 matr2; matr1 matr2') 或
```

```
np.bmat('matr1 matr2')
```

## 10.3.2 NumPy 矩阵运算

在NumPy中，矩阵计算是针对整个矩阵中的每个元素进行的。与使用for 循环相比，其运算速度更快。如：

```
1 import numpy as np
2 matr1 = np.matrix("1 2 3;0 5 6;0 0 9")
3 matr2 = matr1 * 3          # 矩阵数乘
4 matr3 = matr1 + matr2     # 矩阵加法
5 matr4 = matr1 - matr2     # 矩阵减法
6 matr5 = matr1 * matr2     # 矩阵相乘（乘法）
7 matrB = np.multiply(matr1,matr2) # 矩阵对应元素相乘
8 matr6 = matr1.T           # 矩阵的转置
9 matr8 = matr1.H           # 矩阵的共轭转置
10 matr9 = matr1.I          # 矩阵的逆
```

```
matr1 = [[1 2 3]
         [0 5 6]
         [0 0 9]]
```

\*3  
→

```
matr2 = [[ 3  6  9]
         [ 0 15 18]
         [ 0  0 27]]
```

```
matr1 * matr2 = [[ 3 36 126]
                 [ 0 75 252]
                 [ 0  0 243]]
```

```
multiply ( matr1 , matr2 )
          = [[ 3 12 27]
            [ 0 75 108]
            [ 0  0 243]]
```

## 10.3.2 NumPy 矩阵运算

11 `arr = np.arange(6).reshape(2,3)` # 将一维向量重置为 $2 \times 3$ 矩阵

```
arr = [[0 1 2]
       [3 4 5]]
```

12 `print(arr+2)` # 矩阵中的每个元素都加 2

```
arr+2 = [[2 3 4]
        [5 6 7]]
```

13 `print(arr/0)` # 数组中的每个元素都除以 0

```
arr/0 = [[nan inf inf]
        [inf inf inf]]
```

在numpy中，`/0`并不会报错，其中nan（not a number）代表未定义或不可表示的值，inf（infimum）表示无穷。

## 10.3.3 NumPy下的线性代数运算

---

- (1) 求方阵的逆矩阵: `np.linalg.inv(A)`
- (2) 求广义逆矩阵: `np.linalg.pinv(A)`
- (3) 求矩阵的行列式: `np.linalg.det(A)`
- (4) 求矩阵的特征值: `np.linalg.eigvals (A)`
- (5) 求特征值和特征向量: `np.linalg.eig(A)`
- (6) svd分解 (矩阵的奇异值分解) : `np.linalg.svd(A)`

## 10.3.3 NumPy下的线性代数运算

### 例10-4 数组、矩阵之间的线性代数运算。

```
1 import numpy as np
2 x = np.array([[1,2,3],[0,1,-1],[1,0,0]]) # x 为二维数组：形状为 (3,3)
3 print(np.linalg.det(x)) # x 对应矩阵的行列式，输出： -5
4 y = np.linalg.inv(x) # x 对应矩阵的逆，此时的 y 为二维数组：形状为 (3,3)
5 a = np.dot(x,y) # 数组 x 与 y 做点积运算，等价于两个矩阵相乘
6 b = np.mat(x)*np.mat(y) # 将数组 x 与 y 转为矩阵，再做矩阵乘法，等价于 np.dot(x,y)
7 c = x * y # 数组x与y相乘，即两个数组对应元素相乘，注意：不同于np.dot(x,y)
8 print(a==b) # 完全相同，全为True
9 d = np.linalg.eigvals(x) # x 对应矩阵的特征值，返回的 d 为列表
10 print(d) # [-1.51154714+0.j 1.75577357+0.47447678j 1.75577357-0.47447678j ]
11 e = np.linalg.eig(x) # x 对应矩阵的特征值及特征向量，返回的 e 为元组
12 print(e[0],e[1]) # e[0]为 x 的特征值， e[1]为 x 的特征向量
```

```
x = [[1 2 3]
      [0 5 6]
      [0 0 9]]
```

## 10.3.3 NumPy下的线性代数运算

求数组x的范数语法：

```
x_norm = np.linalg.norm(x, ord=None, axis=None, keepdims=False)
```

其中x为矩阵（或向量），ord为范数类型。参数ord取值，如表所示。

序号	参数	说明	计算方法
1	默认	2-范数	$\sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$
2	ord=2	2-范数	同上
3	ord=1	1-范数	$ x_1  +  x_2  + \dots +  x_n $
4	ord=np.inf	无穷大范数	$\max  x_i $

## 10.3.3 NumPy下的线性代数运算

例如：

```
1 >>> x = np.array([3, 4])
2 >>> np.linalg.norm(x) # 向量x的范数（默认为2），输出 5
3 >>> np.linalg.norm(x, ord=2) # 向量x的2范数，输出 5
4 >>> np.linalg.norm(x, ord=1) # 向量x的1范数，输出 7
5 >>> np.linalg.norm(x, ord=np.inf) # 向量x的无穷大范数，输出 4
```

## 10.3.4 Python 中列表、矩阵、数组之间的转换

(1) 将列表转为、数组、矩阵: `np.array(list)`、`np.mat(list)`

```
1 >>> import numpy as np
2 >>> list1 = [[1,2,3],[4,5,6]]           # 列表
3 >>> list1                               # [[1, 2, 3], [4, 5, 6]]
4 >>> arr2 = np.array(list1)             # 列表 -----> 数组
5 >>> arr2                                # array([[1, 2, 3],[4, 5, 6]])
6 >>> mat3 = np.mat(list1)               # 列表 ----> 矩阵
7 >>> mat3                                # matrix([[1, 2, 3],[4, 5, 6]])
8 >>> mat4 = mat3[:,0:2]                  # 对矩阵的操作,选取其前两列的数据
9 >>> mat4                                # matrix([[1, 2],[4, 5]])
```

## 10.3.4 Python 中列表、矩阵、数组之间的转换

(2) 将矩阵、数组转为列表：矩阵变量`.tolist()`、数组变量`.tolist()`

```
1 >>> list2 = mat3.tolist()           # 矩阵 ---> 列表
2 >>> list2                           # [[1, 2, 3], [4, 5, 6]]
3 >>> list2 == list1                   # 输出: True
4 >>> list3 = list(arr2)               # 数组 ---> 列表
5 >>> list3                           # [array([1, 2, 3]), array([4, 5, 6])]
6 >>> list4 = arr2.tolist()           # 数组 ---> 列表,内部数组也转换成列表
7 >>> list4                           # [[1, 2, 3], [4, 5, 6]]
8 >>> arr3 = np.array(mat3)           # 矩阵 ---> 数组
9 >>> arr3                             #      array([[1, 2, 3], [4, 5, 6]])
10 >>> mat5 = np.mat(arr2)            # 数组 ---> 矩阵
11 >>> mat5                            # matrix([[1, 2, 3], [4, 5, 6]])
```

## 10.3.4 Python 中列表、矩阵、数组之间的转换

注意：（1）在一维情况下，矩阵 ---> 数组 ---> 矩阵结果不同  
（2）在一维情况下的列表 ----> 矩阵 ---> 列表结果不同

```
1 >>> a1 =[1,2,3,4,5,6]          # 列表 : [1, 2, 3, 4, 5, 6]
2 >>> a3 = np.mat(a1)            # 列表 --> 矩阵 : matrix([[1, 2, 3, 4, 5, 6]])
3 >>> a4 = a3.tolist()          # 矩阵 -> 列表（一个元素的列表）： [[1, 2, 3, 4, 5, 6]]
4 >>> a4[0]                      # [1, 2, 3, 4, 5, 6]
5 >>> a5 = np.mat(np.array(a1))  # 数组 ---> 矩阵 : matrix([[1, 2, 3, 4, 5, 6]])
6 >>> a6 = np.array(a5)         # 矩阵 ---> 数组
7 >>> a6                        # array([[1, 2, 3, 4, 5, 6]])
```

## 10.4. NumPy 读写文件

NumPy提供了load()、loadtxt()、save()、savetxt()等函数，对两种格式的文件进行读写：

(1) 用load()、save()读写\*.npy 或 \*.npz 文件。这是NumPy数组专用的二进制文件，其中npz (NumPy Zipped Data) 属于NumPy 数据压缩文件。

(2) 用loadtxt()、savetxt()读写\*.txt 或 \*.csv 文本文件。其中\*.csv是一种通过逗号分隔的文本文件。

函数 genfromtxt() 和函数 loadtxt() 相似，区别在于它面向的是结构化数组和缺失数据。另外，NumPy本身不能读写Excel文件，需要用 xlrd 第三方库。

## 10.4.1 用np.load()、np.save()读写npy或npz文件

函数np.load()以读写模式打开npy或npz二进制文件，格式如下，参数如表所示。

```
np.load(file, mmap_mode=None, allow_pickle=True, encoding='ASCII')
```

序号	参数名	参数说明
1	file	要打开的文件名，含文件路径
2	mmap_mode	文件打开模式，可以取None、'r+'、'r'、'w+'、'c'，默认None；如果不选择None，则进行memory-map文件，memory-map文件还存储在磁盘上，然而却可以被访问，memory mapping 对于读取大文件的小部分数据特别有用
3	allow_pickle	是否允许读取pickled对象，默认True；pickled是序列化数据对象
4	fix_imports	默认True，使用Python 3读取Python2 存储的pickled文件时有用
5	encoding	字符编码格式，默认ASCII；使用Python 3读取Python2 存储的pickled文件时有用

## 10.4.1 用np.load()、np.save()读写numpy 或 npz 文件

函数np.save()将一个数组以.npy格式保存二进制文件，格式如下：

```
np.save(file, array, allow_pickle=True, fix_imports=True)
```

序号	参数名	参数说明
1	file	要打开的文件名，含文件路径
2	array	要存储的数组
3	allow_pickle	是否允许读取pickled对象，默认True；pickled是序列化数据对象
4	fix_imports	默认True，使用Python 3读取Python2 存储的pickled文件时有用

## 10.4.1 用np.load()、 np.save()读写npy 或 npz 文件

函数np.savez()将多个数组保存到一个非压缩的.npz格式的二进制文件中，格式：

```
np.savez(file, *args, **kwds)
```

序号	参数名	参数说明
1	file	要打开的文件名，含文件路径
2	args	存储到文件中的数组的名称，如果不指定,则为"arr_0",“arr_1”...
3	kwds	存储的数组，对应与keyword的名称

## 10.4.1 用np.load()、 np.save()读写npy 或 npz 文件

---

说明：

- 1) npy 文件可以保存任意维度的 numpy 数组，不限于一维和二维；
- 2) npy 保存了 numpy 数组的结构，保存的时候是什么 shape 和 dtype，取出来时就是什么样的 shape 和 dtype；
- 3) np.save()只能保存一个numpy 数组， np.savez()可以保存多个数组，读写通过键名keyword进行标识。每次保存会覆盖掉之前文件中存在的内容（如果有的话）。

函数np.savez\_compressed(file, \*args, \*\*kwds)将多个数组存储成压缩的.npz格式文件保存，参数如 np.savez() 。

## 10.4.1 用np.load()、 np.save()读写npy 或 npz 文件

保存一个数组：

```
1 import numpy as np
2 arr = np.array([[1,2,3],[2,3,4],[3,4,5],[4,5,6]])
3 print(arr)                                # np.save()只能保存一个数组
4 np.save("save_arr", arr)                  # 保存时，默认后缀名为npy
5 load_arr = np.load("save_arr.npy")        # 打开时，必须有后缀名 (若没写路径，
                                             # 默认为当前路径下)
6 print(load_arr)                            # 返回的是 'numpy.ndarray'
```

## 10.4.1 用np.load()、 np.save()读写npy 或 npz 文件

### 保存多个数组：

```
1 import numpy as np
2 x = np.array(range(20)).reshape((2, 2, 5))      # 将0-19共20个数，生成一个三维数组： (2,2,5)
3 y = np.array(range(10, 34)).reshape(2, 3 ,4)   # 将10-33共24个数，生成一个三维数组： (2,3,4)
4 print('x:\n', x)
5 print('y:\n', y)
6 filename = 'd:\\test.npz'
7 # 写文件，如果不指定key，那么默认key为'arr_0'、 'arr_1'，一直排下去。
8 np.savez(filename, x, key_y = y)               # 数组x没有指定键名，访问时用arr_0，数组y指定了键名
9 c = np.load(filename)                          # 读文件： 多个数组，通过键名访问
10 print('keys of NpzFile c:\n', c.keys())
11 print("c['arr_0']:\n", c['arr_0'])
12 print("c['key_y']:\n", c['key_y'])
```

## 10.4.2 用np.loadtxt()、 np.savetxt()读写读写txt 或csv 文本文件

csv文件：通过逗号分隔的文本文件，可用记事本，或excel直接读取。  
函数np.loadtxt()以读写模式打开txt 或csv 文本文件，格式如下：

```
np.loadtxt(file, dtype=np.float, comments='#', delimiter=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes')
```

序号	参数名	参数说明
1	file	要打开的文件名，含文件路径
2	dtype	np支持的数据类型，默认float；如果是个structured数据类型，返回的组将会是一维的，每行解释为数组的一个元素，因此，列数应该和数据类型的成员一样
3	comments	注释标识符，默认：#
4	delimiter	数据分隔符，默认：“ ”；txt文件为空格，csv文件为逗号
5	skiprows	跳过的行数，默认为0
6	usecols	读取哪些列，为列表或元组，如：usecols=(x,y,z)获取第x,y,z列数据
7	unpack	默认False，如果为True，分会返回单个列

## 10.4.2 用np.loadtxt()、 np.savetxt()读写读写txt 或csv 文本文件

函数np.savetxt()将一个数组以txt或csv格式保存为文本文件，格式如下：

```
np.savetxt(file, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)
```

序号	参数名	参数说明
1	file	要打开的文件名，含文件路径
2	X	要存储的一维、或二维数组（只能保存一个数组）
3	comments	注释标识符，默认：#
4	delimiter	数据分隔符，默认：“ ”；txt文件为空格，csv文件为逗号
5	newline	新行的标识，默认为换行符：\n
6	header	写在文本的前面，默认：“ ”
7	footer	写在文本的后面，默认：“ ”

## 10.4.2 用np.loadtxt()、 np.savetxt()读写读写txt 或csv 文本文件

说明： csv 和 txt 只能用来存一维或二维numpy 数组；当 numpy 数组很大时，最好使用 hdf5 文件， hdf5 文件相对更小；因为若numpy 数组很大，对整个numpy 数组进行运算容易发生MemoryError，那么此时可以选择对numpy 数组切片，将运算后的数组保存到 hdf5 文件中， hdf5 文件支持切片索引。

下面代码，保存一个二维整数数组，首行加了标题；加载时，跳过第1行。

```
1 import numpy as np
2 X = np.array(range(1,10)).reshape((3, 3))           # 将1-9共9个数，生成一个二维数组： (3,3)
3 np.savetxt('d:\\test.txt', X,fmt='%i',header='A B C' )
4 y = np.loadtxt('d:\\test.txt', dtype=int, skiprows=1)
5 print(y)
```

## 10.4.2 用np.loadtxt()、np.savetxt()读写读写txt 或csv 文本文件

例10-5文件“历年总人口.xls”、“历年新生人口和死亡人口.xls”，收录了我国部分地区1949年至2019年，历年的总人口、出生人口及死亡人口。编写程序，将这3个列数据，及对应的年份读取出来，保存在文件“历年总人口、新生人口和死亡人口.csv”中。

**基本思路：**用xlrd库打开excel文件，分三步：（1）打开文件，返回workbook对象；（2）通过索引号，或sheet名称，获得sheet；（3）操作sheet下的列、行，获取行、列数据，返回给列表，然后对列表进行切片操作，获得所需要的数据。最后，将5个列表的数据，转变成一个二维数组，保存为csv文件。



	A	B
1	年份	年末总人口 (万人)
2	1949	54167
3	1950	55196
4	1951	56300
5	1952	57482
6	1953	58796



	A	B	C	D
1	年度	新生人口	死亡人口	净增人口
2	1949年	1950万	1083万	867万
3	1950年	2042万	994万	1049万
4	1951年	2128万	1002万	1126万
5	1952年	2127万	977万	1150万
6	1953年	2175万	823万	1352万

## 例10-5文件“历年总人口.xls”、“历年新生人口和死亡人口.xls”

```
1 import numpy as np
2 import xlrd
3 wb = xlrd.open_workbook("d:\\历年总人口.xls")
4 sheet = wb.sheet_by_index(0)           # 通过索引号0获取整个sheet数据
5 col_0 = sheet.col_values(0)           # 第0列数据，返回一个列表：年度
6 col_1 = sheet.col_values(1)           # 第1列数据，返回一个列表：总人口
7 year = col_0[1:]                       # 年份：从第1个元素开始，到最后一个元素
8 total = col_1[1:]                       # 总人口：从第1个元素开始，到最后一个元素
9 year = [int(c) for c in year]           # 用列表推导式，把每个元素转为整数
10 total = [int(c) for c in total]        # 用列表推导式，把每个元素转为整数
11
```

## 例10-5文件“历年总人口.xls”、“历年新生人口和死亡人口.xls”

```
12 wb = xlrd.open_workbook("d:\\历年新生人口和死亡人口.xls")
13 sheet = wb.sheet_by_index(0)           # 通过索引号0获取整个sheet数据
14 col_1 = sheet.col_values(1)           # 通过列索引号1获取列内容：出生人口
15 col_2 = sheet.col_values(2)           # 通过列索引号2获取列内容：死亡人口
16 add = col_1[1:]                       # 出生人口：从第1个元素开始，到最后一个元素
17 die = col_2[1:]                       # 死亡人口：从第1个元素开始，到最后一个元素
18 add = [int(c[0:-1]) for c in add]      # 用列表推导式，把最后一个‘万’字去掉
19 die = [int(c[0:-1]) for c in die]     # 用列表推导式，把最后一个‘万’字去掉
20 y = np.array(add)-np.array(die)       # 将两个列表变成数组相减，生成每年净增人口
21 m = len(year)
```

## 例10-5文件“历年总人口.xls”、“历年新生人口和死亡人口.xls”

```
22 arr = np.array(year).reshape(m,1)      # 将年份的列表转为数组，形状调整为： (m, 1)
23 arr = np.insert(arr,1,values=total,axis=1)      # 在第1列后面插入1列
24 arr = np.insert(arr,2,values=add,axis=1)      # 在第2列后面插入1列
25 arr = np.insert(arr,3,values=die,axis=1)      # 在第3列后面插入1列
26 arr = np.insert(arr,4,values=y,axis=1)      # 在第4列后面插入1列
27 file='d:\\大陆历年总人口、新生人口和死亡人口.csv'
28 np.savetxt(file,arr,fmt='%i',delimiter=',',comments='',header='年份,总人口,出生人口,死亡人口,净增人口')
29 x = np.loadtxt(file,dtype=np.int,delimiter=',',skiprows=1)
30 print(x)
```

```
[[ 1949 54167 1950 1083 867]
 [ 1950 55196 2042 994 1048] .....
```

## 10.4.2 用np.loadtxt()、np.savetxt()读写读写txt或csv文本文件

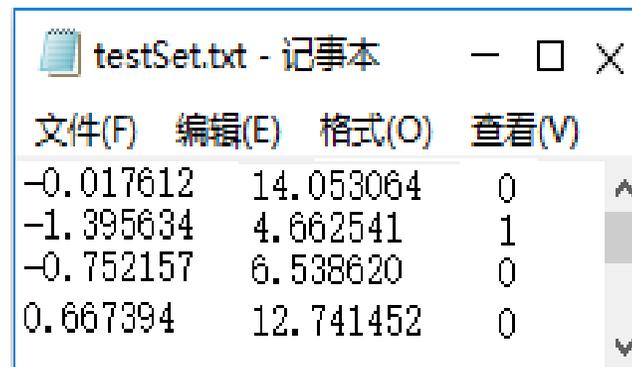
例10-6 读《testSet.txt》文件。testSet.txt是机器学习中的一个经典数据集，共100行、3列，里面包含了对二维随机变量(x,y)的100次观察数据，这些样本数据被分成了两类：0或1。数据如图10-5所示。

**基本思路：**用np.loadtxt()读取文件，返回一个二维数组，形状为(100,3)，按默认的换行符拆分行与行之间的数据，每行的数据按空格字符，拆分为3个数。

```
1 import numpy as np
2 f = np.loadtxt('d:\\testSet.txt')          # 打开文本文件
3 print('返回的二维数组f的形状: ',np.shape(f))
4 print(f)
```

返回的二维数组f的形状: (100, 3)

```
[[ -1.7612000e-02  1.4053064e+01  0.0000000e+00]
 [ -1.3956340e+00  4.6625410e+00  1.0000000e+00]
 [ -7.5215700e-01  6.5386200e+00  0.0000000e+00]
 ..... ]
```



```
testSet.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
-0.017612  14.053064  0
-1.395634  4.662541  1
-0.752157  6.538620  0
0.667394   12.741452  0
```