

# Numpy 使用教程

---

## 一、实验介绍

---

### 1.1 实验内容

如果你使用 Python 语言进行科学计算，那么一定会接触到 Numpy。Numpy 是支持 Python 语言的数值计算扩充库，其拥有强大的高维度数组处理与矩阵运算能力。除此之外，Numpy 还内建了大量的函数，方便你快速构建数学模型。

### 1.2 实验知识点

- Numpy 数组的基本操作

### 1.3 实验环境

- python2.7
- Xfce 终端
- ipython 终端

### 1.4 适合人群

本课程难度为一般，属于初级级别课程，适合具有 Python 基础，并对使用 Numpy 进行科学计算感兴趣的用户。

## 二、Numpy 数组的基本操作

---

上一个章节，我们了解了如何利用 numpy 创建各式各样的 ndarray。本章节，我们将利用学会针对 ndarray 的各种花式操作技巧。

## 2.1 重设形状

`reshape` 可以在不改变数组数据的同时，改变数组的形状。其中，`numpy.reshape()` 等效于 `ndarray.reshape()`。`reshape` 方法非常简单：

```
numpy.reshape(a, newshape)
```

其中，`a` 表示原数组，`newshape` 用于指定新的形状(整数或者元组)。

举个例子：

```
import numpy as np

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

```
In [2]: np.arange(10).reshape((5, 2))
Out[2]:
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```



## 2.2 数组展开

`ravel` 的目的是将任意形状的数组扁平化，变为 1 维数组。`ravel` 方法如下：

```
numpy.ravel(a, order='C')
```

其中，`a` 表示需要处理的数组。`order` 表示变换时的读取顺序，默认是按照行依次读取，当 `order='F'` 时，可以按列依次读取排序。

示例：

```
import numpy as np


a = np.arange(10).reshape((2, 5))

np.ravel(a)
np.ravel(a, order='F')
```

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

In [9]: np.ravel(a)
Out[9]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [10]: np.ravel(a, order='F')
Out[10]: array([0, 5, 1, 6, 2, 7, 3, 8, 4, 9])
```



## 2.3 轴移动

`moveaxis` 可以将数组的轴移动到新的位置。其方法如下：

```
numpy.moveaxis(a, source, destination)
```

其中：

- `a`：数组。
- `source`：要移动的轴的原始位置。
- `destination`：要移动的轴的目标位置。

举个例子：

```
import numpy as np

a = np.ones((1, 2, 3))

np.moveaxis(a, 0, -1)
```

```
In [22]: a
Out[22]:
array([[ [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])

In [23]: np.moveaxis(a, 0, -1)
Out[23]:
array([ [ 1.],
        [ 1.],
        [ 1.]],

       [ [ 1.],
        [ 1.],
        [ 1.]])
```



你可能没有看明白是什么意思，我们可以输出二者的 `shape` 属性：

```
In [36]: a.shape
Out[36]: (1, 2, 3)

In [37]: np.moveaxis(a, 0, -1).shape
Out[37]: (2, 3, 1)
```



## 2.4 轴交换

和 `moveaxis` 不同的是，`swapaxes` 可以用来交换数组的轴。其方法如下：

```
numpy.swapaxes(a, axis1, axis2)
```

其中：

- `a`：数组。
- `axis1`：需要交换的轴 1 位置。
- `axis2`：需要与轴 1 交换位置的轴 1 位置。

举个例子：


```
import numpy as np

a = np.ones((1, 4, 3))
np.swapaxes(a, 0, 2)
```

我们直接输出两个数组的 shape 值。

```
In [48]: a.shape
Out[48]: (1, 4, 3)

In [49]: np.swapaxes(a, 0, 2).shape
Out[49]: (3, 4, 1)
```



## 2.5 数组转置

`transpose` 类似于矩阵的转置，它可以将 2 维数组的横轴和纵轴交换。其方法如下：

```
numpy.transpose(a, axes=None)
```

其中：

- `a`：数组。
- `axis`：该值默认为 `none`，表示转置。如果有值，那么则按照值替换轴。

举个例子：

```
import numpy as np

a = np.arange(4).reshape(2,2)
np.transpose(a)
```

```
In [53]: a
Out[53]:
array([[0, 1],
       [2, 3]])

In [54]: np.transpose(a)
Out[54]:
array([[0, 2],
       [1, 3]])
```



## 2.6 维度改变

`atleast_xd` 支持将输入数据直接视为  $x$  维。这里的  $x$  可以表示：1, 2, 3。方法分别维：

```
numpy.atleast_1d()
numpy.atleast_2d()
numpy.atleast_3d()
```

举个例子：

```
import numpy as np

np.atleast_1d([1])
np.atleast_2d([1])
np.atleast_3d([1])
```

```
In [61]: np.atleast_1d([1])
Out[61]: array([1])

In [62]: np.atleast_2d([1])
Out[62]: array([[1]])

In [63]: np.atleast_3d([1])
Out[63]: array([[[1]]])
```



## 2.7 类型转变

在 numpy 中，还有一系列以 as 开头的方法，它们可以将特定输入转换为数组，亦可将数组转换为矩阵、标量，ndarray 等。如下：

- `asarray(a, dtype, order)`：将特定输入转换为数组。
- `asanyarray(a, dtype, order)`：将特定输入转换为 ndarray。
- `asmatrix(data, dtype)`：将特定输入转换为矩阵。
- `asfarray(a, dtype)`：将特定输入转换为 float 类型的数组。
- `asarray_chkfinite(a, dtype, order)`：将特定输入转换为数组，检查 NaN 或 infs。
- `asscalar(a)`：将大小为 1 的数组转换为标量。

这里以 `asmatrix(data, dtype)` 方法举例：

```
import numpy as np

a = np.arange(4).reshape(2,2)
np.asmatrix(a)
```

```
In [65]: np.asmatrix(a)
Out[65]:
matrix([[0, 1],
        [2, 3]])
```



## 2.8 数组连接

`concatenate` 可以将多个数组沿指定轴连接在一起。其方法为：

```
numpy.concatenate((a1, a2, ...), axis=0)
```

其中：

- `(a1, a2, ...)`：需要连接的数组。
- `axis`：指定连接轴。


举个例子：

```
import numpy as np

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

np.concatenate((a, b, c), axis=0)
```

```
In [96]: np.concatenate((a, b, c), axis=0)
Out[96]:
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```




这里，我们可以尝试沿着横轴连接。但要保证连接处的维数一致，所以这里用到了 `.T` 转置。

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

np.concatenate((a, b.T), axis=1)
```

```
In [15]: np.concatenate((a, b.T), axis=1)
Out[15]:
array([[1, 2, 7],
       [3, 4, 8],
       [5, 6, 9]])
```



## 2.9 数组堆叠

在 numpy 中，还有一系列以 `as` 开头的方法，它们可以将特定输入转换为数组，亦可将数组转换为矩阵、标量，`ndarray` 等。如下：



- `stack(arrays, axis)` : 沿着新轴连接数组的序列。
- `column_stack()` : 将 1 维数组作为列堆叠到 2 维数组中。
- `hstack()` : 按水平方向堆叠数组。
- `vstack()` : 按垂直方向堆叠数组。
- `dstack()` : 按深度方向堆叠数组。

这里以 `stack(arrays, axis)` 方法举例：

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.stack((a, b))
```

```
In [4]: np.stack((a, b))
Out[4]:
array([[1, 2, 3],
       [4, 5, 6]])
```



当然，也可以横着堆叠。

```
np.stack((a, b), axis=-1)
```

```
In [5]: np.stack((a, b), axis=-1)
Out[5]:
array([[1, 4],
       [2, 5],
       [3, 6]])
```



## 2.10 拆分

`split` 及与之相似的一系列方法主要是用于数组的拆分，列举如下：

- `split(ary, indices_or_sections, axis)` : 将数组拆分为多个子数组。
- `dsplit(ary, indices_or_sections)` : 按深度方向将数组拆分成多个子数组。
- `hsplit(ary, indices_or_sections)` : 按水平方向将数组拆分成多个子数

组。

- `vsplit(ary, indices_or_sections)` : 按垂直方向将数组拆分成多个子数组。

下面，我们看一看 `split` 到底有什么效果：

```
import numpy as np
```

```
a = np.arange(10)
np.split(a, 5)
```

```
In [7]: np.split(a, 5)
Out[7]: [array([0, 1]), array([2, 3]), array([4, 5]), array([6, 7]), array([8, 9])]
```


除了 1 维数组，更高维度也是可以直接拆分的。例如，我们可以将下面的数组按行拆分为 2。

```
import numpy as np
```

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

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

In [10]: np.split(a, 2)
Out[10]: [array([[0, 1, 2, 3, 4]]), array([[5, 6, 7, 8, 9]])]
```



numpy 中还有针对数组元素添加或移除的一些方法。

## 2.11 删除

- `delete(arr, obj, axis)` : 沿特定轴删除数组中的子数组。

下面，依次对 4 种方法进行示例，首先是 `delete` 删除：

```
import numpy as np

a = np.arange(12).reshape(3,4)
np.delete(a, 2, 1)
```

这里代表沿着横轴，将第 3 列(索引 2)删除。

```
In [19]: np.delete(a, 2,1)
Out[19]:
array([[ 0,  1,  3],
       [ 4,  5,  7],
       [ 8,  9, 11]])
```



当然，你也可以沿着纵轴，将第三行删除。

```
np.delete(a, 2, 0)
```

```
In [18]: np.delete(a, 2,0)
Out[18]:
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```



## 2.12 数组插入

- `insert(arr, obj, values, axis)`：依据索引在特定轴之前插入值。

再看一看 `insert` 插入, 用法和 `delete` 很相似，只是需要在第三个参数位置设置需要插入的数组对象：

```
import numpy as np

a = np.arange(12).reshape(3,4)
b = np.arange(4)

np.insert(a, 2, b, 0)
```

```

In [24]: a
Out[24]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

In [25]: b
Out[25]: array([0, 1, 2, 3])

In [26]: np.insert(a, 2, b, 0)
Out[26]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 0,  1,  2,  3],
       [ 8,  9, 10, 11]])


```

插入位置索引

沿纵轴插入行

需要插入的数组

a 为基础数组



## 2.13 附加

- `append(arr, values, axis)` : 将值附加到数组的末尾，并返回 1 维数组。

`append` 的用法也非常简单。只需要设置好需要附加的值和轴位置就好了。它其实相当于只能在末尾插入的 `insert`，所以少了一个指定索引的参数。

```

import numpy as np

a = np.arange(6).reshape(2,3)
b = np.arange(3)

np.append(a, b)

```

注意 `append` 方法返回值，默认是展平状态下的 1 维数组。

```

In [39]: np.append(a, b)
Out[39]: array([0, 1, 2, 3, 4, 5, 0, 1, 2])

```

## 2.14 重设尺寸

- `resize(a, new_shape)` : 对数组尺寸进行重新设定。


`resize` 就很好理解了，直接举例子吧：

```
import numpy as np
```

```
a = np.arange(10)
```

```
a.resize(2,5)
```

```
In [42]: a
Out[42]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```



你可能会纳闷了，这个 `resize` 看起来和上面的 `reshape` 一样呢，都是改变数组原有的形状。

其实，它们是有区别的，区别在于对原数组的影响。`reshape` 在改变形状时，不会影响原数组，相当于对原数组做了一份拷贝。而 `resize` 则是对原数组执行操作。

```
In [43]: a = np.arange(10)

In [44]: a
Out[44]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

原始数组

```
In [45]: a.reshape(2,5)
Out[45]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

执行 reshape 操作

```
In [46]: a
Out[46]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

原始数组未变化

```
In [47]: a.resize(2,5)
Out[47]:
```

执行 resize 操作

```
In [48]: a
Out[48]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

原始数组已改变



## 2.15 翻转数组

在 `numpy` 中，我们还可以对数组进行翻转操作：

- `flipplr(m)`：左右翻转数组。

- `flipud(m)` : 上下翻转数组。

举个例子：

```
import numpy as np

a = np.arange(16).reshape(4,4)
np.fliplr(a)
np.flipud(a)
```

```
[In [59]: a
Out[59]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

原始数组

```
[In [60]: np.fliplr(a)
Out[60]:
array([[ 3,  2,  1,  0],
       [ 7,  6,  5,  4],
       [11, 10,  9,  8],
       [15, 14, 13, 12]])
```

左右翻转

```
[In [61]: np.flipud(a)
Out[61]:
array([[12, 13, 14, 15],
       [ 8,  9, 10, 11],
       [ 4,  5,  6,  7],
       [ 0,  1,  2,  3]])
```

上下翻转



## 三、Numpy 随机抽样

Numpy 的随机抽样功能非常强大，主要由 `numpy.random` 模块完成。

首先，我们需要了解如何使用 `numpy` 也就是生成一些满足基本需求的随机数据。主要由以下一些方法完成：

### 3.1 `numpy.random.rand`



`numpy.random.rand(d0, d1, ..., dn)` 方法的作用为：指定一个数组，并使用  $[0, 1)$  区间随机数据填充，这些数据均匀分布。

```
import numpy as np

np.random.rand(2,5)
```

```
In [2]: np.random.rand(2,5)
Out[2]:
array([[ 0.75880975,  0.33397348,  0.12081433,  0.56852955,  0.51973896],
       [ 0.04523553,  0.15822456,  0.30239769,  0.88294404,  0.1293458 ]])
```

## 3.2 `numpy.random.randn`

`numpy.random.randn(d0, d1, ..., dn)` 与 `numpy.random.rand(d0, d1, ..., dn)` 的区别在于，返回的随机数据符合标准正太分布。

```
import numpy as np

np.random.randn(1,10)
```

```
In [3]: np.random.randn(1,10)
Out[3]:
array([[ 0.89527416,  2.4433836 ,  0.01646335,  2.31421497, -0.28329427,
        -1.79071576, -1.39651815,  1.60557547,  0.49427153, -1.11133753]])
```

## 3.3 `numpy.random.randint`

`randint(low, high, size, dtype)` 方法将会生成  $[low, high)$  的随机整数。注意这是一个半开半闭区间。

```
import numpy as np

np.random.randint(2,5,10)
```

```
In [5]: np.random.randint(2,5,10)
Out[5]: array([4, 4, 4, 2, 3, 4, 2, 4, 4, 2])
```

## 3.4 `numpy.random.random_integers`

`random_integers(low, high, size)` 方法将会生成 `[low, high]` 的 `np.int` 类型随机整数。注意这是一个闭区间。

```
import numpy as np

np.random.random_integers(2,5,10)
```

```
In [6]: np.random.random_integers(2,5,10)
Out[6]: array([2, 2, 5, 2, 3, 3, 5, 3, 5, 2])
```

2 和 5 都存在

## 3.5 `numpy.random.random_sample`

`random_sample(size)` 方法将会在 `[0, 1)` 区间内生成指定 `size` 的随机浮点数。

```
import numpy as np

np.random.random_sample([10])
```

```
In [7]: np.random.random_sample([10])
Out[7]:
array([ 0.93293675,  0.4188425 ,  0.57984876,  0.68038368,  0.25688307,
        0.07266157,  0.87866701,  0.11070143,  0.23498011,  0.15871903])
```

与 `numpy.random.random_sample` 类似的方法还有：

- `numpy.random.random([size])`
- `numpy.random.ranf([size])`
- `numpy.random.sample([size])`

它们 4 个的效果都差不多。

## 3.6 `numpy.random.choice`

`choice(a, size, replace, p)` 方法将会给定的 1 维数组里生成随机数。

```
import numpy as np

np.random.choice(10,5)
```



上面的代码将会在 `np.arange(10)` 中生成 5 个随机数。

```
In [9]: np.random.choice(10,5)
Out[9]: array([5, 1, 6, 7, 1])
```

## 3.7 概率密度分布

除了上面介绍的 6 中随机数生成方法，numpy 还提供了大量的满足特定概率密度分布的样本生成方法。它们的使用方法和上面非常相似，这里就不再一一介绍了。列举如下：

1. `numpy.random.beta(a, b, size)`：从 Beta 分布中生成随机数。
2. `numpy.random.binomial(n, p, size)`：从二项分布中生成随机数。
3. `numpy.random.chisquare(df, size)`：从卡方分布中生成随机数。
4. `numpy.random.dirichlet(alpha, size)`：从 Dirichlet 分布中生成随机数。
5. `numpy.random.exponential(scale, size)`：从指数分布中生成随机数。
6. `numpy.random.f(dfnum, dfden, size)`：从 F 分布中生成随机数。
7. `numpy.random.gamma(shape, scale, size)`：从 Gamma 分布中生成随机数。
8. `numpy.random.geometric(p, size)`：从几何分布中生成随机数。
9. `numpy.random.gumbel(loc, scale, size)`：从 Gumbel 分布中生成随机数。
10. `numpy.random.hypergeometric(ngood, nbad, nsample, size)`：从超几何分布中生成随机数。
11. `numpy.random.laplace(loc, scale, size)`：从拉普拉斯双指数分布中生成随机数。
12. `numpy.random.logistic(loc, scale, size)`：从逻辑分布中生成随机数。
13. `numpy.random.lognormal(mean, sigma, size)`：从对数正态分布中生成随机数。
14. `numpy.random.logseries(p, size)`：从对数系列分布中生成随机数。
15. `numpy.random.multinomial(n, pvals, size)`：从多项分布中生成随机数。
16. `numpy.random.multivariate_normal(mean, cov, size)`：从多变量正态

分布绘制随机样本。

17. `numpy.random.negative_binomial(n, p, size)` : 从负二项分布中生成随机数。
18. `numpy.random.noncentral_chisquare(df, nonc, size)` : 从非中心卡方分布中生成随机数。
19. `numpy.random.noncentral_f(dfnum, dfden, nonc, size)` : 从非中心 F 分布中抽取样本。
20. `numpy.random.normal(loc, scale, size)` : 从正态分布绘制随机样本。
21. `numpy.random.pareto(a, size)` : 从具有指定形状的 Pareto II 或 Lomax 分布中生成随机数。
22. `numpy.random.poisson(lam, size)` : 从泊松分布中生成随机数。
23. `numpy.random.power(a, size)` : 从具有正指数  $a-1$  的功率分布中在  $0, 1$  中生成随机数。
24. `numpy.random.rayleigh(scale, size)` : 从瑞利分布中生成随机数。
25. `numpy.random.standard_cauchy(size)` : 从标准 Cauchy 分布中生成随机数。
26. `numpy.random.standard_exponential(size)` : 从标准指数分布中生成随机数。
27. `numpy.random.standard_gamma(shape, size)` : 从标准 Gamma 分布中生成随机数。
28. `numpy.random.standard_normal(size)` : 从标准正态分布中生成随机数。
29. `numpy.random.standard_t(df, size)` : 从具有  $df$  自由度的标准学生  $t$  分布中生成随机数。
30. `numpy.random.triangular(left, mode, right, size)` : 从三角分布中生成随机数。
31. `numpy.random.uniform(low, high, size)` : 从均匀分布中生成随机数。
32. `numpy.random.vonmises(mu, kappa, size)` : 从 von Mises 分布中生成随机数。
33. `numpy.random.wald(mean, scale, size)` : 从 Wald 或反高斯分布中生成随机数。
34. `numpy.random.weibull(a, size)` : 从威布尔分布中生成随机数。
35. `numpy.random.zipf(a, size)` : 从 Zipf 分布中生成随机数。

## 四、实验总结

---

本章节介绍了针对 Nddarray 的常用操作，并了解了 Numpy.randon 类下的随机抽样方法。这两点内容都非常重要，也非常实用。由于随机抽样的方法太多，全部记忆下来不太实际，你可以多浏览几遍留下印象，需要时再查阅官方文档。

*\*本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：[Numpy 多维数组创建及属性 \(/courses/912/labs/3406/document\)](/courses/912/labs/3406/document)

下一节：[Numpy 数学函数及代数运算 \(/courses/912/labs/3424/document\)](/courses/912/labs/3424/document)