# 第3章 Use NumPy 使用NumPy

NumPy的目标是奠定Python科学计算的基石。

***The goal of NumPy is to create the corner-stone for a useful environment for scientific computing.***

——特拉维斯·奥列芬特(Travis Oliphant)

## Core Functions and Syntaxes 本章核心命令代码

- `array.tolist()` 将ndarray对象转化为列表
- `for x, y in np.nditer([a,b])` 应用广播原则，生成两元迭代器
- `numpy.arange(2,10,2)` 生成一个以2为首项，8为末项，公差为2的等差数列
- `numpy.array(['2005-02-25','2011-12-25','2020-09-20'],dtype = 'M')` 生成数据类型为日期的narray对象
- `numpy.array(ndarray_obj,copy = False,dtype = 'f')` 使用array()函数生成ndarray对象，且不复制原ndarray对象，并把数据类型更改为浮点数型
- `numpy.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)` 通过lambda匿名函数生成ndarray对象
- `numpy.fromfunction(sum_of_indices, (5, 3))` 通过自定义函数sum_of_indices和给定的网格范围(5,3) 生成ndarray对象
- `numpy.linspace(2,10,4)` 生成的等差数列是在2和10之间，数列的元素个数为4个
- `numpy.logspace(start =1,stop = 10,num = 3, base = 3)` 生成一个以1为首项，10为末项，3为公比，元素个数为3的等比数列
- `numpy.meshgrid(x, y,indexing = 'xy')` 生成一个几何形式的网格
- `numpy.nditer(x,order = 'C')` 以行优先的次序生成ndarray对象x的迭代器，可以用来遍历x中的所有元素
- `numpy.where(a<5,a+0.1,a+0.2)` 使用where()函数过滤ndarray中只符合要求的元素
- `time.time()` 获得当前时间
- `with np.nditer(data, op_flags=['readwrite']) as it:` 通过nditer()函数生成迭代器以修改data中元素的数值，data是一个自定义的ndarray对象

# 3.1 NumPy简介

大量金融数学建模是构建于矩阵运算基础之上的，这使得矩阵运算在金融建模领域具有极其重要的意义，在这一章将向大家介绍一个支持不同维度数组与矩阵运算的基础程序包——NumPy。在Python中，大多数科学计算的运算包都是以NumPy的数组作为基础的。

NumPy的前身为Jim Hugunin等在1995年创建的Numeric，此后又衍生出另外一个类似于Numeric的第三方库——NumArray。NumPy和NumArray在进行矩阵和数组运算时各有优势，但为了避免在矩阵运算时使用不同的第三方库导致的不兼容性，程序员Travis Oliphant在整合NumPy和NumArray特性的基础上，增加了一些新的矩阵运算功能，并于2006年推出第一版NumPy。

Jim Hugunin, Software programmer
Creator of the Python programming language extension, Numeric (ancestor to NumPy)

NumPy是Numerical Python的缩写。NumPy有两种发音方法：['nʌmpaɪ] 或者 ['nʌmpi]。Nympy的普及得益于它很好地满足了高效数值运算的要求，而这是由于以下几个原因：①NumPy中矩阵的存储效率和输入输出性能远优于Python中对应的其他基本数据存储方式，如多层嵌套的列表。②实现NumPy功能的代码大部分是使用C语言编写的，且NumPy的底层算法是经过精心设计的。③在NumPy中通过直接操作矩阵可以避免在Python代码中使用过多的循环语句。

以下代码对比了分别使用Python的列表和NumPy创建一个行向量，并将行向量所有的元素都增大五倍的运算时间。在作者的电脑中，完成以下操作，Python列表需要花费5.67秒，而NumPy只需要0.13 秒，NumPy的运行时间仅为列表操作时间的2.3%。

B1_Ch3_1.py

```
import numpy as np
import time
#Create a ndarray of integers in the range
#0 up to (but not including) 10,000,000
array = np.arange(1e4)
#Convert it to a list
list_array = array.tolist()
start_time = time.time()
y = [val * 5 for val in list_array]
print("List calculation time is %s seconds." % (time.time() - start_time))
#List calculation time is 5.672233819961548 seconds.
start_time = time.time()
x = array * 5
print("NumPy Array calculation time is %s seconds." % (time.time() - start_time))
#ndarray calculation time is 0.12609171867370605 seconds.
```

如前所述，作为一个Python基础库，NumPy的矩阵操作是诸多Python其他常用库 (如pandas、SciPy和Matplotlib) 的基础，因此本节将会重点介绍如何在NumPy中创建矩阵和使用矩阵的方法。在深入介绍NumPy如何进行矩阵运算前，先回顾矩阵的基本概念。如图3-1所示，线性代数中常用的对

象包括一维的**行向量** (row vector) 和**列向量** (column vector) 、二维的**矩阵** (matrix) 和三维的**元胞数组** (cell array)。在NumPy中，矩阵的维数又称为**轴** (axis)。

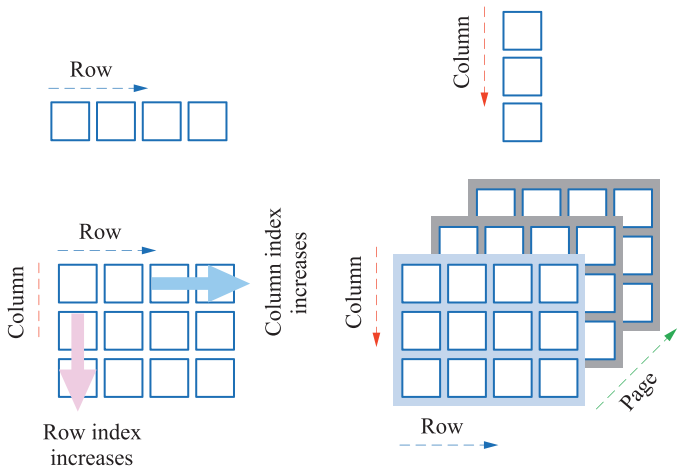


图3-1　几种常见的矩阵数据形式

NumPy提供了一些常见数据类型的缩写，便于用户快速使用和定义变量的数据类型，具体见表3-1。

表3-1　占用空间固定的NumPy数据类型

| 缩写 | 含义 |
|---|---|
| b | 布尔型 |
| i | 有符号整数 |
| u | 无符号整数 |
| f | 浮点型 |
| c | 浮点型复数 |
| m | 日期间隔 |
| M | 日期和时间 |
| S | 字符串 |
| v | void型 |

如表3-2所示，NumPy提供了众多创建ndarray矩阵对象的函数，在接下来的两节中，将会详细介绍如何使用这些函数创建ndarray矩阵对象。

如表3-3展示了一些常用的矩阵对象操作函数，部分常用的重要的函数会在本章和第4章中介绍。

表3-2　常用的创建ndarray矩阵对象的函数

| | 函数 | 描述 |
|---|---|---|
| Type 1 | np.array() | 由数列(列表，元组，ndarray等)创建矩阵 |
| | np.asarray() | 与array() 函数类似，copy默认为false |
| Type 2 | np.empty() | 创建空的ndarray矩阵对象 |
| | np.ones() | 创建元素全为1的ndarray矩阵对象 |
| | np.zeros() | 创建元素全为0的ndarray矩阵对象 |
| | np.identity() | 创建主对角线元素为1 的ndarray矩阵对象 |
| | np.eye() | 创建对角线为1的ndarray矩阵对象，对角线可偏移 |
| Type 3 | np.diag() | 提取某ndarray矩阵对角线元素的值，也可以用来生成对角矩阵 |

续表

| | 函数 | 描述 |
|---|---|---|
| Type 4 | np.arange() | 根据始末位置及步长创建矩阵 |
| | np.linspace() | 创建一维向量，向量元素为等差数列 |
| | np.logspace() | 生成对数ndarray矩阵对象 |
| Type 5 | np.ones_like() | 生成一个与原数据序列形状一样的，元素的值全为1的ndarray矩阵对象 |
| | np.empty_like() | 生成一个与原数据序列形状一样的，元素的值为空的ndarray矩阵对象 |
| | np.zeros_like() | 生成一个与原数据序列形状一样的，元素的值为0的ndarray矩阵对象 |
| | np.full_like() | 生成一个与原数据序列形状一样的，元素的值为指定值的ndarray矩阵对象 |
| Type 6 | np.meshgrid() | 生成网格矩阵对象 |
| Type 7 | np.fromfunction() | 通过函数生成ndarray矩阵对象 |

表3-3　常用的矩阵操作函数

| 函数分类 | 描述 |
|---|---|
| 矩阵形状修改 | reshape，transpose，ravel，flatten，resize，squeeze |
| 元素选择和修改 | take, put, repeat, choose, sort, argsort, partition, argpartition, searchsorted, nonzero, compress, diagonal |
| 计算类 | max, argmax, min, argmin, ptp, clip, conj, round, trace, sum, cumsum, mean, var, std, prod, all, any |
| 算术类 | __add__, __sub__, __mul__, __truediv__, __floordiv__, __mod__, __divmod__, __pow__, __lshift__, __rshift__, __and__, __or__, __xor__ |
| 矩阵转化类 | item, tolist, itemset, tostring, tobytes, tofile, dump, dumps, astype, byteswap, copy, view, getfield, setflags, fill |
| 比较类 | __lt__，__le__，__gt__，__ge__，__eq__，__ne__ |

# 3.2 基本类型的矩阵创建

NumPy运算库的核心是ndarray矩阵对象， 它封装了同质数据类型的N维数组，而ndarray正是**N维数组** (N-dimensional array) 的缩写。本节首先介绍如何由一个已有的数据序列 (如列表和元组) 创建一个ndarray矩阵对象。函数array() 和asarray() 可以帮助实现这一功能。array() 函数的定义如下。

```
np.array(data, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

其中，data是已有的数据序列，如一个列表或者元组。dtype是待创建的ndarray矩阵对象中元素的数据类型。copy参数可用来提示ndarray是否需要复制。order参数可以是“C”或者“F”。后面会对这些参数进行详细讲解。

以下代码展示了分别从列表、元组和集合使用array() 函数创建矩阵的过程。

B1_Ch3_2.py

```
import numpy as np
a_list =[1,2,3,4]
a_tuple = tuple(a_list)
```

```
a_set = set(a_list)
print(f"The original list is {a_list}")
print("The array created from a list is {}".format(np.array(a_list)))
print(f"The array created from a tuple is {np.array(a_tuple)}")
print(f"The array created from a set is {np.array(a_set)}")
print(f"The type of the array created from a tuple is {type(np.array(a_tuple))}")
print(f"The type of the array created from a set is {type(np.array(a_set))}")
```

运行结果如下。

```
The original list is [1, 2, 3, 4]
The array created from a list is [1 2 3 4]
The array created from a tuple is [1 2 3 4]
The type of the array created from a list is <class 'numpy.ndarray'>
The array created from a set is {1, 2, 3, 4}
The type of the array created from a set is <class 'numpy.ndarray'>
```

对比发现，由array() 函数创建的向量的数据类型是ndarray矩阵对象。将列表a_list打印输出时，是以“[1,2,3,4]”显示。而列表对象a_list和元组对象a_tuple转化为ndarray矩阵对象后，则是以“[1 2 3 4]”打印输出，元素中间是没有逗号的。最后由集合对象a_set转化的ndarray矩阵对象则是“{1,2,3,4}”。

在使用array() 函数创建ndarray矩阵对象时，还可以使用dtype来指定矩阵元素的数据类型。以下代码例子展示了如何使用dtype。

B1_Ch3_3.py

```
import numpy as np
import math
degree_list = [10,20,30]
sin_list = [math.sin(i) for i in degree_list]
sin_list_int = np.array(sin_list,dtype='i')
sin_list_float = np.array(sin_list,dtype='f')
date_example = np.array(['2005-02-25','2011-12-25','2020-09-20'],dtype = 'M')
date_increment = np.array([100,200,300],dtype = 'm')
date_example_updated = date_example+date_increment
print(f'Saving the data in the format of integer:{sin_list_int}')
print(f'Saving the data in the format of floating point:{sin_list_float}')
print(f'Datetime example: {date_example}')
print(f'Updated datetime is: {date_example_updated}')
```

运行结果如下。

```
Saving the data in the format of integer:[0 0 0]
Saving the data in the format of floating point:[-0.5440211  0.9129453 -0.9880316]
Datetime example: ['2005-02-25' '2011-12-25' '2020-09-20']
Updated datetime is: ['2005-06-05' '2012-07-12' '2021-07-17']
```

在这个例子中，列表sin_list存储了10°、20°和30°对应的正弦值，ndarray矩阵对象sin_list_int和sin_list_float的数据类型分别是整数型和浮点型，因此sin_list_int只存储了这些正弦值的整数部分，而sin_list_float则存储了浮点数形式的正弦值。另外，在这个例子中，使用dtype ="M"和dtype ="m"指

定了两个ndarray矩阵对象date_example和date_increment的数据类型分别是日期型和日期间隔型，将这两个ndarray矩阵对象相加时，可以获得新的日期并存储在date_example_updated中。

列表和ndarray矩阵对象还有一个很大的不同之处在于，同一个列表中允许同时存储多种不同类型的数据，如整数型、浮点数和字符串等。但ndarray矩阵对象中的数据只能是同一种数据类型。以下例子展示了这个区别。

```
import numpy as np
List_example =[10,20,30,'James']
print(List_example)
ndarray_example1 = np.array(List_example)
print(ndarray_example1)
ndarray_example2 =np.array(List_example,'S')
print(ndarray_example2)
ndarray_example3 = np.array(List_example,'i')
```

运行结果如下。

```
[10, 20, 30, 'James']
['10' '20' '30' 'James']
[b'10' b'20' b'30' b'James']
ValueError: invalid literal for int() with base 10: 'James'
```

在这个例子中，列表List_example中的变量元素包括整数型和字符串两种类型，若转化为ndarray矩阵对象时不指定dtype的参数，系统默认会把原来是整数型的变量都转化为字符串类型，这时候ndarray矩阵对象为“['10','20','30', 'James']”。当指定dtype为整数型“i”时，会出现错误提示，这是因为字符“James”不能被转化为整数型。

array() 函数的copy参数可以用来选择是否复制原来的ndarray矩阵对象。这个copy参数默认为True。然而，即使copy参数为False，在很多情况下，NumPy仍然会复制并创建一个全新的ndarray矩阵对象。如图3-2所示为array() 函数进行复制的几种情况。当原对象A是ndarray矩阵对象，且copy= False以及另一参数dtype不发生改变时，不进行复制，在其他情况下都会进行复制创建一个新的ndarray矩阵对象。

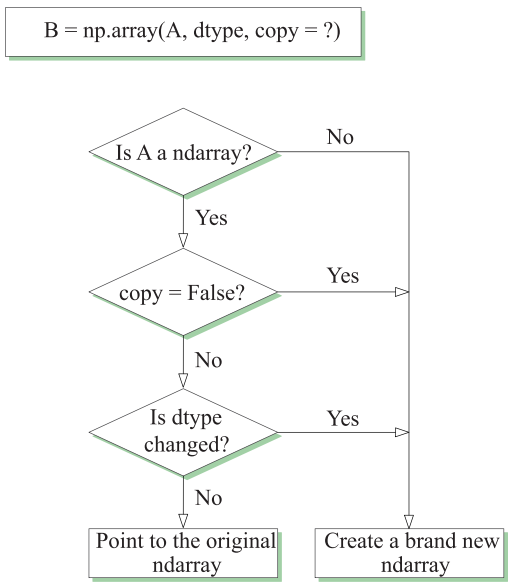


图3-2　array() 函数copy参数的使用

以下代码展示了几个由array() 函数生成的ndarray矩阵对象。在Case 1中，ndarray矩阵对象由列表生成， Case 2到Case 4均是从ndarray矩阵对象生成一个新的ndarray矩阵对象，区别在于：在Case 2中，是从一个ndarray矩阵对象生成一个新的ndarray矩阵对象；在Case 3中，指定copy = False，在Case 4中，虽然copy仍为False，但dtype更改为‘f’，用来表示数据类型是浮点数。

B1_Ch3_4.py

```
import numpy as np
list_obj = [[1,1],[1,1]]
#create a ndarray and a list, respectively
ndarray_obj = np.ones((2,2),dtype = 'i')
list_np = np.array(list_obj)

#Case 1: create a ndarray from a list
nd_1 = np.array(list_obj,copy = False)
#Case 2: use the default value for the copy parameter
nd_2 = np.array(ndarray_obj)
#Case 3: copy = false
nd_3 = np.array(ndarray_obj,copy = False)
#Case 4: change dtype
nd_4 = np.array(ndarray_obj,copy = False,dtype = 'f')

ndarray_obj[1][1]=2
list_obj[1][1] =2

print(f"The ndarray in case 1 is \n {nd_1}\n")
print(f"The ndarray in case 2 is \n {nd_2}\n")
print(f"The ndarray in case 3 is \n {nd_3}\n")
print(f"The ndarray in case 4 is \n {nd_4}\n")
```

上述例子的运行结果如下所示。修改列表和原ndarray对象中某一个元素的值后再观察新生成的ndarray对象可发现，只有Case 3中的ndarray矩阵对象的元素发生更改，这是因为这个ndarray矩阵对象的指针仍然指向原来的ndarray矩阵对象。Case1、Case2和Case 4的ndarray矩阵元素没有发生更改，因此这三个ndarray矩阵对象都是复制后新创建的ndarray矩阵对象。

```
The ndarray in case 1 is
 [[1 1]
 [1 1]]

The ndarray in case 2 is
 [[1 1]
 [1 1]]

The ndarray in case 3 is
 [[1 1]
 [1 2]]

The ndarray in case 4 is
 [[1. 1.]
```

```
[1. 1.]]
```

array() 函数中的参数order可以用来指定数据在内存中的存储方式：行优先和列优先。order参数可以是“C”和“F”，分别对应行优先存储和列优先存储。这两者分别对应C语言和Fortran语言的内存存储方式。这里有必要讨论为什么NumPy提供了两种方式来存储数据。这主要是源自几何空间索引和二维矩阵索引的矛盾。如图3-3 (a)所示，在几何空间的坐标体系里 $(x, y)$ 的第一个数字代表横坐标，沿着水平方向或者行方向索引，第二个数字代表纵坐标，沿着垂直方向或者列方向索引。然而，二维矩阵中的行坐标和列坐标的索引与几何空间坐标体系的索引方式是相反的，矩阵元素的行坐标是沿着列的方向索引，而矩阵元素的列坐标是沿着行方向索引的，如图3-3 (b) 所示。

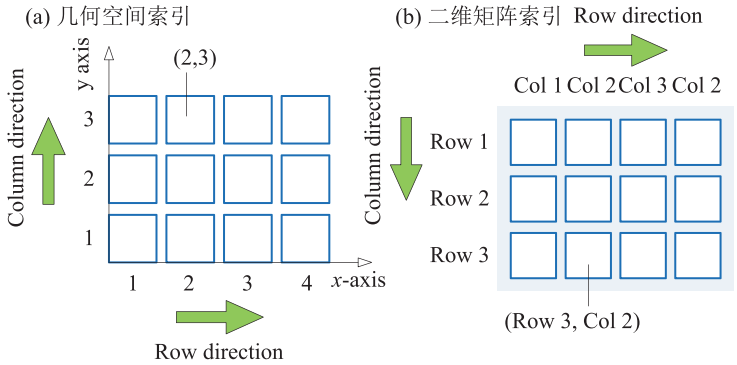


图3-3　几何空间索引和二维矩阵索引的矛盾

这一矛盾体现在数据的存储和索引方式中。如图3-4所示为一张风景图的数据。若以四方格的方式存储，根据行优先和列优先的方式，共有两种内存存储方式。当NumPy读取这一图片的数据时，若能根据图片原数据的存储方式 (行优先或列优先)，选择最优的数据索引方式能大大提高数据处理的速度，这正是array() 函数的order参数提供的便利之处。NumPy部分其函数也有order参数，同样可以指定数据在内存中的存储方式，在本节和第4章会详细介绍这些函数。

除了array() 函数以外，NumPy还提供了asarray() 函数用来创建ndarray矩阵对象。实际上，asarray() 函数可以看作array() 函数的简化版， asarray() 函数的定义如下。

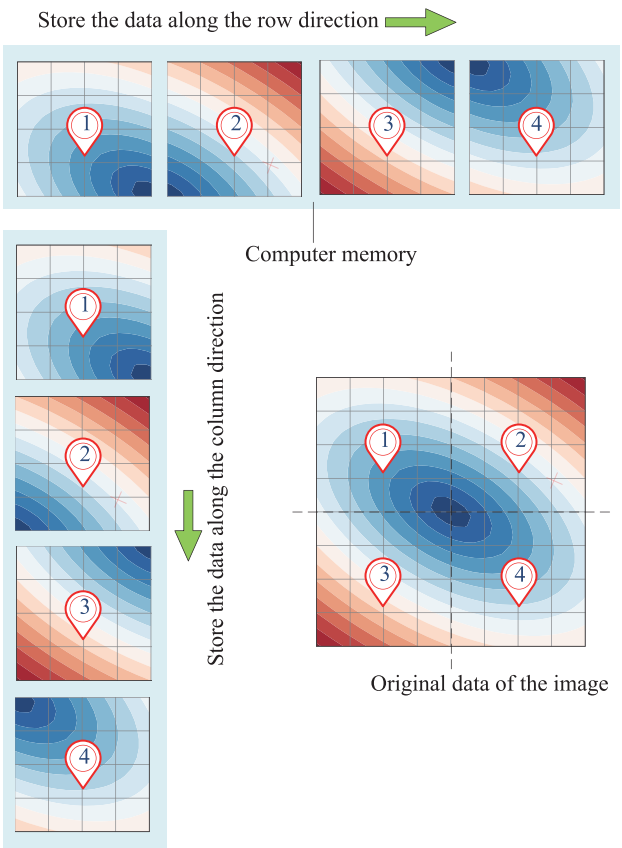


图3-4　图片数据的存储

```
def asarray(data, dtype=None, order=None):
    return array(a, dtype, copy=False, order=order)
```

从asarray() 函数的定义可以看出，它默认是不复制原ndarray矩阵对象的，然而正如图3-2中关于array() 函数的讨论一样，即copy= False，但是当原对象是非ndarray矩阵对象，或者数据格式dtype参数发生变化时，NumPy仍然会复制并生成一个全新的ndarray矩阵对象。

一个ndarray矩阵对象具有众多**属性** (attribute)。如表3-4所示，这些属性包括ndarray的位数、形状、元素总个数和数据类型等。

表3-4　ndarray矩阵对象的属性

| 属性 | 描述 |
|---|---|
| ndarray.ndim | 秩，即轴的数量或维度的数量 |
| ndarray.shape | ndarray的形状，对于矩阵，$n$行 $m$ 列 |
| ndarray.size | ndarray元素的总个数，相当于 .shape 中 $n \times m$ 的值 |
| ndarray.dtype | ndarray 矩阵对象元素的数据类型 |
| ndarray.itemsize | ndarray 矩阵对象中每个元素的大小，以字节为单位 |
| ndarray.real | ndarray元素的实部 |
| ndarray.imag | ndarray 元素的虚部 |
| ndarray. T | ndarray转置 |
| ndarray.flat | 生成一个ndarray元素迭代器 |

以下例子展示了如何使用ndarray矩阵对象的属性。

```
import numpy as np
ndarray_obj = np.array([[11,11,11],[22,22,22],[33,33,33]],dtype ='i')
print(f'The number of axes is {ndarray_obj. ndim}')
print(f'The shape is {ndarray_obj.shape}')
print(f'The size is {ndarray_obj.size}')
print(f'The data type is {ndarray_obj.dtype}')
print(f'The real parts are \n {ndarray_obj.real}')
print(f'The imaginary parts are \n {ndarray_obj.imag}')
```

运行结果如下。

```
The number of axes is 2
The shape is (3, 3)
The size is 9
The data type is int32
The real parts are
[[11 11 11]
[22 22 22]
 [33 33 33]]
The imaginary parts are
[[0 0 0]
[0 0 0]
[0 0 0]]
```

# 3.3 其他矩阵创建函数

NymPy中矩阵的创建除了前面介绍的函数，还有其他许多函数。本节首先介绍empty()、ones() 和zeros()，它们的定义和使用方法都非常类似，如下所示。

```
np.empty(shape, dtype=float, order='C')
np.ones(shape, dtype=float, order='C')
np.zeros(shape, dtype=float, order='C')
```

这三个函数分别可用于创建空矩阵、元素全为1的矩阵和元素全为0的矩阵，以下代码展示了如何使用empty()、ones() 和zeros() 函数生成ndarray矩阵对象。

```
import numpy as np
shape_list = [3,3]
shape_tuple = (2,3)
shape_ndarray = np.array([2,2])
Empty_from_list_shape = np.empty(shape_list)
Ones_from_tuple_shape = np.ones(shape_tuple)
Zeros_from_ndarray_shape = np.zeros(shape_ndarray)
print(f'The empty ndarray is \n {Empty_from_list_shape}')
print(f'The ones ndarray is \n {Ones_from_tuple_shape}')
print(f'The zeros ndarray is \n {Zeros_from_ndarray_shape}')
```

代码运行结果如下所示。这三个函数使用时，需要给定待创建的矩阵的形状shape，它可以是一个列表、元组或ndarray矩阵对象，但不能是一个集合，此外，empty() 函数生成的ndarray矩阵对象的值是系统随机产生的。

```
The empty ndarray is
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
The ones ndarray is
 [[1. 1. 1.]
 [1. 1. 1.]]
The zeros ndarray is
 [[0. 0.]
 [0. 0.]]
```

对于**对角矩阵** (diagonal matrix) 和**单位矩阵** (identity matrix)，它们可以通过identity()、eye() 和diag()三个函数创建。identity() 函数可用于生成**方阵** (square matrix)，其定义如下。

```
np. identity(n,dtype=float)
```

其中n是一个整数，表示方阵的行数或者列数，方阵的形状是 $n \times n$，dtype默认为float。另外一个可以生成对角矩阵和单位矩阵的函数是eye()，它的定义如下。

```
numpy.eye(n,m=None,k=0,dtype=float,order='C')
```

其中，n是矩阵的行数，m是矩阵的列数，m默认等于n，k是对角线的位置，dtype默认为float，

order默认为‘C’。对比这三个创建对角矩阵的函数，eye() 函数比identity() 函数更强大，identity() 只能生成方阵，形状为 $n \times n$，而eye() 函数可以生成 $n \times m$的矩阵。 此外，eye() 函数可以通过$k$调整非零元素所在对角线的位置。如图3-5所示为一个4 × 4的矩阵，该矩阵展示了$k$不同时对应的非零元素所在对角线的位置。

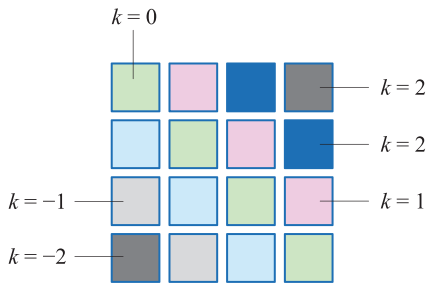


图3-5　$k$不同时对角线位置

查看NumPy的代码可以发现，identity() 函数实际上封装了eye() 的函数，以下代码对比了使用eye() 函数和identity() 函数创建对角矩阵或者单位矩阵的例子。

B1_Ch3_5.py

```
import numpy as np
identity_matrix = np.identity(3,dtype = 'i')
eye_matrix1 = np.eye(3,dtype ='i')
eye_matrix2 = np.eye(3,2,dtype ='f')
eye_matrix3 = np.eye(3,3,1,dtype = 'i')
eye_matrix4 = np.eye(3,3,-1)
print(f'The identity matrix is \n {identity_matrix}')
print(f'The identity matrix created by the eye function: \n {eye_matrix1}')
print(f'The 3×2 matrix is \n {eye_matrix2}')
print(f'The index of the diagonal is 1: \n {eye_matrix3}')
print(f'The index of the diagonal is -1: \n {eye_matrix4}')
```

上述代码的运行结果如下。

```
The identity matrix is
[[1 0 0]
[0 1 0]
[0 0 1]]
The identity matrix created by eye function:
[[1 0 0]
[0 1 0]
[0 0 1]]
The 3×2 matrix is
[[1. 0.]
[0. 1.]
[0. 0.]]
The index of the diagonal is 1:
[[0 1 0]
[0 0 1]
```

```
[0 0 0]]
The index of the diagonal is -1:
[[0. 0. 0.]
[1. 0. 0.]
[0. 1. 0.]]
```

第三种可以用来创建对角矩阵的函数是 diag()，它的定义如下。

```
numpy.diag(v, k=0)
```

其中，v可以是一维向量，也可以是二维矩阵。k表示对角线的位置，默认值为0，即为主对角线。除了可以用来创建对角矩阵，diag() 函数还可以提取某矩阵对角线上的元素。这也是diag() 函数比eye() 函数强大的地方。如图3-6展示了如何使用diag() 函数提取矩阵的对角线元素或者生成对角矩阵。

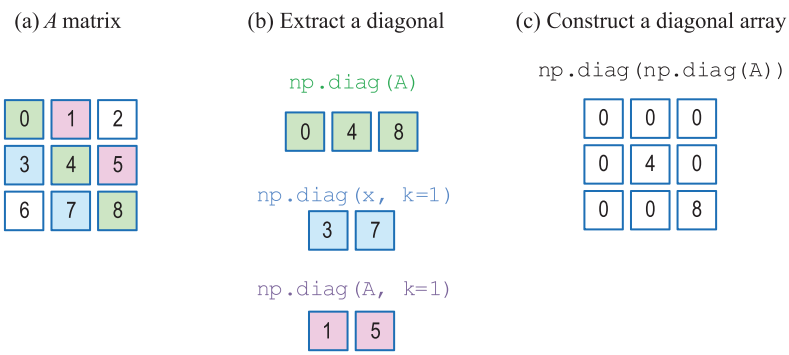


图3-6　np.diag() 使用例子，(a) 原矩阵*A*，(b) diag() 函数用于提取矩阵对角线上的元素，(c) 使用diag() 函数生成对角矩阵

以下代码展示了图3-6中的结果。

```
x = np.arange(9).reshape((3,3))
print(x)
print(np.diag(x))
print(np.diag(x, k=1))
print(np.diag(x, k=-1))
print(np.diag(np.diag(x)))
```

如图3-7 展示了如何在diag() 函数中使用一个一维向量创建不同的对角矩阵，这些对角矩阵非零元素的对角线的位置可以通过在diag() 函数中指定，矩阵的形状也因此发生变化。在图3-7(a) 中，由于对角线的位置默认为0，因此非零元素位于主对角线上，此时矩阵的形状是3 × 3。当对角线的位置分别为1和-1时，生成的矩阵的形状是 4 × 4，非零元素分别位于主对角线的上方和下方。

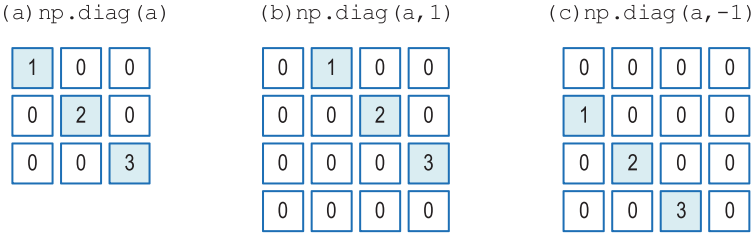


图3-7　np.diag() 通过一个向量创建矩阵 (例子中 a = [1,2,3])

接下来将会讨论表3-2中第四种矩阵生成方法，包括arange() 函数、linspace() 函数和logspace() 函

数。arange() 和linspace() 函数都可以用来生成某一区间内均匀分布的数值构成的向量。arange() 函数根据给定的起始值、终止值和步长创建一维的等差数列向量，而 linspace() 函数根据给定的起始值、终止值和向量元素个数来创建一维等差数列的向量。logspace() 函数则是创建一维等比数列的向量。

arange()、linspace()和logspace() 函数的定义如下。

```
numpy.arange(start, stop, step, dtype)

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)
```

其中，start是起始值，在arange() 函数中默认为0。stop为终止值，arange() 函数生成的等差数列不包含终止值。step为步长，默认为1。在linspace() 和logspace() 函数中需要指定生成的元素个数num，默认值为50。此外，endpoint默认为True，表示在linsapce() 和logspace() 函数生成的数列包含终止值stop，若endpoint = False，则生成的数列不包含终止值。logspace() 函数中还可以指定等比数列的底数，默认为10。虽然这三个函数只能生成一维向量，但可以通过reshape() 函数将这些一维向量元素转化为矩阵形式，在3.5节中将会详细讨论。以下代码展示了在命令窗口中使用这几个函数的例子。

```
>>> np.arange(2,10,2)
>>> array([2, 4, 6, 8])
>>> np.linspace(2,10,4)
>>> array([ 2.,4.66666667,7.33333333,10.])
>>> np.logspace(start =1,stop = 10,num = 3, base = 3)
>>> array([3.00000000e+00, 4.20888346e+02, 5.90490000e+04])
```

此外，zeros_like()、empty_like()、ones_like() 和full_like() 这四个函数可以用来创建一个和某矩阵对象形状一样的矩阵，并填充指定的元素。zeros_like()、empty_like()、ones_like() 函数分别将元素替换为0、None和1，这三个函数的定义很类似，如下所示。

```
numpy.zeros_like(a,dtype=None,order='k',shape =None)

numpy.empty_like(a,dtype=None,order='k',shape =None)

numpy.ones_like(a,dtype=None,order='k',shape =None)
```

在上述函数定义中，a是原来的ndarray矩阵对象，dtype是指定的数据类型，默认为None，即数据类型和a一致，order指定数据在内存中存储的次序，默认为k，即和a的数据在内存中的存储次序一致，shape是用来指定新ndarray矩阵对象的形状，默认为None，即形状和a一致。一般而言，不建议修改形状，因为这些函数的初衷是帮助用户创建和a形状一致的ndarray矩阵对象。

full_like() 函数稍有不同，它多了一个参数fill_value，这个参数可以用来指定新创建的ndarray矩阵对象中需要填充的元素值。full_like() 函数的其他参数和zeros_like()、empty_like() 及ones_like() 函数一致。

```
numpy.full_like(a, fill_value, dtype=None, order='K',shape=None)
```

以下代码例子展示了如何使用这四个函数。

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
ones_like_a = np.ones_like(a)
```

```
zeros_like_a = np.ones_like(a)
empty_like_a = np.empty_like(a)
full_like_a = np.full_like(a,5)
print(f'The ones_like_a matrix is\n {ones_like_a}')
print(f'The zeros_like_a matrix is\n {zeros_like_a}')
print(f'The empty_like_a matrix is\n {empty_like_a}')
print(f'The full_like_a matrix is\n {full_like_a}')
```

运行结果如下。

```
The ones_like_a matrix is
 [[1 1 1]
 [1 1 1]]
The zeros_like_a matrix is
 [[1 1 1]
 [1 1 1]]
The empty_like_a matrix is
 [[1 1 1]
 [1 1 1]]
The full_like_a matrix is
 [[5 5 5]
 [5 5 5]]
```

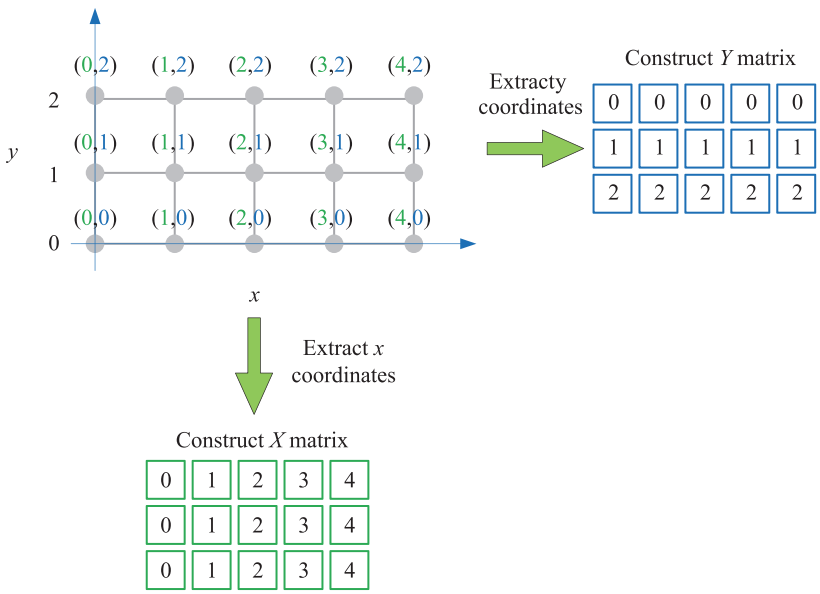


图3-8　几何形式的网格

meshgrid() 函数常常用来生成二维平面坐标系中的横纵坐标，且把横坐标$x$和纵坐标$y$分别存在$\boldsymbol{X}$和$\boldsymbol{Y}$两个矩阵里，如图3-8所示。在3D绘图里，竖坐标$z$可以是横坐标$x$和纵坐标$y$计算的函数。

此外，本节的开头介绍了几何空间的索引和矩阵的索引方式是相反的。如图3-8所示是几何空间索引。meshgrid() 函数还提供了获得二维矩阵索引号的方法。如图3-9所示为矩阵索引形式的网格。对比两种网格，采用几何空间索引方式获得的网格形状是 3 × 5，而采用矩阵形式的网格形状是5 × 3。

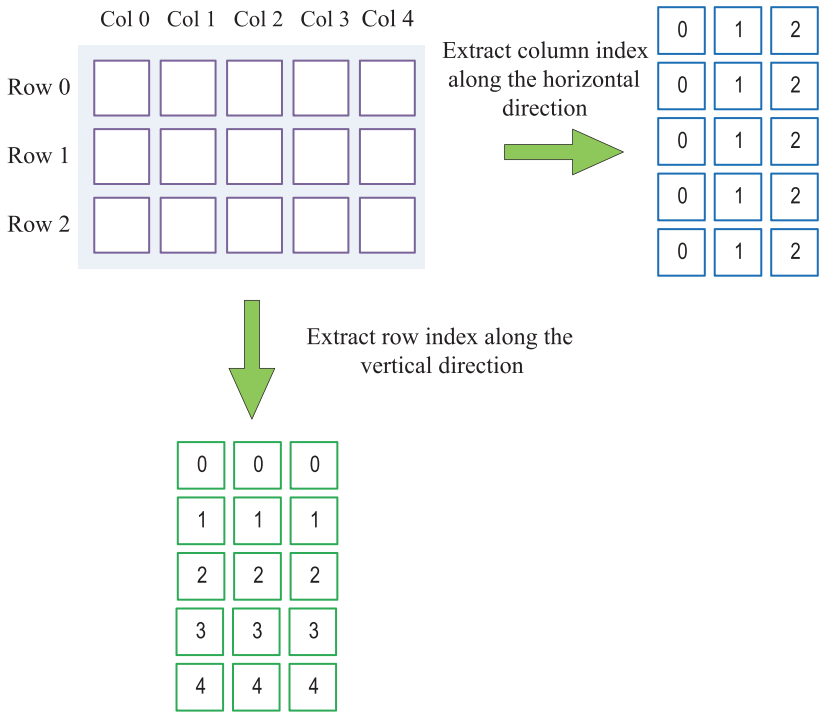


图3-9　矩阵索引形式的网格

NumPy中meshgrid() 函数的定义如下。

```
numpy.meshgrid(*xi, sparse=False, indexing='xy')
```

其中*xi代表数量不确定的一维向量，用户可提供*n*个一维向量用来表示生成*n*维网格。sparse默认为False，表示生成**稠密矩阵** (dense matrix)，当sparse为True时，表示将生成**稀疏矩阵** (sparse matrix)。在矩阵中，若数值为0的元素数目远远多于非0元素的数目，并且非0元素分布无规律时，则该矩阵为稀疏矩阵；若非0元素数目占大多数时，则该矩阵为稠密矩阵。indexing用来控制生成不同索引方式的网格，当indexing ='xy'时，表示生成几何网格，当indexing ='ij'时，表示生成矩阵索引形式的网格。

读者可运行以下代码，生成图3-8和图3-9两种不同的网格。

```
import numpy as np
x = np.linspace(0,4,5,dtype ='i')
y = np.linspace(0,2,3,dtype = 'i')
x_cartesian, y_cartesian = np.meshgrid(x,y,indexing = 'xy')
x_matrix,y_matrix =np.meshgrid(x,y,indexing='ij')
print(f'Meshgrid with Cartesian indexing:\n {x_cartesian}\n {y_cartesian}')
print(f'Meshgrid with matrix indexing:\n {x_matrix}\n {y_matrix}')
```

以下例子展示了使用meshgrid() 函数生成网格并绘制三维图，如图3-10所示。

B1_Ch3_6.py

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
x = y = np.linspace(-10, 10, 150)
X, Y = np.meshgrid(x, y,indexing = 'xy')

Z = np.cos(X) * np.sin(Y) * np.exp(-(X/5)**2-(Y/5)**2)
fig, ax = plt.subplots(figsize=(6, 5))
norm = mpl.colors.Normalize(-abs(Z).max(), abs(Z).max())
p = ax.pcolor(X, Y, Z, norm=norm, cmap=mpl.cm.bwr)
plt.colorbar(p)
```

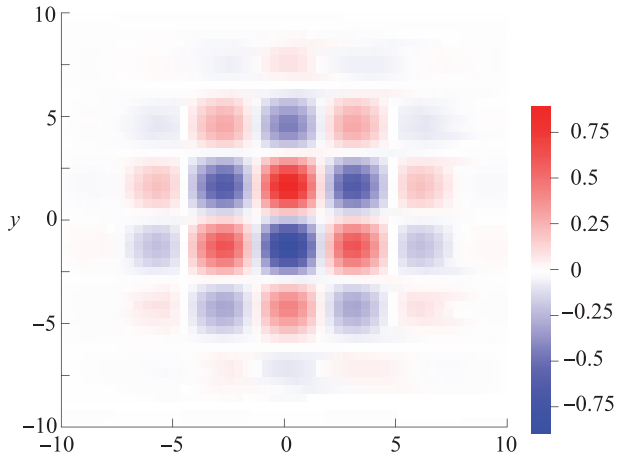


图3-10　meshgrid() 函数用于绘图

本节最后介绍如何使用fromfunction() 函数创建ndarray矩阵对象。fromfunction()函数可以通过别的函数来创建更复杂的矩阵，fromfunction() 函数的定义如下。

```
np.fromfunction(function, shape, dtype)
```

其中传入的函数function既可以是lambda匿名函数，也可以是用def定义的普通函数。shape是一个元组，用来生成矩阵网格。

以下代码展示了如何使用fromfunction() 函数和lambda匿名函数创建ndarray矩阵对象。

```
>>> np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
>>> np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

前文介绍了使用meshgrid() 函数创建二维坐标网格，生成网格的横坐标$x$和纵坐标$y$矩阵，进而通过函数计算竖坐标$z$的值。fromfunction() 函数提供了一种更简捷的方法，可以自动生成二维坐标的网格并根据函数完成计算。在以下例子中，fromfunction() 函数调用了另外一个使用def定义的函数，在调用这个函数时，还提供了一个元组 (5,3)，这个元组会被用来生成矩阵索引型网格。以下例子生成的网格如图3-9所示。

```
import numpy as np
def sum_of_indices(x, y):
    #Getting 3 individual arrays
    print(f"Value of X is:\n {x}")
    print(f"Type of X is:\n {type(x)}")
    print(f"Value of Y is:\n {y}")
    print(f"Type of Y is:\n {type(y)}")
    return x + y
a = np.fromfunction(sum_of_indices, (5, 3))
```

运行结果如下。

```
Value of X is:
 [[0. 0. 0.]
 [1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]
 [4. 4. 4.]]
Type of X is:
 <class 'numpy.ndarray'>
Value of Y is:
 [[0. 1. 2.]
 [0. 1. 2.]
 [0. 1. 2.]
 [0. 1. 2.]
 [0. 1. 2.]]
Type of Y is:
<class 'numpy.ndarray'>
```

# 3.4 索引和遍历

本节将会讨论在NumPy中进行ndarray元素的索引。对于一维ndarray矩阵对象，元素的索引方法与Python的列表和元组类似，如表3-5和图3-11所示，同样的，NumPy矩阵对象的索引号也是从0开始。

表3-5　以一维ndarray矩阵对象a为例进行索引

| 函数 | 描述 |
|---|---|
| a[m] | 索引号为$m$的元素 |
| a[−m] | 倒数第$m$个元素 |
| a[m:n] | 索引号从$m$到$n-1$的元素 |
| a[:] | 所有元素 |
| a[:n] | 索引号从0到$n-1$的元素 |
| a[m:] | 索引号$m$后的所有元素 |

续表

| 函数 | 描述 |
| --- | --- |
| a[m:n:p] | 索引号$m$到$n-1$，以$p$为间隔的元素 |
| a[::−1] | 逆序选择所有元素 |

值得注意的是，图3-11所示的例子7和例子8展示了Python列表和元组没有的索引方法。在例子7中a[a>2] 只索引了a中元素值大于2的元素。而在例子8中~np.isnan(a) 用于索引a中非NaN元素。

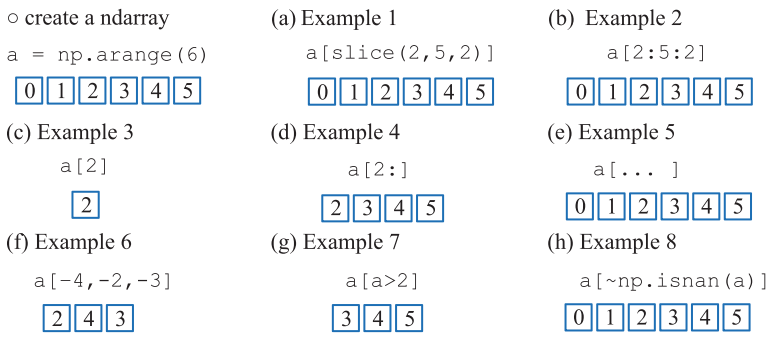


图3-11　一维数组切片和索引例子

读者可以使用函数np.where() 来实现更多复杂的过滤条件。where() 函数的定义如下。

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

使用where() 函数时，根据表达式condition给出的条件，返回表达式$x$或者表达式$y$的值，表达式$x$或者表达式$y$可以不给定。

以下例子展示了如何使用where ()函数实现将a中小于5的元素增大0.1，大于5的元素增大0.2。

```
import numpy as np
a = np.arange(10)
b = np.where(a<5,a+0.1,a+0.2)
print(b)
```

运行结果如下。

```
a vector is :
 [0 1 2 3 4 5 6 7 8 9]
b vector is:
 [0.1 1.1 2.1 3.1 4.1 5.2 6.2 7.2 8.2 9.2]
```

二维ndarray矩阵对象的元素同样可以被索引。在被索引时，需要在方括号中给定元素的行值和列值，具体如图3-12所示。

此外，读者还可以使用take() 和put() 函数进行更高级的矩阵元素索引和复制。take() 和put() 函数的优势在于通过索引号直接访问矩阵元素的数值，这些索引号可以是零散的和数量不限的。

如图3-13展示了将take() 方法应用在一维向量上，索引后同样是一维向量。如图3-14所示为使用take() 方法时，索引号是一个 2 × 4 的矩阵，索引后获得的同样是一个2 × 4 的矩阵。

矩阵对象也可以使用take() 方法进行索引。和使用方括号进行索引不同的是，take() 方法的索引号不是通过元素所在的行和列给定的，而是沿着行的方向获得。如图3-15所示为一个 3 × 4 的矩阵如何沿着行的方向获得每个矩阵元素的索引值。

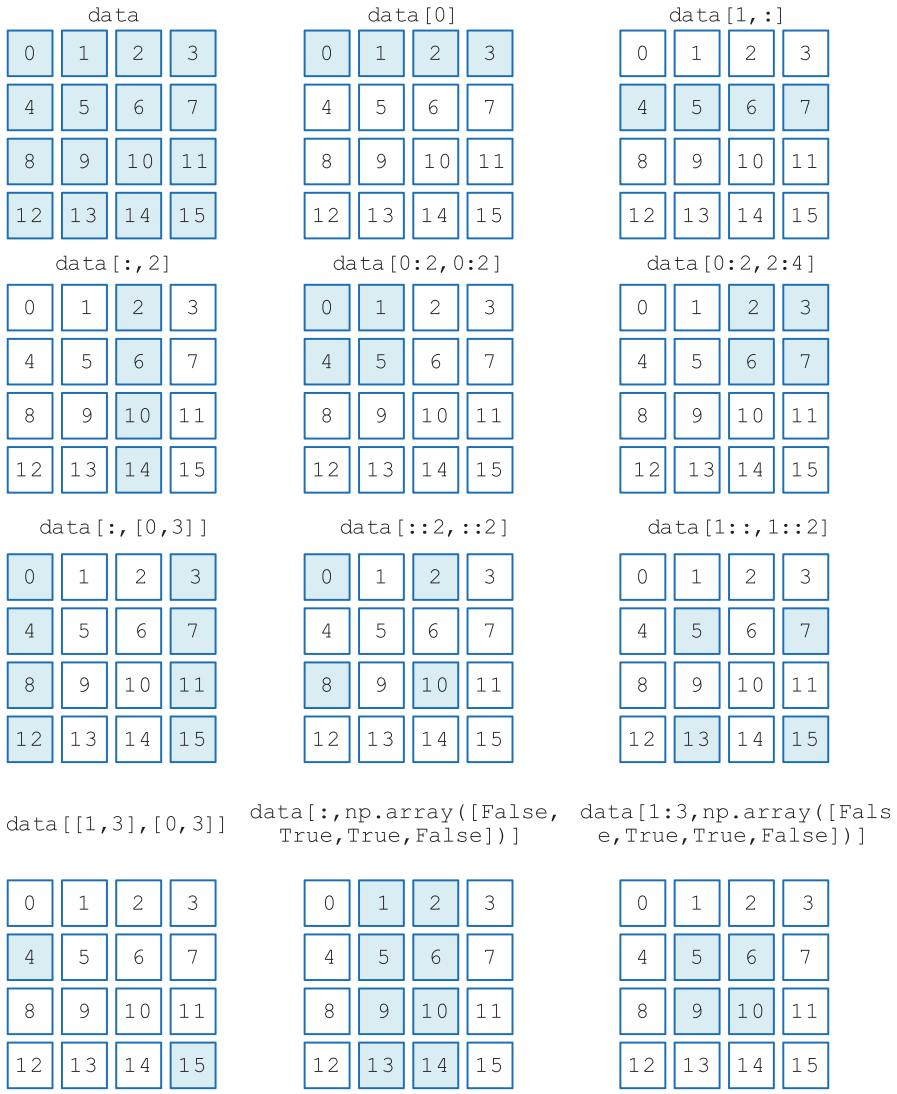


图3-12　二维ndarray矩阵对象索引示例

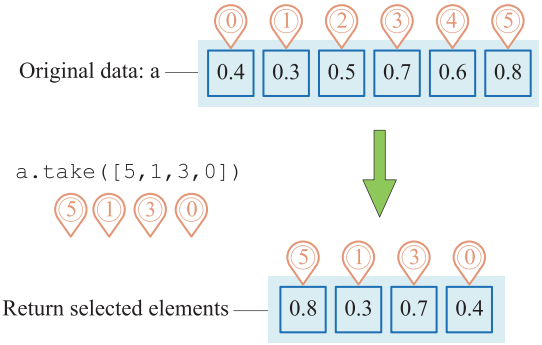


图3-13　take() 方法的应用：原始数据和索引数组都是一维向量

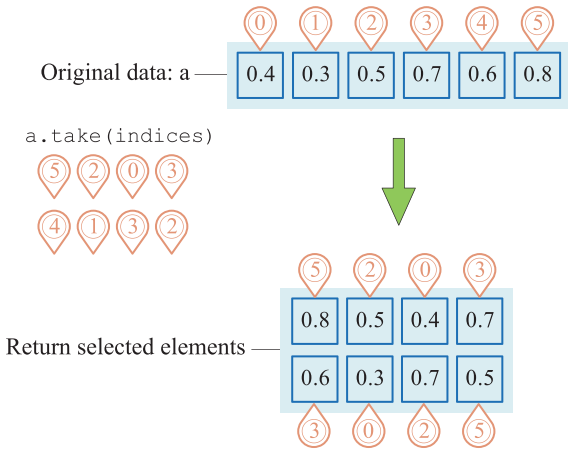


图3-14　take() 方法应用：原始数据是一维向量，索引数组是矩阵

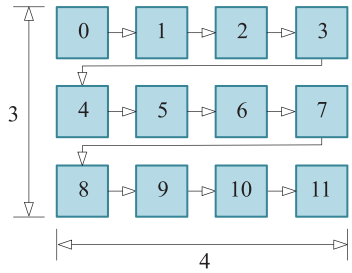


图3-15　take() 沿着行的方向进行矩阵元素索引

在图3-16(b) 中，a.take([2,3,1]) 可以返回矩阵***a***中的索引位置为2、3和1的元素数值。类似地，图3-16(c) 中a.take([12,5,2,9]) 返回矩阵***a***中索引位置为12、5、2、9的元素。

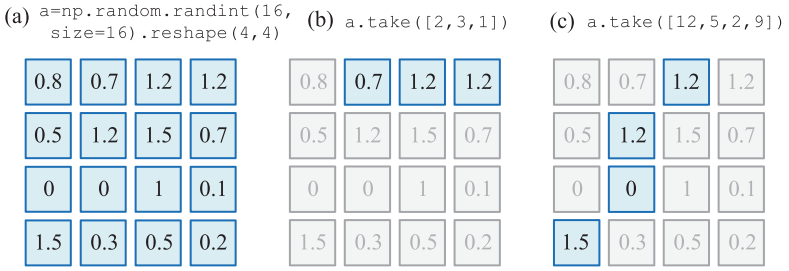


图3-16　take() 方法应用在矩阵上

take() 方法中的axis参数可以用来指定索引矩阵的某一行或某一列。如图3-17(b) 中的例子所示，a.take([3,1,2], axis =1) 可以分别获取矩阵***a***的第三列、第一列和第二列的元素。如图3-18(b)所示a.take([3,1,2], axis =0) 则可以获得矩阵***a***的第三行、第一行和第二行的元素。

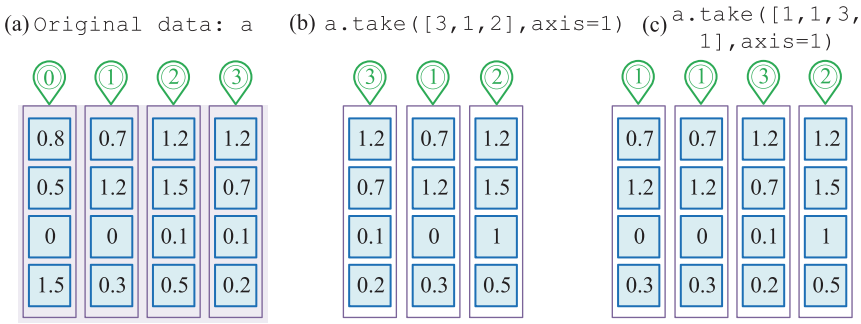


图3-17　take() 方法用于提取矩阵某一列