Python 数据分析与应用

第四章：NumPy 数值计算

丁烨
dingye@dgut.edu.cn
计算机科学与技术学院

2023-10-11

東莞理工學院
DONGGUAN UNIVERSITY OF TECHNOLOGY

目录

NumPy 简介

概述

- NumPy 是 Python 的一个扩展程序库
- 官方网站：https://www.numpy.org/
- 源代码：https://github.com/numpy/numpy

- 支持大规模的多维数组与矩阵运算
- 针对数组运算提供大量的数学函数库
- NumPy 是 SciPy、Matplotlib 等扩展程序库的基础组件

- 原作者：Travis Oliphant
- 初始版本：2006 年
- 最新版本：1.14.5（2018 年 6 月 12 日）

NumPy 简介

安装

- 使用 pip 安装 NumPy：
- `pip3 install --user -U numpy`

- 如果安装不成功，可尝试使用 apt 安装：
- `sudo apt install python3-numpy`

测试

❖ 测试 NumPy 是否安装成功：

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
```

目录

NumPy 的数组

- Numpy 的主要数据结构是一个同构多维数组，大于矩阵的概念
- 维度（dimension）在 Numpy 中称为 axis（原意坐标轴，方便起见仍可称为维度）

- 例如，在三维空间中的一个坐标：

- `[1, 2, 1]`
- 是一个一维数组，包含三个元素（element），即长度为 3

- `[[ 1., 0., 0.], [ 0., 1., 2.]]`
- 是一个二维数组，第一维长度为 2，第二维长度为 3

基本用法

NumPy 的数组

- NumPy 数组的类被称为 ndarray
- 注意，NumPy 的 numpy.array 和 Python 标准库中的 array.array 并不相同
- NumPy 的 array 仅处理一维数组，因此功能更弱

基本用法

NumPy 的数组

- ndarray 包含以下属性（attribute）：
- ndarray.ndim：数组的维度
- ndarray.shape：数组每个维度的大小，例如一个 n 行 m 列的矩阵的大小为 (n, m)
- ndarray.size：数组的元素数量，数值上等于数组每个维度大小的乘积
- ndarray.dtype：数组的数值类型，可以自定义或采用 NumPy 的数值类型，例如 numpy.int32、numpy.int16、numpy.float64 等
- ndarray.itemsize：数组的数值类型的大小，例如 float64 的大小是 8（=64/8）
- ndarray.data：直接读取数组元素，由于没有索引，不推荐使用此属性

基本用法

NumPy 的数组

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
```

```
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

基本用法

创建数组

- NumPy 提供了多种创建数组的方法
- 例如，NumPy 可以直接从标准库的列表（list）或元组（tuple）中创建数组：

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

创建数组

❖ 请注意，传递给 NumPy 数组的构造函数的参数必须是列表或元组，不能是多个元素

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

基本用法

创建数组

- NumPy 数组的构造函数会自动将传入格式转换为 NumPy 的数组格式
- 基本库的列表和元组在构造 NumPy 数组时可以混用

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

创建数组

❖ NumPy 数组的构造函数允许指定数据类型

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

基本用法

创建数组

- 在实际应用中，构造数组时具体元素的取值往往不确定，但是数组大小一般会提前确定
- 因此，NumPy 提供了一系列的函数方便构造空数组或特定元素值的数组

- 函数 zeros 会创建一个所有元素为 0 的数组
- 函数 ones 会创建一个所有元素为 1 的数组
- 函数 empty 会创建一个所有元素为随机数的数组

- 默认情况下使用上述三个函数创建的数组的数值类型是 float64

基本用法

创建数组

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> np.empty( (2,3) )
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

创建数组

❖ NumPy 提供可迅速生成数列的函数

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])

>>> np.arange( 0, 2, 0.3 )
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

基本用法

创建数组

- 由于小数精度问题，使用 arange 函数较难预测生成数组的元素数量
- 如果要指定生成的数组长度，可以使用 linspace 函数

```
>>> from numpy import pi

>>> np.linspace( 0, 2, 9 )
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])

>>> x = np.linspace( 0, 2*pi, 100 )
>>> f = np.sin(x)
```

基本用法

打印数组

- 使用 print 函数打印 NumPy 数组时，NumPy 将使用以下方法格式化输出：

- 最后一维将从左往右打印（按行）
- 倒数第二维将从上至下打印（按列）
- 其他维度将依次从上至下打印，一个元素打印完毕后会换行打印下一元素

- 因此，一维数组将以行形式输出
- 二维数组将以矩阵形式输出
- 三位数组将以矩阵的列表形式输出

基本用法

打印数组

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]

>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

>>> c = np.arange(24).reshape(2,3,4)         # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

基本用法

打印数组

❖ 如果数组太大，NumPy 会自动将中间的数值替换为省略号，只打印头尾的数值

```
>>> print(np.arange(10000))
[   0    1    2 ..., 9997 9998 9999]

>>> print(np.arange(10000).reshape(100,100))
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

❖ 如果需要禁用此行为，可以使用：np.set_printoptions(threshold=np.nan)

基本用法

基本运算

- 数组间的数学运算通常为元素级别的运算
- 运算结果会以新数组的形式返回

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

基本运算

- 请注意，与大部分数学语言（例如 Matlab）不同，乘号（*）在 NumPy 的数组运算中是求元素积，而非求数量积（点积）
- 求数量积需要使用 @ 或 dot() 函数

```
>>> A = np.array( [[1, 1],
...                [0, 1]] )
>>> B = np.array( [[2, 0],
...                [3, 4]] )
>>> A * B                           # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B                           # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                        # another matrix product
array([[5, 4],
       [3, 4]])
```

基本运算

❖ 与标准库类似，某些运算符（例如 += 和 *=）将直接修改左侧变量，而非返回新数组

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.417022  ,  3.72032449,  3.00011437],
       [ 3.30233257,  3.14675589,  3.09233859]])
>>> a += b                  # b is not automatically converted to integer type
Traceback (most recent call last):
  ...
TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule
'same_kind'
```

基本运算

❖ 如果运算包含多个不同类型的数组，运算结果将向更精确的类型对齐（upcasting）

```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = np.exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
       -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

基本运算

❖ 对于大部分一元运算（例如数组求和）来说，ndarray 都直接提供相应的函数

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                       # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

基本用法

通用函数

- NumPy 提供了一些常用的数学函数，例如 sin、cos、exp 等
- 此类数学函数被称为通用函数（universal functions）
- 与数学运算类似，通用函数通常是针对数组元素进行操作的

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```

❖ 一维数组可以直接被索引、裁剪、或迭代，行为类似标准库的列表

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000    # from start to position 6, exclusive, set every 2nd element to -1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,   729])
>>> a[ : :-1]                                 # reversed a
```

❖ 多维数组每个维度都有一套索引，调用时可用逗号隔开

```
>>> def f(x,y):
...     return 10*x+y
>>> b = np.fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]                       # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]                      # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

❖ 如果多维数组的索引没有提供，默认情况下会视为全集

```
>>> b[-1]                                   # the last row. Equivalent to b[-1,:]
array([40, 41, 42, 43])
```

❖ 上述例子中由于其他维度没有提供，NumPy 将视其他纬度为 :，即全集

❖ 除了冒号（:）之外，NumPy 也允许使用省略号代表全集，例如：b[i,...]

基本用法

索引、裁剪、迭代

- 省略号（...）代表索引中能提供的尽可能多的维度
- 例如，如果 x 是一个 5 维数组，那么：

- x[1,2,...] 等同于 x[1,2,:,:,:]
- x[...,3] 等同于 x[:,:,:,:,3]
- x[4,...,5,:] 等同于 x[4,:,:,5,:]

基本用法

索引、裁剪、迭代

```
>>> c = np.array( [[[  0,  1,  2],                # a 3D array (two stacked 2D arrays)
...                 [ 10, 12, 13]],
...                [[100,101,102],
...                 [110,112,113]]])

>>> c.shape
(2, 2, 3)

>>> c[1,...]                                      # same as c[1,:,:] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])

>>> c[...,2]                                      # same as c[:,:,2]
array([[  2,  13],
       [102, 113]])
```

基本用法

索引、裁剪、迭代

- 遍历一个多维数组在默认情况下会遍历第一维度
- 如果想遍历一个数组的全部元素，可以使用 flat 属性

```
>>> for row in b:
...     print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
>>> for element in b.flat:
...     print(element)
...
0
1
2
3
```

目录

数组形式变换

维度控制

❖ 一个数组的形状（shape）即每一维度的元素数量

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.shape
(3, 4)
```

数组形式变换

维度控制

❖ 数组的形状可以通过多种方式修改

```
>>> a.ravel()  # returns the array, flattened
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])
>>> a.reshape(6,2)  # returns the array with a modified shape
array([[ 2.,  8.],
       [ 0.,  6.],
       [ 4.,  5.],
       [ 1.,  1.],
       [ 8.,  9.],
       [ 3.,  6.]])
>>> a.T  # returns the array, transposed
array([[ 2.,  4.,  8.],
       [ 8.,  5.,  9.],
       [ 0.,  1.,  3.],
       [ 6.,  1.,  6.]])
>>> a.T.shape
(4, 3)
```

数组形式变换

维度控制

- 在 ravel() 函数中，输出的一维数组通常是深度优先遍历的（C 风格）
- 因此，如果使用 reshape() 函数，遍历方式也是深度优先
- 如果希望使用广度优先（FORTRAN 风格）或其他遍历方式，可以使用 order 参数

数组形式变换

维度控制

- 承上所述，使用 reshape() 函数调整维度时，会返回一个新的数组
- 使用 nadrray.resize() 则会修改本身的纬度

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.resize((2,6))
>>> a
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

数组形式变换

维度控制

❖ 调整数组维度时，也可以使用 -1 来自动计算剩余维度

```
>>> a.reshape(3,-1)
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
```

数组叠加和拆分

❖ NumPy 提供了数组叠加的函数 hstack 和 vstack

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 8.,  8.],
       [ 0.,  0.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

❖ 类似的，NumPy 提供 hsplit 和 vsplit 两个函数用于拆分数组

```
>>> a = np.floor(10*np.random.random((2,12)))
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])
>>> np.hsplit(a,3)   # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
       [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
       [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
       [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))   # Split a after the third and the fourth column
[array([[ 9.,  5.,  6.],
       [ 1.,  4.,  9.]]), array([[ 3.],
       [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])]
```

数组形式变换

数组叠加和拆分

- vstack 和 vsplit 是对第一维操作（vertically，对于矩阵来说，即“行”）
- hstack 和 hsplit 是对第二维操作（horizontally，对于矩阵来说，即“列”）
- 如要针对任意维操作：
- 叠加：concatenate
- https://www.numpy.org/devdocs/reference/generated/numpy.concatenate.html
- 拆分：array_split
- https://www.numpy.org/devdocs/reference/generated/numpy.array_split.html

目录

数组复制及内存控制

内存控制的概念

- NumPy 对于返回过程中的内存控制有以下三种情况：

- 无复制（No Copy at All）：数组完全不会被复制
- 浅复制（View / Shallow Copy）：数组的部分元素会被复制
- 深度复制（Deep Copy）：整个数组的内存都会被复制

数组复制及内存控制

无复制

❖ 简单的赋值操作不会复制任何内存

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
>>> b is a           # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

数组复制及内存控制

无复制

❖ Python 会将可变（mutable）对象作为引用传递，因此也不会复制任何内存

```
>>> def f(x):
...     x.shape = 4,3
...
>>> f(a)
>>> a.shape
(4, 3)
>>> b.shape
(4, 3)
```

数组复制及内存控制

浅复制

❖ view 函数会使用浅复制的方法创建一个数组对象的「视图」

```
>>> c = a.view()
>>> c is a
False
>>> c.base is a                        # c is a view of the data owned by a
True
>>> c.flags.owndata
False
>>>
>>> c.shape = 2,6                      # a's shape doesn't change
>>> a.shape
(3, 4)
>>> c[0,4] = 1234                      # a's data changes
>>> a
array([[   0,    1,    2,    3],
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```

数组复制及内存控制

浅复制

❖ 对数组进行裁剪即会使用 view 函数返回原数组的部分属性（即元素）

```
>>> s = a[ : , 1:3]     # spaces added for clarity; could also be written "s = a[:,1:3]"
>>> s[:] = 10           # s[:] is a view of s. Note the difference between s=10 and s[:]=10
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

数组复制及内存控制

深度复制

❖ copy 函数会返回一个数组的深度复制，包括所有的属性

```
>>> d = a.copy()                          # a new array object with new data is created
>>> d is a
False
>>> d.base is a                           # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

数组复制及内存控制

深度复制

- 从垃圾处理的角度来说，如果从一个巨大的数组中截取了部分元素，而原数组不再被需要，那么最好使用 copy 函数深度复制截取的子数组，并将原数组销毁

```
>>> a = np.arange(int(1e8))
>>> b = a[:100].copy()
>>> del a  # the memory of ``a`` can be released.
```

- 上面的例子中，如果使用 b = a[:100] 而不使用 copy 函数，那么 a 所占据的内存将不能被销毁

目录

索引技巧

数组索引

- NumPy 相对 Python 标准库来说提供了更多索引工具
- 例如，NumPy 允许通过一组索引来一次性检索一个数组

```
>>> a = np.arange(12)**2                        # the first 12 square numbers
>>> i = np.array( [ 1,1,3,8,5 ] )               # an array of indices
>>> a[i]                                        # the elements of a at the positions i
array([ 1,  1,  9, 64, 25])

>>> j = np.array( [ [ 3, 4], [ 9, 7 ] ] )       # a bidimensional array of indices
>>> a[j]                                        # the same shape as j
array([[ 9, 16],
       [81, 49]])
```

索引技巧

数组索引

- 当被检索的数组是多维数组时，传递数组索引将被视为检索第一维度。例如，我们可以通过检索调色盘来生成一张图片

```
>>> palette = np.array( [ [0,0,0],                # black
...                       [255,0,0],              # red
...                       [0,255,0],              # green
...                       [0,0,255],              # blue
...                       [255,255,255] ] )       # white
>>> image = np.array( [ [ 0, 1, 2, 0 ],           # each value corresponds to a color in the palette
...                     [ 0, 3, 4, 0 ]  ] )
>>> palette[image]                            # the (2,4,3) color image
array([[[  0,   0,   0],
        [255,   0,   0],
        [  0, 255,   0],
        [  0,   0,   0]],
       [[  0,   0,   0],
        [  0,   0, 255],
        [255, 255, 255],
        [  0,   0,   0]]])
```

数组索引

❖ NumPy 也支持传递多维数组作为检索索引，但是每一维的形状必须相同

```
>>> a = np.arange(12).reshape(3,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> i = np.array( [ [0,1],                        # indices for the first dim of a
...                 [1,2] ] )
>>> j = np.array( [ [2,1],                        # indices for the second dim
...                 [3,3] ] )
>>>
>>> a[i,j]                                        # i and j must have equal shape
array([[ 2,  5],
       [ 7, 11]])
```

数组索引

❖ 也可以把 i 和 j 放进一个列表里，再传递给 a

```
>>> l = [i,j]
>>> a[l]                                    # equivalent to a[i,j]
array([[ 2,  5],
       [ 7, 11]])
```

❖ 但是，不能把 i 和 j 放进一个数组里，否则按照 NumPy 的设定将视为查询第一维度

```
>>> s = np.array( [i,j] )
>>> a[s]                                    # not what we want
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: index (3) out of range (0<=index<=2) in dimension 0
>>> a[tuple(s)]                             # same as a[i,j]
array([[ 2,  5],
       [ 7, 11]])
```

数组索引

❖ 数组索引经常被用在检索时间相关的数组中

```
>>> time = np.linspace(20, 145, 5)                 # time scale
>>> data = np.sin(np.arange(20)).reshape(5,4)      # 4 time-dependent series
>>> time
array([  20.  ,   51.25,   82.5 ,  113.75,  145.  ])
>>> data
array([[ 0.        ,  0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ,  0.6569866 ],
       [ 0.98935825,  0.41211849, -0.54402111, -0.99999021],
       [-0.53657292,  0.42016704,  0.99060736,  0.65028784],
       [-0.28790332, -0.96139749, -0.75098725,  0.14987721]])
>>>
>>> ind = data.argmax(axis=0)                 # index of the maxima for each series
>>> ind
array([2, 0, 3, 1])
```

数组索引

❖ 数组索引经常被用在检索时间相关的数组中

```
>>> time_max = time[ind]                          # times corresponding to the maxima
>>>
>>> data_max = data[ind, range(data.shape[1])] # => data[ind[0],0], data[ind[1],1]...
>>>
>>> time_max
array([  82.5 ,   20.  ,  113.75,   51.25])
>>> data_max
array([ 0.98935825,  0.84147098,  0.99060736,  0.6569866 ])
>>>
>>> np.all(data_max == data.max(axis=0))
True
```

索引技巧

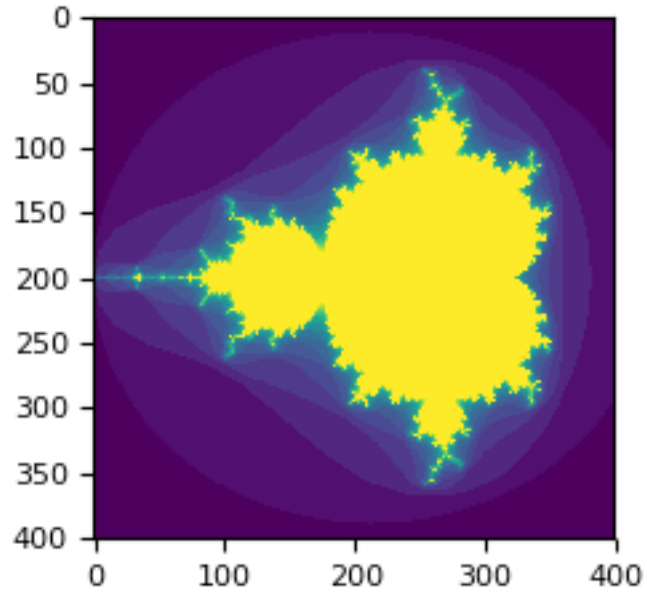
布尔索引

- 使用数组索引时，我们提供了一个索引的序列来决定哪些元素应当被选中
- 使用布尔索引时，我们通过提供布尔值来决定那些元素应当被选中
- 最简单的布尔索引即为对每一个数组元素都指定一个布尔值

```
>>> a = np.arange(12).reshape(3,4)
>>> b = a > 4
>>> b                                  # b is a boolean with a's shape
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]                               # 1d array with the selected elements
array([ 5,  6,  7,  8,  9, 10, 11])
>>> a[b] = 0                           # All elements of 'a' higher than 4 become 0
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
```

布尔索引

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> def mandelbrot( h,w, maxit=20 ):
...     """Returns an image of the Mandelbrot fractal of size (h,w)."""
...     y,x = np.ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
...     c = x+y*1j
...     z = c
...     divtime = maxit + np.zeros(z.shape, dtype=int)
...
...     for i in range(maxit):
...         z = z**2 + c
...         diverge = z*np.conj(z) > 2**2            # who is diverging
...         div_now = diverge & (divtime==maxit)  # who is diverging now
...         divtime[div_now] = i                  # note when
...         z[diverge] = 2                        # avoid diverging too much
...
...     return divtime
>>> plt.imshow(mandelbrot(400,400))
>>> plt.show()
```



布尔索引

❖ 与数组索引类似，布尔索引也可以传递多个给多维数组，从而获取对应元素

```
>>> a = np.arange(12).reshape(3,4)
>>> b1 = np.array([False,True,True])             # first dim selection
>>> b2 = np.array([True,False,True,False])       # second dim selection

>>> a[b1,:]                                   # selecting rows
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a[b1]                                     # same thing
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a[:,b2]                                   # selecting columns
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
>>> a[b1,b2]                                  # a weird thing to do
array([ 4, 10])
```

索引技巧

ix_ 索引

- NumPy 提供一个 ix_ 函数，能把两个一维数组转换为一个用于选取方形区域的索引器
- 在这个例子中，ix_ 函数将数组 [1,5,7,2] 和数组 [0,3,1,2] 产生笛卡尔积，即：(1,0), (1,3), (1,1), (1,2); (5,0), (5,3), (5,1), (5,2); (7,0), (7,3), (7,1), (7,2); (2,0), (2,3), (2,1), (2,2)
- 按照坐标 (1,0), (1,3), (1,1), (1,2) 取得 arr2 所对应的元素 4, 7, 5, 6
- 按照坐标 (5,0), (5,3), (5,1), (5,2) 取得 arr2 所对应的元素 20, 23, 21, 22
- 如此类推

```
>>> arr2 = np.arange(32).reshape((8,4))
>>> arr2
array([[ 0, 1, 2, 3],
[ 4, 5, 6, 7],
[ 8, 9, 10, 11],
[12, 13, 14, 15],
[16, 17, 18, 19],
[20, 21, 22, 23],
[24, 25, 26, 27],
[28, 29, 30, 31]])

>>> arr2[np.ix_([1,5,7,2],[0,3,1,2])]
array([[ 4, 7, 5, 6],
[20, 23, 21, 22],
[28, 31, 29, 30],
[ 8, 11, 9, 10]])
```

目录

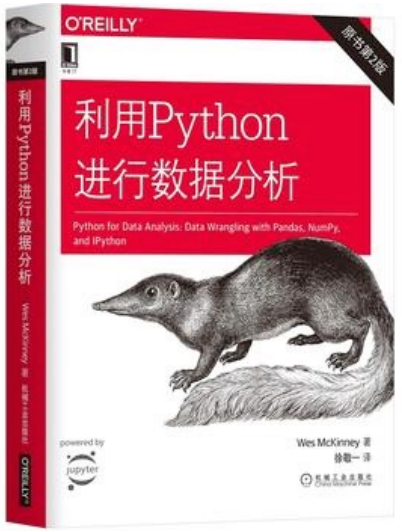
进阶功能

- 更多的进阶功能可以参考官方教程：
- https://www.numpy.org/devdocs/user/quickstart.html

- NumPy 的实用小技巧：
- https://www.numpy.org/devdocs/user/quickstart.html#tricks-and-tips

- 函数详细说明文档及手册：
- https://www.numpy.org/devdocs/reference/routines.html

- Wes McKinney《利用 Python 进行数据分析》：
- http://product.dangdang.com/25312917.html



Thanks!