01

第 1 章

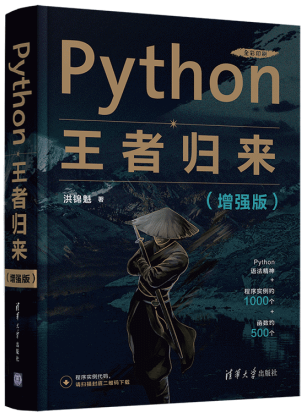
学习 matplotlib 需要的 NumPy 知识

Python 是一个应用范围很广的程序语言，虽然列表 (list) 和元组 (tuple) 可以执行绘制图表所需的一维数组 (one-dimension array) 或是多维数组 (multi-dimension array) 运算的数据，但是，如果我们强调需要使用高速计算时，就必须使用 NumPy，事实上，matplotlib 模块是建立在 NumPy 基础上的绘图函数库。如果使用列表 (list) 和元组 (tuple) 当作绘图的数据来源，虽然简单，伴随的优点却同时产生了下列缺点：

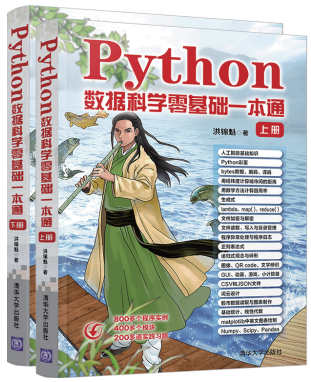
- 执行速度慢。
- 需要较多系统资源。

为此，许多高速运算的模块应运而生，在科学运算或人工智能领域最常见、应用最广的模块是 NumPy，此名称所代表的英文是 Numerical Python。本章将针对未来操作 matplotlib 需要的 NumPy 知识做一个完整的说明。

本书主要是使用 Python 讲解 matplotlib 的完整知识，如果读者不熟悉 Python，可以阅读下列书籍。



Python 王者归来（增强版）



Python 数据科学零基础一本通

1-1 数组 ndarray

NumPy 模块所建立的数组数据形态称为 ndarray(n-dimension array)，n 代表维度，例如一维数组、二维数组、……、n 维数组。ndarray 数组的几个特色如下：

- 数组大小是固定的。
- 数组元素内容的数据形态是相同的。

也因为上述 NumPy 数组的特色，让它运算时可以有较好的执行速度与需要较少的系统资源。

1-2 NumPy 的数据形态

NumPy 支持比 Python 更多的数据形态，下列是 NumPy 所定义的数据形态。

- bool_：和 Python 的 bool 兼容，以一个字节存储 True 或 False。
- int_：默认的整数形态，与 C 语言的 long 相同，通常是 int32 或 int64。
- intc：与 C 语言的 int 相同，通常是 int32 或 int64。
- intp：用于索引的整数，与 C 语言的 size_t 相同，通常是 int32 或 int64。
- int8：8 位整数 (−128 ~ 127)。
- int16：16 位整数 (−32768 ~ 32767)。
- int32：32 位整数 (−2147483648 ~ 2147483647)。
- int64：64 位整数 (−9223372036854775808 ~ 9223372036854775807)。
- uint8：8 位无符号整数 (0 ~ 255)。
- uint16：16 位无符号整数 (0 ~ 65535)。
- uint32：32 位无符号整数 (0 ~ 4294967295)。
- uint64：64 位无符号整数 (0 ~ 18446744073709551615)。

- float_：与 Python 的 float 相同。
- float16：半精度浮点数，符号位，5 位指数，10 位尾数。
- float32：单精度浮点数，符号位，8 位指数，23 位尾数。
- float64：双倍精度浮点数，符号位，11 位指数，52 位尾数。
- complex_：复数，complex_128 的缩写。
- complex64：复数，由 2 个 32 位浮点数表示（实部和虚部）。
- complex128：复数，由 2 个 64 位浮点数表示（实部和虚部）。

1-3 使用 array( ) 函数建立一维或多维数组

1-3-1 认识 ndarray 的属性

当使用 NumPy 模块建立 ndarray 数据形态的数组后，可以使用下列方式获得 ndarray 的属性，下列是几个常用的属性。

`ndarray.dtype`：数组元素形态。

`ndarray.itemsize`：数组元素数据形态大小（或称所占空间），单位为字节。

`ndarray.ndim`：数组的维度。

`ndarray.shape`：数组维度元素个数，数据形态是元组，也可以用于调整数组大小。

`ndarray.size`：数组元素个数。

1-3-2 使用 array( ) 函数建立一维数组

我们可以使用 array( ) 函数建立一维数组，array( ) 函数的语法如下：

```
numpy.array(object, dtype=None, ndmin)
```

上述参数意义如下：

- object：数组数据。
- dtype：数据类型，如果省略，将使用可以容纳数据最省的类型。
- ndmin：建立数组维度。

建立时在小括号内填上中括号，然后将数组数值放在中括号内，彼此用逗号隔开。

实例 1：建立一维数组，数组内容是 1, 2, 3，同时列出数组的数据形态。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> print(type(x))
<class 'numpy.ndarray'>
>>> print(x)
[1 2 3]
```

输出 x 数据类型

输出 x 数组内容

左侧所建立的一维数组图形如下：

x[0]	1
x[1]	2
x[2]	3

数组建立好后，可以用索引方式取得或设定内容。

实例 2：列出数组元素内容。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> print(x[0])
1
>>> print(x[1])
2
>>> print(x[2])
3
```

实例 3：设定数组内容。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x[1] = 10
>>> print(x)
[ 1 10  3]
```

实例 4：认识 ndarray 的属性。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x.dtype          ← 输出 x 数组元素形态
dtype('int32')
>>> x.itemsize       ← 输出 x 数组元素大小
4
>>> x.ndim           ← 输出 x 数组维度
1
>>> x.shape          ← 输出 x 数组外形, 3是第1维元素个数
(3,)
>>> x.size           ← 输出 x 数组元素个数
3
```

上述 x.dtype 获得 int32，表示是 32 位的整数。x.itemsize 是数组元素大小，其中以字节为单位，1 字节是 8 位，由于元素是 32 位整数，所以回传是 4。x.ndim 回传数组维度是 1，表示这是一维数组。x.shape 以元组方式回传第一维元素的个数是 3，未来二维数组还会解说。x.size 则是回传元素个数。

实例 5：array( ) 函数也可以接受使用 dtype 参数设定元素的数据形态。

```
>>> import numpy as np
>>> x = np.array([2, 4, 6], dtype=np.int8)
>>> x.dtype
dtype('int8')
```

因为上述元素是 8 位整数，所以执行 x.itemsize，所得的结果是 1。

```
>>> x.itemsize
1
```

实例 6：浮点数数组的建立与打印。

```
>>> import numpy as np
>>> y = np.array([1.1, 2.3, 3.6])
>>> y.dtype
dtype('float64')
>>> y
array([1.1, 2.3, 3.6])
>>> print(y)
[1.1 2.3 3.6]
```

上述所建立的一维数组图形如下：

x[0]	1.1
x[1]	2.3
x[2]	3.6

1-3-3 使用 array( ) 函数建立多维数组

在使用 array( ) 函数建立数组时，如果设定参数 ndmin，就可以建立多维数组。

程序实例 ch1_1.py：建立二维数组。

```
1  # ch1_1.py
2  import numpy as np
3
4  row1 = [1, 2, 3]
5  arr1 = np.array(row1, ndmin=2)
6  print(f"数组维度 = {arr1.ndim}")
7  print(f"数组外形 = {arr1.shape}")
8  print(f"数组大小 = {arr1.size}")
9  print("数组内容")
10 print(arr1)
11 print("-"*70)
12 row2 = [4, 5, 6]
13 arr2 = np.array([row1,row2], ndmin=2)
14 print(f"数组维度 = {arr2.ndim}")
15 print(f"数组外形 = {arr2.shape}")
16 print(f"数组大小 = {arr2.size}")
17 print("数组内容")
18 print(arr2)
```

执行结果

```
========== RESTART: D:\matplotlib\ch1\ch1_1.py ==========
数组维度 = 2
数组外形 = (1, 3)
数组大小 = 3
数组内容
[[1 2 3]]
----------------------------------------------------------------------
数组维度 = 2
数组外形 = (2, 3)
数组大小 = 6
数组内容
[[1 2 3]
 [4 5 6]]
>>>
```

程序实例 ch1_2.py：采用另一种设定二维数组的方式重新设计程序实例 ch1_1.py。

```
1  # ch1_2.py
2  import numpy as np
3
4  x = np.array([[1, 2, 3], [4, 5, 6]])
5  print(f"数组维度 = {x.ndim}")
6  print(f"数组外形 = {x.shape}")
7  print(f"数组大小 = {x.size}")
8  print("数组内容")
9  print(x)
```

执行结果

```
========== RESTART: D:\matplotlib\ch1\ch1_2.py =
数组维度 = 2
数组外形 = (2, 3)
数组大小 = 6
数组内容
[[1 2 3]
 [4 5 6]]
```

上述所建立的二维数组与二维数组索引的图形如下：

1	2	3
4	5	6

二维数组内容

x[0][0]	x[0][1]	x[0][2]
x[1][0]	x[1][1]	x[1][2]

二维数组索引

也可以用 x[0, 2] 代表 x[0][2]，可以参考下列实例，未来在实际应用中常采用 x[0, 2] 表达方式。

程序实例 ch1_3.py：认识引用二维数组索引的方式。

```
1  # ch1_3.py
2  import numpy as np
3
4  x = np.array([[1, 2, 3], [4, 5, 6]])
5  print(x[0][2])
6  print(x[1][2])
7  # 或是
8  print(x[0, 2])
9  print(x[1, 2])
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_3.py
3
6
3
6
```

上述第 5 行与第 8 行意义相同，读者可以了解引用索引方式。

1-4 使用 zeros( ) 函数建立内容是 0 的多维数组

zeros( ) 函数可以建立内容是 0 的数组，语法如下：

```
np.zeros(shape, dtype=float)
```

上述参数意义如下：

- shape：数组外形。
- dtype：默认是浮点数数据类型，也可以用此设定数据类型。

程序实例 ch1_4.py：分别建立 1×3 一维和 2×3 二维外形的数组，一维数组元素数据类型是浮点数 (float)，二维数组元素数据类型是 8 位无符号整数 (unit8)。

```
1  # ch1_4.py
2  import numpy as np
3
4  x1 = np.zeros(3)
5  print(x1)
6  print("-"*70)
7  x2 = np.zeros((2, 3), dtype=np.uint8)
8  print(x2)
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_4.py
[0. 0. 0.]
----------------------------------------------------------------------
[[0 0 0]
 [0 0 0]]
```

1-5 使用 ones( ) 函数建立内容是 1 的多维数组

ones( ) 函数可以建立内容是 1 的数组，语法如下：

```
np.ones(shape, dtype=None)
```

上述参数意义如下：

- shape：数组外形。
- dtype：默认是 64 浮点数数据类型 (float64)，也可以用此设定数据类型。

程序实例 ch1_5.py：分别建立 1×3 一维和 2×3 二维外形的数组，一维数组元素数据类型是浮点数 (float)，二维数组元素数据类型是 8 位无符号整数 (unit8)。

```
1  # ch1_5.py
2  import numpy as np
3
4  x1 = np.ones(3)
5  print(x1)
6  print("-"*70)
7  x2 = np.ones((2, 3), dtype=np.uint8)
8  print(x2)
```

执行结果

```
====================== RESTART: D:/matplotlib/ch1/ch1_5.py
[1. 1. 1.]
----------------------------------------------------------------------
[[1 1 1]
 [1 1 1]]
```

1-6 使用 random.randint( ) 函数建立随机数数组

random.randint( ) 函数可以建立均匀分布随机数内容的数组，语法如下：

```
np.random.randint(low, high=None, size=None, dtype=int)
```

上述参数意义如下：

- low：随机数的最小值 (含此值)。
- high：这是选项，如果有此参数，代表随机数的最大值 (不含此值)；如果不含此参数，则随机数是 0 ~ low。
- size：这是选项，数组的维数。
- dtype：默认是整数数据类型 (int)，也可以用此设定数据类型。

程序实例 ch1_6.py：分别建立单一随机数、含 10 个元素数组的随机数、3×5 的二维数组的随机数。

```
1  # ch1_6.py
2  import numpy as np
3
4  x1 = np.random.randint(10, 20)
5  print("回传值是10(含)至20(不含)的单一随机数")
6  print(x1)
7  print("-"*70)
8  print("回传一维数组10个元素，值是1(含)至5(不含)的随机数")
9  x2 = np.random.randint(1, 5, 10)
10 print(x2)
11 print("-"*70)
12 print("回传单3*5数组，值是0(含)至10(不含)的随机数")
13 x3 = np.random.randint(10, size=(3, 5))
14 print(x3)
```

执行结果

```
==================== RESTART: D:\matplotlib\ch1\ch1_6.py ====================
回传值是10(含)至20(不含)的单一随机数
16
----------------------------------------------------------------------
回传一维数组10个元素，值是1(含)至5(不含)的随机数
[4 4 3 4 2 1 3 4 4 4]
----------------------------------------------------------------------
回传单3*5数组，值是0(含)至10(不含)的随机数
[[7 1 7 3 8]
 [1 0 2 0 7]
 [8 1 1 8 1]]
>>>
```

1-7 使用 arange( ) 函数建立数组数据

arange( ) 函数是建立数组数据的方法，此函数语法如下：

```
np.arange(start, stop, step)                    # start 和 step 可以省略
```

start 是起始值，如果省略，默认值是 0；stop 是结束值，但是所产生的数组不包含此值；step 是数组相邻元素的间距，如果省略，默认值是 1。

程序实例 ch1_7.py：建立连续数值 0 ~ 15 的一维数组。

```
1  # ch1_7.py
2  import numpy as np
3
4  x = np.arange(16)
5  print(x)
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_7.py =
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

程序实例 ch1_8.py：在 0 和 2(不含) 之间建立间距是 0.1 的一维数组。

```
1  # ch1_8.py
2  import numpy as np
3
4  x = np.arange(0,2,0.1)
5  print(x)
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_8.py ==========
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9]
```

1-8 使用 linspace( ) 函数建立数组

linspace( ) 函数可以建立指定区间均匀间隔的数字数组，语法如下：

```
np.linspace(start, end, num)
```

start 是起始值 (含)，如果省略，默认值是 0；end 是结束值 (含)；num 是区间的元素个数。

程序实例 ch1_9.py：在 0 和 2 之间建立 100 个点的数组。

```
1  # ch1_9.py
2  import numpy as np
3
4  x = np.linspace(0,2,100)
5  print(x)
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_9.py ==========
[0.         0.02020202 0.04040404 0.06060606 0.08080808 0.1010101
 0.12121212 0.14141414 0.16161616 0.18181818 0.2020202  0.22222222
 0.24242424 0.26262626 0.28282828 0.3030303  0.32323232 0.34343434
 0.36363636 0.38383838 0.4040404  0.42424242 0.44444444 0.46464646
 0.48484848 0.50505051 0.52525253 0.54545455 0.56565657 0.58585859
 0.60606061 0.62626263 0.64646465 0.66666667 0.68686869 0.70707071
 0.72727273 0.74747475 0.76767677 0.78787879 0.80808081 0.82828283
 0.84848485 0.86868687 0.88888889 0.90909091 0.92929293 0.94949495
 0.96969697 0.98989899 1.01010101 1.03030303 1.05050505 1.07070707
 1.09090909 1.11111111 1.13131313 1.15151515 1.17171717 1.19191919
 1.21212121 1.23232323 1.25252525 1.27272727 1.29292929 1.31313131
 1.33333333 1.35353535 1.37373737 1.39393939 1.41414141 1.43434343
 1.45454545 1.47474747 1.49494949 1.51515152 1.53535354 1.55555556
 1.57575758 1.5959596  1.61616162 1.63636364 1.65656566 1.67676768
 1.6969697  1.71717172 1.73737374 1.75757576 1.77777778 1.7979798
 1.81818182 1.83838384 1.85858586 1.87878788 1.8989899  1.91919192
 1.93939394 1.95959596 1.97979798 2.        ]
```

1-9 使用 reshape( ) 函数更改数组形式

reshape( ) 函数可以更改数组形式，语法如下：

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

上述 a 是要更改的数组，newshape 是新数组的外形，newshape 可以是整数或是元组。

程序实例 ch1_10.py：将 1×16 数组改为 2×8 数组。

```
1  # ch1_10.py
2  import numpy as np
3
4  x = np.arange(16)
5  print(x)
6  print(np.reshape(x,(2,8)))
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_10.py
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[[ 0  1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14 15]]
```

有时候，reshape( ) 函数的元组 newshape 的其中一个元素是 -1，这表示将依照另一个元素安排元素内容。

程序实例 ch1_11.py：重新设计程序实例 ch1_10.py，但是 newshape 元组的其中一个元素值是 -1，整个 newshape 内容是 (4, -1)。

```
1  # ch1_11.py
2  import numpy as np
3
4  x = np.arange(16)
5  print(x)
6  print(np.reshape(x,(4,-1)))
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_11.py
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

程序实例 ch1_12.py：重新设计程序实例 ch1_10.py，但是 newshape 元组的其中一个元素值是 -1，整个 newshape 内容是 (-1, 8)。

```
1  # ch1_12.py
2  import numpy as np
3
4  x = np.arange(16)
5  print(x)
6  print(np.reshape(x,(-1,8)))
```

执行结果

```
========== RESTART: D:/matplotlib/ch1/ch1_12.py
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[[ 0  1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14 15]]
```