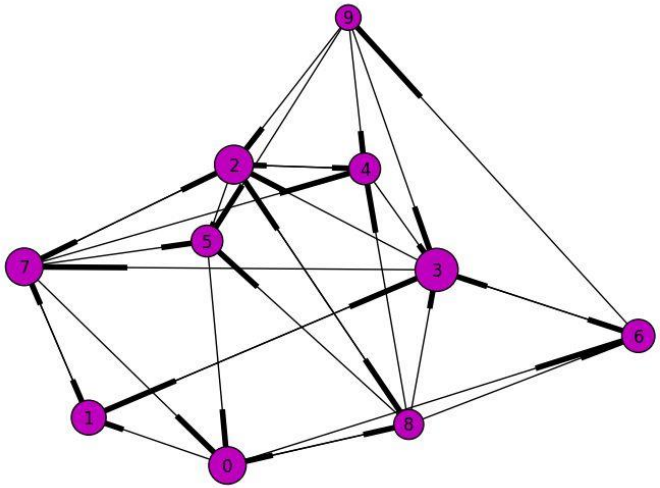
Python程序设计

陈远祥

chenyxmail@gmail.com

北京邮电大学 电子工程学院

Numpy



Numpy

- NumPy是Python语言的一个扩展包。它代表“Numeric Python”。它是一个由多维数组对象和用于处理数组的例程集合组成的库
- 支持多维数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。NumPy提供了与Matlab相似的功能与操作方式，因为两者皆为直译语言

Numpy的导入

- 标准的Python中用列表(list)保存一组值，可以当作数组使用。但由于列表的元素可以是任何对象，因此列表中保存的是对象的指针。对于数值运算来说，这种结构显然比较浪费内存和CPU计算
- Python提供了array模块，它和列表不同，能直接保存数值，但是由于它不支持多维数组，也没有各种运算函数，因此也不适合做数值运算

Numpy的导入

- NumPy的诞生弥补了这些不足，NumPy提供了两种基本的对象：ndarray（n-dimensional array object）和ufunc（universal function object）
- ndarray(数组)是存储单一数据类型的多维数组，而ufunc则是能够对数组进行处理的函数
- 函数库的导入：import numpy as np

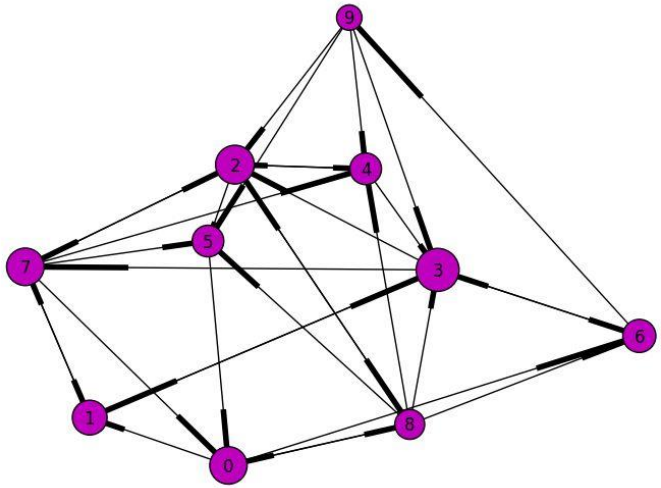
Numpy

- Numpy能做什么？
  - ✓ 数组的算数和逻辑运算
  - ✓ 傅立叶变换和用于图形操作的例程
  - ✓ 与线性代数有关的操作。 NumPy 拥有线性代数和随机数生成的内置函数

Numpy

- Numpy-MatLab的替代之一
  - ✓ NumPy 通 常 与 SciPy(Scientific Python) 和 Matplotlib(绘图库)一起使用，这种组合广泛用于替代Matlab，是一个流行的技术平台
  - ✓ 但是，Python作为MatLab的替代方案，现在被视为一种更加现代和完整的编程语言
  - ✓ NumPy是开源的，这是它的一个额外的优势

Numpy-Ndarray对象

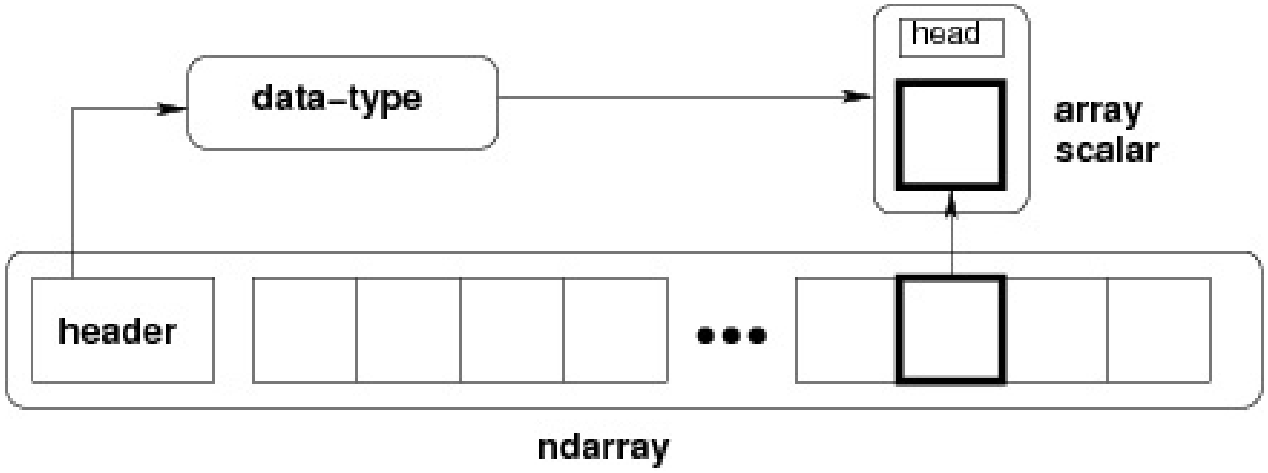


Ndarray

- NumPy中定义的最重要的对象是称为ndarray的N维数组类型。它描述相同类型的元素集合。可以使用基于零的索引访问集合中的项目
- ndarray中的每个元素在内存中使用相同大小的块
- ndarray中的每个元素是数据类型对象的对象（称为dtype）

Ndarray

- 从ndarray对象提取的任何元素（通过切片）由一个数组标量类型的 Python对象表示
- 下图为ndarray，数据类型对象（dtype）和数组标量类型之间的关系



Ndarray

- 创建数组：实例化
  - ✓ 基本的ndarray是使用 NumPy中的数组函数创建的，如下所示：

    numpy.array

Ndarray

- **numpy.array的参数**

  numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

序号	参数及描述
1.	object 任何暴露数组接口方法的对象都会返回一个数组或任何（嵌套）序列。
2.	dtype 数组的所需数据类型，可选。
3.	copy 可选，默认为true，对象是否被复制。
4.	order C（按行）、F（按列）或A（任意，默认）。
5.	subok 默认情况下，返回的数组被强制为基类数组。 如果为true，则返回子类。
6.	ndimin 指定返回数组的最小维数。

Ndarray

数据类型

- ✓ NumPy支持比Python更多种类的数值类型

序号	数据类型及描述
1.	bool_ 存储为一个字节的布尔值（真或假）
2.	int_ 默认整数，相当于 C 的long，通常为int32或int64
3.	intc 相当于 C 的int，通常为int32或int64
4.	intp 用于索引的整数，相当于 C 的size_t，通常为int32或int64
5.	int8 字节（-128 ~ 127）
6.	int16 16 位整数（-32768 ~ 32767）
7.	int32 32 位整数（-2147483648 ~ 2147483647）
8.	int64 64 位整数（-9223372036854775808 ~ 9223372036854775807）
9.	uint8 8 位无符号整数（0 ~ 255）
10.	uint16 16 位无符号整数（0 ~ 65535）
11.	uint32 32 位无符号整数（0 ~ 4294967295）
12.	uint64 64 位无符号整数（0 ~ 18446744073709551615）
13.	float_ float64的简写
14.	float16 半精度浮点：符号位，5 位指数，10 位尾数
15.	float32 单精度浮点：符号位，8 位指数，23 位尾数
16.	float64 双精度浮点：符号位，11 位指数，52 位尾数
17.	complex_ complex128的简写
18.	complex64 复数，由两个 32 位浮点表示（实部和虚部）
19.	complex128 复数，由两个 64 位浮点表示（实部和虚部）

数组属性

- 数组属性
  - ✓ 数组的大小：ndarray.shape
  - ✓ 数组的维度：ndarray.ndim
  - ✓ 数组的元素类型：c.dtype
  - ✓ #数组属性

数组创建

- 数组创建例程
  - ✓ 新的ndarray对象可以通过任何下列数组创建例程
  - ✓ numpy.empty
  - ✓ numpy.zeros
  - ✓ numpy.ones
  - ✓ #数组创建

数组创建

- 数组创建例程
  - ✓ 使用低级ndarray构造函数构造
  - ✓ numpy.asarray
  - ✓ numpy.frombuffer
  - ✓ numpy.fromiter
  - ✓ numpy.arrange
  - ✓ numpy.linspace
  - ✓ #数组创建

数组切片和索引

- ndarray对象的内容可以通过索引或切片来访问和修改，就像Python 的内置容器对象一样
- 包括：字段访问，基本切片和高级索引
  - ✓ slice

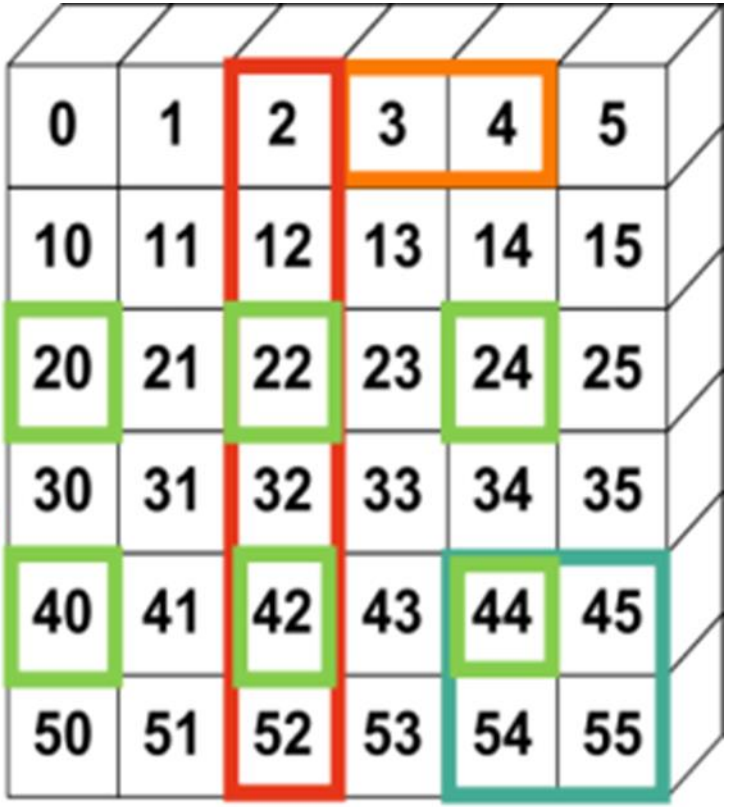
数组切片和索引

- 高级索引
  - ✓ 基于N维索引来获取数组中任意元素
  - ✓ 每个整数数组表示该维度的下标值

数组切片和索引

- 高级索引

```
>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,::2]
array([[20,22,24],
       [40,42,44]])
```



数组切片和索引

- 布尔索引
  - ✓ 当结果对象是布尔运算（例如比较运算符）的结果时，将使用此类型的高级索引
  - ✓ #布尔索引

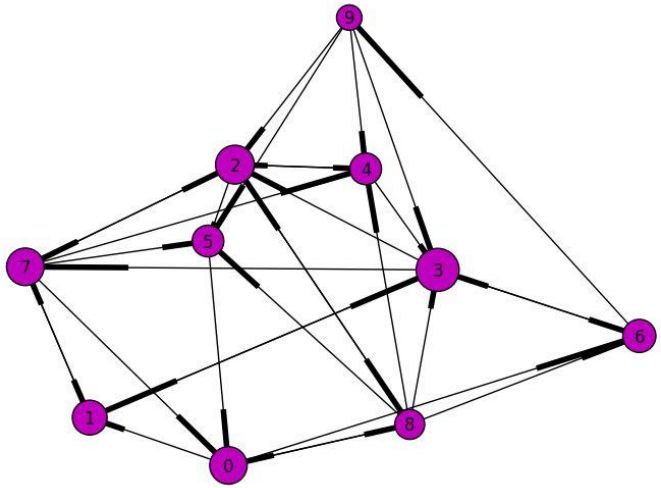
数组切片和索引

- Python在数据预处理的时候，经常遇到需要对空值进行处理的地方。空值在python中的表现一般为：
  - ✓ None
  - ✓ False
  - ✓ ‘’
  - ✓ nan(not a number,无理数)

数组切片和索引

- 如何对这些数字进行判断
  - ✓ 前3个很容易判断，直接==就可以了
  - ✓ nan无法用==进行判断
  - ✓ 通过nan ！= nan判断

Numpy-ufunc运算



ufunc

- ufunc是universal function的缩写，它是一种能对数组的每个元素进行操作的函数
- NumPy内置的许多ufunc函数都是在C语言级别实现的，因此它们的计算速度非常快
- 比较numpy.math和Python标准库的math.sin的计算速度：
  - ✓ #numpy_speed_test

ufunc

- 单个数值的numpy.sin的计算速度只有math.sin的1/4。这是因为numpy.sin为了同时支持数组和单个值的计算，其C语言的内部实现要比math.sin复杂很多
- 矩阵操作的numpy.sin比math.sin快20倍多。这得利于numpy.sin在C语言级别的循环计算

ufunc

- 四则运算：
  - ✓ np.add(a,b)
  - ✓ np.add(a,b,a)
  - ✓ np.subtract()
  - ✓ np.multiply()
  - ✓ np.divide()

ufunc

- 比较和布尔运算：
  - ✓ 使用“==”、“>”等比较运算符对两个数组进行比较，将返回一个布尔数组，它的每个元素值都是两个数组对应元素的比较结果。例如：

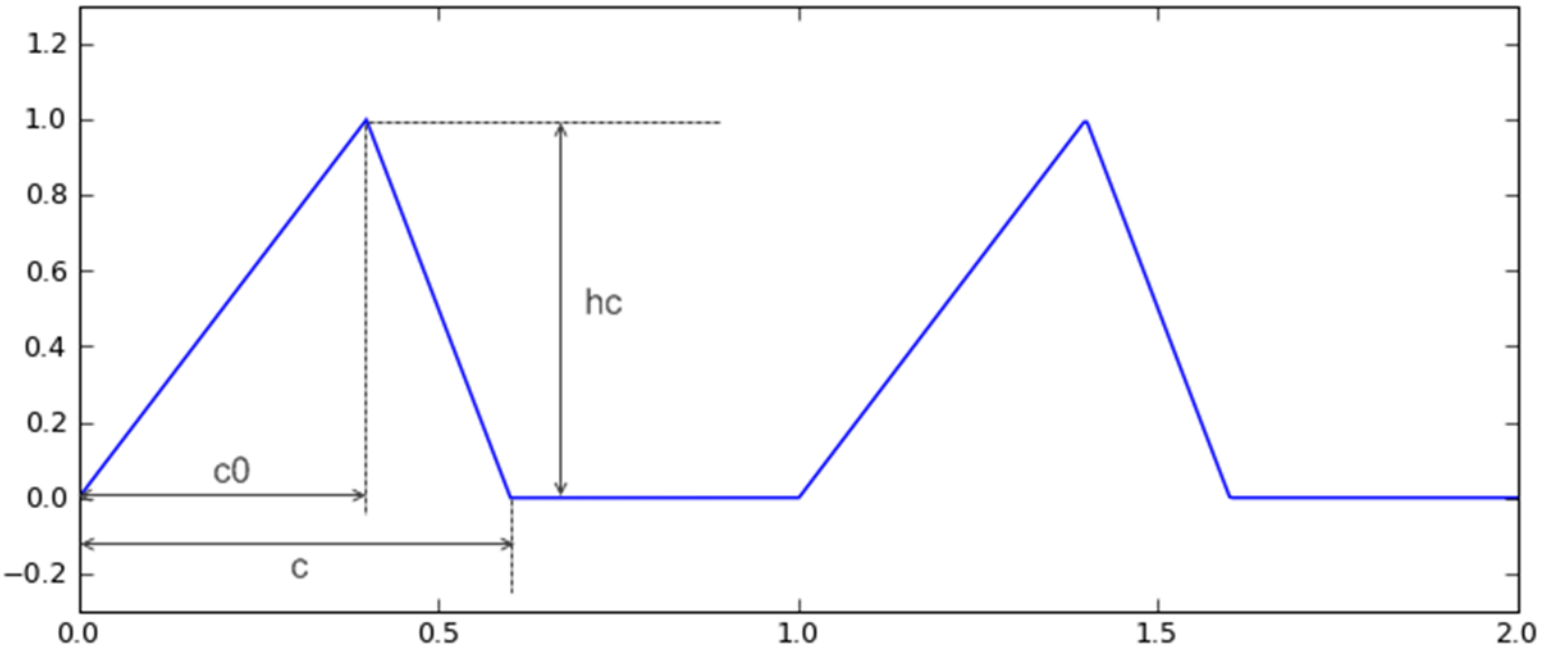
```
np.array([1,2,3]) < np.array([3,2,1])
array([ True, False, False], dtype=bool)
```

ufunc

- 自定义ufunc函数：
  - ✓ 通过组合标准的ufunc函数的调用，可以实现各种算式的数组计算
  - ✓ 不过有些时候这种算式不易编写，而针对每个元素的计算函数却很容易用Python实现，这时可以用frompyfunc函数将一个计算单个元素的函数转换成ufunc函数

ufunc

- 自定义ufunc函数：
  - ✓ 用一个分段函数描述三角波，三角波的样子如下：



ufunc

- 自定义ufunc函数：
  - ✓ 根据上图所示写出如下的计算三角波某点y坐标的函数：

    ```
    def triangle_wave(x, c, c0, hc)
    ```
  - ✓ 显然triangle_wave函数只能计算单个数值，不能对数组直接进行处理

ufunc

- 自定义ufunc函数：
  - ✓ 可以先使用列表计算出一个list，然后用array函数将列表转换为数组
  - ✓ #三角波1
  - ✓ 这种做法每次都需要使用列表调用函数，对于多维数组是很麻烦的。让我们来看看如何用frompyfunc函数来解决这个问题

ufunc

- 自定义ufunc函数：
  - ✓ 通过frompyfunc()可以将计算单个值的函数转换为一个能对数组的每个元素进行计算的ufunc函数。frompyfunc()的调用格式为：

    frompyfunc(func, nin, nout)
  - ✓ 其中func是计算单个元素的函数，nin是func的输入参数的个数，nout是func的返回值个数

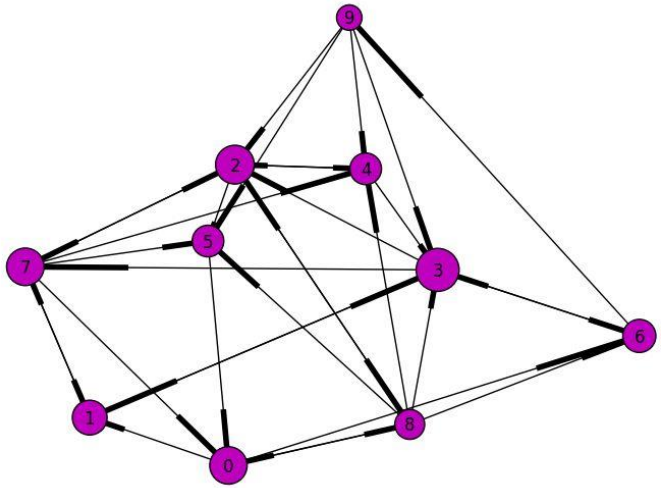
ufunc

- 自定义ufunc函数：
  - ✓ 使用frompyfunc()将triangle_wave()转换为一个ufunc函数对象triangle_ufunc1：
  - ✓ #三角波2

ufunc

- 自定义ufunc函数：
  - ✓ 虽然triangle_wave函数有4个参数，但是由于后三个c，c0，hc在整个计算中值都是固定的，因此所产生的ufunc函数其实只有一个参数。为了满足这个条件，可用一个lambda函数对triangle_wave的参数进行一次包装。这样传入frompyfunc的函数就只有一个参数了
  - ✓ #三角波3

Numpy-广播



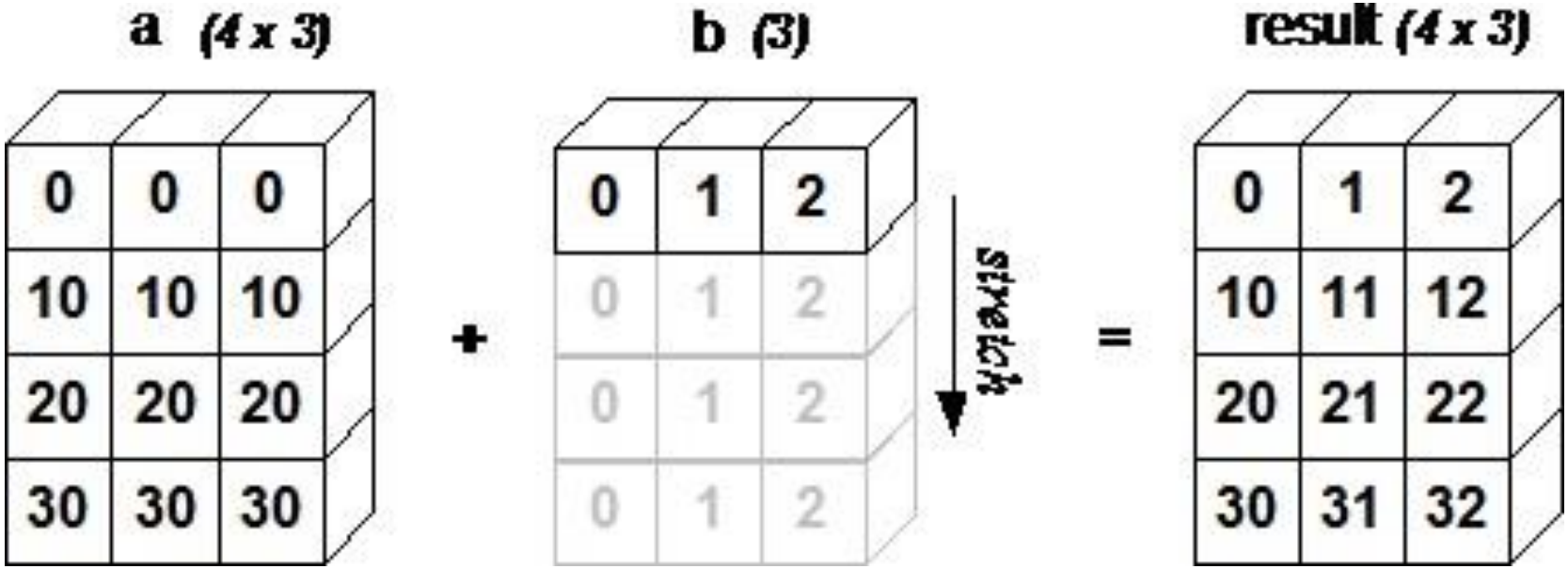
广播

- 当我们使用ufunc函数对两个数组进行计算时，ufunc函数会对这两个数组的对应元素进行计算，因此它要求这两个数组有相同的大小(shape相同)。如果两个数组的shape不同的话，会进行如下的广播(broadcasting)处理

广播

- 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
- 输出数组的shape是输入数组shape的各个轴上的最大值
- 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
- 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

广播


a (4 x 3)
b (3)
result (4 x 3)
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
stretch
=
0 1 2
10 11 12
20 21 22
30 31 32

广播

- 流程：
  - ✓ 由于a和b的shape长度不同，根据规则1，需要让b的shape向a对齐，于是将b的shape前面加1，补齐为(1,5)。相当于做了如下计算：
    b.shape=1,5
  - ✓ 这样加法运算的两个输入数组的shape分别为(6,1)和(1,5)，根据规则2，输出数组的各个轴的长度为输入数组各个轴上的长度的最大值，可知输出数组的shape为(6,5)

广播

- 流程：
  - ✓ 由于b的第0轴上的长度为1，而a的第0轴上的长度为6，因此为了让它们在第0轴上能够相加，需要将b在第0轴上的长度扩展为6，这相当于：
  - ✓ `b=b.repeat(6,axis=0)`

广播

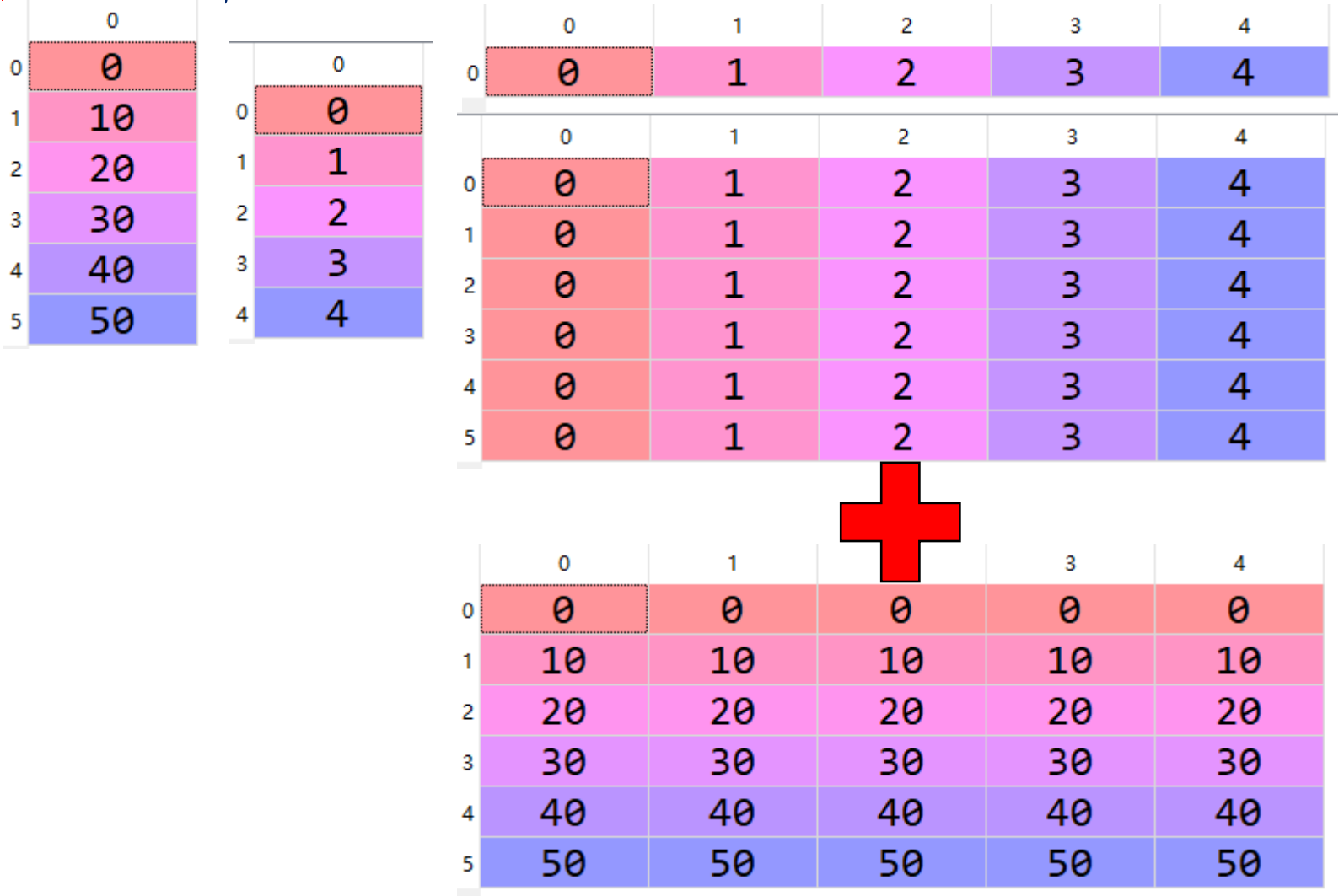
- 流程：
  - ✓ 由于a的第1轴的长度为1，而b的第一轴长度为5，因此为了让它们在第1轴上能够相加，需要将a在第1轴上的长度扩展为5，这相当于：
  - ✓ `a = a.repeat(5, axis=1)`

广播

- 流程：
  - ✓ 经过上述处理之后，a和b就可以按对应元素进行相加运算了

广播

- 流程：


0
0
10
20
30
40
50
0
0
1
2
3
4
0 1 2 3 4
0 0 1 2 3 4
0 0 1 2 3 4
1 0 1 2 3 4
2 0 1 2 3 4
3 0 1 2 3 4
4 0 1 2 3 4
5 0 1 2 3 4
0 1 3 4
0 0 0 0 0 0
1 10 10 10 10 10
2 20 20 20 20 20
3 30 30 30 30 30
4 40 40 40 40 40
5 50 50 50 50 50

广播

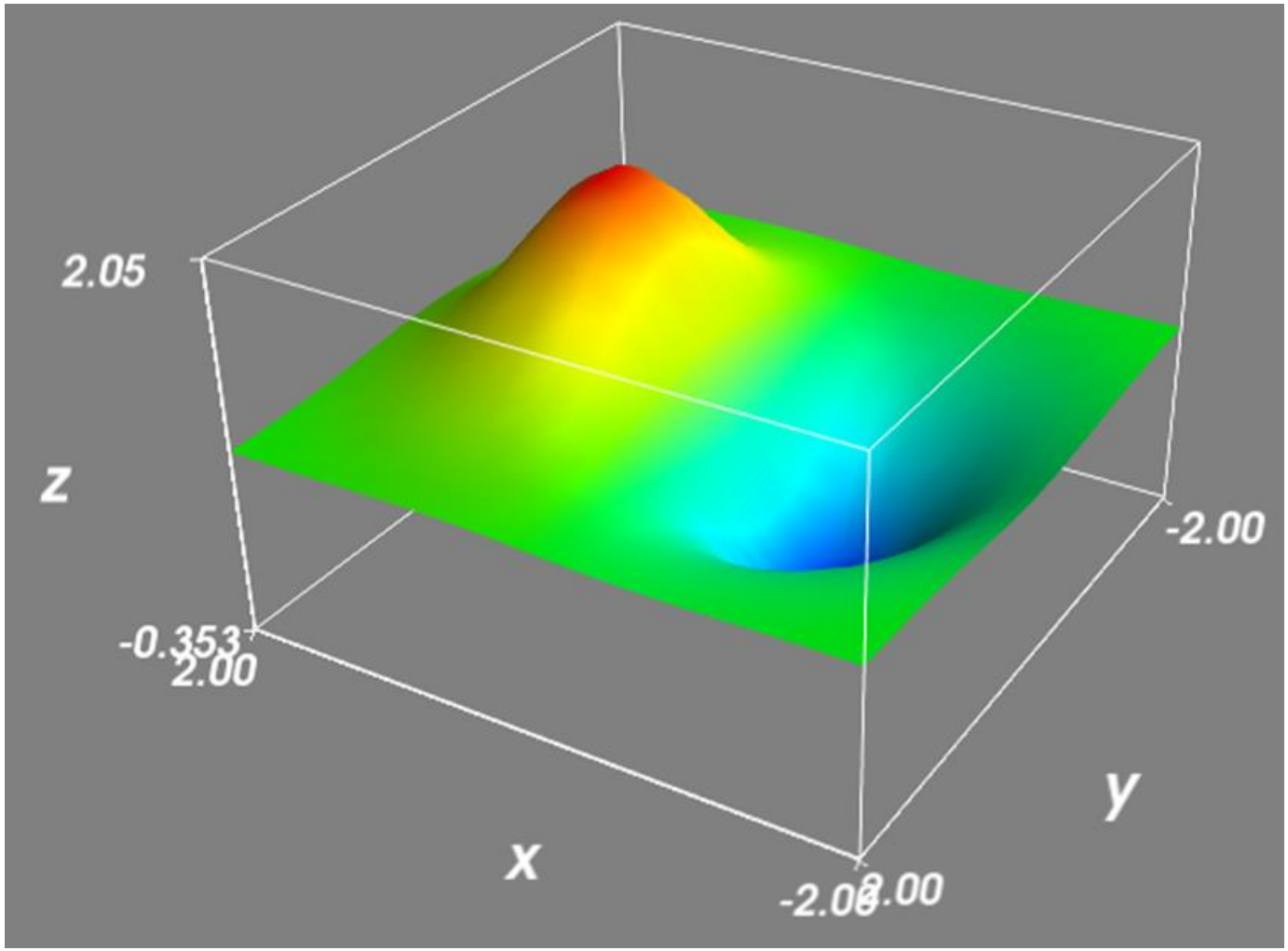
- numpy在执行a+b运算时，其内部并不会真正将长度为1的轴用repeat函数进行扩展，如果这样做的话就太浪费空间了。由于这种广播计算很常用，因此numpy提供了一个快速产生如上面a,b数组的方法： ogrid对象：

```
x,y = np.ogrid[0:5,0:5]
```

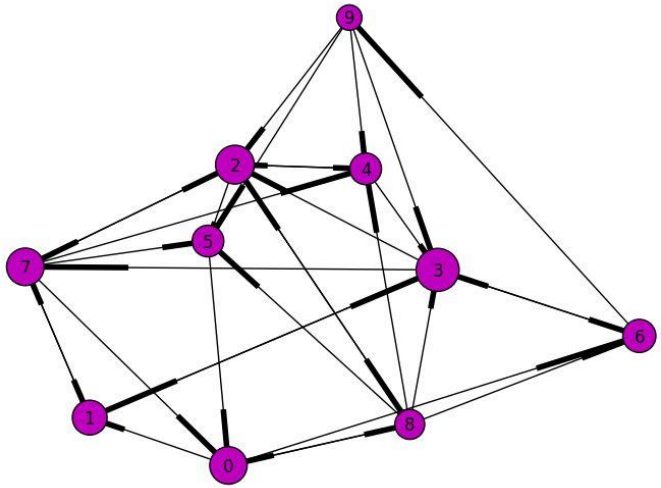
广播

- 利用ogrid的返回值，能很容易计算x，y网格面上各点的值，或者x，y，z网格体上各点的值。下面是绘制三维曲面x * exp(x**2 - y**2) 的程序：(numpy_orid_mlab.py)

```
1 import numpy as np
2 from mayavi import mlab
3 x, y = np.ogrid[-2:2:20, -2:2:20]
4 z = x * np.exp( - x**2 - y**2)
5
6 pl = mlab.surf(x,y,z, warp_scale="auto")
7 mlab.axes(xlabel='x', ylabel='y', zlabel='z')
8 mlab.outline(pl)
9 mlab.show
```


2.05
z
-0.353
2.00
x
-2.00
2.00
y
-2.00

Numpy-矩阵运算



矩阵运算

- NumPy和Matlab不一样，对于多维数组的运算，缺省情况下并不使用矩阵运算，如果你希望对数组进行矩阵运算的话，可以调用相应的函数

矩阵运算

- matrix对象
  - ✓ numpy库提供了matrix类，使用matrix类创建的是矩阵对象，它们的加减乘除运算缺省采用矩阵方式计算，因此用法和matlab十分类似。但是由于NumPy中同时存在ndarray和matrix对象，因此用户很容易将两者弄混。这有违Python的“显式优于隐式”的原则，因此并不推荐在较复杂的程序中使用matrix

矩阵运算

- matrix对象
  - ✓ numpy库提供了matrix类，使用matrix类创建的是矩阵对象，它们的加减乘除运算缺省采用矩阵方式计算，因此用法和matlab十分类似。但是由于NumPy中同时存在ndarray和matrix对象，因此用户很容易将两者弄混。这有违Python的“显式优于隐式”的原则，因此并不推荐在较复杂的程序中使用matrix

矩阵运算

- 矩阵的乘积可以使用dot函数进行计算。对于二维数组，它计算的是矩阵乘积，对于一维数组，它计算的是其点积。当需要将一维数组当作列矢量或者行矢量进行矩阵运算时，推荐先使用reshape函数将一维数组转换为二维数组：
  - ✓ #矩阵运算

矩阵运算

- dot:对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和(数学上称之为内积)；对于二维数组，计算的是两个数组的矩阵乘积；对于多维数组，它的通用计算公式如下：数组a的最后一维上的所有元素与数组b的倒数第二维上的所有元素的乘积和：

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

矩阵运算

- dot乘积的结果c可以看作是数组a,b的多个子矩阵的乘积：

```
>>> np.alltrue( c[0,:,0,:] == np.dot(a[0],b[0]) )
True
>>> np.alltrue( c[1,:,0,:] == np.dot(a[1],b[0]) )
True
>>> np.alltrue( c[0,:,1,:] == np.dot(a[0],b[1]) )
True
>>> np.alltrue( c[1,:,1,:] == np.dot(a[1],b[1]) )
True
```

矩阵运算

- inner：和dot乘积一样，对于两个一维数组，计算的是这两个数组对应下标元素的乘积和；对于多维数组，它计算的结果数组中的每个元素都是：数组a和b的最后一维的内积，因此数组a和b的最后一维的长度必须相同

```
inner(a, b)[i,j,k,m] = sum(a[i,j,:]*b[k,m,:])
```

矩阵运算

```
>>> a = np.arange(12).reshape(2,3,2)
>>> b = np.arange(12,24).reshape(2,3,2)
>>> c = np.inner(a,b)
>>> c.shape
(2, 3, 2, 3)
>>> c[0,0,0,0] == np.inner(a[0,0],b[0,0])
True
>>> c[0,1,1,0] == np.inner(a[0,1],b[1,0])
True
>>> c[1,2,1,2] == np.inner(a[1,2],b[1,2])
True
```

矩阵运算

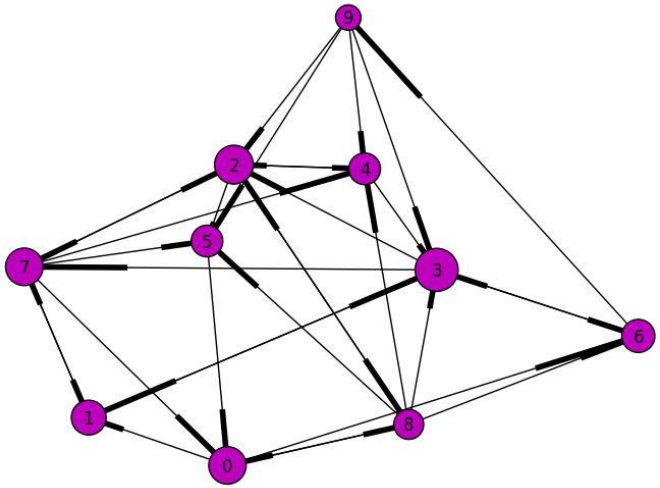
- outer：只按照一维数组进行计算，如果传入参数是多维数组，则先将此数组展平为一维数组之后再进行运算。outer乘积计算的列向量和行向量的矩阵乘积：

```
>>> np.outer([1,2,3],[4,5,6,7])
array([[ 4, 5, 6, 7],
[ 8, 10, 12, 14],
[12, 15, 18, 21]])
```

矩阵运算

```
>>> a = np.arange(12).reshape(2,3,2)
>>> b = np.arange(12,24).reshape(2,3,2)
>>> c = np.inner(a,b)
>>> c.shape
(2, 3, 2, 3)
>>> c[0,0,0,0] == np.inner(a[0,0],b[0,0])
True
>>> c[0,1,1,0] == np.inner(a[0,1],b[1,0])
True
>>> c[1,2,1,2] == np.inner(a[1,2],b[1,2])
True
```

Numpy-函数库



函数库

- 除了前面介绍的ndarray数组对象和ufunc函数之外，NumPy还提供了大量对数组进行处理的函数。充分利用这些函数，能够简化程序的逻辑，提高运算速度

函数库

- sum()计算数组元素之和，也可以对列表、元组等和数组类似的序列进行求和。当数组是多维时，它计算数组中所有元素的和：
  - ✓ #函数库

函数库

- mean()用于求数组的平均值，也可以通过axis参数指定求平均值的轴，通过out参数指定输出数组。和sum()不同的是，对于整数数组，它使用双精度浮点数进行计算，而对于其他类型的数组，则使用和数组元素类型相同的累加变量进行计算

函数库

- 用min()和max()可以计算数组的最大值和最小值，而ptp()计算最大值和最小值之间的差。 它们都有axis和out两个参数。这些参数的用法和sum()相同

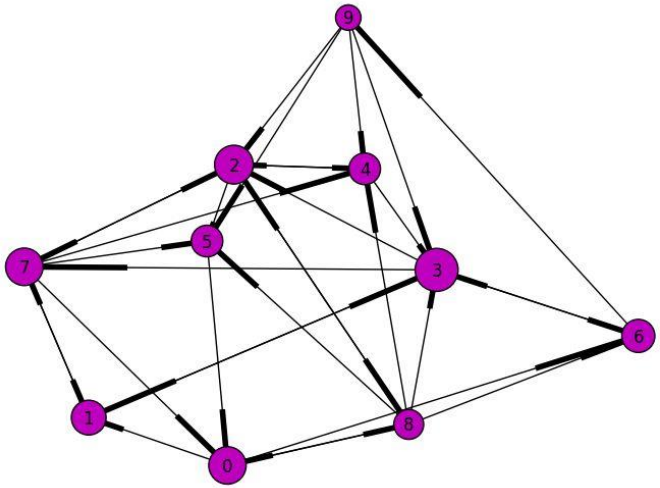
函数库

- 数组的sort()方法用于对数组进行排序，它将改变数组的内容。而sort()函数则返回一个新数组，不改变原始数组。它们的axis参数默认值都为-1，即沿着数组的最后一个轴进行排序。sort()函数的axis参数可以设置为None，此时它将得到平坦化之后进行排序的新数组

函数库

- argsort()返回数组的排序下标，axis参数的默认值为-1
- 用median()可以获得数组的中值，即对数组进行排序之后，位于数组中间位置的值

NumPy模块



NumPy模块

- numpy.linalg模块
- numpy.fft模块
- numpy.random模块
- 直方图(histogram)

NumPy模块

- numpy.linalg模块
  - ✓ numpy.linalg模块包含线性代数的函数。使用这个模块，可以计算逆矩阵、求特征值、奇异值分解、解线性方程组以及求解行列式、秩等

NumPy模块

- numpy.fft模块
  - ✓ 在NumPy中，有一个名为fft的模块提供了快速傅里叶变换的功能。在这个模块中，许多函数都是成对存在的，也就是说许多函数存在对应的逆操作函数

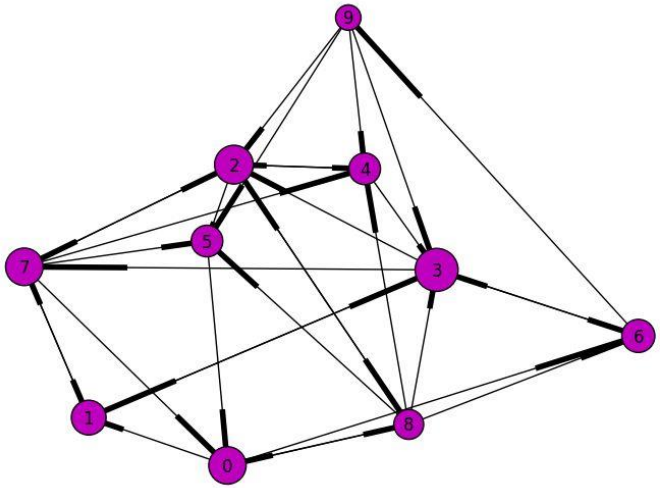
NumPy模块

- numpy.random模块
  - ✓ 随机数的函数可以在NumPy的random模块中找到
  - ✓ 二项分布
  - ✓ 超几何分布
  - ✓ 连续分布
  - ✓ 正态分布
  - ✓ 。。。

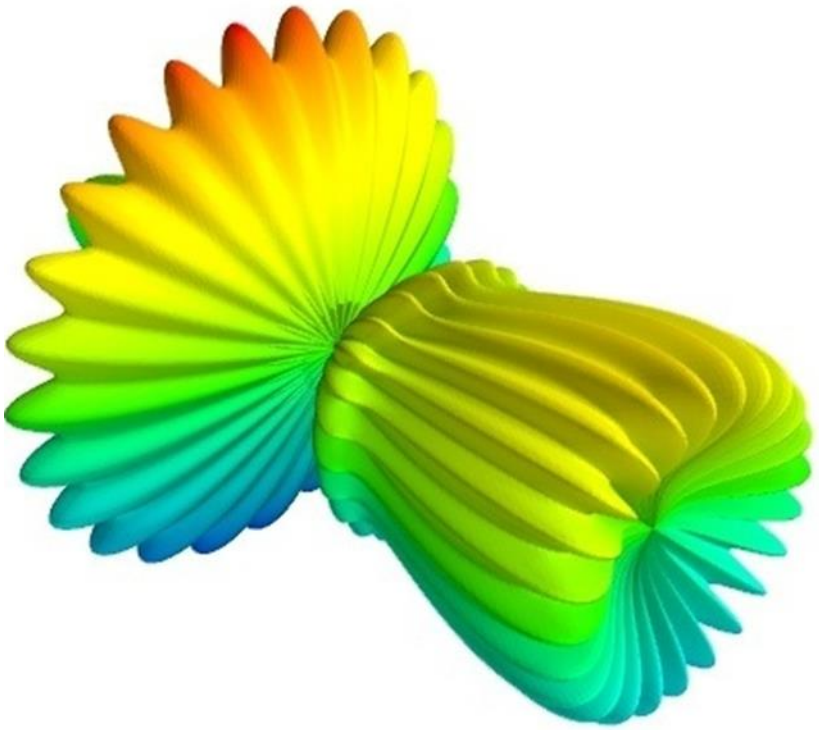
NumPy模块

- 直方图(histogram)
  - ✓ NumPy中histogram函数应用到一个数组返回一对变量：直方图数组和箱式向量

本周作业



- 画图：
  - ✓ 1、安装mayavi库
  - ✓ 2、画出下图



谢谢