PYTHON数据科学系列课程（二）
NDARRAY对象的运算和统计

东南大学 学习科学研究中心 儿童发展与教育研究所
夏小俊
http://www.seuct.com

内容提要

2.1 ndarray对象的属性
2.2 形状和重构
2.3 数学运算和广播
2.4 切割、堆叠和操纵
2.5 排序
2.6 数组上的迭代

内容提要

2.7 字符串处理
2.8 随机模块
2.9 统计函数
2.10 引用、视图与复制
2.11 矩阵库与线性代数
2.12 文件IO

2.1 NDARRAY对象的属性

In[1]:

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr1.ndim, arr2.ndim) #ndim代表数组维度
```

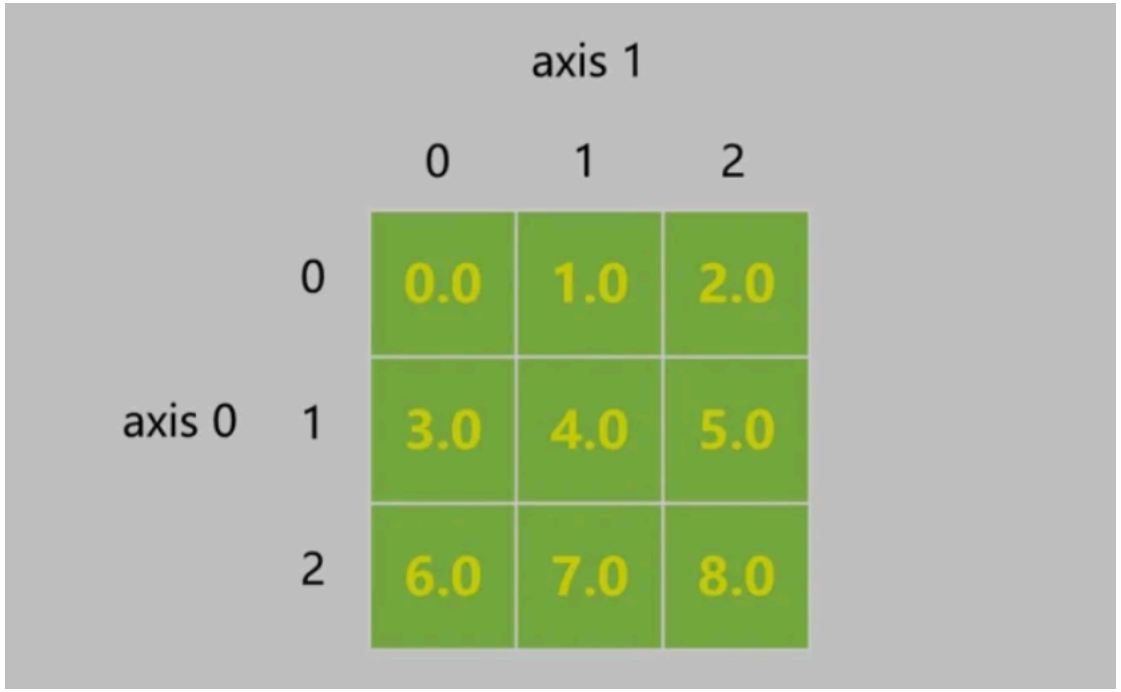
Out[1]: **1 2**

In[2]:

```
print(arr1.shape)
print(arr2.shape)      #shape返回包含各维度大小的元组
```

Out[2]: **(3,)**
(2, 3)

2.1 NDARRAY对象的属性



axis（轴）的概念：轴用来为超过一维的数组定义的属性，二维数据拥有两个轴：第0轴沿着行的垂直往下，第1轴沿着列的方向水平延伸。

在运算时，可以理解为：设axis=i ,则沿着第i个下标变化的方向进行操作。

2.1 NDARRAY对象的属性

In[3]:

```
print(arr1.size)
print(arr2.itemsize)    #和dtype对应，这里是int64
```

Out[3]:

```
3
8
```

In[4]:

```
print(arr1.flags)    #flags返回数组内存信息的描述
```

Out[4]:

```
C_CONTIGUOUS : True #C风格的数组保存方法
...
```

注：类似shape、ndim等属性或者方法还有一种用法如np.ndim(arr)，但不常见

2.1 NDARRAY对象的属性

In[5]:

```
print(arr2.strides)
arr2
```

Out[5]:

```
(24, 8)
array([[1, 2, 3],
       [4, 5, 6]])
```

strides属性记录了从当前维度到下一维度所需要“跨过”的字节数。

arr2[0,0]到arr2[1,0]需要24字节，到arr2[0,1]需要8字节。

2.2 形状与重构

In[6]:

```
arr3 = arr2.reshape(3, 2)  #参数为-1，可自动推算
arr3
```

Out[6]:

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In[7]:

```
print(arr2.shape)   #reshape
```

Out[7]: (2, 3)

2.2 形状与重构

reshape函数在实际应用中很常见，两个特别需要注意的点

1. 新数组和原数组的形状要一致，否则会报错。

2. 得到的新数组和原数组共享同一片内存。

2.2 形状与重构

In[8]:

```
arr2.reshape(4, 2)
```

Out[8]:

```
ValueError: cannot reshape array of size 6 into
shape (4,2)
```

In[9]:

```
arr2[0][0] = -1
arr3
```

Out[9]:

```
array([[-1,  2],
       [ 3,  4],
       [ 5,  6]])
```

2.2 形状与重构

In[10]:
```
arr2.resize(3, 2) #就地修改
arr2
```

Out[10]:
```
array([[-1,  2],
       [ 3,  4],
       [ 5,  6]])
```

In[11]:
```
arr4 = np.arange(1, 7)
arr4.resize(2, 4) # 数量不足会自动填充，多余会舍弃
arr4
```

Out[11]:
```
array([[1, 2, 3, 4],
       [5, 6, 0, 0]])
```

2.2 形状与重构

In[12]:

```
arr4.swapaxes(0,1) # 轴交换
```

Out[12]:

```
array([[1, 5],
       [2, 6],
       [3, 0],
       [4, 0]])
```

In[13]:

```
arr5 = np.arange(1, 9).reshape(2,2,2)
arr5.swapaxes(2,0) #仍然共享同一片区域，存储细节变化
```

Out[13]:

```
array([[[1, 5],
        [3, 7]],
       [[2, 6],
        [4, 8]]])
```

2.2 形状与重构

In[14]:

```
a = np.arange(6).reshape(2,3)
b = a.T #b = np.transpose(a)
b
```

Out[14]:

```
array([[0, 3],
       [1, 4],
       [2, 5]])
```

2.2 形状与重构

In[15]:

```
a = np.arange(24).reshape(2,3,4)
b = np.rollaxis(a,2)   #下标顺序由012改成201
np.where(b == 23) #b.shape
```

Out[15]:

```
(array([3]), array([1]), array([2]))
(4, 2, 3)
```

2.2 形状与重构

In[16]:

```
a = np.arange(8).reshape(2,2,2)
b = np.rollaxis(a,2)   #下标顺序由012改成201
print(a)
print(b)
print(a.strides, b.strides) #a和b的内存空间仍然一样
```

Out[16]:

```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]

(32, 16, 8)
```

```
[[[0 2]
  [4 6]]

 [[1 3]
  [5 7]]]

(8, 32, 16)
```

2.2 形状与重构

In[17]:
```
arr4.flatten()    #折叠为一维数组
```

Out[17]:
```
array([1, 2, 3, 4, 5, 6, 0, 0])
```

In[18]:
```
arr4.tolist() #转换为原生列表
```

Out[18]:
```
[[1, 2, 3, 4], [5, 6, 0, 0]]
```

2.3 数学运算与广播

In[1]:

```
import numpy as np
arr1 = np.arange(1, 10).reshape(3, 3)
arr2 = np.array([10, 10, 10])
arr1 + arr2
```

Out[1]:

```
array([[11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])
```

注：这里的算术运算是广播对齐后，数组同一对应位置的元素运算。

2.3 数学运算与广播

In[2]:

```
arr3 = np.arange(10).reshape(2, 5)
arr4 = np.arange(16).reshape(4, 4)
arr3 + arr4
```

Out[2]:

```
ValueError: operands could not be broadcast

together with shapes (2,5) (4,4)
```

2.3 数学运算与广播

In[3]:

```
arr1 * 10
```

Out[3]:

```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

2.3 数学运算与广播

除了支持运算符之外，np本身还提供了大量的数学函数，如arr1 + arr2可以写成np.add(arr1, arr2)。

二元函数名	功能	二元函数名	功能
add	加法	power	幂运算
subtract	减法	fmax	更大值
multiply	乘法	fmin	更小值
divide	除法	greater, greater_equal	大于，大于等于
less, less_equal	小于，小于等于	equal,not_equal	相等，不等
logical_and	逻辑与	logical_or	逻辑或
logical_xor	逻辑异或		

2.3 数学运算与广播

In[4]:

```
np.add(arr1, arr2)
np.subtract(arr1, arr2)
np.multiply(arr1, arr2)
np.divide(arr1, arr2)
np.power(arr1, arr2)
np.fmax(arr1, arr2)
np.fmin(arr1, arr2)
np.logical_and(arr1, arr2)
np.logical_or(arr1, arr2)
np.logical_xor(arr1, arr2)
```

2.3 数学运算与广播

一元函数名	功能	一元函数名	功能
abs, fabs	绝对值	sqrt，square，exp	根号/平方/e的指数
log，log10，log2	对数	sign, ceil , floor	符号/向上取整/向下取整
around/rint	四舍五入/四舍五入到整数	modf	拆分为小数和整数数组
sin, cos, tan	三角函数	sinh, cosh, tanh	双曲型三角函数
logical_not	逻辑非	isnan	是否为非数字类型

In[5]:

```
np.isnan(arr1)  #算术运算前需要判断是否有非法数据
```

2.4 切割、堆叠与操纵

In[6]:

```
x = np.array([11, 12, 13, 14, 15, 16, 17, 18])
x1, x2, x3 = np.split(x, [3, 5])  # 按列表的顺序切割
print(x1, x2, x3)
```

Out[6]:

```
[11 12 13] [14 15] [16 17 18]
```

In[7]:

```
np.vsplit(arr1, [2]) #np.hsplit(arr1, [2])
```

Out[7]:

```
[array([[1, 2, 3],
        [4, 5, 6]]), array([[7, 8, 9]])]
```

2.4 切割、堆叠与操纵

In[8]:

```
arr5 = np.arange(1,25).reshape(2,3,4)
upper, lower = np.split(arr5 , [1] , axis = 0) #axis = 1
```

Out[8]:

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]])

array([[[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
```

2.4 切割、堆叠与操纵

In[9]:

```
np.concatenate((upper, lower), axis=0)  # 拼接
```

Out[9]:

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
```

2.4 切割、堆叠与操纵

In[10]:

```
np.vstack([upper,lower]) #垂直方向堆叠
```

Out[10]:

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
```

2.4 切割、堆叠与操纵

In[11]:

```
np.hstack([upper, lower]) #水平方向堆叠
```

Out[11]:

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
```

2.4 切割、堆叠与操纵

In[12]:

```
arr5 = np.arange(8)
np.delete(arr5, 2)
```

Out[12]:

```
array([0, 1, 3, 4, 5, 6, 7])
```

In[13]:

```
arr6 = np.arange(9).reshape(3, 3)
np.delete(arr6, [1], axis=0)
```

Out[13]:

```
array([[0, 1, 2],
       [6, 7, 8]])
```

2.4 切割、堆叠与操纵

In[14]:

```
np.insert(arr5, 0, -1)
```

Out[14]:

```
array([-1,  0,  1,  2,  3,  4,  5,  6,  7])
```

In[15]:

```
np.insert(arr6, 1 , [0,0,0] ,axis = 0)
```

Out[15]:

```
array([[0, 1, 2],
       [0, 0, 0],
       [3, 4, 5],
       [6, 7, 8]])
```

2.5 排序

In[1]:
```
import numpy as np
myArray = np.array([11, 18, 13, 12, 19, 15, 14, 17, 16])
myArray
```

Out[1]:
```
array([11, 18, 13, 12, 19, 15, 14, 17, 16])
```

In[2]:
```
np.sort(myArray)
```

Out[2]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In[3]:
```
np.argsort(myArray)  #排序后的索引
```

Out[3]:
```
array([0, 3, 2, 6, 5, 8, 7, 1, 4], dtype=int64)
```

2.5 排序

In[4]:

```
MyArray2 = np.array([[21, 22, 23, 24, 25],
                     [35,  34, 33, 32, 31],
                     [1, 2,  3, 100, 4]])
np.sort(MyArray2, axis=0) #np.sort(MyArray, axis=1)
```

Out[4]:

```
array([[  1,   2,   3,  24,   4],
       [ 21,  22,  23,  32,  25],
       [ 35,  34,  33, 100,  31]])
```

```
array([[ 21,  22,  23,  24,  25],
       [ 31,  32,  33,  34,  35],
       [  1,   2,   3,   4, 100]])
```

2.6 数组上的迭代

In[5]:

```
a = np.arange(6).reshape(2, 3)
for x in np.nditer(a):
    print(x)
    print(type(x)) # x也是ndarray
```

Out[5]:

```
0
<class 'numpy.ndarray'>
...
```

NumPy 包含一个迭代器对象numpy.nditer，可使用 Python 的标准 Iterator接口来访问。

2.6 数组上的迭代

In[6]:

```
a = np.arange(6).reshape(2, 3)
b = a.T
for x in np.nditer(b):
    print(x)
    print(type(x))
```

Out[6]:

```
0
<class 'numpy.ndarray'>
...
```

迭代匹配的是数组内容上的顺序，转置函数等操作不影响数组内容排序。

2.6 数组上的迭代

In[7]:

```
c = a.copy(order='F') #fortran风格
for x in np.nditer(c):
    print(x) #输出和前面不同
```

Out[7]:

```
0
<class 'numpy.ndarray'>
3
...
```

理解strides属性对理解Numpy内存管理很关键。

2.6 数组上的迭代

In[8]:

```
x = np.arange(1, 7).reshape(2, 3)
for a in np.nditer(x, op_flags=['readwrite']):
    a[...] = a * 2  # 这里很神奇!
x
```

Out[8]: **array([[2, 4, 6], [8, 10, 12]])**

2.6 数组上的迭代

In[9]:

```
a = np.arange(12).reshape(3, 4)
b = np.arange(4)
for x, y in np.nditer([a, b]): # 类似广播的效果
    print("%d:%d" % (x, y))
```

Out[9]:

```
0:0
1:1
2:2
3:3
4:0
5:1
6:2
7:3
...
```

2.7 字符串处理

In[1]:

```
import numpy as np

a = np.char.add(['hello'], [' seu'])
b = np.char.add(['hello', 'hi'], [' seu', ' baby’])
a  #b
```

Out[1]: array(['hello seu'], dtype='<U9’)

array(['hello seu', 'hi baby'], dtype='<U10')

对dtype为numpy.string_或numpy.unicode_的数组执行向量化字符串操作，在 numpy.char类当中定义。

2.7 字符串处理

In[2]:

```
a = np.char.multiply('Hello', 3)
b = np.char.center('hello', 20, fillchar='*')
c = np.char.capitalize('hello world')
d = np.char.title('hello how are you?')
e = np.char.lower(['HELLO', 'WORLD'])
f = np.char.upper(['hello', 'world'])
```

2.7 字符串处理

In[3]:

```
a = np.char.split('TutorialsPoint,Hyderabad,Telangana', sep=',')
b = np.char.splitlines('hello\rhow are you?')  # \n\r 都是分行符
c = np.char.strip(['arora', 'admin', 'java'], 'a')
d = np.char.join(':', 'dmy')
e = np.char.join([':', '-'], ['dmy', 'ymd'])
f = np.char.replace('He is a good boy', 'is', 'was')
g = np.char.encode('我', 'utf8')
h = np.char.decode(g, 'utf8')
```

2.8 随机模块

In[4]:

```
rand = np.random.RandomState(32)
x = rand.randint(0, 10, (3, 6))
x
```

Out[4]:

```
array([[7, 5, 6, 8, 3, 7],
       [9, 3, 5, 9, 4, 1],
       [3, 1, 2, 3, 8, 2]])
```

In[5]:

```
rand = np.random.RandomState(32)
x = rand.rand(2, 3)
x
```

Out[5]:

```
array([[0.85888927, 0.37271115, 0.55512878],
       [0.95565655, 0.7366696 , 0.81620514]])
```

2.8 随机模块

In[6]:

```
rand = np.random.RandomState(32)
x = rand.uniform(2, 3, (2, 3)) #均匀分布
x
```

Out[6]:

```
array([[2.85888927, 2.37271115, 2.55512878],
       [2.95565655, 2.7366696 , 2.81620514]])
```

In[7]:

```
rand = np.random.RandomState(32)
x = rand.randn(2, 3) #标准正态分布
x
```

Out[7]:

```
array([[-0.34889445,  0.98370343,  0.58092283],
       [ 0.07028444,  0.77753268,  0.58195875]])
```

2.8 随机模块

In[8]:

```
rand = np.random.RandomState(32)
x = rand.normal(3, 1, (2, 3)) #指定正态分布
x
```

Out[8]:

```
array([[2.65110555, 3.98370343, 3.58092283],
[3.07028444, 3.77753268, 3.58195875]])
```

In[9]:

```
rand = np.random.RandomState(32)
x = rand.random((2, 3)) #均匀0-1分布
x
```

Out[9]:

```
array([[0.85888927, 0.37271115, 0.55512878],
       [0.95565655, 0.7366696 , 0.81620514]])
```

2.9 统计函数

In[10]:

```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print(np.amax(a))
print(np.amax(a,0))
print(np.amax(a,1))
```

Out[10]:

```
9
[8 7 9]
[7 8 9]
```

NumPy 有很多有用的统计函数，用于从数组中给定的元素/轴中查找最小，最大，百分标准差和方差等。

2.9 统计函数

In[11]:

```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print(np.percentile(a , 50)) #百分位数
print(np.percentile(a , 50 , axis=0))
print(np.percentile(a , 50 , axis=1))
```

Out[11]:

```
4.0
[3. 4. 5.]
[5. 4. 4.]
```

百分位数是非常常用的统计方法。

2.9 统计函数

In[12]:

```
a = np.array([[3, 7, 5], [8, 4, 3], [2, 4, 9]])
print(np.median(a))
print(np.median(a, axis=0))
print(np.median(a, axis=1))
```

Out[12]:

```
4.0
[3. 4. 5.]
[5. 4. 4.]
```

中位数就是50百分位数

2.9 统计函数

In[13]:

```
a = np.array([[3, 7, 5], [8, 4, 3], [2, 4, 9]])
print(np.mean(a))
print(np.mean(a, axis=0))
print(np.mean(a, axis=1))
```

Out[13]:

```
5.0
[4.33333333 5.         5.66666667]
[5. 5. 5.]
```

平均值函数

2.9 统计函数

In[14]:

```
a = np.array([1, 2, 3, 4])
b = np.average(a)
wts = np.array([4, 3, 2, 1])
c = np.average(a,weights = wts)
b, c
```

Out[14]: (2.5, 2.0)

加权平均

2.9 统计函数

In[15]:

```
print(np.std([1, 2, 3, 4]))  # 标准差
print(np.var([1, 2, 3, 4]))  # 方差
```

Out[15]:

```
1.1180339887498895
1.25
```

2.9 统计函数

In[16]:
```
a = np.array([1, 2, 3, np.nan])
np.isnan(a)
```

Out[16]: **array([False, False, False, True])**

In[17]:
```
np.any(np.isnan(a))
```

Out[17]: **True**

In[18]:
```
np.all(np.isnan(a))
```

Out[18]: **True**

2.9 统计函数

In[19]:
```
np.sum(a)
```

Out[19]: **nan**

In[20]:
```
np.nansum(a)
```

Out[20]: **6.0**

2.9 统计函数

In[21]:

```
a = np.array([[30, 40, 70], [80, 20, 10], [50, 90, 60]])
print(np.argmax(a))
print(np.argmax(a, axis=0))
print(np.argmax(a, axis=1))
```

Out[21]:

```
7
[1 2 0]
[2 0 1]
```

2.9 统计函数

In[22]:

```
a = np.array([[30, 40, 0], [0, 20, 10], [50, 0, 60]])
print(np.nonzero(a))
b = np.where(a > 40)
print(b)
print(a[b])
c = np.extract(np.mod(a, 3) == 0, a)
print(c)
```

Out[22]:

```
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
(array([2, 2]), array([0, 2]))
[50 60]
[30  0  0  0 60]
```

2.9 统计函数

In[23]:

```
nums = [1,2,3,6,5,6,6]
a = np.bincount(nums)  #返回0～最大值出现的次数
np.argmax(a) #非负整数的众数
```

Out[23]: 6

2.10 引用、视图与复制

In[1]:

```
import numpy as np
a = np.arange(6).reshape(2, 3)
b = a
b.resize(3, 2)
print(a.shape)
print(a)
```

Out[1]:

```
(3, 2)
[[0 1]
 [2 3]
 [4 5]]
```

2.10 引用、视图与复制

In[2]:

```
a = np.arange(6).reshape(2, 3)
b = a.view()
b.resize(3, 2)
print(a.shape)
print(a)
b[0][0] = 7
print(a)
```

Out[2]:

```
(2, 3)
[[0 1 2]
 [3 4 5]]
[[7 1 2]
 [3 4 5]]
```

2.10 引用、视图与复制

In[3]:

```
a = np.arange(6).reshape(2, 3)
b = a.copy()
b.resize(3, 2)
print(a.shape)
print(a)
b[0][0] = 7
print(a)
```

Out[3]:

```
(2, 3)
[[0 1 2]
 [3 4 5]]
[[0 1 2]
 [3 4 5]]
```

2.11 矩阵库与线性代数

In[4]:

```
import numpy.matlib #注意这句话
a = np.matlib.empty((2, 2))
b = np.matlib.zeros((2, 2))
c = np.matlib.ones((2, 2))
d = np.matlib.eye(n=3, M=3, k=0, dtype=float)
e = np.matlib.identity(5, dtype=float)
f = np.matlib.rand((2, 2))
```

NumPy 包包含一个 Matrix库numpy.matlib，此模块的函数返回矩阵而不是返回ndarray对象。

2.11 矩阵库与线性代数

In[5]:

```
a = np.array([[1, 2], [3, 4]])
b = np.asmatrix(a)
c = np.asarray(b)
```

通过函数进行转换

2.11 矩阵库与线性代数

In[6]:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[11, 12], [13, 14]])
print(np.dot(a, b))  # 数组点积
print(np.vdot(a, b))  # 向量点积
print(np.inner(np.array([1, 2, 3]), np.array([0, 1, 0])))  # 向量内积
print(np.inner(a, b))  # 多维数组的内积
```

Out[6]:

```
[[37 40]
 [85 92]]
130
2
[[35 41]
 [81 95]]
```

2.11 矩阵库与线性代数

In[7]:

```
a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
print(np.matmul(a, b))
a = [[1, 0], [0, 1]]
b = [1, 2]
print(np.matmul(a, b))
print(np.matmul(b, a))
```

Out[7]:

```
[[4 1]
 [2 2]]
[ 5 11]
[ 7 10]
```

注：如果第一个参数或者第二个参数是1维的，它会提升该参数为矩阵。

2.11 矩阵库与线性代数

In[8]:

```
a = np.array([[1, 1, 1], [0, 2, 5], [2, 5, -1]])
ainv = np.linalg.inv(a)
print(ainv)
b = np.array([[6], [-4], [27]])
x = np.linalg.solve(a, b)
print(x)
```

Out[8]:

```
[[ 1.28571429 -0.28571429 -0.14285714]
 [-0.47619048  0.14285714  0.23809524]
 [ 0.19047619  0.14285714 -0.0952381 ]]
[[ 5.]
 [ 3.]
 [-2.]]
```

$$x + y + z = 6$$
$$2y + 5z = -4$$
$$2x + 5y - z = 27$$

2.12 文件IO

In[9]:

```
a = np.array([1, 2, 3, 4, 5])
np.save('outfile', a)  # 默认使用python序列化生成一个npy文件
b = np.load('outfile.npy')
b
```

Out[9]:

```
array([1, 2, 3, 4, 5])
```

2.12 文件IO

In[10]:

```
a = np.array([1, 2, 3, 4, 5])
np.savetxt('out.txt', a)
b = np.loadtxt('out.txt')
b
```

Out[10]:

```
array([1., 2., 3., 4., 5.])
```

谢谢大家！