# 计算库

# [numpy](http://www.numpy.org/)

NumPy提供了两种基本的对象：ndarray（N-dimensional array object）和 ufunc（universal function object）。ndarray是存储单一数据类型的多维数组，ufunc则是能够对数组进行处理的函数。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，是一个流行的技术计算平台。

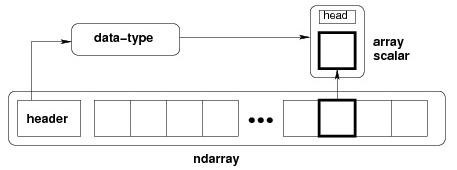
**功能：**

* 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。

## Ndarray 对象

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）。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。



**创建ndarray**

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

| 参数 | 属性 |
| --- | --- |
| 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。 |
| dtype | 可选，数组的所需数据类型 |
| copy | 可选，默认为true，对象是否被复制。 |
| order | C（按行）、F（按列）或A（任意，默认）。 |
| 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。 |
| ndimin | 指定返回数组的最小维数。 |

## 数据类型

### numpy数据类型

| 数据类型 | 描述 |
| --- | --- |
| bool\_ | 存储为一个字节的布尔值（真或假） |
| int\_ | 默认整数，相当于 C 的long，通常为int32或int64 |
| intc | 相当于 C 的int，通常为int32或int64 |
| intp | 用于索引的整数，通常为int32或int64 |
| int8 | 字节（-128 ~ 127） |
| int16 | 16 位整数（-32768 ~ 32767） |
| int32 | 32 位整数（-2147483648 ~ 2147483647） |
| int64 | 64 位整数 |
| 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） |
| float16 | 半精度浮点：符号位，5 位指数，10 位尾数 |
| float32 | 单精度浮点：符号位，8 位指数，23 位尾数 |
| float64、float\_ | 双精度浮点：符号位，11 位指数，52 位尾数 |
| complex64 | 复数，由两个 32 位浮点表示（实部和虚部） |
| complex128、complex\_ | 复数，由两个 64 位浮点表示（实部和虚部） |

### 数据类型对象

数据类型对象描述了对应于数组的固定内存块的解释，取决于以下方面：

* 数据类型（整数、浮点或者 Python 对象）
* 数据大小
* 字节序（小端或大端）
* 在结构化类型的情况下，字段的名称，每个字段的数据类型，和每个字段占用的内存块部分。
* 如果数据类型是子序列，它的形状和数据类型。

字节顺序取决于数据类型的前缀<或>。 <意味着编码是小端（最小有效字节存储在最小地址中）。 >意味着编码是大端（最大有效字节存储在最小地址中）。

**创建dtype**

**numpy.dtype(object, align, copy)**

参数：

* Object：被转换为数据类型的对象。
* Align：如果为true，则向字段添加间隔，使其类似 C 的结构体。
* Copy：生成dtype对象的新副本，如果为flase，结果是内建数据类型对象的引用。

dt=np.dtype([('age',np.int8)])  
a=np.array([(10,),(20),(30,)],dtype=dt)  
print(a['age'])

## 数组

### 广播

**广播**是指 NumPy 在算术运算期间处理不同形状的数组的能力。对数组的算术运算通常在相应的元素上进行。 如果两个阵列具有完全相同的形状，则这些操作被无缝执行。

a = np.array([1,2,3,4])   
b = np.array([10,20,30,40])   
c = a \* b

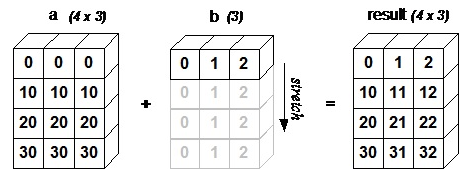
如果两个数组的维数不相同， 较小的数组会**广播**到较大数组的大小，以便使它们的形状可兼容。

满足以下规则，可以进行广播：

* ndim较小的数组会在前面追加一个长度为 1 的维度。
* 输出数组的每个维度的大小是输入数组该维度大小的最大值。
* 如果输入在每个维度中的大小与输出大小匹配，或其值正好为 1，则在计算中可它。
* 如果输入的某个维度大小为 1，则该维度中的第一个数据元素将用于该维度的所有计算。

如果上述规则产生有效结果，并且满足以下条件之一，那么数组被称为**可广播的**。

* 数组拥有相同形状。
* 数组拥有相同的维数，每个维度拥有相同长度，或者长度为 1。
* 数组拥有极少的维度，可以在其前面追加长度为 1 的维度，使上述条件成立。



### 数组操作

1. **数组属性**

**ndarray.shape** #返回一个包含数组维度的元组

调整数组大小

a = np.array([[1,2,3],[4,5,6]])

a.shape = (3,2) #调整数组为三行两列

**ndarray.ndim** #返回数组的维数

**numpy.itemsize**  #返回数组中每个元素的字节单位长度

1. **修改形状**

|  |  |
| --- | --- |
| reshape | 不改变数据的条件下修改形状 |
| flat | 数组上的一维迭代器 |
| flatten | 返回折叠为一维的数组副本 |
| ravel | 返回连续的展开数组 |

**numpy.reshape(arr, newshape, order='C')**

* arr：要修改形状的数组
* newshape：整数或者整数数组，新的形状应当兼容原有形状
* order：'C'为 C 风格顺序，'F'为 F 风格顺序，'A'为保留原顺序。

**numpy.flags**  返回ndarray当前属性值

1. **翻转操作**

|  |  |
| --- | --- |
| 1. | transpose 翻转数组的维度 |
| 2. | ndarray.T 和self.transpose()相同 |
| 3. | rollaxis 向后滚动指定的轴 |
| 4. | swapaxes 互换数组的两个轴 |

**numpy.transpose(arr, axes)**

* arr：要转置的数组
* axes：整数的列表，对应维度，通常所有维度都会翻转。

**numpy.rollaxis(arr, axis, start)**

* arr：输入数组
* axis：要向后滚动的轴，其它轴的相对位置不会改变
* start：默认为零，表示完整的滚动。会滚动到特定位置。

**numpy.swapaxes(arr, axis1, axis2)**

* arr：要交换其轴的输入数组
* axis1：对应第一个轴的整数
* axis2：对应第二个轴的整数

1. **修改维度**

|  |  |
| --- | --- |
| 1. | broadcast 产生模仿广播的对象 |
| 2. | broadcast\_to 将数组广播到新形状 |
| 3. | expand\_dims 扩展数组的形状 |
| 4. | squeeze 从数组的形状中删除单维条目 |

**numpy.expand\_dims(arr, axis)**

* arr：输入数组
* axis：新轴插入的位置

**numpy.squeeze(arr, axis)**

* arr：输入数组
* axis：整数或整数元组，用于选择形状中单一维度条目的子集

1. **数组的连接**

|  |  |
| --- | --- |
| 1. | concatenate 沿着现存的轴连接数据序列 |
| 2. | stack 沿着新轴连接数组序列 |
| 3. | hstack 水平堆叠序列中的数组（列方向） |
| 4. | vstack 竖直堆叠序列中的数组（行方向） |

**numpy.concatenate((a1, a2, ...), axis)**

* a1, a2, ...：相同类型的数组序列
* axis：沿着它连接数组的轴，默认为 0

**numpy.stack(arrays, axis)**

* arrays：相同形状的数组序列
* axis：返回数组中的轴，输入数组沿着它来堆叠

1. **数组分割**

|  |  |
| --- | --- |
| 1. | split 将一个数组分割为多个子数组 |
| 2. | hsplit 将一个数组水平分割为多个子数组（按列） |
| 3. | vsplit 将一个数组竖直分割为多个子数组（按行） |

**numpy.split(ary, indices\_or\_sections, axis)**

* ary：被分割的输入数组
* indices\_or\_sections：可以是整数，表明要从输入数组创建的，等大小的子数组的数量。 如果此参数是一维数组，则其元素表明要创建新子数组的点。
* axis：默认为 0

1. **添加/删除元素**

|  |  |
| --- | --- |
| 1. | resize 返回指定形状的新数组 |
| 2. | append 将值添加到数组末尾 |
| 3. | insert 沿指定轴将值插入到指定下标之前 |
| 4. | delete 返回删掉某个轴的子数组的新数组 |
| 5. | unique 寻找数组内的唯一元素 |

**numpy.resize(arr, shape)**

* arr：要修改大小的输入数组
* shape：返回数组的新形状

**numpy.append(arr, values, axis)**

* arr：输入数组
* values：要向arr添加的值，比如和arr形状相同（除了要添加的轴）
* axis：沿着它完成操作的轴。如果没有提供，两个参数都会被展开。

**numpy.insert(arr, obj, values, axis)**

* arr：输入数组
* obj：在其之前插入值的索引
* values：要插入的值
* axis：沿着它插入的轴，如果未提供，则输入数组会被展开

**Numpy.delete(arr, obj, axis)**

* arr：输入数组
* obj：可以被切片，整数或者整数数组，表明要从输入数组删除的子数组
* axis：沿着它删除给定子数组的轴，如果未提供，则输入数组会被展开

**numpy.unique(arr, return\_index, return\_inverse, return\_counts)**

* arr：输入数组，如果不是一维数组则会展开
* return\_index：如果为true，返回输入数组中的元素下标
* return\_inverse：如果为true，返回去重数组的下标，它可以用于重构输入数组
* return\_counts：如果为true，返回去重数组中的元素在原数组中的出现次数

### 位操作

|  |  |
| --- | --- |
| 1. | bitwise\_and 对输入数组中的整数的二进制表示的相应位执行位与运算 |
| 2. | bitwise\_or 对输入数组中的整数的二进制表示的相应位执行位或运算 |
| 3. | invert 计算位非，对于有符号整数，返回补码。 |
| 4. | left\_shift 将数组元素的二进制表示中的位向左移动到指定位置，右侧补0 |
| 5. | right\_shift将数组元素的二进制表示中的位向右移动到指定位置，左侧补0 |

### 构造函数

1. 创建指定形状和dtype的未初始化数组（数组元素为随机值）

**numpy.empty(shape, dtype = float, order = 'C')**

|  |  |
| --- | --- |
| 1. | Shape 空数组的形状，整数或整数元组 |
| 2. | Dtype 所需的输出数组类型，可选 |
| 3. | Order 'C'为按行的 C 风格数组，'F'为按列的 Fortran 风格数组 |

1. 返回特定大小，以 1 填充的新数组。

**numpy.ones(shape, dtype = None, order = 'C')**

|  |  |
| --- | --- |
| 1. | Shape 空数组的形状，整数或整数元组 |
| 2. | Dtype 所需的输出数组类型，可选 |
| 3. | Order 'C'为按行的 C 风格数组，'F'为按列的 Fortran 风格数组 |

1. Python 序列转换为ndarray

**numpy.asarray(a, dtype = None, order = None)**

|  |  |
| --- | --- |
| 1. | a 任意形式的输入参数，比如列表、列表的元组、元组、元组的元组、元组的列表 |
| 2. | dtype 通常，输入数据的类型会应用到返回的ndarray |
| 3. | order 'C'为按行的 C 风格数组，'F'为按列的 Fortran 风格数组 |

1. 将缓冲区解释为一维数组

**numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)**

|  |  |
| --- | --- |
| 1. | buffer 任何暴露缓冲区借口的对象 |
| 2. | dtype 返回数组的数据类型，默认为float |
| 3. | count 需要读取的数据数量，默认为-1，读取所有数据 |
| 4. | offset 需要读取的起始位置，默认为0 |

1. 从任何可迭代对象构建一个ndarray对象

**numpy.fromiter(iterable, dtype, count = -1)**

|  |  |
| --- | --- |
| 1. | iterable 任何可迭代对象 |
| 2. | dtype 返回数组的数据类型 |
| 3. | count 需要读取的数据数量，默认为-1，读取所有数据 |

1. 从数值范围创建数组

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

|  |  |
| --- | --- |
| 1. | start 范围的起始值，默认为0 |
| 2. | stop 范围的终止值（不包含） |
| 3. | step 两个值的间隔，默认为1 |
| 4. | dtype 返回ndarray的数据类型，如果没有提供，则会使用输入数据的类型。 |

**numpy.linspace(start, stop, num, endpoint, retstep, dtype)**

|  |  |
| --- | --- |
| 1. | start 序列的起始值 |
| 2. | stop 序列的终止值，如果endpoint为true，该值包含于序列中 |
| 3. | num 要生成的等间隔样例数量，默认为50 |
| 4. | endpoint 序列中是否包含stop值，默认为ture |
| 5. | retstep 如果为true，返回样例，以及连续数字之间的步长 |
| 6. | dtype 输出ndarray的数据类型 |

返回一个ndarray对象，其中包含在对数刻度上均匀分布的数字。 刻度的开始和结束端点是某个底数的幂，通常为 10。

**numpy.logspace(start, stop, num, endpoint, base, dtype)**

|  |  |
| --- | --- |
| 1. | start 起始值是base \*\* start |
| 2. | stop 终止值是base \*\* stop |
| 3. | num 范围内的数值数量，默认为50 |
| 4. | endpoint 如果为true，终止值包含在输出数组当中 |
| 5. | base 对数空间的底数，默认为10 |
| 6. | dtype 输出数组的数据类型，如果没有提供，则取决于其它参数 |

### 数组上的迭代

NumPy 包包含一个迭代器对象numpy.nditer。 它是一个有效的多维迭代器对象，可以用于在数组上进行迭代。 数组的每个元素可使用 Python 的标准Iterator接口来访问。

a = np.arange(0,60,5)  
*for* x *in* np.nditer(a):  
 print(x) #迭代输出每一个元素

1. **迭代顺序**

如果相同元素使用 F 风格顺序存储，则迭代器选择以更有效的方式对数组进行迭代。

a = np.arange(0,60,5)  
a = a.reshape(3,4)  
b = a.T  
c = b.copy(order='C') #以 C 风格顺序排序  
*for* x *in* np.nditer(c):  
 print(x) #输出0 20 40 5 25 45 10 30 50 15 35 55  
c = b.copy(order='F') #以 F 风格顺序排序  
*for* x *in* np.nditer(c):  
 print(x) # 输出0 5 10 15 20 25 30 35 40 45 50 55

可以通过显式提醒，来强制nditer对象使用某种顺序

*for* x *in* np.nditer(a, order = 'C'):

1. **修改数组的值**

nditer对象有另一个可选参数op\_flags。 其默认值为只读，但可以设置为读写或只写模式。 这将允许使用此迭代器修改数组元素。

*for* x *in* np.nditer(a, op\_flags=['readwrite']):

1. **外部循环**

nditer类的构造器拥有flags参数，它可以接受下列值

|  |  |
| --- | --- |
| 1. | c\_index 可以跟踪 C 顺序的索引 |
| 2. | f\_index 可以跟踪 Fortran 顺序的索引 |
| 3. | multi-index 每次迭代可以跟踪一种索引类型 |
| 4. | external\_loop 给出的值是具有多个值的一维数组，而不是零维数组 |

1. **广播迭代**

如果两个数组是**可广播的**，nditer组合对象能够同时迭代它们。

## 切片和索引

ndarray对象中的元素遵循基于零的索引。 有三种可用的索引方法类型： 字段访问，基本切片和高级索引。

### 基本切片

基本切片是 Python 中基本切片概念到 n 维的扩展。 通过将start，stop和step参数提供给内置的slice函数来构造一个 Python的slice对象。 此slice对象被传递给数组来提取数组的一部分。

通过将由冒号分隔的切片参数（start:stop:step）直接提供给ndarray对象，也可以获得相同的结果

如果只输入一个参数，则将返回与索引对应的单个项目。 如果使用start:，则从该索引向后的所有项目将被提取。 如果使用两个参数（以:分隔），则对两个索引（不包括停止索引）之间的元素以默认步骤进行切片。

a = np.arange(10)  
s = slice(2,7,2)   
b = a[2:7:2]

切片还可以包括省略号（...），来使选择元组的长度与数组的维度相同。 如果在行位置使用省略号，它将返回包含行中元素的ndarray。

a = np.array([[1,2,3],[3,4,5],[4,5,6]])  
print(a[...,1]) # 第二列元素  
print(a[1,...]) # 第二行元素  
print(a[...,1:]) # 第二列及剩余元素

### 高级索引

如果一个ndarray是非元组序列，数据类型为整数或布尔值的ndarray，或者至少一个元素为序列对象的元组，我们就能够用它来索引ndarray。高级索引始终返回数据的副本。 与此相反，切片只提供了一个视图。

有两种类型的高级索引：整数和布尔值。

1. 整数索引

每个整数数组表示该维度的下标值。

x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])   
rows = np.array([[0,0],[3,3]])   
cols = np.array([[0,2],[0,2]]) #(0,0)(0,2)(3,0)(3,2)的元素  
y = x[rows,cols] #输出一个二维数组

x = np.array([[1, 2], [3, 4], [5, 6]])   
y = x[[0,1,2], [0,1,0]] #(0,0)(1,1)(2,0)  
print y

1. 布尔索引

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。

print (x[x > 5]) #打印出大于 5 的元素

## 字符串函数（numpy.char）

|  |  |
| --- | --- |
| 1. | add() 返回两个str或Unicode数组的逐个字符串连接 |
| 2. | multiply() 返回按元素多重连接后的字符串 |
| 3. | center() 返回给定字符串的副本，其中元素位于特定字符串的中央 |
| 4. | capitalize() 返回给定字符串的副本，其中只有第一个字符串大写 |
| 5. | title() 返回字符串或 Unicode 的按元素标题转换版本 |
| 6. | lower() 返回一个数组，其元素转换为小写 |
| 7. | upper() 返回一个数组，其元素转换为大写 |
| 8. | split() 返回字符串中的单词列表，并使用分隔符来分割 |
| 9. | splitlines() 返回元素中的行列表，以换行符分割 |
| 10. | strip() 返回数组副本，其中元素移除了开头或者结尾处的特定字符 |
| 11. | join() 返回一个字符串，它是序列中字符串的连接 |
| 12. | replace() 返回字符串的副本，其中所有子字符串的出现位置都被新字符串取代 |
| 13. | decode() 按元素调用str.decode |
| 14. | encode() 按元素调用str.encode |

## 函数

### 算数函数

1. 三角函数

NumPy 拥有标准的三角函数，它为弧度制单位的给定角度返回三角函数比值。

arcsin，arccos，和arctan函数返回给定角度的sin，cos和tan的反三角函数。 这些函数的结果可以通过numpy.degrees()函数通过将弧度制转换为角度制来验证。

1. 舍入函数

**numpy.around(a,decimals)**

|  |  |
| --- | --- |
| 1. | a 输入数组 |
| 2. | decimals 要舍入的小数位数，默认值0。如果为负，整数将四舍五入到小数点左侧 |

**numpy.floor()**

此函数返回不大于输入参数的最大整数。 即标量x 的下限是最大的整数i ，使得i <= x。 注意在Python中，向下取整总是从 0 舍入

**numpy.ceil()**

ceil()函数返回输入值的上限，即，标量x的上限是最小的整数i ，使得i> = x。

### 算数运算

用于执行算术运算（如add()，subtract()，multiply()和divide()）的输入数组必须具有相同的形状或符合数组广播规则。

**numpy.reciprocal()**

此函数返回参数逐元素的倒数，。 由于 Python 处理整数除法的方式，对于绝对值大于 1 的整数元素，结果始终为 0， 对于整数 0，则发出溢出警告

**numpy.power()**

此函数将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂

**numpy.mod()**

此函数返回输入数组中相应元素的除法余数。 函数numpy.remainder()也产生相同的结果。

### 统计函数

**numpy.amin() 和 numpy.amax()**

从给定数组中的元素沿指定轴返回最小值和最大值

**numpy.ptp()**

返回沿轴的值的范围（最大值 - 最小值）。

**numpy.percentile(a, q, axis)**

百分位数是统计中使用的度量，表示小于这个值的观察值的百分比。

|  |  |
| --- | --- |
| 1. | a 输入数组 |
| 2. | q 要计算的百分位数，在 0 ~ 100 之间 |
| 3. | axis 沿着它计算百分位数的轴 |

**numpy.median()**

中值定义为将数据样本的上半部分与下半部分分开的值。

**numpy.mean()**

算术平均值是沿轴的元素的总和除以元素的数量。 numpy.mean()函数返回数组中元素的算术平均值。 如果提供了轴，则沿其计算。

**numpy.average()**

加权平均值是由每个分量乘以反映其重要性的因子得到的平均值。 numpy.average()函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值。 该函数可以接受一个轴参数。 如果没有指定轴，则数组会被展开。

**标准差**

与均值的偏差的平方的平均值的平方根。

std = sqrt(mean((x - x.mean())\*\*2))

**方差**

方差的平方根，即mean((x - x.mean())\*\* 2)。 换句话说，标准差是

### 排序、搜索和计数函数

**numpy.sort(a, axis, kind, order)**

sort()函数返回输入数组的排序副本

|  |  |
| --- | --- |
| 1. | a 要排序的数组 |
| 2. | axis 沿着它排序数组的轴，如果没有数组会被展开，沿着最后的轴排序 |
| 3. | kind 默认为'quicksort'（快速排序） |
| 4. | order 如果数组包含字段，则是要排序的字段 |

**numpy.argsort()**

numpy.argsort()函数对输入数组沿给定轴执行间接排序，并使用指定排序类型返回数据的索引数组。 这个索引数组用于构造排序后的数组

**numpy.lexsort()**

函数使用键序列执行间接排序。 键可以看作是电子表格中的一列。 该函数返回一个索引数组，使用它可以获得排序数据。 注意，最后一个键恰好是 sort 的主键。

**numpy.argmax() 和 numpy.argmin()**

这两个函数分别沿给定轴返回最大和最小元素的索引。

**numpy.nonzero()**

numpy.nonzero()函数返回输入数组中非零元素的索引。

**numpy.where()**

where()函数返回输入数组中满足给定条件的元素的索引。

**numpy.extract()**

extract()函数返回满足任何条件的元素。

### 字节交换

存储在计算机内存中的数据取决于 CPU 使用的架构。 它可以是小端（最小有效位存储在最小地址中）或大端（最小有效字节存储在最大地址中）。

**numpy.ndarray.byteswap()**大端和小端之间切换

## 矩阵库

numpy.matlib模块的函数返回矩阵而不是返回ndarray对象

**numpy.matlib.empty(shape, dtype, order)**

返回一个新的矩阵，而不初始化元素

|  |  |
| --- | --- |
| 1. | shape 定义新矩阵形状的整数或整数元组 |
| 2. | Dtype 可选，输出的数据类型 |
| 3. | order C 或者 F |

**numpy.matlib.zeros()**

此函数返回以零填充的矩阵。

**numpy.matlib.ones()**

返回以一填充的矩阵。

**numpy.matlib.eye(n, M,k, dtype)**

返回一个矩阵，对角线元素为 1，其他位置为零

|  |  |
| --- | --- |
| 1. | n 返回矩阵的行数 |
| 2. | M 返回矩阵的列数，默认为n |
| 3. | k 对角线的索引 |
| 4. | dtype 输出的数据类型 |

**numpy.matlib.identity()**

返回给定大小的单位矩阵。单位矩阵是主对角线元素都为 1 的方阵。

**numpy.matlib.rand()**

返回给定大小的填充随机值的矩阵。

## 线性代数

numpy.linalg模块，提供线性代数所需的所有功能

|  |  |
| --- | --- |
| 1. | dot 两个数组的点积 |
| 2. | vdot 两个向量的点积 |
| 3. | inner 两个数组的内积 |
| 4. | matmul 两个数组的矩阵积 |
| 5. | determinant 数组的行列式 |
| 6. | solve 求解线性矩阵方程 |
| 7. | inv 寻找矩阵的乘法逆矩阵 |

**numpy.dot()**

返回两个数组的点积。

* 对于二维向量，其等效于矩阵乘法。
* 对于一维数组，它是向量的内积。
* 对于 N 维数组，它是a的最后一个轴上的和与b的倒数第二个轴的乘积。

**numpy.vdot()**

返回两个向量的点积。

* 如果第一个参数是复数，那么它的共轭复数会用于计算。
* 如果参数id是多维数组，它会被展开。

**numpy.inner()**

返回一维数组的向量内积。

对于更高的维度，它返回最后一个轴上的和的乘积。

**numpy.matmul()**

返回两个数组的矩阵乘积。

* 虽然它返回二维数组的正常乘积，但如果任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。
* 如果任一参数是一维数组，则通过在其维度上附加 1 来将其提升为矩阵，并在乘法之后被去除。

**numpy.linalg.det()**

行列式在线性代数中是非常有用的值。 它从方阵的对角元素计算。 对于 2×2 矩阵

[[a，b]，[c，d]]，行列式计算为ad-bc。 较大的方阵被认为是 2×2 矩阵的组合。

**numpy.linalg.det()**

计算输入矩阵的行列式。

**numpy.linalg.solve()**

给出了矩阵形式的线性方程的解。

**numpy.linalg.inv()**

计算矩阵的逆。

# pandas

## DataFrame

DataFrame由行和列组成。从数据框架中获取列的方式与从字典中获取元素的方式相同。

### 选择列

1. 选择一列

获得dataframe某一列前几行，使用切片**df[‘name’][:5]**或者**df [:5] [‘name’]**

1. 选择多列

**df[[‘name1’,’name2’]][:10]**

### 绘制列

**DataFrame.plot()**

### 转置

**df.T**

## 创建数据

1. 创建数据集

使用**zip()**函数将列表合并在一起

1. 创建DataFrame对象

**df = pd.DataFrame(data = data, columns=['Names', 'Births'])**

1. 导出数据

**df.to\_csv(‘filename’,index=False, header=False)**

|  |  |
| --- | --- |
| ***index*** | false：防止索引(index) 被导出到文件中 |
| ***header*** | false：防止列名(header names) 被导出到文件中 |

索引(index)可以理解为一个sql表中的主键(primary key)，但是索引是可以重复的。

## 获取数据

### 从文件读取

使用read\_csv()函数从CSV文件读取数据。 默认情况下，它假定字段以逗号分隔

Location = r'./births1880.csv' #从 notebook 当前的位置读取 csv 文件

注意字符串之前的***r*** 。 因为斜线(slash)是一个特殊字符，在字符串之前放置前导的 ***r*** 将会把整个字符串进行转义(escape)。

* 如果表中不含有列名，默认将第一行设置为列名，添加header=None后显示正常。

df = pd.read\_csv(Location, header=None)

* 指定一个名字可以去掉header参数,传入names参数

df = pd.read\_csv(Location, names=['Names', 'Births'])

* 指定sheet和列索引（index）

# 读取第一个页签(sheet)，并指定索引列是 StatusDate  
df = pd.read\_excel(Location, sheetname=0, index\_col='StatusDate')

### 从数据库中读取

使用sqlalchemy和pymysql库

* pandas模块提供了read\_sql\_query（）函数实现了对数据库的查询，to\_sql（）函数实现了对数据库的写入
* sqlalchemy模块实现了与不同数据库的连接
* pymysql模块则使得Python能够操作MySQL数据库。

## 添加、删除数据

1. 删除某一列

del df['Status']

1. 添加一列

当使用 dataframe 没有的列时，dataframe 自动增加这个新列

df['NewCol'] = 5

df['col'] = df['Rev']

1. 修改一列的值

df['NewCol'] = df['NewCol'] + 1

1. 修改索引

i = ['a','b','c','d','e','f','g','h','i','j']

df.index = i

## 函数

### 计算一列频数

**DataFrame.value\_counts()**

可切片显示

### 筛选、排序

筛选显示指定列的指定值

1. 一个指定值

**name1=df[df[‘name\_type’]==’name1’]**

**name1[:3]**

1. 多个指定值

用&运算符组合

**a=df[‘type’]==’1’**

**b=df[‘name’]==’2’**

**df[a&b][:5]**

1. 筛选后指定列

**df[a&b][‘name’,’type’,’time’][:5]**

1. 只保留筛选列

# 只保留 Status == 1  
mask = df['Status'] == 1  
df = df[mask]

1. 选中部分数据

**df.loc[起始索引(包含):终止索引(包含)]**

df.iloc[起始索引(包含):终止索引(不包含)] #iloc 非常严格限制在整形的索引上

df.ix[行范围, 列范围]

1. 查看前面的值

**df.head(2)** #默认为5条

1. 查看后面的值

**df.tail()** #默认为5条

1. 排序显示

Sorted = df.sort\_values(['Births'], ascending=False)  
Sorted.head(1)

或者显示'Births'列中的最大值

df['Births'].max()

1. 显示排序后取值对应的其他列的值
   * 直接显示

# 在 Names 列中挑选出 [Births 列的值等于最大值的]   
df['Names'][df['Births'] == df['Births'].max()]

* + 使用排序后的显示

Sorted['Names'].head(1).value

### 查看数据类型

注意dtypes和dtype

# 查看每一列的数据类型  
*print* df.dtypes  
# 查看指定列的数据类型  
*print* df.names.dtype

### 列操作

1. 修改列名

data.columns = ['nums','count']

1. 设置为index

df = pd.read\_excel(Location, sheetname=0, index\_col='StatusDate')

1. index重置为列

**df.reset\_index（）**

将作为索引的列转为普通的列

1. 和并列

**pd.concat(objs, axis=0, join='outer', join\_axes=None, ignore\_index=False,**

**keys=None, levels=None, names=None, verify\_integrity=False,**

**copy=True)**

* + objs：Series，DataFrame或Panel对象的序列或映射。如果传递了dict，则排序的键将用作键参数，除非它被传递，在这种情况下，将选择值（见下文）。任何无对象将被静默删除，除非它们都是无，在这种情况下将引发一个ValueError。
  + axis：{0,1，...}，默认为0。沿着连接的轴。0为

### 获取唯一数据

1. 获取某一列的唯一的记录
2. 只获取值

df['Names'].unique()

*for* x *in* df['Names'].unique():  
 *print*(x)

1. 获取值和所有数据统计

df['Names'].describe()

1. 将唯一性的其他列所有值汇总

**groupby([‘listname1’,’ listname2’])**

# 创建一个 groupby 的对象  
name = df.groupby('names')  
# 在 groupby 对象上执行求和(sum)的功能  
df = name.sum()

# 不把用来分组的列名作为索引  
three=df.groupby(['letter','one'],as\_index=False).sum()

*print* three.index

groupby()使用columns作为输入，如果是index列，使用重置后再进行groupby

### 大小写转换

# 清洗 State 列，全部转换为大写  
df['State'] = df.State.apply(*lambda* x: x.upper())

### 赋值

将 ***NJ*** 转换为 ***NY***

mask = df.State == 'NJ'  
df['State'][mask] = 'NY'

### 创建日期

**date\_range(start=None, end=None, periods=None, freq='D', tz=None, normalize=False, name=None, closed=None)**

返回一个固定频率的日期时间索引，用日历天作为默认的频率

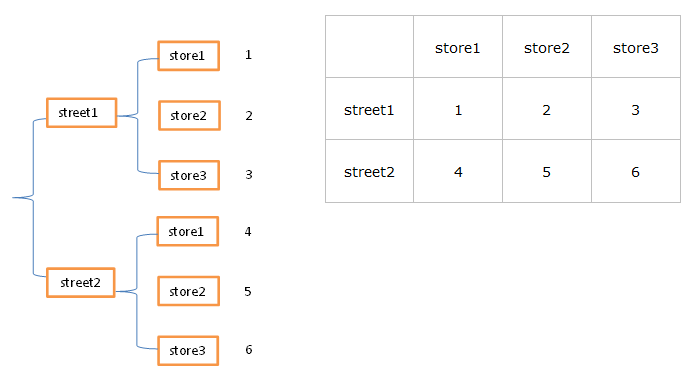
### 重塑层次化结构

stack函数会将数据从”表格结构“变成”花括号结构“，即将其行索引变成列索引

**df.stack()**

unstack函数将数据从”花括号结构“变成”表格结构“，即要将其中一层的列索引变成行索引。

**df.unstack()**



## 绘图

1. 绘制柱状图

df.plot.bar()

# 图形库

# Matplotlib

## 图形

plt.plot(x, y, label='First Line') # label：线条名称，以后可以在图例中显示

plt.xlabel('Plot Number') # 相应的轴创建标签

plt.ylabel('Important var')

plt.title('Interesting Graph\nCheck it out') #创建图的标题

plt.legend() #生成默认图例

plt.show()

### 条形图

plt.bar创建条形图

plt.bar([1,3,5,7,9], [5,3,7,8,3],label='first')  
plt.bar([2,4,6,8,10], [12,3,2,6,4],color='g',label='second')  
plt.title('bar')  
plt.legend()  
plt.ylabel('y')  
plt.xlabel('x')  
plt.show()

### 直方图

直方图倾向于通过将区段组合在一起来显示分布。

plt.hist，首先需要放入所有的值，然后指定放入哪个桶或容器

a=np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])  
plt.hist(a,bins=[0,20,40,60,80,100],color='g',histtype='bar', rwidth=0.8)  
plt.title('histogrm')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.legend()  
plt.show()

### 散点图

x = [1,2,3,4,5,6,7,8]  
y = [5,2,4,2,1,4,5,2] #如果在 3 维绘制则是 3 个。  
plt.scatter(x,y, label='skitscat', color='k', s=25, marker="o")  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title('Interesting Graph\nCheck it out')  
plt.legend()  
plt.show()

### 堆叠图

堆叠图用于显示『部分对整体』随时间的关系。基本上类似于饼图，只是随时间而变化。

days = [1,2,3,4,5]  
sleeping = [7,8,6,11,7]  
eating = [2,3,4,3,2]  
working = [7,8,7,2,2]  
playing = [8,5,7,8,13]  
# 画一些空行，给予它们符合我们的堆叠图的相同颜色  
plt.plot([],[],color='m', label='Sleeping', linewidth=5)  
plt.plot([],[],color='c', label='Eating', linewidth=5)  
plt.plot([],[],color='r', label='Working', linewidth=5)  
plt.plot([],[],color='k', label='Playing', linewidth=5)  
plt.stackplot(days,sleeping,eating,working,playing, colors=['m','c','r','k'])  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title('Interesting Graph\nCheck it out')  
plt.legend()  
plt.show()

### 饼图

Matplotlib 会处理切片大小以及一切事情，我们只需要提供数值

slices = [7,2,2,13]  
activities = ['sleeping','eating','working','playing']  
cols = ['c','m','r','b']  
plt.pie(slices, # 切片  
 labels=activities, #  
 colors=cols, # 颜色列表  
 startangle=90, # 指定图形的起始角度  
 shadow= *True*, # 给绘图添加一个字符大小的阴影  
 explode=(0,0.1,0,0), # 拉出第二个切片  
 autopct='%1.1f%%') # 百分比放置到图表上  
plt.title('Interesting Graph\nCheck it out')  
plt.show()

## 参数

ndarray对象可以保存到磁盘文件并从磁盘文件加载。 可用的 IO 功能有：

* + load()和save()函数处理 numPy 二进制文件（带npy扩展名）
  + loadtxt()和savetxt()函数处理正常的文本文件

save()和load()函数接受一个附加的布尔参数allow\_pickles。 Python 中的pickle用于在保存到磁盘文件或从磁盘文件读取之前，对对象进行序列化和反序列化。

savetxt()和loadtxt()数接受附加的可选参数，例如页首，页尾和分隔符。

### 保存数据

**numpy.save()**

将输入数组存储在具有npy扩展名的磁盘文件中。

**numpy.savetxt()**

以简单文本文件格式存储和获取数组数据

### 加载数据

**numpy.load()**

从\*.npy重建数组

**numpy.loadtxt ()**

从文件中提取数据来图形化，使用选择的分隔符分割文件中的数据。loadtxt函数不要求文件是一个.txt文件，它可以是一个.csv或者 python 列表对象

# 将索引为0的元素存储到x列表，将索引为1的元素存储到y列表中  
x, y = np.loadtxt('E:\Spider\output.txt', delimiter=',', unpack=*True*)  
plt.plot(x,y, label='Loaded from file!')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title('Interesting Graph\nCheck it out')  
plt.legend()  
plt.show()

## basemap

### 地理绘图

basemap投影选项URL：<https://matplotlib.org/basemap/users/mapsetup.html>

坐标需要转换，其中西经和南纬坐标是负值，北纬和东经坐标是正值。

*from* mpl\_toolkits.basemap *import* Basemap  
m = Basemap(projection='mill',  
 llcrnrlat = -90, # 左下角的纬度  
 llcrnrlon = -180, # 左下角的经度  
 urcrnrlat = 90, # 右上角的纬度  
 urcrnrlon = 180, # 右上角的经度  
 resolution='l') #分辨率 c:粗糙，l：低，h：高，f：完整  
m.drawcoastlines()  
m.drawcountries(linewidth=2) # 画出国家，线宽为2  
##m.drawstates(color='b') #蓝色线条画出州  
##m.drawcounties(color='darkred') #画出国家，黑色  
m.etopo() #模型版本  
# m.bluemarble() #模型版本  
plt.title('Basemap Tutorial')  
plt.show()

1. 绘制坐标

[Matplotlib 标记文档](http://matplotlib.org/api/markers_api.html)

m = Basemap(projection='mill',  
 llcrnrlat = 10,  
 llcrnrlon = 73,  
 urcrnrlat = 54,  
 urcrnrlon = 135,  
 resolution='l') #中国区域坐标  
m.drawcoastlines()  
m.drawcountries(linewidth=2)  
m.etopo()  
xs = []  
ys = []  
beij1, beij2 = 116.405913, 39.916237  
xpt, ypt = m(beij1, beij2)  
xs.append(xpt)  
ys.append(ypt)  
m.plot(xpt, ypt, 'r\*', markersize=15)  
  
sh1, sh2 = 121.475941, 31.235435  
xpt, ypt = m(sh1, sh2)  
xs.append(xpt)  
ys.append(ypt)  
m.plot(xpt, ypt, 'r\*', markersize=15)  
  
m.plot(xs, ys, color='k', linewidth=3, label='Flight 98') # 直线连接  
m.drawgreatcircle(beij1, beij2, sh1, sh2, color='r', linewidth=3, label='Arc') # 弧线连接  
plt.legend(loc=4)  
plt.title('Basemap Tutorial')  
plt.show()

### 3D绘图

*from* mpl\_toolkits.mplot3d *import* axes3d  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')