numpy包速览

NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。NumPy 是一个运行速度非常快的数学库，主要用于数组计算，包含：

- 一个强大的N维数组对象 ndarray
- 广播功能函数
- 整合 C/C++/Fortran 代码的工具
- 线性代数、傅里叶变换、随机数生成等功能

In [1]:

```
import numpy as np
```

numpy的几个特点

- NumPy 最重要的一个特点是其 N 维数组对象 ndarray，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引
- ndarray 对象是用于存放同类型元素的多维数组
- ndarray 中每个元素在内存中都有相同存储大小的区域
- ndarray 内部由以下内容组成：
  - 一个指向数据（内存或内存映射文件中的一块数据）的指针。
  - 数据类型或 dtype，描述在数组中的固定大小值的格子。
  - 一个表示数组形状（shape）的元组，表示各维度大小的元组。
  - 一个跨度元组（stride），其中的整数指的是为了前进到当前维度下一个元素需要"跨过"的字节数。跨度可以是负数，这样会使数组在内存中后向移动，切片中 obj[::-1] 就是如此

NumPy Ndarray 对象

In [2]:

```
# 一个维度
a = np.array([1,2,3])
print(a)
```

In [3]:

```
#  多于一个维度
a = np.array([[1, 2], [3, 4]])
print(a)
```

In [4]:

```
# 修改其 dtype 参数
a = np.array([1, 2, 3], dtype = complex)
print(a)
```

NumPy 数据类型

numpy 支持的数据类型比 Python 内置的类型要多很多，基本上可以和 C 语言的数据类型对应上，其中部分类型对应为 Python 内置的类型。下表列举了常用 NumPy 基本类型：

bool_	布尔型数据类型（True 或者 False）	uint16	无符号整数（0 to 65535）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）	uint32	无符号整数（0 to 4294967295）
		uint64	无符号整数（0 to 18446744073709551615）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64	float_	float64 类型的简写
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）	float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
int8	字节（-128 to 127）	float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
int16	整数（-32768 to 32767）	float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位
int32	整数（-2147483648 to 2147483647）	complex_	complex128 类型的简写，即 128 位复数
int64	整数（-9223372036854775808 to 9223372036854775807）	complex64	复数，表示双 32 位浮点数（实数部分和虚数部分）
uint8	无符号整数（0 to 255）	complex128	复数，表示双 64 位浮点数（实数部分和虚数部分）

NumPy数组属性

NumPy 数组的维数称为秩（rank），秩就是轴的数量，即数组的维度，一维数组的秩为 1，二维数组的秩为 2，以此类推。

在 NumPy中，每一个线性的数组称为是一个轴（axis），也就是维度（dimensions）。比如说，二维数组相当于是两个嵌套的一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是 NumPy 中的轴（axis），第一个轴相当于是底层数组，第二个轴是底层数组里的数组。而轴的数量——秩，就是数组的维数。

很多时候可以声明 axis。axis=0，表示沿着第 0 轴进行操作，即对每一列进行操作；axis=1，表示沿着第1轴进行操作，即对每一行进行操作。

ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的维度，对于矩阵，n 行 m 列
ndarray.size	数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小，以字节为单位
ndarray.flags	ndarray 对象的内存信息
ndarray.real	ndarray元素的实部
ndarray.imag	ndarray 元素的虚部
ndarray.data	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

In [5]:

```
# 调整ndarray维度ndim
a = np.arange(24) # a现只有一个维度ndim=1
print(a)
b = a.reshape(2,4,3) # b 现在拥有三个维度 ndim=3
print(b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

 [[12 13 14]
  [15 16 17]
  [18 19 20]
  [21 22 23]]]
```

ndarray.itemsize：以**字节**的形式返回数组中每一个元素的大小。例如

- 一个元素类型为 float64 的数组 itemsize 属性值为 8(float64 占用 64 个 bits，每个字节长度为 8，所以 64/8，占用 8 个字节)
- 一个元素类型为 complex32 的数组 item 属性为 4（32/8）。

In [6]:

```
# 数组的 dtype 为 int8（一个字节）
x = np.array([1,2,3,4,5], dtype = np.int8)
print(x.itemsize)
#输出为1

# 数组的 dtype 现在为 float64（八个字节）
y = np.array([1,2,3,4,5], dtype = np.float64)
print(y.itemsize)
#输出为8
```

```
1
8
```

NumPy 创建数组

ndarray 数组除了可以使用底层 ndarray 构造器来创建外，也可以通过以下几种方式来创建。

- empty()
- zeros()
- ones()

In [7]:

```
x = np.empty([3,5], dtype = int)
print(x)
#输出后为随机值，因为它们未初始化
```

```
[[   0    0    0    0    0]
 [   0    0    0    0    0]
 [1188    0    0    0    0]]
```

In [8]:

```
y1 = np.zeros(5)
# zeros()默认为浮点数
print(y1)

# zeros()自定义类型
y2 = np.zeros((3,3), dtype = [('x', 'i4'), ('y', 'i4')])
print(y2)
```

```
[0. 0. 0. 0. 0.]
[[(0, 0) (0, 0) (0, 0)]
 [(0, 0) (0, 0) (0, 0)]
 [(0, 0) (0, 0) (0, 0)]]
```

In [9]:

```
z = np.ones((3,3), dtype = int)
print(z)
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

从现有数据类型创建数组

在创建numpy数组时，也可以通过numpy.asarray来讲元组等类型的数据直接转化为ndarray数组

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

- a：任意形式的输入参数，可以是，列表, 列表的元组, 元组, 元组的元组, 元组的列表，多维数组
- dtype：数据类型，可选
- order：可选，有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序

In [10]:

```
# 以下两种输出都为 [1 2 3]
x = [1,2,3]
a = np.asarray(x)
print(a)

x = (1,2,3)
a = np.asarray(x)
print(a)
```

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

从数值范围创建数组

（1）更加常用的一种方法时从一组指定的数值范围创建ndarray数组

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

- start：起始值，默认为0
- stop：终止值（不包含）
- step：步长，默认为1
- 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

In [11]:

```
# 默认类型
x = np.arange(5)
print(x)

# 设置了 dtype
x = np.arange(5, dtype = float)
print(x)

# 设置了起始值、终止值及步长
x = np.arange(10,20,2)
print(x)
```

```
[0 1 2 3 4]
[0. 1. 2. 3. 4.]
[10 12 14 16 18]
```

（2）采用numpy.linspace来创建**等差数列数组**

> np.linspace(start, stop, num=50, endpoint=True,...)

In [12]:

```
#  等差数列
a = np.linspace(1,10,10).reshape([10,1])
print(a)
```

```
[[ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]]
```

（3）也可采用numpy.logspace来创建**等比数列数组**

> np.logspace(start, stop, num=50, endpoint=True, base=10.0,...)

In [13]:

```
# 等比数列
a = np.logspace(0,9,10,base=2).reshape([10,1])
print(a)
```

```
[[  1.]
 [  2.]
 [  4.]
 [  8.]
 [ 16.]
 [ 32.]
 [ 64.]
 [128.]
 [256.]
 [512.]]
```

numpy的切片和索引

ndarray 数组可以基于 0~n 的下标进行索引，切片对象可以通过内置的 slice() 函数，并设置 start, stop 及 step 参数进行，从原数组中切割出一个新数组

In [14]:

```
a = np.arange(10)

s = slice(2,7,2) # 从索引 2 开始到索引 7 停止，间隔为2
print(a[s])

b = a[2:7:2] # 从索引 2 开始到索引 7 停止，间隔为 2
print(b)
```

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

切片中**冒号**的解释

- 如果只放置一个参数，如 [2]，将返回与该索引相对应的单个元素
- 如果为 [2:]，表示从该索引开始以后的所有项都将被提取
- 如果使用了两个参数，如 [2:7]，那么则提取两个索引(不包括停止索引)之间的项

numpy切片和索引（二维数据）

在二维数据上的切片类似于一维数组，在行列缺省的情况下，优先考虑行

In [15]:

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

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
```

In [16]:

```
print(a[0:2,2:]) # 分别在行与列进行切片
```

```
[[3]
 [5]]
```

In [17]:

```
print(a[1:]) # 缺省了列下标
```

```
[[3 4 5]
 [4 5 6]]
```

In [18]:

```
print(a[...,1]) # 第2列元素
```

```
[2 4 5]
```

In [19]:

```
print(a[1,...]) # 第2行元素
```

```
[3 4 5]
```

In [20]:

```
print(a[...,1:]) # 第2列后的元素
```

```
[[2 3]
 [4 5]
 [5 6]]
```

numpy数组广播

广播(Broadcast)是 numpy 对不同形状(shape)的数组进行数值计算的方式， 对数组的算术运算通常在相应的元素上进行

- 如果两个数组 a 和 b 形状相同，即满足 a.shape == b.shape，那么 a*b 的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同
- 当运算中的 2 个数组的形状不同时，numpy 将自动触发**广播机制**

In [21]:

```
# 数组维度相同时
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print(c)
```

```
[ 10  40  90 160]
```

In [22]:

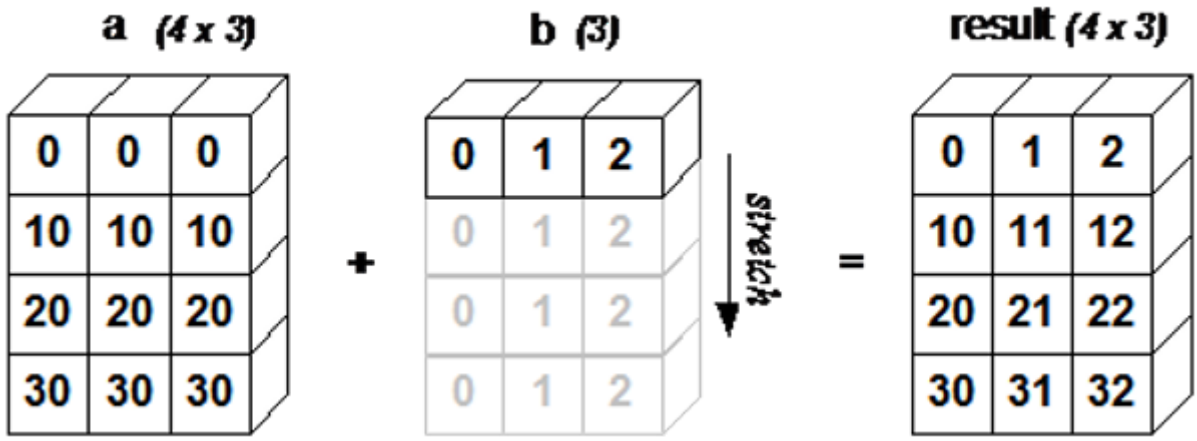
```
# 数组维度不同时
a = np.array([[ 0,  0,  0],
              [10,10,10],
              [20,20,20],
              [30,30,30]])
b = np.array([1,2,3])
c = np.array([[1],[2],[3],[4]])
print(a + b)
print(a + c)
```

```
[[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]]
[[ 1  1  1]
 [12 12 12]
 [23 23 23]
 [34 34 34]]
```

广播规则：两个数组分别比较每个维度（若其中一个数组没有当前维度则忽略），满足：

- 数组拥有相同形状。
- 当前维度的值相等。
- 当前维度的值有一个是 1。

若条件不满足，抛出 "ValueError: frames are not aligned" 异常



numpy数组操作

Numpy 中包含了一些函数用于处理数组，大概可分为以下几类：

- 修改数组形状
- 翻转数组
- 修改数组维度
- 连接数组
- 分割数组
- 数组元素的添加与删除

（1）修改数组形状

```
numpy.reshape(a, newshape, order='C')
```

In [23]:

```
a = np.arange(6)

# 第一种调用方法
b1 = np.reshape(a, [2, 3])
print(b1)

# 第二种调用方法
b2 = a.reshape([2,3])
print(b2)

# 运行时自动确定第二个维度
b3 = np.reshape(a, [2,-1])
print(b3)
```

```
[[0 1 2]
 [3 4 5]]
[[0 1 2]
 [3 4 5]]
[[0 1 2]
 [3 4 5]]
```

（2）翻转数组

以下两种翻转数组的作用等效

> numpy.transpose()

> numpy.ndarray.T

In [24]:

```
a = np.arange(12).reshape(3,4)
print(a.T)
print(np.transpose(a))
```

```
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

（3）连接数组

- concatenate：连接沿现有轴的数组序列

> numpy.concatenate((a1, a2, ...), axis=0, ...)

- stack：沿着新的轴加入一系列数组

> numpy.stack(arrays, axis=0, ...)

In [25]:

```
a = np.array([[1,2],[3,4]])
print('第一个数组：')
print(a)

b = np.array([[5,6],[7,8]])
print('第二个数组：')
print(b)

# 两个数组的维度相同
print('沿轴 0 连接两个数组：')
print(np.concatenate((a,b)))

print('沿轴 1 连接两个数组：')
print(np.concatenate((a,b),axis = 1))
```

```
第一个数组：
[[1 2]
 [3 4]]
第二个数组：
[[5 6]
 [7 8]]
沿轴 0 连接两个数组：
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
沿轴 1 连接两个数组：
[[1 2 5 6]
 [3 4 7 8]]
```

In [26]:

```
a = np.array([[1,2],[3,4]])
print('第一个数组：')
print(a)

b = np.array([[5,6],[7,8]])
print('第二个数组：')
print(b)

print('沿轴 0 堆叠两个数组：')
print(np.stack((a,b),0))

print('沿轴 1 堆叠两个数组：')
print(np.stack((a,b),1))

print('沿轴 2 堆叠两个数组：')
print(np.stack((a,b),2))
```

```
第一个数组：
[[1 2]
 [3 4]]
第二个数组：
[[5 6]
 [7 8]]
沿轴 0 堆叠两个数组：
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
沿轴 1 堆叠两个数组：
[[[1 2]
  [5 6]]

 [[3 4]
  [7 8]]]
沿轴 2 堆叠两个数组：
[[[1 5]
  [2 6]]

 [[3 7]
  [4 8]]]
```

（4）分割数组 numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：

```
numpy.split(ary, indices_or_sections, axis)
```

- ary：被分割的数组
- indices_or_sections：果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置（左开右闭）
- axis：设置沿着哪个方向进行切分，默认为 0，横向切分，即水平方向。为 1 时，纵向切分，即竖直方向

In [27]:

```
import numpy as np
a = np.arange(9)

print('第一个数组：')
print(a)

print('将数组分为三个大小相等的子数组：')
b = np.split(a,3)
print(b)

print('将数组在一维数组中表明的位置分割：')
b = np.split(a,[4,7])
print(b)
```

```
第一个数组：
[0 1 2 3 4 5 6 7 8]
将数组分为三个大小相等的子数组：
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
将数组在一维数组中表明的位置分割：
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]
```

numpy数学函数

三角函数

- 标准的三角函数：sin()、cos()、tan()。
- 反三角函数：arcsin()，arccos()，和 arctan()
- 通过 numpy.degrees() 函数将弧度转换为角度

numpy.floor() 返回小于或者等于指定表达式的最大整数，即向下取整

numpy.ceil() 返回大于或者等于指定表达式的最小整数，即向上取整

numpy算术函数

NumPy 算术函数包含简单的加减乘除:

- add()
- subtract()
- multiply()
- divide()

需要注意的是数组必须具有相同的形状或符合数组广播规则。

In [28]:

```
a = np.arange(9, dtype = np.float).reshape(3,3)
print('第一个数组：')
print(a)

print('第二个数组：')
b = np.array([10,10,10])
print(b)

print('两个数组相加：')
print(np.add(a,b))

print('两个数组相减：')
print(np.subtract(a,b))

print('两个数组相乘：')
print(np.multiply(a,b))

print('两个数组相除：')
print(np.divide(a,b))
```

```
第一个数组：
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
第二个数组：
[10 10 10]
两个数组相加：
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]
两个数组相减：
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]
两个数组相乘：
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]
两个数组相除：
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
```

numpy线性代数

NumPy提供了线性代数函数库linalg，该库包含了线性代数所需的所有功能

- dot：两个数组的点积，即元素对应相乘
- vdot：两个向量的点积
- inner：两个数组的内积
- matmul：两个数组的矩阵积
- determinant：数组的行列式
- solve：求解线性矩阵方程

- inv：计算矩阵的乘法逆矩阵

（1）numpy.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**

In [29]:

```
a = np.array([[1,2],[3,4]])
b = np.array([[10, 20],[30, 40]])
print(np.dot(a,b))
```

```
[[ 70 100]
 [150 220]]
```

（2）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

In [30]:

```
print(np.inner(np.array([1,2,3]),np.array([10,20,30])))
# 等价于 1*10+2*20+3*30
```

```
140
```

（3）numpy.matmul() 函数返回两个数组的**矩阵乘积**

In [31]:

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

```
[[4 1]
 [2 2]]
```

（4）numpy.linalg.det() 函数计算输入矩阵的**行列式**

In [32]:

```
a = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
print(a)
print(np.linalg.det(a))
print(6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2))
```

```
[[ 6  1  1]
 [ 4 -2  5]
 [ 2  8  7]]
-306.0
-306
```

（5）numpy.linalg.inv() 函数计算矩阵的乘法逆矩阵

In [33]:

```
a = [[1,2],[3,4]]
print(np.linalg.inv(a))
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

（6）numpy.linalg.solve() 函数给出了矩阵形式的线性方程的解。

```
linalg.solve(a, b)
```

- a: 线性方程组的系数矩阵
- b：线性方程组的右侧常数

考虑以下线性方程:

$$\begin{cases} x + y + z = 6 \\ 2y + 5z = -4 \\ 2x + 5y - z = 27 \end{cases}$$

In [34]:

```
a = np.array([[1,1,1],[0,2,5],[2,5,-1]])
print('数组 a：')
print(a)

ainv = np.linalg.inv(a)

print('a 的逆：')
print(ainv)

print('矩阵 b：')
b = np.array([[6],[-4],[27]])
print(b)

print('计算：A^{-1}B：')
print(np.matmul(ainv,b))

print('利用linalg.solve求解：')
print(np.linalg.solve(a,b))
```

```
数组 a：
[[ 1  1  1]
 [ 0  2  5]
 [ 2  5 -1]]
a 的逆：
[[ 1.28571429 -0.28571429 -0.14285714]
 [-0.47619048  0.14285714  0.23809524]
 [ 0.19047619  0.14285714 -0.0952381 ]]
矩阵 b：
[[ 6]
 [-4]
 [27]]
计算：A^{-1}B：
[[ 5.]
 [ 3.]
 [-2.]]
利用linalg.solve求解：
[[ 5.]
 [ 3.]
 [-2.]]
```

（7）numpy.linalg.eig()函数返回矩阵的**特征值**和**特征向量**

In [35]:

```
A = np.array([[-1,1,0],[-4,3,0],[1,0,2]])
print('打印A: \n{}'.format(A))

a, b = np.linalg.eig(A)
print('打印特征值a: \n{}'.format(a))
print('打印特征向量b: \n{}'.format(b))
```

```
打印A:
[[-1  1  0]
 [-4  3  0]
 [ 1  0  2]]
打印特征值a:
[2. 1. 1.]
打印特征向量b:
[[ 0.          0.40824829  0.40824829]
 [ 0.          0.81649658  0.81649658]
 [ 1.         -0.40824829 -0.40824829]]
```

本节作业

请利用python的numpy相关函数验证以下定理：

（1）方阵A与A^T的特征值相同

（2）若λ_i为方阵A的特征值，则：

- A的所有特征值之和等于A的对角线元素之和
- A的所有特征值之积等于A的行列式

In [ ]:

```

```