NumPy是一个关于矩阵运算的库，熟悉Matlab的都应该清楚，这个库就是让python能够进行矩阵话的操作，而不用去写循环操作。

下面对numpy中的操作进行总结。

numpy包含两种基本的数据类型：数组和矩阵。

数组(Arrays)

>>> from numpy import \*

>>> a1=array([1,1,1]) #定义一个数组

>>> a2=array([2,2,2])

>>> a1+a2 #对于元素相加

array([3, 3, 3])

>>> a1\*2 #乘一个数

array([2, 2, 2])

##

>>> a1=array([1,2,3])

>>> a1

array([1, 2, 3])

>>> a1\*\*3 #表示对数组中的每个数做平方

array([ 1, 8, 27])

##取值，注意的是它是以0为开始坐标，不matlab不同

>>> a1[1]

2

##定义多维数组

>>> a3=array([[1,2,3],[4,5,6]])

>>> a3

array([[1, 2, 3],

[4, 5, 6]])

>>> a3[0] #取出第一行的数据

array([1, 2, 3])

>>> a3[0,0] #第一行第一个数据

1

>>> a3[0][0] #也可用这种方式

1

##数组点乘，相当于matlab点乘操作

>>> a1=array([1,2,3])

>>> a2=array([4,5,6])

>>> a1\*a2

array([ 4, 10, 18])

Numpy有许多的创建数组的函数：

import numpy as np

a = np.zeros((2,2)) # Create an array of all zeros

print a # Prints "[[ 0. 0.]

# [ 0. 0.]]"

b = np.ones((1,2)) # Create an array of all ones

print b # Prints "[[ 1. 1.]]"

c = np.full((2,2), 7) # Create a constant array

print c # Prints "[[ 7. 7.]

# [ 7. 7.]]"

d = np.eye(2) # Create a 2x2 identity matrix

print d # Prints "[[ 1. 0.]

# [ 0. 1.]]"

e = np.random.random((2,2)) # Create an array filled with random values

print e # Might print "[[ 0.91940167 0.08143941]

# [ 0.68744134 0.87236687]]"

数组索引(Array indexing)

矩阵

矩阵的操作与Matlab语言有很多的相关性。

#创建矩阵

>>> m=mat([1,2,3])

>>> m

matrix([[1, 2, 3]])

#取值

>>> m[0] #取一行

matrix([[1, 2, 3]])

>>> m[0,1] #第一行，第2个数据

2

>>> m[0][1] #注意不能像数组那样取值了

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/usr/lib64/python2.7/site-packages/numpy/matrixlib/defmatrix.py", line 305, in \_\_getitem\_\_

out = N.ndarray.\_\_getitem\_\_(self, index)

IndexError: index 1 is out of bounds for axis 0 with size 1

#将Python的列表转换成NumPy的矩阵

>>> list=[1,2,3]

>>> mat(list)

matrix([[1, 2, 3]])

#矩阵相乘

>>> m1=mat([1,2,3]) #1行3列

>>> m2=mat([4,5,6])

>>> m1\*m2.T #注意左列与右行相等 m2.T为转置操作

matrix([[32]])

>>> multiply(m1,m2) #执行点乘操作，要使用函数，特别注意

matrix([[ 4, 10, 18]])

#排序

>>> m=mat([[2,5,1],[4,6,2]]) #创建2行3列矩阵

>>> m

matrix([[2, 5, 1],

[4, 6, 2]])

>>> m.sort() #对每一行进行排序

>>> m

matrix([[1, 2, 5],

[2, 4, 6]])

>>> m.shape #获得矩阵的行列数

(2, 3)

>>> m.shape[0] #获得矩阵的行数

2

>>> m.shape[1] #获得矩阵的列数

3

#索引取值

>>> m[1,:] #取得第一行的所有元素

matrix([[2, 4, 6]])

>>> m[1,0:1] #第一行第0个元素，注意左闭右开

matrix([[2]])

>>> m[1,0:3]

matrix([[2, 4, 6]])

>>> m[1,0:2]

matrix([[2, 4]])

扩展矩阵函数tile()

例如，要计算[0,0,0]到一个多维矩阵中每个点的距离，则要将[0,0,0]进行扩展。

tile(inX, (i,j)) ;i是扩展个数，j是扩展长度

实例如下：

>>>x=mat([0,0,0])

>>> x

matrix([[0, 0, 0]])

>>> tile(x,(3,1)) #即将x扩展3个，j=1,表示其列数不变

matrix([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]])

>>> tile(x,(2,2)) #x扩展2次，j=2,横向扩展

matrix([[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]])

1.协方差

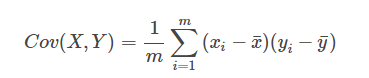
如果两个变量的变化趋势一致，也就是说如果其中一个大于自身的期望值时另外一个也大于自身的期望值，那么两个变量之间的协方差就是正值；如果两个变量的变化趋势相反，即其中一个变量大于自身的期望值时另外一个却小于自身的期望值，那么两个变量之间的协方差就是负值。

可以通俗的理解为：两个变量在变化过程中是同方向变化？还是反方向变化？同向或反向程度如何？

你变大，同时我也变大，说明两个变量是同向变化的，这时协方差就是正的;

你变大，同时我变小，说明两个变量是反向变化的，这时协方差就是负的;

从数值来看，协方差的数值越大，两个变量同向程度也就越大。反之亦然。



import numpy as np

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

b = np.array([4,3,4])

x = np.vstack((a,b))

np.cov(a, b)

np.cov(a, b, bias=True)

np.cov(x)

输出：

>>> np.cov(a, b)

array([[ 1. , 0. ],

[ 0. , 0.33333333]])

>>> np.cov(a, b, bias=True)

array([[ 0.66666667, 0. ],

[ 0. , 0.22222222]])

>>> np.cov(x)

array([[ 1. , 0. ],

[ 0. , 0.33333333]])

这里使用numpy库的cov函数来计算协方差，输出结果为一个协方差矩阵，results[i][j]表示第i个变量与第j个变量的协方差,比如np.cov(x)的结果为

>>> np.cov(x)

array([[ 1. , 0. ],

[ 0. , 0.33333333]])

x为变量矩阵，x的第 i 行表示第 i 个随机变量，这里第0个随机变量与第0个随机变量的协方差为1，第0个随机变量与第1个随机变量的协方差为0，第1个随机变量与第0个随机变量的协方差为0，第1个随机变量与第1个随机变量的协方差为0.33333333.

还有一点就是bais参数的意义，默认为False，那么计算均值的时候除以 n - 1，如果设为True，那么计算均值的时候除以n，其中n为随机变量的维度数。具体理论涉及到统计学知识，可以参考知乎回答。

numpy的cov函数使用的三点：

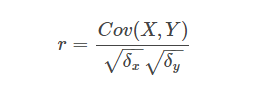
变量矩阵的一行表示一个随机变量；

bais参数控制计算时除以n-1还是n, True表示除以n，False表示除以n-1；

输出结果是一个协方差矩阵, results[i][j]表示第i个随机变量与第j个随机变量的协方差.

2. 相关系数

相关系数是用以反映变量之间相关关系密切程度的统计指标。相关系数也可以看成协方差：一种剔除了两个变量量纲影响、标准化后的特殊协方差,它消除了两个变量变化幅度的影响，而只是单纯反应两个变量每单位变化时的相似程度。



并且

|r|≤1

其中，δx,δy表示X，Y的方差。

import numpy as np

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

b = np.array([2,5,8])

x = np.vstack((a,b))

np.corrcoef(a, b)

np.corrcoef(x)

输出

>>> np.corrcoef(a, b)

array([[ 1., 1.],

[ 1., 1.]])

>>> np.corrcoef(x)

array([[ 1., 1.],

[ 1., 1.]])

numpy中函数corrcoef的用法与函数cov的用法相似，只是corrcoef中bais参数不起作用。

numpy的cov函数使用的两点：

变量矩阵的一行表示一个随机变量；

输出结果是一个相关系数矩阵, results[i][j]表示第i个随机变量与第j个随机变量的相关系数.