1 shape

矩阵的 shape 属性可以获得矩阵的形状。

获得的是一个元组

```
import numpy as np

x = np.array
    ([[0,1,2,3],[3,4,5,6],[6,7,8,9],[9,10,11,12],[12,13,14,15],[15,16,17,18],[18,19,20,21]])

#输出数组的行和列
print(x.shape)      #OUTPUT:(7,4)
#只输出行数
print(x.shape[0])
print(x.shape[-2])  #OUTPUT:7
#只输出列数
print(x.shape[1])
print(x.shape[-1])  #OUTPUT:4
```

2 numpy.percentile

numpy.percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear', keepdims=False) 计算沿指定轴的数据的第 q 个百分位数。

a：array，用来算分位数的对象，可以是多维的数组

q：介于 0-100 的 float，用来计算是几分位的参数，如四分之一位就是 25，如要算两个位置的数就 (25,75)

axis：坐标轴的方向，一维的就不用考虑了，多维的就用这个调整计算的维度方向，取值范围 0/1

out：输出数据的存放对象，参数要与预期输出有相同的形状和缓冲区长度

overwrite_input：bool，默认 False，为 True 时及计算直接在数组内存计算，计算后原数组无法保存

interpolation：取值范围'linear', 'lower', 'higher', 'midpoint', 'nearest' 默认 liner，比如取中位数，但是中位数有两个数字 6 和 7，选不同参数来调整输出

keepdims：bool, 默认 False, 为真时取中位数的那个轴将保留在结果中

百分位数

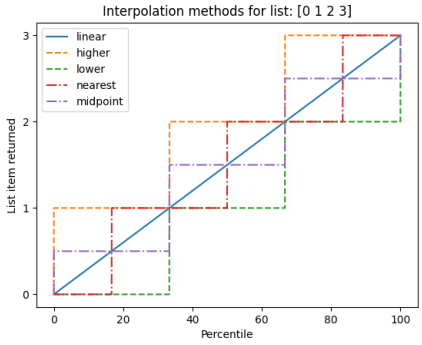
```
import numpy as np

a = np.array([[10, 7, 4],[3, 2, 1]])
print(a)
#   [[10  7  4]
```

```
#    [ 3  2  1]]

b1 = np.percentile(a, 50)
b2 = np.percentile(a, (25, 75))
print(b1)
#   3.5
print(b2)
#   [2.25 6.25]

c = np.percentile(a, 50, axis=0)
d = np.percentile(a, 50, axis=1)
print(c)
#   [6.5 4.5 2.5]
print(d)
#   [7. 2.]

out = np.zeros_like(c)
g = np.percentile(a, 50, axis=0, out=out)
print(g)
#   [6.5 4.5 2.5]

e1 = a.copy()
print(e1)
#   [[10  7  4]
#    [ 3  2  1]]
e2 = np.percentile(e1, 50, axis=1, overwrite_input=True)    #e1无法保
    存，节省内存
print(e2)
#   [7. 2.]
print(e1)
#   [[ 4  7 10]
#    [ 1  2  3]]

a = np.array([2,9,9.8,0,11,12])     # 0, 2, 9, 9.8, 11, 12
b0 = np.percentile(a, 50)
print(b0)   #9.4

b1 = np.percentile(a, 50, interpolation='linear')
print(b1)   #9.4

b2 = np.percentile(a, 50, interpolation='lower')
print(b2)   #9.0

b3 = np.percentile(a, 50, interpolation='higher')
print(b3)   #9.8
```

```
50
51 b4 = np.percentile(a, 50, interpolation='midpoint')
52 print(b4)   #9.4
53
54 b5 = np.percentile(a, 50, interpolation='nearest')
55 print(b5)   #9.0
56
57 f = np.percentile(a, 50, axis=1, keepdims=True)
58 print(f)
59 #   [[7.]
60 #    [2.]]
```

percentile 各个参数

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  a = np.arange(4)
5  p = np.linspace(0, 100, 6001)
6  ax = plt.gca()
7  lines = [
8  ('linear', None),
9  ('higher', '--'),
10 ('lower', '--'),
11 ('nearest', '-.'),
12 ('midpoint', '-.'),
13 ]
14 for interpolation, style in lines:
15 ax.plot(
16 p, np.percentile(a, p, interpolation=interpolation),
17 label=interpolation, linestyle=style)
18 ax.set(
19 title='Interpolation methods for list: ' + str(a),
20 xlabel='Percentile',
21 ylabel='List item returned',
22 yticks=a)
23 ax.legend()
24 plt.show()
```



3 numpy.concatenate

numpy 提供了 numpy.concatenate((a1,a2,...), axis=0) 函数。能够一次完成多个数组的拼接。其中 a1,a2,... 是数组类型的参数,concatenate() 效率更高，适合大规模的数据拼接。

数据的拼接

```
import numpy as np

a1 = np.array([1,2,3])
b1 = np.array([11,22,33])
c1 = np.array([44,55,66])
d = np.concatenate((a1,b1,c1),axis=0)

a2 = np.array([[1,2,3],[4,5,6]])
b2 = np.array([[11,21,31],[7,8,9]])
e = np.concatenate((a2,b2),axis=0)       #第一个轴
f = np.concatenate((a2,b2),axis=1)       #第二个轴
g = np.concatenate((a2,b2),axis=-1)      #倒数第一个轴
h = np.concatenate((a2,b2),axis=-2)      #倒数第二个轴

print(d)
#OUTPUT:
#       [ 1  2  3 11 22 33 44 55 66]
print(e)
#OUTPUT:
#       [[ 1  2  3]
#        [ 4  5  6]
#        [11 21 31]
#        [ 7  8  9]]
print(f)
#OUTPUT:
#       [[ 1  2  3 11 21 31]
#        [ 4  5  6  7  8  9]]
```

```
print(g)
#OUTPUT:
#       [[ 1  2  3 11 21 31]
#        [ 4  5  6  7  8  9]]
print(h)
#OUTPUT:
#       [[ 1  2  3]
#        [ 4  5  6]
#        [11 21 31]
#        [ 7  8  9]]
```

4 numpy.dot

两个数组的点积

数组点积

```
import numpy as np

a = np.dot(3,4)
b = [[1, 0], [0, 1]]
c = [[4, 1], [2, 2]]

print(a)
# 12
print(np.dot(b, c))
#
#  [[4 1]
#   [2 2]]
```

5 numpy.mean

numpy.mean(a, axis=None, dtype=np.float64, out=None, keepdims=<no value>,where=<no value>) 计算沿指定轴的算术平均值。返回数组元素的平均值。默认情况下，在扁平数组上取平均值，否则在指定轴上取平均值。dtype 默认为 float64 中间值和返回值用于整数输入。

求平均值

```
import numpy as np

a = np.array([[1,5,2],[3,4,2]])
b = np.mean(a, dtype=np.float32)
c = np.mean(a, where=[[True],[False]])
d = np.mean(a, axis=0)
f = np.mean(a, axis=1)
```

```
g = np.mean(a, keepdims=True)

print(a)
#OUTPUT:
#       [[1 5 2]
#        [3 4 2]]
print(b)
#OUTPUT: 2.8333333
print(c)
#OUTPUT: 2.6666667
print(d)
#OUTPUT: [2. 4.5 2.]
print(f)
#OUTPUT: [2.66666667 3.]
print(g)
#OUTPUT: [[2.83333333]]
```

6 numpy.random

random 随机函数

6.1 numpy.random.normal

random.normal(loc=0, scale=1, size=None) 从正态（高斯）分布中抽取随机样本。
loc: 为正态分布的中心。
scale: 正态分布的标准偏差。

正太分布

```
    内容...
```

6.2 numpy.random.random()

生成 (0,1）以内的随机浮点数。也可以指定生成的个数，默认为 1.

生成随机浮点数

```
import numpy as np

attentuation1 = np.random.random()
attentuation2 = np.random.random((3,))
attentuation3 = 5 * np.random.random((3,2)) - 5     #前乘的数为了加大改变概率区间，原本区间为位于半开半闭区间[0, 1)之间的浮点数。

print(attentuation1)
```

```
print(attentuation2)
print(attentuation3)

#OUTPUT:
#       0.557983928722099
#
#       [0.46067162 0.66517044 0.68304871]
#
#       [[ 2.35169967 -0.70315046]
#        [-3.01499969  1.26146337]
#        [-3.42997838 -4.03027885]]
```

6.3 numpy.random.random_sample()

numpy 模块中的 np.random.random() 和 np.random.random_sample() 函数生成的均是位于半开半闭区间 [0, 1) 之间的浮点数。

生成随机浮点数

```
import numpy as np

attentuation1 = np.random.random_sample()
attentuation2 = np.random.random_sample((3,))
attentuation3 = 5 * np.random.random_sample((3,2)) - 5  #Three-by-two
    array of random numbers from [-5, 0)

print(attentuation1)
print(attentuation2)
print(attentuation3)

#OUTPUT:
#       0.25288115492026453
#
#       [0.9322892  0.20512729 0.98083572]
#
#       [[-1.96427089 -2.774898  ]
#        [-2.51141288 -2.0688131 ]
#        [-3.99685024 -0.38281138]]
```

6.4 numpy.random.rand()

用法：numpy.random.rand(d0,d1,⋯,dn)

rand 函数根据给定维度生成 [0,1) 之间的数据，包含 0，不包含 1。dn 表示每个维度返回值为指定维度的 array

生成数据

```
import numpy as np

a = np.random.rand(2,2)
b = np.random.rand(3)

print(a)
print('\n')
print(b)
#OUTPUT:
#       [[0.95641013 0.86736593]
#        [0.36199543 0.22519705]]

#       [0.63514646 0.62497891 0.87528597]
```

6.5 numpy.random.seed()

np.random.seed() 的作用：使得随机数据可预测。当我们设置相同的 seed，每次生成的随机数相同。如果不设置 seed，则每次会生成不同的随机数

随机数据可预测

```
import numpy as np

np.random.seed(7)
a = np.random.rand(5)
np.random.seed(7)
b = np.random.rand(6)

print(a)
print(b)
#OUTPUT:
#       [0.07630829 0.77991879 0.43840923 0.72346518 0.97798951]
#       [0.07630829 0.77991879 0.43840923 0.72346518 0.97798951
    0.53849587]
```

6.6 numpy.random.choice()

numpy.random.choice(a, size=None, replace=True, p=None) 从给定的一维数组中生成随机数

参数：a 为一维数组类似数据或整数；size 为数组维度；p 为数组中的数据出现的概率; 当 replace 为 False 时，生成的随机数不能有重复的数值。a 为整数时，对应的一维数组为 np.arange(a)

从指定数组生成随机数

```python
import numpy as np

a = np.random.choice(4)
b = np.random.choice(5, 3)
c = np.random.choice(5, 3, replace=False)
d = np.random.choice(5, size=(3,2))

demo_list = ['1', '4', '6', '3', '7', '2']
e = np.random.choice(demo_list,size=(2,4))
f = np.random.choice(demo_list,size=(2,4), p=[0.1,0.3,0.3,0.1,0.1,0.1])

print(a)
print('\n')
print(b)
print('\n')
print(c)
print('\n')
print(d)
print('\n')
print(e)
print('\n')
print(f)

#OUTPUT:
#2
#
# [0 2 2]
#
# [4 3 0]
#
# [[3 3]
#  [4 2]
#  [2 4]]
#
# [['3' '6' '3' '3']
#  ['2' '2' '4' '6']]
#
# [['7' '4' '4' '7']
#  ['2' '1' '2' '2']]
```

6.7 numpy.random.uniform(low,high,size)

功能：从一个均匀分布 [low,high) 中随机采样，注意定义域是左闭右开，即包含 low，不包含 high.

参数介绍:

low: 采样下界，float 类型，默认值为 0；

high: 采样上界，float 类型，默认值为 1；

size: 输出样本数目，为 int 或元组 (tuple) 类型，例如，size=(m,n,k), 则输出 m*n*k 个样本，缺省时输出 1 个值。

返回值：数组类型，其形状和参数 size 中描述一致。

size 为整数

```
import numpy as np

s = np.random.uniform(0,1,5)
print(s)
#OUTPUT:
#       [0.56753576 0.40667623 0.10188742 0.3892323  0.51867366]
```

size 为元组

```
import numpy as np

s = np.random.uniform(0,1,[2,2])
print(s)
#OUTPUT:
#       [[0.9132833  0.06868765]
#        [0.62944324 0.07345442]]
```

7 其他

7.1 demo1

X[n,:] 是取第 1 维中下标为 n 的元素的所有值。

X[1,:] 即取第一维中下标为 1 的元素的所有值。

X[:, m:n]，即取所有数据的第 m 到 n-1 列数据，含左不含右

数组的切块

```
import numpy as np

X = np.array
    ([[0,1],[2,3],[4,5],[6,7],[8,9],[10,11],[12,13],[14,15],[16,17],[18,19]])

Y = np.array
    ([[0,1,2],[3,4,5],[6,7,8],[9,10,11],[12,13,14],[15,16,17],[18,19,20]])

print(X[:,0])
```

```
6
7  print(X[:,1])
8
9  print(X[1,:])
10
11 print(Y[:,1:3])
12
13 print(Y[...,:3])
14
15 print(Y[:3,2])
16
17 #OUTPUT:
18 #       [ 0  2  4  6  8 10 12 14 16 18]
19 #
20 #       [ 1  3  5  7  9 11 13 15 17 19]
21 #
22 # [2 3]
23 #
24 # [[ 1  2]
25 #  [ 4  5]
26 #  [ 7  8]
27 #  [10 11]
28 #  [13 14]
29 #  [16 17]
30 #  [19 20]]
31 #
32 #[[ 0  1  2]
33 # [ 3  4  5]
34 # [ 6  7  8]
35 # [ 9 10 11]
36 # [12 13 14]
37 # [15 16 17]
38 # [18 19 20]]
39 #
40 #[2 5 8]
```

7.2 demo2

tf.matmul(A,C)=np.dot(A,C)= A@C 都属于叉乘，而 tf.multiply(A,C)= A*C=A C 属于点乘

矩阵的运算

```
1  import tensorflow as tf
2  import numpy as np
3
```

```
a = np.array([[1,2],[3,4]])
b = np.array([5,6])
c = np.array([[5,6],[7,8]])
print('a:'+'\n',a)
print('b:'+'\n',b)
print('c:'+'\n',c)
#叉乘
d1=a@c
d2=tf.matmul(a,c)
d3=np.dot(a,c)
#点乘
f1=a*c
f2=tf.multiply(a,c)

print('d1:叉乘a@c' + '\n', d1)
print('d2:叉乘matmul(a,c)' + '\n', d2)
print('d3:叉乘dot(a,c)' + '\n', d3)
print('f1:点乘a*c' + '\n', f1)
print('f2:点乘multiply(a,c)' + '\n', f2)

#OUTPUT:
#       d1:叉乘a@c
#           [[19 22]
#            [43 50]]
#       d2:叉乘matmul(a,c)
#           tf.Tensor(
#               [[19 22]
#                [43 50]], shape=(2, 2), dtype=int32)
#       d3:叉乘dot(a,c)
#           [[19 22]
#            [43 50]]
#       f1:点乘a*c
#           [[ 5 12]
#            [21 32]]
#       f2:点乘multiply(a,c)
#           tf.Tensor(
#               [[ 5 12]
#                [21 32]], shape=(2, 2), dtype=int32)
```

7.3 demo3

一维数组切片跟 list 差不多 [Start:End:Step] 注意是不包含 End。

一维切片

```
import  numpy as np
```

```
2
3 a=np.array([1,2,3,4])
4 print(a[0:3:2])
5
6 #OUTPUT:
7 #       [1 3]
```

二维数组切片。

二维切片

```
1 import  numpy as np
2
3 a=np.array([[11,12,13,14],[21,22,23,24],[31,32,33,34],[41,42,43,44]])
4 print(a[0:4:2,0:4:2])
5
6 #OUTPUT:
7 #       [[11 13]
8 #        [31 33]]
```

None 增加了一个维度.

None

```
1 import numpy as np
2
3 a1 = np.array([1,2,3])
4 d = -a1[..., None]
5 f = -a1[..., None, :]
6
7 print(d)
8 print(f)
9
10 #OUTPUT:
11 #       [[-1]
12 #        [-2]
13 #        [-3]]
14 #
15 #       [[-1 -2 -3]]
```