numpy数值计算

第二章 numpy数值计算

引言

- numpy 即 Numerical Python的缩写。
- numpy是用于科学计算的一个开源的Python程序库，它为Python提供了高性能数组与矩阵运算处理能力，为数据分析提供了数据处理的基础功能。
- numpy 是 Python 中最常用也是最重要的核心库，是因为许多类似 pandas、scipy、Matplotlib、scikit-learn、tensorflow 等，都对 numpy 有一定程度的依赖。

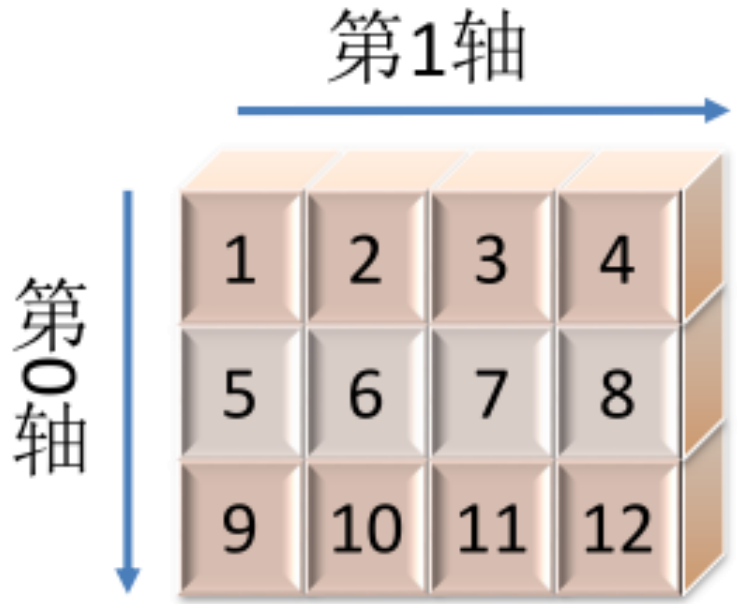
第二章 numpy数值计算

Python

2.1 numpy多维数组

2.1 numpy多维数组

numpy提供了一个名为ndarray的多维数组对象，该数组**元素具有固定大小**，即numpy只能存放同一种数据类型的对象，因此能够确定存储数组所需空间的大小，能够运用向量化运算来处理整个数组，具有较高的运算效率。



2.1 numpy多维数组

2.1.1 数组创建

1、通过array函数创建ndarray数组

np.array(object, dtype=None)

- object：传入的**序列类型数据**，若为嵌套序列，则创建多维数组
- dtype：指定数组元素类型，默认值为None，系统将从输入的数据**推断**数据的类型，可能影响系统速度，最好自行指定。

numpy 支持的数据类型比 Python 内置的类型要多很多：

类型	说明
bool	布尔型（值为True或False），占用1bit
inti	其长度取决于平台的整数（通常为int32或者int64）
int8	字节类型（取值范围从-128~127）
int16	整型（取值范围32768 ~32767）
int32	整型（取值范围为-231~231-1）
int64	整型（取值范围为-263~263-1）
uint8	无符号整型（取值范围为0~255）
uint16	无符号整型（取值范围为0~65535）
uint32	无符号整型（取值范围为0~232-1）
uint64	无符号整型()（取值范围为0~264 1）
float16	半精度浮点型:符号占用1bit，指数占用5bit，尾数占用10 bit
float32	单精度浮点型:符号占用1bit，指数占用8bit，尾数占用23bit
float64位或者float	双精度浮点型:符号占用1 bit，指数占用11 bit，尾数占用52 bit
complex64	复数类型，由两个32位浮点数(实部和虚部)表示
complex128或者complex	复数类型，由两个64位浮点数(实部和虚部)表示

[示例] 列表作为array参数创建数组

```
import numpy as np   # 导入numpy库，取别名为np


lst1 = [1,2,3,4,5,6]   # 定义一个列表
a1=np.array(lst1)    # 创建一维数组
print(a1)    # 打印一维数组


lst2 = [[1,2,3],[4,5,6]]    # 定义一个嵌套列表
a2 = np.array(lst2)   # 创建二维数组
print(a2)  # 打印二维数组
```

```
[1 2 3 4 5 6]

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

[示例] 根据最高数据类型推断数组数据类型

```
import numpy as np   # 导入numpy库，取别名为np


lst1 = [1., 2, 3, 4, 5, 6]   # 定义一个列表
a1=np.array(lst1)    # 创建一维数组
print(a1)    # 向上推断类型为float
```

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

[示例] 创建数组时指定数据类型

```
import numpy as np


lst1 = [1, 2, 3, 4, 5, 6]
a1=np.array(lst1, dtype='float32')    # 若指定为基础类型，可不加引号
print(a1)
```

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

[示例] 创建数组后改变数据类型，用astype函数。

```
import numpy as np


lst1 = [1, 2, 3, 4, 5, 6]
a1 = np.array(lst1, dtype=float)
print(a1)
a1 = a1.astype(int)    # astype在副本上完成，要改变原数组，需重新赋值
print(a1)
```

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

[示例] 用astype将浮点数数组转为整数数组时，小数部分直接截断。

```
import numpy as np

lst1 = [1.9, 2.5, 3.2, 4.4, 5.25, 6.78]
a1 = np.array(lst1)
a2 = a1.astype(int)      # 截断小数位
print(a2)
a3 = a1.round(0).astype(int)   # 保留0位小数（四舍五入），再转为整数
print(a3)
```

```
[1 2 3 4 5 6]
[2 2 3 4 5 7]
```

[示例] 字符串数组转数值型数组，数组元素必须是数值型字符串才能转。

```
import numpy as np


lst1 = ['1.9', '2.5', '3.2', '4.4', '5.25', '6.78']
a1 = np.array(lst1)
# a1 = a1.astype(int)   # 浮点字符串数组不能转直接转整数数组
a1 = a1.astype(float)
print(a1)
```

```
[1.9   2.5   3.2   4.4   5.25  6.78]
```

2、通过函数创建特殊数组

函数名	功能
ones(shape, dtype, order)	创建全1数组
ones_like(a)	按照a数组的形状和元素类型创建全1数组
zeros(shape, dtype, order)	创建全0数组
zeros_like(a)	按照a数组的形状和元素类型创建全0数组
eye(N,M=None,k,dtype,order)	创建对角线为1的对角矩阵
identity(N,dtype=None)	创建单位矩阵
diag(v,k=0)	以一维数组的形式返回方阵的对角线元素，或将一维数组转换成方阵，两种功能角色转变取决于参数v

参数名	数据类型	作用
shape	整数或者整数组成的元组	数组的形状，例如：(2, 3)或者2
dtype	数值类型，可选参数	指定数组的数值类型，默认float
order	{‘C’，‘F’}，可选参数	是否在内存中以C或Fortran(行或列)顺序存储多维数据，默认为C

[示例] 创建全1数组　ones(shape, dtype=float, order='C')

```
import numpy as np


a = np.ones((4,2))
print(a)
```

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

[示例] 创建全0数组　zeros(shape，dtype=float，order = 'C')

```
import numpy as np


a = np.zeros((3,5))
print(a)
```

```
[[0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.]]
```

[示例] 创建对角为1的矩阵　eye(N,M=None, k=0, dtype=float)

```
import numpy as np
a = np.eye(3)
print(a)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
import numpy as np
a = np.eye(3, k=-1)
print(a)
```

```
[[0. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
```

```
import numpy as np
a = np.eye(3, k=1)
print(a)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

```
import numpy as np
a = np.eye(3, k=2)
print(a)
```

```
[[0. 0. 1.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
import numpy as np
a = np.eye(3,4)
print(a)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

[示例] 创建单位矩阵　identity(n, dtype=float)

```
import numpy as np


a = np.identity(3)
print(a)
```

```
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
```

[示例] 创建对角为指定值的矩阵或获得二维数组的对角线　diag(v,k=0)

```
import numpy as np
a1 = np.array([1,2,3,4])
# 一维数组为参数，是创建以之为对角线的方阵
a2 = np.diag(a1)
print(a2)
```

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

```
import numpy as np
a1 = np.array([[1,2,3],[4,5,6]])
print(a1)
# 二维数组为参数，是获得该数组的对角线元素数组
a2 = np.diag(a1)
print(a2)
```

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

[1 5]
```

[示例] 创建仿矩阵(与指定数组具有相同形状和数据类型的数组)

```
import numpy as np

a1 = np.eye(3)
a2 = np.ones_like(a1)
print(a2)
```

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

```
import numpy as np

a1 = np.array([[1,2,3],[4,5,6]])
a2 = np.zeros_like(a1)
print(a2)
```

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

[示例] 创建填充矩阵(原地填充，不改变原数组的形状和类型)

```
import numpy as np


a = np.array([[1,2,3],[4,5,6]])   # 原数组元素类型为int
a.fill(5.0)   # 参数类型为float
print(a)
```

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

3、从数值范围创建数组

从数值范围创建数组的numpy函数有三个：arange、linspace和logspace函数。

（1）arange函数

函数arange根据start与stop指定的范围以及step设定的步长，生成一个**一维数组**，函数格式如下所示：

numpy.arange(start=0, stop, step=1, dtype=None)

如果没有提供dtype，则会**使用start和stop中较高级别类型。**

[示例] 从数值范围创建数组

```
import numpy as np

a1 = np.arange(10)
print('a1=', a1)

a2 = np.arange(0.,10, 2)      #0.和10中较高级别的类型是float
print('a2=', a2)

a3 = np.arange(10, 0, -2, dtype='float')
print('a3=', a3)
```

```
a1= [0 1 2 3 4 5 6 7 8 9]
a2= [0. 2. 4. 6. 8.]
a3= [10.  8.  6.  4.  2.]
```

（2）linspace

linspace 函数用于创建一个**一维数组**，数组是一个**等差数列**构成的，其格式如下：

np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

参数	描述
start	序列的起始值（没有默认值，不能省）
stop	序列的终止值
num	要生成的等步长的样本数量，默认为50
endpoint	该值为 True 时，数列中包含stop值，反之不包含，默认是True。
retstep	如果为 True 时，生成的数组中会显示间距，反之不显示。
dtype	ndarray 的数据类型

[示例] 从数值范围创建数组linspace

```
import numpy as np

a1= np.linspace(1,10,5)
print('a1=', a1)

a2 = np.linspace(1,10, 5, retstep=True)   # 返回值包含公差
print('a2=', a2)
```

```
a1= [1.  3.25  5.5  7.75  10.]
a2= (array([1.,  3.25,  5.5 ,  7.75, 10. ]), 2.25)
```

（3）logspace函数

创建开始点为start，结束点为stop的，以base为底的幂组成的，num个数的**数列**，其格式如下所示：

np.logspace(start, stop, num=50, endpoint=True, **base=10.0**, dtype=None)

参数	描述
start	序列的起始值
stop	序列的终止值。如果endpoint为True，该值包含于数列中
num	要生成的等步长的样本数量，默认为50
endpoint	该值为 Ture 时，数列中中包含stop值，反之不包含，默认是True
base	底数
dtype	ndarray 的数据类型

[示例]从数值范围创建数组

```
import numpy as np


a1= np.logspace(1, 9, 5)
print('a1=', a1)


a2 = np.logspace(1, 9, 5, base=2)
print('a2=',a2)
```

```
a1= [1.e+01  1.e+03  1.e+05  1.e+07  1.e+09]
a2= [  2.    8.   32.  128.  512.]
```

4、随机数数组

随机数对数据分析、统计、机器学习等领域有着重要的作用，可以运用随机数模拟均匀概率事件、统计抽样等，大大方便了数据的生成。

函数	使用说明
rand	产生均匀分布的样本值
randint	给定范围内取随机整数
randn	产生正态分布的样本值
seed	随机数种子
permutation	对一个序列随机排序，不改变原数组
shuffle	对一个序列随机排序，改变原数组
uniform(low,high,size)	产生具有均匀分布的数组，low起始值，high结束值，size形状
normal(loc,scale,size)	产生具有正态分布的数组，loc表示均值，scale表示标准差
poisson(lam,size)	产生具有泊松分布的数组，lam表示随机事件发生率

概率分布知识回顾

（1）随机变量

离散数据（例如抛硬币的结果），连续数据（例如时间）。

（2）分布

数据在统计图中的形状。

（3）概率分布

概率分布就是在统计图中表示概率，横轴是数据的值，纵轴是横轴上对应数据值的概率。

根据数据类型的不同，概率分布分为两种：离散概率分布，连续概率分布。

（4）常见概率分布

离散型分布：二项分布、多项分布、伯努利分布、**泊松分布**

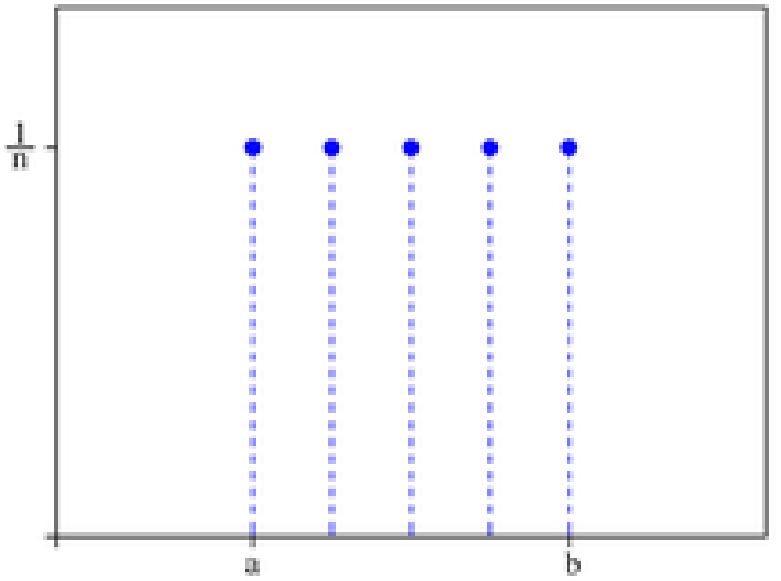
连续型分布：**均匀分布**、**正态分布**、指数分布、偏态分布

均匀分布

在现实世界中，抽奖、抽签等活动，靠人为手段难以产生公平的结果。通过生成符合均匀分布的随机数，可以实现公平性。

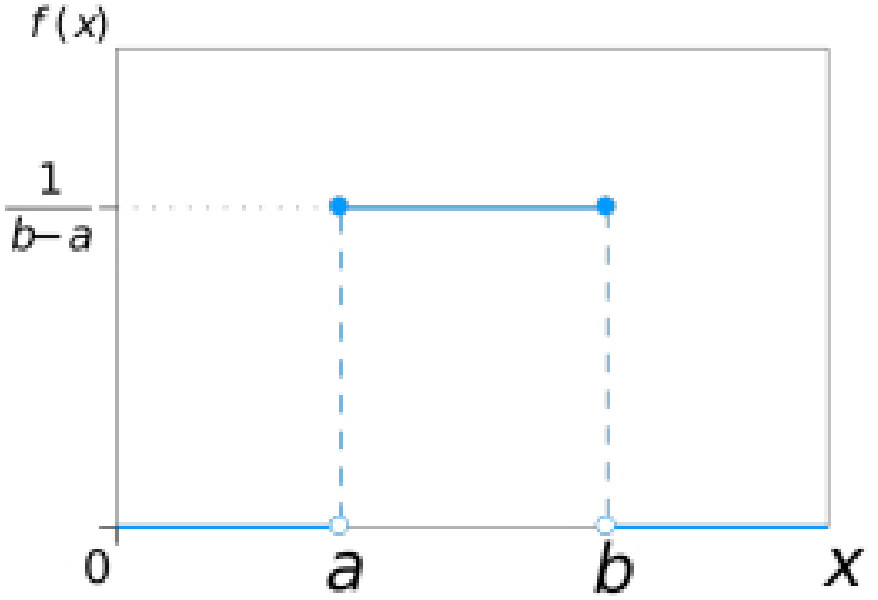
1) 离散随机变量的均匀分布：假设 X 有 k 个取值：x1, x2, ..., xk 则均匀分布的概率密度函数为：

$$p(X = x_i) = \frac{1}{k}, \quad i = 1, 2, \cdots, k$$



2) 连续随机变量的均匀分布：假设 X 在 [a, b] 上均匀分布，则其概率密度函数为：

$$p(X = x) = \begin{cases} 0, & x \notin [a, b] \\ \frac{1}{b-a}, & x \in [a, b] \end{cases}$$



正态分布

正态分布也称为高斯分布，是一种常见的**连续性**概率分布。

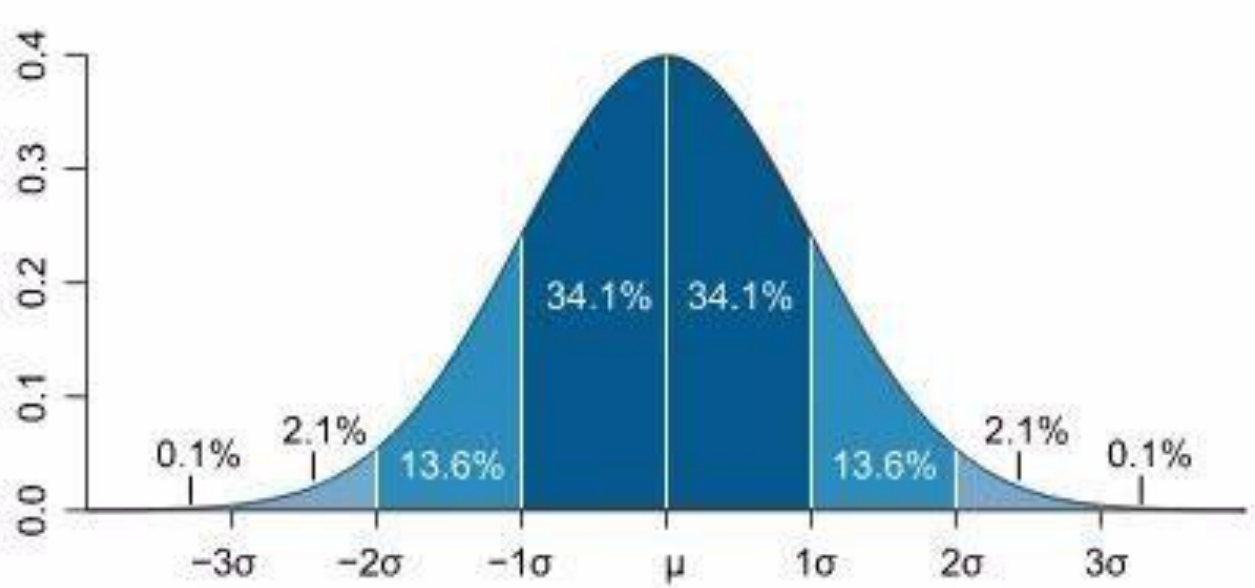
正态分布曲线呈钟型，故常称之为钟形曲线。

随机变量x服从均值为μ，方差为δ^2的正态分布记为x~N(μ，δ^2)，其概率密度函数如下：

$$f(x)=\frac{1}{\delta\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\delta^2}}$$

标准正态分布是正态分布的一种特殊形式，即当μ=0，δ=1时，称此正态分布为标准正态分布，记为x~N(0,1)。

$$f(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$$



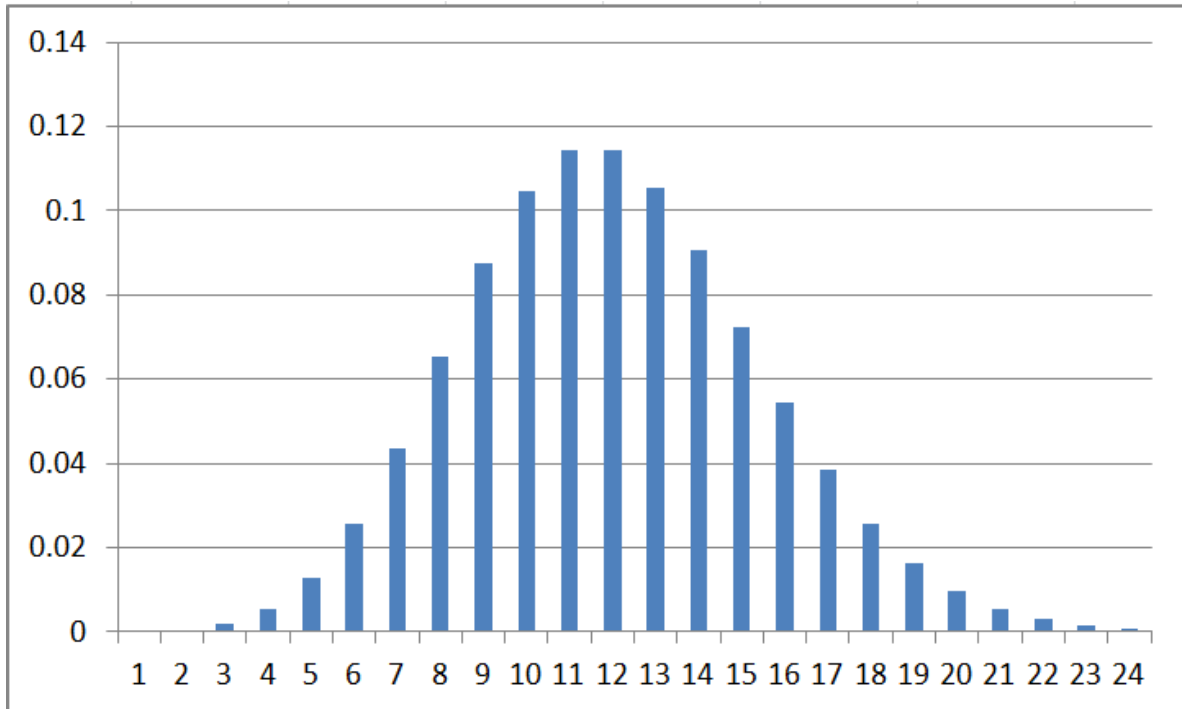
泊松分布

- 泊松分布的参数λ是单位时间(或单位面积)内随机事件的平均发生率。
- 泊松分布适合于描述单位时间内随机事件发生的次数。如某一服务设施在一定时间内到达的人数，电话交换机接到呼叫的次数，汽车站台的候客人数，机器出现的故障数，自然灾害发生的次数等等。
- 概率密度函数如下：

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

- k 是事件发生的次数（$k = 0, 1, 2, \ldots$）。
- λ 是单位时间（或单位空间）内事件发生的平均次数（$\lambda > 0$）。

可以看到，在事件平均次数附近，事件的发生概率最高，然后向两边下降，即变得越大和越小都不太可能。



$\boldsymbol{\lambda = 12}$

（1）rand函数

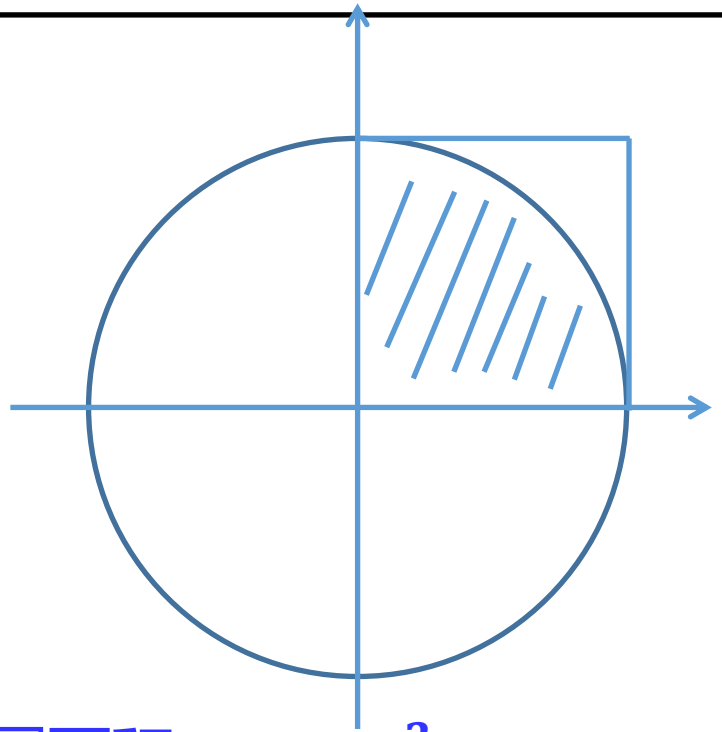
产生一个给定形状的数组，数组中的值服从**[0, 1)之间的均匀分布**，其格式如下：

numpy.random.rand(d0, d1, ..., dn)

- 参数d0, d1, ..., dn表示每一维的长度（size），为int型，可选。
- 若没有参数则返回**一个符合均匀分布的float型随机数。**

通过均匀分布，随机生成很多坐标在[0,1]上的点，构成一个矩形。
通过产生大量的随机数，算出概率P，那么圆周率就是4P。

```
from numpy.random import rand
darts = 100000
hits = 0.0
for i in range(1, darts + 1):
    # 生成两个[0,1)之间均匀分布的随机数
    x, y = rand(), rand()
    dist = pow(x ** 2 + y ** 2, 0.5)
    if dist <= 1.0:  # 若点离原点的距离<=半径1
        hits = hits + 1   # 命中+1
pi = 4 * (hits / darts)
print("pi={}".format(pi))
```



$$\frac{\text{圆面积}}{\text{正方形面积}} = \frac{\frac{\pi r^2}{4}}{r^2} = \frac{\pi}{4} = P$$

P为随机点落在1/4圆内的概率

pi=3.14016

（2）uniform函数

函数uniform返回一个在区间**[low, high)中均匀分布**的数组，其格式如下：

uniform(low=0.0, high=1.0, size=None)

- 参数low, high是float型，low的默认值为0.0，high的默认值为1.0；
- size指定array的形状，**是int型数值**（一维数组长度）或**int型元组**（多维数组形状），不指定该参数则返回**一个**服从均匀分布的**随机数**。

```
import numpy as np


a = np.random.uniform(3, 5, (2,3,2))
print(np.round(a,2))
```

```
[[[3.11 4.43]
  [4.13 3.17]
  [3.18 3.44]]

 [[4.71 4.63]
  [4.25 3.31]
  [4.46 3.7 ]]]
```

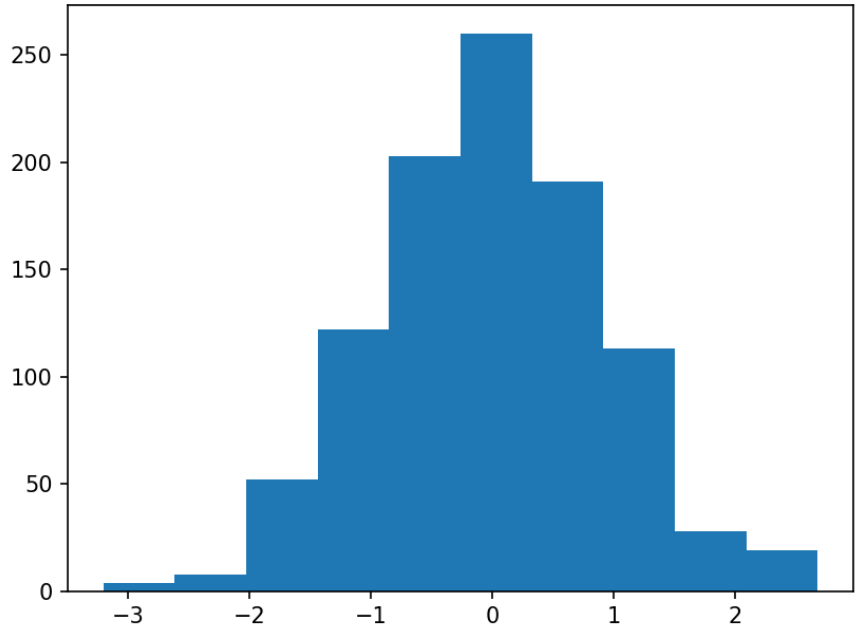
（3）randn函数

返回一个指定形状的数组，数组中的值服从**标准正态(normal)分布**，格式如下：

numpy.random.randn(d0, d1, ..., dn)

- 参数：d0, d1, ..., dn表示对应维度的大小，int型，可选。
- 如果没有参数，则返回一个服从标准正态分布的float型随机数。

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(10)
a= np.random.randn(1000)
plt.hist(a)   # 频数直方图
plt.show()
```



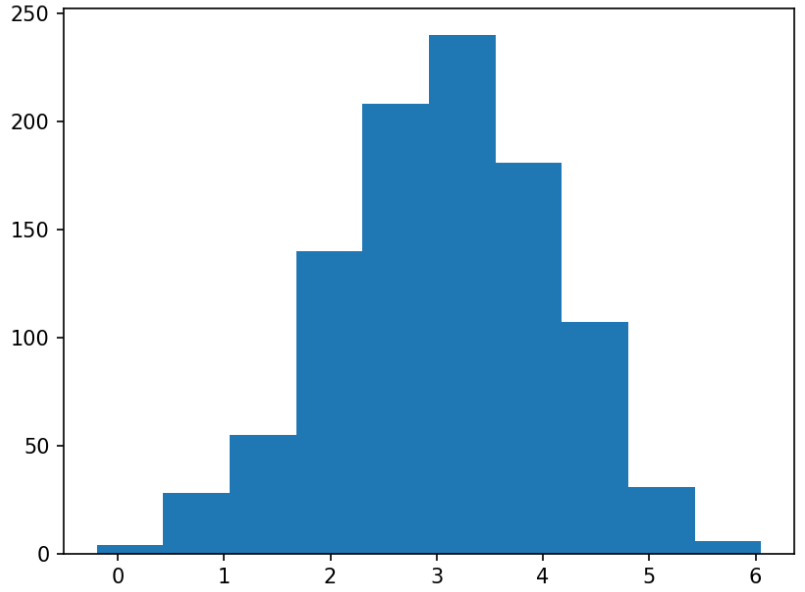
（4）normal函数

生成**指定正态分布**的概率密度随机数，其格式如下：

numpy.random.normal(loc=0.0, scale=1.0, size=None)

- loc（μ）：float，此概率分布的均值
- scale（δ）：float， 此概率分布的标准差
- size：int or tuple of ints， 输出的shape，默认为None，只输出一个值。

```
import numpy as np
import matplotlib.pyplot as plt
a=np.random.normal(3,1,size=1000)
plt.hist(a)
plt.show()
```



（5）randint函数

函数randint生成一个在区间[low, high)中**离散均匀分布**的整数数组，格式如下：

numpy.random.randint(low, high=None, size=None, dtype='l')

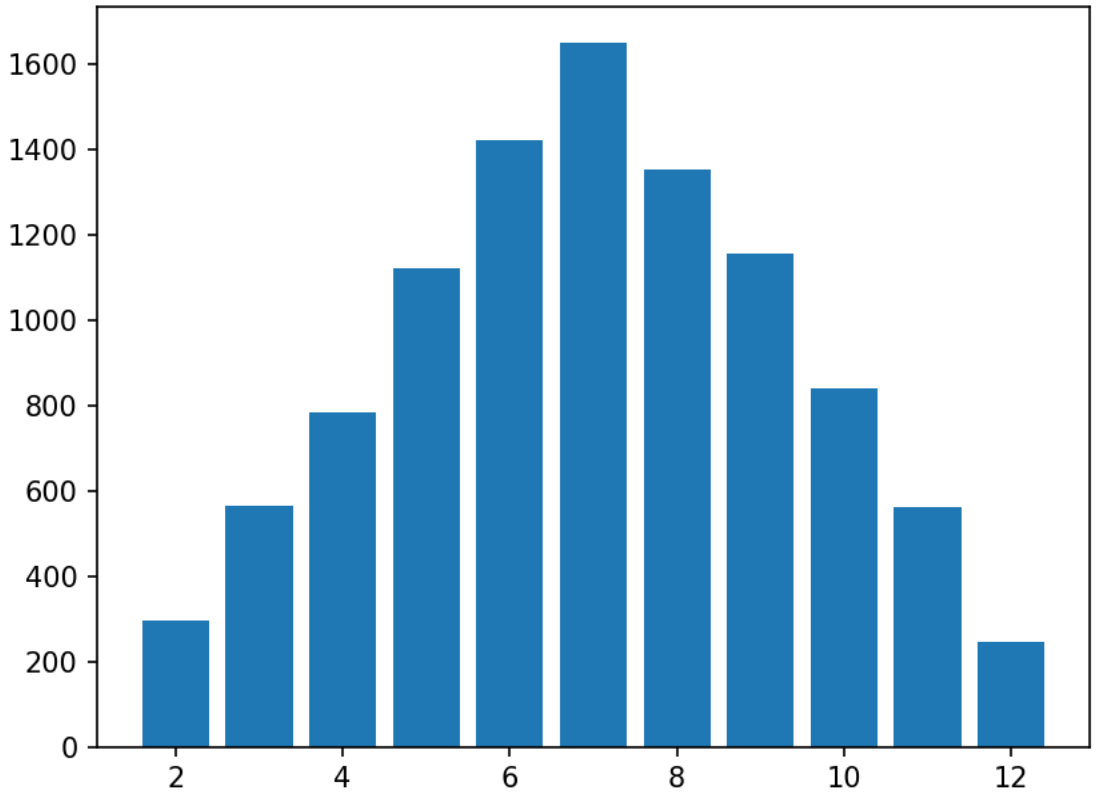
- low, high：int型，指定抽样区间[low, high)，**如果没有写参数high的值，则指定抽样区间为[0,low)。**
- size：int型或int型的元组，指定形状。
- dypte：可选参数，指定**整数数据类型**，比如int、int64(i8)等。
- 返回值：如果指定了size，则返回一个int型的ndarray对象，否则返回一个服从该分布的int型随机数。

[示例] 均匀分布之和服从正态分布

```
import numpy.random as np
import matplotlib.pyplot as plt

d = {}
for i in range(10000):
    t1 = np.randint(1,7)
    t2 = np.randint(1,7)
    sum = t1 + t2
    d[sum] = d.get(sum,0) + 1

x = d.keys()
y = d.values()
plt.bar(x,y)
plt.show()
```



（6）permutation函数

随机排列一个序列，或者数组。

numpy.random.permutation(x)

- 如果x是整数，则随机排列**np.arange(x)**
- 如果x是一维数组或序列，对其**复制**之后再打乱其元素。
- 如果x是多维数组，则**沿最小维索引**随机排列数组。

[示例] permutation函数

```
import numpy as np
a = np.random.permutation(5)  # 随机排列np.arange(5)
print('a=', a)
b = np.array([1,3,5,7,9])
b = np.random.permutation(b)
print('b=\n', b)
c = np.array([[1,2,3],[4,5,6],[7,8,9]])
c = np.random.permutation(c)  # 对第0维索引随机排列
print("c=\n", c)
```

```
a= [2 1 4 0 3]

b=
 [3 9 7 5 1]

c=
 [[7 8 9]
 [4 5 6]
 [1 2 3]]
```

（7）shuffle函数

对numpy的数组进行**原地**重新随机排序，如果是多维数组，沿最小维进行随机排序，返回值为None。

numpy. random. shuffle(x)

shuffle与permutation都是对原来的数组进行重新洗牌（即随机打乱原来的元素顺序）。
区别在于shuffle直接在原来的数组上进行操作，改变原来数组的顺序，无返回值；
而permutation不直接在原来的数组上进行操作，而是返回一个新的打乱顺序的数组，并不改变原来的数组。
此外，shuffle不能传入int参数。

[示例] shuffle函数

```
import numpy as np


b = np.array([1,3,5,7,9])
np.random.shuffle(b)
print('b=\n', b)


c = np.array([[1,2,3],[4,5,6],[7,8,9]])
np.random.shuffle(c)
print("c=\n", c)
```

```
b=
 [7 9 3 1 5]

c=
 [[1 2 3]
  [7 8 9]
  [4 5 6]]
```

2.1 numpy多维数组

2.1.2 数组对象属性

ndarray对象具有很多有用的属性，如表所示。

属性	使用说明
ndim	数组维度
shape	数组形状
size	数组元素个数
dtype	数组元素的数据类型
itemsize	数组中每个元素所占字节数
nbytes	存储整个数组元素所需字节，nbytes=itemsize*size
T	数组的转置
flat	返回一个numpy.flatiter对象，可以使用flat的迭代器来遍历数组

[示例] array的属性

```
import numpy as np
a = np.array( [ [0,1,2,3], [4,5,6,7], [8,9,10,11] ] )
print("a=\n",a)
print("a.ndim=", a.ndim)
print("a.shape=", a.shape)
print("a.size=", a.size)
print("a.dtype=", a.dtype)
print("a.itemsize=", a.itemsize)
print("a.nbytes=", a.nbytes)
```

```
a=
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a.ndim= 2
a.shape= (3, 4)
a.size= 12
a.dtype= int32
a.itemsize= 4
a.nbytes= 48
```

[示例] array的转置

```
import numpy as np


a = np.array([[1,2,3],[4,5,6]])
b = a.T
print("a=\n", a)
print("b=\n", b)
```

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

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

[示例] array的flat(多维数组扁平化为一维迭代对象)

```
import numpy as np


a = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
print("a=", a)
for x in a:   # 这样遍历只能按第一维取数
    print("x=", x)


for y in a.flat:  # 这样遍历才能取每一个元素
    print(y,end=' ')
```

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

x= [0 1 2 3]

x= [4 5 6 7]

x= [ 8  9 10 11]

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

[示例] array的flat

```
import numpy as np
a = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
print(a.flat)          # 返回一个一维扁平化对象
print(a.flat[:])       # 利用切片将扁平化对象全部元素输出，也可转list输出
a.flat[2] = 1          # 将a.flat的第2个元素替换为1
print("a=\n", a)      # a的元素改变，维度未变
```

```
<numpy.flatiter object at 0x00000147B2F4AAA0>
[ 0  1  2  3  4  5  6  7  8  9 10 11]
a=
 [[ 0  1  1  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

END