

Chapter 3 – NumPy金融场景

Frank Ziwei Zhang
School of Finance



上海對外經貿大學
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

Contents

Q1

一个投资案例

Q2

N 维数组

Q3

数组索引、切片和排序

Q4

主要数组运算

Q5

*NumPy*随机数

3.1 从一个投资案例开始

3.1 从一个投资案例开始

【例3-1】假定某投资者拥有一个投资组合，该组合的初始投资金额是1亿元，组合中配置了4只在A股市场上市的股票，分别是工商银行、中国石油、宝钢股份以及上汽集团，配置的比例分别是15%、20%、25%以及40%。与此同时，表3-1描绘了2018年9月3日至9月7日这5个交易日中相关股票的涨跌幅情况，投资者希望通过Python快速计算这5个交易日整个投资组合的涨跌幅情况。

股票简称	9月3日	9月4日	9月5日	9月6日	9月7日
中国石油	0.3731%	2.1066%	-0.4854%	0.6098%	-0.6060%
工商银行	-0.1838%	0.1842%	-1.6544%	-0.3738%	0.3752%
上汽集团	-0.3087%	-0.0344%	-3.3391%	0.7123%	0.4597%
宝钢股份	-2.4112%	1.1704%	-2.9563%	-1.4570%	1.6129%

3.1 NumPy是个啥

NumPy的前身Numeric最早是由Jim Hugunin与其他协作者共同开发。2005年，Travis Oliphant在Numeric中结合了另一个同性质的程序库Numarray的特色，并加入了其他扩展程序而开发了NumPy。

NumPy是运用Python进行科学计算的基础包（模块），可以定义任意数据类型，它的内容包括：

- （1）强大的N维数组对象；
- （2）复杂的广播（broadcasting）功能；
- （3）用于集成C、C++和Fortran代码的工具；
- （4）实用的线性代数、傅里叶变换和随机数功能等。

NumPy可以用作通用数据的高效多维容器（multi-dimensional container），这使NumPy能够与各种数据库无缝集成。

```
import numpy as np
np.__version__
```

3.2 N维数组

3.2.1 数组结构

NumPy最显著的特征在于它的数据结构是运用了数组。数组（`array`）和前面第2章的列表有相似之处，但是数组是可以定义维度的，因此数组的全称是N维数组（`ndarray`）。数组适合做数学代数运算，其结构如下：

一维数组 `np.array`（一个数列）

二维数组 `np.array`（[数列1， 数列2， ...数列m]）

注意，小括号中的数列可以是一个数列（相当于 $1 \times n$ 的向量），也可以是由m个数列作为元素所组成的一个数列（相当于 $m \times n$ 的矩阵）。当然，也可以有三维甚至是更高维度的组。

3.2.1 数组结构

【例3-2】根据例3-1中的信息，将4只股票的配置比例以一维数组方式直接在Python中进行输入，具体的代码如下：

```
weight = np.array([0.15,0.2,0.25,0.4])  
type(weight) #显示变量类型  
weight.shape #显示数组的结构维度
```


3.2.1 数组结构

【例3-3】根据例3-1中的信息，将这4只股票涨跌幅以数组方式在python中进行输入，具体的代码如下：

```
stock_return=np.array([[0.003731,      0.021066, -0.004854, 0.006098, -0.006060,],  
[-0.001838,      0.001842, -0.016544, -0.003738, 0.003752,],  
[-0.003087,      -0.000344, -0.033391, 0.007123, 0.004597,],  
[-0.024112,      0.011704, -0.029563, -0.014570, 0.016129]])  
stock_return.shape #显示维度结构
```

stock_return - NumPy object array

	0	1	2	3	4
0	0.003731	0.021066	-0.0048...	0.006098	-0.00606
1	-0.0018...	0.001842	-0.0165...	-0.0037...	0.003752
2	-0.0030...	-0.0003...	-0.0333...	0.007123	0.004597
3	-0.0241...	0.011704	-0.0295...	-0.01457	0.016129

3.2.1 数组结构

【例3-4】根据例3-1中的信息，将4只股票的配置比例先以列表的方式在Python中输入，然后用array函数将列表转为一维数组，具体的代码如下：

```
weight_list = [0.15,0.2,0.25,0.4]
weight_array = np.array(weight_list)
```

【例3-5】根据例3-1中的信息，将这4只股票涨跌幅先以列表方式在 Python 中输入，然后用array和reshape函数将列表变为二维数组，具体的代码如下：

```
return_list = [0.003731,      0.021066, -0.004854, 0.006098, -0.006060,      0.001838,
               0.001842, -0.016544, -0.003738, 0.003752, -0.003087, -0.000344, -0.033391,
               0.007123, 0.004597, -0.024112, 0.011704, -0.029563, -0.014570, 0.016129]
retrun_array = np.array(return_list)
retrun_array = retrun_array.reshape(4,5)
```

当然，也可以用rave函数将多维数组降维至一维数组，具体的代码如下：

```
return_array1 = retrun_array.ravel()
```

3.2.1 数组结构

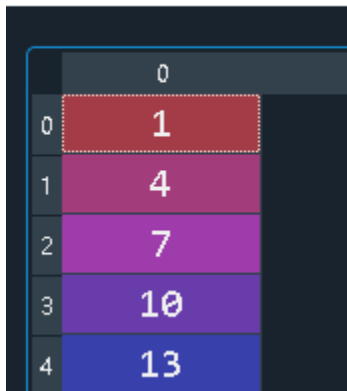
函数名	功能	Python的演示
ndim	查看数组的维度	<pre>weight_array.ndim Out[25]: 1 return_array.ndim Out[26]: 2</pre>
size	查看数组的元素数量	<pre>weight_array.size Out[27]: 4 return_array.size Out[28]: 20</pre>
dtype	查看数组的元素类型	<pre>weight_array.dtype Out[29]: dtype('float64') return_array.dtype Out[30]: dtype('float64')</pre>

3.2.2 数组的便捷生成

【例3-6】通过 NumPy快速生成包含0-9整数的数组以及1-14且步长为3的数组，具体的代码如下：

```
a = np.arange(10)
b = np.arange(1,15,3)
```

b - NumPy object array



	0
0	1
1	4
2	7
3	10
4	13

3.2.2 数组的便捷生成

【例3-7】通过NumPy生成一个1-100并且元素个数为51的等差序列并且以数组形式存放，具体的代码如下：

```
c = np.linspace(1,100,51)
```

Out[33]:

```
array([ 1. ,  2.98,  4.96,  6.94,  8.92, 10.9 , 12.88, 14.86,  
       16.84, 18.82, 20.8 , 22.78, 24.76, 26.74, 28.72, 30.7 ,  
       32.68, 34.66, 36.64, 38.62, 40.6 , 42.58, 44.56, 46.54,  
       48.52, 50.5 , 52.48, 54.46, 56.44, 58.42, 60.4 , 62.38,  
       64.36, 66.34, 68.32, 70.3 , 72.28, 74.26, 76.24, 78.22,  
       80.2 , 82.18, 84.16, 86.14, 88.12, 90.1 , 92.08, 94.06,  
       96.04, 98.02, 100. ])
```

3.2.2 数组的便捷生成

【例3-8】创建一个一维的零数组，数组的元素个数为5，具体的代码如下：

```
zeros_array1 = np.zeros(5)
```

【例3-9】创建一个二维的零数组，并且是 3×4 形式的数组，具体的代码如下：

```
zeros_array2 = np.zeros((3,4))
```

【例3-10】创建与前面例3-4、例3-5中已生成的weight_array，return_array同维度、同形状的零数组，具体的代码如下：

```
zero_weight = np.zeros_like(weight_array)  
zero_return = np.zeros_like(return_array)
```

3.2.2 数组的便捷生成

【例3-11】创建与前面例3-4、例3-5中已生成的 `weight_array`、`return_array`同维度同形状并且元素均为1的数组，具体的代码如下：

```
one_weight = np.ones_like(weight_array)
one_return = np.ones_like(return_array)
```

one_return - NumPy object array

	0	1	2	3	4
0	1	1	1	1	1
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1

3.2.2 数组的便捷生成

【例3-12】在 NumPy中，快速创建一个 5×5 的单位矩阵，具体的代码如下：

```
I = np.eye(5)
```

I - NumPy object array

	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

3.3 数组索引、切片和排序

【例3-13】沿用例3-1中的信息，投资者希望找到工商银行这只股票在2018年9月5日的涨跌幅，对应于数组中第2行第3列，具体的代码如下：

```
return_array[1,2]
```

【例3-14】沿用例3-1中的信息，投资者希望找出涨跌幅低于-1%的数据所在数组中的索引值，具体的代码如下：

```
np.where(return_array<-0.01)
```

```
Out[45]: (array([1, 2, 3, 3, 3], dtype=int64), array([2, 2, 0, 2, 3], dtype=int64))
```

这里需要说明一下，由于数组`return_array`是一个二维数组，因此对应的索引值必然应着两个数值，一个代表第几行，另一个代表第几列。因此，在输出结果中，第1个数组代表行的索引值，第2个数组代表列的索引值。

【例3-15】沿用例3-1中的信息，投资者希望提取上汽集团、宝钢股份在2018年9月4日至9月6日的涨跌幅数据，也就是提取第3行、第4行中第2~4列的数据，具体的代码如下：

```
return_array[2:,1:4]
```

Out[46]:

```
array([[ -0.000344, -0.033391,  0.007123],  
       [ 0.011704, -0.029563, -0.01457 ]])
```

注意，在中括号中，“2:”代表选择从第3行开始一直到最后一行，“1:4”代表了选择第2列至第4列。

【例3-16】沿用例3-1中的信息，投资者希望分别提取第2行的全部数据和第3列的全部数据，相关的操作如下

```
return_array[1]
```

```
Out[48]: array([ 0.001838, 0.001842, -0.016544, -0.003738, 0.003752])
```

```
return_array[:,2]
```

```
Out[49]:  
array([-0.004854, -0.016544, -0.033391, -0.029563])
```

3.3.3 排序

【例3-17】沿用例3-1中的信息，投资者希望针对股票按照日涨跌幅进行排序，具体的代码如下：

```
np.sort(return_array,axis=0)
```

```
Out[50]:
```

```
array([[ -0.024112, -0.000344, -0.033391, -0.01457 , -0.00606 ],  
       [ -0.003087,  0.001842, -0.029563, -0.003738,  0.003752],  
       [  0.001838,  0.011704, -0.016544,  0.006098,  0.004597],  
       [  0.003731,  0.021066, -0.004854,  0.007123,  0.016129]])
```

```
np.sort(return_array,axis=1)
```

```
Out[51]:
```

```
array([[ -0.00606 , -0.004854,  0.003731,  0.006098,  0.021066],  
       [ -0.016544, -0.003738,  0.001838,  0.001842,  0.003752],  
       [ -0.033391, -0.003087, -0.000344,  0.004597,  0.007123],  
       [ -0.029563, -0.024112, -0.01457 ,  0.011704,  0.016129]])
```

```
np.sort(return_array)
```

```
Out[52]:
```

```
array([[ -0.00606 , -0.004854,  0.003731,  0.006098,  0.021066],  
       [ -0.016544, -0.003738,  0.001838,  0.001842,  0.003752],  
       [ -0.033391, -0.003087, -0.000344,  0.004597,  0.007123],  
       [ -0.029563, -0.024112, -0.01457 ,  0.011704,  0.016129]])
```

3.4 主要数组运算

3.4.1 数组内运算

针对数组内部元素的求和，需要运用sum函数，并且有参数axis=0或者axis=1可以输入，其中，axis=0代表按列求和，axis=1则代表按行求和，如果不输入参数，表示对所有元素求和。

【例3-18】沿用例3-1中的信息，投资者按照 return_array数组中的列、行分别求和，具体的代码如下：

```
return_array.sum(axis=0)
```

```
Out[5]: array([-0.02163 , 0.034268, -0.084352, -0.005087, 0.018418])
```

```
return_array.sum(axis=1)
```

```
Out[6]: array([ 0.019981, -0.01285 , -0.025102, -0.040412])
```

```
return_array.sum()
```

```
Out[7]: -0.05838299999999999
```

3.4.1 数组内运算

针对数组内部元素求乘积，需要运用prod函数（prod是乘积英文product的缩写），同时，输入参数axis=0代表按列求乘积，axis=1则代表按行求乘积，如果不输入参数，所有元素求乘积。

【例3-19】沿用例3-1中的信息，投资者按照return_array数组中的列、行分别求乘积，具体的代码如下：

```
return_array.prod(axis=0)
```

```
Out[8]:
```

```
array([ 5.10435205e-10, -1.56230010e-10, 7.92717092e-08, 2.36564304e-09,  
       -1.68584406e-09])
```

```
return_array.prod(axis=1)
```

```
Out[9]: array([ 1.40983129e-11, 7.85557141e-13, -1.16107947e-12, -  
1.96057312e-09])
```

```
return_array.prod()
```

```
Out[10]: 2.521099098076826e-44
```


3.4.1 数组内运算

针对数组内部元素求最值，需要运用`max`函数求最大值，用`min`函数来求最小值，同时，输入参数`axis=0`代表按列求最值，`axis=1`则代表按行求最值，如果不输入参数，表示对所有元素求最值。

【例3-20】沿用例3-1中的信息，投资者按照`return`数组中的列、行分别求最大值和最小值，具体的代码如下：

```
return_array.max(axis=0)
Out[11]: array([ 0.003731, 0.021066, -0.004854, 0.007123, 0.016129])

return_array.max(axis=1)
Out[12]: array([0.021066, 0.003752, 0.007123, 0.016129])

return_array.max()
Out[13]: 0.021066
```

3.4.1 数组内运算

对数组内部元素求算术平均值，需要运用`mean`函数，同时，输入参数`axis=0`代表按列求平均值，`axis=1`则代表按行求平均值，如果不输入参数，表示对所有元素求平均值。

【例3-21】沿用例3-1中的信息，投资者按照 `return_array`数组中的列、行分别求算术平均值，具体的代码如下：

```
return_array.mean(axis=0)
```

```
Out[14]: array([-0.0054075 , 0.008567 , -0.021088 , -0.00127175, 0.0046045 ])
```

```
return_array.mean(axis=1)
```

```
Out[15]: array([ 0.0039962, -0.00257 , -0.0050204, -0.0080824])
```

```
return_array.mean()
```

```
Out[16]: -0.0029191499999999997
```

3.4.1 数组内运算

对数组内部元素求方差和标准差，需要分别运用到`var`、`std`函数，其中，`var`是方差英文`variance`的缩写，`std`是标准差英文`standard deviation`的缩写。同时，输入参数`axis=0`代表按列求方差或标准差，`axis=1`则代表按行求方差或标准差，如果不输入参数，表示对所有元素求方差或标准差。

【例3-22】沿用例3-1中的信息，投资者按照 `return_array`数组中的列、行分别求方差、标准差，具体的代码如下：

```
return_array.var(axis=0)
```

```
Out[17]:
```

```
array([1.22813123e-04, 7.26743290e-05, 1.26845031e-04, 7.69277212e-05,  
       6.18181183e-05])
```

```
return_array.var(axis=1)
```

```
Out[18]: array([9.50638330e-05, 5.51001584e-05, 2.14090849e-04,  
3.47629344e-04])
```

```
return_array.var()
```

```
Out[19]: 0.00019772563942750004
```

3.4.1 数组内运算

对数组内的每个元素计算开方、平方以及以e为底的指数次方，需要分别运用到函数`sqrt`、`square`、`exp`。

【例3-23】沿用例3-1中的信息，投资者对`return_array`数组中每个元素分别计算开平方、平方，以及以e为底的指数次方，具体的代码如下：

```
np.sqrt(return_array)
```

```
np.square(return_array)
```

```
np.exp(return_array)
```

3.4.1 数组内运算

对数组内的每个元素计算自然对数，底数10的对数，底数2的对数，需要分别运用函数`log`、`log10`、`log2`。

【例3-24】沿用例3-1中的信息，投资者对`return_array`数组中每个元素分别计算自然对数，底数10的对数，底数2的对数，具体的代码如下：

```
np.log(return_array)
```

```
np.log10(return_array)
```

```
np.log2(return_array)
```

3.4.2 数组间的运算

数组间的运算依然是包括了加（+）、减（-）、乘（*）、除（/）、幂（**）等，这些运算可以适合于具有相同的行数和列数的多个数组，并且是对数组的全部元素进行运算。

【例3-25】沿用例3-1和例3-11的信息，将数组return_array以及与该数组具有相同的行数与列数且元素等于1的数组one_return进行数组间的加、减运算，再将新生成的数组进行乘，除和幂运算，具体的代码如下：

```
new_array1 = return_array+one_return #数组相加  
new_array2 = return_array-one_return #数组相减  
new_array3 = new_array1*new_array2 #数组相乘  
new_array4 = new_array1/new_array2 #数组相除  
new_array5 = new_array1**new_array2 #数组幂运算
```

3.4.2 数组间的运算

【例3-26】沿用例3-1中的信息，对 `return_array` 数组的每个元素依次加上1、减去1、乘以2、除以2以及平方，具体的代码如下：

```
new_array6 = return_array+1  
new_array7 = return_array-1  
new_array8 = return_array*2  
new_array9 = return_array/2  
new_array10 = return_array**2
```

【例3-27】沿用例3-5中生成的数组 `return_array` 以及例3-10生成的数组 `zero_return`，分别生成由这两个数组之间对应元素的最大值、最小值作为元素的新数组，具体的代码如下：

```
return_max = np.maximum(return_array,zero_return);  
return_min = np.minimum(return_array,zero_return);
```

3.4.3 矩阵的操作

【例3-28】沿用例3-1中的信息，计算4只股票涨跌幅的相关系数矩阵，运用corrcoef函数可以直接得到计算结果，corrcoef是相关系数英文名correlation coefficient的缩写，具体的代码如下：

```
corrcoef_return = np.corrcoef(return_array)
```


3.4.3 矩阵的操作

函数名称	函数功能	在Python中的代码
diag	矩阵的对角线（diag是对角线英文diagonal line的缩写）	<pre>np.diag(corrcoef_return) Out[45]: array([1., 1., 1., 1.]</pre>
triu	矩阵上三角（triu是上三角英文 upper triangular的编写）	<pre>np.triu(corrcoef_return) Out[46]: array([[1. , 0.36417773, 0.36338676, 0.30254781], [0. , 1. , 0.86956454, 0.68996042], [0. , 0. , 1. , 0.60483848], [0. , 0. , 0. , 1.]])</pre>
tril	矩阵下三角（tril是下三角英文 lower triangular的编写）	<pre>np.tril(corrcoef_return) Out[47]: array([[1. , 0. , 0. , 0.], [0.36417773, 1. , 0. , 0.], [0.36338676, 0.86956454, 1. , 0.], [0.30254781, 0.68996042, 0.60483848, 1.]])</pre>
trace	矩阵的迹	<pre>np.trace(corrcoef_return) Out[48]: 4.0</pre>
Transpose (或T)	矩阵的转置	<pre>np.transpose(return_array) Out[49]: array([[0.003731, 0.001838, -0.003087, -0.024112], [0.021066, 0.001842, -0.000344, 0.011704], [-0.004854, -0.016544, -0.033391, -0.029563], [0.006098, -0.003738, 0.007123, -0.01457], [-0.00606 , 0.003752, 0.004597, 0.016129]])</pre>

3.4.3 矩阵的操作

【例3-29】沿用例3-1中的信息，按照每只股票在投资组合中的配置比例（权重）求出相应每个交易日投资组合的平均收益率，也就相当于求矩阵之间的内积，是运用到函数`dot`，具体的代码如下：

```
average_return = np.dot(weight_array,return_array)
```

3.4.3 矩阵的操作

NumPy有一个很重要的子模块 `linalg`，是一个专门用于线性代数运算的工具包。为了能够方便地调用该子模块的函数，导入该子模块并且用缩写 `la` 进行命名，具体的代码如下：

```
import numpy.linalg as la
```

函数名	函数功能	以例3-28的相关系数矩阵作为示例
<code>det</code>	矩阵的行列式（ <code>det</code> 是行列式英文 <code>determinant</code> 的缩写）	<code>la.det(corrcoef_return)</code>
<code>inv</code>	逆矩阵（ <code>inv</code> 是逆矩阵英文 <code>inverse matrix</code> 的编写）	<code>la.inv(corrcoef_return)</code>
<code>eig</code>	特征值分解（ <code>eig</code> 是特征值英文 <code>eigenvalue</code> 的缩写）	<code>la.eig(corrcoef_return)</code>
<code>svd</code>	奇异值分解（ <code>svd</code> 是奇异值分解 <code>singular value decomposition</code> 的英文缩写）	<code>la.svd(corrcoef_return)</code>

3.5 NumPy生成随机数

3.5.1 正态分布

NumPy中的random子模块抽取随机样本，限于篇幅仅演示正态分布、对数正态分布、卡方分布，学生t分布、F分布、贝塔分布和伽玛分布。首先，需要导入 NumPy中的 random子模块，具体的代码是：

```
import numpy.random as npr
```

【例3-30】假定从均值为1、标准差为2的正态分布中抽取随机数，同时设定抽取随数的次数为1万次，具体的代码如下：

```
import numpy.random as npr
x_norm = npr.normal(loc=1.0,scale=2.0,size=10000)
print('从正态分布中抽取的平均值',x_norm.mean())
print('从正态分布中抽取的标准差',x_norm.std())
```

3.5.1 正态分布

NumPy中的random子模块抽取随机样本，限于篇幅仅演示正态分布、对数正态分布、卡方分布，学生t分布、F分布、贝塔分布和伽玛分布。首先，需要导入 NumPy中的 random子模块，具体的代码是：

```
import numpy.random as npr
```

【例3-30】假定从均值为1、标准差为2的正态分布中抽取随机数，同时设定抽取随数的次数为1万次，具体的代码如下：

```
import numpy.random as npr
x_norm = npr.normal(loc=1.0,scale=2.0,size=10000)
print('从正态分布中抽取的平均值',x_norm.mean())
print('从正态分布中抽取的标准差',x_norm.std())
```

3.5.1 正态分布

【例3-31】假定从标准正态分布中抽取随机数，并且抽取随机数的次数依然是1万次，有3个函数可供选择，分别是 `randn`、`standard_normal`以及 `normal`函数，具体的代码如下：

```
x_snorm1 = npr.randn(10000)
x_snorm2 = npr.standard_normal(size=10000)
x_snorm3 = npr.normal(loc=0,scale=1.0,size=10000)
print('运用randn函数从正态分布中抽取的样本平均值',x_snorm1.mean())
print('运用randn函数从正态分布中抽取的样本标准差',x_snorm1.std())
print('运用standard_normal函数从正态分布中抽取的样本平均值',x_snorm2.mean())
print('运用standard_normal函数从正态分布中抽取的样本标准差',x_snorm2.std())
print('运用normal函数从正态分布中抽取的样本平均值',x_snorm3.mean())
print('运用normal函数从正态分布中抽取的样本标准差',x_snorm3.std())
```

3.5.2 对数正态分布

【例3-32】假定随机变量 x 的对数服从均值为0.5，标准差为1.0的正态分布，对变量进行随机抽样，并且抽取随机数的次数依然是1万次，具体的代码如下：

```
x_logn = npr.lognormal(mean=0.5,sigma=1.0,size=10000)
print('从对数正态分布中抽样的平均值',x_logn.mean())
print('从对数正态分布中抽样的标准差',x_logn.std())

print('对数正态分布总体的数学期望： ',np.exp(0.5+1**2/2))
print('对数正态分布总体的标准差：
',np.sqrt(np.exp(2*0.5+1**2)*(np.exp(1**2)-1)))
```

$$E(x) = e^{\mu + \frac{\sigma^2}{2}}$$

$$D(X) = e^{2\mu + \sigma^2} (e^{\sigma^2} - 1)$$

3.5.3 卡方分布

【例3-33】假定分别从自由度是4和100的卡方分布中抽取随机数，并且抽取的次数依然是1万次，具体的代码如下：

```
x_chi1 = npr.chisquare(df=4,size=10000)
x_chi2 = npr.chisquare(df=100,size=10000)
print('从自由度为4的卡方分布中抽样的平均值',x_chi1.mean())
print('从自由度为4的卡方分布中抽样的标准差',x_chi1.std())
print('从自由度为100的卡方分布中抽样的平均值',x_chi2.mean())
print('从自由度为100的卡方分布中抽样的标准差',x_chi2.std())
```

【例3-34】假定分别从自由度是2和120的学生t分布中抽取随机数，并且抽取随机数的次数依然是1万次，具体的代码如下：

```
x_t1 = npr.standard_t(df=2,size=10000)
x_t2 = npr.standard_t(df=120,size=10000)
print('从自由度为2的t分布中抽样的平均值',x_t1.mean())
print('从自由度为2的t分布中抽样的标准差',x_t1.std())
print('从自由度为120的t分布中抽样的平均值',x_t2.mean())
print('从自由度为120的t分布中抽样的标准差',x_t2.std())
```

3.5.5 F分布

【例3-35】假定从自由度 $n1=6$ 和 $n2=8$ 的F分布中抽取随机数，并且抽取随机数的数依然是1万次，具体的代码如下：

```
x_f = npr.f(dfnum=6,dfden=8,size=10000)
print('从自由度n1=6,n2=8的F分布中抽样的平均值',x_f.mean())
print('从自由度n1=6,n2=8的F分布中抽样的标准差',x_f.std())
```

3.5.6 贝塔分布

【例3-36】假定从 $a=2$ 、 $\beta=4$ 的贝塔分布中抽取随机数，并且抽取随机数的次数依然是1万次，具体的代码如下：

```
x_beta = npr.beta(a=2,b=4,size=10000)
print('从贝塔分布中抽样的平均值',x_beta.mean())
print('从贝塔分布中抽样的标准差',x_beta.std())
```

3.5.7 伽马分布

【例3-37】假定从 $\alpha=1$ 、 $\beta=3$ 的伽玛分布中抽取随机数，并且抽取随机数的次数依然是1万次，具体的代码如下：

```
x_gamma = npr.gamma(shape=1.0,scale=3.0,size=10000)
print('从伽马分布中抽样的平均值',x_gamma.mean())
print('从伽马分布中抽样的标准差',x_gamma.std())
```

**Your appreciation makes me a miracle.
Thank you!**

**Frank Ziwei Zhang
18117228563
frank8027@163.com**



上海對外經貿大學
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS