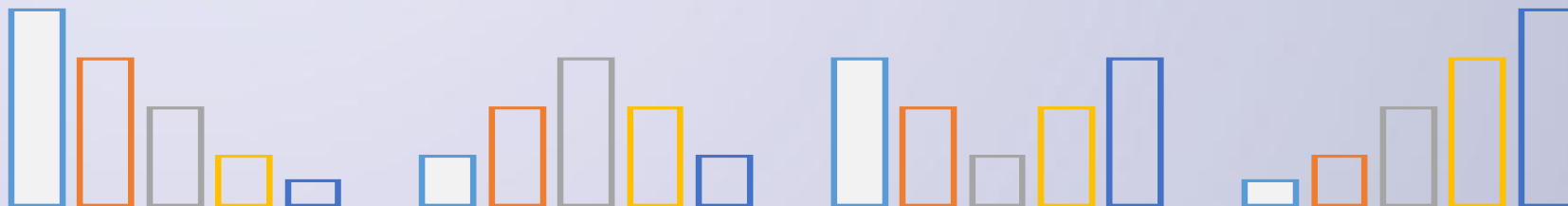


第6章 NumPy和SciPy模块



内容

- NumPy 模块
- SciPy 模块
- 线性回归和资本资产定价模型



NumPy模块

NumPy模块简介

- 提供了强大的多维数组`ndarray`类
- 全面的函数和方法，可以操纵数组、实现这些对象上的复杂运算
- 比纯python可以更紧凑的实现，往往更容易理解和维护
- 体现了向量化的运算，带来速度的提升，避免不必要的循环操作
- 列表形成数组
- 专门高效处理数组设计的数据结构



NumPy 库概述

- Python标准库中提供了一个array类型，用于保存数组类型数据，然而这个类型不支持多维数据，处理函数也不够丰富，不适合用于做数值运算。
- 因此，Python语言的第三方库numpy得到了迅速发展，numpy已经成为了科学计算事实上的标准库。



NumPy 库概述

- numpy 库处理的最基础数据类型是由同种元素构成的多维数组 (ndarray)，简称“数组”。
- 数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从0开始。ndarray 类型的维度 (dimensions) 叫做轴 (axes)，轴的个数叫做秩 (rank)。一维数组的秩为1，二维数组的秩为2，二维数组相当于由两个一维数组构成。



NumPy 库概述

- 由于numpy库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用numpy库：

```
>>>import numpy as np
```

- 其中，as 保留字与import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单说，在程序的后续部分中，np代替numpy。



numpy库常用的创建数组函数

| 函数 | 描述 |
|---|---------------------------------|
| <code>np.array([x,y,z], dtype=int)</code> | 从 Python 列表和元组创造数组 |
| <code>np.arange(x,y,i)</code> | 创建一个由 x 到 y，以 i 为步长的数组 |
| <code>np.linspace(x,y,n)</code> | 创建一个由 x 到 y，等分成 n 个元素的数组 |
| <code>np.indices((m,n))</code> | 创建一个 m 行 n 列的矩阵 |
| <code>np.random.rand(m,n)</code> | 创建一个 m 行 n 列的随机数组 |
| <code>np.ones((m,n),dtype)</code> | 创建一个 m 行 n 列全 1 的数组，dtype 是数据类型 |
| <code>np.empty((m,n),dtype)</code> | 创建一个 m 行 n 列全 0 的数组，dtype 是数据类型 |



ndarray 类的常用属性

创建一个简单的数组后，可以查看ndarray类型有一些基本属性

| 属性 | 描述 |
|-------------------------------|---|
| <code>ndarray.ndim</code> | 数组轴的个数，也被称作秩 |
| <code>ndarray.shape</code> | 数组在每个维度上大小的整数元组 |
| <code>ndarray.size</code> | 数组元素的总个数 |
| <code>ndarray.dtype</code> | 数组元素的数据类型， <code>dtype</code> 类型可以用于创建数组中 |
| <code>ndarray.itemsize</code> | 数组中每个元素的字节大小 |
| <code>ndarray.data</code> | 包含实际数组元素的缓冲区地址 |
| <code>ndarray.flat</code> | 数组元素的迭代器 |



ndarray类的常用属性

```
>>>import numpy as np
>>>a = np.ones((4,5))
>>>print(a)
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
>>>a.ndim
2
>>>a.shape
(4,5)
>>>a.dtype
dtype('float64')
```




ndarray 类的形态操作方法

| 方法 | 描述 |
|---|---|
| <code>ndarray.reshape(n,m)</code> | 不改变数组 <code>ndarray</code> ，返回一个维度为(n,m)的数组 |
| <code>ndarray.resize(new_shape)</code> | 与 <code>reshape()</code> 作用相同，直接修改数组 <code>ndarray</code> |
| <code>ndarray.swapaxes(ax1, ax2)</code> | 将数组 <code>n</code> 个维度中任意两个维度进行调换 |
| <code>ndarray.flatten()</code> | 对数组进行降维，返回一个折叠后的一维数组 |
| <code>ndarray.ravel()</code> | 作用同 <code>np.flatten()</code> ，但是返回数组的一个视图 |



ndarray类的形态操作方法

- 数组在numpy中被当作对象，可以采用<a>.()方式调用一些方法。这里给出了改变数组基础形态的操作方法，例如改变和调换数组维度等。
- 其中，.flatten()函数用于数组降维，相当于平铺数组中数据，该功能在矩阵运算及图像处理中用处很大。



ndarray类的索引和切片方法

```
>>>a = np.random.rand(5,3) #生成 5x3 的数组，用随机数填充
>>>a[2] #获得第 2 行数据
array([ 0.78426574,  0.60171943,  0.98825306])
>>>a[1:3]
array([[ 0.49276756,  0.44735929,  0.10356773],
       [ 0.78426574,  0.60171943,  0.98825306]])
>>>a[-5:-2:2]
array([[ 0.95517757,  0.3634953 ,  0.34138831],
       [ 0.78426574,  0.60171943,  0.98825306]])
```



numpy 库的算术运算函数

| 函数 | 描述 |
|--|------------------------|
| <code>np.add(x1, x2 [, y])</code> | $y = x1 + x2$ |
| <code>np.subtract(x1, x2 [, y])</code> | $y = x1 - x2$ |
| <code>np.multiply(x1, x2 [, y])</code> | $y = x1 * x2$ |
| <code>np.divide(x1, x2 [, y])</code> | $y = x1 / x2$ |
| <code>np.floor_divide(x1, x2 [, y])</code> | $y = x1 // x2$, 返回值取整 |
| <code>np.negative(x [,y])</code> | $y = -x$ |
| <code>np.power(x1, x2 [, y])</code> | $y = x1^{**}x2$ |
| <code>np.remainder(x1, x2 [, y])</code> | $y = x1 \% x2$ |



numpy库的算术运算函数

这些函数中，输出参数`y`可选，如果没有指定，将创建并返回一个新的数组保存计算结果；如果指定参数，则将结果保存到参数中。例如，两个数组相加可以简单地写为`a+b`，而`np.add(a, b, a)`则表示`a+=b`。



numpy库的比较运算函数

| 函数 | 符号描述 |
|---|------------------------|
| <code>np. equal(x1, x2 [, y])</code> | $y = x1 == x2$ |
| <code>np. not_equal(x1, x2 [, y])</code> | $y = x1 != x2$ |
| <code>np. less(x1, x2, [, y])</code> | $y = x1 < x2$ |
| <code>np. less_equal(x1, x2, [, y])</code> | $y = x1 \leq x2$ |
| <code>np. greater(x1, x2, [, y])</code> | $y = x1 > x2$ |
| <code>np. greater_equal(x1, x2, [, y])</code> | $y = x1 \geq x2$ |
| <code>np.where(condition[x,y])</code> | 根据给出的条件判断输出 x 还是 y |



numpy库的比较运算函数

- 其将返回一个布尔数组，它包含两个数组中对应元素值的比较结果，例子如下。
- `where()` 函数是三元表达式 `x if condition else y` 的矢量版本。

```
>>>np.less([1,2],[2,2])  
array([ True, False], dtype=bool)
```



numpy库的其他运算函数

| 函数 | 描述 |
|---|--------------------------------------|
| <code>np.abs(x)</code> | 计算基于元素的整形，浮点或复数的绝对值。 |
| <code>np.sqrt(x)</code> | 计算每个元素的平方根 |
| <code>np.square(x)</code> | 计算每个元素的平方 |
| <code>np.sign(x)</code> | 计算每个元素的符号：1(+), 0, -1(-) |
| <code>np.ceil(x)</code> | 计算大于或等于每个元素的最小值 |
| <code>np.floor(x)</code> | 计算小于或等于每个元素的最大值 |
| <code>np rint (x[, out])</code> | 圆整,取每个元素为最近的整数,保留数据类型 |
| <code>np.exp(x[, out])</code> | 计算每个元素指数值 |
| <code>np.log(x), np.log10(x), np.log2(x)</code> | 计算自然对数(e),基于 10,2 的对数, $\log(1 + x)$ |



numpy 库还包括三角运算函数、傅里叶变换、随机和概率分布、基本数值统计、位运算、矩阵运算等非常丰富的功能...

使用NumPy的示例

```
>>>import numpy as np
>>>x= np.array([[1, 2, 3], [3, 4, 6]])    # 2 by 3 matrix
>>>np.size(x) # number of data items
>>>6
>>>np.size(x,1) # show number of columns
3
>>>np.std(x)
1.5723301886761005
>>>np.std(x,1)
Array([ 0.81649658,  1.24721913])
>>>total=x.sum()          # pay attention to the format
>>>z=np.random.rand(50)    # 50 random obs from [0.0, 1)
>>>y=np.random.normal(size=100) # from standard normal
>>>r=np.array(range(0, 100), float)/100 # from 0, .01, to .99
```

help(np.std) #可查看函数具体信息

构建不同矩阵

```
>>>import numpy as np
>>>a=np.zeros(10)      # array with 10 zeros
>>>b=np.zeros((3,2),dtype=float)  # 3 by 2 with zeros
>>>c=np.ones((4,3),float)  # 4 by 3 with all ones
>>>d=np.array(range(10),float)  # 0,1, 2,3 .. up to 9
>>>e1=np.identity(4)    # identity 4 by 4 matrix
>>>e2=np.eye(4)         # same as above
>>>e3=np.eye(4,k=1)     # index of the diagonal
>>>f=np.arange(1,20,3,float)  # from 1 to 19 interval 3
>>>g=np.array([[2,2,2],[3,3,3]])  # 2 by 3 matrix
>>>h=np.zeros_like(g)  # all zeros
>>>i=np.ones_like(g)   # all ones
```

改变矩阵或数组形状

```
>>>pv=np. array([[100, 10, 10. 2], [34, 22, 34]]) # 2 by 3  
>>>x=pv.flatten() # matrix becomes a vector  
>>>vp2=np. reshape(x, [3, 2])
```

转置运算:

```
>>> np.transpose(B)  
array([[1, 3],  
       [2, 1],  
       [3, 2]])
```

数组和矩阵运算

- 加减运算

```
>>> A=np.array([[1, 2, 3], [2, 1, 3]])
>>> B=np.array([[1, 2, 3], [3, 1, 2]])
>>> A+B
array([[2, 4, 6],
       [5, 2, 5]])
```

- 乘法运算

```
>>> Bt=np.transpose(B)
>>> np.dot(A, Bt)
array([[14, 11],
       [13, 13]])
```

```
>>> A1=np.matrix("1, 2, 3;2, 1, 3")
>>> B1=np.matrix("1, 2, 3;3, 1, 2")
>>> A1*np.transpose(B1)
matrix([[14, 11],
        [13, 13]])
```

- 同型矩阵或数组逐项相乘

```
>>> A*B
array([[1, 4, 9],
       [6, 1, 6]])
```

其他...

关于x.func()使用：当变量x被定义为一个NumPy数组后

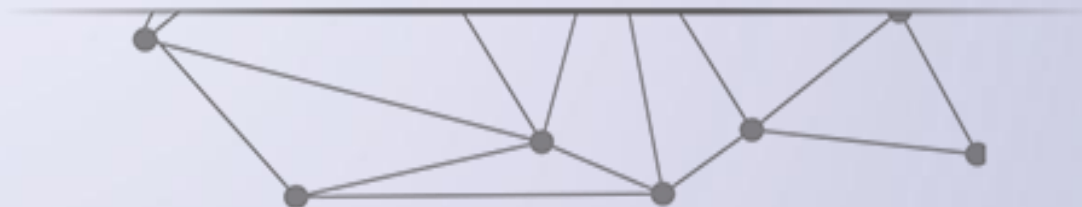
- x.min()
- x.max()
- x.sum()
- x.transpose()
- x.flatten()
- X.reshape()

遍历数组的循环语句：

```
>>> dataset=np.array(np.random.normal(size=10))  
>>> for data in dataset:  
    print(data)
```




SciPy模块



使用SciPy的示例

```
>>>import scipy as sp
>>>cashflows=[50, 40, 20, 10, 50]
>>>npv=sp.npv(0.1, cashflows) #estimate NPV
>>>round(npv, 2)
>>>144.56
```

```
>>>payment=sp.pmt(0.045/12, 30*12, 250000)
>>>round(payment, 2)
-1266.71
```

```
>>>ret=sp.array([0.1, 0.05, -0.02])
>>>sp.mean(ret) # arithmetic mean
0.04333
>>>pow(sp.prod(ret+1), 1./len(ret))-1 # geometric mean
0.04216
```

SciPy子函数包(表6-1)

| 子函数包N | 描述 |
|-------------|-----------|
| Cluster | 聚类算法 |
| Constants | 物理和数学常量 |
| Fftpack | 快速傅里叶变换 |
| Integrate | 积分和微分方程 |
| Interpolate | 插值和光滑样条函数 |
| Io | 输入和输出 |
| Linalg | 线性代数 |
| Ndimimage | N-维图像处理 |

| 子函数包 | 描述 |
|----------|---------------|
| Odr | 基于正交距离测度的回归模型 |
| Optimize | 优化和求解方程 |
| Signal | 信号处理 |
| Sparse | 稀疏矩阵和相关算法 |
| Spatial | 空间数据结构 and 算法 |
| Special | 特殊函数 |
| Stats | 概率分布函数和统计分析 |
| Weave | C/C++编程接口 |

统计子模块stats

- 累积标准正态分布

```
>>>from scipy.stats import norm
>>>norm.cdf(0)
0.5
```

```
>>>from scipy import stats
>>>dir(stats)
```

```
>>> help(stats.chi2)
```

- 分布

norm、chi2、f

- 概括性度量

skew

- 参数、非参数检验

ttest_1samp

- 列联分析

chi2_contingency

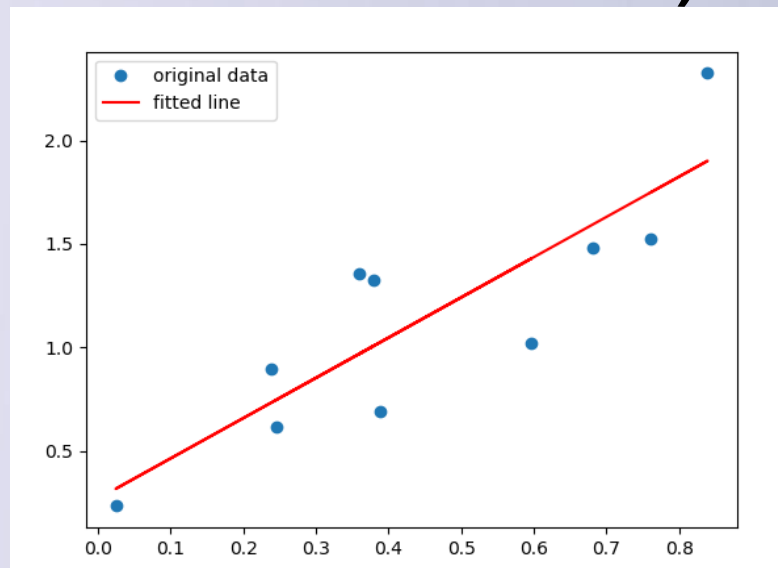
- 线性回归分析

linregress

回归分析示例(scipy.stats.linregress)

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy import stats
>>> np.random.seed(12345678)
>>> x = np.random.random(10)
>>> y = 1.6*x + np.random.random(10)
```

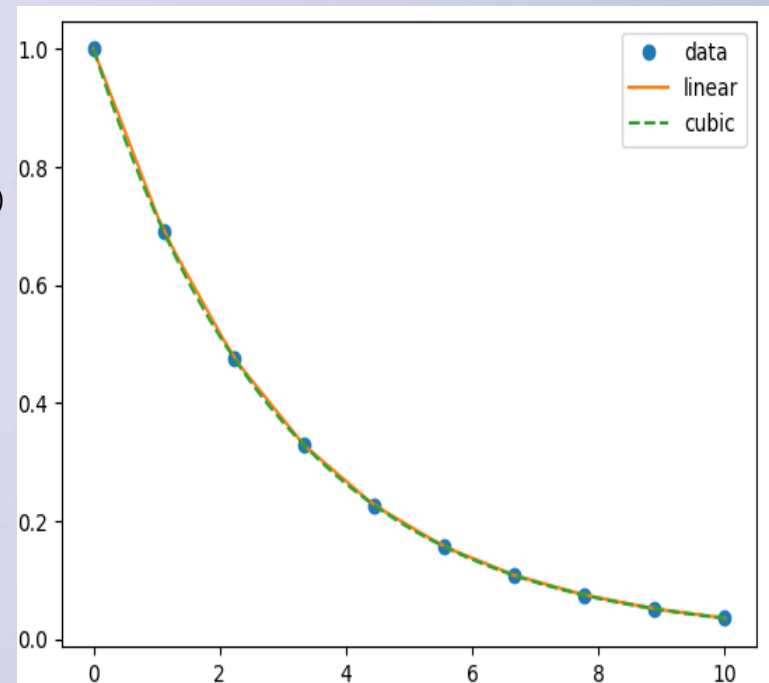
注意np.random.seed可生成重复的
随机数



```
>>> slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
>>> print("slope: %f      intercept: %f" % (slope, intercept))
      slope: 1.944864      intercept: 0.268578
>>> print("r-squared: %f" % r_value**2)
      r-squared: 0.735498
>>> plt.plot(x, y, 'o', label='original data')
>>> plt.plot(x, intercept + slope*x, 'r', label='fitted line')
>>> plt.legend()
>>> plt.show()
```

插值(scipy.interpolate)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
x = np.linspace(0, 10, 10) #generate 10 evenly spaced numbers from (0,10)
y = np.exp(-x/3.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 10, 40) #40 values from (0,10)
plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```




优化(scipy.optimize)

在金融领域，许多问题需要用到优化算法，给定目标函数和一组约束条件选择最佳投资组合。

```
>>>import scipy.optimize as optimize
>>>def my_f(x):
Return 3 + x**2
>>>optimize.fmin(my_f,5) # 5 is initial value
Optimization terminated successfully
Current function values: 3:000000
Iterations: 20
Function evaluations: 40
Array([ 0. ])
```

问题：目标函数 $SSE = \sum_{i=1}^n (y_i - \beta x_i)^2$ ，有数据 (x_i, y_i) ，求 β 的最小二乘估计。



线性回归分析和CAPM

CAPM

根据CAPM模型，单只股票的收益率和市场收益率线性相关，考虑个股超额收益率和市场的超额收益率之间的关系：

$$R_i - R_f = \alpha + \beta_i(R_m - R_f)$$

R_i : 个股收益率; β_i : 度量市场风险; R_m : 市场收益率

R_f : 无风险利率

如果我们有 R_i 、 R_m 、 R_f 的数据，就可以估计 α 和 β_i ，这是一个线性回归模型的最小二乘估计。

问题：数据怎么获得？如何导入数据？如何进行回归？

小 结

- NumPy模块和SciPy模块功能
- 回归分析应用