Chapter 6 – SciPy等模块的金融场景

Frank Ziwei Zhang
School of Finance

上海对外经贸大学
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

Contents

6.1 SciPy模块

6.1.1 求积分

SciPy模块是一款集数学与工程的开源软件，该模块依赖于NumPy模块，能提供方便快捷的N维数组操作。SciPy模块可以与NumPy的数组、Pandas的序列和数据框一起使用，并提供包括积分、优化等许多高效的数值运算。SciPy模块不仅易于使用，而且功能强大，使得一些世界顶尖的科学家和工程师都依赖它，目前金融领域也比较依赖这个模块。下面导入SciPy模块并且查看版本信息，具体的代码如下：

```
import scipy                          #导入SciPy模块
scipy.__version__                     #查看版本信息

Out[1]: '1.5.2'
```

6.1.1 求积分

子模块名称	功能
cluster	聚类算法
constants	物理数学常数
ffpack	快速傅里叶变换
intergrate	积分和常微分方程求解
interpolate	插值处理
io	输入输出
linalg	线性代数
ndimage	多维图像处理模块
odr	正交距离回归
opyimize	优化和求根
signal	信号处理
sparse	稀疏矩阵
spatial	空间数据结构和算法
special	特殊函数模块
stats	统计分布和函数
weave	调用C/C++

6.1.1 求积分

在对复杂的金融产品进行估值时，会经常用到积分。下面，就以金融领域最常用的标准正态分布作为示例介绍如何通过SciPy求解积分。在统计学中，如果随机变量x服从标准正态分布，则它的概率密度函数是：

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

假定需要计算变量x处区间[a,b]的概率，具体就是对公式（6-1）求以下积分

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

【例6-1】假定变量是服从标准正态分布，需要计算当该变量处于区间[-1,1]的概率，通过SciPy模块求解，具体的计算分为两个步骤。

第1步：导入SciPy的子模块 integrate，并且在Python中自定义一个标准正态分布的概率密度函数，具体的代码如下：

```
import scipy.integrate as sci              #导入SciPy的子模块integrate
def f(x):
 import numpy as np                        #导入Numpy模块
 return 1/pow(2*np.pi,0.5)*np.exp(-0.5*x**2)  #输出标准正态分布的函数概率密度的表达式
```

6.1.1 求积分

第2步：在integrate子模块中，有多个求解积分的函数，函数形式及主要参数如下：

函数（func,a,b）

其中，func表示被积函数，a是区间的下限，b是区间的上限；具体的函数及针对例6-1的代码演示见表6-2。

表格	功能	针对例6-1的具体运用
quad	自适应求积分	sci.quad(func=f,a=-1.0,b=1.0) Out[2]: (0.682689492137086, 7.579375928402476e-15) 输出的结果依次是积分值和最大误差，下同
fixed_quad	固定高斯求积分	sci.fixed_quad(func=f,a=-1.0,b=1.0) Out[3]: (0.6826897353882191, None)
quadrature	自适应高斯求积分	sci.quadrature(func=f,a=-1.0,b=1.0) Out[4]: (0.6826894922280757, 5.174690009823735e-09)
romberg	自适应龙贝格求积分	sci.romberg(function=f,a=-1.0,b=1.0) Out[5]: 0.6826894921481355 需要注意的是，在romberg函数中，表示被积函数的参数是用function而非func

6.1.2 插值法

通过导入SciPy的子模块interpolate可以进行插值运算，并且最常用的是一维数据的插值计算，需要通过函数interpld完成。该函数的主要格式和参数如下：

interpld(x,y,kind)

其中，x和y是一系列已知的数据点，并且有y=f(x)的函数关系式；kind代表了求插值的具体方法，常用的方法见表6-3：

参数名称	对应的插值方法
nearest	最邻近插值法
zero	阶梯插值法、也就是0阶样条曲线插值法
slinear	线性插值法。也就是1阶样条曲线插值法
quadratic	2阶样条曲线辅值法
cubic	3阶样条曲线辅值法

6.1.2 插值法

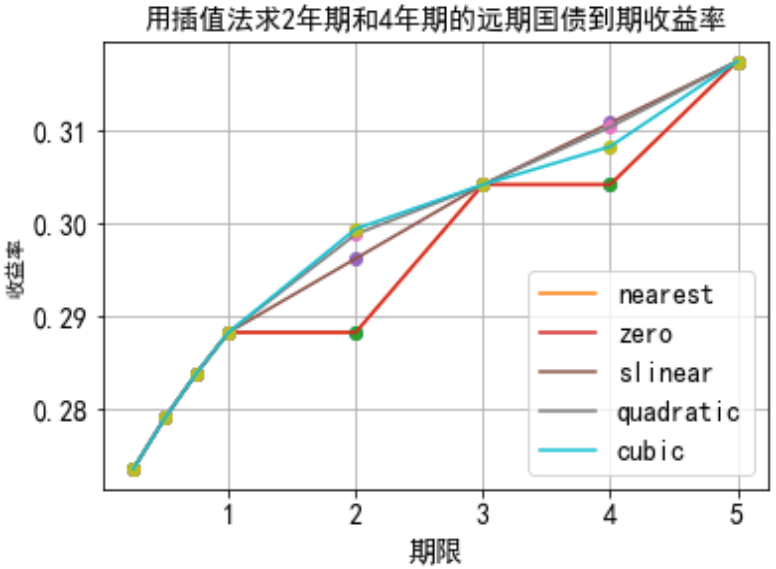
【例6-2】以2018年12月28日的远期国债到期收益率作为例子，远期国债到期收益率的信息如表6-4所示，考虑到表中缺少2年期、4年期的远期国债收益率，因此需要通过插值法得到相关的收益率。

期限	0.25年	0.5年	0.75年	1年	3年	5年
远期收益率	2.7344%	2.7898%	2.8382%	2.882%	3.0414%	3.1746%

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import mpl
mpl.rcParams['font.sans-serif']=['SimHei']
mpl.rcParams['axes.unicode_minus']=False
from scipy import interpolate                                  #导入SciPy的子模块interpolate
t=np.array([0.25,0.5,0.75,1.0,3.0,5.0])                        #生成仅包含已有期限的数组
t_new=np.array([0.25,0.5,0.75,1.0,2.0,3.0,4.0,5.0])            #生成包括2年和4年的新数组
rates=np.array([0.27344,0.27898,0.28382,0.2882,0.30414,0.31746]) #生成仅包含已有利率的数组
types=['nearest','zero','slinear','quadratic','cubic']        #生成包含插值方法的列表
plt.figure(figsize=(8,6))
```

6.1.2 插值法

```
for i in types:                                    #用for循环计算不同插值方法的结果并输出
    f=interpolate.interp1d(x=t,y=rates,kind=i)
    rates_new=f(t_new)
    print(i,rates_new)
    plt.plot(t_new,rates_new,'o')
    plt.plot(t_new,rates_new,'-',label=i)
    plt.xticks(fontsize=14)
    plt.xlabel(u'期限',fontsize=14)
    plt.yticks(fontsize=14)
    plt.ylabel(u'收益率',rotation=90)
    plt.legend(loc=0,fontsize=14)
    plt.grid()
plt.title(u'用插值法求2年期和4年期的远期国债到期收益率',fontsize=14)
```



6.1.3 求解方程组

【例6-3】沿用3.1节例3-1的相关股票信息：除了已知在2018年9月3日至9月6日每只股票的涨跌幅以外，同时也已知整个投资组合的收益率（见表6-5的最后一列）。假定在这些交易日，投资组合中每只股票的权重保持不变，根据这些已知的信息求解这4只股票在整个投资组合中所占的权重。

股票简称	中国石油	工商银行	上汽集团	宝钢股份	投资组合收益率
2018-09-03	0.3731%	-0.1838%	-0.3087%	-2.4112%	-0.105654%
2018-09-04	2.1066%	0.1842%	-0.0344%	1.1704%	0.70534%
2018-09-05	-0.4854%	-1.6544%	-3.3391%	-2.9563%	-2.56367%
2018-09-06	0.6098%	-0.3738%	0.7123%	-1.4570%	-0.38289%

$$\begin{bmatrix} 0.3731\% & -0.1838\% & -0.3087\% & -2.4112\% \\ 2.1066\% & 0.1842\% & -0.0344\% & 1.1704\% \\ -0.4854\% & -1.6544\% & -3.3391\% & -2.9563\% \\ 0.6098\% & -0.3738\% & 0.7123\% & -1.4570\% \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -1.05654\% \\ 0.70534\% \\ -2.56367\% \\ -0.38289\% \end{bmatrix}$$

6.1.3 求解方程组

求解方程组是一项比较繁琐的工作，但是运用SciPy子模块linalg就可以轻松求解线性方程组，需要调用函数solve，该函数的格式和参数如下：

Solve(a,b)

其中，参数a必须是N行、N列的数组，相当于是方程组等号左边的系数矩阵，b是包括N个元素的一维数组，相当于是方程组等号右边的矩阵。下面,就运用 solve函数求解例6-3中4只股票的权重，具体的代码如下：

```
from scipy import linalg                #导入SciPy的子模块linalg
stock_return=np.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]])  #创建包含4只股票涨跌幅的数组
port_return=np.array([-0.0105654,0.0070534,-0.0256367,-0.0038289])  #创建投资组合收益率的数组
weight=linalg.solve(a=stock_return,b=port_return)    #计算每只股票的权重
stock=np.array(['中国石油','工商银行','上汽集团','宝钢股份'])
for i in range(0,4):
   print(stock[i],round(weight[i],2))
```

```
中国石油 0.1
工商银行 0.2
上汽集团 0.3
宝钢股份 0.4
```

6.1.3 求解方程组

同时，对于例6-3也可以运用SciPy的子模块optimize中的fsolve函数求解，该函数的格式如下：

fsolve(func, x0)

其中，func代表求解的方程式，需要通过def自定义；x0表示初始猜测的方程组的解，该函数在后面第7章讨论债券零息利率曲线时会发挥很大的作用。下面针对例6-3，运用 fsolve函数演示具体的求解过程，具体的代码如下：

```
def g(w):                    #定义求解每只股票权重的方程组
    w1,w2,w3,w4 = w
    eq1=0.003731*w1-0.001838*w2-0.003087*w3-0.024112*w4+0.0105654      #第一个等于0的方程式
    eq2=0.021066*w1+0.001842*w2-0.000344*w3+0.011704*w4-0.0070534      #第二个等于0的方程式
    eq3=-0.004854*w1-0.016544*w2-0.033391*w3-0.029563*w4+0.0256367     #第三个等于0的方程式
    eq4=0.006098*w1-0.003738*w2+0.007123*w3-0.01457*w4+0.0038289       #第四个等于0的方程式
    return [eq1,eq2,eq3,eq4]
import scipy.optimize as sco             #导入SciPy的子模块optimize
result=sco.fsolve(g,[0.01,0.01,0.01,0.01])    #求方程组的解
Result

Out[15]: array([0.1, 0.2, 0.3, 0.4])
```

6.1.4 最优化

【例6-4】假定一家投资机构拟配置4只A股股票，分别是贵州茅台、工商银行、上汽集团、宝钢股份，表6-6列出了这4只股票的相关信息。该投资机构的资金为1亿元，以12月28日的收盘价投资，希望实现投资组合收益率的最大化，同时要求整个投资组合的贝塔值不超过1.4，此外，每只股票不允许卖空，需计算应该配置的每只股票权重和股数。

证券简称	2017年至2018年 平均年化收益率	收盘价 （2018年12月28日）	股票贝塔值
贵州茅台	34.9032%	590.01	1.64
工商银行	15.5143%	5.29	1.41
上汽集团	13.2796%	26.67	1.21
宝钢股份	5.5905%	6.50	1.06

$$\max_{w_i}\left(\sum_{i=1}^{4} R_i w_i\right)$$

约束条件一共有3个，分别是

$$\sum_{i=1}^{4} w_i = 1$$

$$\sum_{i=1}^{4} \beta_i w_i \leqslant 1.4$$

$$w_i > 0$$

6.1.4 最优化

第1步：计算每只股票的最优投资权重，具体的代码如下：

```
import scipy.optimize as sco                #导入SciPy的子模块optimize
P=np.array([590.01,5.29,26.67,6.50])              #输入股票价格
R=np.array([0.349032,0.155143,0.132796,0.055905])    #输入股票收益率
b=np.array([1.64,1.41,1.21,1.06])                 #输入股票贝塔值
def f(w):                              #定义求最优值得函数
   w=np.array(w)
   return -np.sum(R*w)
cons=({'type':'eq', 'fun': lambda w: np.sum(w)-1},{'type':"ineq",'fun':lambda w: 1.4-
np.sum(w*b)})
bnds=((0,1),(0,1),(0,1),(0,1))
result=sco.minimize(f,[0.25,0.25,0.25,0.25], method="SLSQP",
bounds=bnds,constraints=cons)      #计算最优的解
result

Out[17]:
 fun: -0.22804894822767985
     jac: array([-0.349032, -0.155143, -0.132796, -0.055905])
 message: 'Optimization terminated successfully'
    nfev: 20
     nit: 4
    njev: 4
  status: 0
 success: True
x: array([5.11920517e-01, 2.22044605e-16, 2.87240675e-01, 2.00838808e-01])
```

6.1.4 最优化

第2步：根据每只股票的最优投资权重，计算得到该投资组合的收益率，具体的代码如下：

```
result['x'].round(3)
Out[18]: array([0.512, 0.   , 0.287, 0.201])
-f(result['x']).round(3)
Out[19]: 0.228
```

通过以上的计算，可以得到该投资组合的最大收益率是22.8%。

第3步：计算投资组合的收益率最大时，购买每只股票的股票数量，具体的代码如下：

```
shares=100000000*result["x"]/P
shares=shares.round(0)                                    #结果去整数，因为最少是1股
print('贵州茅台的股数:',shares[0])
print('工商银行的股数:',shares[1])
print ('上汽集团的股数:',shares[2])
print('宝钢股份的股数:',shares[3])
贵州茅台的股数: 86765.0
工商银行的股数: 0.0
上汽集团的股数: 1077018.0
宝钢股份的股数: 3089828.0
```

6.1.5 统计功能

【例6-6】沿用5.5节例5-8中运用的沪深300指数和上证180指数2016年至2018年的日涨跌幅数据，用于演示子模块stats中的统计函数及其用法，具体代码如下：

```
import scipy.stats as st
import pandas as pd
HS300_sz180 = pd.read_excel('D:\Zhangzw\Python\Python金融数据分析\RawData\第5章\沪深300指数与上证180指数的日涨跌幅_2016_2018.xlsx',header=0,index_col=0)  #注意导入的是sheet1
HS300_sz180.describe()
```

Out[2]:

	沪深300涨跌幅	上证180涨跌幅
count	731.000000	731.000000
mean	-0.000223	-0.000154
std	0.011826	0.011400
min	-0.070206	-0.067217
25%	-0.005144	-0.004876
50%	0.000350	0.000126
75%	0.005075	0.005022
max	0.043167	0.041031

6.1.5 统计功能

五数	描述	针对例6-6的代码演示
describe	描述性统计信息 注：与Pandas的descibe 函数有部分相似之处	st.describe(HS300_sz180) Out[4]: DescribeResult(nobs=731, minmax=(array([-0.07020589, -0.06721739]), array([0.04316748, 0.04103086])), mean=array([-0.00022295, -0.00015435]), variance=array([0.00013986, 0.00012996]), skewness=array([-1.04965359, -1.05201293]), kurtosis=array([6.21642174, 6.47677757])) 以上输出的结果依次是样本数量、最小最大值、均值、方差、偏度以及峰度。 其中，偏度和蜂度在Pandas的describe函数输出结果中是没有的。
kurtosis	峰度	st.kurtosis(HS300_sz180) Out[5]: array([6.21642174, 6.47677757])
moment	n阶短	st.moment(HS300_sz180,moment=2) Out[6]: array([0.00013967, 0.00012978]) 输入的参数 moment=2代表了2阶矩，以此类推。
mode	众数	st.mode(HS300_sz180) Out[7]: ModeResult(mode=array([[-0.07020589, -0.06721739]]), count=array([[1, 1]])) 在输出的结果中，count=array([1,1])代表了两个变量的样本中众数均只有一个
skew	偏度	st.skew(HS300_sz180) Out[8]: array([-1.04965359, -1.05201293])

6.2 StatsModels模块

6.2 StatsModels模块

Stats Models模块最早起源于SciPy子模块stats中的models工具包，最初由Jonathan Taylor编写，但后来从SciPy中被移除了。然而，在2009年的谷歌代码夏季峰会（Google Summer of Code）期间，经过修正、测试、改进并最终以全新的独立模块Stats Models对外发布。此后，StatsModels的开发团队不断添加新模型、绘图工具和统计方法，使得它最终成为了一款功能强大的统计分析工具，详细的功能介绍可以访问官方网站进行查询。

在使用之前依然需要导入并且查看版本信息，具体的代码如下：

```
import statsmodels                #导入StatsModels模块
statsmodels.__version__

Out[30]: '0.11.1'
```

6.2 StatsModels模块

本书中，运用Stats Model模块主要是解决线性回归的问题，因此本节的讨论侧重于如何运用该模块进行线性回归，具体可以分为以下若干步骤。

第1步：导入 Stats Models的子模块api；

第2步：除了导入相关的因变量（被解释变量）、自变量（解释变量）的数据以外，还需要对自变量的数据增加一列常数项；

第3步：构建相关的线性回归模型，在这一步中，可以根据需要运用不同的线性回归模型，具体的模型类别和对应的函数如表6-11所示。

第4步：用fit函数生成一个线性回归的结果对象，结果对象包含了回归模型的结果参数和模型诊断信息。

函数	模型的类型
OLS	普通最小二乘法回归（ ordinary least square regression）
GLS	广义最小二乘法回归（ generalized least squares regression）
WLS	加权最小二乘法回归（weighted least square regression）
GLASAR	带有自相关误差模型的广义最小二乘法回归（ GLs with autoregressive errors model）
GLM	广义线性模型（generalized linear models）
RLM	使用M个估计量的鲁棒线性模型（ robust linear models using M estimators）
mixed	混合效应模型（mixed effects models）
gam	广义加性模型（generalized additive models）

6.2 StatsModels模块

【例6-12】以2016年至2018年期间工商银行A股股价涨跌幅作为因变量，沪深300指数的涨跌幅作为自变量，构建普通最小二乘法回归模型，具体分为两个步骤完成。

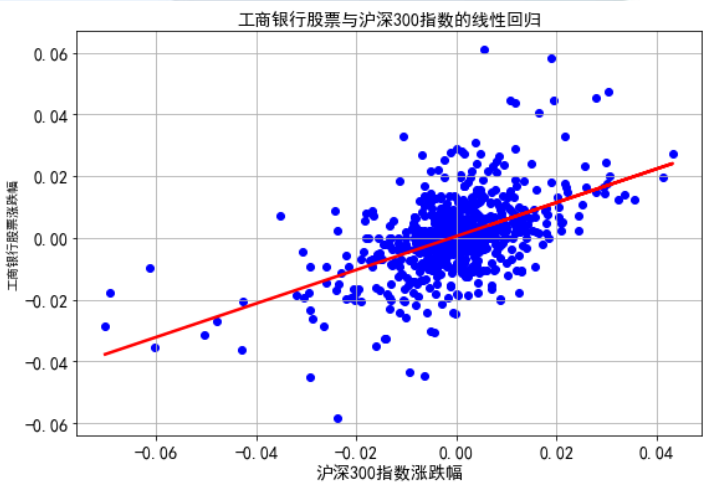
第1步：导入数据并计算普通最小二乘法回归模型的结果，具体的代码如下：

```
import statsmodels.api as sm            #导入StatsModels的子模块api
ICBC_HS300=pd.read_excel('D:\Zhangzw\Python\Python金融数据分析\RawData\第6章\工商银行与沪深300指数.xlsx',sheet_name='Sheet1',header=0,index_col=0) #导入外部数据
ICBC_HS300=ICBC_HS300.dropna()                  #删除缺失值
Y=ICBC_HS300.iloc[:,0]
X=ICBC_HS300.iloc[:,1]
X_addcons=sm.add_constant(X)
model=sm.OLS(endog=Y,exog=X_addcons) #构建普通最小二乘法的线性回归模型
result=model.fit()                  #生成一个线性回归的结果对象
result.summary()
```

第2步：对线性回归模型进行可视化，具体的代码如下：

```
import matplotlib
import matplotlib.pyplot as plt
import pylab as mpl
mpl.rcParams['font.sans-serif']=['SimHei']
mpl.rcParams['axes.unicode_minus']=False

plt.figure(figsize=(9,6))
plt.scatter(X,Y,c="b",marker="o")
plt.plot(X,result.params[0]+result.params[1]*X,'r-',lw=2.5)  #生成拟合的一条直线
plt.xticks(fontsize=14)
plt.xlabel(u'沪深300指数涨跌幅',fontsize=14)
plt.yticks(fontsize=14)
plt.ylabel(u'工商银行股票涨跌幅',rotation=90)
plt.title(u'工商银行股票与沪深300指数的线性回归',fontsize=14)
plt.grid()
```



6.3 波动率模型与arch模块

6.3.1 估计波动率

根据 ui 在最近 m 个交易日的观测数据推算出的方差率 σ_n^2 的无偏估计：

$$\sigma_n^2 = \frac{1}{m-1}\sum_{i=1}^{m} (u_{n-1} - \bar{u})^2 \qquad (6\text{-}5)$$

其中 $\bar{u}$ 是 ui 的平均值，即

$$\bar{u} = \frac{1}{m}\sum_{i=1}^{m} u_{n-i}$$

为有效跟踪方差率 σ_n^2 的变化，对式子（6-5）中的参数做些变化，主要的变化体下 3 个方面。

第一，ui 被定义为变量在第 i-1 个交易日末至第 i 个交易日末的百分比变化，类似涨跌幅比例，具体就是：

$$u_i = \frac{S_i - S_{i-1}}{S_{i-1}}$$

第二，$\bar{u}$ 假设是等于零，这样的处理在金融领域比较常见，比如股票收益率的均值就可以假设等于零。

第三，用 m 来代替 m-1

以上 3 个变化对最终计算的结果影响并不大，同时最重要的是式子（6-5）可以简化为

$$\sigma_n^2 = \frac{1}{m}\sum_{i=1}^{m} u_{n-j}^2$$

6.3.1 估计波动率

由于是估计当前波动率σ_n，对于距离估计日最近的数据（比如u_{n-1}^2, u_{n-2}^2，等）应该给予高的权重，而对于比较久远的数据（比如u_{n-m}^2等）给予较低的权重，这样或许更符合实际情况。

设想出以下的这个模型：

$$\sigma_n^2 = \sum_{i=1}^{m} \alpha_i u_{n-i}^2 \qquad (6\text{-}10)$$

其中，变量α_i是从现在往前推算的第 i 天（交易日）观察值u_{n-i}^2所对应的权重，有 3 个特征：一是α_i均取正值，即$\alpha_i>0$；二是如果 i>j，则取值$\alpha_i<\alpha_j$，也就是对于较久远的数赋予较小的权重；三是权重之和设定等于 1；

$$\sum_{i=1}^{m} \alpha_i = 1$$

对于式子（6-10）做进一步的推广。假定存在某一个长期平均的方差率 V_L，并给予该长期平均的方差率一定权重 γ，式子（6-10）就变为

$$\sigma_n^2 = \gamma V_L + \sum_{i=1}^{m} \alpha_i u_{n-i}^2 \qquad (6\text{-}12)$$

由于所有的权重之和依然是等于 1，因此就有等式

$$\gamma + \sum_{i=1}^{m} \alpha_i = 1 \qquad (6\text{-}13)$$

平稳性要求

式子（6-12）和式子（6-13）所构成的模型就是由罗伯特·恩格尔（ Robert Engle 最先提出的 ARCH（m）模型（简称“ ARCH 模型”，这里的 m 就对应观测到的 m 个交易日。

6.3.1 估计波动率

在 ARCH 模型的基础上，Bollerslev 提出了 GARCH 模型，并且最基础的模型就是 GARCH（1,1）模型，GARCH（1,1）模型表达式为

$$\sigma_n^2 = \gamma V_L + \alpha u_{n-1}^2 + \beta \sigma_{n-1}^2 \quad (6\text{-}15)$$

其中，VL 依然表示长期平均方差率，γ 依然表示对应于 VL 的权重，a 是对应 u_{n-1}^2 的权重，β 是对应于 σ_{n-1}^2 的权重。所有的权重之和依然等于 1，也就是

$$\gamma + \alpha + \beta = 1$$

通过式子（6-15）不难发现，在 GARCH（1，1）模型中，σ_n^2 是由长期平均方差率 VL、越近一个交易日（第 n-1 个交易日）变量的百分比变动 u_{n-1} 以及波动率估计值 σ_{n-1} 共同确定的。

此外， GARCH（1，1）模型中的第 1 个 1 是代表模型中变量变化的百分比（u_{n-1}）是选择最近一个交易日（第 n-1 个交易日），第 2 个 1 是指模型中变量波动率估计值（σ_{n-1}）也选择最近一个交易日。

GARCH 模型的一般表达式是 GARCH（p，q）模型，具体的公式为

$$\sigma_n^2 = \gamma V_L + \sum^{p} \alpha_i u_{n-i}^2 + \sum^{q} \beta_j \sigma_{n-j}^2 \quad (6\text{-}16)$$

令 $\omega = \gamma V_L$ 我们可以将GARCH(1,1)模型写成

$$\sigma_n^2 = \omega + \alpha u_{n-1}^2 + \beta \sigma_{n-1}^2 \quad (6\text{-}17)$$

在估计模型参数时，通常会采用这种形式。一旦估计出 ω, α 和 β 后，我们可由 $\gamma = 1 - \alpha - \beta$ 来计算 γ，而长期方差 $V_L = \omega / \gamma$。为了保证GARCH(1,1)模型的稳定，我们需要 $\alpha + \beta < 1$，否则对应于长期方差的权重会为负值。

6.3.1 估计波动率

标准的GARCH(1,1)模型：

$$y_t = x_t\gamma + u_t \quad \text{均值方程}$$

$$\delta_t^{\,2} = \omega + \alpha\ u_{t-1}^2 + \beta\ \delta_{t-1}^2 \quad \text{条件方差方程}$$

模型的经济学含义：

投资者常常通过建立长期均值的加权平均（ α_0 ）、上期的预期方差（GARCH项）、以前各期中观察到的关于变动性的信息（ARCH项）来预测本期的方差。因此，收益的巨大变化可能伴随着更进一步的巨大变化。

估计长期波动率：

$$\begin{aligned}\mathrm{Var}(u_t) &= E(u_t^2) = E[\delta_t^2 E(\varepsilon_t^2)] = \mathrm{E}(\delta_t^2)\\ &= \mathrm{E}[\omega + \alpha\ u_{t-1}^2 + \beta\ \delta_{t-1}^2]\\ &= \omega + \alpha E(u_{t-1}^2) + \beta E[E(u_{t-1}^2/F_{t-2})]\\ &= \omega + (\alpha+\beta)E(u_{t-1}^2)\end{aligned}$$

令$E(u_t^2) = E(u_{t-1}^2)$:

$$\mathrm{Var}(u_t) = E(u_t^2) = \frac{w}{1-\alpha-\beta}$$

6.3.2 arch模块

arch模块是一个计算波动率模型和其他金融计量模型的 Python第三方模块，目前主要功能包括单变量波动率模型、拔靴或自举、多对比分析过程以及单位根检验。

由于该模块未能集成在Anaconda3版本中，因此需要自行下载arch模块并安装，或者直接打开Anaconda Prompt界面并且输入pip install arch进行在线安装。

下面通过导入arch模块并且查看版本信息，具体的代码如下：

```
import arch
arch.__version__
Out[35]: '4.15'
```

6.3.2 arch模块

在arch模块中，构建ARCH模型和GARCH模型需要运用到arch_model函数，该函数的形式如下：

arch_model（y, x, mean, lags, vol, p, o, q, dist)

参数名称	功能和用法
y	因变量，也就是拟分析的波动率变量的样本值
x	外生变量（ Exogenous regressors）。如果未输入则模型自动省略
mean	均值模型的类型，具体可选类型如下： ‘Constant'：表示平均方差是一个常数 ‘Zero'：表示平均方差是零； ‘ARX'：表示带外生变量的自回归模型（ AutoRegressive eXogenous,ARX）
lags	表明滞后项的阶数，默认值为0
vol	表示波动率模型的类型，具体可选的类型包括 GARCH'（默认）、 ‘ARCH’、 ‘EGARCH’、’ FIARCH'以及" HARCH
p	对称随机数的滞后项阶数，默认值为1
o	非对称数据的滞后项阶数，默认值为0
q	波动率或对应变量的滞后项阶数，默认值为1
dist	表示误差项服从的分布类型，可选类型如下： ‘normal’或者 ‘gaussian'：代表正态分布，并且是默认值 ‘t’或' studentst'；代表学生t分布； ‘skewstudent'或 ‘skewt'：代表偏态学生t分布； ‘ged’或 ‘generalized error’：代表通用误差分布

6.3.2 arch模块

【例6-13】沿用前面例6-6的相关信息，对2016年至2018年沪深300指数的涨跌幅构建波动率模型，选用的模型是ARCH（1）模型和 GARCH（1，1）模型，具体的过程分为5个步骤。

第1步：从arch模块中导入 arch_model函数，并且选择相关的参数进行输入，具体的代码如下：

```
from arch import arch_model                                    #从arch模块中导入arch_model函数
model_arch=arch_model(y=HS300_sz180.iloc[:,0],mean='Constant',lags=0,vol='ARCH',p=1,o=
0,q=0,dist='normal')
#构建ARCH(1)模型
model_garch=arch_model(y=HS300_sz180.iloc[:,0],mean='Constant',lags=0,vol='GARCH',p=1,
o=0,q=1,dist='normal')
#构建GARCH(1,1)模型
```

6.3.2 arch模块

第2步：对ARCH（1）模型进行拟合并且输出模型的参数，具体的代码如下：

```
result_arch=model_arch.fit()                #对ARCH模型进行拟合
result_arch.summary()
```

第3步：对 GARCH（1,1）模型进行拟合并且输出模型的参数，具体的代码如下：

```
result_garch=model_garch.fit()              #对GARCH模型进行拟合
result_garch.summary()                      #对拟合结果进行输出
```

第4步：可以通过函数 params输出模型的相关参数并且对参数进行运算，具体的代码如下

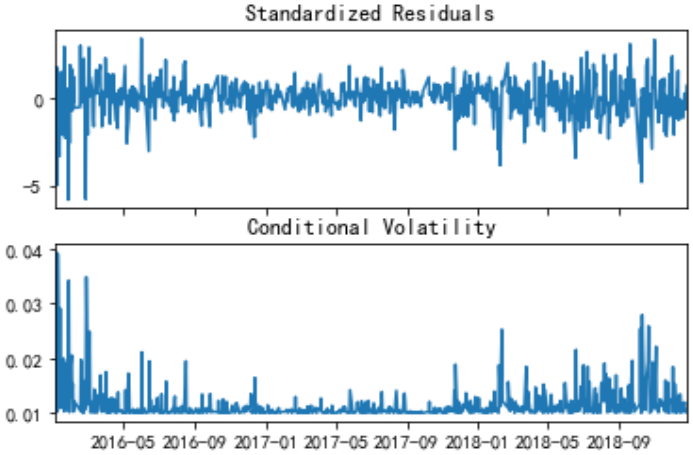
```
result_garch.params

vol=np.sqrt(result_garch.params[1]/(1-result_garch.params[2]-result_garch.params[3]))
print('利用GARCH(1,1)模型得到的长期波动率（每日）：',round(vol,4))
```

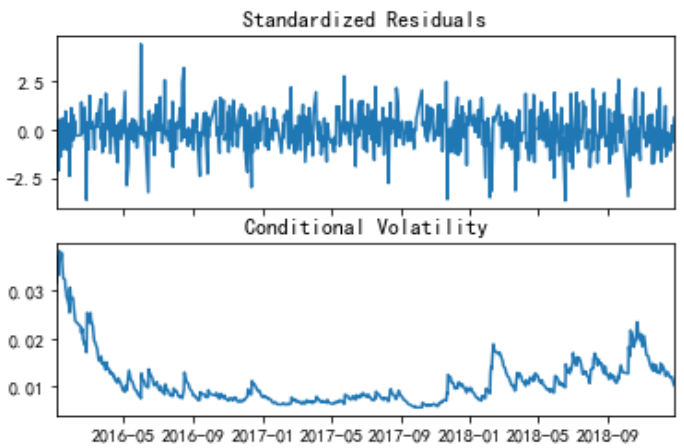
6.3.2 arch模块

第5步：将结果进行可视化。运用内嵌的plot函数将标准化残差和条件波动率通过图形方式显示出来（见图6-4和图6-5）。具体的代码如下：

```
result_arch.plot()                    #ARCH模型结果的可视化
```




```
result_garch.plot()                   #GARCH模型结果的可视化
```



6.4 datetime模块

6.4 datetime模块

金融变量的取值往往和时间是密不可分的，在 Python中，有一个内置的专门处理时间的模块datetime，该模块以简单的方式提供日期和时间，不仅支持日期和时间的算法，而且也能实现有效的属性提取并用于格式输出和操作。

datetime模块主要包含五大类，具体如表6-13所示

类名	功能说明
date	以日期作为对象，常用的属性包括year（年），month（月）,day（日）
time	以时间作为对象，常用的属性包括hour（小时)、minute（分钟） sceond（秒） microsecond(微秒）和 tzifo（时区)
datetime	以日期和时间作为对象，是date和time的结合
datetime_CAPI	也是以日期时间为对象，不过是C语言的接口
timedelta	时间间隔，也就是两个不同时点间的长度
tzinfo	时区信息对象

首先依然需要导入 datetime模块，具体的代码如下：

```
import datetime as dt                    #导入datatime模块
```

6.4.1 创建时间的对象

【例6-14】在 Python中输入2018年12月28日，具体的代码如下：

```
T1=dt.datetime(2018,12,28)
T1
Out[45]: datetime.datetime(2018, 12, 28, 0, 0)
```

【例6-15】在 Python中输入2018年8月8日下午14点38分58秒88微秒，具体的代码如下：

```
T2=dt.datetime(2018,8,8,14,38,58,88)
T2
Out[46]: datetime.datetime(2018, 8, 8, 14, 38, 58, 88)
```

此外，可以用now和 today函数创建当前的时间对象。

【例6-16】在 Python中创建当前的时间对象，具体的代码如下：

```
now=dt.datetime.now()
today=dt.datetime.today()
now
Out[48]: datetime.datetime(2020, 10, 31, 20, 52, 50, 638426)

today
Out[49]: datetime.datetime(2020, 10, 31, 20, 52, 50, 638427)
```

需要注意的是，微秒的取值区间是0-1000000。

6.4.2 访问时间对象的属性

属性	说明	时间对象T2为例的代码
year	时间对象的年份	T2.year Out[50]: 2018
month	时间对象的月份	T2.month Out[51]: 8
weekday	时间对象处于星期几	T2.weekday() Out[52]: 2 结果显示为星期三，因为0代表星期一，1代表星期二，以此类推
day	时间对象处于当月的日数	T2.day Out[53]: 8
isocalendar	时间对象的以ISO标准化日期的方式显示，显示方式是年份、当年的周数以及星期数	T2.isocalendar() Out[55]: (2018, 32, 3) 结果显示是2018年第32周的周三
date	时间对象的日期	T2.date() Out[54]: datetime.date(2018, 8, 8)
hour	时间对象的小时数	T2.hour Out[56]: 14
minute	时间对象的分钟数	T2.minute Out[57]: 38
second	时间对象的秒数	T2.second Out[58]: 58
microsecond	时间对象的微秒数	T2.microsecond Out[59]: 88
ctime	时间对象以字符串方式输出，输出的内容依次是“星期数、月份、日期数、时/分/秒、年份”	T2.ctime() Out[60]: 'Wed Aug 8 14:38:58 2018' now.ctime() Out[61]: 'Sat Oct 31 20:52:50 2020'

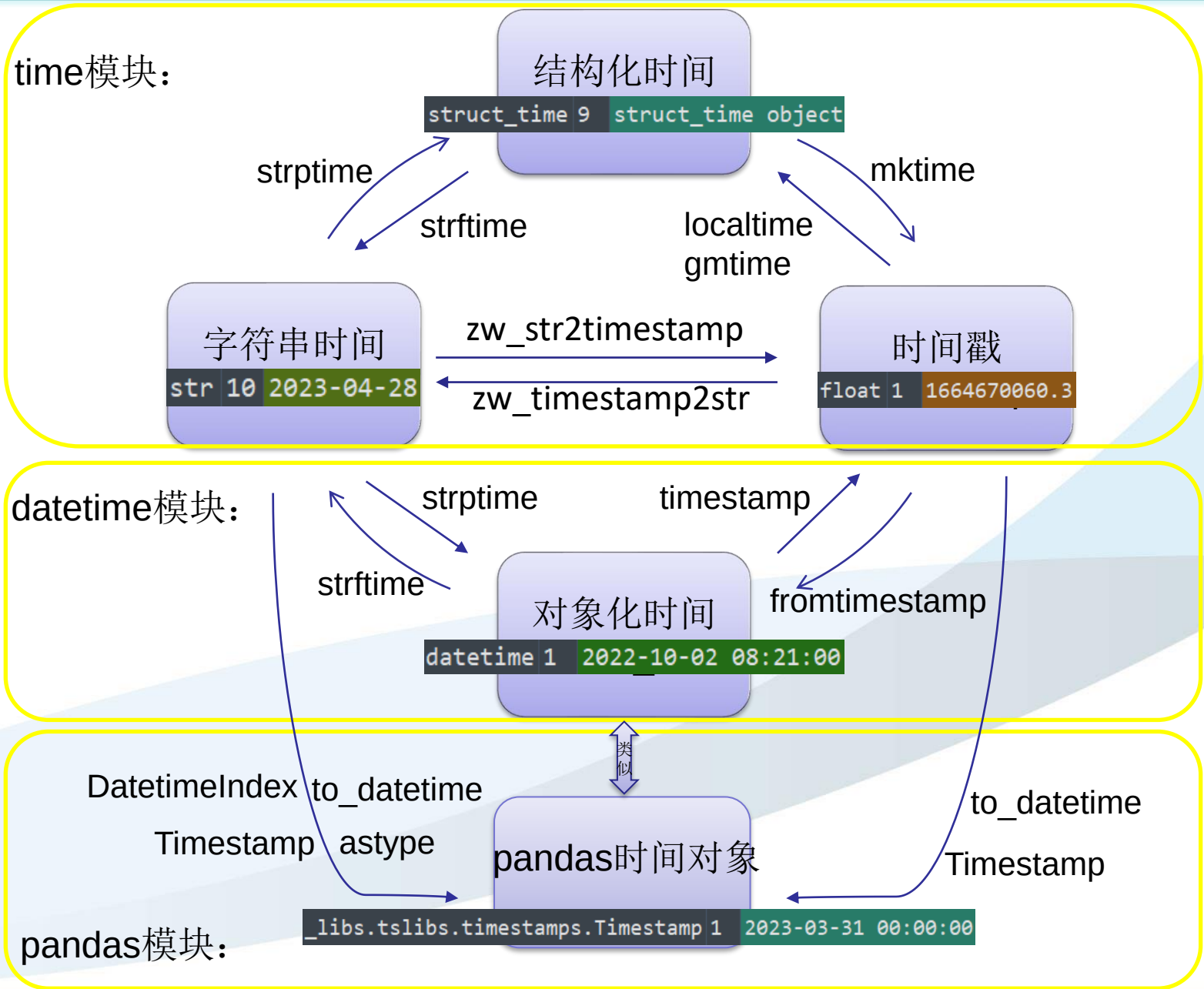
6.4.3 时间对象的运算

datetime模块内置函数	Python比较运算符号	含义	Python的代码
__eq__()	==	等于	T1.__eq__(T2) Out[62]: False T1==T2 Out[63]: False
__ge__()	>=	大于等于	T1.__ge__(T2) Out[65]: True T1>=T2 Out[66]: True
__gt__()	>	大于	T1.__gt__(T2) Out[67]: True T1>T2 Out[68]: True
__le__()	<=	小于等于	T1.__le__(T2) Out[69]: False T1<today Out[70]: True
__lt__()	<	小于	T2.__lt__(today) Out[71]: True T2<today Out[72]: True
__ne__()	!=	不等于	T2.__ne__(today) Out[73]: True T2!=today Out[74]: True

6.4.3 时间对象的运算

函数	说明	Python的代码
days	计算间隔天数	T_delta=T1-T2 T_delta.days Out[76]: 141 该结果显示两个时间对象相距141天
seconds	计算间隔秒数，输出结果的取值范围是大于0且小于86400（即1天对应的秒数）	T_delta2=today-T2 T_delta2.seconds Out[78]: 22432 该结果显示的是时间对象today中的时间与时间对象T2中的14点38分58秒之间的间隔秒数
microseconds	计算间隔的微秒数，输出结果的取值范围是大于0且小于1000000即秒对应的微秒数）	T_delta2.microseconds Out[79]: 638339 该结果显示的是时间对象today中的第638339微秒与时间对象T2中的第88微秒之间的间隔微秒数

6.4.4 时间处理


time模块：
结构化时间
struct_time 9 struct_time object
strptime
strftime
mktime
localtime
gmtime
字符串时间
str 10 2023-04-28
zw_str2timestamp
zw_timestamp2str
时间戳
float 1 1664670060.3
datetime模块：
strptime
strftime
timestamp
fromtimestamp
对象化时间
datetime 1 2022-10-02 08:21:00
类似
DatetimeIndex
Timestamp
to_datetime
astype
pandas时间对象
to_datetime
Timestamp
_libs.tslibs.timestamps.Timestamp 1 2023-03-31 00:00:00
pandas模块：

Your appreciation makes me a miracle. Thank you!

Frank Ziwei Zhang
18117228563
frank8027@163.com

上海对外经贸大学
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS