# 第 1 章 Volatility 波动率

> 对于了解自己的投资者来说，波动率创造了机会。
> ***For the investor who knows what he is doing, volatility creates opportunity.***
>
> ——约翰 • 特雷恩 (John Train)

> 在低的波动率下持续超越市场是不可能的。我在30多年里超越了市场，但这不是在低的波动率下。
> ***Outperforming the market with low volatility on a consistent basis is an impossibility. I outperformed the market for 30-odd years, but not with low volatility.***
>
> ——乔治 • 索罗斯 (George Soros)

“波浪无时潮失信”，金融市场或许会有风平浪静，但是更多的是“失信”的潮水和“无时”的波浪。在金融市场上，无论是股票、利率、汇率、大宗商品的价格，甚至波动本身，每时每刻都涌动着“潮水”和“波浪”，金融市场本身就是一个波动的市场。

波动率作为衡量金融资产价格波动程度的一个指标，毫无疑问地成为金融领域一个非常重要的概念，它不仅反映了资产收益的不确定性，而且也反映了金融资产的风险水平。波动率越高，金融资产价格的变动越剧烈，资产收益率的不确定性和金融风险就越高；波动率越低，金融资产价格的变动越平缓，资产收益率的确定性就越强，对应的金融风险就越低。

## Core Functions and Syntaxes 本章核心命令代码

- `arch.arch_model().fit()` ARCH模型拟合
- `arch.archmodel.params` 打印输出模型参数
- `ax.get_xlim()` 获取x轴范围
- `ax.get_ylim()` 获取y轴范围
- `ax.hist()` 绘制柱状图
- `DataFrame.cumprod()` 计算累积回报率
- `DataFrame.ewm()` 计算指数权重移动平均
- `DataFrame.expanding().std()` 产生扩展标准差
- `DataFrame.ffill()` 按前差法补充数据
- `DataFrame.pct_change()` 生成回报率
- `matplotlib.pyplot.gca().xaxis.set_major_formatter()` 设定主标签格式
- `mdates.DayLocator()` 设定日期选择
- `pandas.to_datetime()` 转换为日期格式
- `Series.resample()` 对序列重新组合，可选择周、双周、月等参数
- `Series.rolling().std()` 产生流动标准差

# 1.1 回报率

回报率是一个非常宽泛的概念，比如经常提到的**投资回报率** (Return On Investment, ROI)，顾名思义是指通过投资而获得的回报，它利用投资的增量与初始投资的比值来表示，如图1-1所示。

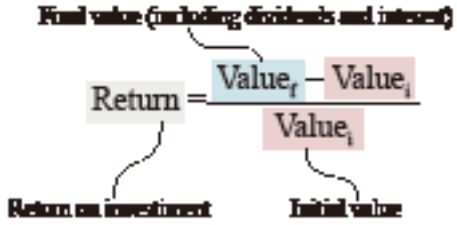



图1-1　投资回报率

其中，投资的增量 $\text{Value}_f - \text{Value}_i$ 被称为**净回报** (net return)。

下面以某股票的价格为例详细讨论股票的回报率。这里只考虑**工作日** (Business Day) 的收盘价。如图1-2所示，$t$时刻的股票价格为$S_t$，相应的，$t-i$时刻的股票价格为$S_{t-i}$。如果已知$t$时刻和$t-1$时刻的股票价格，通过式(1-1)可以计算出相应的**损益** (Profit and Loss, PnL, P&L)。

$$\text{PnL}_t = S_t - S_{t-1} \tag{1-1}$$

在不考虑分红的情况下，如果$\text{PnL}_t > 0$，则$t-1$时刻买进的股票，$t$时刻卖出，投资者会从股票获利；反之，如果$\text{PnL}_t < 0$，$t-1$时刻买进的股票，$t$时刻卖出，投资者则会由于投资该股票而导致亏损。

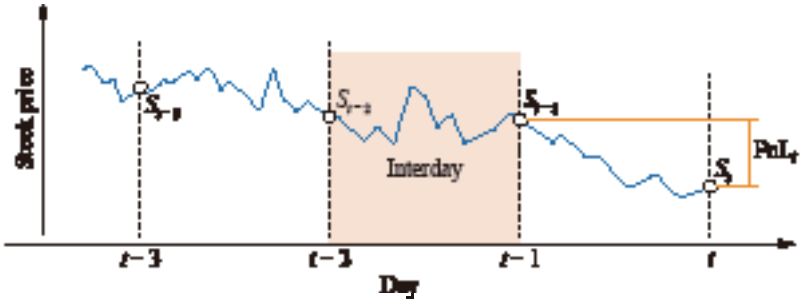



图1-2　某股票的价格变动

在没有**分红** (dividend) 的情况下，单日**简单回报率** (simple return) 可以通过式(1-2)计算。

$$r_t = \frac{S_t - S_{t-1}}{S_{t-1}} \tag{1-2}$$

通过下面的例子可以帮助理解日简单回报率。首先利用如下代码，下载亚马逊公司 (Amazon) 在2020年12月21日到28日的股票的调整收盘价格。

B2_Ch1_1_A.py

```
import numpy as np
import pandas_datareader
```

```
ticker = 'AMZN'
stock = pandas_datareader.data.DataReader(ticker, data_source='yahoo',
start='12-21-2020', end='12-28-2020')['Adj Close']
print(stock)
```

展示结果如下。

```
Date
2020-12-21    3206.179932
2020-12-22    3206.520020
2020-12-23    3185.270020
2020-12-24    3172.689941
2020-12-28    3283.959961
Name: Adj Close, dtype: float64
```

根据前面对于回报率介绍的公式，用Python写出计算式，并计算日简单回报率。

B2_Ch1_1_B.py

```
#via formula
returns_daily = (stock / stock.shift(1)) - 1
print(returns_daily)
```

日简单回报率如下。

```
Date
2020-12-21         NaN
2020-12-22    0.000106
2020-12-23   -0.006627
2020-12-24   -0.003949
2020-12-28    0.035071
Name: Adj Close, dtype: float64
```

Python中还提供了一个单独的函数pct_change() 来计算日简单回报率，如果借助这个函数，代码如下所示。

B2_Ch1_1_C.py

```
#alternative via pct_change() function
returns_daily = stock.pct_change()
print(returns_daily)
```

得到的日简单回报率结果如下所示，可见这与前面根据公式计算的结果完全相同。

```
Date
2020-12-21         NaN
2020-12-22    0.000106
2020-12-23   -0.006627
2020-12-24   -0.003949
2020-12-28    0.035071
Name: Adj Close, dtype: float64
```

另外，对于股价的分析，常常会用到对数回报率，其数学表示式为：

$$r_t = \ln\left(\frac{S_t}{S_{t-1}}\right) = \ln(S_t) - \ln(S_{t-1}) \tag{1-3}$$

相应地，下面代码可以用来计算对数回报率。

B2_Ch1_1_D.py

```
#log return
log_return_daily = np.log(stock / stock.shift(1))
print(log_return_daily)
```

对数回报率结果如下。

```
Date
2020-12-21          NaN
2020-12-22     0.000106
2020-12-23    -0.006649
2020-12-24    -0.003957
2020-12-28     0.034470
Name: Adj Close, dtype: float64
```

回报率是经过一段时间的回报，前面例子介绍的是时间为一天的回报，如果经过的时间为$k$天，那么简单回报率公式为：

$$\begin{aligned} r_t(k) &= \frac{S_t - S_{t-k}}{S_{t-k}} \\ &= \frac{S_t}{S_{t-k}} - 1 \\ &= \frac{S_t}{S_{t-1}}\frac{S_{t-1}}{S_{t-2}}\cdots\frac{S_{t-k+1}}{S_{t-k}} - 1 \\ &= (r_t + 1)(r_{t-1} + 1)\ldots(r_{t-k+1} + 1) - 1 \end{aligned} \tag{1-4}$$

相应地，单周简单回报率可以通过式(1-5)求得。

$$r_t(5) = \frac{S_t - S_{t-5}}{S_{t-5}} \tag{1-5}$$

双周简单回报率可以通过式(1-6)求得。

$$r_t(10) = \frac{S_t - S_{t-10}}{S_{t-10}} \tag{1-6}$$

单月简单回报率可以通过式(1-7)求得。

$$r_t(20) = \frac{S_t - S_{t-20}}{S_{t-20}} \tag{1-7}$$

下面的代码从**弗莱德数据库** (FRED：https://fred.stlouisfed.org) 获得从2010年12月28日到2020年12月28日十年间**标普指数** (S&P500) 的价格数据，并进行了绘图展示。

B2_Ch1_2_A.py

```
import pandas_datareader
import matplotlib.pyplot as plt

#sp500 price
sp500 = pandas_datareader.data.DataReader(['sp500'], data_source='fred',
start='12-28-2010', end='12-28-2020')
#plot sp500 price
plt.plot(sp500['sp500'], color='dodgerblue')
plt.title('S&P 500 price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

如图1-3所示为上述代码运行结果。从图中可以看出，标普指数的价格总体趋势为上升，这与全球经济在这十年中的走势相符。但是，价格的波动始终伴随其中，有时甚至会有巨大的上升或者下降。比如，从图中明显可见，2020年标普指数的价格出现了断崖式下降，这是因为新型冠状病毒对全球经济的巨大冲击。

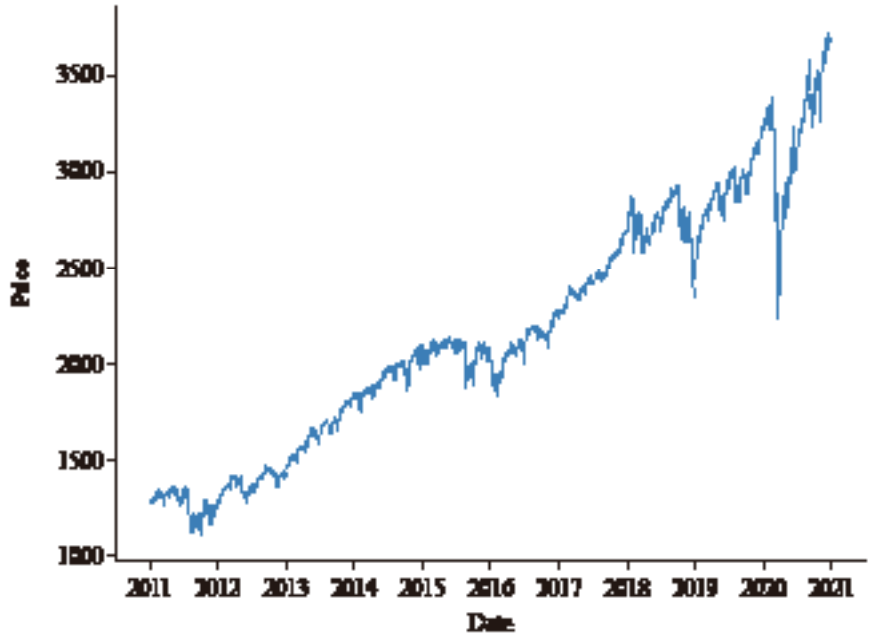



图1-3　标准普尔价格

套用最初介绍的日简单回报率计算公式，可以用下面代码，绘制标普指数的日回报率曲线，如图1-4所示。

B2_Ch1_2_B.py

```
#daily return
sp500['return_daily'] = sp500['sp500'].pct_change()
sp500.dropna(inplace=True)
#plot daily return
plt.plot(sp500['return_daily'], color='dodgerblue')
plt.title('S&P 500 daily returns')
plt.xlabel('Date')
plt.ylabel('Daily return')
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

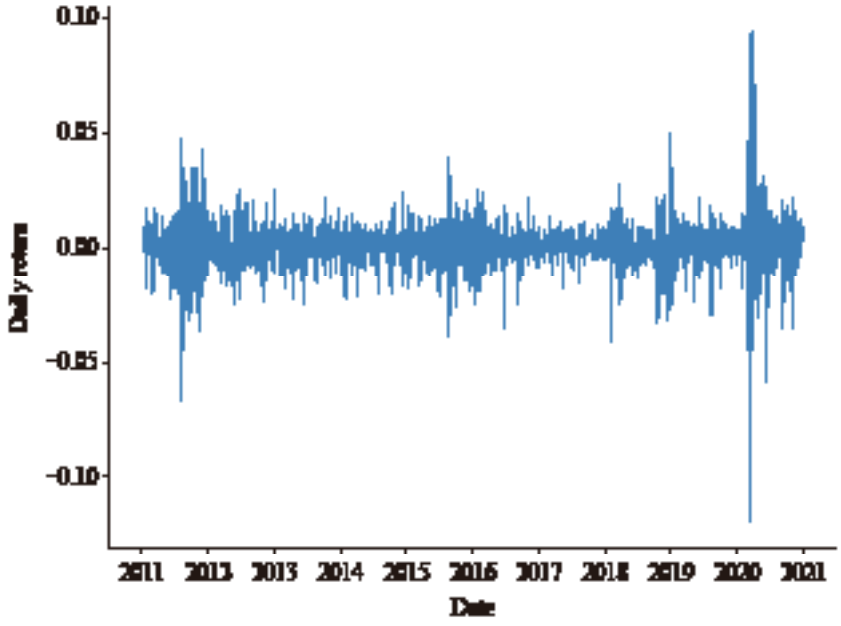



图1-4　标普指数日回报率

利用下面代码，可以很容易地计算月回报率，并绘制如图1-5所示的曲线图。代码中函数resample('M') 可以计算月回报率，如果需要得到周回报率、双周回报率，将该函数的参数相应改为'W'、'BW'即可。这个函数非常有用，大家可以修改代码进行尝试，以更好掌握该函数。

B2_Ch1_2_C.py

```
#monthly return
sp500_monthly_returns = sp500['sp500'].resample('M').ffill().pct_change()
#plot monthly return
plt.plot(sp500_monthly_returns, color='dodgerblue')
plt.title('S&P 500 monthly returns')
plt.xlabel('Date')
plt.ylabel('Monthly return')
```

```
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

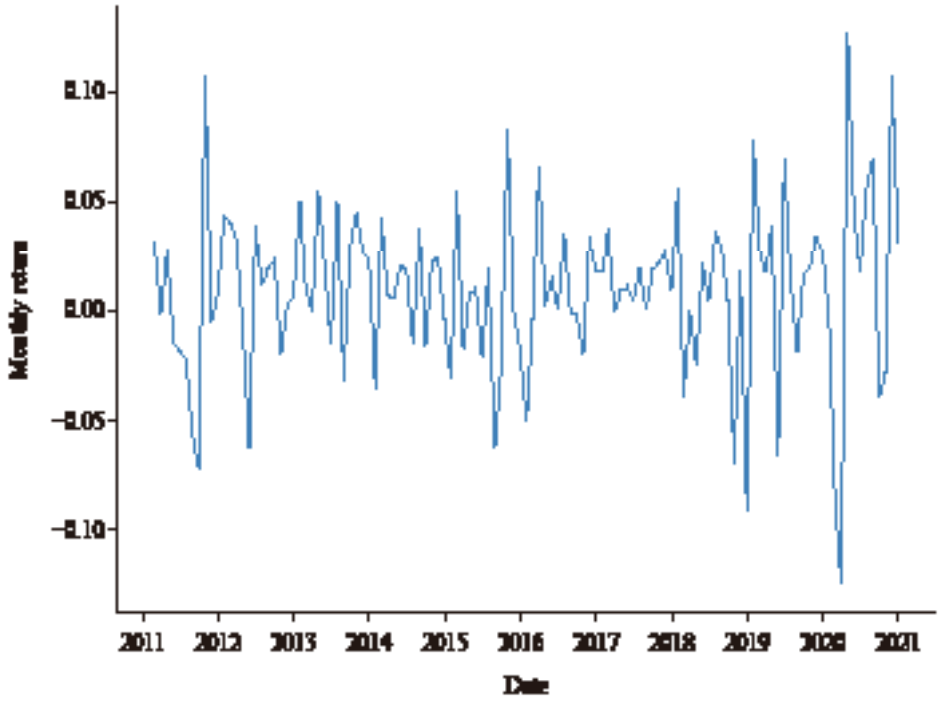



图1-5　标普指数月回报率

前面介绍的日回报率、月回报率等可以帮助理解投资回报的单日或单月等的波动程度，为了计算投资的回报，通常要利用总回报，这就需要计算**累积回报率** (cumulative return)。下面代码，利用 cumprod() 函数计算得到累积回报率，并绘制了如图1-6所示的曲线图。

B2_Ch1_2_D.py

```
#daily cumulative return
sp500_cum_returns_daily = (sp500['return_daily'] + 1).cumprod()
#plot daily cumulative return
plt.plot(sp500_cum_returns_daily, color='dodgerblue')
plt.title('S&P 500 daily cumulative returns')
plt.xlabel('Date')
plt.ylabel('Cumulative return')
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

大家可以发现，图1-6与代表标普指数价格的图1-3除了坐标以外，完全一致，这是因为累积回报实际上是价格的标准化，即假定初始投资为一个货币单位，之后得到的回报。

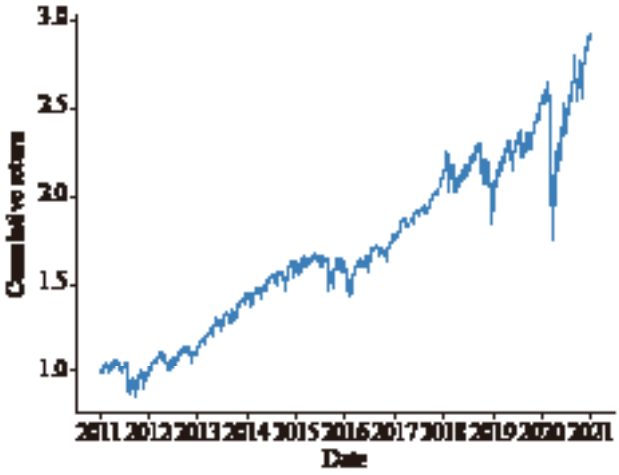



图1-6　标普指数累积日回报率

类似的，利用以下代码可以计算并可视化累积月回报率。

B2_Ch1_2_E.py

```
#monthly cumulative return
sp500_cum_returns_monthly = (sp500_monthly_returns + 1).cumprod()
#plot monthly cumulative return
plt.plot(sp500_cum_returns_monthly, color='dodgerblue')
plt.title('S&P 500 daily cumulative returns')
plt.xlabel('Date')
plt.ylabel('Cumulative return')
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

如图1-7所示即为上述代码生成的累积月回报率曲线，由于平均效应，曲线要比日累积回报率平滑许多。

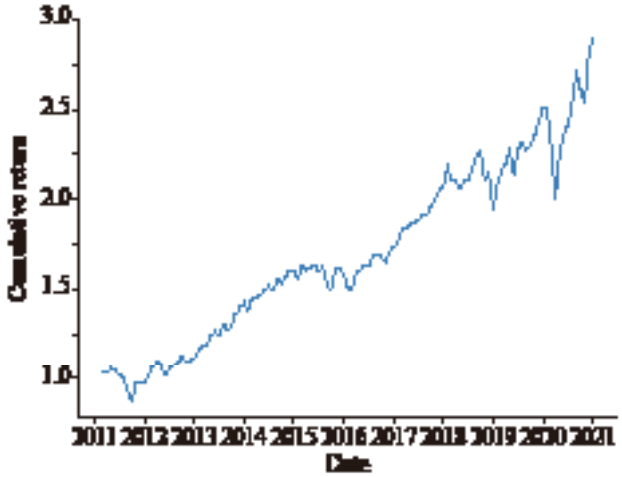



图1-7　标普指数累积月回报率

在实际工作中，经常遇到数据量不足的问题。比如，假设一年有252个工作日，如果每周只采集一次单周回报率，只能得到50个数据。但是，如果每天都向前回溯一周，采集一个单周回报率，即每天都能得到新的过去一周的回报率。也就是，以周为单位的数据窗口每天都随着时间不停向前移动，这样在一年之中就可以得到247个周回报率数据。这种周回报率也称为**重叠** (overlapping) 单周回报率，类似的，其概念也可以用到其他的时间单位上，比如月、季度和年。相较于**非重叠** (non-overlapping) 回报率，重叠回报率大大增加了数据量。但需要注意的是，由于数据存在较高的自相关性，重叠回报率样本序列的波动率会降低。

非重叠单周回报率和重叠单周回报率如图1-8所示。

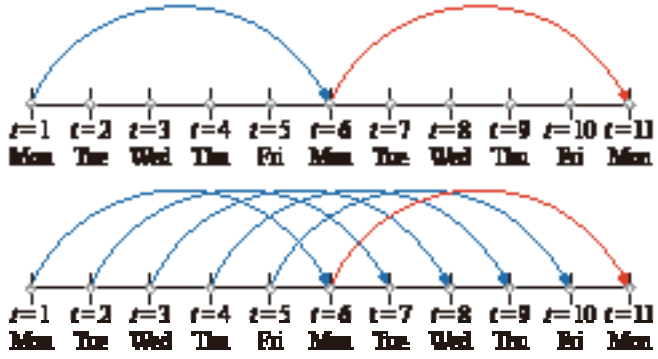



图1-8　非重叠单周回报率和重叠单周回报率　(本图来自MATLAB系列丛书第二本第1章)

前面的讨论均没有涉及存在分红的情况，那么如果考虑分红，股票的回报率则可以由公式(1-8)计算得到。

$$y_t = \frac{S_t - S_{t-1} + D_t}{S_{t-1}} \tag{1-8}$$

其中，$D$为**分红收益率** (dividend rate)。

在回报率已知的情况下，可以很容易地计算投资的损益，比如，对于某个**投资组合** (investment portfolio)，如果只包含有同一股票，当前这个投资组合的价值为$A_t$，那么$t$时刻投资组合的损益可以通过式(1-9)计算得到。

$$Q_t = A_t \frac{S_t - S_{t-1}}{S_{t-1}} \tag{1-9}$$

对于回报率，既可以用**小数** (decimal) 表示，也可以用**百分数** (percentage) 来表示。前面的介绍，主要涉及简单回报率，但是也提及了对数回报率。对数回报率实质上是**连续回报率** (continuously compounded return)。通常来说，简单回报率广泛用于各种会计计算，而连续回报率则在各种数学模型中得到大量应用。

连续回报率与简单回报率有着密切的关系。比如，当$S_t/S_{t-1}$的比值很小时，对数回报率$\ln(S_t/S_{t-1})$和$(S_t - S_{t-1})/S_{t-1}$很接近。此外，采用连续回报率可以简化多阶段收益率的计算。对于横跨几个时间单位的多期连续回报率，可以通过式(1-10)得到。

$$\begin{aligned} r_t(k) &= \ln\left(\frac{S_t}{S_{t-k}}\right) \\ &= \ln(S_t) - \ln(S_{t-k}) \\ &= \left[\ln(S_t) - \ln(S_{t-1})\right] + \left[\ln(S_{t-1}) - \ln(S_{t-2})\right] + \cdots + \left[\ln(S_{t-k+1}) - \ln(S_{t-k})\right] \\ &= r_t + r_{t-1} + \cdots + r_{t-k+1} \end{aligned} \tag{1-10}$$

下面的公式变换，展示了连续回报率和简单回报率的关系，在这里为了区别，连续回报率标记为$r_t$，简单回报率标记为$y_t$。

$$r_t = \ln\left(\frac{S_t - S_{t-1} + S_{t-1}}{S_{t-1}}\right) = \ln(y_t + 1) \tag{1-11}$$

另外，通过观察下面的泰勒展开式，也可以看到在回报率数值较小的情况下，连续回报率和简单回报率的差别非常小。

$$\begin{aligned}\ln(y_t + 1) &= \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(y_t)^n \\ &= y_t - \frac{y_t^2}{2} + \frac{y_t^3}{3} - \frac{y_t^4}{4} + \frac{y_t^5}{5}\cdots, \forall y_t \in (-1, +\infty)\end{aligned} \tag{1-12}$$

# 1.2 历史波动率

**波动率** (volatility) 是用统计的方法对资产价格偏离基准程度的一个度量，通常用希腊字母$\sigma$表示。在金融数学中，波动率实质就是资产价格变化的标准差。期权定价中使用的波动率通常以一年作为时间单位，因此此时波动率为一年连续复利回报率的标准差；而风险控制领域的波动率通常以一天为时间单位，此时的波动率对应每天连续复利回报率的标准差。

如图1-9所示，波动率一般可以分为两种：一种是利用历史数据计算得到，称为**历史波动率** (historical volatility)，它是通过回溯并分析历史上已经出现的价格而得到的，因此也叫**回望波动率** (backward looking volatility)；另外一种是根据当前市场的期权价格，用Black-Scholes期权定价模型反推出来，称为**隐含波动率** (implied volatility)，它是对资产未来价格波动率的预测，因此也称为**前瞻波动率** (forward looking volatility)。本节中会对它们分别进行详细介绍。

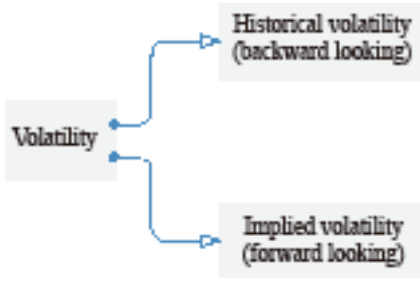



图1-9　波动率分类

在计算波动率时，通常使用交易的天数，而不是日历天数，这是因为波动率植根于交易，累积的是不同交易日的“交易的不确定性”，当然，也有解释认为波动率的值在交易日要远高于非交易日，因此在波动率的计算中，非交易日可以忽略。在日回报独立同分布，且具有相同方差的假设下，$T$天回报的方差为$T$与日回报率的乘积，也就是说，$T$天回报的标准差为日回报标准差的$\sqrt{T}$倍。这与大家熟知的“不确定性随时间长度的平方根增长”这一法则是一致的。

使用单日连续回报率，单日波动率可以通过式(1-13)求得。

$$\sigma_{\text{daily}} = \sqrt{\frac{1}{N-1}}\sqrt{\sum_{i=1}^{N}(r_i - \mu)^2} \tag{1-13}$$

其中，$N$为样本数据个数，比如一年单日数据一般取工作日天数$N$ = 250或252；$r$为对数回报率，这里是单日对数回报率；$\mu$为对数回报率的平均值。

回报率的方差，也就是波动率的平方，可以通过式(1-14)求得。

$$\text{var} = \frac{\sum_{i=1}^{N}(r_i - \mu)^2}{N-1} \tag{1-14}$$

公式中的每个回报值减去所有回报值的均值得到的数据叫作**去均值数据** (demeaned data)，也叫作**均值中心化数据** (mean-centered data)。均值$\mu$则可以通过式(1-15)求得。

$$\mu = \frac{\sum_{i=1}^{N} r_i}{N} \tag{1-15}$$

一般情况下，这个平均值近似为0，往往可以忽略。另外，当$N$足够大时，$N-1$可以被$N$替换。因此，方差的公式可以简化为：

$$\text{var} = \frac{\sum_{i=1}^{N}(r_i)^2}{N} \tag{1-16}$$

因此，单日波动率的计算可以简化为：

$$\sigma_{\text{daily}} = \sqrt{\frac{\sum_{i=1}^{N}(r_i)^2}{N}} \tag{1-17}$$

下面的代码，从Fred数据库获取了标普指数一年的价格数据，计算了每天的对数回报率，并根据前面介绍的公式，计算得到年化回报率。

B2_Ch1_3.py

```
import matplotlib as mpl
import numpy as np
import matplotlib.pyplot as plt
import pandas_datareader

#sp500 price
sp500 = pandas_datareader.data.DataReader(['sp500'], data_source='fred',
start='12-28-2019', end='12-28-2020')

#daily log return
log_return_daily = np.log(sp500 / sp500.shift(1))
log_return_daily.dropna(inplace=True)

#calculate daily standard deviation of returns
daily_std = np.std(log_return_daily)[0]

#annualize daily standard deviation
std = daily_std * 252 ** 0.5
```

```
#Plot histograms
mpl.style.use('ggplot')
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
n, bins, patches = ax.hist(
    log_return_daily['sp500'],
    bins='auto', alpha=0.7, color='dodgerblue', rwidth=0.85)

ax.set_xlabel('Log return')
ax.set_ylabel('Frequency of log return')
ax.set_title('Historical volatility for SP500')

#get x and y coordinate limits
x_corr = ax.get_xlim()
y_corr = ax.get_ylim()

#make room for text
header = y_corr[1] / 5
y_corr = (y_corr[0], y_corr[1] + header)
ax.set_ylim(y_corr[0], y_corr[1])

#print historical volatility on plot
x = x_corr[0] + (x_corr[1] - x_corr[0]) / 30
y = y_corr[1] - (y_corr[1] - y_corr[0]) / 15
ax.text(x, y , 'Annualized volatility: ' + str(np.round(std*100, 1))+'%',
    fontsize=11, fontweight='bold')
x = x_corr[0] + (x_corr[1] - x_corr[0]) / 15
y -= (y_corr[1] - y_corr[0]) / 20

fig.tight_layout()
```

代码运行的结果如图1-10所示，年化回报率为35.2%。另外，从图中可以看到，通过分析回报率的具体分布情况，可以对投资情况进行考量和评估。

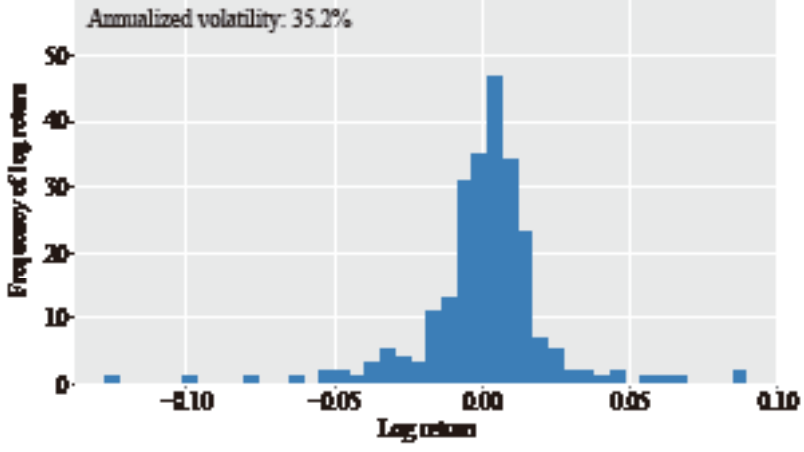



图1-10　标普指数回报率分布及历史年化波动率

# 1.3 移动平均（MA）计算波动率

**移动平均** (Moving Average, MA) 又称**滑动平均**、**滚动平均** (running average, rolling average)，是一个统计学的概念，是指通过创建总体数据中一系列子集的平均数来对总体数据进行分析的一种技术手段。移动平均可以分为简单移动平均、累积移动平均和加权移动平均等。

**简单移动平均** (Simple Moving Average, SMA) 是指周期性计算某确定数量数据的平均值，如图1-11所示。

$$\bar{p}_n = \frac{p_1 + p_2 + \cdots + p_n}{n} = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{1-18}$$

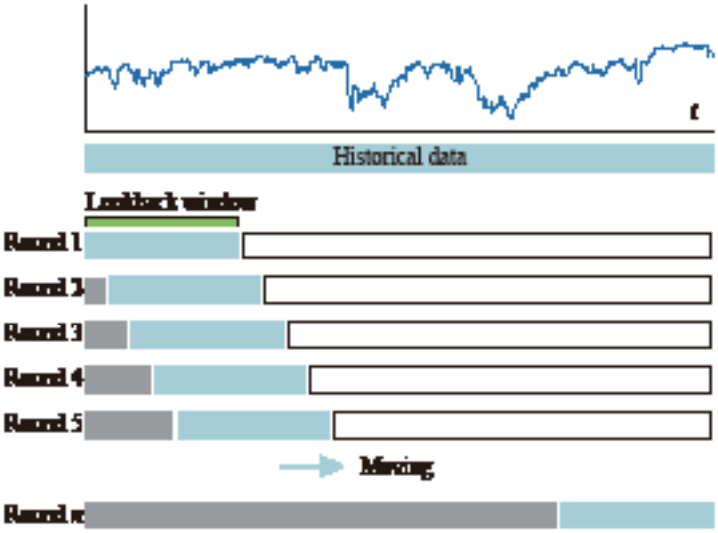



图1-11　简单移动平均

在每次计算时，剔除最老的数据，加入最新的数据，形成一个新的子数据，再进行取平均计算。但是，没有必要对整个子数据集进行求和再取平均的计算，可以直接借助上一步已经得到的均值，简化的计算公式为：

$$\bar{p}_n = \bar{p}_{n-1} + \frac{1}{n}\left(p_n - \bar{p}_{n-1}\right) \tag{1-19}$$

Pandas运算包提供了一种非常简单的方法来计算简单移动平均波动率，即通过rolling().std() 函数计算某**移动窗口** (rolling window) 的标准差，下面的代码利用获取的标普指数10年的历史数据，分别以5天、50天、100天和250天为移动窗口，绘制相应的历史波动率的曲线。

B2_Ch1_4.py

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

#sp500 price
```

```
df = pandas_datareader.data.DataReader(['sp500'],
data_source='fred', start='12-28-2010', end='12-28-2020')
df.dropna(inplace=True)

#daily log return
df['Daily return squared'] = np.log(df['sp500'] /
df['sp500'].shift(1))*np.log(df['sp500'] / df['sp500'].shift(1))
df.dropna(inplace=True)

#calculate simple moving average
win_list = [5, 50, 100, 250]
for win in win_list:
    ma = df['sp500'].rolling(win).std()
    df[win] = ma
    df.rename(columns={win:'Vol via '+str(win)+' days MA'}, inplace=True)

#plot dataframe
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 12))
#sp500 price
ax1.plot(df['sp500'])
ax1.set_title('SP500 price')
ax1.set_xlabel("Date")
ax1.set_ylabel("Price")
ax1.spines['right'].set_visible(False)
ax1.spines['top'].set_visible(False)
ax1.yaxis.set_ticks_position('left')
ax1.xaxis.set_ticks_position('bottom')
#daily log return squared
ax2.plot(df['Daily return squared'])
ax2.set_title('Daily return squared')
ax2.set_xlabel("Date")
ax2.set_ylabel("Daily return squared")
ax2.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.yaxis.set_ticks_position('left')
ax2.xaxis.set_ticks_position('bottom')
#ma vol
ax3.plot(df.loc[:, (df.columns != 'sp500') & (df.columns != 'Daily
return squared')])
ax3.legend(df.loc[:, (df.columns != 'sp500') & (df.columns != 'Daily
return squared')].columns)
ax3.set_title('SP500 price volatility via moving average analysis')
ax3.set_xlabel("Date")
ax3.set_ylabel("Volatility")
ax3.spines['right'].set_visible(False)
ax3.spines['top'].set_visible(False)
ax3.yaxis.set_ticks_position('left')
ax3.xaxis.set_ticks_position('bottom')
```

```
fig.tight_layout()
```

如图1-12所示，简单移动平均使得数据曲线更加平滑，便于从冗杂的数据中整理出较为清晰的脉络，以此可以帮助分析隐含于数据中的真正趋势。随着移动窗口的变大，数据曲线会变得更简单平滑，但是需要注意，数据的细节在这个过程中会丢失，因此需要根据具体情况，合理地选择移动窗口的大小。

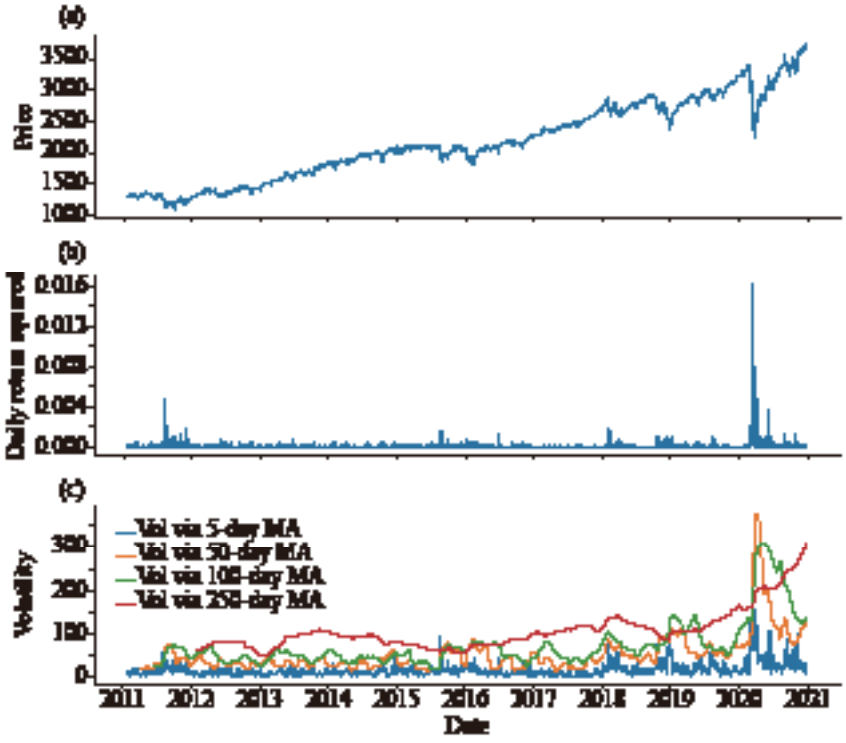



图1-12　简单移动平均计算波动率

与简单移动平均类似，**累积移动平均** (Cumulative Moving Average, CMA) 也是周期性计算一系列数据的平均值，公式为：

$$\mathrm{CMA}_n = \frac{p_1 + p_2 + \cdots + p_n}{n} = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{1-20}$$

但是，不同于简单移动平均，移除旧的数据，加入新的数据，累积移动平均会考虑所有的数据。具体地说，累积移动平均通过有序加入新的数据，并对该数据与原来所有数据进行平均，得到平均值，直到当前数据点为止，如图1-13所示。

同样，类似于简单移动平均，在每次计算中，没有必要计算所有数据的总和，然后除以数据个数，得到平均值。累积移动平均在得到新的数据后，可以简单地更新累积平均值，用式(1-21)来简化计算。

$$\mathrm{CMA}_{n+1} = \frac{p_{n+1} + n \times \mathrm{CMA}_n}{n+1} \tag{1-21}$$

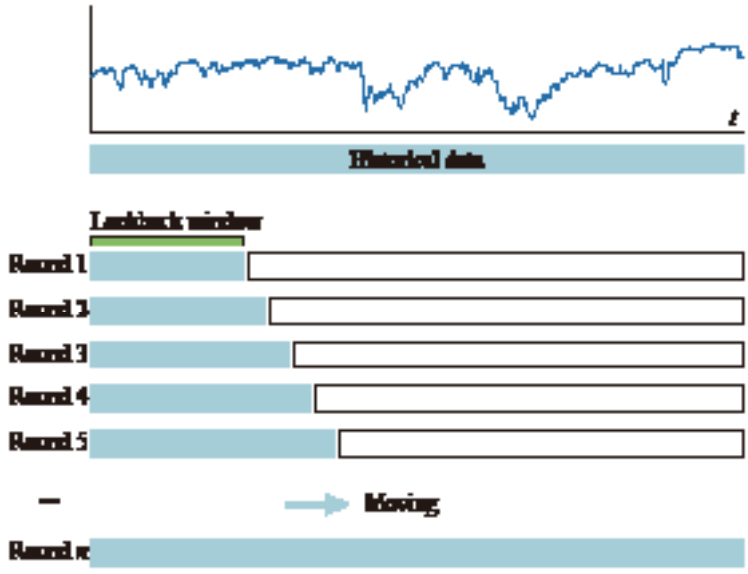


图1-13　累积移动平均方法示意图

前面介绍过的简单移动平均，可以利用rolling() 函数进行计算。对于累积移动平均，Pandas运算包也提供了一个函数——expanding().std() 来计算累积移动标准差，rolling() 函数移动窗口的大小会被预先设定，因此是固定的，而expanding() 函数移动窗口是不断变化的，即每次移动窗口会加1，也正因如此，该函数名被称为“**扩展**”(expanding)。

下面的代码利用获取的标普指数10年的历史数据，通过expanding().std() 函数计算累积波动率，并绘制曲线。

B2_Ch1_5.py

```
import pandas_datareader
import matplotlib.pyplot as plt

#sp500 price
df = pandas_datareader.data.DataReader(['sp500'],
data_source='fred', start='12-28-2010', end='12-28-2020')
df.dropna(inplace=True)

#calculate cumulative moving average
df['cma'] = df['sp500'].expanding(1).std()
df.dropna(inplace=True)
#df.rename(columns={win:'Vol via '+str(win)+' days MA'}, inplace=True)

#plot dataframe
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(df['sp500'])
ax1.set_title('SP500 price')
ax1.set_xlabel("Date")
ax1.set_ylabel("Price")
ax1.spines['right'].set_visible(False)
```

```
ax1.spines['top'].set_visible(False)
ax1.yaxis.set_ticks_position('left')
ax1.xaxis.set_ticks_position('bottom')

ax2.plot(df['cma'])
ax2.set_title('S&P500 price cumulative volatility via moving average analysis')
ax2.set_xlabel("Date")
ax2.set_ylabel("Volatility")
ax2.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.yaxis.set_ticks_position('left')
ax2.xaxis.set_ticks_position('bottom')

fig.tight_layout()
```

累积移动平均是考虑所有数据的叠加，如图1-14所示，它对于累积的数据曲线具有平滑的作用，但是，与简单移动平均不同，它并不能很好地反映整个数据的走势。

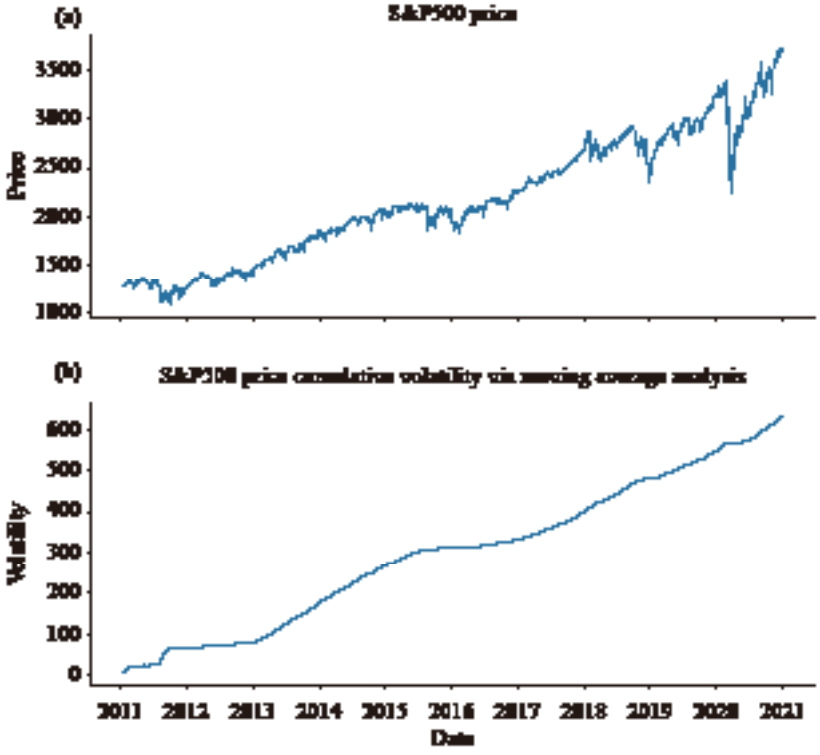



图1-14　累积移动平均波动率

前面介绍的，无论是简单移动平均还是累积移动平均，对于所有数据，无论距离当前时刻的远近，它们的权重是相同的，但是在实际应用中，近期的数据往往对当前数据有相对较大的影响，也更能反映当前以及未来的趋势。基于以上的认知，出现了**加权移动平均法** (weighted moving average)，它是根据数据的时间序列，赋予不同的权重，再依照权重求得移动平均值。如前所述，采用加权移动平均法，近期值对预测值有较大影响，它更能反映近期变化的趋势。如图1-15所示对比了未加权与加权移动平均的不同。

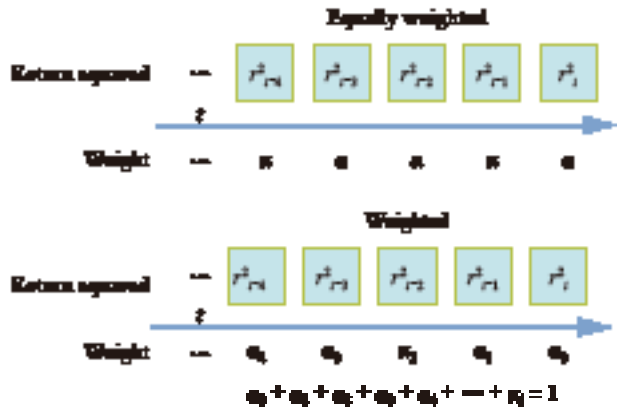


图1-15　未加权与加权移动平均对照图

**指数移动加权平均** (Exponentially Weighted Moving Average, EWMA) 是常用的一种加权平均方法，是指各数值的加权系数随时间呈指数式递减，越靠近当前时刻的数值加权系数就越大。如图1-16所示为指数加权移动平均的权重以指数形式的变化。

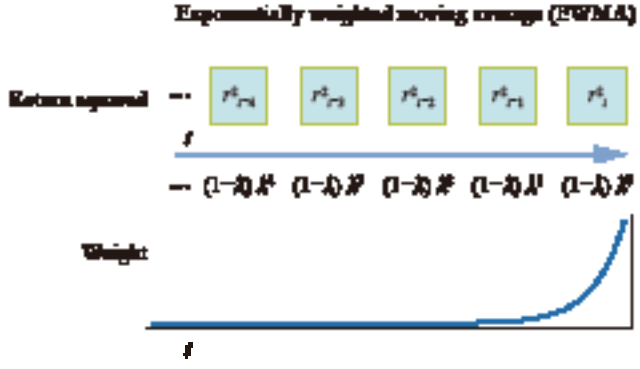


图1-16　指数加权移动平均示意图

指数加权移动平均在波动率上的应用最初是由RiskMetrics于1996年首次提出的。理论上，这种方法需要计算如图1-16所示一系列权重的序列，但是在实际应用上，通常会用到式(1-22)。

$$\sigma_n^2 = \lambda\sigma_{n-1}^2 + (1-\lambda)r_{n-1}^2 \tag{1-22}$$

其中：$\lambda$ 为**衰减因子** (decay factor)；$\sigma_n$ 是当前时刻的波动率；$\sigma_{n-1}$ 是上一时刻的波动率；$r_{n-1}$ 是上一时刻的回报率。

为了方便大家理解，通过下面的例子进行简单推导，讨论衰减因子如何影响波动率计算。

如下列出$n$、$n-1$、$n-2$和$n-3$四个时间点的EWMA波动率计算式为：

$$\begin{cases} \sigma_n^2 = \lambda\sigma_{n-1}^2 + (1-\lambda)r_{n-1}^2 \\ \sigma_{n-1}^2 = \lambda\sigma_{n-2}^2 + (1-\lambda)r_{n-2}^2 \\ \sigma_{n-2}^2 = \lambda\sigma_{n-3}^2 + (1-\lambda)r_{n-3}^2 \\ \sigma_{n-3}^2 = \lambda\sigma_{n-4}^2 + (1-\lambda)r_{n-4}^2 \end{cases} \tag{1-23}$$

对它们依次代入，即将$\sigma_{n-3}$代入$\sigma_{n-2}$，然后将$\sigma_{n-2}$代入$\sigma_{n-1}$，最后$\sigma_{n-1}$代入$\sigma_n$，可以得到式(1-24)。

$$\sigma_n^2=(1-\lambda)\left(r_{n-1}^2+\lambda r_{n-2}^2+\lambda^2 r_{n-3}^2+\lambda^3 r_{n-4}^2\right)+\lambda^4\sigma_{n-4}^2 \tag{1-24}$$

把推导出来的这个等式与图1-17对照，可以看到离当前时刻越远，其权重随指数衰减越厉害。较大的衰减因子，意味着较慢的衰减。以RiskMetrics使用的94%衰减因子为例，前一天的权重为 $(1-0.94)\times 0.94^0=6\%$，之前第二天权重为 $(1-0.94)\times 0.94^1=5.64\%$，之前第三天权重则为 $(1-0.94)\times 0.94^2=5.30\%$。

EWMA波动率迭代公式告诉我们，当前一天的波动率是前一天波动率的函数，这也提供了一种用过去波动率预测未来波动率的方法。这种方法，不需要保存过去所有的数值，而且计算量较小，因此在实际中广泛应用。

下面的例子，首先从Fred数据库中提取了标准普尔指数一年的价格数据，然后利用ewm() 函数计算指数移动平均。ewm()+函数使得EWMA的计算变得非常方便，但是它并没有直接指定衰减因子，而是提供了与平滑系数$\alpha$的转换关系。衰减因子$\lambda$与平滑系数$\alpha$有下面的关系。

$$\lambda=1-\alpha \tag{1-25}$$

其中，alpha为平滑系数$\alpha$，且 $0<\alpha\leqslant 1$。

ewm() 函数衰减参数介绍如下。com为根据质心指定衰减，$\alpha$可以通过式(1-26)计算得到。

$$\alpha=\frac{1}{1+\text{com}},\ \text{com}\geqslant 0 \tag{1-26}$$

span为根据范围指定衰减，$\alpha$可以通过式(1-27)计算得到。

$$\alpha=\frac{2}{1+\text{span}},\ \text{span}\geqslant 1 \tag{1-27}$$

halflife为根据半衰期指定衰减，$\alpha$可以通过式(1-28)计算得到。

$$\alpha=1-\exp\left(\frac{\ln(0.5)}{\text{halflife}}\right),\ \text{halflife}>0 \tag{1-28}$$

下面的代码，通过指定平滑系数为0.01、0.03和0.06，即衰减因子分别为0.99、0.97和0.94，计算得到波动率曲线。感兴趣的读者，可以修改代码，尝试用其他几种方式来指定衰减。

B2_Ch1_6.py

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

#sp500 price
df = pandas_datareader.data.DataReader(['sp500'], data_source='fred',
start='12-28-2010', end='12-28-2020')
df.dropna(inplace=True)

#daily log return
df['Daily return squared'] = np.log(df['sp500'] / df['sp500'].shift(1))*np.
log(df['sp500'] / df['sp500'].shift(1))
```

```
df.dropna(inplace=True)

#calculate exponentially weighted moving average
alpha_list = [0.01, 0.03, 0.06]
for alpha in alpha_list:
    ma = df['sp500'].ewm(alpha=alpha, adjust=False).std()
    df[alpha] = ma
    df.rename(columns={alpha:'$\lambda$ = '+str(1-alpha)}, inplace=True)

#plot dataframe
#sp500 price
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 12))
ax1.plot(df['sp500'])
ax1.set_title('SP500 price')
ax1.set_xlabel("Date")
ax1.set_ylabel("Price")
ax1.spines['right'].set_visible(False)
ax1.spines['top'].set_visible(False)
ax1.yaxis.set_ticks_position('left')
ax1.xaxis.set_ticks_position('bottom')

#daily log return squared
ax2.plot(df['Daily return squared'])
ax2.set_title('Daily return squared')
ax2.set_xlabel("Date")
ax2.set_ylabel("Daily return squared")
ax2.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.yaxis.set_ticks_position('left')
ax2.xaxis.set_ticks_position('bottom')
#ewma vol
ax3.plot(df.loc[:, (df.columns != 'sp500') & (df.columns != 'Daily return
squared')])
ax3.legend(df.loc[:, (df.columns != 'sp500') & (df.columns != 'Daily return
squared')].columns)
ax3.set_title('SP500 price volatility via EWMA analysis')
ax3.set_xlabel("Date")
ax3.set_ylabel("Volatility")
ax3.spines['right'].set_visible(False)
ax3.spines['top'].set_visible(False)
ax3.yaxis.set_ticks_position('left')
ax3.xaxis.set_ticks_position('bottom')

fig.tight_layout()
```

代码运行后，生成图1-17，可见，衰减系数越小，估算的波动率峰值越高，而且波动也会越显著。

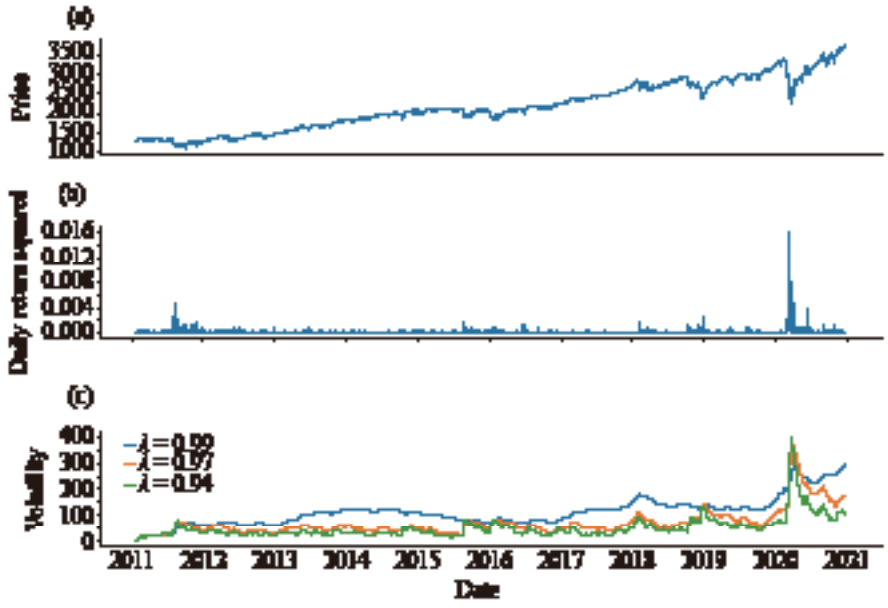
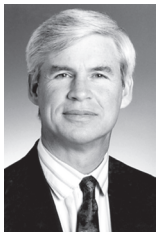



图1-17　指数权重移动平均计算波动率

# 1.4 自回归条件异方差模型ARCH

传统的计量经济学假定时间序列变量的方差 (波动率) 是固定不变的，然而，这与实际情况是不相符的。比如，股票收益的波动率就是随着时间而变化的。因此，传统的计量经济学在分析许多实际问题时，陷入了困境。

1982年美国统计学家**罗伯特·弗莱·恩格尔三世** (Robert Fry Engle Ⅲ) 在研究英国通货膨胀率的波动性问题时，提出了**自回归条件异方差模型** (Autoregressive Conditional Heteroscedasticity model, ARCH)，即**ARCH模型**。在这里，**异方差** (heteroscedasticity) 是指一系列的随机变量值的方差不同。这个模型以自回归方式，通过刻画随时间变异的条件方差，成功解决了时间序列的波动性问题。正是因为在ARCH模型上的杰出贡献，罗伯特·弗莱·恩格尔三世在2003年获得了诺贝尔经济学奖。



Robert F. Engle Ⅲ (1942–) Developed methods to study the volatility properties of time series in economics, particular in financial markets. His method (ARCH) could, in particular, clarify market developments where turbulent periods, with large fluctuations, are followed by calmer periods, with modest fluctuations. (Sources: https://www.nobelprize.org/prizes/economic-sciences/2003/engle/facts/)

观察波动率曲线，可以看到波动率变化大往往会持续一段时间，这就是**波动率聚集** (volatility clustering) 现象，传统计量经济学的模型中，干扰项的方差均被假定为常数，这对波动率来说，显然

是不适合的。ARCH模型则将波动率定义为条件标准差，收益率序列是前后不相关的，但是前后也并不独立，而是用一系列滞后值的线性组合来表示。

为了简便，对于ARCH模型的具体讲解，在这里只考虑波动率项，即令 $\mathrm{E}(Z_t)=0$。在实际处理收益率的历史数据时，可以通过归一化移除其期望值，来达到处理后的数据期望值为零的效果。

假设收益率 $\{X_t\}$ 服从期望为零的独立同分布随机过程，在时刻$t$的收益率可表达为：

$$X_t=\sigma_t Z_t \tag{1-29}$$

其中，随机变量$Z_t$可以服从标准正态分布，也可以服从$t$-分布。$\sigma_t^2$ 则可以表示为：

$$\sigma_t^2=\omega+\sum_{i=1}^{L_1}\alpha_i X_{t-i}^2 \tag{1-30}$$

其中，$\omega>0$，$\alpha_i\geqslant 0$，$L_1>0$，即各期收益以非负数线性组合，常数项为正数，这是为了保证波动率为正值。为了保证**协方差的平稳性** (covariance stationarity)，还需要满足 $\sum_{i=1}^{L_1}\alpha_i<1$。$L_1$ 是模型中含有的滞后序列的个数。这就是ARCH模型的基本表示式。在这个表示式的等号右侧的滞后序列均没有新增的随机项，为确定函数，所以ARCH模型属于确定性的波动率模型。

另外，分析上面模型的结构，如果存在较大的随机变化，将导致条件异方差变大，因此有取绝对值较大的值的趋势。反映在ARCH模型上，即大的随机变化出现后，紧接着会有倾向继续出现另一个大的随机变化，这与波动率聚集现象非常相似。

当 $L_1=1$ 时，便是经常用到的ARCH(1) 模型，其表示式为：

$$\sigma_t^2=\omega+\alpha X_{t-1}^2 \tag{1-31}$$

即时刻$t$的波动率平方 $\sigma_t^2$ 等于一个常数 $\omega$ 加上时刻$t-1$的收益率 $X_{t-1}$ 的平方。可见，此时的波动率依赖于已知观测值，所以是一个**条件波动率** (conditional volatility)。

下面考察，收益率$X$的$m$阶**矩** (moment)，它与其在时间序列上的观测值 $\{X_t\}$ 有如下关系：

$$\mathrm{E}(X^m)=\mathrm{E}\left(\mathrm{E}_t(X^m)\right)=\mathrm{E}(X_t^m) \tag{1-32}$$

当 $m=2$ 时：

$$\mathrm{E}(X^2)=\sigma^2=\mathrm{E}(X_t^2)=\mathrm{E}(\sigma_t^2 Z_t^2)=\mathrm{E}(\sigma_t^2) \tag{1-33}$$

将其代入ARCH(1) 模型中可以得到：

$$\sigma^2=\mathrm{E}(\omega+\alpha X_{t-1}^2)=\omega+\alpha\sigma^2 \tag{1-34}$$

因此，式(1-35)成立。

$$\sigma^2=\frac{\omega}{1-\alpha} \tag{1-35}$$

注意，与ARCH(1) 模型自身的表达式不同的是，此处 $\sigma^2$ 的解与具体时刻无关，也不直接依赖之前的收益率观测值，所以它被称为**无条件波动率** (unconditional volatility)。

下面的代码获取了从2009年12月28 日到2020年12月28日标准普尔指数11年的历史价格数据，并利用Arch运算包的arch_model() 函数进行拟合。在参数设定时，vol设定为 'ARCH'，p设定为1，即要使用的拟合模型为ARCH(1)。在完成拟合之后，打印出这个模型的所有汇总信息，并利用内建的函数对标准残差和条件波动率直接进行了可视化操作，如图1-18所示。

B2_Ch1_7_A.py

```
import numpy as np
import pandas_datareader
import matplotlib.pyplot as plt
from arch import arch_model

#sp500 price
sp500 = pandas_datareader.data.DataReader(['sp500'],
data_source='fred', start='12-28-2009', end='12-28-2020')

#daily log return
log_return_daily = np.log(sp500 / sp500.shift(1))
log_return_daily.dropna(inplace=True)

#ARCH(1) model
arch=arch_model(y=log_return_daily,mean='Constant',
lags=0,vol='ARCH',p=1,o=0,q=0,dist='normal')
archmodel=arch.fit()
archmodel.summary()
archmodel.plot()
```

拟合ARCH模型所有信息汇总。

```
"""
                     Constant Mean - ARCH Model Results
==============================================================================
Dep. Variable:                    s&p500   R-squared:                      -0.001
Mean Model:             Constant Mean   Adj. R-squared:                 -0.001
Vol Model:                        ARCH   Log-Likelihood:               7797.43
Distribution:                   Normal   AIC:                          -15588.9
Method:            Maximum Likelihood   BIC:                          -15571.5
                                        No. Observations:                 2417
Date:                Fri, Jan 08 2021   Df Residuals:                     2414
Time:                        13:58:58   Df Model:                            3
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
```

```
mu          7.6186e-04   1.934e-04       3.939   8.184e-05 [3.828e-04,1.141e-03]
                              Volatility Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega       6.7599e-05   5.039e-06     13.415   4.920e-41 [5.772e-05,7.748e-05]
alpha[1]        0.4500   8.258e-02      5.449   5.057e-08     [  0.288,  0.612]
==============================================================================

Covariance estimator: robust
"""
```

从图1-18可以看出，标准化残差近似为一个平稳序列，这也说明该模型具有较好的表现力。

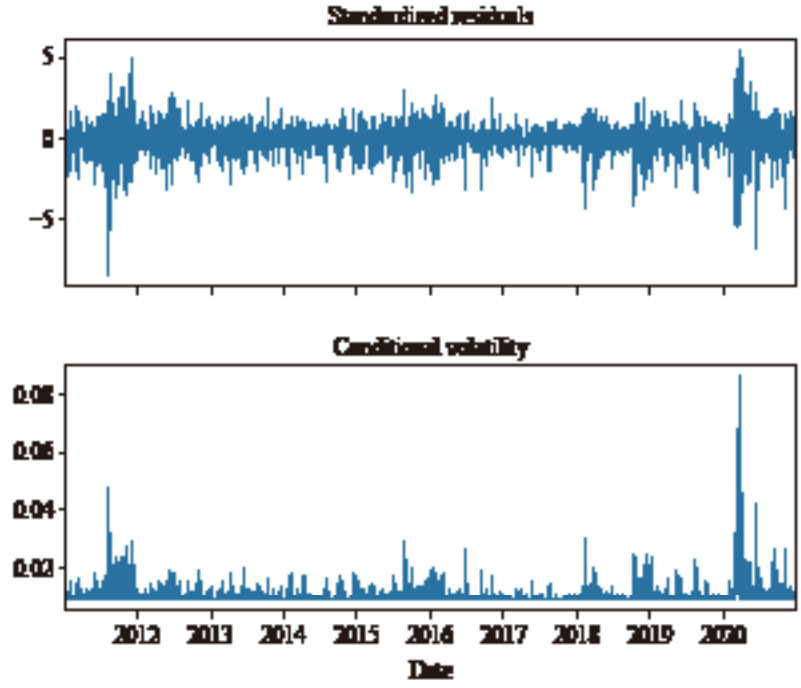



图1-18　ARCH(1)模型标准残差和条件波动率

另外，利用下面代码，可以把日回报率和条件波动率用图形展示出来。

B2_Ch1_7_B.py

```
plt.figure(figsize=(12,8))
plt.plot(log_return_daily,label='Daily return')
plt.plot(archmodel.conditional_volatility, label='Conditional volatility')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Return/Volatility')
```

如图1-19中的蓝色线代表日回报率的波动，橘色线代表条件波动率即条件异方差，由图可见，条件异方差曲线很好地反映了日回报率的变化趋势。

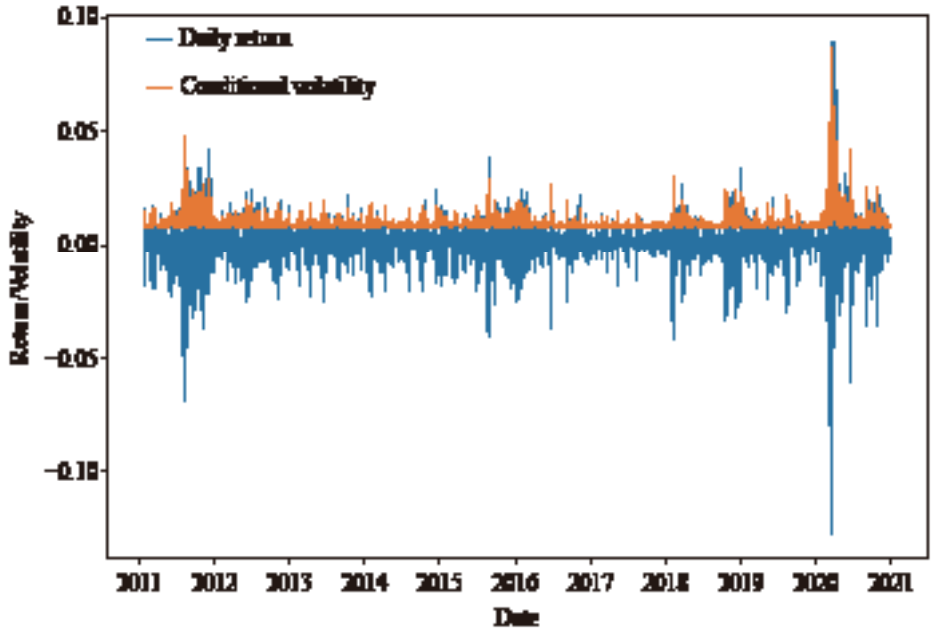



图1-19　日回报率和ARCH(1)模型条件波动率

通过查前面输出的汇总表，或者简单地输入下面的命令，可以得到ARCH(1) 模型的参数。

```
archmodel.params
```

模型参数输出如下。

```
mu          0.000762
omega       0.000068
alpha[1]    0.450000
Name: params, dtype: float64
```

因此，上面例子最终拟合得到的ARCH(1) 模型为：

$$\sigma_t^2 = 0.000068 + 0.45X_{t-1}^2 \tag{1-36}$$

# 1.5 广义自回归条件异方差模型GARCH

ARCH模型形式非常简单，也可以很好地描述波动率，但是为了保证条件方差为正值，往往需要引入很多滞后值，建立高阶模型，这就需要很多的参数。为了解决这个问题，丹麦经济学家**提姆·波勒斯勒夫** (Tim Bollerslev) 在ARCH模型基础上，通过引用条件方差滞后值，在1986年提出了**广义自回归条件异方差模型** (generalized ARCH model, GARCH)，即**GARCH模型**。GARCH模型是对ARCH波动率建模的一种重要推广，迅速在金融领域得到了巨大的成功。在其提出之后，又有诸如NGARCH、IGARCH、EGARCH等一系列针对不同应用的衍生模型相继出现。

Tim Bollerslev (1958–) is a Danish economist, currently the Juanita and Clifton Kreps Professor of Economics at Duke University. Professor Bollerslev conducts research in the areas of time-series econometrics, financial econometrics, and empirical asset pricing finance. He is particularly well known for his developments of econometric models and procedures for analyzing and forecasting financial market volatility. (Sources: https://scholars.duke.edu/person/tim.bollerslev)

GARCH模型的表达式如下：

$$\sigma_t^2 = \omega + \sum_{i=1}^{L_1} \alpha_i X_{t-i}^2 + \sum_{j=1}^{L_2} \beta_j \sigma_{t-j}^2 \tag{1-37}$$

可见，GARCH模型在形式上与ARCH模型相似，本质上它是在ARCH模型的基础上，引入了滞后的波动率平方项 $\sigma_{t-j}^2$。另外，表达式中的 $L_1$ 和 $L_2$ 是模型中含有的滞后序列的个数，分别对应之前时刻的收益率$X$项和波动率$\sigma$项。在表达式中，需要满足$\omega > 0$， $\alpha_i \geqslant 0$， $\beta_j \geqslant 0$，$L_1 > 0$以及$L_2 > 0$，这可以确保得到正的波动率。另外，为了保证模型的无条件方差有限且不变，并且条件方差可以随时间变化，参数$\alpha_i$和$\beta_i$还需要满足式(1-38)。

$$0 < \sum_{i=1}^{L_1} \alpha_i + \sum_{j=1}^{L_2} \beta_j < 1 \tag{1-38}$$

对于GARCH模型，当 $L_1 = 1,\ L_2 = 1$ 时，便是其中形式最简单的GARCH(1,1) 模型，即时刻$t$的波动率平方 $\sigma_t^2$ 等于一个大于0的常数 $\omega$，加上时刻$t-1$的收益率 $X_{t-1}$ 的平方项，再加上时刻$t-1$的波动率 $\sigma_{t-1}$ 的平方，即：

$$\sigma_t^2 = \omega + \alpha X_{t-1}^2 + \beta \sigma_{t-1}^2 \tag{1-39}$$

式(1-39)中的 $\sigma_{t-1}^2$ 项，在ARCH模型中没有，它的引入使得历史波动率的影响能在模型中体现，从而弥补了ARCH模型的不足。

下面还是以最简单的GARCH(1,1)模型为例研究GARCH模型的性质。GARCH(1,1) 模型的无条件波动率可以做如下的变换与推导。

$$\sigma^2 = \mathrm{E}\left(\omega + \alpha_1 X_{t-1}^2 + \beta_1 \sigma_{t-1}^2\right) = \omega + \alpha\sigma^2 + \beta\sigma^2 \tag{1-40}$$

从而得到：

$$\sigma^2 = \frac{\omega}{1-\alpha-\beta} \tag{1-41}$$

无论从ARCH模型和GARCH模型的表达式还是实际意义来看，都可以认为ARCH模型是GARCH模型的特殊形式，即ARCH模型是GARCH模型中波动率平方项 $\sigma_{t-j}^2$ 前系数 $\beta_j = 0$ 时的情况。因此，也与ARCH模型一样，GARCH模型可以很好地反映波动率聚集现象，在条件方差变大的情况下，后面会倾向于出现较大的对数收益率。另外，相对于ARCH模型高阶的情况，GARCH往往会给出一个更加简洁有效的波动率模型。

下面的代码使用了与1.4节完全相同的标准普尔的数据，计算对数回报率。接着，同样地利用arch_model() 函数进行拟合，但是在设定这个函数的参数时，需要设定参数vol为'GARCH'，p为1，q为1，即这个拟合模型为GARCH(1,1)。通过拟合，最终打印输出GARCH模型的拟合结果汇总，以及绘制标准残差和条件波动率的图形，如图1-20所示。

B2_Ch1_7_C.py

```
#GARCH(1,1) model
garch=arch_model(y=log_return_daily,mean='Constant',lags=0,vol='GARCH',
p=1,o=0,q=1,dist='normal')
garchmodel=garch.fit()
garchmodel.summary()
garchmodel.plot()
```

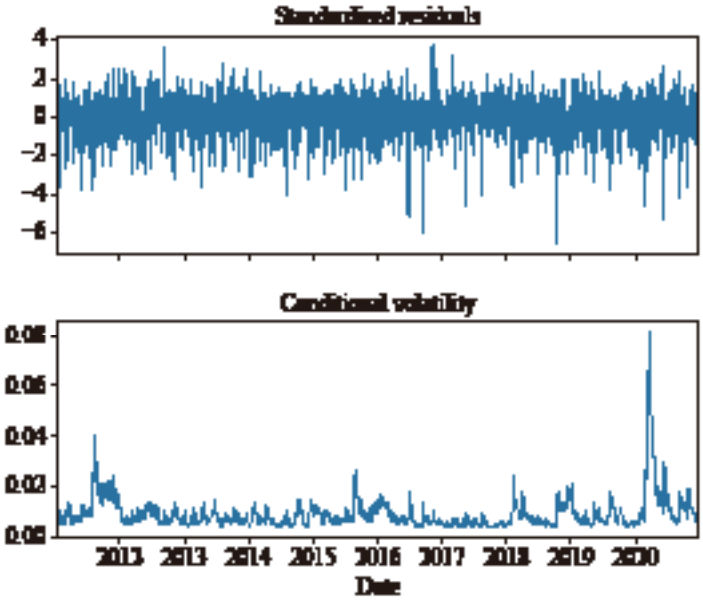



图1-20　GARCH(1,1)模型标准残差和条件波动率

GARCH模型结果汇总如下。

```
"""
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                   sp500   R-squared:                      -0.001
Mean Model:             Constant Mean   Adj. R-squared:                 -0.001
Vol Model:                      GARCH   Log-Likelihood:                8152.99
Distribution:                  Normal   AIC:                           -16298.0
Method:            Maximum Likelihood   BIC:                           -16274.8
                                        No. Observations:                 2417
Date:                Fri, Jan 08 2021   Df Residuals:                     2413
Time:                        14:24:52   Df Model:                            4
                                 Mean Model
==============================================================================
```

```
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
mu          8.1981e-04  3.342e-06    245.339      0.000 [8.133e-04,8.264e-04]
                            Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega       2.4582e-06  1.055e-11  2.331e+05      0.000 [2.458e-06,2.458e-06]
alpha[1]        0.2000  2.885e-02      6.933  4.121e-12     [  0.143,  0.257]
beta[1]         0.7800  2.247e-02     34.707 6.180e-264     [  0.736,  0.824]
============================================================================

Covariance estimator: robust
"""
```

观察图1-20，标准化残差近似为一个平稳序列，这从一个角度说明GARCH(1, 1) 模型对于分析该问题是适合的。

另外，利用下面代码，也可以把日回报率和条件波动率用图形展示出来。

B2_Ch1_7_D.py

```
plt.figure(figsize=(12,8))
plt.plot(log_return_daily,label='Daily return')
plt.plot(archmodel.conditional_volatility, label='Conditional volatility')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Return/Volatility')
```

图1-21中反映出本例子中所用的GARCH(1,1)模型，条件异方差非常好地表现了波动率的变化。

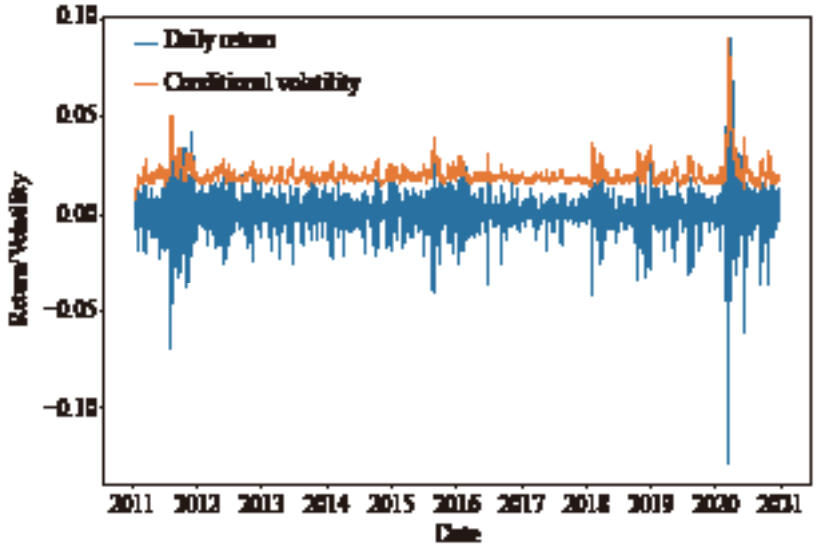



图1-21　日回报率和GARCH(1)模型条件波动率

同样地，GARCH(1,1) 模型参数也可以通过查阅汇总表格或者运行下面命令而获得。

```
garchmodel.params
```

拟合得到的模型参数输出如下。

```
mu            0.000820
omega         0.000002
alpha[1]      0.200000
beta[1]       0.780000
Name: params, dtype: float64
```

因此，该GARCH(1,1) 模型的表达式为：

$$\sigma_t^2 = 0.000002 + 0.2X_{t-1}^2 + 0.78\sigma_{t-1}^2 \tag{1-42}$$

# 1.6 波动率估计

本章前面几节分别介绍了几种分析波动率的方法和模型。表1-1展示了EWMA、ARCH和GARCH三种方法的公式对比，可以看到三种方法的内在联系。如前面提到过的，当GARCH(1, 1) 模型的参数$\beta$为零时，即为ARCH(1) 模型；而当 $\omega=0,\alpha=1-\lambda,\beta=\lambda$ 时，GARCH(1, 1)模型变换成为EWMA模型。

表1-1　EWMA、ARCH与GARCH数学表达式对比

| 模型 | 表达式 |
|---|---|
| EWMA | $\sigma_n^2 = \lambda\sigma_{n-1}^2 + (1-\lambda)r_{n-1}^2$ |
| ARCH | $\sigma_t^2 = \omega + \sum_{i=1}^{L_1}\alpha_i X_{t-i}^2$ |
| ARCH (1) | $\sigma_t^2 = \omega + \alpha X_{t-1}^2$ |
| GARCH | $\sigma_t^2 = \omega + \sum_{i=1}^{L_1}\alpha_i X_{t-i}^2 + \sum_{j=1}^{L_2}\beta_j\sigma_{t-j}^2$ |
| GARCH (1,1) | $\sigma_t^2 = \omega + \alpha X_{t-1}^2 + \beta\sigma_{t-1}^2$ |

对于这几个模型的介绍，前面主要侧重于分析具体的数据并建立模型。而借助这些模型，也可以对波动率进行预测。下面的例子，对于EWMA模型，利用了通常用的0.94，即JP摩根RiskMetrics采用的设定。而另外的ARCH(1) 模型和GARCH(1, 1) 模型则采用了前面拟合得到的参数。三个模型具体如下所示。

$$\begin{cases} \sigma_n^2 = 0.94\sigma_{n-1}^2 + (1-0.94)r_{n-1}^2 \\ \sigma_t^2 = 0.000068 + 0.45X_{t-1}^2 \\ \sigma_t^2 = 0.000002 + 0.2X_{t-1}^2 + 0.78\sigma_{t-1}^2 \end{cases} \tag{1-43}$$

下面代码继续使用标准普尔的历史数据，分别利用上面三个模型估算了2009年12月28日到2020年12月28日的波动率。

B2_Ch1_8.py

```
import numpy as np
import pandas_datareader
import matplotlib.pyplot as plt
import datetime
import matplotlib.dates as mdates


#sp500 price
sp500 = pandas_datareader.data.DataReader(['sp500'],
data_source='fred', start='12-28-2009', end='12-28-2020')

#daily log return
log_return_daily = np.log(sp500 / sp500.shift(1))
log_return_daily.dropna(inplace=True)

n = 250
r = log_return_daily.iloc[-n:]

#volatility prediction by EWMA with λ=0.94
lmd = 0.94
vol_ewma = np.zeros(n)
vol_ewma[0] = log_return_daily[(-n+1):(-n+6)].std()
for i in range(n-1):
    vol_ewma[i+1] = np.sqrt(lmd*vol_ewma[i]**2 + (1-lmd)*r.iloc[i]**2)

#volatility prediction by ARCH(1)
omega_arch = 0.000068
alpha1 = 0.45
vol_arch = np.zeros(n)
vol_arch[0] = np.sqrt(omega_arch + alpha1*log_return_daily.iloc[-n-1]**2)
for i in range(n-1):
    vol_arch[i+1] = np.sqrt(omega_arch + alpha1*r.iloc[i]**2)

#GARCH(1,1)
omega = 0.000002
alpha1 = 0.2
beta1 = 0.78

vol_garch = np.zeros(n)
vol_garch[0] = log_return_daily[-n+1:-n+6].std()
for i in range(n-1):
    vol_garch[i+1] = np.sqrt(omega + alpha1*r.iloc[i]**2 + beta1*vol_garch[i]**2)
```

```
#plot the curves
xdate=(r.index+datetime.timedelta(days=1))
plt.figure(figsize=(12,8))
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.title('Volatility comparison')
plt.plot(xdate,vol_arch, label='ARCH(1)')
plt.plot(xdate,vol_garch, label='GARCH(1,1)')
plt.plot(xdate,vol_ewma, label='EWMA')

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d/%Y'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator(15))
plt.xticks(rotation=30)
plt.legend()
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

如图1-22所示绘制了这三种方法估算的波动率，可以发现它们的预测结果总体趋势是大致相似的。大家可以尝试改变预测天数，以及调整参数深化对这三种模型的理解。在实际应用中，需要根据具体的情况，灵活选用适合的模型。

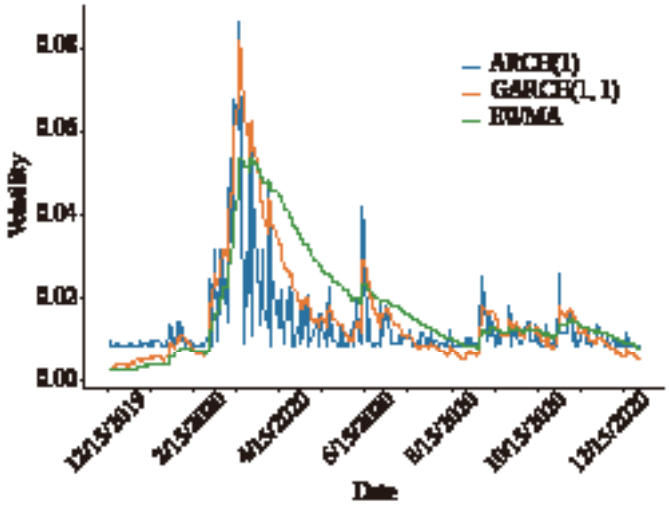



图1-22　EWMA、ARCH(1)和GARCH(1, 1)模型估算波动率

# 1.7 隐含波动率

通过对本书前面关于Black-Scholes模型的章节的学习，我们利用五个基本参数，即**标的物价格** (underlying price)、**到期时间** (time to maturity)、**无风险利率** (risk-free interest rate)、**期权执行价格**

(strike price)、**资产价格回报波动率** (volatility)，可以对期权进行定价，如图1-23所示。

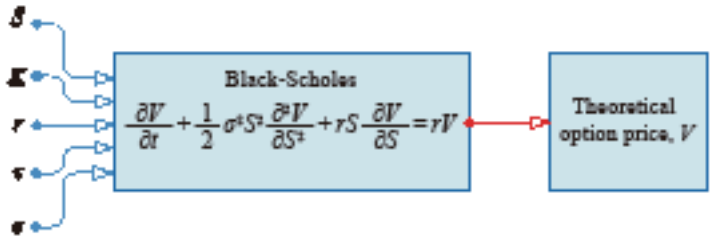



图1-23　Black-Scholes模型期权定价示意图

相应地，如图1-24所示，如果将从市场获取的期权交易价格代入Black-Scholes模型，利用其他参数，可以反推出波动率的数值，这个数值被称为**隐含波动率** (implied volatility)。隐含波动率反映了投资者所投资的资产未来一段时间内波动的预期。通常来说，隐含波动率与历史波动率会有一些差别，但是一般会比较接近。

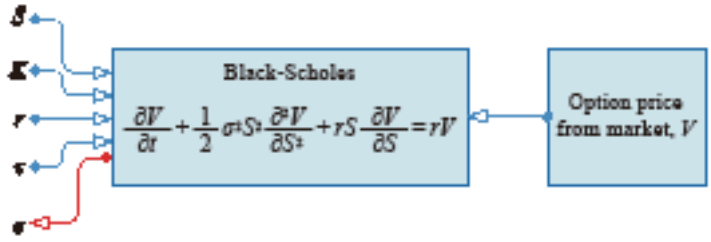



图1-24　Black-Scholes模型期权定价模型推导隐含波动率

下面的代码，首先定义了一个基于Black-Scholes模型的计算期权价格的函数option_price_BS()，然后利用这个函数构建了一个借助**二分法** (bisection method) 估算隐含波动率的函数implied_vol()。最后，用这个函数计算了一个期权类型为看涨期权，标的物价格为586.08，期权执行价格为585，距离到期时间为30天，无风险利率为0.0002的期权的隐含波动率。

B2_Ch1_9.py

```
import numpy as np
from scipy import stats

#function to calculate price of options (call or put) by BS
def option_price_BS(option_type, sigma, s, k, r, T, q=0.0):
    d1 = (np.log(s / k) + (r - q + sigma ** 2 * 0.5) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    if option_type == 'call':
        option_price = np.exp(-r*T) * (s * np.exp((r - q)*T) *
stats.norm.cdf(d1) - k *  stats.norm.cdf(d2))
        return option_price
    elif option_type == 'put':
        option_price = np.exp(-r*T) * (k * stats.norm.cdf(-d2) - s *
np.exp((r - q)*T) *  stats.norm.cdf(-d1))
        return option_price
    else:
        print('Option type should be call or put.')
```

```
#funciton to calculate implied volatility by bisection method
def implied_vol(option_type, option_price, s, k, r, T, q):
    precision = 0.00001
    upper_vol = 500.0
    lower_vol = 0.0001
    iteration = 0

    while iteration >= 0:

        iteration +=1
        mid_vol = (upper_vol + lower_vol)/2.0
        price = option_price_BS(option_type, mid_vol, s, k, r, T, q)

        if option_type == 'call':
            lower_price = option_price_BS(option_type, lower_vol, s, k, r, T, q)
            if (lower_price - option_price) * (price - option_price) > 0:
                lower_vol = mid_vol
            else:
                upper_vol = mid_vol
            if abs(price - option_price) < precision:
                break

        elif option_type == 'put':
            upper_price = option_price_BS(option_type, upper_vol, s, k, r, T, q)

            if (upper_price - option_price) * (price - option_price) > 0:
                upper_vol = mid_vol
            else:
                lower_vol = mid_vol
            if abs(price - option_price) < precision:
                break

        if iteration == 100:
            break
    print('Implied volatility: %.2f' % mid_vol)
    return mid_vol

implied_vol('call', 17.5, 586.08, 585, 0.0002, 30.0/365, 0.0)
```

隐含波动率计算结果如下。

```
Implied volatility: 0.25
```

对于具有相同到期时间和标的资产价值的欧式期权，隐含波动率会随着执行价格的变化而变化。如果对一系列该类期权的隐含波动率的执行价格绘制曲线图，往往会得到一个凹线，即**虚值期权** (out of money) 和**实值期权** (in the money) 的波动率高于**平值期权** (at the money) 的波动率，使得波动率曲线看起来像是一个人在微笑，这就是著名的**波动率微笑** (volatility smile)，如图1-25所示。从图中可以看

出，具有相同到期时间和标的资产价值而执行价格不同的期权，其执行价格偏离标的资产现货价格越远，其实值程度或虚值程度越大，通过Black-Scholes公式计算得到的隐含波动率也会越大。

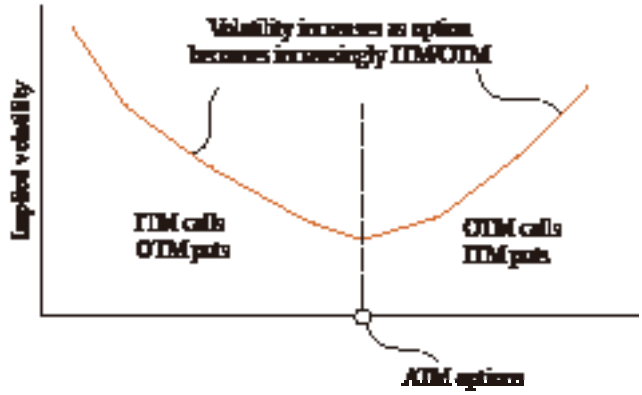



图1-25　波动率微笑

波动率微笑的形状取决于标的资产和市场的状况，而波动率并不总是微笑的，如图1-26所示，其曲线有可能是偏斜的，被称为**波动率偏斜** (volatility skew)。波动率偏斜通常指的是低行权价的隐含波动率高于高行权价隐含波动率的波动率曲线。但是，有时也泛指各种偏斜形状的波动率曲线。

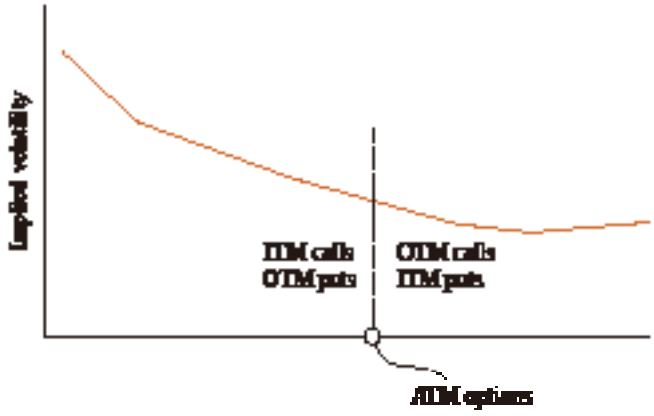



图1-26　波动率偏斜

在外汇期权市场，更多地会看到“波动率微笑”，而在股票期权市场，则常常会看到“波动率偏斜”。

波动率微笑以及波动率偏斜是金融领域的一个热点研究课题。对于其出现的原因从不同角度有一些探讨和解释。

首先，从隐含波动率的来源，即推导隐含波动率的Black-Scholes模型来看，它的一个前提假设是收益率服从正态分布，但是实际的金融资产的收益率一般会有厚尾现象，也就是收益率出现极端值的概率高于正态分布。因此，使用Black-Scholes模型计算的隐含波动率会低估到期时期权价值变为实值与虚值的概率，亦即低估了深度实值和深度虚值期权的价格。另外，Black-Scholes模型采用的是**风险中性** (risk neutral) 定价，并假设资产价格服从带漂移项的维纳过程，而现实市场的资产价格在很多情况下会有发生跳跃的可能，这会导致期权的市场价格与理论价格发生偏离。

其次，从期权的市场交易机制来看。理论上，期权从平值状态变为实值状态和虚值状态的概率应该基本相同，并且在平值状态时其时间价值最大。但是，深度实值期权的Delta接近1，在投资中的杠杆作用最大，相应市场需求量很大，然而，深度实值期权的市场供应量却很小，因此，供需的不平衡会导致其溢价较高。相应地，这也推高了隐含波动率。在市场交易中，对于市场巨幅动荡的担忧，

会导致交易员对深度虚值看跌期权赋予较大的价值，从而造成较高的波动率。另外，交易成本的不对称，买卖价差的不对称，尤其对于深度实值和深度虚值期权，买卖差价会更明显。这些原因都可能导致波动率微笑或者波动率偏斜的产生。

为了帮助大家更加感性地理解波动率微笑，下面代码绘制了基于真实市场数据的波动率微笑曲线。首先，从芝加哥期货交易所数据库 (http://www.cboe.com/delayedquote/quote-table-download) 获取2021年1月15日标普指数头寸期权的数据，存入csv文件，然后根据**叫价**(ask)和**成交价**(bid) 的平均计算期权的价格。

B2_Ch1_10_A.py

```
from mibian import BS
import pandas as pd
import matplotlib.pyplot as plt

#convert data to dataframe and initialize "Implied volatility" column
option_data = pd.read_csv(r"C:\Users\Ran\Dropbox\FRM Book\Volatility\SPX_Option.
csv")
option_data['date'] = pd.to_datetime(option_data['date'])
option_data['Implied volatility'] = 0
option_data.head
```

展示前五行，预览数据。

```
<bound method NDFrame.head of             date Expiration Date   ...   days to
maturity  Implied volatility
0   2021-01-15      2022-01-21  ...              266                   0
1   2021-01-15      2022-01-21  ...              266                   0
2   2021-01-15      2022-01-21  ...              266                   0
3   2021-01-15      2022-01-21  ...              266                   0
4   2021-01-15      2022-01-21  ...              266                   0
..         ...             ...  ...              ...                 ...
74  2021-01-15      2022-01-21  ...              266                   0
75  2021-01-15      2022-01-21  ...              266                   0
76  2021-01-15      2022-01-21  ...              266                   0
77  2021-01-15      2022-01-21  ...              266                   0
78  2021-01-15      2022-01-21  ...              266                   0

[79 rows x 16 columns]>
```

接着，定义计算隐含波动率的函数compute_implied_volatility()，并应用于每一列。当然，也可以通过读入每一行，进行计算，但是计算速度会很慢，感兴趣的读者可以自行尝试，并对比两种方法的运算速度。

B2_Ch1_10_B.py

```
#function to calculate implied volatility
def compute_implied_volatility(row):
    underlyingPrice = row['underlying value']
    strikePrice = row['strike']
```

```
    interestRate = 0.002
    daysToMaturity = row['days to maturity']
    optionPrice = row['call price']
    result = BS([underlyingPrice, strikePrice, interestRate,
daysToMaturity], callPrice= optionPrice)
    return result.impliedVolatility

option_data['Implied volatility'] =
option_data.apply(compute_implied_volatility, axis=1)
```

通过绘制隐含波动率相对于行权价格的曲线，可以得到波动率微笑，如图1-27所示。

B2_Ch1_10_C.py

```
#plot volatility smile
option_data = option_data[option_data['date'] == pd.to_datetime('1/15/2021')]
plt.plot(option_data['strike'], option_data['Implied volatility'])
plt.title('Volatility smile')
plt.ylabel('Implied volatility')
plt.xlabel('Strike price')

plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.gca().yaxis.set_ticks_position('left')
plt.gca().xaxis.set_ticks_position('bottom')
```

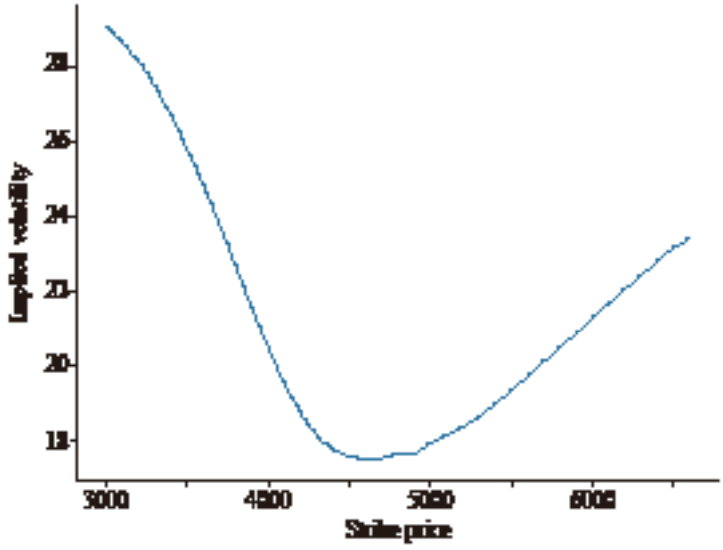



图1-27　隐含波动率微笑曲线

图1-27展示了一个典型的波动率微笑。感兴趣的读者可以从前面介绍的网站，下载更多期权价格数据，修改代码，自行绘制更多波动率曲线。

本章从基本的回报率谈起，紧接着介绍了历史波动率，以及分析历史波动率的几种方法，大家可以借助图1-28回顾介绍过的EWMA、ARCH、GARCH等模型，更加系统地加深对这些重要且基本的

方法的理解，这些方法不但应用于本章的波动率，对**风险价值** (Value at Risk) 等金融度量也有着重要的应用。

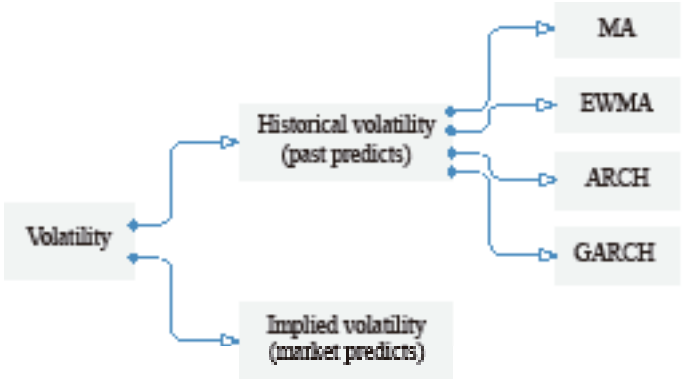



图1-28　波动率概览

并且用可视化的方法比较了上述几种模型。最后，介绍了隐含波动率，并分析了波动率微笑产生的原因。时至今日，对于波动率曲面的研究仍然是热点领域。另外，除了本章介绍的几种波动率模型，随机波动率模型、局部波动率模型等也是实际工作中常常用到的波动率模型，有兴趣的读者可以翻阅相关资料。“大海浩瀚，逐波而动”，希望大家通过本章的学习，立足于波动率的基本概念，更加深切体会“波动”对于“金融海洋”的重要意义。