第10章 隐含波动率

内容

- 隐含波动率
- Python循环语句
- 代码

隐含波动率

隐含波动率含义

- 在B-S-M模型期权定价中，给定一组参数S(当前股价)、X（执行价格）、T(有效期)、r(连续复利无风险利率)和σ（股票收益率的标准方差，也称波动率），可以计算欧式看涨期权的定价c。
- 反过来，如果知道S、X、T、r和期权价格c，求解得出的σ值就是隐含波动率。
- 可行的计算方法：试错方法+收敛条件

标准求解

- 给定期权报价的隐含波动率 $C(S_t, X, T, r, \sigma^{imp}) = C^*$
- 对这个方程没有闭合解，可使用牛顿迭代法等数值求根过程估算正确的解。用相关函数的起始值进行迭代，直到达到某种精度。对于起始值σ_0^{imp}和0<n<∞，有：
- 方程数值化求根的牛顿迭代法

$$\sigma_{\mathrm{n+1}}^{\mathrm{imp}} = \sigma_n^{imp} - \frac{C\left(\sigma_n^{\mathrm{imp}}\right) - C^*}{\partial C\left(\sigma_n^{\mathrm{imp}}\right) / \partial \sigma_n^{\mathrm{imp}}}$$

- BSM模型中欧式期权的Vega（期权定价公式对于波动率的偏微分）

$$\frac{\partial C}{\partial \sigma} = S_t \mathrm{N}(d_1)\sqrt{T}$$

循环语句

遍历循环：for语句

遍历循环：

根据循环执行次数的确定性，循环可以分为确定次数循环和非确定次数循环。确定次数循环指循环体对循环次数有明确的定义循环次数采用遍历结构中元素个数来体现

Python通过保留字for实现“遍历循环”：

```
for <循环变量> in <遍历结构>:
    <语句块>
```

遍历循环：for语句

遍历结构可以是字符串、文件、组合数据类型或range()函数：

循环N次	遍历文件fi的每一行	遍历字符串s	遍历列表ls
for i in range(N):	for line in fi:	for c in s:	for item in ls:
<语句块>	<语句块>	<语句块>	<语句块>

遍历循环还有一种扩展模式，使用方法如下：

```
for <循环变量> in <遍历结构>:
    <语句块1>
else:
    <语句块2>
```

遍历循环：for语句

- 当for循环正常执行之后，程序会继续执行else语句中内容，else语句只在循环正常执行之后才执行并结束，
- 因此，可以在<语句块2>中放置判断循环执行情况的语句。

```
for s in "BIT":
    print("循环进行中： " + s)
else:
    s = "循环正常结束"
print(s)
```

```
>>>
循环进行中： B
循环进行中： I
循环进行中： T
循环正常结束
```

无限循环：while语句

- 无限循环一直保持循环操作直到特定循环条件不被满足才结束，不需要提前知道确定循环次数。

- Python通过保留字while实现无限循环，使用方法如下：

```
while <条件>:
    <语句块>
```

无限循环：while语句

- 无限循环也有一种使用保留字else的扩展模式：

```
while <条件>:
    <语句块1>
else:
    <语句块2>
```

```python
s, idx = "BIT", 0
while idx < len(s):
    print("循环进行中： " + s[idx])
    idx += 1
else:
    s = "循环正常结束"
print(s)
```

```
>>>
循环进行中： B
循环进行中： I
循环进行中： T
循环正常结束
```

循环保留字：break和continue

- 循环结构有两个辅助保留字：break和continue，它们用来辅助控制循环执行；
- break用来跳出最内层for或while循环，脱离该循环后，程序从循环后的代码继续执行；

```
for s in "BIT":
    for i in range(10):
        print(s, end="")
        if s=="I":
            break
```

其中，break语句跳出了最内层for循环，但仍然继续执行外层循环。每个break语句只有能力跳出当前层次循环 。

```
>>>
BBBBBBBBBBITTTTTTTTTT
```

循环保留字：break和continue

- continue用来结束当前当次循环，即跳出循环体中下面尚未执行的语句，但不跳出当前循环。
- 对于while循环，继续求解循环条件。而对于for循环，程序流程接着遍历循环列表
- 对比continue和break语句，如下

```
for s in "PYTHON":
    if s=="T":
        continue
    print(s, end="")
```

```
>>>
PYHON
```

```
for s in "PYTHON":
    if s=="T":
        break
    print(s, end="")
```

```
>>>
PY
```

循环保留字：break和continue

continue语句和break语句的区别是：

- continue语句只结束本次循环，而不终止整个循环的执行。
- break语句则是结束整个循环过程，不再判断执行循环的条件是否成立

```python
for s in "PYTHON":
    if s=="T":
        continue
    print(s, end="")
```

```
>>>
PYHON
```

```python
for s in "PYTHON":
    if s=="T":
        break
    print(s, end="")
```

```
>>>
PY
```

循环保留字：break和continue

- for循环和while循环中都存在一个else扩展用法。
- else中的语句块只在一种条件下执行，即for循环正常遍历了所有内容没有因为break或return而退出。
- continue保留字对else没有影响。看下面两个例子：

```
for s in "PYTHON":
    if s=="T":
        continue
print(s, end="")
else:
    print("正常退出")
```

```
>>>
PYHON正常退出
```

```
for s in "PYTHON":
    if s=="T":
        break
print(s, end="")
else:
print("正常退出")
```

```
>>>
PY
```

代码

欧式看涨/跌期权的隐含波动率

```
#Implied volatility function based on a European call
def implied_vol_call(S,X,T,r,c):
    from scipy import log,exp,sqrt,stats
    for i in range(200):
        sigma=0.005*(i+1)
        d1=(log(S/X)+(r+sigma*sigma/2.)*T)/(sigma*sqrt(T))
        d2 = d1-sigma*sqrt(T)
        diff=c-(S*stats.norm.cdf(d1)-X*exp(-r*T)*stats.norm.cdf(d2))
        if abs(diff)<=0.01:
            return i,sigma, diff


#Implied volatility based on a put option model
def implied_vol_put_min(S,X,T,r,p):
    from scipy import log,exp,sqrt,stats
    implied_vol=1.0
    min_value=100.0
    for i in range(1,10000):
        sigma=0.0001*(i+1)
        d1=(log(S/X)+(r+sigma*sigma/2.)*T)/(sigma*sqrt(T))
        d2 = d1-sigma*sqrt(T)
        put=X*exp(-r*T)*stats.norm.cdf(-d2)-S*stats.norm.cdf(-d1)
        abs_diff=abs(put-p)
        if abs_diff<min_value:
            min_value=abs_diff
            implied_vol=sigma
            k=i
            put_out=put
            print ('k,implied_vol, put_out,abs_diff')
            return k,implied_vol, put_out,min_value
```

bsm_functions.py

```
# Valuation of European call options in Black-Scholes-Merton
model
# incl. Vega function and implied volatility estimation
# bsm_functions.py
#
# Analytical Black-Scholes-Merton (BSM) Formula

def bsm_call_value(S0, K, T, r, sigma):
    """
Valuation of European call option in BSM model.
Analytical formula.
Parameters
==========
S0 : float
initial stock/index level
K : float
strike price
T : float
maturity date (in year fractions)
r : float
constant risk-free short rate
sigma : float
volatility factor in diffusion term
Returns
=======
value : float
present value of the European call option
    """
    from math import log, sqrt, exp
    from scipy import stats
    S0 = float(S0)
    d1 = (log(S0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * sqrt(T))
    d2 = (log(S0 / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * sqrt(T))
    value = (S0 * stats.norm.cdf(d1, 0.0, 1.0)
             - K * exp(-r * T) * stats.norm.cdf(d2, 0.0, 1.0))
     # stats.norm.cdf --> cumulative distribution function
     # for normal distribution
    return value
```

```
# Vega function
def bsm_vega(S0, K, T, r, sigma):
    '''
Vega of European option in BSM model.
Parameters
==========
S0 : float
initial stock/index level
K : float
strike price
T : float
maturity date (in year fractions)
r : float
constant risk-free short rate
sigma : float
volatility factor in diffusion term
Returns
=======
vega : float
partial derivative of BSM formula with
Respect to sigma, i.e. Vega
    '''
    from math import log, sqrt
    from scipy import stats
    S0 = float(S0)
    d1 = (log(S0/K)+(r+0.5*sigma**2)*T)/(sigma*sqrt(T))
    vega = S0*stats.norm.cdf(d1,0.0,1.0)*sqrt(T)
    return vega
```

```
Implied volatility of European call option in BSM model.
Parameters
==========
S0 : float
initial stock/index level
K : float
strike price
T : float
maturity date (in year fractions)
r : float
constant risk-free short rate
sigma_est : float
estimate of impl. volatility
it : integer
number of iterations

Returns
=======
simga_est : float
numerically estimated implied volatility
     '''
    for i in range(it):
        sigma_est -= ((bsm_call_value(S0, K, T, r, sigma_est) - C0)
                        / bsm_vega(S0, K, T, r, sigma_est))
    return sigma_est
```

小 结

- 隐含波动率
- Python循环语句
- 编写计算隐含波动率