5 树形计算亚式期权价格

使用申明*

2021 年 2 月 22 日

目录

1 简介

1.1 亚式期权

亚式期权的价格由股票价格在期权有效期内的价格变化路径决定。特别地，对于亚式平均价格看涨期权，在执行时刻期权价格为 $\max(S_{ave}-K,0)$。其中 K 为约定的执行价，S_{ave} 为历史股价平均值。对于平均价格看跌期权，在执行时其价格为 $\max(K-S_{ave},0)$。

1.2 亚式期权分类

1. 平均价格期权：如上文所述，平均价格看涨和看跌期权在执行时刻的价格分别为 $\max(S_{ave}-K,0)$ 和 $\max(K-S_{ave},0)$。
2. 平均执行价格期权：对于平均执行价格期权，其看涨和看跌期权在执行时刻的价格分别为 $\max(S_T-S_{ave},0)$，$\max(S_{ave}-S_T,0)$，其中 S_T 为执行时刻 T 时的股票价格。
3. 美式/欧式期权：平均价格期权或者平均执行价格期权，都可以是美式或者欧式。如果期权为美式，则持有者有权在期权有效期内任一时刻选择执行期权。

2 树形计算亚式期权价格步骤

我们依旧首先建立一个描述股价变化的二叉树。然后确定对于每个节点，到达该节点所有可能路径的股价平均值的极大和极小值，在极大和极小值之间等间距地插入一定数量数值点。然后根据亚式期权具体类型确定叶子层每

个节点上历史股价极大和和极小值之间所有数值点对应的期权价格。再逐层递推回根节点处。递推时每个平均股价对应的期权价格的计算需要用到插值法。这里用固定一定数量的数值点而不是所有可能的历史路径平均股价是因为所有可能的路径平均股价的数量可能非常大。

步骤具体为：

1. 根据给定的相关参数计算出二叉树分叉参数 p, u, d，并建立股价变化树形。

$$u = e^{\sigma\sqrt{\Delta t}}, \quad d = \frac{1}{u}, \quad p = \frac{e^{r\Delta t} - d}{u - d}\ . \tag{1}$$

2. 计算并保存树形上每个节点处历史路径平均股价的极大值 S_{ave_max} 和极小值 S_{ave_min}，

$$S_{ave\_max} = \frac{1}{i+1}\left[S_0\frac{1-u^{j+1}}{1-u} + S_0u^jd\frac{1-d^{i-j}}{u-d}\right]\ , \tag{2}$$

$$S_{ave\_min} = \frac{1}{i+1}\left[S_0\frac{1-d^{i-j+1}}{1-d} + S_0d^{i-j}u\frac{1-u^j}{1-u}\right]\ . \tag{3}$$

其中 S_0 为初始化股价，i 为节点所在层数（根节点为 0），j 为节点在该层的位置（最低股价处对应 0）。然后在极大和极小值之间插入固定数量的数值点。

3. 在树形叶子层，根据期权具体类型，计算出所有历史平均股价数值点对应的期权价格并保存下来。

4. 往上一层递推。计算出该层每个节点的每个可能的历史平均股价数值点对应的期权价格。考虑一个具体的平均股价数值点，该点在股价下一步上升或下降后分别对应于下一层一个节点的一个历史平均股价数值，但该价格不一定是在已计算出的数值点中，所以一般需要使用插值法求出数值点对应的期权价格。然后对股价上升或下降后对应数值所对应期权价格进行加权平均并贴现，即为当前层考虑节点的一个平均股价数值点对应的期权价格。

5. 重复过程 4，直到计算出根节点处期权的价格。

如果期权为美式，则在过程 4 中需要考虑在每个节点的每个历史平均股价数值点处应不应该执行期权，其它部分的过程不变。

3 步骤 Python 代码实现

```
import math

E = math.e

class Tree_asian_option:
    def __init__(self, r, sigma, S_0, K, T, steps, points):
        """ 初始化实例，points为每个节点处历史平均股票价格数值点的数量。
        """
        self.r = r
        self.sigma = sigma
        self.S_0 = S_0
        self.K = K
        self.T = T
        self.steps = steps
        self.points = points

        self.dt = self.T/self.steps
        self.u = E**(self.sigma*self.dt**0.5)
        self.d = 1/self.u
        self.p = (E**(self.r*self.dt)- self.d)/(self.u- self.d)
```

```
        self.call_price = None
        self.tree = list()
        self.build_tree()

    def get_max_min_ave(self, i, j):
        """ 计算(i,j)节点处历史路径平均股票价格的极大和极小值。
        """
        u, d, S_0 = self.u, self.d, self.S_0
        max_ave = S_0*(1-u**(j+1))/(1-u)+S_0*(u**j)*d*(1-d**(i-j))/(1-d)
        max_ave /= 1+i
        min_ave = S_0*(1-d**(i-j+1))/(1-d)+S_0*(d**(i-j))*u*(1-u**j)/(1-u)
        min_ave /= 1+i

        return (max_ave, min_ave)

    def interpolation(self, x, ref_list):
        """ 线性插值，ref_list 为维度为 points x 2 的数组，默认ref_list按ref_list[i][0]的大小由小到大排序。
        """
        left, right = 0, len(ref_list)-1
        pos = 0

        # 如果在范围外，则返回边界值。实际上我们考虑的树形上不会出现这种情况。
        if x > ref_list[right][0]:
            return ref_list[right][1]
        if x < ref_list[left][0]:
            return ref_list[left][1]

        # 二分法查找出x在{ref_list[i][0]}中的位置。
        while left < right:
            pos = int((left+right)/2)
            if x == ref_list[pos][0]:
                return ref_list[pos][1]
            if x > ref_list[pos][0]:
                left = pos+1
            else:
                right = pos-1
        if x > ref_list[left][0]:
            pos = left+1
        else:
            pos = left

        # 线性插值。
        result = (ref_list[pos][1]-ref_list[pos-1][1])/(ref_list[pos][0]-ref_list[pos-1][0])
        result *= (x-ref_list[pos-1][0])
        result += ref_list[pos-1][1]

        return result

    def build_tree(self):
        S_0, steps, points = self.S_0, self.steps, self.points
        u, d, p = self.u, self.d, self.p
```

```
        self.tree = list()
        for lvl in range(steps+1):
            row = list()
            for j in range(lvl+1):
                node = dict()
                node["S"] = S_0*(u**j)*(d**(lvl-j))
                node["F_S"] = list()
                max_ave, min_ave = self.get_max_min_ave(lvl, j)
                for k in range(points):
                    node["F_S"].append([(max_ave-min_ave)/(points-1)*k+min_ave, None])
                row.append(node)
            self.tree.append(row)

        return

    def calculate_call_price(self):
        """ 计算欧式平均价格看涨期权的价格。
        """
        r, S_0, K = self.r, self.S_0, self.K
        steps, points = self.steps, self.points
        dt, u, d, p = self.dt, self.u, self.d, self.p

        a = E**(-r*dt)
        # 边界条件。
        for node in self.tree[-1]:
            for k in range(points):
                node["F_S"][k][1] = max(0, node["F_S"][k][0]-K)

        # 递推回根节点。
        for lvl in range(steps-1, -1, -1):
            for j in range(lvl+1):
                node = self.tree[lvl][j]
                for k in range(points):
                    new_ave_u = (node["F_S"][k][0]*(lvl+1)+self.tree[lvl+1][j+1]["S"])/(lvl+2)
                    new_ave_d = (node["F_S"][k][0]*(lvl+1)+self.tree[lvl+1][j]["S"])/(lvl+2)
                    option_price_u = self.interpolation(new_ave_u, self.tree[lvl+1][j+1]["F_S"])
                    option_price_d = self.interpolation(new_ave_d, self.tree[lvl+1][j]["F_S"])
                    node["F_S"][k][1] = a*p*option_price_u+a*(1-p)*option_price_d

        self.call_price = self.tree[0][0]["F_S"][0][1]

        return
```

4 计算示例

当无风险利率为 0.1，股价波动率为 0.4，股价初始值为 50。考虑一个平均价格看涨期权，执行时间为 1 年后，执行价格为 50。我们使用步数为 60 步的二叉树，每个节点的历史股价平均值由 100 个等间隔数值代替。计算如下：

```
print("r=0.1, sigma=0.4, S_0=50, K=50, T=1, steps=60, points=100 . \n")
tree_obj = Tree_asian_option(0.1, 0.4, 50, 50, 1, 60, 100)
```

```
tree_obj.calculate_call_price()
print("欧式平均价格看涨期权价格为：{0:.5f}".format(tree_obj.call_price))

Output:
    r=0.1, sigma=0.4, S_0=50, K=50, T=1, steps=60, points=100 .
    欧式平均价格看涨期权价格为：5.57973
```

5 相关说明

5.1 历史平均股价的极大和极小值

对于第 i 层，第 j 个节点 (i,j)，不考虑上升或下降的次序，股价到达该节点一共经历了 j 次上升，$(i-j)$ 次下降。到达该节点的路径的平均股价的极大情况对应于一条股价先上升 j 次后下降 $(i-j)$ 次的路径。到达该节点的平均股价的极小情况，对应于一条先下降 $(i-j)$ 次后上升 j 次的路径。具体如下图：

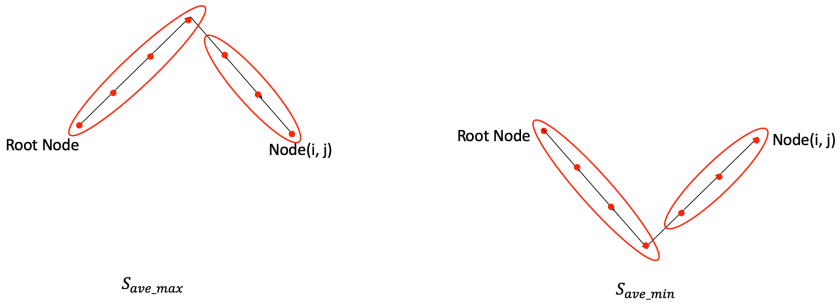

Fig. 1

$$S_{ave\_max} = \frac{1}{i+1}\left[S_0\frac{1-u^{j+1}}{1-u} + S_0u^jd\frac{1-d^{i-j}}{1-d}\right], \tag{4}$$

$$S_{ave\_min} = \frac{1}{i+1}\left[S_0\frac{1-d^{i-j+1}}{1-d} + S_0d^{i-j}u\frac{1-u^j}{1-u}\right]. \tag{5}$$

5.2 期权价格的递推

节点 (i,j) 在代码中对应 $self.tree[i][j]$，是一个 dict() 型变量。其中有元素"S" 和"F_S"，"S" 对应的值为当前节点股价 $S_{i,j}$。"F_S" 对应一个二维 list() 变量，存储平均股价数值点和对应期权价格共 $N = points$ 对。"F_S" 对应的数组相当于以离散的数值点的方式描述了一个历史平均股价到期权价格的函数。

考虑一个平均股价数值点 $S_{ave}(k) = \frac{k}{N-1}(S_{ave\_max} - S_{ave\_min}) + S_{ave\_min}$，我们需要计算其对应期权价格。由于股价由当前节点有 p 的概率上升，$(1-p)$ 的概率下降，如果股价上升，则历史平均股价将变成 $\frac{1}{i+2}(S_{ave}(k)\times(i+1)+uS_{i,j})$，可以使用下一层的节点 $(i+1,j+1)$ 的"F_S" 数组进行插值计算出期权价格。股价若下降，历史平均股价变为 $\frac{1}{i+2}(S_{ave}(k)\times(i+1)+dS_{i,j})$，可以使用下一层的节点 $(i+1,j)$ 的"F_S" 数组进行插值计算出期权的价格。当前节点的 $S_{ave}(k)$ 对应的期权价格即为以上两个期权价格加权平均并贴现后的值。

6 参考资料

参考文献

[1]《期权、期货及其他衍生产品》（原书第 9 版）第 27 章，John C. Hull 著，王勇、索吾林译，机械工业出版社，2014.11 。