15 单因子利率模型蒙卡模拟

被使用申明*

2021 年 1 月 30 日

目录

1 **简介** 1
1.1 单因子模型蒙卡抽样 1
1.2 由抽样结果计算债券期权价格 2
2 **蒙卡模拟步骤** 2
2.1 蒙卡抽样利率变化路径 2
2.2 计算欧式零息债券期权价格 3
3 **步骤 C 代码实现** 3
4 **计算示例** 7
5 **参考资料** 8

1 简介

1.1 单因子模型蒙卡抽样

在单因子模型下，瞬时无风险利率的变化过程为

$$dr(t) = [\theta(t) - ar(t)]dt + \sigma dz \ . \tag{1}$$

其中 a 和 σ 为常数，$\theta(t)$ 由初始零息利率期限结构确定。

如果我们考虑在 0 至 T 时间段内，利率 $r(t)$ 的变化过程被离散化为 $steps$ 次变化，在不同时间区间 $r(t)$ 的取值为 $r(t_i)$, $t_i = i \times \Delta t$, $i = 0, 1, ..., steps$, $\Delta t = T/(steps + 1)$。$r(t_i)$ 由瞬时无风险利率变为 t_i 至 $t_{i+1} = t_i + \Delta t$ 之间的短期无风险利率。离散化后的 $r(t_i)$ 的变化过程可以表示为

$$r(t_i + \Delta t) = r(t_i) + [\theta_i - ar(t_i)]\Delta t + \sigma\sqrt{\Delta t}\,\varepsilon \ . \tag{2}$$

ε 为一标准正态分布随机数，θ_i 为由初始零息利率表示的变量，下面说明如何确定 θ_i。

由《期权、期货及其他衍生产品》书中第 31 章内容 [1, chapter 31]，我们知道在利率 $r(t)$ 连续变化的情况下，

$$\theta(t) = F_t(0,t) + aF(0,t) + \frac{\sigma^2}{2a}\left(1 - e^{-2at}\right) \ . \tag{3}$$

其中 $F(0,t)$ 和 $F_t(0,t)$ 分别为 t 时刻的瞬时远期利率和 t 时刻瞬时远期利率的导数。

然后为了清晰方便起见，我们先统一将各种利率都用 $R(t_0, t_1, t_2)$ 表示，其中 t_0 为观察时刻，t_1 为利率作用开始时刻，t_2 为结束时刻。此时初始零息利率 $R(t) = R(0, 0, t)$，离散化后的短期无风险利率 $r(t_i) = R(t_i, t_i, t_i + \Delta t)$，短期远期利率 $F(0, t_i) = R(0, t_i, t_i + \Delta t)$。考虑在连续复利时，

$$\exp[-R(0,0,t_i) \times t_i] \cdot \exp[-R(0,t_i,t_{i+1}) \times \Delta t] = \exp[-R(0,0,t_{i+1}) \times t_{i+1}] \ , \tag{4}$$

得

$$F(0,t_i)=R(0,t_i,t_i+\Delta t)=\frac{1}{\Delta t}\left[R(0,0,t_i+\Delta t)\times(t_i+\Delta t)-R(0,0,t_i)\times t_i\right]\ . \tag{5}$$

对于 $F_t(0,t_i)$，可以向后取近似，

$$F_t(0,t_i)=\frac{1}{\Delta t}\left[F(0,t_i)-F(0,t_{i-1})\right]\ . \tag{6}$$

由于在短期无风险利率 $r(t_i)$ 的离散化变化过程中，$r(t_{i+1})$ 为 t_{i+1} 至 t_{i+2} 之间的利率，$r(t_i)$ 为 t_i 至 t_{i+1} 之间的利率。发现建立两者变化关系时，将 θ_i 表示为如下形式会比较合适，

$$\theta_i=F_t(0,t_{i+1})+aF(0,t_{i+1})+\frac{\sigma^2}{2a}\left(1-e^{-2at_{i+1}}\right)\ . \tag{7}$$

用初始零息利率可以表示为

$$\begin{aligned}\theta_i=&\frac{1}{\Delta t^2}\left[R(t_i+2\Delta t)\times(t_i+2\Delta t)+R(t_i)\times t_i-2R(t_i+\Delta t)\times(t_i+\Delta t)\right]\\&+\frac{a}{\Delta t}\left[R(t_i+2\Delta t)\times(t_i+2\Delta t)-R(t_i+\Delta t)\times(t_i+\Delta t)\right]+\frac{\sigma^2}{2a}\left(1-e^{-2a(t_i+\Delta t)}\right)\ .\end{aligned} \tag{8}$$

确定了 θ_i 的表达式之后，我们就建立了 $r(t_{i+1})$ 和 $r(t_i)$ 之间的递推关系。然后我们还需要确定初始利率 $r(t_0)$，由 $r(t_0)=R(0,t_0,t_0+\Delta t)=R(0,0,\Delta t)$，知 $r(t_0)$ 即为初始零息利率曲线上 Δt 处的利率。这样我们就可以从 $r(t_0)$ 开始，使用式 $r(t_i+\Delta t)=r(t_i)+[\theta_i-ar(t_i)]\Delta t+a\sqrt{\Delta t}\,\varepsilon$ 和正态分布随机数 ε 逐步抽样出一条 $r(t_i)$ 的变化路径。

1.2 由抽样结果计算债券期权价格

考虑一在零息债券上的欧式看涨期权，债券的本金为 L，期权的执行价格为 K，期权的到期时间为 T_1，债券的到期时间为 T_2，$T_2>T_1$。该债券期权在当前的价格可以表示为

$$C=\int_0^{r_{max}}Q(r(T_1))\,\max\left[L\cdot P(r(T_1))-K,\,0\right]dr(T_1)\ . \tag{9}$$

其中 r_{max} 为一设定的合适利率上限，$Q(r(T_1))$ 为由 $t=0$ 利率为 $r(t_0)$ 至 $t=T_1$ 利率为 $r(T_1)$ 时，所有可能的 $r(t)$ 变化的路径下对应的贴现按路径出现的概率进行加权平均所得值，$P(r(T_1))$ 为在 T_2 时刻到期的 1 美元零息债券在 T_1 时刻当瞬时无风险利率为 $r(T_1)$ 时的价格。

如果我们想要使用蒙卡抽样出的一些 $r(t)$ 由 $t=0$ 至 $t=T_2$ 的变化路径来计算该期权的价格 C。首先我们需要将 $r(T_1)$ 的取值范围由 $[0,\,r_{max}]$ 划分为一些小的区间。然后找到 T_1 处于抽样路径中哪一个小的时间间隔，即找到 j，使得 $t_j\le T_1<t_{j+1}$。再将每条路径依其 $r(t_j)$ 的大小处于哪一个 $r(T_1)$ 取值区间进行分类，并近似认为同一类路径中的 $r(t_j)$ 都是相同的。这时，

$$C\approx\sum_{k=1}^{M}Q'_k\,\max\left(P_k-L,\,0\right)\ . \tag{10}$$

M 为 $r(T_1)$ 可能取值的区间的数量。Q'_k 为第 k 个区间对应的那些利率抽样路径中 $t=0$ 至 $t=T_1$ 部分的贴现的平均值乘以这些路径占总抽样数量的比例。P_k 是第 k 个区间对应的那些路径中 $t=T_1$ 至 $t=T_2$ 部分的贴现的平均值。

如果我们将 $r(T_1)$ 的取值区间划分得更加精细，增加利率在 0 至 T_2 之间的变化次数，然后增加利率路径的抽样数量，那么由上式所得的看涨期权价格将逐渐收敛于其解析结果。

2 蒙卡模拟步骤

2.1 蒙卡抽样利率变化路径

1. 假定 a，σ，初始零息利率期限结构已知。我们将瞬时无风险利率 $r(t)$ 由 $t=0$ 至 $t=T_2$ 之间的变化过程离散化为 $steps$ 次变化，此时 $\Delta t=T_2/(steps+1)$。$r(t_i)$ 为 $t=i\times\Delta t$ 至 $t=(i+1)\times\Delta t$ 之间的短期无风险利率。

2. 写一个函数 R0t(rates, t)，输入初始零息利率期限结构 rates 和时间 t，输出线性插值得到的期限为 t 的零息利率。在线性插值时，对于期限在已知的初始利率结构时间范围外的点，将其值设为最接近的利率。

3. 写一个函数 Theta_ts(a, sigma, T, steps, rates)，输入相关参数，输出将在蒙卡抽样中被用到的所有的 θ_i 的值。

$$\theta_i = \frac{1}{\Delta t^2}\left[R(t_i+2\Delta t)\times(t_i+2\Delta t)+R(t_i)\times t_i-2R(t_i+\Delta t)\times(t_i+\Delta t)\right] + \frac{a}{\Delta t}\left[R(t_i+2\Delta t)\times(t_i+2\Delta t)-R(t_i+\Delta t)\times(t_i+\Delta t)\right]+\frac{\sigma^2}{2a}\left(1-e^{-2a(t_i+\Delta t)}\right).$$

4. 写一个蒙卡抽样函数，输入 a, sigma, T, steps, theta_ts, R0, 输出一条蒙卡抽样利率变化路径。此处 R0 为 $R(0,0,\Delta t)$，即利率初始取值。利率 $r(t_i)$ 的递推关系为

$$r(t_i+\Delta t) = r(t_i)+[\theta_i - ar(t_i)]\Delta t+\sigma\sqrt{\Delta t}\,\varepsilon\,.$$

使用该函数我们就可以进行蒙特卡罗抽样，得到一条 $r(t_i)$ 可能的变化路径。

2.2 计算欧式零息债券期权价格

1. 考虑期权期限为 T_1，债券期限为 T_2，期权执行价格为 K，债券本金为 L。设短期无风险利率 $r(t)$ 在 $t=0$ 至 $t=T_2$ 之间变化 $steps$ 次，将要抽取 num_paths 条利率变化路径。此时 $dt = T_2/(steps+1)$，并可以由 dt 确定一指标 tp，使得 $t_{tp} \le T_1 < t_{tp+1}$。

2. 由上面所写的 R0t() 和 Theta_ts() 函数计算出 $R0$ = R0t(rates, Δt) 和 theta_ts 数组。

3. 将 $r(T_1)$ 的取值区间由 [0, r_{max}] 划分为等大小的 $intervals$ 个区间，这里 r_{max} 可以设为 0.3，此时区间的大小为 $0.3/intervals$。

4. 初始化两个 $intervals\times 2$ 大小的数组 Qs 和 Ps。Qs 留作存储每个 $r(T_1)$ 取值区间内相应的路径数量和各路径上 $t=0$ 至 $t=T_1$ 之间的贴现值之和，Ps 留作存储每个区间内路径数量和各路径上 $t=T_1$ 至 $t=T_2$ 之间的贴现值之和。Qs 和 Ps 内所有元素都初始化为 0。

5. 由 R0，theta_ts 和相关参数，使用上面写的蒙卡抽样函数抽样出一条利率变化路径。根据 $r(t_{tp})$ 的大小判断其位于 $r(T_1)$ 的哪一个取值区间，并计算出该路径下 $t=0$ 至 $t=T_1$ 的贴现，和 $t=T_1$ 至 $t=T_2$ 之间的贴现，更新 Qs 和 Ps 数组。重复这个过程 num_paths 次。

6. 初始化看涨期权和看跌期权价格均为 0。同时遍历 Qs 和 Ps，如果区间 i 中的路径数量 Ps[i][0] 不为零，则 Ps[i][1]/Ps[i][0] 为 T_1 时刻债券价格。如果该价格大于期权执行价格 K，则将其减去 K 再乘以 Qs[i][1]/num_paths，并将结果加入看涨期权价格。如果债券价格小于 K，类似计算并将结果加入看跌期权价格。最终得到的看涨期权价格和看跌期权价格即为蒙卡模拟计算出的零息债券期权价格。

3 步骤 C 代码实现

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double normal()
{
    double x, y;
    double result;
    x = (0.5+rand())/((double)RAND_MAX+1.0);
    y = (0.5+rand())/((double)RAND_MAX+1.0);
```

```
    result = sqrt(-2.0*log(x))*cos(6.283185307179586*y);
    return result;
}

double R0t(double ** rates, int num_rates, double t)
{
    double result;
    if (t<=rates[0][0])
    {
        result = rates[0][1];
    }
    else if (t>=rates[num_rates-1][0])
    {
        result = rates[num_rates-1][1];
    }
    else
    {
        int p = 0;
        while (rates[p][0]<t)
        {
            p++;
        }
        result = rates[p-1][1]+(rates[p][1]- rates[p-1][1]) /(rates[p][0]- rates[p-1][0]) \
                *(t-rates[p-1][0]) ;
    }
    return result;
}

double * Theta_ts(double a, double sigma, double T, int steps, double ** rates, int num_rates)
{
    int nR = num_rates;
    double dt = T/(1.0+steps);
    double * result = malloc(steps*sizeof(double));
    double theta_t;
    double ti;
    for (int i=0; i<steps; i++)
    {
        ti = dt*(1+i);
        theta_t = sigma*sigma/2.0/a*(1.0-exp(-2.0*a*ti));
        theta_t += a*(R0t(rates,nR,ti+dt)*(ti+dt)-R0t(rates,nR,ti)*ti)/dt;
        theta_t += ((ti+dt)*R0t(rates,nR,ti+dt)+(ti-dt)*R0t(rates,nR,ti-dt)\
                -2.0*ti*R0t(rates,nR,ti))/dt/dt;
        result[i] = theta_t;
    }
    return result;
}

double * MC_one_factor_sample(double a, double sigma, double T, int steps, double R0, double * theta_ts)
{
    double dt = T/(1.0+steps);
    double * result = malloc((steps+1)*sizeof(double));
    result[0] = R0;
```

```
    double Ri = R0;
    for (int i=1; i<steps+1; i++)
    {
        Ri = Ri+(theta_ts[i-1]-a*Ri)*dt+sigma*sqrt(dt)*normal();
        result[i] = Ri;
    }
    return result;
}

double * MC_one_factor_bond_option(double K, double L, double T1, double T2, \
                                   double a, double sigma, int steps, int num_paths,\
                                   int intervals, double ** rates, int num_rates)
{
    double dt = T2/(steps+1.0);
    int tp = (int)(T1/dt);
    double dt1 = T1-tp*dt;
    double dt2 = dt-dt1;

    double * theta_ts = Theta_ts(a, sigma, T2, steps, rates, num_rates);
    double R0 = R0t(rates, num_rates, dt);

    double dR = 0.3/intervals;
    double ** Qs = malloc(intervals*sizeof(double *));
    double ** Ps = malloc(intervals*sizeof(double *));
    for (int i=0; i<intervals; i++)
    {
        Qs[i] = malloc(2*sizeof(double));
        Qs[i][0] = 0.0;
        Qs[i][1] = 0.0;
        Ps[i] = malloc(2*sizeof(double));
        Ps[i][0] = 0.0;
        Ps[i][1] = 0.0;
    }

    double * R_path;
    double discount_factor;
    int idx;
    for (int i=0; i<num_paths; i++)
    {
        R_path = MC_one_factor_sample(a, sigma, T2, steps, R0, theta_ts);
        discount_factor = 0.0;
        for (int j=steps; j>tp; j--)
        {
            discount_factor += dt*R_path[j];
        }
        discount_factor += dt2*R_path[tp];

        idx = (int) (R_path[tp]/dR);
        idx = (idx<intervals) ? idx : intervals-1;
        idx = (idx>=0) ? idx : 0;
        Ps[idx][0] += 1.0;
        Ps[idx][1] += L*exp(-discount_factor);
```

```
116
117         discount_factor = dt1*R_path[tp];
118         for (int j=tp-1; j>-1; j--)
119         {
120             discount_factor += dt*R_path[j];
121         }
122         Qs[idx][0] += 1.0;
123         Qs[idx][1] += exp(-discount_factor);
124
125         free(R_path);
126     }
127
128     double bond_price;
129     double * call_put_prices = malloc(2*sizeof(double));
130     call_put_prices[0] = 0.0;
131     call_put_prices[1] = 0.0;
132     for(int i=0; i<intervals; i++)
133     {
134         if (Ps[i][0]<1.E-14)
135         {
136             continue;
137         }
138         else
139         {
140             bond_price = Ps[i][1]/Ps[i][0];
141             if (bond_price>K)
142             {
143                 call_put_prices[0] += (bond_price-K)*Qs[i][1]/num_paths;
144             }
145             else
146             {
147                 call_put_prices[1] += (K-bond_price)*Qs[i][1]/num_paths;
148             }
149         }
150     }
151
152     for (int i=0; i<intervals; i++)
153     {
154         free(Qs[i]);
155         free(Ps[i]);
156     }
157     free(Qs);
158     free(Ps);
159     free(theta_ts);
160
161     return call_put_prices;
162 }
```

4 计算示例

我们参考《期权、期货及其他衍生产品》书中第 31 章例 31-4，债券期权为零息债券上的看跌期权，期权执行时间为 $T_1 = 3.0$，债券到期时间为 $T_2 = 9.0$，期权执行价为 $K = 63.0$，债券本金为 $L = 100.0$。初始利率结构已知，Hull-White 单因子模型中已知参数 $a = 0.1$, $\sigma = 0.01$。

然后利率变化次数 *steps* 设为 500，抽样路径数量 num_paths 设 100000，将 [0, 0.3] 均匀划分为 $intervals = 200$ 个小区间作为 $r(T_1)$ 的可能取值区间。此时我们可以进行一次模拟计算，并得到该欧式零息看涨期权和看跌期权的价格分别为 1.05396 和 1.80120。

```
int main()
{
    srand(time(NULL));

   int num_rates = 15;
   double rates_array[15][2] = {{3/365.0, 0.0501722}, {31/365.0, 0.0498284}, {62/365.0, 0.0497234},\
                {94/365.0, 0.0496157}, {185/365.0, 0.0499058}, {367/365.0, 0.0509389},\
                {731/365.0, 0.0579733}, {1096/365.0, 0.0630595}, {1461/365.0, 0.0673464},\
                {1826/365.0, 0.0694816}, {2194/365.0, 0.0708807}, {2558/365.0, 0.0727527}, \
                {2922/365.0, 0.0730852}, {3287/365.0, 0.0739790}, {3653/365.0, 0.0749015}};
    double ** rates = malloc(num_rates*sizeof(double *));
    for (int i=0; i<num_rates; i++)
    {
        rates[i] = malloc(2*sizeof(double));
        rates[i][0] = rates_array[i][0];
        rates[i][1] = rates_array[i][1];
    }

    double a = 0.1;
    double sigma = 0.01;
    double T1 = 3.0;
    double T2 = 9.0;
    double K = 63.0;
    double L = 100.0;
    int steps = 500;
    int num_paths = 100000;
    int intervals = 200;

    double * call_put_prices = MC_one_factor_bond_option(K, L, T1, T2, a, sigma, steps,\
                                                         num_paths, intervals, rates, num_rates);

    printf("Steps: %d , number of paths: %d, R(T1) sectors: %d .\n", steps, num_paths, intervals);
    printf("Call price: %.5lf , put price: %.5lf .\n", call_put_prices[0], call_put_prices[1]);

    for (int i=0; i<num_rates; i++)
    {
        free(rates[i]);
    }
    free(rates);
    free(call_put_prices);
}
```

Output:

Steps: 500 , number of paths: 100000, R(T1) sectors: 200 . 44
Call price: 1.05396 , put price: 1.80120 . 45

5 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), John C. Hull 著, 王勇、索吾林译, 机械工业出版社, 2014.11 。