R包—OptionPricing讲解文档

王子豪、甘剑煌、林宇波

期权交易应用广泛，而期权定价是期权交易的首要问题。期权定价是根据影响期权价格的因素，通过适当的数学模型，去分析模拟期权价格的市场变动情况，最后获得合理的理论价格。期权市场的价格有时候会偏离公允，所以需要帮助定价的工具，即为期权定价模型。

常见的期权定价模型包括布莱克-斯科尔斯-默顿模型、二叉树模型、蒙特卡罗模拟法以及差分法，便是我们下文需要着重讨论的。

一. 期权

期权，是指一种合约，源于十八世纪后期的美国和欧洲市场，该合约赋予持有人在某一特定日期或该日之前的任何时间以固定价格购进或售出一种资产的权利。期权定义的要点如下：

1、期权是一种权利。期权合约至少涉及买家和出售人两方。持有人享有权利但不承担相应的义务。

2、期权的标志物。期权的标志物是指选择购买或出售的资产。它包括股票、政府债券、货币、股票指数、商品期货等。期权是这些标的物“衍生”的，因此称衍生金融工具。值得注意的是，期权出售人不一定拥有标的资产。期权是可以“卖空”的。期权购买人也不一定真的想购买资产标志物。因此，期权到期时双方不一定进行标的物的实物交割，而只需按价差补足价款即可。

3、到期日。双方约定的期权到期的那一天称为“到期日”，如果该期权只能在到期日执行，则称为欧式期权；如果该期权可以在到期日及之前的任何时间执行，则称为美式期权。

4、期权的执行。依据期权合约购进或售出标的资产的行为称为“执行”。在期权合约中约定的、期权持有人据以购进或售出标的资产的固定价格，称为“执行价格”。

1. 划分

按期权的权利划分，有看涨期权和看跌期权两种类型：

看涨期权（Call Options）：期权的买方向期权的卖方支付一定数额的权利金后，即拥有在期权合约的有效期内，按事先约定的价格向期权卖方买入一定数量的期权合约规定的特定商品的权利，但不负有必须买进的义务。而期权卖方有义务在期权规定的有效期内，应期权买方的要求，以期权合约事先规定的价格卖出期权合约规定的特定商品。

看跌期权（Put Options）：期权买方按事先约定的价格向期权卖方卖出一定数量的期权合约规定的特定商品的权利，但不负有必须卖出的义务。而期权卖方有义务在期权规定的有效期内，应期权买方的要求，以期权合约事先规定的价格买入期权合约规定的特定商品。

按期权的种类划分，有欧式期权和美式期权两种类型：

1.对于欧式期权，买方持有者只能在到期日选择行权。

2.对于美式期权，买方可在成交后、到期日之前的交易时段选择行权。

对于欧式期权定价存在解析解，可以直接用布莱克-斯科尔斯-默顿模型进行求解。

而对于美式期权，一般没有像布莱克-斯科尔斯-默顿模型这样的解析解，所以我们讨论三种用于美式期权定价的方法。

蒙特卡罗方法主要适用于当衍生产品的收益依赖标的资产的历史价格，或者依赖多个标的资产的情形；二叉树和有限差分主要适用于期权持有者可以提前行使的美式期权。

二. 方法概述

1. 布莱克-斯科尔斯-默顿模型

c和p分别为欧式看涨与看跌的价格，S_0为股票在时间0的价格，K为执行价格，r为连续复利的无风险利率，σ为股票价格波动率，T为期权的期限。

$$c = S_0 N(d_1) - K e^{-rT} N(d_2)$$

$$p = K e^{-rT} N(-d_2) - S_0 N(-d_1)$$

$$d_1 = \frac{ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{ln(S_0/K) + (r - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

代码实现：

```
// [[Rcpp::depends(RcppArmadillo)]]
//' compute cumulated distribution function of standard normal distribution
//'
//' @param x: value to count standard normal distribution cdf
// [[Rcpp::export]]
double cdfnorm(double x)
{
  return 0.5 * (1 + std::erf(x / std::sqrt(2)));
}


//' use BS model to compute European call option price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @return European call option price by BS model
// [[Rcpp::export]]
```

```
double European_Call(double S0, double K,double T, double r, double q, double sigma)
{
  double d1 = (std::log(S0/K)+((r-q)+sigma*sigma/2)*T)/(sigma*std::sqrt(T));
  double d2 = d1 - sigma*std::sqrt(T);
  return S0*cdfnorm(d1)-K*std::exp(-(r-q)*T)*cdfnorm(d2);
}
//' use BS model to compute European put option price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @return European put option price by BS model
// [[Rcpp::export]]
double European_Put(double S0, double K, double T,double r, double q, double sigma)
{
  double d1 = (std::log(S0/K)+((r-q)+sigma*sigma/2)*T)/(sigma*std::sqrt(T));
  double d2 = d1 - sigma*std::sqrt(T);

  return K*std::exp(-(r-q)*T)*cdfnorm(-d2)-S0*cdfnorm(-d1);
}
```

1.1 例：布莱克斯科尔斯默顿模型应用于欧式看涨期权

```
#e.g. European_Call by BS model
StockPrice = 50
StrikePrice = 50
SpotTime = 0.4167
RiskFreeRate = 0.1
DividendRate = 0.02
Sigma = 0.4
EuropeanCallPrice = European_Call(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma)
# EuropeanCallPrice = 5.913526
```

2. 二叉树

思想：

将期权的期限分成许多长度为Δt的小时间区间，在每个区间内股票价格S会上涨或者下跌，以概率p上涨为$Su(u>1)$或者以概率$1-p$下跌为$Sd(d<1)$，因此我们要确定p、u和d：
股票的期望收益率为无风险利率r，资产提供收益率q的收入。

$$p=\frac{a-d}{u-d}$$

$$u=e^{\sigma\sqrt{\Delta t}}$$

$$d=e^{-\sigma\sqrt{\Delta t}}$$

$$a=e^{(r-q)\Delta t}$$

假设股票当前价格为S_0，则在二叉树节点(i,j)的价格为$S_0 u^j d^{N-j}$。

看涨期权到期日T价格$f_{N,j} = max(S_0 u^j d^{N-j} - K, 0)$

看跌期权到期日T价格$f_{N,j} = max(K - S_0 u^j d^{N-j}, 0)$

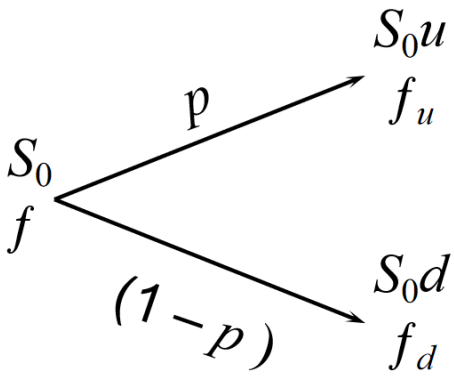
若考虑提前行使期权，则

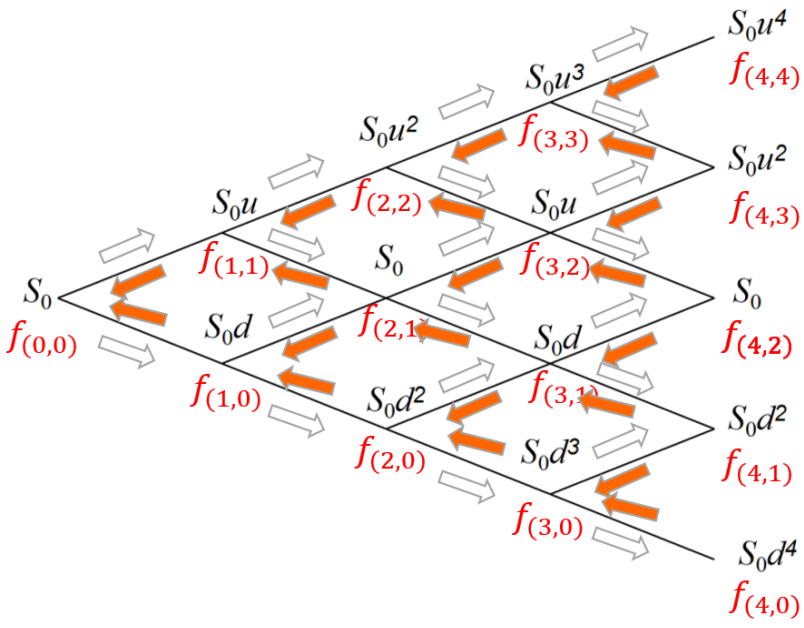
看涨期权价格$f_{i,j} = max\{S_0 u^j d^{i-j} - K, e^{(r-q)\Delta t}[p f_{i+1,j+1} + (1-p) f_{i+1,j}]\}$

看跌期权价格$f_{i,j} = max\{K - S_0 u^j d^{i-j}, e^{(r-q)\Delta t}[p f_{i+1,j+1} + (1-p) f_{i+1,j}]\}$

这样我们就可以以二叉树方法得到期权的价格，将其输出为矩阵，下三角矩阵为股票价格，(i,j)关于主对角线的对称点(j,i)即为对应的期权价格。

算法 1.	**二叉树算法**
Step 1.	确定p、u、d和a。
Step 2.	由父节点得到两个子节点的股票价格。
Step 3.	重复Step 2得到所有节点的股票价格。
Step 4.	计算二叉树尾部子节点所对应的期权价格$f_{N,j}$。
Step 5.	由尾部子节点推算对应父节点的期权价格。
Step 6.	重复向前迭代直到得到当前时刻的股票对应的期权价格。





代码实现：

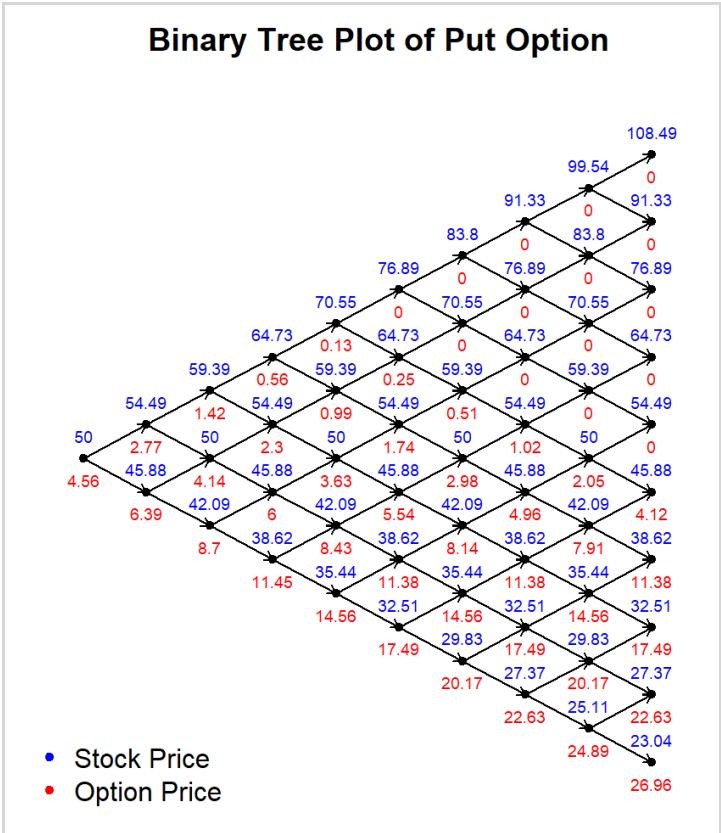
```
//' use Binary Tree method to compute American option call price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of binary tree
//' @return Binary Tree matrix of American call option, lower triangle is stock price binary tree, upper triangle is o
//' @examples
//' library(OptionPricing)
//' StockPrice = 50
//' StrikePrice = 50
//' SpotTime = 0.4167
//' RiskFreeRate = 0.1
//' DividendRate = 0.02
//' Sigma = 0.4
//' N = 10
//' BinaryTreeMatrixCall = Bitree_Call(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N)
// [[Rcpp::export]]
arma::mat Bitree_Call(double S0, double K, double T, double r, double q, double sigma, int N)
{
  arma::mat S(N+1,N+1);
  double u=std::exp(sigma*sqrt(T/(N-1)));
  double d=std::exp(-sigma*sqrt(T/(N-1)));
  double a=std::exp((r-q)*(T/(N-1)));
  double p=(a-d)/(u-d);
  int i,j;
  for(i=1; i<=N; i++){
    for(j=0; j<=i-1; j++){
      S(i,j)=S0*std::pow(u,j)*std::pow(d,i-j-1);
    }
  }
  for(i=N-1; i>=0; i--){
    S(i,N)=std::max(S(N,i)-K,0.0);
  }
  for(j=N-1; j>=1; j--){
    for(i=j-1; i>=0; i--){
      S(i,j)=std::max(S(j,i)-K,std::exp(-(r-q)*(T/(N-1)))*(p*S(i+1,j+1)+(1-p)*S(i,j+1)));
    }
  }
  return S;
}
```

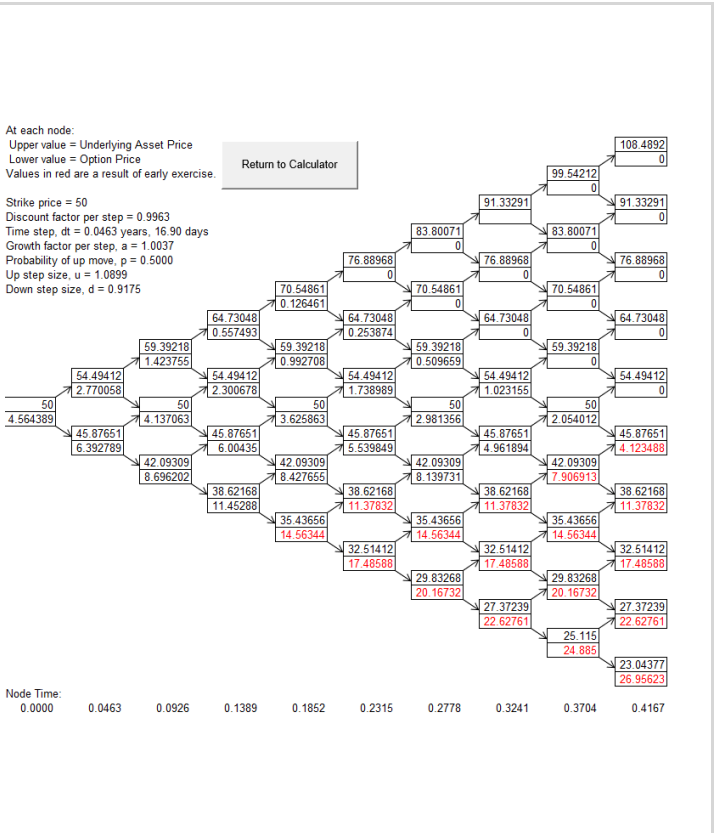
```
//' use Binary Tree method to compute American option put price
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of binary tree
//' @return Binary Tree matrix of American put option, lower triangle is stock price binary tree, upper triangle is op
//' @examples
//' library(OptionPricing)
//' StockPrice = 50
//' StrikePrice = 50
//' SpotTime = 0.4167
//' RiskFreeRate = 0.1
//' DividendRate = 0.02
//' Sigma = 0.4
//' N = 10
//' BinaryTreeMatrixPut = Bitree_Put(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N)
// [[Rcpp::export]]
arma::mat Bitree_Put(double S0, double K, double T, double r, double q, double sigma, int N)
{
  arma::mat S(N+1,N+1);
  double u=std::exp(sigma*sqrt(T/(N-1)));
  double d=std::exp(-sigma*sqrt(T/(N-1)));
  double a=std::exp((r-q)*(T/(N-1)));
  double p=(a-d)/(u-d);
  int i,j;
  for(i=1; i<=N; i++){
    for(j=0; j<=i-1; j++){
      S(i,j)=S0*std::pow(u,j)*std::pow(d,i-j-1);
    }
  }
  for(i=N-1; i>=0; i--){
    S(i,N)=std::max(-S(N,i)+K,0.0);
  }
  for(j=N-1; j>=1; j--){
    for(i=j-1; i>=0; i--){
      S(i,j)=std::max(-S(j,i)+K,std::exp(-(r-q)*(T/(N-1)))*(p*S(i+1,j+1)+(1-p)*S(i,j+1)));
    }
  }
  return S;
}
```

2.1 例：二叉树应用于美式看跌期权

```
# Call Option by Binary Tree
StockPrice = 50
StrikePrice = 50
SpotTime = 0.4167
RiskFreeRate = 0.1
DividendRate = 0.02
Sigma = 0.4
N = 10
BinaryTreeMatrixCall = Bitree_Call(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N)
Bitreeplot_Call(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	0.00000	4.564389	6.392789	8.696202	11.4528844	14.5634402	17.4858848	20.1673160	22.627610	24.885005	26.956232
2	50.00000	0.000000	2.770058	4.137063	6.0043498	8.4276551	11.3783196	14.5634402	17.485885	20.167316	22.627610
3	45.87651	54.494117	0.000000	1.423755	2.3006778	3.6258626	5.5398488	8.1397308	11.378320	14.563440	17.485885
4	42.09309	50.000000	59.392175	0.000000	0.5574926	0.9927080	1.7389887	2.9813562	4.961894	7.906913	11.378320
5	38.62168	45.876512	54.494117	64.730482	0.0000000	0.1264609	0.2538738	0.5096589	1.023155	2.054012	4.123488
6	35.43656	42.093087	50.000000	59.392175	70.5486090	0.0000000	0.0000000	0.0000000	0.000000	0.000000	0.000000
7	32.51412	38.621680	45.876512	54.494117	64.7304823	76.8896826	0.0000000	0.0000000	0.000000	0.000000	0.000000
8	29.83268	35.436560	42.093087	50.000000	59.3921750	70.5486090	83.8007067	0.0000000	0.000000	0.000000	0.000000
9	27.37239	32.514115	38.621680	45.876512	54.4941167	64.7304823	76.8896826	91.3329097	0.000000	0.000000	0.000000
10	25.11500	29.832684	35.436560	42.093087	50.0000000	59.3921750	70.5486090	83.8007067	99.542125	0.000000	0.000000
11	23.04377	27.372390	32.514115	38.621680	45.8765121	54.4941167	64.7304823	76.8896826	91.332910	108.489203	0.000000





右侧是通过Derivagem进行验证，但是Derivagem只能输出十步以内的二叉树，因此我们的程序在引用范围上做了很大的改进

3. 蒙特卡罗方法

利用蒙特卡罗模拟法，我们首先在风险中性世界里随机产生标的资产价格的路径，并由此取得收益的期望值，然后再对其按无风险利率贴现。考虑依赖某单个市场变量S并在T时刻产生收益的衍生产品。假定利率r为常数，我们用一下算法对衍生产品定价：

算法 2.	蒙特卡罗模拟法
Step 1.	在风险中性世界对S的随机路径进行抽样。
Step 2.	计算衍生产品收益。
Step 3.	重复Step1和Step2，取得许多该衍生产品收益的样本。
Step 4.	计算收益平均值作为衍生产品在风险中性世界的期望收益的估计值。
Step 5.	以无风险利率对期望值贴现，所得结果即为风险中性世界里期望收益值的估计值

代码实现：

```
//'use Monte Carlo method to compute American option call price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: Monte Carlo iteration times
//' @return American call option price by Monte Carlo method
// [[Rcpp::export]]
double MonteCarlo_Call(double S0, double K, double T,double r, double q, double sigma, int N)
{
  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();

  std::default_random_engine gen(seed);
  double sumOpCall=0.0;
  for(int i(0); i<N; ++i)
  {
    std::normal_distribution<double> dis(0,1);
    sumOpCall += std::exp(-(r-q)*T)*std::max(S0*exp(((r-q)-sigma*sigma/2)*T + sigma*dis(gen)*std::sqrt(T))-K,0.0);
  }
  return sumOpCall/N;
}
//'use Monte Carlo method to compute American option put price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: Monte Carlo iteration
//' @return American put option price by Monte Carlo method
// [[Rcpp::export]]
```

```
double MonteCarlo_Put(double S0, double K, double T,double r, double q, double sigma, int N)
{
  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();

  std::default_random_engine gen(seed);
  double sumOpPut=0.0;
  for(int i(0); i<N; ++i)
  {
    std::normal_distribution<double> dis(0,1);
    sumOpPut += std::exp(-(r-q)*T)*std::max(K-S0*std::exp(((r-q)-sigma*sigma/2)*T + sigma*dis(gen)*std::sqrt(T)),0.0);
  }
  return sumOpPut/N;
}
```

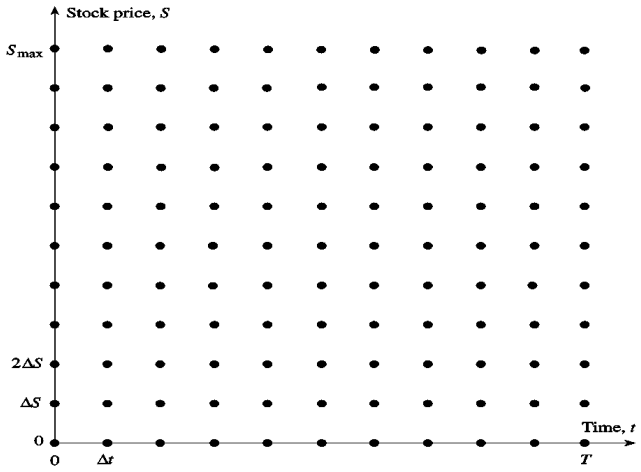
3.1 例：蒙特卡罗模拟法应用于美式看涨期权

```
#e.g. Call Option by Monte Carlo
StockPrice = 50
StrikePrice = 50
SpotTime = 0.4167
RiskFreeRate = 0.1
DividendRate = 0.02
Sigma = 0.4
N = 100
MonteCarloCallOption = MonteCarlo_Call(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N)
# MonteCarloCallOption = 5.236239
```

4. 有限差分法

有限差分法通过求解衍生产品价格所满足的微分方程达到定价的目的，在求解过程中，微分方程被转换成一组差分方程，通过迭代的方式求出差分方程的解。
期权价格满足微分方程：

$$\frac{\partial f}{\partial t}+(r-q)S\frac{\partial f}{\partial S}+\frac{1}{2}\sigma^2S^2\frac{\partial^2 f}{\partial S^2}=rf$$



算法 3.	有限差分算法
Step 1.	确定边界条件，对网格点里上、右、下三条边进行赋值。
Step 2.	将尾列的期权价格代入方程求出次尾列的期权价格。
Step 3.	重复Step2，即将后一列的期权价格代入方程求出前一列的期权价格， 直到求出所有格点的期权价格。
Step 4.	待求解的期权价格即为第一列中当前股价对应格点的值。

4.1 隐式有限差分法

隐式有限差分法的优点在于其稳定性：当ΔS和Δt趋于0时，由隐式差分方法得出的数值解总是收敛于微分方程的解。但缺点是由$f_{i+1,j}$的值计算$f_{i,j}$时，必须同时对M-1个方程求解。

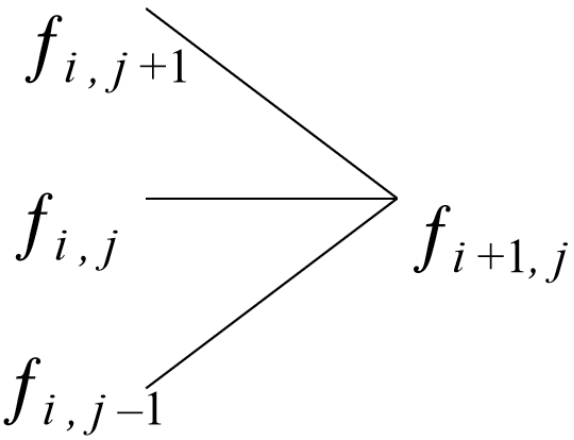
$$a_j f_{i,j-1} + b_j f_{i,j} + c_j f_{i,j+1} = f_{i+1,j}$$

$$a_j = \frac{1}{2}(r-q)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$

$$b_j = 1 + \sigma^2 j^2 \Delta t + r\Delta t$$

$$c_j = -\frac{1}{2}(r-q)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$

通过求解以上方程组得到期权的价格。



代码实现：

```cpp
//'use finite difference implicit method to compute American option put price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of finite difference
//' @param delts: time step size of finite difference
//' @return put option result matrix of finite difference implicit method
// [[Rcpp::export]]
arma::mat FD_Implicit_Put(double S0, double K, double T, double r, double q, double sigma, int N, double delts)
{
  int Smax = floor(2*S0);
  double delta_S = delts;
  int M = Smax/delta_S;
  arma::mat Option_result(M+1,N+1);

  for(int j = 0; j <= N; j++){
    Option_result(M,j) = K;
    Option_result(0,j) = 0.0;
  }
  for(int i = 0; i <= M; i++){
    Option_result(i,N) = std::max(K-(M-i)*delta_S,0.0);
  }
  arma::vec a(M-1,arma::fill::zeros);
  arma::vec b(M-1,arma::fill::zeros);
  arma::vec c(M-1,arma::fill::zeros);
  for(int k = 1; k <= M-1; k++){
    a(k-1) = 0.5*(r-q)*k*(T/N)-0.5*sigma*sigma*k*k*(T/N);
    b(k-1) = 1+sigma*sigma*k*k*(T/N)+r*(T/N);
    c(k-1) = -0.5*(r-q)*k*(T/N)-0.5*sigma*sigma*k*k*(T/N);
  }
  arma::mat coef(M+1,M+1,arma::fill::zeros);
  coef(0,0)=1;
  coef(M,M)=1;
  for(int i = 1; i < M; i++){
    coef(i,i-1) = a(i-1);
    coef(i,i) = b(i-1);
    coef(i,i+1) = c(i-1);
  }
  arma::vec y(M+1,arma::fill::zeros);
  for(int k = N; k > 0; k--){
    y = Option_result.col(k);
    Option_result.col(k-1) = arma::solve(coef,y);
  }
  return Option_result;
}
//'use finite difference implicit method to compute American option call price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of finite difference
//' @param delts: time step size of finite difference
//' @return call option result matrix of finite difference implicit method
// [[Rcpp::export]]
arma::mat FD_Implicit_Call(double S0, double K, double T, double r, double q, double sigma, int N, double delts)
{
  int Smax = floor(2*S0);
```

```
  double delta_S = delts;
  int M = Smax/delta_S;
  arma::mat Option_result(M+1,N+1);

  for(int j = 0; j <= N; j++){
    Option_result(M,j) = 0.0;
    Option_result(0,j) = K;
  }
  for(int i = 0; i <= M; i++){
    Option_result(i,N) = std::max((M-i)*delta_S-K,0.0);
  }
  arma::vec a(M-1,arma::fill::zeros);
  arma::vec b(M-1,arma::fill::zeros);
  arma::vec c(M-1,arma::fill::zeros);
  for(int k = 1; k <= M-1; k++){
    a(k-1) = 0.5*(r-q)*k*(T/N)-0.5*sigma*sigma*k*k*(T/N);
    b(k-1) = 1+sigma*sigma*k*k*(T/N)+r*(T/N);
    c(k-1) = -0.5*(r-q)*k*(T/N)-0.5*sigma*sigma*k*k*(T/N);
  }
  arma::mat coef(M+1,M+1,arma::fill::zeros);
  coef(0,0)=1;
  coef(M,M)=1;
  for(int i = 1; i < M; i++){
    coef(i,i-1) = a(i-1);
    coef(i,i) = b(i-1);
    coef(i,i+1) = c(i-1);
  }
  arma::vec y(M+1,arma::fill::zeros);
  for(int k = N; k > 0; k--){
    y = Option_result.col(k);
    Option_result.col(k-1) = arma::solve(coef,y);
  }
  return Option_result;
}
```

4.1.1 例：有限差分隐式模型应用于美式看跌期权

```
#e.g. Put option by Implicit Finite Difference
StockPrice = 50
StrikePrice = 50
SpotTime = 0.4167
RiskFreeRate = 0.1
DividendRate = 0.02
Sigma = 0.4
N = 10
delts = 5
FDImMatrixPut = FD_Implicit_Put(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N, delts)
View(FDImMatrixPut)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
2	1.387275e-07	7.529360e-08	3.843392e-08	1.822114e-08	7.887397e-09	3.042980e-09	1.009079e-09	2.710535e-10	5.274337e-11	5.615761e-12	0
3	1.298635e-05	7.534481e-06	4.125506e-06	2.106063e-06	9.858954e-07	4.133408e-07	1.497796e-07	4.424582e-08	9.539045e-09	1.135230e-09	0
4	3.510274e-04	2.184158e-04	1.287660e-04	7.108999e-05	3.616970e-05	1.657526e-05	6.608005e-06	2.163932e-06	5.218041e-07	7.020515e-08	0
5	4.422876e-03	2.956266e-03	1.880877e-03	1.126470e-03	6.254032e-04	3.148487e-04	1.389710e-04	5.084979e-05	1.385162e-05	2.133600e-06	0
6	3.233855e-02	2.321271e-02	1.594670e-02	1.037599e-02	6.302621e-03	3.499764e-03	1.720141e-03	7.088346e-04	2.204660e-04	3.944378e-05	0
7	1.561953e-01	1.201005e-01	8.894644e-02	6.284864e-02	4.180828e-02	2.567720e-02	1.412356e-02	6.606637e-03	2.374165e-03	5.020245e-04	0
8	5.436342e-01	4.455589e-01	3.543026e-01	2.710989e-01	1.972668e-01	1.341327e-01	8.288934e-02	4.436371e-02	1.867255e-02	4.770948e-03	0
9	1.452744e+00	1.259654e+00	1.068125e+00	8.799012e-01	6.974317e-01	5.241172e-01	3.645887e-01	2.249377e-01	1.126737e-01	3.584465e-02	0
10	3.133401e+00	2.846042e+00	2.547937e+00	2.238203e+00	1.916135e+00	1.581602e+00	1.235907e+00	8.836452e-01	5.366119e-01	2.220471e-01	0
11	5.691152e+00	5.352951e+00	4.995927e+00	4.616218e+00	4.208277e+00	3.763761e+00	3.269483e+00	2.703497e+00	2.027418e+00	1.170946e+00	0
12	9.039274e+00	8.704817e+00	8.355046e+00	7.988164e+00	7.602162e+00	7.195014e+00	6.765281e+00	6.313714e+00	5.847248e+00	5.388704e+00	5
13	1.297869e+01	1.268067e+01	1.237463e+01	1.206118e+01	1.174154e+01	1.141804e+01	1.109477e+01	1.077854e+01	1.048007e+01	1.021470e+01	10
14	1.730711e+01	1.705613e+01	1.680264e+01	1.654809e+01	1.629455e+01	1.604487e+01	1.580278e+01	1.557290e+01	1.536028e+01	1.516912e+01	15
15	2.186716e+01	2.166227e+01	2.145742e+01	2.125386e+01	2.105313e+01	2.085703e+01	2.066752e+01	2.048645e+01	2.031514e+01	2.015366e+01	20
16	2.655014e+01	2.638684e+01	2.622410e+01	2.606258e+01	2.590295e+01	2.574581e+01	2.559159e+01	2.544043e+01	2.529202e+01	2.514559e+01	25
17	3.128444e+01	3.115844e+01	3.103264e+01	3.090715e+01	3.078189e+01	3.065658e+01	3.053061e+01	3.040305e+01	3.027271e+01	3.013850e+01	30
18	3.602311e+01	3.593132e+01	3.583929e+01	3.574676e+01	3.565327e+01	3.555801e+01	3.545979e+01	3.535695e+01	3.524739e+01	3.512888e+01	35
19	4.073490e+01	4.067524e+01	4.061515e+01	4.055430e+01	4.049214e+01	4.042780e+01	4.036000e+01	4.028680e+01	4.020540e+01	4.011165e+01	40
20	4.539866e+01	4.536954e+01	4.534013e+01	4.531020e+01	4.527940e+01	4.524721e+01	4.521276e+01	4.517473e+01	4.513078e+01	4.507622e+01	45
21	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	5.000000e+01	50

4.2 显式有限差分法

显示有限差分法的优点是便于求解，缺点是稳定性不如隐式差分。

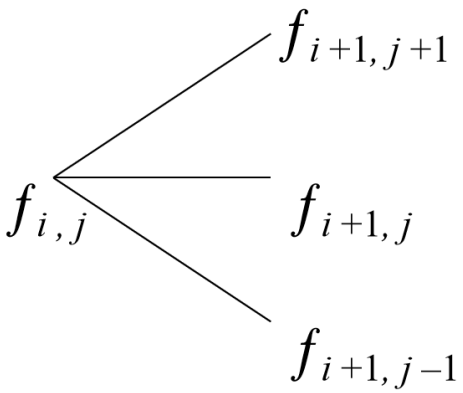
$$a_j^* f_{i+1,j-1} + b_j^* f_{i+1,j} + c_j^* f_{i+1,j+1} = f_{i,j}$$

$$a_j^* = \frac{1}{1+r\Delta t}(-\frac{1}{2}(r-q)j\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t)$$

$$b_j^* = \frac{1}{1+r\Delta t}(1-\sigma^2 j^2 \Delta t)$$

$$c_j^* = \frac{1}{1+r\Delta t}(\frac{1}{2}(r-q)j\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t)$$

通过求解以上方程得到期权的价格。



代码实现：

```
//'use finite difference explicit method to compute American option put price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of finite difference
//' @param delts: time step size of finite difference
//' @return put option result matrix of finite difference explicit method
// [[Rcpp::export]]
arma::mat FD_Explicit_Put(double S0, double K, double T, double r, double q, double sigma, int N, double delts)
{
  int Smax = floor(2*S0);
  double delta_S = delts;
  int M=Smax/delta_S;
  arma::mat Option_result(M+1,N+1);
  for(int j = 0; j <= N; j++){
    Option_result(M,j) = K;
    Option_result(0,j) = 0.0;
  }
  for(int i = 1; i <= M-1; i++){
    Option_result(i,N) = std::max(K-(M-i)*delta_S ,0.0);
  }
  for(int j = N-1; j >=0 ; j--){
    for(int i = 1; i <= M -1; i++){
      double a = 1/(1+r*(T/N))*(-0.5*(r-q)*(M-i)*(T/N) + 0.5*(T/N)*sigma*sigma*(M-i)*(M-i));
      double b = 1/(1+r*(T/N))*(1-(T/N)*sigma*sigma*(M-i)*(M-i));
      double c = 1/(1+r*(T/N))*((T/N)*0.5*(r-q)*(M-i) + 0.5*(T/N)*sigma*sigma*(M-i)*(M-i));
      Option_result(i,j)= a*Option_result(i+1,j+1) + b*Option_result(i,j+1) + c*Option_result(i-1,j+1);

    }
  }
  return Option_result;
}
```

```
//'use finite difference explicit method to compute American option put price
//'
//' @param S0: stock price
//' @param K: strike price
//' @param T: spot time
//' @param r: risk-free interest rate
//' @param q: dividend rate
//' @param sigma: stock volatility
//' @param N: time step of finite difference
//' @param delts: time step size of finite difference
//' @return call option result matrix of finite difference explicit method
// [[Rcpp::export]]
arma::mat FD_Explicit_Call(double S0, double K, double T, double r, double q, double sigma, int N, double delts){
  int Smax = floor(2*S0);
  double delta_S = delts;
  int M=Smax/delta_S;
  arma::mat Option_result(M+1,N+1);
  for(int j = 0; j <= N; j++){
    Option_result(M,j) = 0.0;
    Option_result(0,j) = Smax-K;
  }
  for(int i = 1; i <= M-1; i++){
    Option_result(i,N) = std::max((M-i)*delta_S-K ,0.0);
  }
  for(int j = N-1; j >=0 ; j--){
    for(int i = 1; i <= M -1; i++){
      double a = 1/(1+r*(T/N))*(-0.5*(r-q)*(M-i)*(T/N) + 0.5*(T/N)*sigma*sigma*(M-i)*(M-i));
      double b = 1/(1+r*(T/N))*(1-(T/N)*sigma*sigma*(M-i)*(M-i));
      double c = 1/(1+r*(T/N))*((T/N)*0.5*(r-q)*(M-i) + 0.5*(T/N)*sigma*sigma*(M-i)*(M-i));
      Option_result(i,j)= a*Option_result(i+1,j+1) + b*Option_result(i,j+1) + c*Option_result(i-1,j+1);
    }
  }
  return Option_result;
}
```

4.2.1 例：有限差分显式模型应用于美式看跌期权

```
#e.g. Put option by Explicit Finite Difference
StockPrice = 50
StrikePrice = 50
SpotTime = 0.4167
RiskFreeRate = 0.1
DividendRate = 0.02
Sigma = 0.4
N = 10
delts = 5
FDExMatrixPut = FD_Explicit_Put(StockPrice, StrikePrice, SpotTime, RiskFreeRate, DividendRate, Sigma, N, delts)
View(FDExMatrixPut)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	0.00000000	0.00000000	0.000000000	0.00000000	0.00000000	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
2	0.06333226	0.00000000	0.000000000	0.00000000	0.00000000	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
3	-0.11841961	0.05427403	0.000000000	0.00000000	0.00000000	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
4	0.30877249	-0.05327746	0.051900793	0.00000000	0.00000000	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
5	-0.15211353	0.22121699	-0.005771439	0.05573572	0.00000000	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
6	0.48642125	0.06108068	0.208511361	0.04064033	0.06769777	0.00000000	0.0000000	0.0000000	0.0000000	0.000000	0
7	0.32890525	0.47932572	0.234586925	0.25722995	0.10298030	0.09375783	0.0000000	0.0000000	0.0000000	0.000000	0
8	0.94118785	0.69198362	0.654207365	0.45547631	0.38739873	0.20977944	0.1494301	0.0000000	0.0000000	0.000000	0
9	1.49830511	1.38892660	1.182455593	1.04147350	0.82374594	0.66594908	0.4238336	0.2769984	0.0000000	0.000000	0
10	2.59244868	2.39780526	2.218033415	1.99559591	1.78075293	1.51225633	1.2544263	0.9011040	0.6046314	0.000000	0
11	4.20461242	4.03635128	3.845090483	3.64230535	3.40848987	3.15577447	2.8501244	2.5158710	2.0465961	1.576889	0
12	6.55844834	6.43400813	6.300459763	6.15326880	5.99379822	5.81512788	5.6204477	5.3935676	5.1552523	4.829862	5
13	9.74416385	9.70886823	9.673504059	9.63976169	9.60912889	9.58589624	9.5751005	9.5913088	9.6522580	9.825712	10
14	13.73275722	13.79984374	13.875362382	13.96134998	14.06048377	14.17577786	14.3107210	14.4671948	14.6439620	14.821563	15
15	18.31273653	18.45518031	18.604891738	18.76208342	18.92663899	19.09795209	19.2746792	19.4547560	19.6356660	19.817413	20
16	23.18169409	23.35576607	23.532491290	23.71150722	23.89239466	24.07471799	24.2581010	24.4423172	24.6273700	24.813263	25
17	28.12943134	28.31246871	28.496507725	28.68147545	28.86731798	29.05400386	29.2415228	29.4298784	29.6190741	29.809113	30
18	33.08751996	33.27501285	33.463335930	33.65248869	33.84247254	34.03329015	34.2249447	34.4174396	34.6107781	34.804964	35
19	38.04653079	38.23804571	38.430410081	38.62362487	38.81769096	39.01260912	39.2083800	39.4050042	39.6024821	39.800814	40
20	43.01970464	43.21252880	43.406487716	43.60158949	43.79784231	43.99525443	44.1938342	44.3935901	44.5945306	44.796664	45
21	50.00000000	50.00000000	50.000000000	50.00000000	50.00000000	50.00000000	50.0000000	50.0000000	50.0000000	50.000000	50

Github网址：

王子豪：https://github.com/ZihaoWang22/R-package-OptionPricing

甘剑煌：https://github.com/JhuangGan/R-package-OptionPricing

林宇波：https://github.com/HHHyp-Lin/R-Package-OptionPricing