**Computer Assignment #4**

**Numerical Methods for Option Pricing**

Group 4

Beier Chen

Chunlei Meng

Yibing Xie

Yi Zhu

# Executive Summary

This report is intended to implement numerical methods to price options and compare the model output with the market data. Specifically, we use binomial lattice method and explicit finite difference method to price an American put option on a dividend paying stock, and then use binomial lattice method and explicit finite difference method as well as Black-Scholes-Merton model to price an American call option on a non-dividend paying stock. We find that the outputs of the numerical methods are very close the market price, and the result confirms that numerical methods like binomial lattice method and finite difference method work well in the market.

# Introduction

The Black-Scholes-Merton formula is the foundation for pricing the European option, but sometimes the analytic result of the Black-Scholes-Merton formula cannot be implemented in practice. Some options have complicated payoffs and do not have a closed form solution, like the American and Asian options. In such situation, the numerical methods can be very helpful to handle the derivatives valuation problems. In this report, we will introduce two numerical methods, the binomial lattice method and the finite difference method.

The binomial lattice method is a widely used method because of its ease of implementation and flexibility in applying to all kinds of options. This method involves building a lattice to value the asset price, assuming that for each step, the asset increase in value by a constant factor, *u,* or decrease in value by a constant factor, *d.* The advantage of the binomial lattice method is that it is easy to use and extend to value the options. The disadvantage is that it is not adapted to some exotic options, like the barrier options, and that the exponential growth may cause problems.

The finite difference method is very similar to the binomial lattice method. This method is used to price options by approximating the differential equation of the option price evolvement with the difference equations. It can be applied to price American options and many exotic options. The advantage of this method is that we can find the option price at every point in the domain, so we don’t have to solve the problem over and over to get different price. The finite difference method includes explicit difference methods, the implicit methods and the Crank-Nicolson method. In this report we will focus on the explicit finite difference method.

# Data

The data used are collected from Bloomberg at their closing price on 12/03/2014. The call option selected is Verisign, Inc. (VRSN), which pays no dividend, expiring on 1/15/2016. And the put option selected is International Business Machines (IBM) which pays dividend at annual rate of 3.73%, expiring on 1/15/2016. The risk-free rate is one-month LIBOR, which is 0.24% annually.

# Methodology

***Binomial Lattice Method***

The binomial tree valuation model price a derivative by dividing the life of the option into a large number of small time intervals of length Δt*.* It assumes that at each node before maturity the price of the underlying asset moves from its current value to one of two new values, and. In general, u > 1 and d < 1. Therefore, the underlying asset price moves from to is an up movement and from to a down movement. We denote the probability of an up movement and the probability of a down movement by p and 1p, respectively.

To construct a binomial tree, we need to determine the parameter *u* and *d*, and the corresponding risk neutral probabilities. In this computer assignment we apply Cox, Ross, and Rubinstein’s model.

where is the volatility of the underlying asset price, r is the continuous compounding risk-free interest rate and q is the continuous compounding dividend yield.

Based on these assumptions, the underlying asset price at each node on the binomial tree is known:

S (i, j) = S for j =0,…, i

where S (i, j) denotes the stock price at period if j out of i times the stock price goes up.

After constructing the binomial tree, we can apply it to price derivatives by working backward through the tree. One nontrivial advantage of binomial lattice method is that it can price path-dependent derivatives.

***Finite Difference Method***

Finite difference methods price a derivative by approximating the differentials in PDE using differences and then solve the approximated difference equations iteratively.

Recall the partial differential equation for the value of a derivative,

We dividend the maturity and the underlying asset price into N and M equally spaced intervals and then construct a grid. Then we define the values of f(t, S) at three edges of the grid and solve for f(t, S) recursively approximating where

Substituting the differentials with differences, we get

The equation above refers to what we call implicit finite difference method. The implicit finite difference method has the advantage of being very robust.

However, one problem with this method is that when solving backward we need to solve M1 simultenous equations with each contains more than one unknowns. One way to get rid of the messy calculation is to apply explicit finite difference method where and at point (i, j) on the grid are assumed to be the same as at point (i+1, j) and the difference equation becomes

# Results and Analysis

The basic information of the stocks and the options is shown in the table 1 below.

Table 1: Basic information

|  |  |  |
| --- | --- | --- |
| **Ticker** | **VRSN** | **IBM** |
| Quote Date | 2015/12/3 | 2015/12/3 |
| Last Price | 92.04 | 140.815 |
| Dividend | 0 | 0.0373 |
| Option Type | American Call | American Put |
| Strike Price | 90 | 140 |
| Maturity | 2016/1/15 | 2016/1/15 |
| Price | 4.4500 | 2.8050 |
| Implied Volatility | 28.97% | 16.81% |

Table 2: Comparison among call option prices on VRSN

|  |  |  |  |
| --- | --- | --- | --- |
| VRSN | Price | Delta | Gamma |
| Bloomberg | 4.4500 | 0.6106 | 0.0386 |
| Binomial n = 30 | 4.7189 | 0.6104 | 0.0428 |
| Explicit FDM | 4.6618 | 0.6074 | 0.0418 |
| BSM | 4.6936 | 0.6103 | 0.0423 |

Table 3: Comparison among put option prices on IBM

|  |  |  |  |
| --- | --- | --- | --- |
| IBM | Price | Delta | Gamma |
| Bloomberg | 2.8050 | -0.4479 | 0.0694 |
| Binomial n = 30 | 3.0786 | -0.4738 | 0.0500 |
| Explicit FDM | 3.3097 | -0.5149 | 0.0473 |

Table 2 and Table 3 show the comparison among the prices calculated using the two numerical methods and the market data. We find that the option prices, Deltas and Gammas we calculated using the numerical methods are very close the market price, which proves that binomial lattice method and finite difference method work well in the market. We also find that the two numerical methods give better approximation of Delta than Gamma, which is not surprising because we use linear approximation twice when we calculate Gamma.

Also, as shown in the Table 4 and Table 5 in the appendices, the IBM put option will not be exercised before maturity under binomial method. For the finite difference method, unless the stock price will be as low as $7, which is almost impossible considering its current price $140.815, the put option will not be exercised earlier.

It can be justified by the fact that the dividend yield, which is 3.73% is much higher than risk-free rate, which is 0.24%, therefore the holders of the put option tend to hold it to maturity to enjoy the benefit of the dividend paid.

# Conclusions

Given American options allows the holders to exercise the options at any time before maturity, there is no closed form formula to price the American options. In this report, we use two numerical methods, including the binomial lattice method and the finite difference method to price American options. We then compare the calculated option prices, Deltas and Gammas with the market data. The comparison shows that the result of the numerical method is very close to the real data, and this confirms the feasibility and usefulness of the numerical methods.

# Appendix

Table 4. Snippet of Binomial Put

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binomial Tree: |  |  |  |  |
| 0 0 | Option Value: | **3.07857** | Early Exercise: | No |
| 1 0 | Option Value: | 3.76063 | Early Exercise: | No |
| 1 1 | Option Value: | 2.37148 | Early Exercise: | No |
| 2 0 | Option Value: | 4.53932 | Early Exercise: | No |
| 2 1 | Option Value: | 2.95337 | Early Exercise: | No |
| 2 2 | Option Value: | 1.76823 | Early Exercise: | No |
| 3 0 | Option Value: | 5.41378 | Early Exercise: | No |
| 3 1 | Option Value: | 3.63278 | Early Exercise: | No |
| 3 2 | Option Value: | 2.24901 | Early Exercise: | No |
| 3 3 | Option Value: | 1.26978 | Early Exercise: | No |
| 4 0 | Option Value: | 6.37957 | Early Exercise: | No |
| 4 1 | Option Value: | 4.41257 | Early Exercise: | No |
| 4 2 | Option Value: | 2.82436 | Early Exercise: | No |
| 4 3 | Option Value: | 1.65254 | Early Exercise: | No |
| 4 4 | Option Value: | 0.87297 | Early Exercise: | No |
| 5 0 | Option Value: | 7.42866 | Early Exercise: | No |
| 5 1 | Option Value: | 5.292 | Early Exercise: | No |
| 5 2 | Option Value: | 3.50087 | Early Exercise: | No |
| 5 3 | Option Value: | 2.12302 | Early Exercise: | No |
| 5 4 | Option Value: | 1.16478 | Early Exercise: | No |
| 5 5 | Option Value: | 0.570437 | Early Exercise: | No |
| 6 0 | Option Value: | 8.54988 | Early Exercise: | No |
| 6 1 | Option Value: | 6.26632 | Early Exercise: | No |
| 6 2 | Option Value: | 4.28194 | Early Exercise: | No |
| 6 3 | Option Value: | 2.69112 | Early Exercise: | No |
| 6 4 | Option Value: | 1.53405 | Early Exercise: | No |
| 6 5 | Option Value: | 0.781948 | Early Exercise: | No |
| 6 6 | Option Value: | 0.351155 | Early Exercise: | No |
| 7 0 | Option Value: | 9.72975 | Early Exercise: | No |
| 7 1 | Option Value: | 7.32676 | Early Exercise: | No |
| 7 2 | Option Value: | 5.16697 | Early Exercise: | No |
| 7 3 | Option Value: | 3.36442 | Early Exercise: | No |
| 7 4 | Option Value: | 1.99311 | Early Exercise: | No |
| 7 5 | Option Value: | 1.05814 | Early Exercise: | No |
| 7 6 | Option Value: | 0.495613 | Early Exercise: | No |
| 7 7 | Option Value: | 0.201388 | Early Exercise: | No |

Table 5. Snippet of explicit FDM put

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IBM | American | Put |  |  |  |  |
|  | Time to Maturity | 0.115079 |  |  | 0.111243 |  |
| Stock Price |  |  |  |  |  |  |
| 0 |  | 140 |  |  | 140 |  |
| 2.34692 |  | 137.653 | Yes |  | 137.653 | Yes |
| 4.69383 |  | 135.306 | Yes |  | 135.306 | Yes |
| 7.04075 |  | 132.959 | Yes |  | 132.959 | Yes |
| 9.38767 |  | 130.614 | No |  | 130.614 | No |
| 11.7346 |  | 128.277 | No |  | 128.277 | No |
| 14.0815 |  | 125.94 | No |  | 125.939 | No |
| … … |  | … … |  |  | … … |  |
| 131.427 |  | 9.55298 | No |  | 9.51212 | No |
| 133.774 |  | 7.59433 | No |  | 7.54454 | No |
| 136.121 |  | 5.83672 | No |  | 5.77925 | No |
| 138.468 |  | 4.32153 | No |  | 4.2595 | No |
| **140.815** |  | **3.07301** | No |  | 3.01076 | No |
| 143.162 |  | 2.09339 | No |  | 2.03542 | No |
| 145.509 |  | 1.36341 | No |  | 1.31328 | No |
| 147.856 |  | 0.847662 | No |  | 0.807364 | No |
| 150.203 |  | 0.502506 | No |  | 0.472337 | No |
| 152.55 |  | 0.283799 | No |  | 0.262721 | No |
| 154.897 |  | 0.152599 | No |  | 0.138829 | No |
| 157.243 |  | 0.0780785 | No |  | 0.0696536 | No |
| 159.59 |  | 0.0379966 | No |  | 0.0331611 | No |
| 161.937 |  | 0.0175781 | No |  | 0.0149715 | No |
| 164.284 |  | 0.00772611 | No |  | 0.00640528 | No |

## **C++ code**

Explicit Finite Difference Method

#**include** <algorithm>

#**include** <fstream>

#**include** "ExplicitFDM.h"

**using** namespace std;

ExplicitFDM::ExplicitFDM(

string type,

string ticker,

double SMAX,

double K,

double T,

double r,

double sig,

double q,

size\_t M,

size\_t N) :

type\_(type), ticker\_(ticker), max\_price\_(SMAX), strike\_price\_(K),

time\_to\_maturity\_(T), risk\_free\_rate\_(r), volatility\_(sig), yield\_(q),

num\_of\_price\_intervals\_(M), num\_of\_periods\_(N)

{

delta\_t\_ = T / N;

delta\_s\_ = SMAX / M;

grid\_.**resize**(num\_of\_periods\_ + 1);

for (auto& x : grid\_) {

x.**resize**(num\_of\_price\_intervals\_ + 1);

}

}

ExplicitFDM::~ExplicitFDM() { }

void ExplicitFDM::set\_boundary() {

if (type\_ == "Call") {

for (size\_t i = 0; i <= num\_of\_periods\_; ++i) {

get<0>(grid\_[i][0]) = 0.0;

get<0>(grid\_[i][num\_of\_price\_intervals\_]) = max\_price\_;

}

for (size\_t j = 0; j <= num\_of\_price\_intervals\_; ++j) {

get<0>(grid\_[num\_of\_periods\_][j]) =

max(0.0, j \* delta\_s\_ - strike\_price\_);

}

}

else if (type\_ == "Put") {

for (size\_t i = 0; i <= num\_of\_periods\_; ++i) {

if (style\_ == "American") {

get<0>(grid\_[i][0]) = strike\_price\_;

}

else if (style\_ == "European") {

get<0>(grid\_[i][0]) = strike\_price\_ \*

exp(-risk\_free\_rate\_ \* i \* delta\_t\_);

}

get<0>(grid\_[i][num\_of\_price\_intervals\_]) = 0.0;

}

for (size\_t j = 0; j <= num\_of\_price\_intervals\_; ++j) {

get<0>(grid\_[num\_of\_periods\_][j]) =

max(0.0, strike\_price\_ - j \* delta\_s\_);

}

}

}

void ExplicitFDM::output\_results() {

string file = "Explicit" + style\_ + type\_ + ticker\_ + "**.csv**";

ofstream **output**(file);

output << ticker\_ << "," << style\_ << "," << type\_ << ",\n";

output << ",Time to Maturity,";

for (size\_t i = 0; i <= num\_of\_periods\_; ++i) {

output << (num\_of\_periods\_ - i) \* delta\_t\_ <<

",,,";

}

output << "\nStock Price";

for (size\_t j = 0; j <= num\_of\_price\_intervals\_; ++j) {

output << "\n" << j \* delta\_s\_ << ",,";

for (size\_t i = 0; i <= num\_of\_periods\_; ++i) {

output << get<0>(grid\_[i][j]) << "," << get<1>(grid\_[i][j]) << ",,";

}

}

for (size\_t j = 0; j <= num\_of\_price\_intervals\_; ++j) {

if (j < 100) {

printf("Stock Price := %.2f Option Price := %.2f\n", j \* delta\_s\_,

get<0>(grid\_[0][j]));

}

else

break;

}

output.**close**();

}

// European Method

EuropeanExplicitFDM::EuropeanExplicitFDM(

string type,

string ticker,

double SMAX,

double K,

double T,

double r,

double sig,

double q, // Dividend yield.

size\_t M, // Number of intervals in price.

size\_t N // Number of periods in time.

) : ExplicitFDM(type, ticker, SMAX, K, T, r, sig, q, M, N) {

style\_ = "European";

}

EuropeanExplicitFDM::~EuropeanExplicitFDM() { }

void EuropeanExplicitFDM::do\_exiplicit\_fdm() {

set\_boundary();

for (int i = num\_of\_periods\_ - 1; i >= 0; --i) {

for (int j = 1; j < num\_of\_price\_intervals\_; ++j) {

// Initial coefficients a, b, c.

double a = (-0.5 \* (risk\_free\_rate\_ - yield\_) \* j \* delta\_t\_ +

0.5 \* volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

double b = (1 - volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

double c = (0.5 \* (risk\_free\_rate\_ - yield\_) \* j \* delta\_t\_ +

0.5 \* volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

get<0>(grid\_[i][j]) = a \* get<0>(grid\_[i + 1][j - 1]) +

b \* get<0>(grid\_[i + 1][j]) + c \* get<0>(grid\_[i + 1][j + 1]);

get<1>(grid\_[i][j]) = "No";

}

}

}

// American

AmericanExplicitFDM::AmericanExplicitFDM(

string type,

string ticker,

double SMAX,

double K,

double T,

double r,

double sig,

double q, // Dividend yield.

size\_t M, // Number of intervals in price.

size\_t N // Number of periods in time.

) : ExplicitFDM(type, ticker, SMAX, K, T, r, sig, q, M, N) {

style\_ = "American";

}

AmericanExplicitFDM::~AmericanExplicitFDM() { }

void AmericanExplicitFDM::do\_exiplicit\_fdm() {

set\_boundary();

for (int i = num\_of\_periods\_ - 1; i >= 0; --i) {

for (int j = 1; j < num\_of\_price\_intervals\_; ++j) {

// Initial coefficients a, b, c.

double a = (-0.5 \* (risk\_free\_rate\_ - yield\_) \* j \* delta\_t\_ +

0.5 \* volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

double b = (1 - volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

double c = (0.5 \* (risk\_free\_rate\_ - yield\_) \* j \* delta\_t\_ +

0.5 \* volatility\_ \* volatility\_ \* j \* j \* delta\_t\_) /

(1 + risk\_free\_rate\_ \* delta\_t\_);

double wait = a \* get<0>(grid\_[i + 1][j - 1]) +

b \* get<0>(grid\_[i + 1][j]) + c \* get<0>(grid\_[i + 1][j + 1]);

double early = strike\_price\_ - j \* delta\_s\_;

get<0>(grid\_[i][j]) = max(wait, early);

get<1>(grid\_[i][j]) = early > wait ? "Yes" : "No";

}

}

}

Binomial Lattice

#**include** <algorithm>

#**include** <fstream>

#**include** "BinomialMethod.h"

**using** namespace std;

BinomialMethod::BinomialMethod(

string type,

string ticker,

size\_t num\_of\_periods,

double S,

double K,

double discount,

double up,

double down,

double risk\_neutral\_q

) :

type\_(type),

ticker\_(ticker),

num\_of\_periods\_(num\_of\_periods),

spot\_(S),

strike\_price\_(K),

discount\_(discount),

up\_(up),

down\_(down),

risk\_neutral\_q\_(risk\_neutral\_q),

lattice\_(Lattice<double, 2>(num\_of\_periods)),

payoff\_ptr\_(nullptr)

{

if (type == "Call") {

payoff\_ptr\_.**reset**(new PayOffCall(K));

}

else if (type == "Put") {

payoff\_ptr\_.**reset**(new PayOffPut(K));

}

}

BinomialMethod::~BinomialMethod() { }

void BinomialMethod::fill\_lattice\_forward() {

for (size\_t i = 0; i <= num\_of\_periods\_; ++i) {

get<0>(lattice\_[i][0]) = spot\_ \* pow(down\_, i);

if (i > 0) {

size\_t length = lattice\_[i].**size**();

for (int j = 1; j < length; ++j) {

// Update the binomial tree

// =================================

get<0>(lattice\_[i][j]) = get<0>(lattice\_[i][j - 1])

/ down\_ \* up\_;

}

}

}

}

void BinomialMethod::output\_results() const {

// Write into csv file.

string file\_name = "Binomial" + ticker\_ + style\_ + type\_ + "**.csv**";

ofstream **output**(file\_name);

double option\_price = std::get<1>(lattice\_[0][0]);

printf("Option Price := %.2f\n", option\_price);

output << "Ticker:," << ticker\_ << ",\n";

output << "Option Type:," << type\_ << ",\n";

output << "Stock Price:," << spot\_ << ",\n";

output << "Strike Price:," << strike\_price\_ << ",\n";

output << "Option Price:," << option\_price << ",\n";

if (up\_ == 1 / down\_) {

double delta = (get<1>(lattice\_[1][1]) - get<1>(lattice\_[1][0])) /

(get<0>(lattice\_[1][1]) - get<0>(lattice\_[1][0]));

double gamma = ((get<1>(lattice\_[2][2]) - get<1>(lattice\_[2][1])) /

(get<0>(lattice\_[2][2]) - get<0>(lattice\_[2][1])) -

(get<1>(lattice\_[2][1]) - get<1>(lattice\_[2][0])) /

(get<0>(lattice\_[2][1]) - get<0>(lattice\_[2][0]))) /

(0.5 \* (get<0>(lattice\_[2][2]) - get<0>(lattice\_[2][0])));

printf("Delta := %.4f\nGamma := %.4f\n", delta, gamma);

output << "Delta:," << delta << ",\n";

output << "Gamma:," << gamma << ",\n";

}

if (num\_of\_periods\_ <= 50) {

output << "Binomial Tree:" << ",\n";

for (size\_t i = 0; i < num\_of\_periods\_; ++i) {

for (size\_t j = 0; j <= i; ++j) {

output << i << " " << j << ",Option Value:," <<

get<1>(lattice\_[i][j]) << ",Early Exercise:," <<

get<2>(lattice\_[i][j]) << ",\n";

}

}

}

output.**close**();

}

// Definition of European

// ======================

EuropeanBinomialMethod::EuropeanBinomialMethod(

string type,

string ticker,

size\_t num\_of\_periods,

double S,

double K,

double r,

double up,

double down,

double risk\_neutral\_q) :

BinomialMethod(type, ticker, num\_of\_periods, S, K, r, up,

down, risk\_neutral\_q)

{

style\_ = "European";

}

EuropeanBinomialMethod::~EuropeanBinomialMethod() { }

void EuropeanBinomialMethod::do\_binomial\_pricing() {

fill\_lattice\_forward(); // Update the selected lattice

for (int i = lattice\_.**size**() - 1; i >= 0; --i) {

size\_t length = lattice\_[i].**size**();

if (i == lattice\_.**size**() - 1)

for (size\_t j = 0; j < length; ++j){

get<1>(lattice\_[i][j]) = payoff\_ptr\_->**calculate\_payoff**(

get<0>(lattice\_[i][j]));

}

else

for (size\_t j = 0; j < length; ++j) {

get<1>(lattice\_[i][j]) =

(risk\_neutral\_q\_ \* get<1>(lattice\_[i + 1][j + 1]) +

(1 - risk\_neutral\_q\_) \* get<1>(lattice\_[i + 1][j])) \*

discount\_;

get<2>(lattice\_[i][j]) = "No";

}

}

}

// Definition of American

// ======================

AmericanBinomialMethod::AmericanBinomialMethod(

string type,

string ticker,

size\_t num\_of\_periods,

double S,

double K,

double r,

double up,

double down,

double risk\_neutral\_q) :

BinomialMethod(type, ticker, num\_of\_periods, S, K, r, up,

down, risk\_neutral\_q)

{

style\_ = "American";

}

AmericanBinomialMethod::~AmericanBinomialMethod() { }

void AmericanBinomialMethod::do\_binomial\_pricing()

{

fill\_lattice\_forward(); // Update the selected lattice

for (int i = lattice\_.**size**() - 1; i >= 0; --i) {

size\_t length = lattice\_[i].**size**();

if (i == lattice\_.**size**() - 1)

for (size\_t j = 0; j < length; ++j){

get<1>(lattice\_[i][j]) = payoff\_ptr\_->**calculate\_payoff**(

get<0>(lattice\_[i][j]));

}

else

for (size\_t j = 0; j < length; ++j) {

double wait =

(risk\_neutral\_q\_ \* get<1>(lattice\_[i + 1][j + 1]) +

(1 - risk\_neutral\_q\_) \* get<1>(lattice\_[i + 1][j])) \*

discount\_;

double strike\_now = payoff\_ptr\_->**calculate\_payoff**(

get<0>(lattice\_[i][j]));

get<1>(lattice\_[i][j]) = (max)(strike\_now, wait);

get<2>(lattice\_[i][j]) = strike\_now > wait ? "Yes" : "No";

}

}

}