COMPUTATIONAL FINANCE

Lecture 0: Review of Binomial Option Pricing
A Simple Binomial Option Pricing Program

Philip H. Dybvig
Washington University
Saint Louis, Missouri

Copyright © Philip H. Dybvig 1995

The Binomial Option Pricing Model

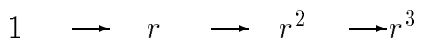
The option pricing model of Black and Scholes revolutionized a literature previously characterized by clever but unreliable rules of thumb. The Black-Scholes model uses continuous-time stochastic process methods that interfere with understanding the simple intuition underlying these models. We will use instead the binomial option pricing model of Cox, Ross, and Rubinstein, which captures all of the economics of the continuous time model but is simple to understand and program. For option pricing problems not appropriately handled by Black-Scholes, some variant of the binomial model is the usual choice of practitioners since it is relatively easy to program, fast, and flexible.

Cox, John C., Stephen A. Ross, and Mark Rubinstein (1979) "Option Pricing: A Simplified Approach" *Journal of Financial Economics* **7**, 229–263

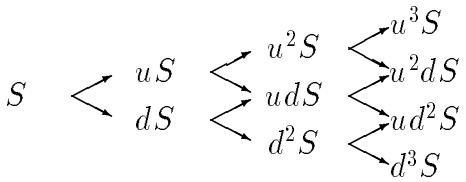
Black, Fischer, and Myron Scholes (1973) "The Pricing of Options and Corporate Liabilities" *Journal of Political Economy* **81**, 637–654

Binomial Process (3 periods)

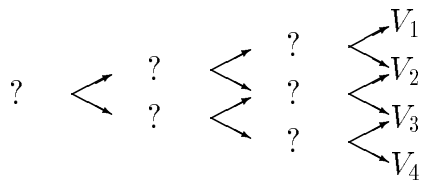
Riskless bond:



Stock ($u > r > d$):



Derivative security (option or whatever):



What is the price of the derivative security?

A Simple Option Pricing Problem in One Period

Riskless bond (interest rate is 0):

$$100 \longrightarrow 100$$

Stock:

$$50 \begin{cases} \nearrow 100 \\ \searrow 25 \end{cases}$$

Derivative security (on-the-money call option):

$$? \begin{cases} \nearrow 50 \\ \searrow 0 \end{cases}$$

To duplicate the call option with α_S shares of stock and α_B bonds:

$$\begin{aligned} 50 &= 100\alpha_S + 100\alpha_B \\ 0 &= 25\alpha_S + 100\alpha_B \end{aligned}$$

Therefore $\alpha_S = 2/3$, $\alpha_B = -1/6$, and the duplicating portfolio is worth $50\alpha_S + 100\alpha_B = 100/6 = 16\ 2/3$. By absence of arbitrage, this must also be the price of the call option.

In-class Exercise: One-period Contingent Claim Valuation

Compute the duplicating portfolio and the price of the general derivative security below. Assume $u > r > d > 0$.

Riskless bond:

$$1 \longrightarrow r$$

Stock:

$$S \begin{cases} \nearrow uS \\ \searrow dS \end{cases}$$

Derivative security:

$$??? \begin{cases} \nearrow V_u \\ \searrow V_d \end{cases}$$

Multi-period Valuation and Artificial Probabilities

In general, exactly the same valuation procedure is used many times, taking as given the value at maturity and solving back one period of time until the beginning. This valuation can be viewed in terms of state prices p_u and p_d or risk-neutral probabilities π_u^* and π_d^*, which give the same answer (which is the only one consistent with no arbitrage):

$$Value = p_u V_u + p_d V_d = r^{-1}(\pi_u^* V_u + \pi_d^* V_d)$$

where

$$p_u = r^{-1}\frac{r-d}{u-d} \quad p_d = r^{-1}\frac{u-r}{u-d}$$

and

$$\pi_u = \frac{r-d}{u-d} \quad \pi_d = \frac{u-r}{u-d}.$$

In-class Exercise: Artificial Probabilities

In the binomial model (with parameters u, d, and r), show that the stock and the bond have the same one-period expected return computed using the artificial probabilities.

Consider the binomial model with $u = 2$, $d = 1/2$, and $r = 1$. What are the risk-neutral probabilities? Assuming the stock price is initially $\$100$, what is the price of a call option with a $\$90$ strike price maturing in two periods? Do the valuation two ways, using the risk neutral probabilities to solve backwards through the tree, and directly using the two-period risk neutral probabilities.

Some Orders of Magnitude

- Expected Excess Returns
  - Common stock indices: 8–10% per year or 8–$10\%/250 \approx 3$ or 4 basis points daily
  - Individual common stocks: 50%–150% of the index expected excess return.
- Standard Deviation
  - Common stock indices: 20–25% per year or 20–$25\%/\sqrt{250} \approx 1$–$1.5\%$ per day
  - Individual common stocks: 35%–40% per year

Theoretical observation: for the usual case, standard deviations over short periods are almost exactly the same in actual probabilities as in risk-neutral probabilities.

Binomial Parameters in Practice

Most texts seem to have unreasonably complicated expressions for u, d, and r in binomial models. From the theory, we know that a good choice is

$$u = 1 + r * \Delta t + \sigma * \sqrt{\Delta t}$$

$$d = 1 + r * \Delta t - \sigma * \sqrt{\Delta t}$$

with $\pi_u^* = \pi_d^* = 1/2$ and Δt the time increment. This has the two essential features: it equates expected stock and bond returns, and it has the right standard deviation. In addition, it has a continuous stock price (like Black-Scholes) as a limit.

One alternative is to choose the positive solution of $ud = 1$ and $u - d = 2\sigma\sqrt{\Delta t}$ (good for option price as a function of time) or a recombining trinomial (good for including some dependence of variance on the stock price).

A Simple Binomial Option Pricing Program

The file *crr.h* is a header file containing type declarations for the class of binomial option pricing objects including their member functions and variables. This is a description of what the compiler needs to know about the way a binomial option pricing object of type *crr* is accessed. This file is included at the top of each of the other files.

The file *crr.cc* contains the code that does the actual option pricing. The member *crr::crr* is a constructor that initializes the member parameters when a an object of type *crr* is declared. The function *crr::eurcall* is a member function that computes the price of a European call option. The function *crr::binopt* is a member function that computes the price of a *binary* option (also called a *digital option*—an even worse term) that pays off in a given price range.

The file *crrtest.cc* has the code that is the interface between the user and the pricing program in *crr.cc*. Note that only the public parts of an object of type *crr* can be accessed here.

The Header File crr.h

```
// crr.h
//
// Simple binomial option pricing model: declarations
//
#define MAXTERNODES 400
class crr {
public:
  crr(double ttm=.25,int npers=4,double r=.05,double sigma=.3);
  double eurcall(double S0,double X);
  double binopt(double S0,double lower,double upper);

private:
  int nper;
  double tinc,r1per,disc,up,down,prup,prdn,Sprice;
  double val[MAXTERNODES];};
```

The Computations File crr.cc: Part 1

```
// crrtest.cc
//
// Binomial option pricing model: test
//
#include <iostream.h>
#include "crr.h"
main() {
  crr c1;
  double stockP=0.0,strikeP;
  cout << "\nType the stock price, a space, the strike"
    << " price, and then ENTER." << "\n"
    << "Make the stock price negative to terminate." << "\n\n";
  while(1) {
    cout << "stock-price strike-price: ";
    cin >> stockP >> strikeP;
    if(stockP < 0.0){
      cout << "\nBye!\n" << endl;
```

```
  return(0);};
if(!cin){
  cout << "\nError: enter stock-price strike-price\n"
    << "Terminating\n" << endl;
  return(1);}
cout << "call option value = " << c1.eurcall(stockP,strikeP)
  << "\n\n";}}
```

The Computations File crr.cc: Part 1

```
// crr.cc
//
// Simple binomial option pricing model: implementation
// (following Cox, Ross, and Rubinstein)
//
#include <math.h>
#include <iostream.h>
#include "crr.h"
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
```

The Computations File crr.cc: Part 2

```
crr::crr(double ttm,int npers,double r,double sigma) {
  nper = npers;
  tinc = ttm/(double) nper;
  r1per = 1.0 + r * tinc;
  disc = 1.0/r1per;
  up = r1per + sigma * sqrt(tinc);
  down = r1per - sigma * sqrt(tinc);
  prup = disc * 0.5;
  prdn = disc * 0.5;}
```

The Computations File crr.cc: Part 3

```
double crr::eurcall(double S0,double X) {
  int i,j;
// initialize terminal payoffs
// i is the number of up moves over the whole life
  for(i=0;i<=nper;i++) {
    Sprice = S0*pow(up,(double) i)*pow(down,(double) (nper-i));
    val[i] = MAX(Sprice - X,0);}
// compute prices back through the tree
// j+1 is the number of periods from the end
// i is the number of up moves from the start
  for(j=0;j<nper;j++) {
    for(i=0;i<nper-j;i++) {
      val[i] = prdn * val[i] + prup * val[i+1];}}
  return(val[0]);}
```

The Computations File crr.cc: Part 4

```
double crr::binopt(double S0,double lower,double upper) {
  int i,j;
// initialize terminal payoffs
// i is the number of up moves over the whole life
  for(i=0;i<=nper;i++) {
    Sprice = S0*pow(up,(double) i)*pow(down,(double) (nper-i));
    val[i] = (((Sprice>=lower) && (Sprice<=upper)) ? 1.0 : 0.0);}
// compute prices back through the tree
// j+1 is the number of periods from the end
// i is the number of up moves from the start
  for(j=0;j<nper;j++) {
    for(i=0;i<nper-j;i++) {
      val[i] = prdn * val[i] + prup * val[i+1];}}
  return(val[0]);}
```