

# IE523: Financial Computing

## Fall, 2017

### Lesson 6: Computational Aspects of Option Pricing

©Prof. R.S. Sreenivas

In this lesson we will cover *Binomial Option Pricing Models*, followed by Option Pricing Models that use *Dynamic Programming*. Although we verify the correctness of our implementation by comparing our results to that given by the *Black-Scholes formula* for European Options, our models are not limited to pricing *just* European Options.

## 1 Dynamics of Asset Price

The material presented in this section is a significantly boiled-down version of what you can find in chapter 2 of Wilmott et al [6]. We work under the following assumptions – the past history of any asset is completely present/reflected in its current price, and markets respond immediately to any new information about an asset. These assumptions are basically saying that the asset price is Markovian in nature.

The asset price dynamics is described by a *stochastic differential equation* in terms of the *relative change* in asset price as follows

$$\frac{dS}{S} = \underbrace{\sigma dX}_{\text{random part}} + \underbrace{\mu dt}_{\text{deterministic part}}, \quad (1)$$

where  $dX$  has the following properties:

1.  $dX$  is a normal variate,
2. the mean of  $dX$  is zero, and
3. the variance of  $dX$  is  $dt$

That is,

$$dX = \phi \sqrt{dt},$$

where  $\phi$  is a unit-normal variate. All of this means,

$$E \left\{ \frac{dS}{S} \right\} = \mu dt \text{ and } \text{var} \left\{ \frac{dS}{S} \right\} = \sigma^2 dt$$

and if  $dS \leftarrow \Delta S$  and  $dt \leftarrow \Delta t$ , then  $\frac{\Delta S}{S}$  is a normal random variable with mean  $\mu \Delta t$  and variance  $\sigma^2 \Delta t$ .

#### For the Mathematically Inclined (Start)

Suppose the asset price is described by

$$\frac{dS}{S} = \sigma dX + \mu dt$$

as described earlier. If  $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ <sup>1</sup> is a smooth function in terms of its two variables denoted by  $S$  and  $t$ , then [Ito's Lemma \(Hyperlink\)](#) says

$$df = \underbrace{\sigma S \frac{\partial f}{\partial S} dX}_{\text{random part}} + \underbrace{\left\{ \mu S \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right\} dt}_{\text{deterministic part}}. \quad (2)$$

Its derivation (which can be found at the above hyperlink) is a direct application of Taylor's Series which was covered in lesson 4 of my notes, and uses the fact that the standard deviation of the returns (i.e.  $\frac{\Delta S}{S}$ ) is proportional to  $\sqrt{t}$ .

### Quick-and-Dirty Ito's Lemma Proof (Start)

As noted in the footnote 1, think of  $f(S, t)$  as the price of a derivative-instrument (e.g. the call option on MSFT) which depends on  $S$ , the price of the underlying (i.e. MSFT price), and time  $t$ . We know that the price of the underlying satisfies equation 1, which means

$$\begin{aligned} dS &= (S\mu dt) + (S\sigma \times dX), \text{ where } dX = (\overbrace{\phi}^{\text{get.gaussian()}} \times \sqrt{dt}), \\ \Rightarrow (dS)^2 &= \underbrace{(S\mu dt)^2 + 2(S\mu dt)(S\sigma \times dX)}_{\text{ignore these; powers of infinitesimals!}} + (S\sigma \times dX)^2 \\ \Rightarrow (dS)^2 &\approx (S\sigma)^2 \phi^2 dt. \\ \Rightarrow E\{(dS)^2\} &\approx E\{(S\sigma)^2 \phi^2 dt\} = (S\sigma)^2 dt \underbrace{E\{\phi^2\}}_{=1 \text{ (Why?)}}. \end{aligned}$$

Note, that  $\text{Var}\{x\} = E\{x^2\} - (E\{x\})^2$  for any r.v.  $x$ . If  $E\{x\} = 0$ , then  $E\{x^2\} = \text{Var}\{x\}$ . Since  $E\{\phi\} = 0$ , it follows that  $E\{\phi^2\} = \text{Var}\{\phi\} = 1$ .

We are looking for a similar differential equation that describes the dynamics of the price of the derivative-instrument. From Taylor's series we know

$$\begin{aligned} df &= \left( \frac{\partial f}{\partial S} \right) dS + \left( \frac{\partial f}{\partial t} \right) dt + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) (dS)^2 + \underbrace{\left\{ \left( \frac{\partial^2 f}{\partial S \partial t} \right) dS dt + \frac{1}{2} \left( \frac{\partial^2 f}{\partial t^2} \right) (dt)^2 + \dots \right\}}_{\text{ignore; higher powers of infinitesimals!}} \\ &\approx \left( \frac{\partial f}{\partial S} \right) \underbrace{dS}_{=S\mu dt + S\sigma dX} + \left( \frac{\partial f}{\partial t} \right) dt + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) \underbrace{(dS)^2}_{=(S\sigma)^2 \phi^2 dt} \\ &= \left( \frac{\partial f}{\partial S} \right) (S\mu dt) + \left( \frac{\partial f}{\partial S} \right) (S\sigma dX) + \left( \frac{\partial f}{\partial t} \right) dt + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) (S\sigma)^2 \phi^2 dt \\ &= \left\{ \left( \frac{\partial f}{\partial S} \right) (S\mu) + \left( \frac{\partial f}{\partial t} \right) + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) (S\sigma)^2 \phi^2 \right\} dt + \left( \frac{\partial f}{\partial S} \right) (S\sigma) dX \end{aligned}$$

That is,  $df$  has a random component,  $\left( \frac{\partial f}{\partial S} \right) (S\sigma) dX$ , and a deterministic component,  $\left\{ \left( \frac{\partial f}{\partial S} \right) (S\mu) + \left( \frac{\partial f}{\partial t} \right) + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) (S\sigma)^2 \phi^2 \right\} dt$ . The expected-value of the deterministic com-

<sup>1</sup>Think of  $f(\bullet, \bullet)$  to be the price of a derivative instrument – say, the price of a *call option* (described in the next section).

ponent is  $\left\{ \left( \frac{\partial f}{\partial S} \right) (S\mu) + \left( \frac{\partial f}{\partial t} \right) + \frac{1}{2} \left( \frac{\partial^2 f}{\partial S^2} \right) (S\sigma)^2 \right\} dt$  as  $E\{\phi^2\} = 1$ . This is what we have in equation 2.

### Quick-and-Dirty Ito's Lemma Proof (End)

There is no reason why we should always interpret  $f(S, t)$  as the price of a derivative-instrument, as before. Often we will use  $f(S, t) = \log(S) \Rightarrow \frac{\partial f}{\partial S} = \frac{1}{S} \Rightarrow S \frac{\partial f}{\partial S} = 1; \frac{\partial^2 f}{\partial S^2} = -\frac{1}{S^2} \Rightarrow S^2 \frac{\partial^2 f}{\partial S^2} = -1$ ; and,  $\frac{\partial f}{\partial t} = 0$ . From equation 2 we have

$$df = \sigma dX + \left( \mu - \frac{1}{2} \sigma^2 \right) dt.$$

That is,  $df$  is normally distributed with mean  $(\mu - \frac{1}{2} \sigma^2) dt$  and variance  $\sigma^2 dt$ . You could view  $f(S) - f(S_0)$ , where  $S_0$  is the starting price of the asset, to be a sum of several independent  $df$ 's. Since the normal distribution is a *stable* distribution – that is, sum of two normally distributed rv's is also normally distributed – we have

$$\log \left( \frac{S_t}{S_0} \right) = \sigma B_t + \left( \mu - \frac{1}{2} \sigma^2 \right) t,$$

where  $B_t$  is normally distributed with zero-mean and variance  $t$ , and

$$\Rightarrow \left( \frac{S_t}{S_0} \right) = e^{\sigma B_t + (\mu - \frac{1}{2} \sigma^2) t}$$

$$\Rightarrow S_t = S_0 e^{\sigma B_t + (\mu - \frac{1}{2} \sigma^2) t}$$

$$\Rightarrow S_t = S_0 e^{\phi \sigma \sqrt{t} + \left( \mu - \frac{1}{2} \sigma^2 \right) t},$$

where  $\phi$  is a unit-normal rv (i.e.  $\phi = \text{get.gaussian}()$ ). We will see later on that this is quite useful when it comes to *Monte-Carlo Simulations* of asset price movements<sup>2</sup>.

## Delta Hedging

You can effectively remove the “random part” in equation 2 (which you can think of as the price of a derivative instrument, if you want), by using what is known as the principle of *hedging*. This is strongly related to the process of *bond immunization* that was done in lesson 4 of my notes.

To be precise, suppose we have one unit of a derivative instrument with price  $f(S, t)$ , which is described by equation 2. We purchase  $\Delta$ -many shares of the underlying (at price  $S$ ) amount of this instrument to form a portfolio. The portfolio value will be  $g(S, t) = f(S, t) - \Delta S$ , and consequently, the change in its value (after time  $dt$ ) is

<sup>2</sup>This supports the widely accepted model that stock prices are *log-normally* distributed. That is, we assume the existence of a normally distributed r.v.  $X$  (characterized by its mean and variance/standard-deviation, such that the change in the logarithm of the stock price over some time period  $T$  is given by  $X$ ). That is,  $\log S_T = \log S_0 + X$ , or  $S_T = S_0 e^X$ . In our case,  $X = \sigma B_T + (\mu - \frac{1}{2} \sigma^2) T$ . As noted above,  $E\{X\} = (\mu - \frac{1}{2} \sigma^2) T$  and  $\text{var}(X) = \sigma^2 T$ . From our statistics course, we know that  $E\{S_T\} = S_0 E\{e^X\} = S_0 (e^{E\{X\} + \frac{1}{2} \text{var}(X)}) = S_0 e^{\mu T}$ , confirming what we already know.

$$\Delta = \frac{\partial f}{\partial S}$$

given by

$$\begin{aligned} dg &= df - \Delta dS \\ &= \underbrace{\sigma S \frac{\partial f}{\partial S} dX + \left\{ \mu S \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right\}}_{\text{equation 2 for } df} - \Delta \underbrace{(\sigma S dX + \mu S dt)}_{\text{equation 1 for } dS} \\ &= \underbrace{\sigma S \left( \frac{\partial f}{\partial S} - \Delta \right) dX}_{\text{random part}} + \underbrace{\left( \mu S \left\{ \frac{\partial f}{\partial S} - \Delta \right\} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right) dt}_{\text{deterministic part}}. \end{aligned}$$

So, if  $\Delta = \frac{\partial f}{\partial S}$ , we have effectively removed/eliminated the “random part” in the dynamics of the portfolio’s price. This is the principle of *hedging*. The dynamics of the portfolio price after hedging is given by

$$dg = \left( \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right) dt \quad (3)$$

Even though *continuous hedging*, as the equation suggests, will run into practical problems (transaction costs, etc.), there are approximations to this process that can (almost) accomplish the same objective. You should read Taleb’s book [5] before you go about “transliterating” this nifty theory into practice.



## Black-Scholes PDE for Derivative Instrument’s Value

Suppose you have an asset, whose price follows

$$\frac{dS}{S} = \sigma dX + \mu dt,$$

and we have some instrument whose price  $V$  (like a *call option*, or something) is derived its value from this asset  $S$  and time  $t$ , then by Ito’s Lemma, we have

$$dV = \sigma S \frac{\partial V}{\partial S} dX + \left\{ \mu S \frac{\partial V}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right\} dt.$$

We can use Delta-Hedging to create a portfolio with instantaneous value  $Y = V - \frac{\partial V}{\partial S} S$ , where there is no random-component in the dynamics of  $Y$ , and (cf. equation 3)

$$dY = \left( \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt$$

You could have sold the portfolio and gotten  $Y$  \$’s and invested the amount in a riskless instrument with rate  $r$ . If you did that, you would have made  $rYdt$  extra in time  $dt$ .

Using a “no-arbitrage” argument you can show that

$$\left( \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt = rYdt.$$

Substituting  $Y = V - \frac{\partial V}{\partial S}S$  into the above equation we get

$$\left(\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t}\right) + rS \frac{\partial V}{\partial S} - rV = 0$$

→ If I can solve this  
⇒ can price every derivatives

This is the celebrated **Black-Scholes Partial Differential Equation**, which describes the dynamics of the price of the derivative instrument  $V$ . There a bunch of assumptions that I have outlined in section 2 that form the basis of this equation.

For the Mathematically Inclined (End)

## 2 Black-Scholes Formula

Assuming (1) the asset price follows the stochastic differential equation described earlier, (2) there are no transaction costs associated with (hedging) a portfolio, (3) no dividends are paid, (4) there is no arbitrage opportunities at any time, (5) continuous trading is possible, (6) short selling is permitted, and (7) assets are infinitely divisible, the Black-Scholes partial differential equation describes the dynamics of the portfolio. Based on this PDE, it is possible to derive expressions for the price of various derivative instruments.

An *European Call Option* (*European Put Option*) gives the holder the right, but not the obligation to buy (sell) an underlying security at some agreed-upon exercise time  $T$  at a (*strike*) price  $K$ . If  $S_T$  is the price of the underlying security at the exercise time  $T$ , then the payoff to holder of a European Call (European Put) option is  $\max(0, (S_T - K))$  ( $\max(0, (K - S_T))$ ). The issue that we are faced with is this – how much are we willing to pay for these options?

A function that computes the price (under appropriate assumptions, that will be covered next semester) will take as input the following parameters

1. The expiration time  $T$ ,
2. The continuously-compounded, risk-free, interest rate  $r$ ,
3. The volatility of the underlying security  $\sigma$ , that is measured as the standard-deviation of the logarithm of price changes,
4. The initial price of the underlying security  $S$ , and
5. The strike price  $K$ .

The Black-Scholes Formula for the price of an European Call Option is

$$C = SN(d_1) - Ke^{-rT}N(d_2)$$

where  $N(\bullet)$  is the *Cumulative Normal Distribution*,

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}},$$

and

$$d_2 = d_1 - \sigma \sqrt{T}.$$

The price of an European Call Option is

$$C = Ke^{-rT}N(-d_2) - SN(-d_1).$$

The C++ code `black.scholes.cpp` that can be found on the Compass Website implements these formulae. In order to compile this, you will need `normdist.h`, which you can find at the same site. This C++ program borrows heavily from Prof. Ødegaard's implementation in page 90 of his notes. A sample output is shown in figure 1.

```

Terminal — bash — 78x16
vpn3-144151:Debug sreenivas$ ./Black\ Scholes 0.5 0.08 0.3 60.0 50.0
Black-Scholes European Option Pricing
Expiration Time (Years) = 0.5
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Call Price according to Black-Scholes = 12.8226
Put Price according to Black-Scholes = 0.862071
-----
Verifying Put-Call Parity: S+P-C = Kexp(-r*T)
60 + 0.862071 - 12.8226 = 50exp(-0.08 * 0.5)
48.0395 = 48.0395
-----
vpn3-144151:Debug sreenivas$

```

Figure 1: Sample output of the C++ code `black.scholes.cpp`.

There is a relationship between the Put and Call prices when it comes to European Options, and it is called the **Put-Call Parity**, which says

$$S + P - C = Ke^{-rT},$$

where  $S$  is the price of the underlying asset/stock,  $P$  ( $C$ ) is the price of the put (call) option,  $K$  is the strike price, and  $T$  is the expiration-time. The C++ code `black.scholes.cpp` verifies this assertion as shown in figure 1.

### 3 Parable of the Bookmaker: **No-Arbitrage Probability**

As noted in Lesson 5, the probability-theorist and statistician **Bruno de Finetti** said:

PROBABILITY DOES NOT EXIST. The abandonment of superstitious beliefs about the existence of Phlogiston, the Cosmic Ether, Absolute Space and Time... or Fairies and Witches, was an essential step along the road to scientific thinking. Probability too, if regarded as something endowed with some kind of objective existence, is no less a misleading conception, and illusory attempt to exteriorize or materialize our true probabilistic beliefs

Buy P  
Buy 1 share S  
sell call  
 $P + S - C$   
If  
 $C + Ke^{-rT} > P + S$   
if  $S_T < K$

$$\underbrace{(K - S_T)}_{\text{Call}} + S_T - \underbrace{(P + S - C)}_{\text{bank}} e^{rT} = K - (P + S - C) e^{rT} > 0$$

$$Q: \text{if } S_T > K$$

$$S_T - (S_T - K) - (p + S - C)e^{rT} = 1 - (p + S - C)e^{rT} > 0$$

He suggests that probability is essentially an individual's belief about the chance/likelihood of an event. Using the concept of *arbitrage*, de Finetti proposed a definition of probability in terms of prices placed on a lottery. He suggests that  $p(E)$  is the unit-price at which one would be indifferent between buying and selling a lottery ticket that paid \$1 if  $E$  occurred, and \$0 otherwise. Likewise,  $p(E | F)$  is the unit price at which one would be indifferent between buying and selling a ticket paying \$1 if  $E \cap F$  occurs, \$0 if  $F$  occurs without  $E$ , and refund of the purchase price if  $F$  fails to occur. de Finetti showed that such a system of prices eliminates arbitrage if and only if the prices satisfied the requirements of a probability measure<sup>3</sup>. That is, he argues that probabilities must be interpreted as the above mentioned prices, and that it has no meaning beyond this.

This notion of arbitrage-free pricing yields a unique probability measure in many instances – it is this probability measure that forms the basis of several probabilistic models for pricing that we will use in this chapter. It might be worthwhile to go over what Baxter and Rennie call “*The Parable of the Boomaker*” in their book [1]. These guys are basically arguing (quite eloquently!) for pricing/probability-model that is based on the no-arbitrage principle. Some betting-terminology first:

1. if the odds are “ $m : n$  against an event  $E$ ,” then we expect the event will not occur  $m$  times for every  $n$  times it does occur. That is  $p(E) = \frac{n}{m+n}$ . In terms of cash-transactions, these odds would mean that if  $E$  does occur, a bet of \$ $n$  will be rewarded with \$ $m$ , plus stake returned (i.e. the cost of placing the bet, will be refunded).
2. if the odds are “ $m : n$  on an event  $E$ ” or “ $m : n$  for an event  $E$ ,” then we expect the event will occur  $m$  times for every  $n$  times it does not occur. That is  $p(E) = \frac{m}{m+n}$ . In terms of cash-transactions, these odds would mean that if  $E$  does occur, a bet of \$ $m$  will be rewarded with \$ $n$ , plus stake returned (i.e. the cost of placing the bet, will be refunded).

Baxter and Rennie consider a situation where there are two horses  $A$  and  $B$  that are to run in race. The bookmaker, after considering factors such as training, diet and choice of jockey, calculates that  $A$  (resp.  $B$ ) has a 25% (resp. 75%) chance of winning. Let us suppose the bets that are placed add up to \$5,000 for  $A$  winning, and \$10,000 for  $B$  winning.

Suppose the bookmaker sets the odds as “3 : 1 against  $A$ ” and “3 : 1 on  $B$ ,” respectively. If  $A$  wins the race, the bookmaker has to return the \$5,000 to those that bet on  $A$  winning. In addition, the bookmaker will have to give  $3 \times \$5,000 = \$15,000$  to these folks. This would result in a loss of \$5,000 to the bookmaker. If  $B$  wins the race, the bookmaker returns the \$10,000 to the folks that bet on  $B$ ; in addition, give them a total of  $\frac{1}{3} \times \$10,000 = \$3,333$  for their bets. This will result in a profit of \$1,667 to the bookmaker. The expected profit, under these odds, would be

$$\left(\frac{1}{4} \times -\$5,000\right) + \left(\frac{3}{4} \times \$1,667\right) = \$0.$$

<sup>3</sup>That is,  $p(E) \geq 0$ ,  $p(E) + p(\bar{E}) = 1$ ,  $p(E \cup F) = p(E) + p(F)$  if  $E \cap F = \emptyset$ ,  $p(E \cap F) = p(E | F)p(F)$ , etc.

That is, in the long-run, over a number of similar but independent races, the law of averages would allow the bookmaker to break even.

To see this in the general setting, suppose \$x (resp. \$y) was bet on A (resp. B), and the odds are set m : n against A, and m : n on B, the expected profit would be

$$\underbrace{\left(\frac{n}{n+m}\right)}_{=p(A)} \times \overbrace{\left(y - x \frac{m}{n}\right)}^{\text{=Profit if A wins}} + \underbrace{\left(\frac{m}{n+m}\right)}_{=p(B)} \times \overbrace{\left(x - y \frac{n}{m}\right)}^{\text{=Profit if B wins}} = \frac{1}{n+m} (\cancel{ny} - \cancel{mx} + \cancel{mx} - \cancel{ny}) = 0.$$

However, until the long term comes, there is a chance of making a huge loss.

Suppose the bookmaker sets the odds according to the money wagered – we would have “2 : 1 against A” and “2 : 1 on B,” instead. In this case, if A wins the race, the bookmaker has to return \$5,000 + \$10,000 to those that bet on A winning. If B wins the race, the bookmaker returns \$10,000 + \$5,000 to the folks that bet on B. That is, the bookmaker breaks even in both cases, and is consequently indifferent to the outcome of the race if the odds are set based on the amounts wagered. To see this in the general setting, suppose \$x (resp. \$y) was bet on A (resp. B), and the odds are set y : x against A, and y : x on B, the bookmaker will pay \$x + \$y, irrespective of which horse wins – that is, the bookmaker breaks even in each instance.

Even if the cost to the bookmaker is accounted for, using the actual probabilities will, on an-average, pay for the bookmaker’s costs, but there is a chance of substantial short-term loss. For the bookmaker to cover his costs in a riskless manner, he is best advised to change the odds using the amounts that are bet in each race. Keep in mind that this would free the bookmaker from spending any resources on estimating the actual probabilities (i.e. look at the training/diet/jockey etc for each horse). **In more ways than one, this will make Bruno de Finetti (see above) happy!**

When we use probability measures in pricing algorithms later on in this chapter, it is important to note that we are essentially using the probability measure that comes out of a no-arbitrage argument. That is, if the actors/players used a different probability measure than the no-arbitrage probability, then we can definitely claim that one of them is not acting in his/her best “self-interest.”

### 3.1 **Concept of Replication Portfolios and the No-Arbitrage Probability Measure for Option Pricing**

The objective is to create a portfolio (usually consisting of the underlying stock and a risk-free Bond) that has the same payoff as an Option. The natural (no arbitrage) conclusion is that the Option must have the same price as the equivalent/replicated portfolio.

To illustrate this, suppose we had a discrete-time economy – that is, stock prices jump in discrete time-steps (as opposed to changing in a continuous manner) – and we had a stock that is valued at \$100 at the current discrete time-step, and it could be \$125 or \$80 in the next discrete time-step (cf. figure 2). Suppose, we are looking at a call Option with strike \$100, then the payoff would be as shown in figure 2. Now, let us look at a portfolio that consists of x units of the stock, and y units of a risk-free Bond



with rate 10%. If the payoff of this portfolio is to match the payoff of the call, then we require

$$\underbrace{\begin{array}{c} 125 \times x + y(1 + 0.1) \\ 80 \times x + y(1 + 0.1) \end{array}}_{\text{Portfolio payoff}} = \underbrace{\begin{array}{c} 25 \\ 0 \end{array}}_{\text{Call Option payoff}},$$

which yields  $x = \frac{25}{45} = \frac{5}{9} \approx 0.5556$  and  $y = \frac{-80 \times \frac{5}{9}}{1.1} \approx -40.404$ . That is, you borrow \$40.40 today (at interest rate 10%) and buy 0.5556 of the face-value of the stock (which requires \$55.556 from you). You will have to cough-up \$15.15 (= \$55.556 - \$40.404) from your pocket for this – which is effectively the cost of this single time-step Call Option at strike price \$100.

Let us look at the put Option for the same example. Under the same logic as before, we are required to solve

$$\underbrace{\begin{array}{c} 125 \times x + y(1 + 0.1) \\ 80 \times x + y(1 + 0.1) \end{array}}_{\text{Portfolio payoff}} = \underbrace{\begin{array}{c} 0 \\ 20 \end{array}}_{\text{Put Option payoff}},$$

which yields  $x = -\frac{20}{45} = -\frac{4}{9} \approx -0.4444$  and  $y = \frac{-125 \times -\frac{4}{9}}{1.1} \approx 50.505$ . That is, you short  $\frac{4}{9}$  of the stock (which has current value \$44.44) and then lend \$50.505 at 10%. In net effect you will be out by \$6.06, which is the cost of the Put Option.

**It is worth noting that the Put-Call Parity holds –**

$$\underbrace{\$100}_S + \underbrace{\$6.06}_P - \underbrace{\$15.15}_C = \underbrace{\$100}_K \times \underbrace{\frac{1}{1.1}}_{e^{-rT}}$$

You might want to try changing the rate from 10% to 25% and see what happens to the prices of the Call and Put Option. See if you can justify the price of the Put Option that you get after you make this change.

If there are additional stages (i.e. 3 or higher stages) to the model, then we will see that the strict equality constraints are replaced by inequality constraints and we seek the smallest value for the replicating portfolio at stage (0,0). This will become clear later.

**There are a couple of important things for you to take away from these examples –**

1. **When you replicate a Call Option, you go long on the stock (i.e.  $x$  is positive) and you will borrow money (i.e.  $y$  is negative).**
2. **When you replicate a Put Option, you will short the stock (i.e.  $x$  is negative) and you will lend money (i.e.  $y$  is positive).**

These observations are important to the LP formalism that prices these options. This is because, the standard LP-solvers (like `lp_solve`, for example) require all variables to be positive. This will require us to make appropriate sign-changes to those variables that will go negative. This will also become clear later.

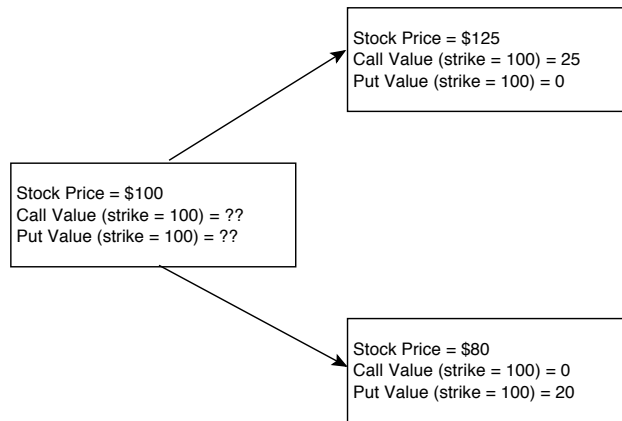


Figure 2: Two-Step process to illustrate the idea of replication portfolios.

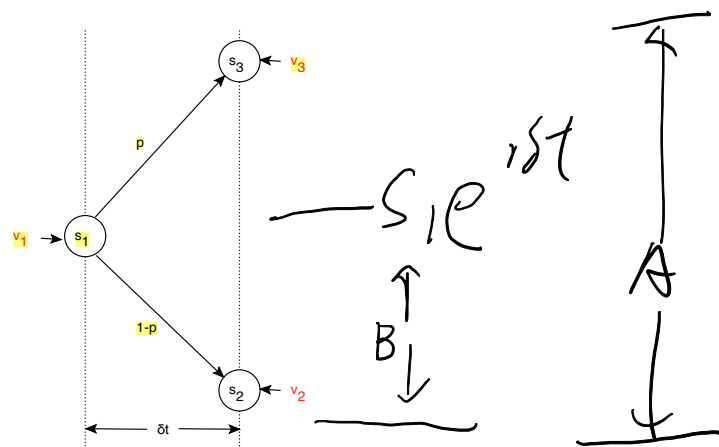


Figure 3: Relating equivalent portfolios and the Binomial Model. The underlying's initial price is  $s_1$ , and it changes to  $s_3$  (resp.  $s_2$ ) with probability  $p$  (resp.  $1 - p$ ), where  $p$  is an arbitrary probability. Just for clarity, we suppose  $s_2 < s_1 < s_3$ , and  $\delta t$  is the amount of time that has elapsed between the two-stages. The value of the option that we are trying to price is  $v_1, v_2$  and  $v_3$  at the appropriate discrete-states of the economy. We also suppose the risk-free rate is  $r$  (continuous compounding).

$$p = \frac{S_1 e^{rt} - S_2}{S_3 - S_1} = \frac{B}{A}$$

$$S + P - C = Ke^{rT}$$

### Replication Portfolios & Binomial Models (Start)

Let us connect the model of figure 3 and some stuff we covered in Binomial models. Suppose, we had an equivalent portfolio with  $x$ -many shares of the underlying stock, and  $y$ -many shares of a risk-free bond with face-value  $B_0$ . After  $\delta t$ , equating the values of the option at each future discrete-state, we have:

$$\begin{aligned}(x \times s_3) + (y \times B_0 e^{r\delta t}) &= v_3 \text{ ("uptick")} \\ (x \times s_2) + (y \times B_0 e^{r\delta t}) &= v_2 \text{ ("downtick").}\end{aligned}$$

This results in

$$\begin{aligned}x &= \frac{v_3 - v_2}{s_3 - s_2} \text{ and ,} \\ y &= B_0^{-1} e^{-r\delta t} \left( v_3 - \frac{(v_3 - v_2)s_3}{s_3 - s_2} \right),\end{aligned}$$

which in turn implies that the price of the option is

$$\begin{aligned}v_1 &= s_1 \left( \frac{v_3 - v_2}{s_3 - s_2} \right) + e^{-r\delta t} \left( v_3 - \frac{(v_3 - v_2)s_3}{s_3 - s_2} \right) \quad (4) \\ &= \left( \frac{v_3 - v_2}{s_3 - s_2} \right) (s_1 - e^{-r\delta t} s_3) + e^{-r\delta t} v_3 \\ &= (v_3 - v_2) \left( \frac{s_1 - e^{-r\delta t} s_3}{s_3 - s_2} \right) + e^{-r\delta t} v_3 \\ &= e^{-r\delta t} (v_3 - v_2) \left( \frac{s_1 e^{r\delta t} - s_3}{s_3 - s_2} \right) + e^{-r\delta t} v_3 \\ &= e^{-r\delta t} \left( (v_3 - v_2) \left( \frac{s_1 e^{r\delta t} - s_3}{s_3 - s_2} \right) + v_3 \right) \quad (5)\end{aligned}$$

Equation 4 is the no-arbitrage price of the option we are trying to price for obvious reasons.

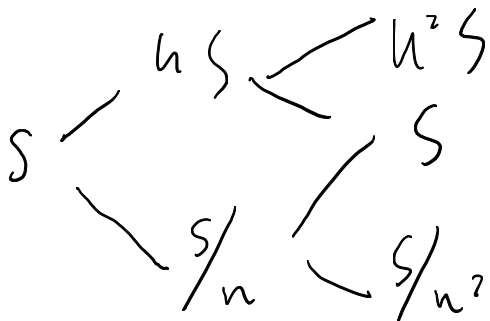
Suppose, we create a variable  $q$ , where

$$q = \frac{s_1 e^{r\delta t} - s_2}{s_3 - s_2} \left( \Rightarrow 1 - q = \frac{s_3 - s_1 e^{r\delta t}}{s_3 - s_2} \right) \quad (6)$$

we note that  $0 < q < 1$ . To see this, suppose  $q \leq 0$ , then because  $s_3 > s_2$ , we have  $s_1 e^{r\delta t} \leq s_2 (< s_3)$ . We could borrow  $s_1$  at the beginning, and purchase a share of the underlying. After  $\delta t$  time, we would owe  $s_1 e^{r\delta t}$ , but the underlying would be worth at least  $s_2 (< s_3)$ , and a guaranteed risk-free profit can be made. If  $q \geq 1$ , then we have  $s_2 < s_3 \leq s_1 e^{r\delta t}$ , and this time we can short the underlying and buy the riskless bond, and we have a way of making riskless profits, which is not possible. Therefore,  $0 < q < 1$ .

Combining equations 6 and 5 we have

$$v_1 = e^{-r\delta t} ((v_3 - v_2)(q - 1) + v_3) = e^{-r\delta t} (qv_3 + (1 - q)v_2).$$



This would mean that we can interpret  $q$  as a probability of an uptick (irrespective of what  $p$  is, per se). If  $s_2 = \frac{s_1}{u}$  and  $s_3 = us_1$ , then equation 6 will result in

$$q = \frac{s_1 \overbrace{e^{r\delta t}}^{=R} - \frac{s_1}{u}}{us_1 - \frac{s_1}{u}} = \frac{R - 1/u}{u - 1/u}, \quad (7)$$

which is what we used as the uptick probability we looked at binomial models earlier, which is a special case of equation 6 (and notice  $q$  has nothing to do with  $p$  shown in figure 3).

Let us check the martingale property under the value of  $q$  identified by equation 6. We have

$$\begin{aligned} E\{S_{\delta t} \mid s_1\} &= qs_3 + (1-q)s_2 \\ &= \frac{s_1R - s_2}{s_3 - s_2}s_3 + \frac{s_3 - s_1R}{s_3 - s_2}s_2 \\ &= \frac{s_1s_3R - s_2s_3 + s_2s_3 - s_1s_2R}{s_3 - s_2} = s_1R. \end{aligned}$$

That is, if we used  $q$  of equation 6 as if it were the uptick probability for any  $s_2$  and  $s_3$ , we have the martingale property to be true.

**Note, we are essentially replacing the original probability measure  $p$  with the no-arbitrage probability measure  $q$ . This would make Bruno de Finetti happy – the probability measure  $q$  is not an individual belief like  $p$  would be (which is a “no-no” in his book).**

This essentially gives us a plethora of possibilities for the binomial model – we do not always have to assume  $s_3 = us_1$ ,  $s_2 = s_1/u$ , and  $u = e^{\sigma\sqrt{\delta t}}$ , every time. If we picked a different relationship between  $s_1$ ,  $s_2$ ,  $s_3$ ,  $r$  and  $\delta t$ , we have to ensure that some desirable statistical discipline is enforced. The traditional binomial model which uses  $s_2 = \frac{s_1}{u}$ ,  $s_3 = us_1$  where  $u = e^{\sigma\sqrt{\delta t}}$  makes sure the log-normal conditions of equations 16 and 17 of reference [4] are satisfied. If you believe that these conditions are not a true reflection of reality, and you have an alternate model, then you have to come up with something that parallels equations 16 and 17 of reference [4], for your definition of reality, and from those equations you have to infer the relationship between  $s_1$ ,  $s_2$ ,  $s_3$ ,  $r$  and  $\delta t$ . The programming aspect of this exercise will be pretty much the same as the traditional binomial model described in the next section.

Replication Portfolios & Binomial Models (End)

## 4 Binomial Models

The purpose of this section is to introduce you to the use of recursion in the *Binomial Option Pricing Models* that you have been introduced to in your other courses. Note, the complexity (i.e. the amount of time it takes to run the code) is not pretty with the recursive formulation (as with all multi-state binomial models). That said, you will see that the recursive implementation has a representational elegance.

The nitty-gritty details of the Binomial Model can be found in Jabbour, Kramin and Young's paper [4] that you can find on the Compass website. I will just give you the basics of the binomial model here. We are interested in computing the price of an option on an underlying asset, we will just call it a stock. The inputs for the model are

1. The expiration time of the option in years ( $T$ ).
2. The number  $n$  of equal-length periods into which  $T$  is to be divided. Each period has length  $\Delta t = \frac{T}{n}$ . For  $k = 0, 1, 2, \dots, n$ , we have the  $k$ -th discrete time instant to refer to  $k\Delta t$  in continuous time.
3. The continuously compounded annualized risk-free interest rate  $r$ . So, a dollar in a risk-free instrument at discrete-time step  $k$  will be worth  $R = e^{r\Delta t}$  in step  $k + 1$ .
4. The annual volatility  $\sigma$  of the stock price (expressed as a %-age of the stock price).

For these inputs we compute

1. The *up-factor*  $u$  and the *down-factor*  $\frac{1}{u}$ , where

$$u = e^{\sigma\sqrt{\Delta t}} = e^{\sigma\sqrt{T/n}} \text{ (cf. equation 20, page 991, [4]).}$$

The *up-tick probability*  $p$  and the *down-tick probability*  $q = 1 - p$ , where

$$p = \frac{R - \frac{1}{u}}{u - \frac{1}{u}} \text{ (cf. equation 25, page 992, [4]).}$$

As the discussion following equation 25, page 992 of [4] indicates, there are reasons why you must choose  $n$  such that  $\frac{T}{n} < \frac{\sigma^2}{r^2}$ . This model is accurate in the limit as  $n \rightarrow \infty$ . That said, we only need to pick a large enough value of  $n$  that makes the results accurate upto the second decimal place (assuming you are buying/selling single options).

The stock price at discrete time step  $k$  is denoted by  $S_k$ . The initial stock price  $S_0$  is non-random and known. The process  $S_k$  is specified as

$$S_{k+1} = \begin{cases} uS_k & \text{with probability } p, \\ \frac{S_k}{u} & \text{with probability } 1 - p. \end{cases}$$

With this, we have

$$E\{S_{k+1} \mid S_k\} = R \times S_k$$

that is, the discounted stock price process is a *martingale*.

The stock price process can be described by a *binomial tree*, where the initial stock price  $S_0$  is the root-node, and each path in the tree represents a possible sequence of stock prices  $S_0, S_1, \dots, S_n$ . A price path  $\omega$  is characterized by a string of random variables  $X_1, X_2, \dots, X_n$  where  $X_n = 1$  if there is an up-tick at time  $k$  and -1 otherwise. That is,

$$S_{k+1} = S_k u^{X_{k+1}}, k = 0, 1, \dots, n - 1$$

and

$$Prob\{X_k = 1\} = p; Prob\{X_k = -1\} = 1 - p$$

If we let  $H_k$  be the number of up-ticks that have occurred by the  $k$ -th discrete time instant, we have

$$S_k = S_0 u^{H_k} \left(\frac{1}{u}\right)^{k-H_k} = S_0 u^{2H_k - k}.$$

This would mean that several nodes at depth  $k$  actually represent the *same* stock price  $S_k$ . The resulting combined/collapsed binomial tree is called a *binomial lattice*. We will number each node in the lattice as  $(k, i)$  where the discrete-time step is  $k$  and  $Y_k = i$ , where

$$Y_k = \sum_{i=1}^k X_i.$$

At the  $k$ -th level (or depth  $k$ ) the binomial lattice will contain nodes

$$(k, -k), (k, -k+2), \dots, (k, k).$$

This can be inferred from the lattice of figure 4. The stock price  $S$  at state  $(k, i)$  is denoted by  $S(k, i)$ , and it is

$$S(k, i) = S(0, 0)u^i = S_0 u^i.$$

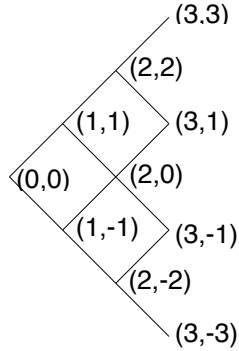


Figure 4: A binomial lattice. Each node is indexed as  $(k, i)$  where  $k$  is the discrete-time and  $i = (\text{\#up-ticks} - \text{\#down-ticks})$ . It is not hard to see that at the  $k$ -th “level” there are  $(k + 1)$  vertices in the lattice. That is, there are  $\frac{(k+1)(k+2)}{2} (= O(k^2))$  many vertices in the lattice till the  $k$ -th “level.” When  $k = 3$  as shown above, we have  $\frac{(3+1)(3+2)}{2} = 10$  vertices in total in the lattice.

## 5 Options

An *European-style option* which expires at discrete time  $n$  gives the owner the right to *exercise* the option at time  $n$ . When the owner exercises the option at time  $n$ , she

receives a payoff  $G_n$ , which depends only on the value of the stock at  $n$  (i.e.  $S_n$ ). In a *call* option with *strike price*  $K$ , the payoff  $G_n = \max\{0, (S_n - K)\}$ . In a *put* option with strike price  $K$ , the payoff  $G_n = \max\{0, (K - S_n)\}$ . Since the payoff in these cases only depends on  $S_n$  (and not on the path  $S_0, S_1, \dots, S_n$  that the stock-price takes), this option is *path-independent*.

The payoff for an european-style *Asian call option* with strike price  $K$  is given by  $G_n = \max\{0, A_n - K\}$ , where  $A_k$  is the average stock price from discrete-time 0 to discrete-time  $k$ , given by

$$A_k = \frac{S_0 + S_1 + \dots + S_k}{k+1} \text{ (Or, } A_k = \frac{S_1 + \dots + S_k}{k} \text{)}$$

Likewise, the payoff for european-style *Asian put option* with strike price  $K$  is given by  $G_n = \max\{0, K - A_n\}$ . Unlike the previous ones, the payoff for these option are *path-dependent*.

An *American-style option* which expires at discrete time  $n$  gives the owner the right to exercise the option at any time on-or-before the expiration time  $n$ . Such an option is characterized by a *payoff process*  $G_k$  ( $k = 0, 1, \dots, n$ ), where  $G_k$  only depends on  $S_k$ . An american *call* option with strike price  $K$  that expires at discrete time  $n$  has a payoff  $G_k = \max\{0, (S_k - K)\}$ , ( $k = 0, 1, \dots, n$ ). The put option will have a payoff  $G_k = \max\{0, (K - S_k)\}$ , ( $k = 0, 1, \dots, n$ ).

An american-style *asian call* option has payoff  $G_k = \max\{0, A_k - K\}$ , ( $k = 0, 1, \dots, n$ ), and the *put* option would have a payoff  $G_k = \max\{0, K - A_k\}$ , ( $k = 0, 1, \dots, n$ ).

## 6 Pricing Options

For the present discussion consider the binomial tree (i.e. not the binomial lattice). Let  $v^+$  ( $v^-$ ) be the up- and down-successor nodes of a node  $v$  in the tree. Each node  $v$  at depth  $k$  (i.e.  $D(v) = k$ ) defines a particular path of stock prices  $S_0, S_1, \dots, S_k$ . Let  $P(v)$  denote the probability of the path from the root to node  $v$ . If node  $v \rightarrow w$  (i.e. node  $v$  is connected by a single arc to  $w$ ) then  $P(w | v)$  is the (conditional) probability of reaching  $w$  if we started at  $v$ . That is,

$$P(w | v) = \frac{P(w)}{P(v)}.$$

We can have a process  $Z(v)$  that is defined at each node  $v$  (for instance,  $Z(v) = S(v)$  the stock price at node  $v$ ; or,  $Z(v) = A(v)$  the average stock price on the path from the root node to node  $v$ ; etc etc). The expectation of  $Z(v)$  is given by

$$E\{Z(v) | D(v) = k\} = \sum_{v: D(v)=k} Z(v)P(v).$$

For a european-style option with a payoff function  $G_n$ , the arbitrage-free value of the option at a depth  $k$  node  $v$  (i.e.  $D(v) = k$ ) is given by

$$V(v) = R^{k-n} E\{G_n | v\},$$

which can be written recursively as

$$V(v) = \begin{cases} \frac{pV(v^+) + (1-p)V(v^-)}{R} & \text{if } v \text{ is not a terminal node,} \\ G(v) & \text{if } v \text{ is a terminal node.} \end{cases}$$

For path-independent options the recursive expression shown above remains valid even if  $v, v^+, v^-$  are vertices in the binomial lattice (instead of vertices in the binomial tree).

You can download a recursive binomial european options pricing code that I wrote in C++ from the Compass website, where it is listed as `euro_option_recursion.cpp`. A sample output is presented below.

Recursive Binomial European Option Pricing

Expiration Time (Years) = 0.5

Number of Divisions = 20

Risk Free Interest Rate = 0.08

Volatility (%age of stock value) = 30

Initial Stock Price = 60

Strike Price = 50

---

Up Factor = 1.04858

Uptick Probability = 0.509239

---

Binomial Price of an European Call Option = 12.8055

Call Price according to Black-Scholes = 12.8226

---

Binomial Price of an European Put Option = 0.845018

Put Price according to Black-Scholes = 0.862071

---

Verifying Put-Call Parity:  $S + P - C = K \exp(-r \cdot T)$

$60 + 0.845018 - 12.8055 = 50 \exp(-0.08 \cdot 0.5)$

$48.0395 = 48.0395$

---

The valuation of american-style options is complicated by the owner's *strategy* to exercise her option – at any time  $k \leq n$ , she can decide whether or not to exercise the option, based on the history of the stock price so far. It so happens that the arbitrage-free value of an american-style option at node  $v$  can be expressed recursively in a manner similar to the european-case. For any on the binomial-tree,

$$V(v) = \begin{cases} \max \left\{ G(v), \frac{pV(v^+) + (1-p)V(v^-)}{R} \right\} & \text{if } v \text{ is a non-terminal vertex,} \\ G(v) & \text{if } v \text{ is a terminal vertex.} \end{cases}$$

The optimal exercise strategy (i.e. the one that gives the maximum expected payoff) for the owner of the option can be found by backward-recursive computation – while moving forward on a path from the root node, if the current node is  $v$ , exercise immediately if and if  $V(v) = G(v)$ , that is, if

$$G(v) \geq \frac{pV(v^+) + (1-p)V(v^-)}{R}.$$



That is, the optimal exercise point is the earliest node  $v$  where the option payoff  $G(v)$  from immediate exercise equals the option value  $V(n)$ . The C++ code that I wrote called `american_option_pricing_by_binomial_model.cpp` computes the value of an American Option using the binomial model. A sample output is shown in figure 5.

```

Terminal — bash — 80x22
vpn3-14487:Debug sreenivas$ ./American\ Option\ Binomial\ Model 1.0 30 0.1 0.3 1
00 100
Recursive Binomial American-Asian Option Pricing
Expiration Time (Years) = 1
Number of Divisions = 30
Risk Free Interest Rate = 0.1
Volatility (%age of stock value) = 30
Initial Stock Price = 100
Strike Price = 100
-----
Up Factor = 1.0563
Uptick Probability = 0.516775
-----
Binomial Price of an American Call Option = 16.6354
Binomial Price of an American Put Option = 8.29462
-----
Let us verify the Put-Call Parity: S+P-C = Kexp(-r*T) for American Options
100 + 8.29462 - 16.6354 = 100exp(-0.1 * 1)
91.6592 == 90.4837
Looks like Put-Call Parity does NOT hold
-----
vpn3-14487:Debug sreenivas$

```

Figure 5: Sample output of the C++ code `american_option_pricing_by_binomial_model.cpp`.

## 7 Making Binomial (and Trinomial) Models Faster

By now you should be convinced of the programming elegance that recursion brings to the pricing of options – without recursion, we would have a hard time coding the pricing routines. There is one issue that you might have encountered with Binomial (or, Trinomial/Multinomial) models is that the naive implementation takes a long time for problems of modest size, even. For instance, from figure 6 we see that it takes about 1min 54secs to compute the price of an European Option using a 30 stage Binomial model. A similar attempt using a 20 stage Trinomial model takes 3mins 40secs (cf. figure 7. Figures 8 (9) shows that it took the naive 30 stage Binomial (20 stage Trinomial) model 1min 2secs (3mins 1sec) to price an American Option.

We could speed these computations up significantly by using **memoization**. Figure 10 (11) shows that it takes 1.5secs (3.3secs) to compute the price of an European Option using a 5000 stage memoized Binomial (Trinomial) model. Notice how accurately these models match the Black-Sholes price. Figure 12 (13) shows that it takes 3.7secs (7.6secs) to compute the price of an American Option using 5000 stage memoized Binomial (Trinomial) model.

```

mobile-96-74:Debug sreenivas$ time ./European\ via\ Binomial 0.5 30 0.08 0.3 60 50
Recursive Binomial European Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 30
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.03949
Uptick Probability = 0.507539
-----
Binomial Price of an European Call Option = 12.8353
Call Price according to Black-Scholes = 12.8226
-----
Binomial Price of an European Put Option = 0.874851
Put Price according to Black-Scholes = 0.862071
-----
Verifying Put-Call Parity: S+P-C = Kexp(-r*T)
60 + 0.874851 - 12.8353 = 50exp(-0.08 * 0.5)
48.0395 = 48.0395
-----
real    1m54.597s
user    1m54.558s
sys      0m0.037s
mobile-96-74:Debug sreenivas$

```

Figure 6: Running-time illustration for pricing an European Option using a 30 stage (Naive) Binomial Model.

```

Ramavarapus-MacBook-Air:Debug sreenivas$ time ./European\ via\ Trinomial 0.5 20 0.08 0.3 60 50
European Put Option Pricing (Trinomial Model)
Expiration Time (Years) = 0.5
Number of Divisions = 20
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.06938
Uptick Probability = 0.25657
Downtick Probability = 0.243515
Nottick Probability = 0.499915
-----
Trinomial Price of an European Call Option = 12.8311
Call Price according to Black-Scholes = 12.8226
-----
Trinomial Price of an European Put Option = 0.870603
Put Price according to Black-Scholes = 0.862071
-----
real    3m40.109s
user    3m39.889s
sys      0m0.105s
Ramavarapus-MacBook-Air:Debug sreenivas$

```

Figure 7: Running-time illustration for pricing an European Option using a 20 stage (Naive) Trinomial Model.

```

Ramavarapus-MacBook-Air:Debug sreenivas$ time ./American\ Option\ Binomial\ Model 0.5 30 0.08 0.3 60 50
Recursive Binomial American-Asian Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 30
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.03949
Uptick Probability = 0.507539
-----
Binomial Price of an American Call Option = 12.8354
Binomial Price of an American Put Option = 0.907469
-----
Let us verify the Put-Call Parity:  $S+P-C = K\exp(-r \cdot T)$  for American Options
 $60 + 0.907469 - 12.8354 = 50\exp(-0.08 \cdot 0.5)$ 
48.0721 ?=? 48.0395
Looks like Put-Call Parity does NOT hold
-----

real    1m2.624s
user    1m2.529s
sys      0m0.047s
Ramavarapus-MacBook-Air:Debug sreenivas$

```

Figure 8: Running-time illustration for pricing an American Option using a 30 stage (Naive) Binomial Model.

```

mobile-96-74:Debug sreenivas$ time ./American\ Option\ Trinomial\ Model 0.5 20 0.08 0.3 60 50
Recursive Trinomial American Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 20
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
R = 1.002
Up Factor = 1.06938
Uptick Probability = 0.25657
Downtick Probability = 0.243515
Notick Probability = 0.499915
-----
Trinomial Price of an American Call Option = 12.8312
Trinomial Price of an American Put Option = 0.903112

real    3m1.838s
user    3m1.632s
sys      0m0.171s
mobile-96-74:Debug sreenivas$

```

Figure 9: Running-time illustration for pricing an American Option using a 20 stage (Naive) Trinomial Model.

```

mobile-96-74:Debug sreenivas$ time ./European\ via\ Memoized\ Binomial 0.5 5000 0.08 0.3 60 50
(Memoized) Recursive Binomial European Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 5000
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.003
Uptick Probability = 0.500583
-----
Binomial Price of an European Call Option = 12.8227
Call Price according to Black-Scholes = 12.8226
-----
Binomial Price of an European Put Option = 0.862168
Put Price according to Black-Scholes = 0.862071
-----
Verifying Put-Call Parity: S+P-C = Kexp(-r*T)
60 + 0.862168 - 12.8227 = 50exp(-0.08 * 0.5)
48.0395 = 48.0395
-----
real    0m1.516s
user    0m1.414s
sys     0m0.094s
mobile-96-74:Debug sreenivas$

```

Figure 10: Running-time illustration for pricing an European Option using a 5000 stage (Memoized) Binomial Model.

```

mobile-96-74:Debug sreenivas$ time ./European\ via\ Memoized\ Trinomial 0.5 5000 0.08 0.3 60 50
(Memoized) Recursive Trinomial European Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 5000
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.00425
Uptick Probability = 0.250413
-----
Trinomial Price of an European Call Option = 12.8225
Call Price according to Black-Scholes = 12.8226
-----
Trinomial Price of an European Put Option = 0.862011
Put Price according to Black-Scholes = 0.862071
-----
Verifying Put-Call Parity: S+P-C = Kexp(-r*T)
60 + 0.862011 - 12.8225 = 50exp(-0.08 * 0.5)
48.0395 = 48.0395
-----
real    0m3.364s
user    0m3.115s
sys     0m0.203s
mobile-96-74:Debug sreenivas$

```

Figure 11: Running-time illustration for pricing an European Option using a 5000 stage (Memoized) Trinomial Model.

```

mobile-96-74:Debug sreenivas$ time ./American\ Option\ via\ Memoized\ Binomial 0.5 5000 0.08 0.3 60 50
(Memoized) Recursive Binomial American Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 5000
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.003
Uptick Probability = 0.500583
-----
Binomial Price of an American Call Option = 12.8227
Binomial Price of an American Put Option = 0.89684
-----
Let us verify the Put-Call Parity:  $S+P-C = Kexp(-r*T)$  for American Options
 $60 + 0.89684 - 12.8227 = 50exp(-0.08 * 0.5)$ 
48.0741 ?=? 48.0395
Looks like Put-Call Parity does NOT hold
-----

real    0m3.698s
user    0m3.618s
sys     0m0.075s
mobile-96-74:Debug sreenivas$

```

Figure 12: Running-time illustration for pricing an American Option using a 5000 stage (Memoized) Binomial Model.

```

mobile-96-74:Debug sreenivas$ time ./American\ Option\ via\ Memoized\ Trinomial 0.5 5000 0.08 0.3 60 50
(Memoized) Recursive Trinomial American Option Pricing
Expiration Time (Years) = 0.5
Number of Divisions = 5000
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.00425
Uptick Probability = 0.250413
Notick Probability = 0.5
Downtick Probability = 0.249588
-----
Trinomial Price of an American Call Option = 12.8225
Trinomial Price of an American Put Option = 0.896696
-----
Let us verify the Put-Call Parity:  $S+P-C = Kexp(-r*T)$  for American Options
 $60 + 0.896696 - 12.8225 = 50exp(-0.08 * 0.5)$ 
48.0742 ?=? 48.0395
Looks like Put-Call Parity does NOT hold
-----

real    0m7.595s
user    0m7.363s
sys     0m0.207s
mobile-96-74:Debug sreenivas$

```

Figure 13: Running-time illustration for pricing an American Option using a 5000 stage (Memoized) Trinomial Model.

### Reconfirming some material from Finance (Start)

As you can see from the computation results reported in figures 11 and 13, under identical circumstances, the price of an American Call is the same as that of an European Call. It is well-known that this must be the case for a call option that involves a stock that does not pay dividends. This observation is based on the following observation – you will never exercise the call involving a stock that does not pay dividends, in which case it is essentially an European call. On the flip-side, the American call may (or may not) be priced higher than a European call on a stock that pays dividends.

Suppose we have an American call at strike  $K$ , the current value of the (non-dividend-paying) underlying is  $S$ , and the time left to expiration is  $t$ . If you were to exercise the option now, then obviously  $S > K$ , and the realized-profit would be valued at  $(S - K)e^{rt}$  at expiration. If you let the option ride, and waited till expiration your expected profit would be  $E\{S_T\} - K$  at expiration. Since  $E\{S_T\} = Se^{rt}$  (by the Martingale property), we have the expected profit to be  $Se^{rt} - K$ . If you are risk-neutral, then you would exercise the option  $t$  units before expiration only if  $(S - K)e^{rt} > E\{S_T\} - K (= (Se^{rt} - K)) \Rightarrow -Ke^{rt} > -K \Rightarrow Ke^{rt} < K$ , which is impossible. Therefore, you will never exercise the American call on a non-dividend-paying stock before expiration. It is comforting to know that our implementation, which is oblivious to this argument, confirms this – and this should renew your faith in the computational machinery that you are in the process of mastering this semester!

On the flip-side, if the underlying stock pays dividends, then you might think of exercising the American call before expiration (and before the expected dividend-date) if the compounded-growth in the dividend payments makes it look attractive to you. That is, you will exercise the American call if the dividend payment (and its interest) is greater than  $Ke^{rt} - K$ . As a consequence, the American call on this stock will price higher than the corresponding European call.

### Reconfirming some material from Finance (Start)

As you can see, with *recursion* and *memoization*, we can have (1) programming elegance, and (2) computational speed. I would also like to call your attention to the fact that this code can be easily augmented to include issues that are usually ignored in theory (transaction-costs, etc). It is up to you to take it to the “next-level,” if necessary. Hopefully, with these illustrative examples you can see the reason behind my insistence (in programming assignments) that everyone understands these two processes clearly.

Finally, a question that is worth pondering over –

Can we speed up the calculation of an Asian Option using Memoization?

## 8 Option Pricing via Dynamic Programming

We are going to make some simplifying assumptions here. These assumptions can be easily relaxed to cover a realistic scenario. But, first let us get the procedure/algorithm straight. Assume the (underlying) stock price takes on discrete-values over the range  $[1, 2, \dots b]$ . Suppose we have a probabilistic model (i.e. a Markov Chain) that describes the stock behavior. Suppose  $p_{i,j}$  is the probability that the stock will take on a value

quiz

of  $j$  on day  $t + 1$ , given that it was  $i$  on day  $t$ . Note,  $\forall i, \sum_{j=1}^b p_{i,j} = 1$ . Let us say we are interested in calculating the value of an American call option with an exercise price of  $K$  that expires  $T$  days from now, and the current price of the stock is  $S$ . It is important that  $b \times b$  matrix of probabilities be selected carefully to confirm with the martingale/no-arbitrage requirement. There are several candidate probability matrices that meet this requirement. I am giving you one below. You can get others by using the binomial model, for instance.

### A Suggestion for the Probability Matrix (Start)

If you wish, you may think of a trinomial model on  $n$  stages, so the  $n$ -th stage will have  $b = 2n + 1$  states. Under this interpretation, we assign the following stock values to each named-state as follows:

- state 1 has a stock value of  $Su^{-n}$  associated with it,
- state 2 has a stock value of  $Su^{-n+1}$  associated with it,
- .....
- state  $n$  has a stock value of  $Su^{-1}$  associated with it,
- state  $n + 1$  has a stock value of  $S$  associated with it,
- state  $n + 2$  has a stock value of  $Su$  associated with it,
- .....
- state  $b - 1 (= 2n)$  has a stock value of  $Su^{n-1}$  associated with it, and
- state  $b (= 2n + 1)$  has a stock value of  $Su^n$  associated with it.

That is, state  $i$  ( $i \in \{1, 2, \dots, b\}$ ) has a stock value of  $Su^{i-n-1}$  associated with it.

Assuming  $i \neq 1$  and  $i \neq b$  (i.e. we are not at the boundary-states), then the probability of transitioning to state  $j$  ( $j \in \{1, 2, \dots, b\}$ ) is given by the trinomial model as

$$p_{i,j} = \begin{cases} p_d & \text{if } j = i - 1, \\ 1 - p_u - p_d & \text{if } j = i, \\ p_u & \text{if } j = i + 1, \\ 0 & \text{otherwise.} \end{cases}$$

For the sake of completeness, if  $i = b$ <sup>4</sup> and  $i = 1$ <sup>5</sup>, we have

$$p_{b,j} = \begin{cases} 1 - p_d & \text{if } j = b, \\ p_d & \text{if } j = b - 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{and} \quad p_{1,j} = \begin{cases} 1 - p_d & \text{if } j = 1, \\ p_u & \text{if } j = 2, \\ 0 & \text{otherwise.} \end{cases}$$

### A Suggestion for the Probability Matrix (End)

<sup>4</sup>Because there can be no state with a higher stock price associated with it, that can be visited on the next-step from  $i = b$ .

<sup>5</sup>Because there can be no state with a lower stock price associated with it, that can be visited on the next-step from  $i = 1$ .

Following the principles of Dynamic Programming, we work backwards from the final time  $T$ , which corresponds to  $n$ , the number of stages in the model. Let  $\mathbf{V}_m$  be a  $b \times 1$  vector, where  $\mathbf{V}_m(i)$  is the value of the option on stage  $m$  if the price of the stock on that day is  $Su^{i-n-1}$ .

We begin by initializing the value of the call option at the final stage  $\mathbf{V}_n$  as

$$\mathbf{V}_n(i) = \max\{Su^{i-n-1} - K, 0\}.$$

Keep in mind that the above expression represents one row in a column vector of option values that is indexed by the states  $\{1, 2, \dots, b\}$ . That is, we have

$$\underbrace{\begin{pmatrix} V_n(1) \\ \dots \\ V_n(i) \\ \dots \\ V_n(b) \end{pmatrix}}_{\mathbf{V}_n} = \begin{pmatrix} \max\{Su^{-n} - K, 0\} \\ \dots \\ \max\{Su^{i-n-1} - K, 0\} \\ \dots \\ \max\{Su^n - K, 0\} \end{pmatrix}$$

Then for  $t = (n - 1) : -1 : 1$ , we compute  $\mathbf{V}_t$  from  $\mathbf{V}_{t+1}$  using the backward-recursion

$$\mathbf{V}_t(i) = \max \left\{ (Su^{i-n-1} - K), \frac{\sum_{j=1}^b p_{i,j} \mathbf{V}_{t+1}(j)}{R} \right\}. \quad (8)$$

The arguments of the  $\max\{\bullet, \bullet\}$  function are as follows:  $(Su^{i-n-1} - K)$  is what we get if we exercise the option on stage  $t$ , and the other term is the expected gain if we waited another stage (i.e. till  $t + 1$ ). Keep in mind that equation 8 represents one entry in a column vector as shown below in matrix form

$$\begin{aligned} \underbrace{\begin{pmatrix} V_t(1) \\ \dots \\ V_t(i) \\ \dots \\ V_t(b) \end{pmatrix}}_{\mathbf{V}_t} &= \max \left\{ \begin{pmatrix} (Su^{-n} - K) \\ \dots \\ (Su^{i-n-1} - K) \\ \dots \\ (Su^n - K), 0 \end{pmatrix}, \begin{pmatrix} \frac{\sum_{j=1}^b p_{1,j} \mathbf{V}_{t+1}(j)}{R} \\ \dots \\ \frac{\sum_{j=1}^b p_{i,j} \mathbf{V}_{t+1}(j)}{R} \\ \dots \\ \frac{\sum_{j=1}^b p_{b,j} \mathbf{V}_{t+1}(j)}{R} \end{pmatrix} \right\} \\ \Rightarrow \underbrace{\begin{pmatrix} V_t(1) \\ \dots \\ V_t(i) \\ \dots \\ V_t(b) \end{pmatrix}}_{\mathbf{V}_t} &= \max \left\{ \begin{pmatrix} (Su^{-n} - K) \\ \dots \\ (Su^{i-n-1} - K) \\ \dots \\ (Su^n - K), 0 \end{pmatrix}, \frac{\mathbf{\Pi}}{R} \times \underbrace{\begin{pmatrix} V_{t+1}(1) \\ \dots \\ V_{t+1}(i) \\ \dots \\ V_{t+1}(b) \end{pmatrix}}_{\mathbf{V}_{t+1}} \right\}, \quad (9) \end{aligned}$$

where  $\mathbf{\Pi} = [\{p_{i,j}\}_{i=1, j=1}^{i=b, j=b}]$  is the  $b \times b$  (martingale/no-arbitrage) probability matrix. We



have to repeatedly execute the above line till we get  $\mathbf{V}_0$ . That is,

$$\underbrace{\begin{pmatrix} V_n(1) \\ \dots \\ V_n(i) \\ \dots \\ V_n(b) \end{pmatrix}}_{\text{Start Here: } \mathbf{v}_T} \rightarrow \underbrace{\begin{pmatrix} V_{n-1}(1) \\ \dots \\ V_{n-1}(i) \\ \dots \\ V_{n-1}(b) \end{pmatrix}}_{\mathbf{v}_{T-1}} \rightarrow \dots \rightarrow \dots \underbrace{\begin{pmatrix} V_0(1) \\ \dots \\ V_0(i) \\ \dots \\ V_0(b) \end{pmatrix}}_{\text{Finish Here: } \mathbf{v}_0}$$

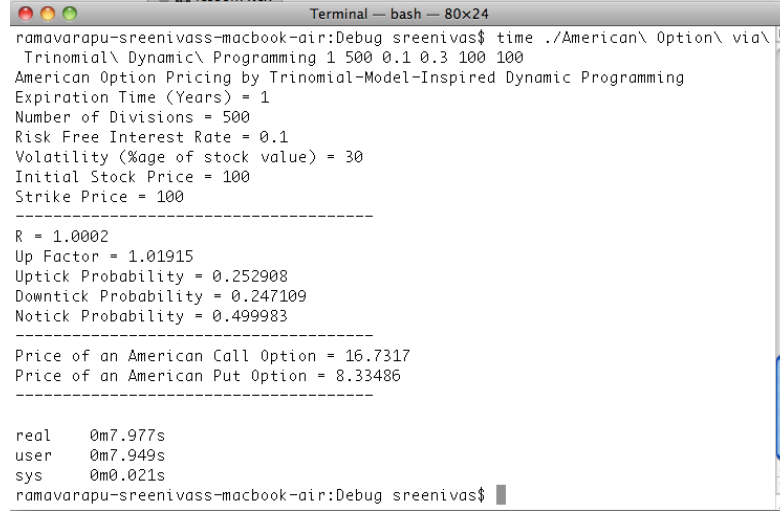
The “middle” entry/term of  $\mathbf{V}_0$  is the price of the option. Specifically, if  $b = 2n + 1$ , then  $V_0(n + 1)$  is the price of the American Option.

Equation 8 (or, 9) will take  $O(b^2)$  time, and since we have to do the calculation of equation 8 for each time step, the entire process of working backwards till we get  $\mathbf{V}_0$  will take  $O(b^2n)$  time.

**American Option Pricing** The C++ program called

`american_option_pricing_by_dynamic_programming.cpp`

prices an american option using the method described above. The probabilities I used in this code are borrowed from the Trinomial Model. A sample run is shown in figure 14. As you can see, this 500-stage problem took 7.977 seconds to compute on my MacBook Air, which is quite good. A recursive trinomial model of a similar size would take several days to run on my machine.



```

Terminal — bash — 80x24
ramavarapu-sreenivass-macbook-air:Debug sreenivas$ time ./American\ Option\ via\
Trinomial\ Dynamic\ Programming 1 500 0.1 0.3 100 100
American Option Pricing by Trinomial-Model-Inspired Dynamic Programming
Expiration Time (Years) = 1
Number of Divisions = 500
Risk Free Interest Rate = 0.1
Volatility (%age of stock value) = 30
Initial Stock Price = 100
Strike Price = 100

-----
R = 1.0002
Up Factor = 1.01915
Uptick Probability = 0.252908
Downtick Probability = 0.247109
Notick Probability = 0.499983
-----
Price of an American Call Option = 16.7317
Price of an American Put Option = 8.33486
-----

real    0m7.977s
user    0m7.949s
sys     0m0.021s
ramavarapu-sreenivass-macbook-air:Debug sreenivas$

```

Figure 14: Sample output of the C++ code `american_option_pricing_by_dynamic_programming.cpp`.

**European Option Pricing:** Suppose we are interested in european-option pricing – we note that we only need to compute  $\mathbf{\Pi}^n \mathbf{V}_T$ , where  $\mathbf{\Pi}$  is the matrix of  $\{p_{i,j}\}$ 's and  $n$  is the total number of stages. We can compute  $\mathbf{\Pi}^n$  in  $O(b^3 \ln n)$  time by repeated

squaring. You could of course improve its performance by using *Strassen's Method* of matrix multiplication along with repeated squaring.

The C++ code `euro_option_pricing_dynamic_prog.cpp` on the *Compass Website* is my attempt at using dynamic programming to price options. If you looked at the code, you will notice that the probability matrix that I have used here is from the Binomial model. A sample output is shown in figure 15.

```

Terminal — bash — 80x23
ramavarapu-sreenivass-macbook-air:Debug sreenivas$ time ./European\ via\ Dyn\ Pr
og\ Binomial 0.5 500 0.08 0.3 60 50
European Put Option Pricing by Dynamic Programming
Expiration Time (Years) = 0.5
Number of Divisions = 500
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.00953
Uptick Probability = 0.501844
-----
Price of an European Call Option = 12.815
Call Price according to Black-Scholes = 12.8226
-----
Price of an European Put Option = 0.85958
Put Price according to Black-Scholes = 0.862071

real    2m44.778s
user    2m44.220s
sys     0m0.505s
ramavarapu-sreenivass-macbook-air:Debug sreenivas$

```

Figure 15: Sample output of the C++ code `euro_option_pricing_dynamic_prog.cpp`, which uses the probability matrix derived from a Binomial Lattice.

To illustrate the point that there can several probability matrices that yield the martingale/no-arbitrage property, I wrote a C++ program

`euro_option_pricing_dynamic_prog_alt.cpp`,

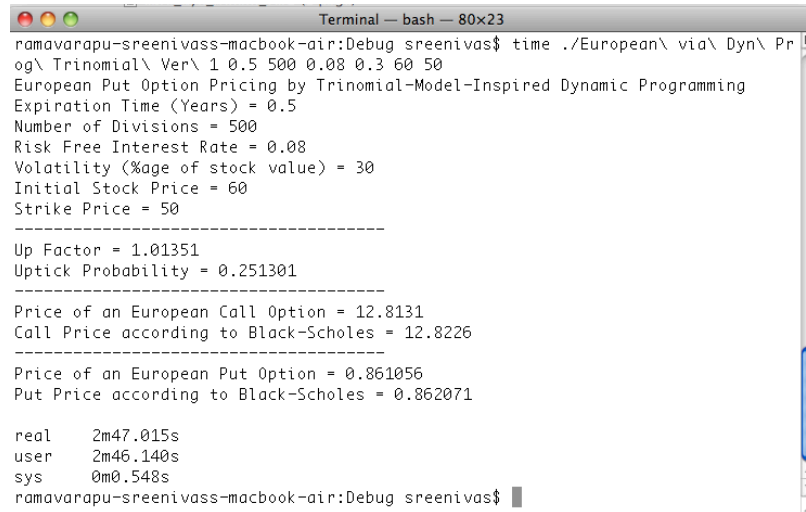
that you can find on the *Compass* website. This program uses a probability matrix that is derived from a (slightly truncated version of a) Trinomial Lattice. The number of states  $b = 2n - 1$  for the probability matrix used in this program. A sample output is shown in figure 16. For what ever it is worth, the Put price is closer to the Black-Scholes formula (as compared to what you can see in figure 15), but it takes a tad longer to run.

To overstate the case, I wrote yet another C++ program

`euro_option_pricing_dynamic_prog_alt_alt.cpp`

and uploaded it to the *Compass* website. This program uses the same probability matrix as the one used for pricing the American options. For this probability matrix  $b = 2n + 1$ . The sample output is shown in figure 17. This takes a little longer to run than the previous code (cf. figure 16), but there really is no major improvement in the quality of the final solution. The message here is this – there are several probability matrices that

work (i.e. give you the martingale/no-arbitrage property), and we might have to pick the one that works the best. Even after all the work, you might not get a significant improvement – sometimes, you have to learn when to stop over doing it!



```

Terminal — bash — 80x23
ramavarapu-sreenivass-macbook-air:Debug sreenivas$ time ./European\ via\ Dyn\ Pr
og\ Trinomial\ Ver\ 1 0.5 500 0.08 0.3 60 50
European Put Option Pricing by Trinomial-Model-Inspired Dynamic Programming
Expiration Time (Years) = 0.5
Number of Divisions = 500
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.01351
Uptick Probability = 0.251301
-----
Price of an European Call Option = 12.8131
Call Price according to Black-Scholes = 12.8226
-----
Price of an European Put Option = 0.861056
Put Price according to Black-Scholes = 0.862071

real    2m47.015s
user    2m46.140s
sys      0m0.548s
ramavarapu-sreenivass-macbook-air:Debug sreenivas$

```

Figure 16: Sample output of the C++ code `euro_option_pricing_dynamic_prog_alt.cpp`, which uses the probability matrix derived from a (slightly truncated) Trinomial Lattice.

## 9 Pricing Options by Linear Programming

The material presented here is a simplified form of what is in Edirisinghe et al's work on optimal replication of Options when there are transaction costs and other restrictions [3]. It must be noted that transaction costs are almost always ignored in the published material you will see frequently in this area. The method that is presented in this paper is quite general and powerful. But, I am going to use it to price European Options<sup>6</sup>, and for this I have to ignore transaction costs (because Black-Scholes do not take it into consideration anyways).

### 9.1 Pricing European Options using Linear Programming

To understand what is going on in Edirisinghe et al's paper [3], you might first want to look at the two-stage binomial example described below. At each stage  $(k, i)$  (of the discrete-time economy) we suppose we have a replication portfolio consisting of  $x_j$  shares of the stock (with the proviso that if  $x_j$  is negative, you are shorting it) and  $y_j$  of cash. The interpretation is that, if  $y_k$  is positive (negative) you are effectively lending (borrowing) money at the risk-free rate. There might be change in your position as the economy jumps from one discrete-state to another – this would mean you have to take

<sup>6</sup>This way we can compare our results with the Black-Scholes formula to see how well we are doing!

```

ramavarapu-sreenivass-macbook-air:Debug sreenivas$ time ./European\ Option\ by\
Dynamic\ Prog 0.5 500 0.08 0.3 60 50
European Put Option Pricing by Trinomial-Model-Inspired Dynamic Programming
Expiration Time (Years) = 0.5
Number of Divisions = 500
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Up Factor = 1.01351
Uptick Probability = 0.251301
-----
Price of an European Call Option = 12.8131
Call Price according to Black-Scholes = 12.8226
-----
Price of an European Put Option = 0.861056
Put Price according to Black-Scholes = 0.862071

real    2m50.930s
user    2m49.914s
sys     0m0.555s
ramavarapu-sreenivass-macbook-air:Debug sreenivas$

```

Figure 17: Sample output of the C++ code `euro-option-pricing-dynamic-prog-alt-alt.cpp`, which uses the probability matrix derived from a Trinomial Lattice used when we priced the American Option.

appropriate actions to adjust your portfolio – we are assuming these adjustments do not incur any cost for the present. As noted earlier, this is because Black-Scholes do not take transaction costs into effect either.

Edirisinghe et al. assume there is a transaction cost, and modify the LP formalism accordingly – this is their main contribution. They suppose that if you sell a stock at price  $S_k$ , due to transaction costs, you will only net  $S_k(1 - \theta)$ . Likewise, if you buy a stock at price  $S_k$ , due to transaction costs, you will have to pay  $S_k(1 + \theta)$ . They have a clever way of handling these restrictions as linear constraints in an appropriately posed LP formalism (cf. Proposition 1; proof is in the Appendix of their paper; [3]). Regrettably, we are going to overlook the ingenuity of their work by assuming  $\theta = 0$  to be able to compare results with the Black-Scholes formula.

Consider the two-stage binomial lattice shown in figure 18. Suppose we are interested in a Call Option that expires at the end of the second-stage with some strike price  $K$ . The terminal payoffs at the final stage are  $(2, 2) \Leftrightarrow \max\{0, Su^2 - K\}$ ,  $(2, 0) \Leftrightarrow \max\{0, S - K\}$ , and  $(2, 2) \Leftrightarrow \max\{0, Su^{-2} - K\}$ . Edirisinghe et al. compute the smallest value of the portfolio in the state  $(0, 0)$  that guarantees the portfolio is self-financing at each stage (of the discrete-time economy).

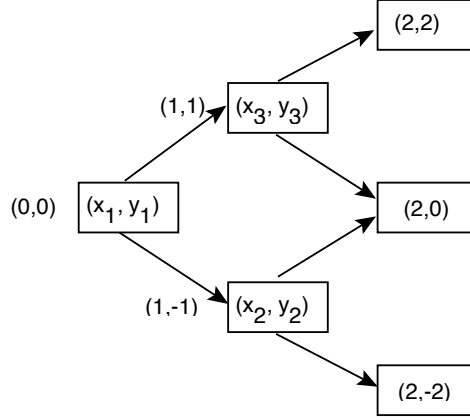


Figure 18: A Two-Stage Binomial Lattice used to illustrate the idea of replication portfolios.

This is equivalent to the following optimization problem:

$$\begin{aligned}
 & \text{minimize : } Sx_1 + y_1 \\
 & \text{subject to} \\
 & (0,0) \rightarrow (1,1): \quad Sux_1 + Ry_1 \geq Sux_3 + y_3 \\
 & (0,0) \rightarrow (1,-1): \quad Su^{-1}x_1 + Ry_1 \geq Su^{-1}x_2 + y_2 \\
 & (1,1) \rightarrow (2,2): \quad Su^2x_3 + Ry_3 \geq \max\{0, Su^2 - K\} \\
 & (1,1) \rightarrow (2,0): \quad Sx_3 + Ry_3 \geq \max\{0, S - K\} \\
 & (1,-1) \rightarrow (2,0): \quad Sx_2 + Ry_2 \geq \max\{0, S - K\} \\
 & (1,-1) \rightarrow (2,-2): \quad Su^{-1}x_2 + Ry_2 \geq \max\{0, Su^{-2} - K\}
 \end{aligned}$$

Some of the variables (specifically  $y_i$ 's) will be negative if we solved the above optimization problem. We can however make some changes to conform to the standard LP formulation where all variables are required to be positive:

$$\begin{aligned}
 & \text{minimize : } Sx_1 - y_1 \\
 & \text{subject to} \\
 & (0,0) \rightarrow (1,1): \quad Sux_1 - Ry_1 \geq Sux_3 - y_3 \\
 & (0,0) \rightarrow (1,-1): \quad Su^{-1}x_1 - Ry_1 \geq Su^{-1}x_2 - y_2 \\
 & (1,1) \rightarrow (2,2): \quad Su^2x_3 - Ry_3 \geq \max\{0, Su^2 - K\} \\
 & (1,1) \rightarrow (2,0): \quad Sx_3 - Ry_3 \geq \max\{0, S - K\} \\
 & (1,-1) \rightarrow (2,0): \quad Sx_2 - Ry_2 \geq \max\{0, S - K\} \\
 & (1,-1) \rightarrow (2,-2): \quad Su^{-1}x_2 - Ry_2 \geq \max\{0, Su^{-2} - K\} \\
 & \text{positivity constraint} \quad \forall i \in \{1, 2, 3\}, x_i \geq 0, y_i \geq 0.
 \end{aligned}$$

If you ran this for  $u = 1.25$  and  $R = 1.07$ , using `lp.solve` you would have an input file that looks like what is shown in figure 19, the resulting output is shown in figure 20. The value of this call option is therefore  $100x_1^* - y_1^* = 100 \times 0.700935 - 52.4063 = 17.6871$ .

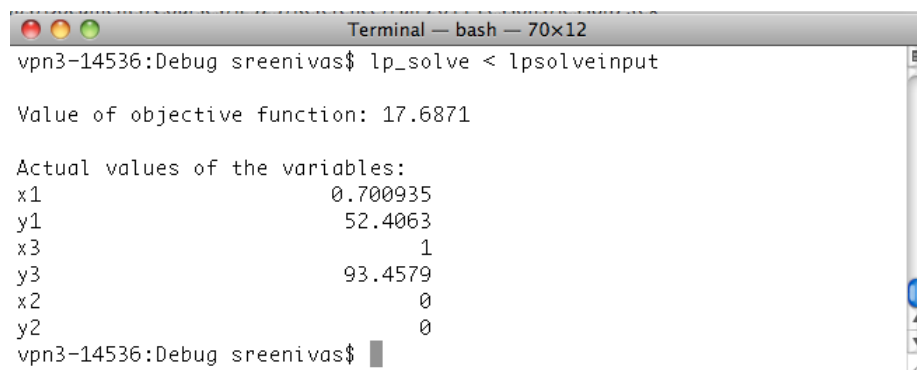
---

```
min: 100x1 - y1;  
125x1 - 1.07y1 >= 125x3 - y3;  
80x1 - 1.07y1 >= 80x2 - y2;  
156.25x3 - 1.07y3 >= 56.25;  
100x3 - 1.07y3 >= 0;  
100x2 - 1.07y2 >= 0;  
64x2 - 1.07y2 >= 0;
```

---

Figure 19: `lpsolveinput`: Input File for `lp_solve` for the 2-stage problem shown in figure 18 with  $u = 1.15$ ,  $R = 1.07$ ,  $S = K = 100$ .

---

A terminal window titled "Terminal — bash — 70x12" showing the execution of the lp\_solve command. The user input is "lp\_solve < lpsolveinput". The output shows the value of the objective function and the actual values of the variables x1, y1, x3, y3, x2, and y2.

```
vpn3-14536:Debug sreenivas$ lp_solve < lpsolveinput  
  
Value of objective function: 17.6871  
  
Actual values of the variables:  
x1          0.700935  
y1          52.4063  
x3           1  
y3         93.4579  
x2           0  
y2           0  
vpn3-14536:Debug sreenivas$
```

Figure 20: The output of `lp_solve` for the input file shown in figure 19.

The LP that computes the value of the Put Option is:

$$\begin{aligned}
 & \text{minimize : } S \times (-x_1) + y_1 && \text{subject to} \\
 (0,0) \rightarrow (1,1): & -Sux_1 + Ry_1 \geq -Sux_2 + y_2 \\
 (0,0) \rightarrow (1,-1): & -Sux_1 + Ry_1 \geq -Su^{-1}x_3 + y_3 \\
 (1,1) \rightarrow (2,2): & -Su^2x_1 + Ry_1 \geq \max\{0, K - Su^2\} \\
 (1,1) \rightarrow (2,0): & -Sx_1 + Ry_1 \geq \max\{0, K - S\} \\
 (1,-1) \rightarrow (2,0): & -Sx_2 + Ry_2 \geq \max\{0, K - S\} \\
 (1,-1) \rightarrow (2,-2): & -Su^{-1}x_2 + Ry_2 \geq \max\{0, K - Su^{-2}\} \\
 \text{positivity constraint} & \forall i \in \{1, 2, 3\}, x_i \geq 0, y_i \geq 0.
 \end{aligned}$$

Notice, that to get the positivity constraint to work, we have used the fact that the  $x_i$ 's are negative and  $y_i$ 's are positive in the equivalent portfolio for a Put Option.

This procedure can be generalized to more than two stages. I wrote a C++ program `european_option_via_lp.cpp` that uses this approach to compute the value of an European Option. A sample output is shown in figure 21.

```

Terminal — bash — 73x19
vpn3-14536:Debug sreenivas$ ./European\ via\ LP 0.5 10 0.08 0.3 50 60
-----
European Option Pricing via Linear Programming
Expiration Time (Years) = 0.5
Number of Divisions = 10
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 50
Strike Price = 60
R = 1.00401
Up-Factor = 1.06938
-----
Call Price according the LP formulation = 1.73906
Call Price according to Black-Scholes = 1.70357
-----
Put Price according the LP formulation = 9.3864
Put Price according to Black-Scholes = 9.35093
-----
vpn3-14536:Debug sreenivas$

```

Figure 21: Sample output of `european_option_via_lp.cpp`.

This program has to be linked with the `lp_solve55` library, as I am using the `lp_solve` API to solve the LP. The constraints are written into the appropriate `lprec` pointer via a recursive procedure as a row with appropriate entries. There is a lot of thought that went into the indexing scheme for each of these constraint-rows that you will have to digest/infer after looking at my code. The final LP is solved to find the price of the option.

Edirisinghe et al. show how this LP formalism can be extended to account for transaction costs. I leave that as an exercise for you to complete at leisure.

## 10 Pricing Options using the Fast Fourier Transform (FFT)

In 1999, Carr and Madan [2] introduced a new approach to computing the price of an option. This numerical method prices options using the *characteristic function* of the underlying stock's price movement process, and is based on a well-understood approach (in the signal-processing area) called the *Fast Fourier Transform* (FFT). This algorithm has been used extensively in the processing/filtering of image and voice signals. There are many efficient implementations of the FFT algorithm, and we will be using the one that comes with the *NEWMAT* library. Keep in mind that the C++ code that uses the FFT procedure has to include the *newmatap.h* file. We will not go over the FFT algorithm, as we do not have the time – but, you should keep in mind that this algorithm has served as the backbone of several applications (like voice-recognition in your *iPhone*, etc). It is amazing that this “work-horse” has found use in Financial Engineering!

There is a significant amount of background material in Carr and Madan's paper [2], and I will not go over the motivation/reasoning behind it, as we do not have the time for it. If you have questions after you read the paper, I will be happy to answer any questions you might have. The first thing you should note is that there is an expression for the call price (equation 5, [2])

$$C_T(k) = \frac{e^{-\alpha k}}{\pi} \int_0^\infty e^{-ivk} \psi_T(v) dv,$$

where  $k = \log K$  and  $\alpha$  is a positive real number<sup>7</sup>,

$$\psi_T(v) = \frac{e^{-rT} \Phi_T(v - (\alpha + 1)i)}{\alpha^2 + \alpha - v^2 + i(2\alpha + 1)v} \text{ (equation 6, [2])},$$

and

$$\Phi_T(x) = e^{-\frac{T\sigma^2 x^2}{2} + i[\log S + (r - \sigma^2)T]x}.$$

They note that the integration that is required for computing  $C_T(k)$  is essentially a **Fourier Transform** (with appropriate changes). Loosely speaking, you could approximate the continuous integral by a discrete-sum (i.e. you are doing numerical quadrature), which is essentially the *Fast Fourier Transform* (FFT). There are plenty of small details that have to be worked out before this is fait accompli – these can be found in the paper. When the dust settles after all the effort has been completed, you will end up with equation 24. This equation says the price of an European Call Option is given by the expression

$$C(k_u) = \frac{e^{-\alpha k_u}}{\pi} \sum_{j=1}^N \overbrace{e^{-i \frac{2\pi}{N} (j-1)(u-1)}}^{\text{FFT of } \{\mathbf{A}\}_{j=1}^N} \underbrace{\left\{ e^{ibv_j} \psi_T(v_j) \frac{\eta}{3} (3 + (-1)^j - \delta_{j-1}) \right\}}_{\mathbf{A}_j}.$$

---

<sup>7</sup>In my implementation,  $\alpha = 5$ .



The objective is to create an  $N$ -long string of (complex-valued) numbers  $\{\mathbf{A}_i\}_{i=1}^N$  that is presented as input to the FFT function. Technically, the FFT routine will output another  $N$ -long string of (complex-valued) numbers. It can be shown that the first term in this string will always be real-valued, and it equals the summation shown above. We then have to multiply this first term with  $\frac{e^{-\alpha k_u}}{\pi}$ , and we have (an approximation to) the price of the Call option at price  $k_u = \log K$ . This approximation can be made better by increasing the grid-size<sup>8</sup>. There are some additional theoretical issues about numerical quadrature – Carr and Madan use something called **Simpsons Rule** to deal with it, and you will have to read the paper to get the fine-print about it.

The gist of Carr and Madan’s paper is this – they have used the well-known *Fast Fourier Transform* (FFT) algorithm to simplify the computation of the summation shown above. You will have to go over my C++ code to get a firm handle on the use of the FFT algorithm from the NEWMAT library.

My C++ program is called `european_put_via_fft.cpp`. I have put this up on the *Compass* site, and it implements the above formula. A couple of sample outputs of this program are presented in figures 22 and 23. Note that it hardly takes any time to compute the price of an option, and it is quite accurate too. Interestingly, the pricing routine for a Put option essentially involves using the same code as the one that prices the Call option, but we have to change the sign of  $\alpha$ . You will have to figure the reasons behind this, on your own...

```

Terminal — bash — 83x20
vpn3-144157:Debug sreenivas$ time ./European\ via\ FFT 1 32 0.1 0.25 100 100
European Put Option Pricing by FFT
Expiration Time (Years) = 1
Grid Size = 32
Risk Free Interest Rate = 0.1
Volatility (%age of stock value) = 25
Initial Stock Price = 100
Strike Price = 100
-----
Price of an European Call Option Via Carr & Madan's FFT method = 14.9758
Price of an European Call from the Black-Scholes Formula = 14.9758
-----
Price of an European Put Option Via Carr & Madan's FFT method = 5.45953
Price of an European Put from the Black-Scholes Formula = 5.45954
-----
real    0m0.020s
user    0m0.001s
sys     0m0.004s
vpn3-144157:Debug sreenivas$

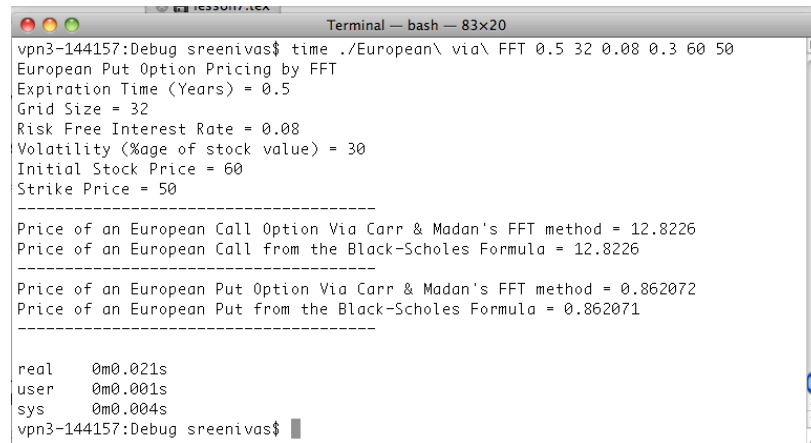
```

Figure 22: Sample output of `european_put_via_fft.cpp`.

## References

- [1] M. Baxter and A. Rennie. *Financial Calculus: An Introduction to Derivative Pricing*. Cambridge University Press, Cambridge, UK, 1996.

<sup>8</sup>You have to look this stuff up in the paper – it has to do with how we approximate the continuous integral with a numerical quadrature, which has the FFT algorithm as its foundation.

A terminal window titled "Terminal — bash — 83x20" showing the execution of a program. The user enters a command to run a program with various parameters. The output displays the option pricing results for both European Call and Put options using Carr & Madan's FFT method and the Black-Scholes Formula, along with execution time statistics.

```
vpn3-144157:Debug sreenivas$ time ./European\ via\ FFT 0.5 32 0.08 0.3 60 50
European Put Option Pricing by FFT
Expiration Time (Years) = 0.5
Grid Size = 32
Risk Free Interest Rate = 0.08
Volatility (%age of stock value) = 30
Initial Stock Price = 60
Strike Price = 50
-----
Price of an European Call Option Via Carr & Madan's FFT method = 12.8226
Price of an European Call from the Black-Scholes Formula = 12.8226
-----
Price of an European Put Option Via Carr & Madan's FFT method = 0.862072
Price of an European Put from the Black-Scholes Formula = 0.862071
-----
real    0m0.021s
user    0m0.001s
sys     0m0.004s
vpn3-144157:Debug sreenivas$
```

Figure 23: Yet another sample output of `european_put_via_fft.cpp`.

- [2] P. Carr and D.B. Madan. Option valuation using the fast fourier transform. *Journal of Computational Finance*, 3:463–520, 1999.
- [3] C. Edirisinghe, V. Naik, and R. Uppal. Optimal Replication of Options with Transactions Costs and Trading Restrictions. *Journal of Financial and Quantitative Analysis*, 28(1):117–138, 1993.
- [4] G.M. Jabbour, M.V. Kramin, and S.D. Young. Two-State Option Pricing: Binomial Models Revisited. *Journal of Futures Markets*, 21(11):987–1001, 2001.
- [5] Nassim Taleb. *Dynamic Hedging: Managing Vanilla and Exotic Options*. John Wiley & Sons, 1996.
- [6] P. Wilmott and S. Howison and J. Dewynne. *The Mathematics of Financial Derivatives: A Student Introduction*. Cambridge University Press, 2009.