Columbia Business School

Numerical Option Pricing

In these notes, we first give some background on option pricing theory and then present the main numerical methods for pricing complex options — the binomial method and simulation. Both numerical methods draw on a fundamental result from the underlying theory: if a derivative security can be perfectly hedged by trading in other assets, then its price is the expected present value of its cash flows based on *risk-neutral* probabilities. The binomial method and simulation are two techniques for computing these expected present values.

We begin in Section 1 with a simple example to motivate the concepts of relative pricing and risk-neutral probabilities. We then show how these concepts lead to the Black-Scholes formula under a continuous model of asset prices. We also introduce the discrete binomial approximation for asset prices.

Although many options can be priced using variants of the Black-Scholes formula, there are many others for which no pricing formula exists, so a computational method is required. Section 2 introduces the binomial method. It starts with simple European calls and puts, then extends the method to American and *exotic* options. Exotics include options with *path-dependent* payoffs and options on multiple underlying assets. Section 3 discusses option pricing by simulation, and the relative merits of simulation and the binomial method.

1. Review of Option Pricing Theory

1.1. Relative Valuation and Risk-Neutral Probabilities

The defining feature of a derivative security is that its cash flows are *derived* from underlying assets. The payoff of a stock option, for example, is determined by the price evolution of the underlying stock. If all the underlying variables to which a derivative security is tied correspond to actively traded assets, it is typically possible to replicate the security's cash flows by trading in the underlying assets. A stock option, for example, can be replicated by trading in the underlying stock and borrowing and lending cash. If a derivative security and a replicating portfolio of underlying assets generate the same cash flows, the absence of arbitrage requires that they have the same price. Thus, a derivative security can be valued by valuing a portfolio that replicates its cash flows. To put it another way, derivative securities can be priced *relative* to other securities in the market. This is the basis of both theoretical and computational models for pricing derivatives.

Before discussing real securities, we will illustrate the notion of relative pricing in detail through a simple example. Let q denote the chance of rain tomorrow, and to be concrete let us suppose $q = 0.30$. Suppose the alternative to "RAIN" is "SHINE," so we may describe our uncertainty about the future through the following diagram:

$$\begin{tikzcd}
 & \bullet\text{RAIN} \\
\bullet \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\
 & \bullet\text{SHINE} \\
\text{Today} & \text{Tomorrow}
\end{tikzcd}$$

$$q = 0.30$$
$$1 - q = 0.70$$

Now consider a security that will pay \$4 in the event of RAIN and \$1 in the event of SHINE. We may illustrate its payoffs tomorrow and its price today through the following diagram:

$$\begin{tikzcd} & 4 \\ \text{Price} = D \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\ & 1 \end{tikzcd} \qquad (1)$$

Security D is a type of derivative security whose payoff is determined by the weather.[1] What would the market price of such a security be? In attempting to answer this question, you might be led to compute the expected value of the security's payoffs. This is just

$$\text{Expected value} = q \cdot (4) + (1 - q) \cdot (1) = .30(4) + .70(1) = 1.90; \qquad (2)$$

i.e., the expected value of the payoffs is \$1.90. But this is not the same as the price of the security because the expected value calculation ignores two factors: the time-value of money and the risk-aversion of investors. The first issue refers to the fact that the payoffs will be received in the future, and are therefore worth less than immediate payoffs. This issue is easily addressed by discounting. To separate the second issue, suppose for a moment that payoffs are to be received immediately, so that no discounting is necessary. A risk-averse investor would still prefer a sure \$1.90 over a 30% chance at \$4.00 and a 70% at \$1.00. Consequently, the security D must sell for less than \$1.90, even if payoffs are received immediately, to compensate investors for taking on risk.

We cannot determine the price of the security solely from (1) because we cannot determine how the market prices the risk in D solely from (1). But it might be possible to get around this problem by observing the prices of other securities in the market whose payoffs are determined by tomorrow's weather. Suppose, in particular, that there are two securities B_1 and B_2 with the following payoffs and prices:

$$\begin{tikzcd} & 1 & & & 0 \\ B_1 = 0.25 \arrow[ur, "q"] \arrow[dr, "1-q"'] & & & B_2 = 0.65 \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\ & 0 & & & 1 \end{tikzcd}$$

Thus, B_1 pays \$1 in case of RAIN, and B_2 pays \$1 in case of SHINE. The market prices of these securities are \$0.25 and \$0.65 respectively. These prices are simply observed in the market; they result from investor preferences but beyond that we cannot say how they are determined. We will see that the prices of B_1 and B_2 give us just enough information to price D *relative* to these securities.

First observe that from B_1 and B_2 we can determine the risk-free interest rate. A portfolio

[1]We often use the same letter, in this case D, to refer to both a security and its price.

holding one share of each of B_1 and B_2 pays one dollar tomorrow, regardless of the weather:

$$\begin{tikzcd}
 & 1 \\
B_1 + B_2 \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\
 & 1
\end{tikzcd}$$

This portfolio is thus equivalent to a zero coupon bond maturing tomorrow with a face value of \$1. To preclude arbitrage, the price of this portfolio (equivalently, the zero coupon bond) must be the sum $B_1 + B_2$ of the prices of constituent securities, which in our example is $\$0.25 + \$0.65 = \$0.90$. If we let the R denote the risk-free interest rate over one day, then by definition,

$$\text{Price of zero coupon bond} = \frac{1}{1+R}.$$

But then

$$B_1 + B_2 = \frac{1}{1+R}, \tag{3}$$

so

$$R = \frac{1}{B_1 + B_2} - 1 = \frac{1}{.90} - 1 = 11.1\%.$$

Now let's go back to pricing D. Observe that D has exactly the same payoffs as a portfolio holding four shares of B_1 and one share of B_2. To preclude arbitrage, the price of D must equal the value of the portfolio:

$$\begin{aligned} D &= (4)B_1 + (1)B_2 \qquad (4) \\ &= (4)(0.25) + (1)(0.65) \\ &= 1.65 \end{aligned}$$

Thus, simply by ruling out arbitrage we are able to price D relative to observed prices in the market. If D sold at a higher price, buying the portfolio and shorting D would result in riskless profits; if D sold at a lower price, riskless profits would result from the reverse trade.

The form of the payoffs of B_1 and B_2, though convenient, is not essential. Suppose we had instead started from securities

$$\begin{tikzcd}
 & 2 & & 1 \\
B_1' = 2.45 \arrow[ur, "q"] \arrow[dr, "1-q"'] & & B_2' = 0.90 \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\
 & 3 & & 1
\end{tikzcd}$$

The prices indicated are consistent with those of B_1 and B_2. In particular, $B_1' = 2B_1 + 3B_2$, and $B_2' = B_1 + B_2$. The cash flows of D are replicated by a portfolio short three shares of B_1' and long ten shares of B_2'. The price D must then be

$$\begin{aligned} D &= (-3)B_1' + (10)B_2' \\ &= (-3)(2.45) + (10)(0.90) \\ &= 1.65 \end{aligned}$$

We arrive at the same price because the prices of B_1' and B_2' were chosen to be consistent with those of B_1 and B_2.

Notice that the value \$1.65 determined by relative pricing is different from the expected value \$1.90 of the payoff found in (2). Indeed, even if we discount to find the expected *present* value, we get

$$\begin{aligned}\text{Expected present value} &= \frac{q(\text{RAIN payoff}) + (1-q)(\text{SHINE payoff})}{1+R} \qquad (5)\\ &= \frac{1.90}{1.111} = 1.71,\end{aligned}$$

which differs from the price \$1.65. Let us revisit (4) from this perspective, writing it now as follows:

$$\begin{aligned}D &= (4)B_1 + (1)B_2\\ &= \left(4\frac{B_1}{B_1+B_2} + 1\frac{B_2}{B_1+B_2}\right)(B_1+B_2)\\ &= \left(4\frac{B_1}{B_1+B_2} + 1\frac{B_2}{B_1+B_2}\right)\frac{1}{1+R}. \qquad (6)\end{aligned}$$

The last step uses (3). If we set

$$p = \frac{B_1}{B_1+B_2} = \frac{0.25}{0.25+0.65} = 0.278, \quad 1-p = \frac{B_2}{B_1+B_2} = \frac{0.65}{0.65+0.25} = 0.722$$

then we can rewrite (6) as

$$\begin{aligned}D &= ((4)p + (1)(1-p))\frac{1}{1+R} \qquad (7)\\ &= \frac{(4)(0.278)+(1)(0.722)}{1.111} = 1.65.\end{aligned}$$

Comparing (7) with (5) we conclude that the price \$1.65 of the security is the expected present value of its payoffs, *provided* the expectation is computed using the new weights p and $1-p$ rather than the original probabilities q and $1-q$. In fact, it is not necessary to know the original probabilities q and $1-q$ to compute the price of D.

The new weights p and $1-p$ are called *risk-neutral probabilities.*[2] To see why, let us compute the expected rate of return on D based on the risk-neutral probabilities:

$$\begin{aligned}\text{Expected rate of return} &= \frac{\text{Expected payoff} - \text{Current Price}}{\text{Current Price}}\\ &= \frac{[p(4)+(1-p)(1)] - 1.65}{1.65} \qquad (8)\\ &= \frac{[0.278(4)+0.722(1)] - 1.65}{1.65} = 11.1\% \qquad (9)\end{aligned}$$

[2] In this note, we use q to represent a real-world probability and p to represent a risk-neutral probability. Other books often do the opposite.

In particular, the expected rate of return on D, based on risk-neutral probabilities is equal to the risk-free rate R. Still using risk-neutral probabilities, the expected rate of return on B_1 is

$$\frac{[0.278(1) + (0.722)(0)] - 0.25}{0.25} = 11.1\%$$

and that on B_2 is

$$\frac{[0.278(0) + (0.722)(1)] - 0.65}{0.65} = 11.1\%$$

Indeed, in a risk-neutral world, all assets have the same rate of return. In contrast, in the real world different assets have different rates of return because of investor attitudes towards risk. To compute the expected rate of return of D in the real world, we would use q in place of p in (8):

$$\frac{[0.30(4) + 0.70(1)] - 1.65}{1.65} = 15.2\%.$$

The actual expected rate of return on D is therefore greater than the risk-free rate of 11.1%. But, again, if we want to price D as the expected present value of its cash flows, we use the risk-neutral probabilities, not the real probabilities.

Although this example is simple and obviously artificial, its essential features hold much more generally and underlie virtually all pricing of derivative securities. The crucial points are these:

1. If the cash flows of a security can be replicated by a portfolio, then to preclude riskless profits the value of the security must equal the value of the portfolio. In this sense, a derivative security can be priced *relative* to market prices of other securities.

2. When a derivative security can be replicated by trading in other assets, its price is the expected present value of its cash flows, the expectation taken with respect to risk-neutral probabilities.

3. The risk-neutral probabilities are the ones that give all assets the same expected rate of return, which is then the risk-free rate.

1.2. Review of the Black-Scholes Model

The payoff of the security D discussed in the previous section is contingent on the outcome of the weather. The payoff of a stock option is similarly contingent on the outcome of future stock prices. A *European* option can be exercised only at a fixed expiration date T, so its payoff is determined by the price of the underlying stock at time T. In contrast, an *American* option may be exercised at any time in the life of the option; its payoff therefore depends on the evolution of the stock price between the time the option is initiated and when it expires. We begin by considering European options.

Option payoffs

A *call* grants its owner the right to buy shares of the underlying stock at a predetermined price, called the *strike*. Let S_t denote the stock price at time t. Time $t = 0$ denotes the present and $t = T$

is the option's expiration time (in years). Let K denote the strike price. If at time T the stock price is above the strike, the owner of a call may exercise the option to buy at K and then immediately sell at the market price of S_T to realize a profit of $S_T - K$. If instead the stock price is below the strike at time T, there is no incentive to exercise the call and the option expires worthless. Thus, we have

$$\text{Call option payoff} = \begin{cases} S_T - K, & \text{if } S_T > K; \\ 0, & \text{otherwise.} \end{cases}$$

The two cases can be summarized in the following formula:[3]

$$\text{Call option payoff} = \max\{S_T - K, 0\}.$$

This formula gives the *intrinsic value* of the option at expiration, More generally, the intrinsic value at an arbitrary time t in the life of the option is $\max\{S_t - K, 0\}$.

A *put* grants the right to *sell* shares of the underlying at the strike price. Thus, a put will be exercised if the market price at time T is *below* the strike K. Its intrinsic value at expiration is given by $\max\{K - S_T, 0\}$.

The formulas for the intrinsic value of calls and puts determine how option payoffs are derived from stock prices, just as (1) determines how the payoffs on security D are derived from the weather. In (1) we had a model of the possible outcomes of the weather and their probabilities. To value calls and puts, we need a corresponding model of the possible outcomes of S_T and their probabilities.

A model of stock-price movements

What are the possible values of the terminal stock price S_T? A reasonable model specifies that all values between 0 and ∞ are possible. Certainly, the stock price cannot drop below 0; this reflects the limited liability of the shareholders of a corporation. And though extremely high prices may be very unlikely, it seems artificial to specify in advance any maximum value for the stock price. Another consideration is that real prices typically move in increments no smaller than 1/8 of a dollar; but these increments are sufficiently small that allowing a continuum of values for S_T is a reasonable approximation.

Having specified that the possible values of S_T are all numbers in the interval $[0, \infty)$, we next need to specify the probabilities of these values. For a continuous random variable, probabilities are specified through a *density*, which determines the probability of any range of values. The choice of density follows inevitably once we make the following key assumption:

(A1) The returns on the stock over non-overlapping time intervals are independent of each other. Over time intervals of small length Δt approaching 0, the mean return is $\mu\Delta t$ and the variance of the return is $\sigma^2\Delta t$, for some constants μ and σ^2.

The first part of this assumption implies that future price changes cannot be predicted from past price changes. This is therefore an assumption of market efficiency. The second part of the assumption specifies that the *parameters* of the return distribution remain constant over time. This part

[3]The notation x^+ is frequently used for $\max\{x, 0\}$. Thus, $\max\{S_T - K, 0\}$ is often written as $(S_T - K)^+$.

seems less plausible, but may be reasonable over periods of three to six months — the life of typical options. For a stock that pays dividends, (A1) is reasonable only between dividend payments, since the payment of a dividend is ordinarily accompanied by a drop in the stock price.

Imagine dividing the interval $[0, T]$ into m subintervals $[0, \Delta t], [\Delta t, 2\Delta t], \ldots, [(m-1)\Delta t, T]$, with $\Delta t = T/m$ for some integer m. Let R_i be the return on the stock over the interval $[(i-1)\Delta t, i\Delta t]$, $i = 1, \ldots, m$. Then the terminal price S_T can be written as the product of the initial price and the product of (one plus) the returns:

$$S_T = S_0(1 + R_1)(1 + R_2) \cdots (1 + R_m)$$

By assumption (A1), the returns in the different subintervals are independent of each other and all have the same mean and variance. Moreover, we can take m to be arbitrarily large and Δt arbitrarily small. Thus, S_T is the product of a constant (S_0) and a large number of independent random variables, all having the same mean and variance. It follows from a general result of probability theory that S_T must therefore have a *lognormal* distribution.[4] Specifically, S_T can be represented as

$$S_T = S_0 e^{(\mu - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z}, \tag{10}$$

where Z is a standard normal random variable. (*Standard* indicates mean 0 and variance 1.) From now on, our discussion will assume that (10) is an exact description of the evolution of stock prices.

To understand (10), think of it as making predictions about the likelihood of an outcome of the future stock price S_T. For example, suppose we want to know the probability $P(S_T > x)$ that the stock price at time T will exceed some value x. From (10) we get

$$\begin{aligned} P(S_T > x) &= P(S_0 e^{(\mu - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z} > x) \\ &= P(\ln(S_0) + (\mu - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z > \ln(x)) \\ &= P\left(Z > \frac{\ln(x) - \ln(S_0) - (\mu - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}\right). \end{aligned} \tag{11}$$

We have thus reduced a probability involving S_T to one involving the standard normal random variable Z. Letting $N(\cdot)$ denote the standard normal cumulative distribution; i.e., $N(z) = P(Z \leq z)$, we get

$$\begin{aligned} P(S_T > x) &= 1 - N\left(\frac{\ln(x) - \ln(S_0) - (\mu - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}\right), \\ &= N\left(\frac{\ln(S_0/x) + (\mu - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}\right), \end{aligned} \tag{12}$$

where the second equality follows from the identity $1 - N(z) = N(-z)$. Although this expression may look somewhat complicated, the point is that from (10) we can evaluate the probability that

[4]Recall that the central limit theorem states that the *sum* of a large number of independent random variables tends to a *normal* distribution. Similarly, a *product* of independent random variables tends to a *lognormal* distribution. The logarithm of a lognormal random variable is normally distributed.

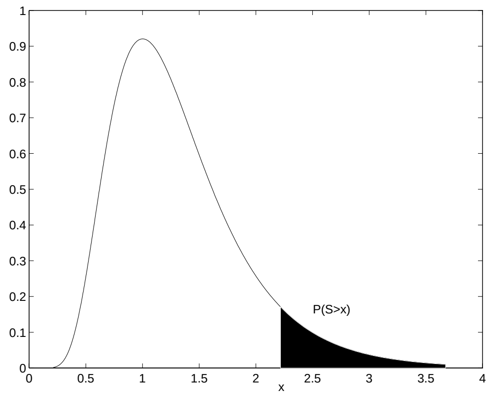


Figure 1: Density of the lognormal distribution with $\mu = 0.10$, $\sigma = 0.50$, $S_0 = 1$, and $T = 1$. The shaded area to the right of x is $P(S_T > x)$.

S_T will fall in some range. We obtain the density of S_T by differentiating $P(S_T \le x)$ with respect to x. This yields

$$f_{\mu,\sigma}(x) = \frac{1}{x\sigma\sqrt{T}} N' \left(\frac{\ln(x/S_0) - (\mu - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \right), \tag{13}$$

where N' is the standard normal density. Figure 1 shows a graph of the lognormal density in the case $S_0 = 1$, $T = 1$, $\mu = 0.10$, and $\sigma = 0.50$. The shaded area to the right of x is $P(S_T > x)$.

Once the distribution of S_T is fixed, so are its mean and variance. It can be shown that equation (10) implies

$$E[S_T] = S_0 e^{\mu T}, \tag{14}$$

which is consistent with our interpretation of μ as the mean rate of return. Also,

$$\mathrm{Var}[S_T] = S_0^2 e^{2\mu T}(e^{\sigma^2 T} - 1). \tag{15}$$

Notice that $\mathrm{Var}[S_T]$ is not equal to σ^2. Whereas $\mathrm{Var}[S_T]$ is the price variance, we call σ the *volatility* of returns. Because we measure T in years, σ is the annual volatility and μ is the annual mean rate of return.

It might be surprising to see the term $(\mu - \frac{1}{2}\sigma^2)T$ in the exponent of (10), rather than μT. But equation (14) shows that the *random* returns described by (10) do indeed result in a *mean* rate of return μ.

Replication and risk-neutral valuation

In developing an analogy between option pricing and the pricing of the weather-dependent security D, we have so far specifed the rules determining payoffs and the distribution specifying outcomes of the underlying. The next step in pricing D was to construct a replicating portfolio of assets with known prices, and to show that D could then be priced as the present value of its cash flows using risk-neutral probabilities. We now carry out analogous steps for options.

We need further assumptions supplementing (A1):

(A2) There is a constant continuously-compounded interest rate r for both borrowing and lending.

(A3) Shares of S can be bought and sold continuously over time in arbitrarily small units without transactions costs or taxes.

(A4) The stock pays no dividends in the life of the option.

We do not suggest that these assumptions are entirely realistic. Rather, they are convenient simplifications which nevertheless lead to results that are useful in practice. Through appropriate modifications, prices obtained under these assumptions can often be extended to cover weaker assumptions.

Let C_t denote the time-t price of a call expiring at time T. Our objective is to replicate the evolution of C_t by trading in other assets. In replicating D, the only source of uncertainty was the weather, so the key to constructing a replicating portfolio was matching the payoff of D in every outcome of the weather. Under assumptions (A1)–(A4), the only source of uncertainty affecting the option value is the price of the underlying stock. To replicate the option we want to construct a portfolio that changes in value in response to changes in the underlying the same way the option does. The portfolio will hold a_t shares of stock at time t, with a_t to be determined. Buying and selling of the stock is financed by borrowing and lending at the risk-free rate r. Borrowing and lending are equivalent, respectively, to selling and buying a bond. Let B_t denote the time-t price of a risk-free bond paying \$1 at time T. Then it follows from (A2) that

$$B_t = e^{-r(T-t)}. \tag{16}$$

The replicating portfolio will hold b_t shares of the bond at time t, with b_t to be determined. Changes in the number of shares a_t of the stock are financed by changes in b_t. A derivation of the appropriate a_t and b_t is given in Figure 2. The result is

$$a_t = \frac{\partial C_t}{\partial S_t}, \qquad b_t = e^{r(T-t)}[C_t - a_t S_t].$$

In particular, the number of shares of stock required is the derivative of the option price with respect to the price of the underlying. This derivative is called the option's *delta.* It is a crucial tool in *hedging* an option.

At this point, we invoke the principle of risk-neutral valuation, which we motivated through the discussion of Section 1.1: if a derivative security can be replicated by trading in other assets, then to preclude arbitrage, its price must be the expected present value of its cash flows based on risk-neutral probabilities. We know the present value of the cash flows of a European call; it is simply

$$e^{-rT}\max\{0, S_T - K\}.$$

The option price is then the expected value of this expression when S_T is given a risk-neutral distribution. Recall that, by definition, risk-neutral probabilities give all assets the same expected rate of return. We saw previously in (14) that using "real" probabilities the expected (continuously compounded) rate of return on the stock is μ. Using risk-neutral probabilities, the expected rate of return on the stock should be the same as that on a risk-free bond which, by assumption, is r.

Suppose at time t we hold a_t shares of stock. If in the next Δt time units the stock price moves by an amount ΔS, then the call price will move by approximately

$$\frac{\partial C_t}{\partial S_t}\Delta S,$$

the approximation becoming increasingly exact as $\Delta t \to 0$. The corresponding change in value of a_t shares is

$$a_t \Delta S.$$

To make these two changes equal — i.e., to make the portfolio respond to stock-price changes in the same way the option does — we should hold

$$a_t = \frac{\partial C_t}{\partial S_t}$$

shares of the stock at time t.
Because $a_t S_t$ cancels the instantaneous response of the option price to changes in the stock price, the difference $[C_t - a_t S_t]$ is perfectly immunized against changes in the stock price over the next Δt time units, as $\Delta t \to 0$. Since the stock price is the only source of risk affecting the option value, this means that the difference $[C_t - a_t S_t]$ is risk-free (over the next Δt time units) and is thus equivalent to a long or short position in the bond B_t. To find the equivalent number of bonds, we divide by the bond price in (16) and get

$$\begin{aligned} b_t &= \frac{C_t - a_t S_t}{B_t} \\ &= e^{r(T-t)}[C_t - (\partial C_t/\partial S_t)S_t]. \end{aligned} \tag{17}$$

When b_t is negative, we interpret it as the amount of borrowing required for stock purchases; when it is positive, we interpret it as invested proceeds from sales of the stock. In this way, the continuously rebalanced portfolio holding a_t shares of stock and b_t bonds perfectly replicates the option.

Figure 2: Derivation of replicating portfolio for call option.

To go from real probabilities to risk-neutral probabilities we replace μ with r. In particular, we replace (10) with

$$S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z}, \tag{18}$$

and this means replacing the density $f_{\mu,\sigma}$ in (13) with $f_{r,\sigma}$. The expected present value of the option payoff using risk-neutral probabilities is thus

$$\begin{aligned} C &= e^{-rT}E[\max\{0, S_T - K\}] \\ &= e^{-rT}\int_0^\infty \max\{0, x-K\} f_{r,\sigma}(x)\,dx. \end{aligned}$$

Some calculus shows that this is

$$C = S_0 N(d_1) - e^{-rT}KN(d_2) \tag{19}$$

with

$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \tag{20}$$

$$d_2 = \frac{\ln(S_0/K) + (r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}. \tag{21}$$

Equation (19) is the Black-Scholes formula for the price of a European call option. The corresponding formula for a European put is

$$P = Ke^{-rT}N(-d_2) - S_0 N(-d_1). \tag{22}$$

Both formulas should be interpreted as the expected present value of option payoffs based on risk-neutral probabilities for the underlying stock. By differentiating (19) with respect to S_0 and carrying out some algebraic simplification, we arrive at the formula

$$\frac{\partial C}{\partial S} = N(d_1)$$

for the call option delta. Recall that this is the number of shares of stock required to replicate or hedge the option. In the notation of Figure 2, this is a_0. From (17) we find that the number of bonds required is

$$e^{rT}[C - N(d_1)S] = e^{rT}[(SN(d_1) - e^{-rT}KN(d_2)) - N(d_1)S] = -KN(d_2).$$

In the notation of Figure 2, this is b_0. The delta of a put is $-N(-d_1)$ and the number of bonds required to hedge a put is $KN(-d_2)$.

All of these formulas are very easily evaluated in a spreadsheet. In EXCEL, the function `=NORMSDIST` returns values of the cumulative standard normal distribution. The computational simplicity of the Black-Scholes formula makes it a valuable starting point, even when the underlying assumptions are not fully in force.

If we set $\mu = r$ in (12) and compare it with $N(d_2)$, we see that

$$N(d_2) = \text{Risk-neutral probability that } S_T > K;$$

	Security D	Call Option
Underlying outcomes	RAIN or SHINE	any S_T in $[0,\infty)$
Payoff	1 if RAIN, 4 if SHINE	$\max\{0, S_T - K\}$
Real probabilities	RAIN q, SHINE $1-q$	Lognormal (μ,σ)
Risk-neutral prob.	RAIN p, SHINE $1-p$	Lognormal (r,σ)
Replication	$4B_1 + 1B_2$	$N(d_1)$ shares of stock
		and $-KN(d_2)$ bonds
Value	$(4p+1(1-p))/(1+R)$	$e^{-rT}E[\max\{0,S_T-K\}]$

Table 1: Analogy between the call option and security D of Section 1.1.

i.e., the risk-neutral probability that the option will be exercised. Thus, in (19), $e^{-rT}KN(d_2)$ is the expected present value of the cash outlay in exercising the option. The first term in (19), $S_0N(d_1)$, should similarly be interpreted as the expected present value of stock bought by exercising the option, though the justification of this interpretation is a bit more involved.

Summary

Table 1 illustrates the analogy between the pricing of the weather derivative of Section 1.1 and the call option of this section. Both securities have payoffs contingent on an uncertain event: the weather or a stock price. Both can be replicated by a portfolio of other assets, and both can therefore be priced as the expected present value of their payoffs using risk-neutral probabilities. For D, the risk-neutral probabilities change the chance of RAIN from q to p; for the call, they change the parameter μ to r in the lognormal distribution. In both settings, the effect of the risk-neutral probabilities is to change the expected rate of return on all assets to the risk-free rate. In Section 1.1, we considered a single-period model and set R to be the rate for borrowing for one day. In this section we have worked with a continuously compounded interest rate. We could have re-defined R to be the rate for borrowing from time 0 to time T. This would make

$$e^{-rT} = \frac{1}{1+R},$$

and would make the discounting formulas in the last row of the table more obviously analogous.

From now on, all our pricing calculations will be based on the principle of risk-neutral valuation. To price essentially any derivative security, we will seek to compute the expected present value of the security's cash flows using probabilities that make the expected rate of return on all assets equal to the risk-free rate. It is important to keep in mind that this approach to valuation is a consequence of the ability to replicate a security with a portfolio of other assets. It is not based on the obviously incorrect view that investors are risk-neutral.

1.3. The Binomial Approximation

Thus far, our discussion of option pricing has been based on continuous price movements described by a lognormal distribution. It is often convenient, both conceptually and computationally, to approximate this continuous description of stock movements with a model in which stock prices make discrete moves at discrete time instants. Suppose, then, that time is divided into small increments of length Δt. In each time increment, the stock price can either increase by a factor of u or decrease by a factor of d; an increase occurs with probability q, a decrease with probability $1-q$. The situation at the end of one time step is described by the following diagram:

$$\begin{tikzcd}
 & uS_0 & & e^{r\Delta t} \\
S_0 \arrow[ur, "q"] \arrow[dr, "1-q"'] & & 1 \arrow[ur, "q"] \arrow[dr, "1-q"'] & \\
 & dS_0 & & e^{r\Delta t}
\end{tikzcd}$$

The left half shows the possible changes in the stock price over the interval Δt. The right half illustrates the fact that \$1 invested risk-free grows to \$$e^{r\Delta t}$ in the same interval, regardless of the change in S.

If the stock moves to uS_0, then in the next time step its possible values are u^2S_0 and udS_0. If instead it first moves to dS_0, then in the next time step the possible values are udS_0 and d^2S_0. If we choose $d = 1/u$, the $udS_0 = S_0$, and the evolution S over two steps is thus described by the following diagram:

$$\begin{tikzcd}
 & & u^2S_0 \\
 & uS_0 \arrow[ur] \arrow[dr] & \\
S_0 \arrow[ur] \arrow[dr] & & S_0 \\
 & dS_0 \arrow[ur] \arrow[dr] & \\
 & & d^2S_0
\end{tikzcd} \tag{23}$$

More generally, the possible stock values after n steps are

$$u^nS_0,\ u^{n-1}dS_0,\ \ldots,\ ud^{n-1}S_0,\ d^nS_0,$$

for a total of $n+1$ possible values. In particular, if $T = n\Delta t$, then the number of possible stock values at time T is $n+1$. For fixed T, we can make n large by choosing a small time increment Δt. So, the assumption that only two changes are possible in a single time increment is not a restriction, provided we make Δt small.

Suppose stock movements in different time increments are independent of each other. Then the number of up movements in n time increments has a *binomial* distribution. The number of down movements is just n minus the number of up movements. Thus,

$$P(S_T = u^kd^{n-k}S_0) = \frac{n!}{k!(n-k)!}q^k(1-q)^{n-k}, \quad k = 0, 1, \ldots, n.$$

To see that this binomial model is a plausible approximation to (10), recall that the binomial distribution is well-approximated by the normal distribution if n is large. Once again, this means that Δt should be small.

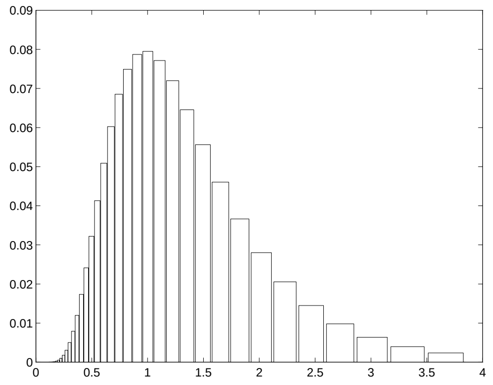


Figure 3: Probabilities for binomial approximation after 100 time steps using the same parameters as the lognormal density in Figure 1. The bars of the histogram are centered around the possible values of the binomial distribution. These become increasingly spread out, resulting in a shape similar to that of the lognormal density.

How should we choose u, d, and q? It is customary to choose $d = 1/u$, so that a decrease followed by an increase (or vice versa) returns the stock to its original price. In order to obtain an approximation to the "true" stock price (10), we set

$$u = e^{\sigma\sqrt{\Delta t}}, \quad d = 1/u = e^{-\sigma\sqrt{\Delta t}}, \tag{24}$$

and

$$q = \frac{e^{\mu\Delta t} - d}{u - d}. \tag{25}$$

With this choice of parameters, the expected percentage change in the stock price at the end of one time increment is

$$\begin{aligned} \text{Prob(up move)}\cdot(\%\text{ up}) + \text{Prob(down move)}\cdot(\%\text{ down}) &= qu + (1-q)d \\ &= q(u-d) + d \\ &= (e^{\mu\Delta t} - d) + d = e^{\mu\Delta t}, \end{aligned}$$

which matches the expected change (14) in the lognormal model.[5] Thus, the mean rate of return in the binomial approximation is equal to that in the lognormal model. Moreover, it can be shown that in the limit as Δt becomes very small, the return variances in the two models coincide.

Figure 3 illustrates the binomial approximation to the lognormal density of Figure 1 after 100 time steps. The shapes of the distributions are similar. The vertical scales for the two graphs are different because for the discrete binomial distribution the individual probabilities sum to 1, whereas for the continuous lognormal distribution it is the area under the curve that must equal 1.

[5]Equation (14) holds with T replaced by any other time horizon; e.g., $E[S_{\Delta t}] = e^{\mu\Delta t}S_0$.

2. Binomial Option Pricing

We have seen that standard European calls and puts can be priced explicitly using the Black-Scholes formulas. For more complex options, it is not always possible to find a formula for the option price, even though the price is still given by the risk-neutral expected present value of the option's payoff. Important classes of options that cannot be priced in closed form include most American options; *capped, barrier,* and *lookback* options with discrete price fixings or time-varying volatilities; *Asian* options; and *shout* options, to name just a few. When a formula is not available, the binomial method often provides a powerful tool for carrying out the expected present value calculation. Different options require different implementations of the binomial method. Our objective is to illustrate the general approach through examples.

2.1. European Calls and Puts

We begin our discussion of binomial pricing in a setting where it is not needed: the pricing of European calls and puts. For these options, we have the formulas (19) and (22), which are easily evaluated. Starting with these easy cases should help make the general method clear.

Consider a call option expiring Δt time units from now. According to the binomial approximation, the evolution of the stock and option prices over the next Δt time units are described by the following diagrams:

$$\begin{tikzcd}
& uS_0 & & C_u = \max\{0, uS_0 - K\} \\
S_0 \arrow[ur, "p"] \arrow[dr, "1-p"'] & & C \arrow[ur] \arrow[dr] & \\
& dS_0 & & C_d = \max\{0, dS_0 - K\}
\end{tikzcd}$$

Since our objective is to price the option, p should have the risk-neutral value

$$p = \frac{e^{r\Delta t} - d}{u - d} \tag{26}$$

obtained by replacing μ with r in (25). The value C_u denotes the option payoff at maturity if the stock price goes up, and C_d denotes the corresponding payoff if the stock price goes down. These values are known, because we have the formula $\max\{0, S_{\Delta t} - K\}$ for the terminal payoff. The objective is to find the unknown current value of the option, denoted by C. But we know that the current value is just the expected present value of the terminal payoff (using the risk-neutral p), so

$$C = e^{-r\Delta t}[pC_u + (1-p)C_d]. \tag{27}$$

This is the fundamental relation between the option value at a node in a binomial tree and the values at the two successor nodes.

To price an option expiring at time T, we divide T into $n = T/\Delta t$ time steps and repeat the calculation above at each time step, for each stock value. For $n = 2$, the tree looks like this:

$$\begin{tikzcd}
 & & C_{uu} = \max\{0, u^2 S_0 - K\} \\
 & C_u \arrow[ur] \arrow[dr] & \\
C \arrow[ur] \arrow[dr] & & C_{ud} = \max\{0, S_0 - K\} \\
 & C_d \arrow[ur] \arrow[dr] & \\
 & & C_{dd} = \max\{0, d^2 S_0 - K\}
\end{tikzcd}$$

The values at the terminal nodes are the option payoffs. The value at each non-terminal node is the expected present value of the values at the two successor nodes; i.e.,

$$\begin{aligned}
C_u &= e^{-r\Delta t}[pC_{uu} + (1-p)C_{ud}] \\
C_d &= e^{-r\Delta t}[pC_{ud} + (1-p)C_{dd}] \\
C &= e^{-r\Delta t}[pC_u + (1-p)C_d].
\end{aligned}$$

Thus, by working backwards through the tree, repeatedly applying (27), we eventually obtain the current option value C.

Let's consider a simple example with $n = 2$ time steps. Take

$$\sigma = 0.30, \quad r = 0.05, \quad \text{and} \quad \Delta t = 0.01.$$

Then by applying (24) and (26) we get

$$u = 1.0305, \quad d = 0.9704, \quad \text{and} \quad p = 0.5008.$$

Let the strike price be $K = 50$ and suppose $S_0 = 50$ so the option is at the money. The following tree illustrates the evolution of the stock price in the binomial tree:

$$\begin{tikzcd}
 & & 53.092 \\
 & 51.523 \arrow[ur] \arrow[dr] & \\
50.000 \arrow[ur] \arrow[dr] & & 50.000 \\
 & 48.522 \arrow[ur] \arrow[dr] & \\
 & & 47.088
\end{tikzcd} \tag{28}$$

These values were obtained by plugging numbers into (23). It is easy to verify that the stock price at each node in the tree is the present value of the stock prices at the two successor nodes. For example,

$$\begin{aligned}
51.523 &= e^{-(.05)(.01)}[(.5008)(53.902) + (.4992)(50.000)] \\
&= e^{-r\Delta t}[p \cdot (53.902) + (1-p) \cdot (50.000)].
\end{aligned}$$

In other words, the tree "prices" the stock at the present value of its future price. This is a consequence of using risk-neutral p, u, and d.

Now consider the option price. At the three terminal nodes we know the option value because we know its terminal payoff $\max\{0, S_T - K\}$. For example, at the top terminal node we have a payoff of $\max\{0, 53.092 - 50.000\} = 3.092$. The other two terminal node values are 0. Once we have the terminal values we can find all other values by working backwards and using (27) at each step. The result is the following tree of option values:

$$\begin{tikzcd}
 & & 3.092 \\
 & 1.548 \arrow[ur] \arrow[dr] & \\
0.775 \arrow[ur] \arrow[dr] & & 0 \\
 & 0 \arrow[ur] \arrow[dr] & \\
 & & 0
\end{tikzcd} \tag{29}$$

In particular, the current value of the option is 0.775.

The calculation of a put price would be exactly the same, except that the terminal payoffs would be given by $\max\{0, K - S_T\}$. The present value calculations would be the same as for a call.

Indeed, there is nothing special about the standard call and put payoffs, as far as the binomial method is concerned. Any other derivative security whose payoff depends solely on the terminal stock price could be valued using this approach. To value a *quadratic power option*, for example, we would assign a payoff of $(S_T - K)^2$ at each terminal node and then work backwards in the same way as before. However, in cases where the cash flows of the option depend on more than just the terminal stock price, further modification is required, as we will see in Section 2.3.

Greeks

It is a simple matter to get approximate values of various hedge ratios — collectively referred to as *Greeks* — from a binomial tree. Recall that an option's delta is the derivative of its price with respect to the price of the underlying: $\Delta = \partial C/\partial S$. An approximate delta can be computed from the tree by examining the change in the option price resulting from a step in the tree. If C_u is the option price at uS and C_d is the option price at dS, then

$$\Delta \approx \frac{C_u - C_d}{uS - dS}.$$

This approximation ignores the fact that C_u and C_d correspond to prices Δt time units later than the current price C.

Gamma is the second derivative of the option price with respect to the underlying. We can approximate gamma as the difference of two deltas. Two steps into the tree we have three nodes, with option values C_{uu}, C_{ud} and C_{dd}. The ratio

$$\Delta_u = \frac{C_{uu} - C_{ud}}{u^2 S - S}$$

is the approximate delta at a stock price of uS. Similarly,

$$\Delta_d = \frac{C_{ud} - C_{dd}}{S - d^2 S}$$

is the approximate delta at dS. To approximate gamma, we take the difference of these deltas and divide by the difference in the stock prices at which they apply. This yields

$$\Gamma \approx \frac{\Delta_u - \Delta_d}{(uS - dS)}.$$

This approximation ignores the fact that C_{uu}, C_{ud}, and C_{dd} correspond to prices $2\Delta t$ time units later than the current price.

Spreadsheet implementation

To obtain accurate option values, it is necessary to build a reasonably large tree. But building even a 50-step tree in a spreadsheet is quite cumbersome if we insist on a literal representation of the tree. Fortunately, the process can be simpified considerably if we are willing to include additional values in the spreadsheet.

To implement a binomial tree in a spreadsheet we need cells with the following information:

- ◦ binomial stock prices;
- ◦ terminal option payoffs at terminal nodes;
- ◦ option present values at non-terminal nodes.

The difficulty lies in copying the formulas for the option payoffs and present values to just the right cells while skipping the cells that do not correspond to nodes in the tree. But it turns out that skipping is not really necessary. We can work with an *array* of nodes rather than a tree, and that is more convenient in a spreadsheet.

The following array shows $5 = 2n + 1$ stock values (after $n = 2$ steps) and intermediate calculations of the option value for the example carried out previously in (29):

$$\begin{tikzcd}
\underline{C_{T-2\Delta t}} & \underline{C_{T-\Delta t}} & \underline{C_T} & \underline{\text{Stock prices}} \\
1.927 \arrow[r] \arrow[dr] & 2.307 \arrow[r] \arrow[dr] & 3.092 & u^2S_0 = 53.092 \\
1.535 \arrow[ur] \arrow[dr] & 1.548 \arrow[ur] \arrow[dr] & 1.523 & uS_0 = 51.523 \\
0.775 \arrow[ur] \arrow[dr] & 0.762 \arrow[ur] \arrow[dr] & 0 & S_0 = 50.000 \\
0.382 \arrow[ur] \arrow[dr] & 0 \arrow[ur] \arrow[dr] & 0 & dS_0 = 48.522 \\
0 \arrow[ur] \arrow[r] & 0 \arrow[ur] \arrow[r] & 0 & d^2S_0 = 47.088
\end{tikzcd} \tag{30}$$

The last column gives all stock prices that can be reached *within* $n = 2$ time steps rather than just those that can be reached *at the end* of n time steps. For this reason, there are $2n + 1 = 5$ stock prices listed rather than just the $n + 1$ prices one finds at the end of a tree.

The column of the array labeled C_T gives the option payoffs corresponding to the possible stock prices, using the formula $\max\{0, S_T - K\}$. We encountered the value 3.092 in building (29); in contrast, the value 1.523 is not a genuine terminal payoff because $51.523 = uS_0$ is not a genuine terminal stock price. Now look at the column labeled $C_{T-\Delta t}$. The second entry 1.548 is obtained from the first and third entries of column C_T through the usual expected present value calculation:

$$\begin{aligned} e^{-r\Delta t}[p \times 3.092 + (1-p) \times 0] &= (0.9995) * (0.5008) * (3.092) \\ &= 1.548; \end{aligned}$$

i.e., we just average the discounted values obtained one step northeast and one step southeast. The third and fourth entries of the column $C_{T-\Delta t}$ are obtained in the same way. In calculating the first entry of column $C_{T-\Delta t}$ we face a slight difficulty: there is no northeast entry. Instead, we just use the adjacent entry, 3.092, to get

$$e^{-r\Delta t} p \times 3.092 + e^{-r\Delta t}(1-p) \times 1.523 = 2.307.$$

This value has no particular meaning, but it serves to fill in the array without affecting the final option price. At the bottom of the column we face a similar problem; there is no southeast entry, so we use the adjacent value 0. The procedure is repeated to obtain the $C_{T-2\Delta t}$ column from the $C_{T-\Delta t}$ column.

Notice that the ordinary binomial tree (29) is embedded in the array (30). In particluar, the option price 0.775 is the middle value of the first column. The other numbers we calculated did not interfere with the calculation of the option price. So why bother calculating all the other numbers? The answer is that the formulas used to get the second column from the third column can then be copied to the first column without modification. This becomes particularly convenient when we have a large number of time steps.

In (30) we have illustrated the evolution of the calculations from right to left to be consistent with the usual of way of drawing binomial trees. In a spreadsheet, it is convenient to carry out the calculations from left to right: this makes it easier to change the number of time steps. Figure 4 gives a schematic illustration of a generic spreadsheet implementation. Column A contains the $2n+1$ possible stock values; Column B contains the terminal payoffs; and the remaining columns carry out the present value calculation for the remaining $n-1$ steps. The option price is the middle value (the $(n+1)^{\text{th}}$ entry) of the last column. It is not necessary to enter all terminal stock prices in column A directly. It suffices to enter a formula for $u^n S_0$ in cell A1, to enter a formula for $d{*}$A1 in cell A2, and then to copy cell A2 to the remaining entries of column A.

Matrix representation

For fast computation using the binomial method with a large number of steps, spreadsheets are inadequate. In fact, the array implementation just described actually increases the computational burden by significantly increasing the number of calculations. However, this representation is also the key to fast computation using MATLAB.

MATLAB works best with problems involving matrix calculations. The array representation of the binomial methods allows us to implement a step backwards through the tree as a *matrix*

	A	B	C	D
	x =	EXP(-r*Δt)	u=EXP(σ*SQRT(Δt))	p=(EXP(r*Δt)-d)/(u-d)
	K =	strike	d = 1/u	q=1-p
	S_T	T	$T-\Delta t$	$T-2\Delta t$
1	$u^n S_0$	MAX(A1-K,0)	x*(p*B1+q*B2)	x*(p*C1+q*C2)
2	$u^{n-1}S_0$	MAX(A2-K,0)	x*(p*B1+q*B3)	x*(p*C1+q*C3)
3	$u^{n-2}S_0$	MAX(A3-K,0)	x*(p*B2+q*B4)	x*(p*C2+q*C4)
⋮	⋮	⋮	⋮	⋮
2n	$d^{n-1}S_0$	MAX(A2n-K,0)	x*(p*B2n-1+q*B2n+1)	x*(p*C2n-1+q*C2n+1)
2n+1	$d^n S_0$	MAX(A2n+1-K,0)	x*(p*B2n+q*B2n+1)	x*(p*C2n+q*C2n+1)

Figure 4: Spreadsheet for binomial array

multiplication. Continuing with the example treated above, we set $q = 1 - p$ and construct the *present value matrix*

$$P_V = \begin{pmatrix} e^{-r\Delta t}p & e^{-r\Delta t}q & 0 & 0 & 0 \\ e^{-r\Delta t}p & 0 & e^{-r\Delta t}q & 0 & 0 \\ 0 & e^{-r\Delta t}p & 0 & e^{-r\Delta t}q & 0 \\ 0 & 0 & e^{-r\Delta t}p & 0 & e^{-r\Delta t}q \\ 0 & 0 & 0 & e^{-r\Delta t}p & e^{-r\Delta t}q \end{pmatrix} = \begin{pmatrix} 0.501 & 0.499 & 0 & 0 & 0 \\ 0.501 & 0 & 0.499 & 0 & 0 \\ 0 & 0.501 & 0 & 0.499 & 0 \\ 0 & 0 & 0.501 & 0 & 0.499 \\ 0 & 0 & 0 & 0.501 & 0.499 \end{pmatrix} \tag{31}$$

The present value matrix will always have $2n + 1$ rows and columns with entries following the pattern of this example. Multiplying the column of terminal option payoffs C_T in (30) by this matrix yields the column of intermediate option values $C_{T-\Delta t}$:

$$\begin{pmatrix} 0.501 & 0.499 & 0 & 0 & 0 \\ 0.501 & 0 & 0.499 & 0 & 0 \\ 0 & 0.501 & 0 & 0.499 & 0 \\ 0 & 0 & 0.501 & 0 & 0.499 \\ 0 & 0 & 0 & 0.501 & 0.499 \end{pmatrix} \begin{pmatrix} 3.092 \\ 1.523 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2.307 \\ 1.548 \\ 0.762 \\ 0 \\ 0 \end{pmatrix}$$

Multiplying by the present value matrix again yields the $C_{T-2\Delta t}$ values in (30):

$$\begin{pmatrix} 0.501 & 0.499 & 0 & 0 & 0 \\ 0.501 & 0 & 0.499 & 0 & 0 \\ 0 & 0.501 & 0 & 0.499 & 0 \\ 0 & 0 & 0.501 & 0 & 0.499 \\ 0 & 0 & 0 & 0.501 & 0.499 \end{pmatrix} \begin{pmatrix} 2.307 \\ 1.548 \\ 0.762 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1.927 \\ 1.535 \\ 0.775 \\ 0.382 \\ 0 \end{pmatrix}$$

Each step backwards through the binomial tree is thus executed as a matrix multiplication. As before, the option price 0.775 appears in the middle of the final column.

In general, then, we can implement the binomial tree calculation through the following steps:

1. Construct the vector $(u^n S_0, u^{n-1}S_0, \ldots, uS_0, S_0, dS_0, \ldots, d^{n-1}S_0, d^n S_0)$ of terminal stock values, as a column vector. This vector has $2n+1$ components, where n is the number of time steps.

2. Compute the terminal option payoffs for these stock values, using the formula $\max\{0, S-K\}$ for a call and the formula $\max\{0, K-S\}$ for a put.

3. Construct a $(2n+1)\times(2n+1)$ present value matrix P_V having the same form as (31).

4. Multiply the option payoffs by P_V n times. The $(n+1)^{\text{th}}$ entry of the resulting vector is the present value of the option payoff at the initial stock price, and thus yields the option value.

We will use this general approach, with various specific modifications, to price a variety of options.

Figure 21 in the Appendix shows a MATLAB implementation `binom.m` of this approach. Line 15 constructs the terminal stock values; lines 19–22 construct the present value matrix; line 25 computes the terminal option payoffs; lines 28–30 do the n matrix multiplications. To price a European put option change `s_vec-strike` in line 25 to read `strike-s_vec`.

Figure 5 gives some indication of the importance of running the binomial method for a sufficiently large number of time steps, and shows why small spreadsheet implementations are not suitable for obtaining precise values. As the number of time steps increases, the binomial price approaches the true value, indicated by the horizontal line in the graph. However, it does so erratically. For small values of n, the binomial price can be quite far from the true price, and it is impossible to know in advance whether the error is positive or negative at a particular n. Figure 5 might seem to suggest that one could move along the jagged line until the oscillations settle down to get an accurate price, but this is not practical: increasing the number of time steps from, say, 90 to 100 entails re-running all 100 steps, not just an additional 10 steps. The parameters p, u, and d of the binomial method change as the number of steps changes.

Generally speaking, it is a good idea to build at least a 200-step tree to obtain reasonable price estimates; using less than 50 steps should certainly be avoided except when the complexity of the option makes more steps infeasible. But it is important to keep in mind that the number of steps required for a fixed precision can vary substantially with the type of option, and the precision required may depend on the context. For standard European calls and puts, the average error over a fairly broad range of parameters is about 0.1%, which means penny-accuracy on a \$10 option. However, for a particular set of parameters the error may be five to ten times as large as the average. We give more precise information at the end of these notes, after discussing simulation.

2.2. American Options

How should the binomial method be modified to price *American* options; i.e., options that can be exercised at any time up to expiration, rather than just at expiration? The key to this and all other extensions of the binomial method is to find the right modification of the basic recursion (27). The stock-price tree is just as before; only the structure of the option has changed. If we are pricing an American call, then the terminal payoffs are the same as for a European call. The change we need

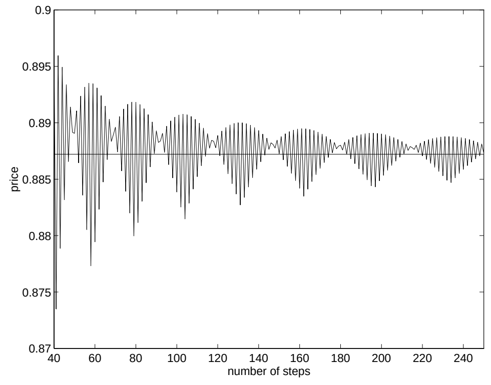


Figure 5: Convergence of binomial method for a European call with $\sigma = 0.30$, $r = 0.05$, $T = 0.20$, and $S_0 = 45$, $K = 50$. The horizontal line is at the Black-Scholes price of 0.8872. The binomial price approaches this limit erratically as the number of times steps increases.

to make is in the relation between the option price at a node and at its two successor nodes; i.e., the relation between C, C_u, and C_d in the following diagram:

$$\begin{tikzcd} & C_u \\ C \arrow[ur, "p"] \arrow[dr, "1-p"'] & \\ & C_d \end{tikzcd}$$

Once we know how C is related to C_u and C_d, we can apply the same relation at all non-terminal nodes.

In the European case, C is just the present value of C_u and C_d, as in (27). In the American case, C will clearly be at least that large since the owner may choose to hold the option for the next Δt time units. However, the holder may also choose to exercise the option and receive its intrinsic value. The intrinsic value is $\max\{0, S - K\}$, where S is the stock price corresponding to the current node. The option should be exercised only if the intrinsic value exceeds the present value of holding the option for another time step. Thus, we have

$$C = \max\{S - K, e^{-r\Delta t}[pC_u + (1-p)C_d]\}.$$

This is the formula we use to work backwards through the tree in pricing an American call. For an American put, we use

$$P = \max\{K - S, e^{-r\Delta t}[pP_u + (1-p)P_d]\},$$

where P_u and P_d are the put values after a step up and step down, respectively.

Let's consider an example. We use the same parameters as in the example surrounding (29), except that now we consider a put. Here are the trees for the European and American puts, respectively:

$$\begin{tikzcd}
 & \underline{\text{European}} & & & \underline{\text{American}} & & \underline{\text{Stock}} \\
 & & 0 & & & 0 & 53.092 \\
 & 0 \arrow[ur] \arrow[dr] & & & 0 \arrow[ur] \arrow[dr] & & 51.523 \\
0.725 \arrow[ur] \arrow[dr] & & 0 & 0.737 \arrow[ur] \arrow[dr] & & 0 & 50.000 \\
 & 1.453 \arrow[ur] \arrow[dr] & & & 1.478 \arrow[ur] \arrow[dr] & & 48.522 \\
 & & 2.912 & & & 2.912 & 47.088
\end{tikzcd}$$

The numbers on the far right are stock prices corresponding to the different tree heights. From these, we can easily compute intrinsic values. For example, at the initial node, the immediate-exercise value of the option is 0, because both the current price S_0 and the strike K are 50. If in the next Δt time increment the stock price moves to $dS_0 = 48.522$, the intrinsic value becomes $50 - 48.522 = 1.478$. At that node, the value of holding the option for another time step is

$$e^{-r\Delta t}[p \cdot 0 + (1-p) \cdot 2.912] = 1.453.$$

Indeed, this is the value of the European option at time Δt in node dS_0. Since the intrinsic value exceeds the continuation value, the option should be exercised, resulting in a node value of 1.478 for the American option.

We now get our first indication of the power and simplicity of the matrix formulation of binomial trees. To modify the MATLAB program `binom.m` to price an American put (instead of a European call), we just need to make two modifications. Line 25 becomes

```
op_vec = max(strike-s_vec,0)
```

so that the terminal payoffs correspond to puts rather than calls. And line 29 becomes

```
op_vec = max(strike-s_vec,pv*op_vec).
```

This way, the intermediate option values that get propagated backwards correspond to the maximum of the present value of holding the option or exercising it immediately. In the spreadsheet implementation of Figure 4, we would replace the entries in column B to read `MAX(K-An,0)`. Cell C2 becomes

```
MAX(K-$A2,x*(p*B1+q*B3))
```

and the rest of the array is modified analogously.

2.3. Path-Dependent Options

Standard European calls and puts are examples of *path-independent* options: their payoffs are determined solely by the terminal stock price, regardless of the intermediate stock prices by which

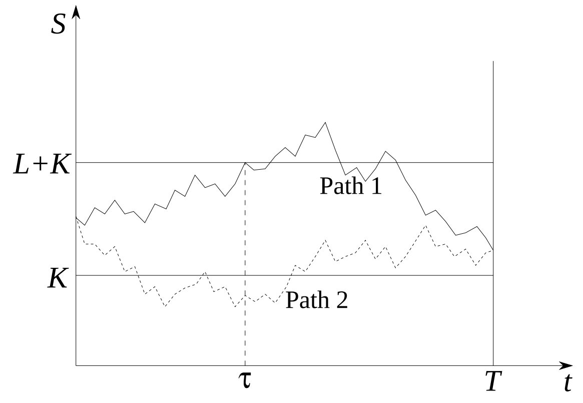


Figure 6: Path-dependent payoff of a capped call. Path 1 pays L at time τ and Path 2 pays $S_T - K$ at time T, though the two paths end at the same S_T.

the terminal price was reached. In contrast, many important classes of so-called exotic options are *path-dependent,* in the sense that their payoffs depend on the path followed by the underlying asset, and not just its terminal price. Tailoring the binomial method to path-dependent options is a bit of an art and is not always possible. As with the extension to American options, the key lies in finding how the option price at a node is related to the prices at its successor nodes. We will illustrate this through examples.

Capped calls with automatic exercise

A capped call is an ordinary call option in which the payoff is capped at some level L. The intrinsic value of a capped call is thus $\min\{L, \max\{0, S-K\}\}$. A European capped call can be priced in a binomial tree by evaluating this intrinsic value at every terminal node and then working backwards in the usual way. But some capped calls, including in particular the Chicago Board's "CAPS" on the S&P 100 and S&P 500 have an automatic exercise feature: if the intrinsic value ever reaches L, the amount L is paid out immediately. This makes the options path-dependent, because the timing of the cash flows depends on intermediate values of the underlying, not just the terminal value.

Figure 6 illustrates the path-dependent feature of the payoff on a capped call. Paths 1 and 2 in the figure show two possible trajectories for the price of the underlying. They end up at the same terminal value S_T, but result in different payoffs. If the underlying follows Path 1 the option pays L at time τ because of the automatic exercise feature. Path 2 results in a payoff of $S_T - K$ at time T.

To use the binomial method to price these options, it should be clear that we want to evaluate the intrinsic value $\min\{L, \max\{0, S-K\}\}$ at every terminal node. However, we cannot simply work backwards using (27); we need to develop a rule that accounts for the automatic exercise feature.

To find the right rule, we consider the following simple situation:

$$\begin{tikzcd}[row sep=small] & uS & & & C_u \\ S \arrow[ur] \arrow[dr] & & & C \arrow[ur] \arrow[dr] & \\ & dS & & & C_d \end{tikzcd}$$

The figure on the right is meant to indicate the change in the capped-call value associated with the possible changes in the underlying illustrated on the left. The C-value at each node in the tree records the present value of the capped call from that node forward. To develop a recursive rule, we ask ourselves the following question: Suppose C_u and C_d have already been determined, how do we find C?

There are two cases to consider, depending on whether or not automatic exercise is triggered at S. Since the payoff is capped at L, the smallest underlying price at which automatic exercise occurs is $K + L$. So the cases are as follows:

(i) $S \geq K + L$. Automatic exercise is triggered at the current underlying price, so $C = L$.

(ii) $S < K + L$. In this case, the cap plays no direct role at the current node, and we just use the usual rule:
$$C = e^{-r\Delta t}[pC_u + (1 - p)C_d].$$
This includes the possibility that $uS \geq K + L$, in which case C_u must already have been set equal to L.

We can combine the two cases by writing

$$C = \begin{cases} e^{-r\Delta t}[pC_u + (1 - p)C_d], & \text{if } S < K + L; \\ L, & \text{if } S \geq K + L. \end{cases}$$

This is easily implemented in a spreadsheet using an `IF` statement.

Let's look at an example based on the following parameters:

Parameter	Value
Volatility (σ)	0.30
Riskless rate (r)	0.05
Strike price (K)	45
Stock price (S_0)	47
Cap (L)	5
Time to maturity (T)	0.10 years

The resulting parameters for a four-step tree are

$$u = 1.049, \quad d = .9537, \quad p = .5013, \quad e^{-r\Delta t} = .9988.$$

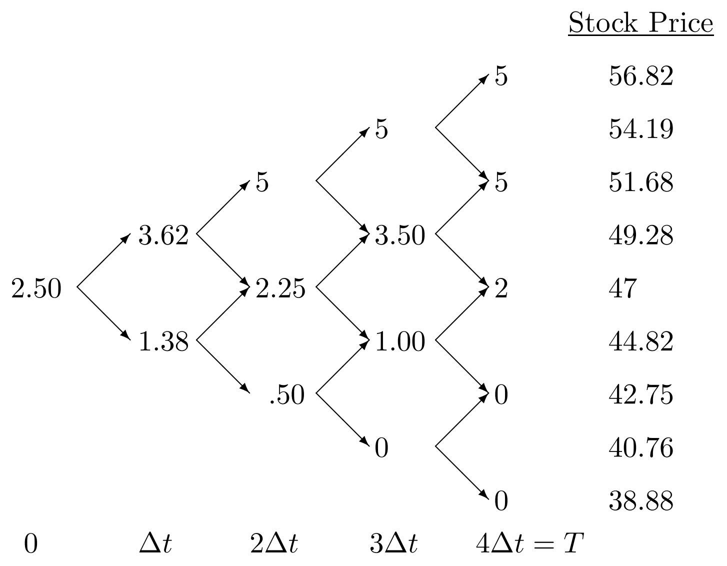


Figure 7: Binomial tree for capped call with automatic exercise. The initial stock price is 47, the strike is 45, and the cap is 5. Automatic exercise is triggered when the stock price is 50 or greater. Values at the far right give stock prices for each level of nodes.

The tree is illustrated in Figure 7. The prices of the underlying associated with each level of nodes are listed at the far right. The terminal node values 5, 5, 2, 0, 0 result from the intrinsic-value formula $\min\{5, \max\{0, S - 45\}\}$. The option is automatically exercised any time the underlying reaches a price of 50 or greater, resulting in an immediate payout of 5 at every node above 50. Values at all other nodes are present values of their successor nodes.

To check that the answer of 2.50 in Figure 7 is correct, we can enumerate all possible paths through the tree, compute the discounted payoff on each path, multiply the discounted payoff by the path's probability, and sum over paths. In fact, in Figure 7 there are only three distinct possible outcomes resulting in non-zero payoffs: an automatic exercise may occur at time $2\Delta t$ for a discounted payoff of $e^{-2r\Delta t}5$; the option may expire with an intrinsic value of 5 for a discounted payoff of $5e^{-4r\Delta t}$; or it may expire with an intrinsic of 2 for a discounted payoff of $e^{-4r\Delta t}2$. It is a simple matter to find the probabilities of these outcomes from the tree. An automatic exercise at time $2\Delta t$ requires two consecutive upmoves and thus has probability p^2. Expiration with an intrinsic value of 5 requires a downmove followed by three upmoves and has probability $p^3(1-p)$. There are five paths terminating with a payoff of 2; each has two upmoves and two downmoves for a probability of $p^2(1-p)^2$. We can now combine probabilities and payoffs in the following calculation:

Discounted Payoff	×	Probability	=	Product
$e^{-2r\Delta t}5 = 4.988$		$p^2 = .2513$		1.2535
$e^{-4r\Delta t}5 = 4.976$		$2p^3(1-p) = .1257$		.6255
$e^{-4r\Delta t}2 = 1.990$		$5p^2(1-p)^2 = .3125$		.6220
				2.501

Except for rounding, this is the value found in the tree. Indeed, the tree simply provides a way of carrying out this calculation without having to list all possibilities explicitly.

A MATLAB implementation `cap_bin.m` appears in Figure 23.

Knock-Out Options

Among the most widely traded path-dependent options are *barrier options.* In the currency markets, for example, barrier options are reported to make up more than half the volume in exotics and roughly 10% of the total volume in derivatives. The payoff of a barrier option depends on whether or not the underlying asset (or exchange rate, or index) reaches a specified level within the life of the option. In the case of a knock-out, the option is extinguished if the barrier is crossed, whereas a knock-in becomes active only when the barrier is crossed. A *down* barrier option has the barrier below the underlying at the inception of the contract; an *up* option has the barrier above the underlying. These options can be calls or puts.

Consider the pricing of a down-and-out call; i.e., a call that gets knocked out if the underlying ever drops below the barrier. To develop a binomial pricing method for this barrier option, we start with the following diagram:

$$\begin{tikzcd} & uS & & B_u \\ S \arrow[ur] \arrow[dr] & & B \arrow[ur] \arrow[dr] & \\ & dS & & B_d \end{tikzcd} \tag{32}$$

Suppose the option values B_u and B_d starting from the indicated nodes have already been computed. How do we determine the option value B at a node one step back?

There are two cases to consider depending on where the barrier lies. Let H denote the level of the barrier.

(i) $S \leq H$. The option is knocked out at the current node, so $B = 0$.

(ii) $S > H$: The current price of the underlying is above the barrier so no knock-out occurs. We have simply
$$B = e^{-r\Delta t}[pB_u + (1-p)B_d],$$
just as for a standard call. This includes the possibility that $dS \leq H$, in which case B_d must already have been set to 0.

A simple way to visualize these rules is this: at all nodes below the barrier the option price is 0; at all other nodes the option price is the present value of the prices at the successor nodes. A MATLAB implementation `doc_bin.m` appears in Figure 24.

To illustrate this with an example, we use the same parameters we used above for the capped call, except that we move the strike to 50 and set the barrier at $H = 45$. Figure 8 illustrates the calculations. It is worth noting that each intermediate node value can be interpreted as the price of a *newly issued* option originating at that node. For example, 0.42 is the price of an option with maturity $2\Delta t$ and underlying price 47. Whether or not this will be the value of the original option

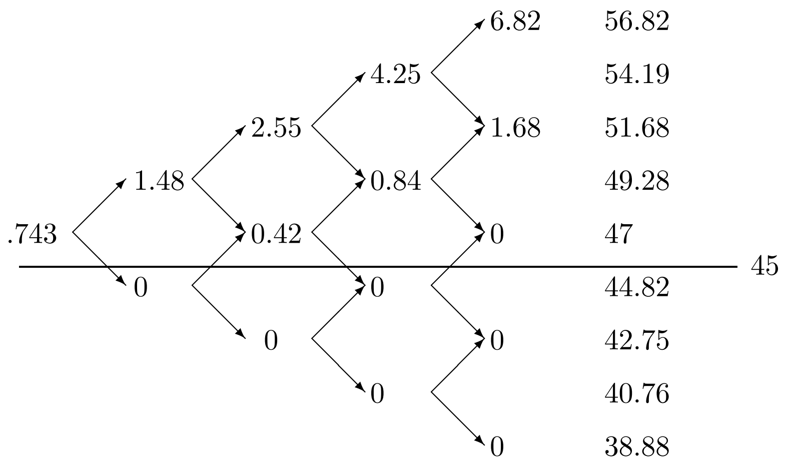


Figure 8: Binomial tree for down-and-out call. The initial asset price is 47, the strike is 50, and the barrier is at 45. Below the barrier, the option is worthless.

at time $2\Delta t$ depends on the path followed by the underlying. If the underlying first increases then decreases, the option value will indeed be 0.42; but if it first decreases and then increases, the option value will be 0.

A variant of the Black-Scholes formula for pricing knock-out options is given in Appendix A. Because these options can be priced in closed form, the binomial method is not really necessary. However, the formula applies only to the simplest knock-outs. It does not account for provisions specifying that knock-outs can occur only at daily closing prices, nor can it accommodate more complex pricing features like time-varying volatilities. Numerical methods are required to handle these important types of restrictions.

How many steps to use?

We noted previously (see Figure 5) that the convergence of the binomial method to the true option price can be rather erratic. In the case of a standard option, Figure 5 indicates that the binomial price oscillates around the true price as the number of time steps n increases. Figure 9 shows the rather different behavior of the binomial method when applied to a down-and-out call. The parameters are those used for the example of Figure 8 but with the number of time steps ranging from 40 to 250. The horizontal line is at the true price of 0.6506, obtained from the formula in Appendix A. Instead of oscillating, the binomial price appears to stay consistently above the true price, occasionally coming very close to the true price and then jumping away from it. The figure indicates that we might actually get a much worse price by increasing n, unless we know how to choose the number of steps correctly.

What's going on? The fact that the binomial method is consistently overpricing the option suggests that it is underestimating the probability of a knock-out. This, in turn, is a result of the relative positions of the barrier and the nodes in the tree. Consider Figure 8, for example. The barrier at 45 does not lie exactly at a layer of stock-price nodes: it lies between 44.82 and 47. So, for a knock-out to occur in this tree, the stock price must drop as low as 44.82, even though the

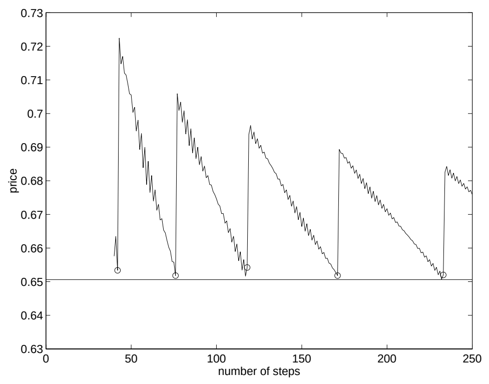


Figure 9: The jagged line is the binomial price of a down-and-out call as a function of the number of time steps n. The horizontal line gives the correct price. The points marked with an 'x' correspond to values of n produced by equation (35).

barrier is at 45. Since it is less likely that the price will drop to 44.82 than to 45, the tree implicitly underestimates the chance of a knock-out. As the number of time steps n increases, the relative positions of the barrier and the nodes change. For some values of n, the extra distance from the barrier to the next layer of nodes will be small. These are the values of n that make the binomial method most accurate.[6]

How do we find these good values of n? Recall that the possible stock prices in an n-step tree are

$$u^nS_0, u^{n-1}S_0, \dots uS_0, S_0, dS_0, \dots, d^{n-1}S_0, d^nS_0.$$

Also, $u = e^{\sigma\sqrt{\Delta t}}$ and $d = e^{-\sigma\sqrt{\Delta t}}$, with $\Delta t = T/n$. Thus, another way to write the possible prices is

$$e^{-m\sigma\sqrt{T/n}}S_0, \quad \text{for } -n \le m \le n.$$

Suppose the barrier is at H, with $H < S_0$. How far does the price have to fall in the tree to cross the barrier H? The answer is given by the smallest integer m for which

$$e^{-m\sigma\sqrt{T/n}}S_0 \le H. \tag{33}$$

The smallest m that satisfies this inequality is the least number of down moves required to reach a node at or below the barrier. The barrier would fall exactly on a layer of nodes if

$$e^{-m\sigma\sqrt{T/n}}S_0 = H, \quad \text{for some integer } m.$$

Solving for n, this requires

$$n = \frac{m^2\sigma^2T}{\left(\log\frac{S_0}{H}\right)^2}, \quad \text{for some integer } m. \tag{34}$$

[6]This observation and the method for choosing n are from P. Boyle and S.H. Lau, "Bumping up against the Barrier with the Binomial Method," *Journal of Derivatives,* pp.6-14, Summer 1994.

But we need n to be an integer as well, and the expression on the right may not be an integer. If we round up, we violate (33) and end up putting a layer of nodes just *above* the barrier. If we round down, we put a layer of nodes at or just *below* the barrier. Since our objective is to minimize the extra amount by which the stock price must fall for a barrier-crossing to be detected in the tree, just below is better than just above. Indeed, putting a layer of nodes just above the barrier *maximizes* the distance from the barrier to the first layer of nodes below the barrier and can therefore be expected to give poor results.

It is a simple matter to use (34) to get good values of n in MATLAB. The operation "`.*`" yields the entry-by-entry product of two arrays of equal dimension. The expression `[1:20]` yields an array consisting of the integers 1 to 20. Thus, `[1:20].*[1:20]` generates the first 20 values of m^2, $m = 1, 2, \ldots$. The expression

```
sig^2*t*([1:20].*[1:20])/(log(s/barrier))^2
```

generates (34) for $m = 1, 2, \ldots, 20$ (once the variables `sig`, `t`, `s` and `barrier` have been defined). To round down, we apply the function `floor` to this expression. Thus, typing

```
floor(sig^2*t*([1:20].*[1:20])/(log(s/barrier))^2)          (35)
```

yields the first 20 "good" values of n corresponding to $m = 1, \ldots, 20$. Of these, a value of n around 200 should be satisfactory. If none of the values returned by MATLAB is that large, replace "20" in (35) with a larger number and try again. In the example of Figure 9, the first ten values produced by (35) are

$$4, 19, 42, 76, 118, 171, 233, 304, 385, 475.$$

In Figure 9, the binomial prices at $n = 42, 76, 118, 171$ and 233, obtained from (35), are marked with a "$\circ$." These come very close to the true value. Also, as predicted in the discussion above, the worst values of n come right after the best values. These put a layer of nodes just above the barrier rather than just below it.

Knock-In Options

Whereas a knock-out option pays nothing if the barrier is crossed, a knock-in pays nothing *unless* the barrier is crossed. A method for pricing knock-outs can be used to price knock-ins in light of the evident relation

$$\text{Knock-in call} + \text{Knock-out call} = \text{Standard call},$$

and the corresponding relation for puts. This reflects the fact that the payoff on a portfolio holding a knock-in call and a knock-out call is the same as the payoff on a standard call, regardless of whether or not the barrier is ever crossed. In spite of this symmetry relation, we will consider knock-ins separately because they serve to illustrate some general principles in binomial pricing of path-dependent options.

To develop a binomial method that prices knock-ins directly, we would like to start with a recursion for a single step like the one illustrated in (32). But we face a difficulty: the value at a node depends on how the node was reached. In particular, the value of the option at the current node is greater if the barrier was previously crossed than if it was not. This suggests that we need to keep track of *two* prices at each node — one that applies if the option was previously knocked in, another that applies if the barrier has yet to be crossed.

Let's therefore define

$$\begin{aligned} B^+ &= \text{Value of option if previously knocked in} \\ B^- &= \text{Value of option if not previously knocked in} \end{aligned}$$

At each node we keep track of the vector

$$\begin{pmatrix} B^+ \\ B^- \end{pmatrix}$$

and update both values as we move backwards through the tree. There are now two cases to consider:

(i) $S > H$. The knock-in status is unknown at the current node; if a knock-in occurred, it must have occurred previously, not at the current node. Thus, the two values are updated separately:

$$\begin{aligned} B^+ &= e^{-r\Delta t}[pB_u^+ + (1-p)B_d^+] \\ B^- &= e^{-r\Delta t}[pB_u^- + (1-p)B_d^-] \end{aligned}$$

(ii) $S \leq H$. The option is definitely knocked-in. Hence,

$$B^+ = e^{-r\Delta t}[pB_u^+ + (1-p)B_d^+],$$

and $B^- = B^+$ because whether or not the option was previously knocked-in it is knocked-in now.

These two cases are illustrated in Figure 10. A MATLAB implementation `dic_bin.m` appears in Figure 25.

If we apply these rules throughout the tree, we end up with two values at the initial node. Which one is the option price? Assuming $S_0 > H$ (the only interesting case for a down option), no knock-in has occurred, so the option price is B^-. If for some reason we were to have $S_0 \leq H$, then the option would be knocked in from the start, so the price would be B^+. In fact, the B^+ value at the initial node is just the price of a standard (no-barrier) European call.

Let's consider the parameters used in the knock-out example of Figure 8, but now applied to a knock-in call. In particular, then, $S_0 = 47$, $K = 50$, and the barrier is at 45. At the terminal nodes, B^+ is the usual call-option terminal payoff $\max\{0, S_T - K\}$, becuase B^+ records the payoff assuming a knock-in occurred previously. Since B^- records the payoff assuming no knock-in occurred previously, its value is set to 0 at every terminal node. Repeatedly applying the rules above to work backwards results in the node values illustrated in Figure 11.

As always, we can check the value calculated from a binomial tree by listing all paths, evaluating their discounted payoffs, multiplying by their probabilities, and summing. A down-and-in call has a non-zero payoff only on those paths that drop below the barrier and then rise above the strike

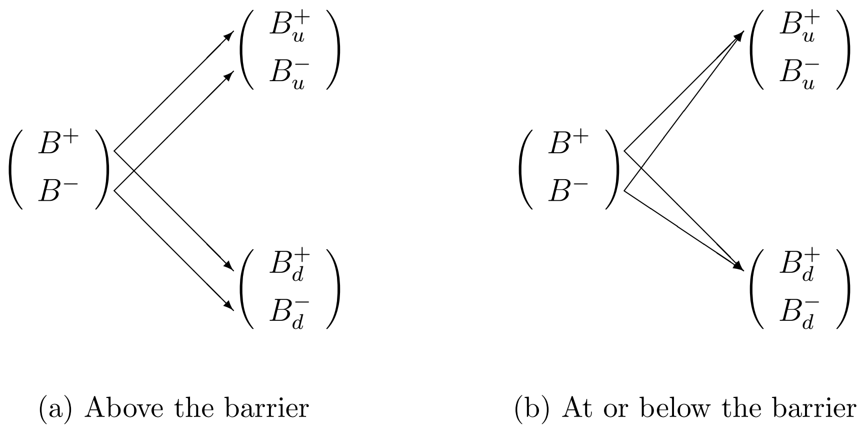

$\begin{pmatrix} B^+ \\ B^- \end{pmatrix}$
$\begin{pmatrix} B_u^+ \\ B_u^- \end{pmatrix}$
$\begin{pmatrix} B_d^+ \\ B_d^- \end{pmatrix}$
(a) Above the barrier
(b) At or below the barrier

Figure 10: Updating of the knock-in price. Above the barrier (a), the two entries are updated separately. At or below the barrier (b), a knock-in has occurred so both entries become the present value of the B^+ entries at the successor nodes.

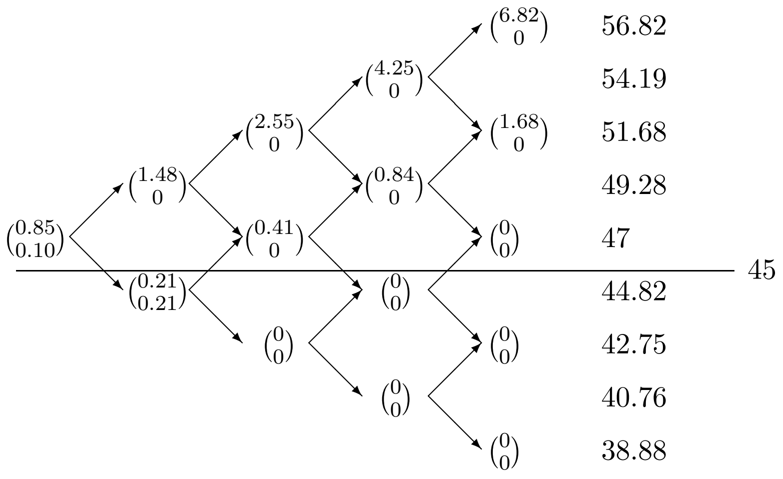


Figure 11: Binomial tree for down-and-in call. The initial asset price is 47, the strike is 50, and the barrier is at 45. At each node, the first number is the call value given that the barrier was previously crossed; the second number is the call value given that the barrier was not previously crossed. The option value at time 0 is 0.10.

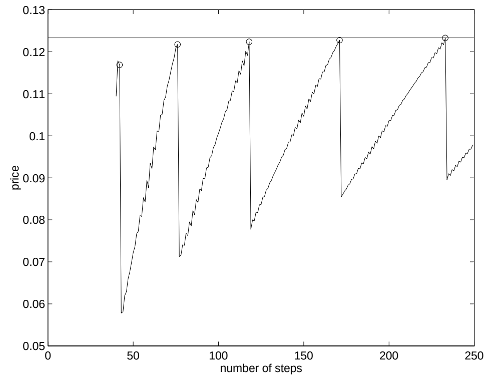


Figure 12: The jagged line is the binomial price of a down-and-in call as a function of the number of time steps n. The horizontal line gives the correct price. The points marked with a "$\circ$" correspond to values of n produced by equation (35).

at expiration. In the example of Figure 11, there is exactly one path that accomplishes this — the path that consists of a down move followed by three up moves. The probability of this path is $(1-p)p^3$ and its discounted payoff is $e^{-rT} \times 1.68$, so the option value should be

$$(1-p)p^3 e^{-rT} \times 1.68 = (.5013)^3(1-.5013)(.9900)(1.68) = 0.104$$

which agrees with the value in the tree (after rounding).

Prices generated by MATLAB program `doc_bin.m` for time steps ranging from 30 to 250 are illustrated in Figure 12. The parameters are the same ones used for Figure 9, except that the option is now knocked in at the barrier rather than knocked out. As in the case of a knock-out, the binomial price changes erratically with the number of time steps. However, for a knock-in, the binomial price consistently underestimates the true price of 0.1233, because it implicitly underestimates the probability of crossing the barrier. The best values of the number of time steps n are produced by the same formula (35) as before. These are marked with a "$\circ$" in the figure.

This pricing of a knock-in option illustrates one general approach to adapting the binomial method to path-dependent options: record additional information at each node to eliminate the path-dependence. By keeping track of B^+ and B^- at every node, we were able to update the option price at every step, without having to know whether or not the path by which the node was reached included a barrier crossing.

Why was it unnecessary to record additional information in pricing the knock-out? In case of a knock-out, we know the value of the option at a barrier crossing — it's 0, so we can substitute 0 for the option value at all nodes below the barrier. Thus, in pricing the knock-out, 0 plays the role played by B^+ in pricing the knock-in. We could have recorded 0 as a second value at every node in Figure 8, but there was no reason to. In contrast, with a knock-in option we do not know the option value at a barrier crossing; the option value depends on the future evolution of the underlying asset as well. So, we need to compute the B^+ values as we work backwards through the tree. If we knew the B^+ values in advance, we could have placed these at all nodes below the barrier (just as we put 0s below the barrier with the knock-out) and then recorded just one value at each node.

Lookback options

Our final example of resolving path-dependence in a binomial tree is the pricing of *lookback* options. A lookback call grants the holder the right to buy the underlying at the lowest price reached in the life of the option and to sell at the terminal price. It is therefore like a regular call in which the strike K is replaced with the minimum price $\min_{0\leq t\leq T} S_t$. Its payoff is

$$\max(0, S_T - (\min_{0\leq t\leq T} S_t))$$

Since $S_T \geq (\min_{0\leq t\leq T} S_t)$, the "max" in this expression is superfluous and the payoff can be written simply as

$$S_T - (\min_{0\leq t\leq T} S_t).$$

The payoff on a lookback put is

$$(\max_{0\leq t\leq T} S_t) - S_T.$$

The value of a lookback call at any time during the life of the option depends on the minimum price reached so far, and the value of a lookback put similarly depends on the maximum reached so far. Different paths through a tree reaching the same node will contribute different payoffs if their minimum or maximum prices so far are different. In this sense, lookbacks introduce even greater path dependence than our previous examples. For whereas in the case of a barrier option there are at most two possible option values at each node, in the case of a lookback the number of possible option values at a node equals the number of different minimum or maximum prices that could be reached prior to that node.

Consider the pricing of a lookback put. Suppose the current node lies k layers down from the top layer of nodes. This means that the stock price at the current node is $d^k(u^n S_0) = u^{n-k}S_0$. The possible values of the maximum stock price reached so far are $u^n S_0$, $u^{n-1}S_0$, …, $u^{n-k}S_0$; in particular, no price less than the current price could be the maximum so far. Let P^i denote the value of the put if the maximum reached so far is $u^{n-i}S_0$, for $i = 0, 1, \ldots, k$. Then at the current node we need to keep track of the entire vector

$$P = \begin{pmatrix} P^0 \\ P^1 \\ \vdots \\ P^{k-1} \\ P^k \end{pmatrix}$$

of possible option values. At the two successor nodes we have the values

$$P_u = \begin{pmatrix} P_u^0 \\ P_u^1 \\ \vdots \\ P_u^{k-1} \end{pmatrix} \qquad P_d = \begin{pmatrix} P_d^0 \\ P_d^1 \\ \vdots \\ P_d^{k-1} \\ P_d^k \\ P_d^{k+1} \end{pmatrix}.$$

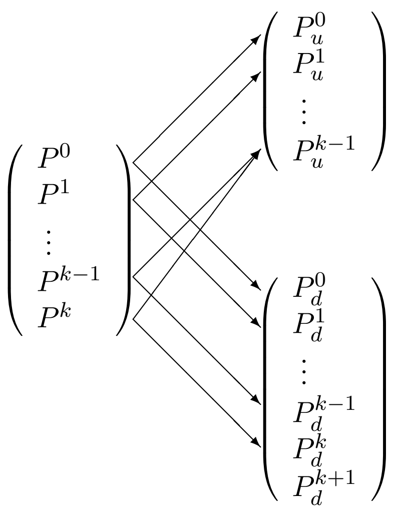


Figure 13: Illustration of recursive rule for pricing a lookback put in a binomial tree. The current node has underlying price $u^{n-k}S_0$. Each P^i denotes the put value if the maximum underlying price reached so far is u^{n-i}, $i = 0, 1, \ldots, k$.

The length of the vector at a node equals the number of node layers that lie at or above the current node.

It remains to find the rule by which the entries of P are computed from those of P_u and P_d. In computing P^i, $i = 0, 1, \ldots, k$, there are two cases:

(a) $i < k$: The maximum so far is greater than the current price. This means that the maximum so far will not change in the next time step, so

$$P^i = e^{-r\Delta t}[pP_u^i + (1-p)P_d^i].$$

(b) $i = k$. The maximum reached so far is the current price $u^{n-k}S_0$. If the price goes down in the next time step, the maximum is unchanged, but if the price goes up the new maximum becomes $u^{n-k+1}S_0$. So,

$$P^k = e^{-r\Delta t}[pP_u^{k-1} + (1-p)P_d^k].$$

These rules are illustrated in Figure 13.

At a terminal node, each option payoff P^i is equal to $u^{n-i}S_0 - S_T$, where S_T is the underlying price at the terminal node. From these terminal payoffs and the recursive rules above, we obtain payoff vectors at every intermediate node. In particular, at the initial node, we end up with $n+1$ options values because there are $n+1$ node layers at or above S_0. At time 0, the maximum so far is S_0, so the option price is the bottom number in the payoff vector.

This example illustrates the increasing complexity required by the binomial method to handle complex path dependence. Indeed, the method above should be viewed as pushing the limit of what can be done in a tree.[7] A formula for the price of a continuous lookback is given in Appendix A. Though a formula exists, numerical methods are necessary for lookbacks with discrete price fixings and for American lookbacks.

2.4. Options on Multiple Assets

Another important class of exotics are options whose payoffs depend on multiple assets. Some examples of option payoffs depending on two assets S^1, S^2 are the following. We list the call versions; the put versions are defined analogously.

- Option on the max: $\max(0, \max(S^1, S^2) - K)$
- Spread option: $\max(0, (S^2 - S^1) - K)$
- Portfolio option: $\max(0, (n_1 S^1 + n_2 S^2) - K)$
- Dual-strike option: $\max(0, S^1 - K_1, S^2 - K_2)$

A spread option with zero strike is an *exchange* option, granting the holder the right to exchange asset 1 for asset 2. More complex examples arise in options on foreign equities where the exchange rate then plays the role of the second asset. For example, a contract might specify that an option on a foreign stock is knocked out if the exchange rate rises above a barrier.

Options on multiple assets have been traded in the over-the-counter market for some time, and exchange-traded versions have recently emerged. In 1994, the New York Mercantile Exchange began listing options on the *crack spread* — the difference between heating-oil and crude-oil futures prices. The holder of this option can benefit even if both prices drop, so long as crude drops more than heating oil. Some other spread options on commodities and on Treasuries are actively traded as well. It is also worth noting that multi-asset options are often embedded in other instruments, such as futures contracts that allow multiple deliverables. For example, a Treasury futures contract has an embedded "option on the min," because the seller of the contract will deliver the cheapest admissible bond at maturity.

It should be clear that any method for pricing these types of options must keep track of the value of both underlying assets at each time step, and must, in particular, capture possible *correlations* in the movements of the two assets. Indeed, the proper modeling of correlation is the main technical issue arising in multi-asset options.

To keep track of two stocks in a binomial lattice, we replace our basic building block

$$\begin{tikzcd} & uS \\ S \arrow[ur, "p"] \arrow[dr, "1-p"'] & \\ & dS \end{tikzcd} \qquad (36)$$

[7]The method above was proposed by J. Hull and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Fall 1993. A different binomial method has been proposed by S. Babbs, "Binomial Valuation of Lookback Options," Working Paper, Midlands Global Markets, 1992.

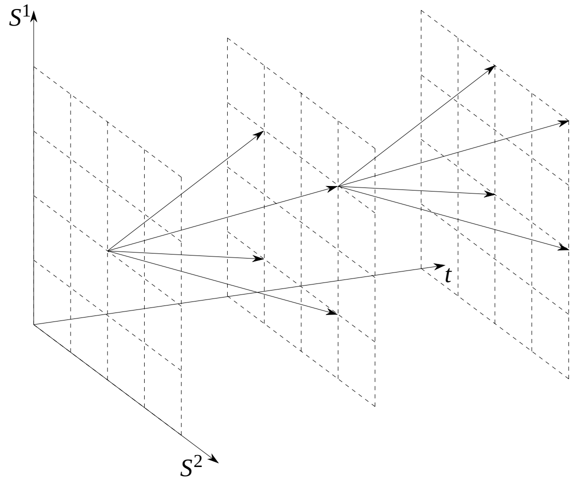


Figure 14: Evolution of a two-dimensional binomial tree. Each node has four successor nodes

with a new building block

$$\begin{tikzcd}
(d_1S^1, u_2S^2) & & (u_1S^1, u_2S^2) \\
& (S^1, S^2) \arrow[ul, "p_{du}"] \arrow[ur, "p_{uu}"'] \arrow[dl, "p_{dd}"'] \arrow[dr, "p_{ud}"] & \\
(d_1S^1, d_2S^2) & & (u_1S^1, d_2S^2)
\end{tikzcd} \tag{37}$$

Here, u_i, d_i are the 1-step increase or decrease in stock i, $i = 1, 2$. The probabilities p_{uu}, p_{ud}, p_{dd}, and p_{du} are the probabilities of the four possible combined moves of the two stocks in one step. For an ordinary binomial tree, we repeat the basic building block (36) at each node and get a two-dimensional lattice shaped like a triangle. In much the same way, repeating (37) at each node results in a three-dimensional lattice shaped like a pyramid. The evolution of the pyramid is partially illustrated in Figure 14. Each node has four successor nodes. For clarity, branching from the second time step to the third is shown in the figure for only one of the four nodes. Branching at the other nodes follows the same pattern.

It remains to specify the parameters in (37). Just as in a single-asset tree, we set

$$u_i = e^{\sigma_i \sqrt{\Delta t}}, \quad d_i = 1/u_i, \quad i = 1, 2,$$

where σ_i is the volatility parameter for stock i. In particular, by choosing $d_i = 1/u_i$ we ensure that an upmove and a downmore return the asset price S_i to its original value, regardless of the order in which they occur. (This is the case in Figure 14.)

Next, we choose the transition probabilities. These should satisfy the following conditions:

- They should give each asset an expected rate of return of r, to qualify as risk-netural probabilities.

- They should give asset i a volatility of approximately σ_i, the approximation becoming exact as $\Delta t \to 0$.
- They should make the correlation between the asset returns approximately the true correlation ρ, the approximation becoming exact as $\Delta t \to 0$.

There are many ways of choosing the probabilities consistent with these requirements; we present one such method.[8] To simplify notation, we first define

$$\nu_i = r - \frac{1}{2}\sigma_i^2, \quad i = 1, 2; \tag{38}$$

the expression on the right already appeared in the exponent of (10). We now set

$$\begin{aligned}
p_{uu} &= \frac{1}{4}\left(1 + \rho + \sqrt{\Delta t}\left(\frac{\nu_1}{\sigma_1} + \frac{\nu_2}{\sigma_2}\right)\right) \\
p_{ud} &= \frac{1}{4}\left(1 - \rho + \sqrt{\Delta t}\left(\frac{\nu_1}{\sigma_1} - \frac{\nu_2}{\sigma_2}\right)\right) \\
p_{du} &= \frac{1}{4}\left(1 - \rho + \sqrt{\Delta t}\left(-\frac{\nu_1}{\sigma_1} + \frac{\nu_2}{\sigma_2}\right)\right) \\
p_{dd} &= \frac{1}{4}\left(1 + \rho - \sqrt{\Delta t}\left(\frac{\nu_1}{\sigma_1} + \frac{\nu_2}{\sigma_2}\right)\right),
\end{aligned}$$

where ρ is the correlation between the returns on the two assets over a small time increment. These parameters give us all the information we need to implement the binomial pyramid.

Let's apply this to a two-stock example. Both initial stock prices are 100; the volatilities are $\sigma_1 = 0.30$ and $\sigma_2 = 0.20$; the correlation is $\rho = 0.50$; the riskless rate is 0.05; the time to expiration is $T = 1$ year. To illustrate, we divide this interval into $n = 2$ steps, so $\Delta t = 0.5$. These parameters result in

$$(u_1, d_1) = (1.2363,\ 0.8089) \quad (u_2, d_2) = (1.1519,\ 0.8681) \tag{39}$$

and

$$(p_{uu}, p_{ud}, p_{dd}, p_{du}) = (0.4045,\ 0.1014,\ 0.3455,\ 0.1486). \tag{40}$$

These probabilities are assigned to the branches in Figure 14 according to the convention illustrated in (37). In Figure 15, we verify that the values in (39) and (40) do indeed result in reasonable approximations to the asset volatilities and correlation.

Based on these parameters, we find that the possible terminal stock values for S^1 are

$$(u_1^2 S^1, S^1, d_1^2 S^1) = (152.85, 100, 65.42)$$

and for S^2 they are

$$(u_2^2 S^2, S^2, d_2^2 S^2) = (132.69, 100, 75.36).$$

[8]This method is from P. Boyle, J. Evnine, and S. Gibbs, "Numerical Evaluation of Multivariate Contingent Claims," *Review of Financial Studies,* vol.2, pp.241-250, 1989.

The evolution of the pairs (S^1, S^2) of possible stock prices looks like this:

$$[100, 100] \begin{bmatrix} (80.89, 115.19) & (123.63, 115.19) \\ \\ (80.89, 86.81) & (123.63, 86.81) \end{bmatrix} \begin{bmatrix} (65.42, 132.69) & (100, 132.69) & (152.85, 132.69) \\ \\ (65.42, 100) & (100, 100) & (152.85, 100) \\ \\ (65.42, 75.36) & (100, 75.36) & (152.85, 75.36) \end{bmatrix}$$

Once we know the evolution of the possible pairs (S^1, S^2) of asset prices, we can price many types of options. Let's consider the pricing of a max call with strike $K = 100$. The intrinsic value is

$$\max\{\max\{S^1, S^2\} - K, 0\} = \max\{\max\{S^1, S^2\} - 100, 0\}.$$

To get the possible terminal payoffs, we apply this formula to the nine terminal nodes, corresponding to the nine possible pairs of stock prices after two steps. These are displayed in the rightmost array above. In the upper-left corner, we get

$$\max\{\max\{65.42, 132.69\} - 100, 0\} = 32.69;$$

in the upper-right corner we get

$$\max\{\max\{152.85, 132.69\} - 100, 0\} = 52.85;$$

and in the lower-left corner we get

$$\max\{\max\{65.42, 75.36\} - 100, 0\} = 0.$$

Filling in the other cases, we arrive at the following array of terminal payoffs:

$$\begin{bmatrix} 32.69 & 32.69 & 52.85 \\ 0 & 0 & 52.85 \\ 0 & 0 & 52.85 \end{bmatrix}$$

We now work backwards through the pyramid. At each node, we need to calculate the expected present value of the prices at the *four* successor nodes. The evolution of the option-value pyramid looks like this:

$$[17.29] \qquad \begin{bmatrix} 17.63 & & 30.81 \\ & \times & \\ 0 & & 26.07 \end{bmatrix} \qquad \begin{bmatrix} 32.69 & & 32.69 & & 52.85 \\ & \times & & \times & \\ 0 & & 0 & & 52.85 \\ & \times & & \times & \\ 0 & & 0 & & 52.85 \end{bmatrix} \tag{41}$$

The third array repeats the terminal payoffs. The dots indicate where the present values should be computed. Specifically, we need to compute the expected present value of the four prices surrounding each dot, using the probabilities p_{du}, p_{uu}, p_{ud}, and p_{dd}, as indicated in (37). For

Let's verify that the transition probabilities calculated in (40) and the u_i, d_i calculated in (39) do indeed approximate the desired rate of return $r = 0.05$, volatilities $\sigma_1 = 0.30, \sigma_2 = 0.20$ and correlation $\rho = 0.50$. Define

$$X_i = \text{1-step return on } S_i,\ i = 1, 2.$$

Then

$$X_1 = \begin{cases} u_1, & \text{with probability } p_{uu} + p_{ud} \\ d_1, & \text{with probability } p_{du} + p_{dd} \end{cases} \qquad X_2 = \begin{cases} u_2, & \text{with probability } p_{uu} + p_{du} \\ d_2, & \text{with probability } p_{ud} + p_{dd} \end{cases}$$

Thus,

$$\begin{aligned} E[X_1] &= (p_{uu} + p_{ud})u_1 + (p_{du} + p_{dd})d_1 \\ &= (0.4045 + 0.1014)1.2363 + (0.1486 + 0.3455)0.8089 = 1.025 \\ E[X_2] &= (p_{uu} + p_{du})u_2 + (p_{ud} + p_{dd})d_2 \\ &= (0.4045 + 0.1486)1.1519 + (0.1014 + 0.3455)0.8681 = 1.025. \end{aligned}$$

These match the desired expected return $e^{r\Delta t} = e^{(.05)(.5)} = 1.025$.
To check the volatilities, we first compute the return variances. Recall that $\text{Var}[X_i] = E[X_i^2] - (E[X])^2$. Thus,

$$\begin{aligned} \text{Var}[X_1] &= (p_{uu} + p_{ud})u_1^2 + (p_{du} + p_{dd})d_1^2 - (E[X_1])^2 \\ &= (0.4045 + 0.1014)(1.2363)^2 + (0.1486 + 0.3455)(0.8089)^2 - (1.025)^2 \\ &= 0.0457. \end{aligned}$$

An analogous calculation gives $\text{Var}[X_2] = 0.0199$. To annualize these parameters we divide by Δt; to convert variances to volatilities we take square-roots:

$$\begin{aligned} \sqrt{\text{Var}[X_1]/\Delta t} &= \sqrt{.0457/0.5} = 0.302 \\ \sqrt{\text{Var}[X_2]/\Delta t} &= \sqrt{.0199/0.5} = 0.199. \end{aligned}$$

These are close to the target values of 0.30 and 0.20.
To check the correlation, we first compute the covariance. Recall that $\text{Cov}[X_1, X_2] = E[X_1X_2] - E[X_1]E[X_2]$. Thus,

$$\begin{aligned} \text{Cov}[X_1, X_2] &= p_{uu}u_1u_2 + p_{ud}u_1d_2 + p_{dd}d_1d_2 + p_{du}d_1u_2 - E[X_1]E[X_2] \\ &= .4045(1.2363)(.8089) + .1014(1.2363)(.8681) + .3455(.8089)(.8681) \\ &\quad + .1486(.8089)(1.1519) - (1.25)^2 = 0.0151. \end{aligned}$$

Finally,

$$\begin{aligned} \text{Corr}[X_1, X_2] &= \text{Cov}[X_1, X_2]/\sqrt{\text{Var}[X_1] \cdot \text{Var}[X_2]} \\ &= 0.0151/\sqrt{0.0457 \cdot 0.0199} = 0.502, \end{aligned}$$

which is close to the target value of 0.50. The parameters for a smaller Δt can be expected to yield an even better approximation.

Figure 15: Verification of correlation approximation in two-dimensional tree

example, the upper-right entry in the second array of (41) is the present value of the four entries in the upper-right box of the third array:

$$30.81 = e^{-r\Delta t}(p_{du} * 32.69 + p_{uu} * 52.85 + p_{ud} * 52.85 + p_{dd} * 0).$$

The first number in (41), 17.29, is similarly the present value of the entries in the second array.

A multi-step, two-dimensional binomial tree can be implemented in a spreadsheet by putting each step of the tree on a different spreadsheet page. Figure 16 illustrates this technique. In the figure, Page A is used to set up the transition probabilities, the u_i and the d_i, and the possible stock prices. The S^1 prices are in row 10 and the S^2 prices are in column A. The technique described to generate stock prices in Figure 4 applies here as well. For each combination of S^1 price and S^2 price on Page A, we need a terminal payoff. The exact formula depends on the type of option; the figure illustrates a max option. Using absolute (with-\$) and relative (without-\$) addresses as indicated, a single formula can be copied to all other cells. Page B illustrates the present value calculation one step back from expiration. The averaged values are taken from Page A, and their positions relative to the current cell match the relative positions in (37). (Defining the discount factor $D = e^{-r\Delta t}$ on Page A and using its cell address in the present value calculation greatly reduces the number of calls to the exponential function and saves recalculation.) The displayed formula on Page B can be copied throughout the relevant range of Page B, *and then to subsequent pages as well.* This simplifies the construction of the spreadsheet. The present value calculation at "boundary" cells (e.g., column B and row 11 in the figure) will not be correct, but this will not interfere with the calculations at the cells of interest. The current price of the option will be the middle value (corresponding to the current S^1 and S^2) on the final page.

It is convenient (though not strictly necessary) to repeat the column-A and row-10 stock prices on each pages. This makes it easy to compute the option's intrinsic value at each cell and thus to extend the method to price American options.

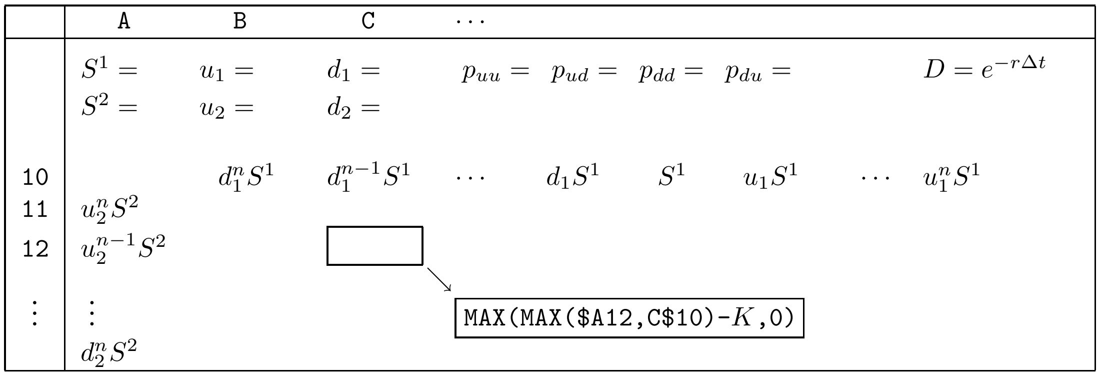
Figure 26 in Appendix B shows a MATLAB implementation `binom_2d.m` of the two-dimensional binomial method to price options on two assets.

3. Simulation Option Pricing

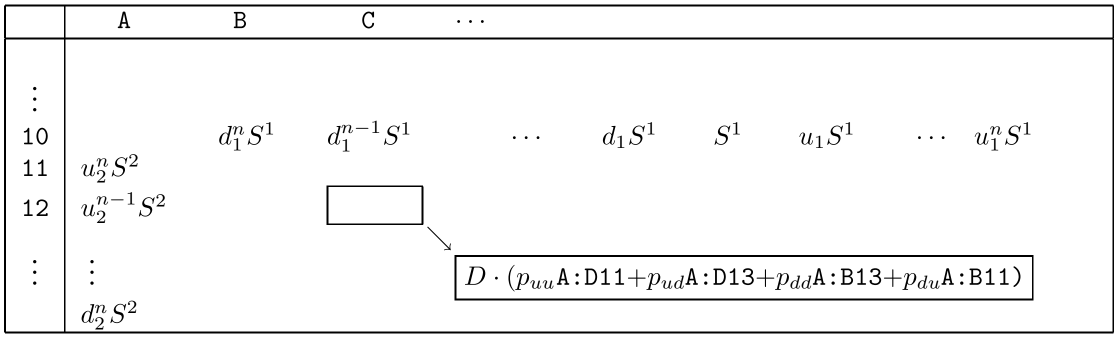
Thus far, we have seen how one basic technique — the binomial method — can be adapted to price a variety of options. We now discuss another method, *Monte Carlo simulation,* as a general tool for pricing options. We describe its advantages and disadvantages compared with binomial methods.

3.1. European Calls and Puts

As we did for binomial pricing (in Section 2.1), we begin our discussion of simulation pricing with European calls and puts. These options can, of course, be priced directly from the Black-Scholes formula, so neither the binomial method nor simulation is required; however, this simple example serves to introduce key ideas.


A B C ···
$S^1 =$ $u_1 =$ $d_1 =$ $p_{uu} =$ $p_{ud} =$ $p_{dd} =$ $p_{du} =$ $D = e^{-r\Delta t}$
$S^2 =$ $u_2 =$ $d_2 =$
10 $d_1^n S^1$ $d_1^{n-1} S^1$ ··· $d_1 S^1$ S^1 $u_1 S^1$ ··· $u_1^n S^1$
11 $u_2^n S^2$
12 $u_2^{n-1} S^2$
$d_2^n S^2$
MAX(MAX($A12,C$10)-K,0)

Page A: Terminal Payoffs


A B C ···
10 $d_1^n S^1$ $d_1^{n-1} S^1$ ··· $d_1 S^1$ S^1 $u_1 S^1$ ··· $u_1^n S^1$
11 $u_2^n S^2$
12 $u_2^{n-1} S^2$
$d_2^n S^2$
$D \cdot (p_{uu}$A:D11+p_{ud}A:D13+p_{dd}A:B13+p_{du}A:B11)

Page B: Present value calculation one step back

Figure 16: Schematic spreadsheet for two-dimensional binomial tree. Page A gives terminal payoffs for all pairs of terminal asset prices. The displayed formula can be copied to all other cells corresponding to an S^1 column and an S^2 row. Page B illustrates the present value calculation one step back using the terminal payoffs and the discount factor $D = e^{-r\Delta t}$ defined on Page A. This formula can be copied to other cells and then to other pages as well, to build a multi-step tree.

Pricing a standard European call means evaluating the expected value

$$C = E[e^{-rT} \max\{S_T - K, 0\}].$$

To approximate this expected value by simulation, we need to generate a sequence of terminal stock prices $S_T^1, S_T^2, \dots, S_T^n$, where n is the number of simulation *replications* we want to carry out. We then evaluate the discounted option payoff $e^{-rT} \max\{S_T^i - K, 0\}$ for each replication and average over replications to get an approximation to the option price.

Each S_T^i can be generated from the formula

$$S_T^i = S_0 e^{(r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T} Z_i}, \tag{42}$$

where $Z_1, Z_2, \dots$ are independent standard normal random variables. In EXCEL, the command `=NORMSINV(RAND())` generates a sample from the standard normal distribution.[9] In MATLAB, the command `randn` produces a standard normal, and the command `randn(n,m)` produces an $n \times m$ matrix of independent standard normals.

To each terminal stock price S_T^i, there corresponds a call-option payoff $\max\{0, S_T^i - K\}$, and discounted payoff $C_i = e^{-rT} \max\{0, S_T^i - K\}$. Our price estimate is simply the average of these discounted payoffs over the n replications:

$$\hat{C} = \frac{1}{n} \sum_{i=1}^{n} C_i = \frac{1}{n} \sum_{i=1}^{n} \left(e^{-rT} \max\{0, S_T^i - K\} \right). \tag{43}$$

To price a European put, we would generate the terminal stock prices in the same way but replace the payoff in (43) with $\max\{0, K - S_T^i\}$. All the simulation-based estimates we will encounter will have the same general form as (43): they will be *sample means* over some number n of *independent replications* of discounted payoffs. Regardless of the choice of n, the estimator in (43) is *unbiased*, meaning that $E[\hat{C}]$ is the true option value C. For large n, the error in the estimator $\hat{C}$ is approximately normally distributed with variance σ_C^2/n, where $\sigma_C^2 = \mathrm{Var}[C_i]$. We can estimate this variance using the sample variance

$$s_C^2 = \frac{1}{n-1} \sum_{i=1}^{n} (C_i - \hat{C})^2,$$

and then form the $(1-\alpha)$ confidence interval

$$\hat{C} \pm z_{\alpha/2} \frac{s_C}{\sqrt{n}} \tag{44}$$

for the option price. As usual, $z_{\alpha/2}$ is defined by the condition $P(Z > z_{\alpha/2}) = \alpha/2$ and can be obtained from the normal table. For a 95% confidence interval, set $\alpha = .05$ and $z_{\alpha/2} = z_{.025} \approx 1.96$.

[9]After generating samples using these commands, you probably want to *freeze* their values. Otherwise, the values will be resampled with every recalculation of the spreadsheet. In EXCEL, copy the samples and then use `Paste Special...Values`.

Figure 32 in Appendix B shows a MATLAB program `bs_sim.m` that estimates European call option prices using simulation. This program automatically computes standard errors.[10] To estimate a European put price, change line 9.

It is apparent from (44) that the larger we make n, the closer our estimate will be to the true value, on average, as we would expect. However, it also follows from (44) that the *rate* at which the error decreases is rather slow. For example, to cut our uncertainty (as measured by $s_C/\sqrt{n}$) in half, we need to quadruple n; to cut our uncertainty by a factor of 10 (i.e., to go from dime-accuracy to penny-accuracy) we need to increase n by a factor of 100. Empirical evidence suggests that the binomial method actually has the same convergence rate, but simulation typically requires a lot more computing time to get equally precise values.

Based on this simple example, we can already make a few observations on the difference between simulation and the binomial method:

1. Simulation can work directly with the continuous lognormal model of stock prices, whereas all tree methods require some sort of discrete approximation.
2. Prices computed by simulation are statistical estimates and can therefore be analyzed by statistical methods. In particular, simulation produces information about the likely error in a price estimate, via confidence intervals. Binomial methods do not provide any information about likely errors.
3. When both simulation and binomial methods are applicable, binomial methods generally require less computing time to produce accurate results.

Much of the rest of our discussion aims towards elaborating these issues. We can already add something to the last point: the main problems to which simulation is not easily applied are American options. To price an option by simulation, we typically need to be able to generate samples of the stock price at the time the option is exercised. For a European option, the exercise time is the terminal time, and since we know the distribution of the stock price at the terminal time, we can generate stock values at that time. With an American option, we generally do not know when the option will be exercised and so cannot directly generate sample stock values at the exercise time.

3.2. Path-Dependent Options

Simulation sometimes beats binomial methods in pricing complex exotic options. Recall that a (European) path-dependent option is one whose payoff depends not just on the terminal stock price, but on the *path* by which the stock price reached its terminal value. In a simulation, we can divide the time-to-maturity T of an option into m increments of length $\Delta t = T/m$ and generate a path

$$S_0, S_{\Delta t}, S_{2\Delta t}, \ldots, S_{(m-1)\Delta t}, S_{m\Delta t} = S_T \tag{45}$$

[10]Rather than fix a confidence level, the MATLAB program provides the standard error. You can multiply this standard error by a z value, as in (44), to get a desired confidence level. An interval halfwidth of one standard error corresponds to a confidence level of roughly 68%.

that approximates the continuous-time path of stock prices. From this simulated path, we can then determine the payoff of a path-dependent option. To generate the $(k+1)^{\text{th}}$ stock price along a path from the k^{th}, we use the formula

$$S_{(k+1)\Delta t} = S_{k\Delta t} e^{(r-\frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z_{k+1}},$$

where $Z_1, \dots, Z_m$ are independent standard normals. Repeating this procedure with different Z's n times yields n independent stock paths of m steps each. (*Comment on notation:* In discussing binomial methods, we used n for the number of time steps. In the simulation context, we prefer to use n for the number of simulation runs, and so use m for the number of time steps.)

Barrier Options Revisited

As a first illustration of how simulation might be applied to a path-dependent option, consider again the knock-in option of Section 2.3. Given a path (45), we can compute the option payoff for the path as follows. First, we need to check whether this path ever crossed the knock-in barrier B; i.e., whether or not

$$\min(S_0, S_{\Delta t}, S_{2\Delta t}, \dots, S_{(m-1)\Delta t}, S_T) \leq B. \qquad (46)$$

If it was crossed, then the option payoff (in the case of a call) is simply

$$\max(0, S_T - K);$$

otherwise, the payoff is zero. Repeating this for n paths, then averaging and discounting yields an estimate of the option value. Figure 34 in the Appendix shows a MATLAB program that carries this out. Line 18 checks if the barrier was crossed and records the option payoff if it was.

A moment's thought suggests that if a knock-in or knock-out may occur at any instant in the life of the option, then the number of time increments m will have to be quite large for this method to yield accurate results. The simulation checks whether or not the knock-in barrier is reached only at times $0, \Delta t, 2\Delta t, \dots, (m-1)\Delta t, T$. If m is not very large, then Δt is not very small, and it is certainly possible that the continuous-time stock path drops below the barrier *between* two times $k\Delta t$ and $(k+1)\Delta t$ but not at either of these times. As a result, the simulated discrete-time path may fail to cross the barrier even when the continuous-time path does cross. Moreover, the option payoff is extremely sensitive to this distinction: if (46) fails to hold, the estimated option payoff is zero, whereas if the continuous-time path did cross the barrier the payoff could be quite large. As a result of these considerations, simulation requires a large m and therefore quite a bit of computing time to produce accurate price estimates. The binomial method does much better in this case.

The situation is different in pricing contracts specifying that knock-ins and knock-outs can occur only at specified times — e.g., at daily closings. In this case, simulation can be applied directly by taking the time step Δt to be the interval between instants at which the barrier is monitored — e.g., 1 day. In contrast, discrete monitoring presents some difficulty for the binomial method. On one hand, we would like the monitoring interval to coincide with an integer multiple of the tree's time step, and this puts a restriction on the number of steps in the tree. On the other hand, we saw previously that only carefully chosen values for the number of time steps result in accurate prices. In this sense, simulation has an advantage over the binomial method in pricing discretely monitored barrier options.[11] Simulation is also better suited to pricing options with time-varying barriers and options on assets best modeled as having time-varying volatilities.

Lookback Options

Pricing lookbacks is somewhat simpler using simulation than using the binomial method. To price a lookback call by simulation, we first generate a path (45). We then compute the payoff

$$S_T - (\min_{0 \leq k \leq m} S_{k\Delta t}).$$

Repeating this for n paths, then averaging and discounting yields our estimate. Figure 35 in the Appendix shows a MATLAB program `lb_sim.m` that carries this out. To price a put rather than a call, change line 18.

As in the case of a barrier option, the fact that we are using a discrete-time path to approximate a continuous-time path introduces some error, if the contract is based on continuous prices rather than discrete fixings. If the contract specifies discrete fixings (and this is typically the case) the simulation estimator is unbiased. However, even for lookbacks based on continuous prices, the error from a discrete approximation is not as severe as for a barrier option. With a barrier option, even a very small change in the stock path can introduce a large change in the option payoff. With a lookback, a small change in the stock path introduces only a small change in the minimum and maximum stock prices achieved, and thus only a small change in the option payoff.

3.3. Options on Multiple Assets

In Section 2.4, we showed how to price options on two assets using a binomial lattice by branching in two dimensions rather than one. Suppose we want to price an option whose payoff depends on, say, stocks in two foreign countries. Attaching a dollar value to the option could well require keeping track of two stocks and two exchange rates, or a total of four state variables.[12] Pricing such an option on a binomial lattice requires branching in four dimensions. For a fixed number of times steps, the number of nodes in a lattice grows exponentially in the number of branching dimensions, indicating that binomial methods become infeasible as the number of state variables increases.

In contrast, simulation is not fundamentally more difficult with multiple state variables than with a single state variable. Suppose we want to price a European option whose payoff depends on the terminal values

$$S_T^{(1)}, S_T^{(2)}, \ldots, S_T^{(k)}$$

of k assets or state variables. All we need to do is generate samples of these terminal values, compute the option payoff, discount, and average over multiple samples.

[11] A *trinomial* tree, with three branches at each node, has an extra degree of freedom. This makes it possible to put a layer of nodes on the barrier for all choices of the number of time steps and thus to accommodate discrete monitoring.

[12] The term *state variable* refers to information required to price a derivative security. Some state variables may not correspond directly to assets, like exchange rates, interest rates, or fluctuating volatility parameters. We generally ignore this distinction and use the terms *multiple state variables* and *multiple assets* interchangeably.

Implementing this approach would require no new ideas if the asset prices were independent. For if they were independent, we could set

$$S_T^{(i)} = S_0^{(i)} e^{(r-\frac{1}{2}\sigma_i^2)T+\sigma_i\sqrt{T}Z_i}, \quad i = 1, \ldots, k, \tag{47}$$

where $Z_1, \ldots, Z_k$ are independent standard normals, and $\sigma_1, \ldots, \sigma_k$ are the return volatilities for the assets. However, asset prices are in general correlated, and capturing this correlation is an important part of correctly pricing options on multiple assets.

Let's begin with the case of two assets. To simulate one time step, we need to generate a pair of standard normal random variables X_1 and X_2 having the desired correlation ρ. If we start with two *independent* standard normals, Z_1 and Z_2, and then define

$$\begin{aligned} X_1 &= Z_1 \\ X_2 &= \rho Z_1 + Z_2\sqrt{1-\rho^2} \end{aligned} \tag{48}$$

then (X_1, X_2) will have the desired joint distribution. In particular, X_1 and X_2 are normally distributed because they are *linear transformations* of standard normals. Clearly, X_1 is a standard normal; for X_2, we have $E[X_2] = 0$ and, because the Zs are independent,

$$\begin{aligned} \mathrm{Var}[X_2] &= \mathrm{Var}[\rho Z_1 + Z_2\sqrt{1-\rho^2}] \\ &= \rho^2 \mathrm{Var}[Z_1] + (1-\rho^2)\mathrm{Var}[Z_2] \\ &= \rho^2 + (1-\rho^2) = 1. \end{aligned}$$

Most importantly, (48) gives (X_1, X_2) the right correlation. Because X_1 and X_2 have variance 1, their correlation equals their covariance, and for this we get

$$\begin{aligned} \mathrm{Cov}(X_1, X_2) &= \mathrm{Cov}(Z_1, \rho Z_1 + Z_2\sqrt{1-\rho^2}) \\ &= \mathrm{Cov}(Z_1, \rho Z_1) = \rho \mathrm{Cov}(Z_1, Z_1) = \rho, \end{aligned}$$

as required. The transformation (48) is easily implemented in a spreadsheet.

Just as (48) generates (X_1, X_2) through a linear transformation of (Z_1, Z_2), the returns on any number of correlated assets can be generated through a linear transformation of equally many independent normals. Suppose that the correlations among k assets are described by the $k \times k$ matrix

$$\boldsymbol{\rho} = \begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1k} \\ \rho_{21} & 1 & \cdots & \rho_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{k1} & \rho_{k2} & \cdots & 1 \end{pmatrix}.$$

The ij-entry of this matrix is the correlation ρ_{ij} between assets i and j. In (47), instead of using independent Z_i's, we now want to use dependent normal variables $X_1, \ldots, X_k$, with X_i and X_j having correlation ρ_{ij}.

To express the *multivariate normal* random variables $(X_1, \dots, X_k)$ as linear transformations of independent normals we need to find cofficients A_{ij}, $i, j = 1, \dots, k$, for which we can write

$$\begin{aligned} X_1 &= A_{11}Z_1 + \cdots + A_{1k}Z_k \\ X_2 &= A_{21}Z_1 + \cdots + A_{2k}Z_k \\ \vdots & \qquad \vdots \\ X_k &= A_{k1}Z_1 + \cdots + A_{kk}Z_k, \end{aligned} \tag{49}$$

and get the desired variances and correlations. In MATLAB, the required matrix A is easily computed[13] from the correlation matrix $\boldsymbol{\rho}$ and the volatilities. Once we have the X_i's, we generate terminal asset prices from the formula

$$S_T^{(i)} = S_0^{(i)} \exp((r - \frac{1}{2}\sigma_i^2)T + \sigma_i\sqrt{T}X_i), \quad i = 1, \dots, k. \tag{50}$$

We then apply the option payoff to these prices, discount, and average over multiple simulation runs to get our estimated option price. Figure 36 in Appendix B shows a MATLAB program `multisim.m` that carries this out. Changing the type of option (i.e., the payoff) requires modifying only line 17; in the figure, the program is set up for a call option on the maximum asset price. To run a spreadsheet simulation with more than two assets, you can use line 8 of the MATLAB program to get the A matrix, and then use these coefficients in a spreadsheet.

Taken together, (49) and (50) have an economic interpretation that is also relevant to other contexts. The Z_i's can be interpreted as independent *factors* or sources of uncertainty in the economy. The coefficient A_{ij} determines how strongly factor j influences X_i, which in turn determines the prices of asset i via (50). So, asset prices are influenced by the factors, and the asset prices are correlated because they are influenced by the same factors.

3.4. Variance Reduction Techniques

As we discussed earlier, the confidence interval (44) on a simulation estimate gives some indication of the precision of the estimate. The key measure of precision is the standard error $s_C/\sqrt{n}$. The simplest way to increase precision is therefore to increase the number of simulated replications n.

A risk-management office that needs to price, say, 1000 options in 1 hour at the end of each day can allocate only about one-third of a second to pricing each option, which limits the size of n. The only way to increase n is to buy a faster machine, or buy two machines and let them run at the same time.

But increasing n is only one way to increase precision; the other way is to make s_C smaller. Methods for doing this are called *variance reduction techniques.* We briefly discuss two such methods.

[13]Any matrix A for which A times its transpose gives the desired covariance matrix will do. It is customary to choose the unique lower-triangular A with this property, called the *Cholesky* matrix. The MATLAB function `chol` produces this matrix; see Figure 36.

Antithetic Variates

This method rests on the observation that if Z is a standard normal random variable, then so is $-Z$. To price a European call, in (42) and (43) we set

$$\begin{aligned} S_T^i &= S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_i} && (51)\\ C_i &= e^{-rT}\max\{0, S_T^i - K\}; && (52)\end{aligned}$$

but we could just as well set

$$\begin{aligned} \tilde{S}_T^i &= S_0 e^{(r-\frac{1}{2}\sigma^2)T-\sigma\sqrt{T}Z_i}\\ \tilde{C}_i &= e^{-rT}\max\{0, \tilde{S}_T^i - K\}.\end{aligned}$$

The values $\tilde{C}_1, \ldots, \tilde{C}_n$, generated from $-Z_1, \ldots, -Z_n$ are equally valid samples, in the sense that the $\tilde{C}_i$'s have the same distribution as the C_i's. The averaged pairs

$$\bar{C}_i = \frac{1}{2}(C_i + \tilde{C}_i), \quad i = 1, \ldots, n,$$

have the same expected value as the C_i's and $\tilde{C}_i$'s but will typically have lower variance. So, the sample mean

$$\frac{1}{n}\sum_{i=1}^{n}\bar{C}_i \tag{53}$$

typically has lower variance than the original estimator

$$\frac{1}{n}\sum_{i=1}^{n}C_i. \tag{54}$$

Of course, (53) is also based on twice as many samples, so for (53) to be preferred over (54) its variance should be less than half that of (54). This requires $\text{Var}[\bar{C}_i] < \frac{1}{2}\text{Var}[C_i]$; i.e.,

$$\frac{1}{4}(\text{Var}[C_i] + \text{Var}[\tilde{C}_i] + 2\text{Cov}[C_i, \tilde{C}_i]) < \frac{1}{2}\text{Var}[C_i].$$

But because C_i and $\tilde{C}_i$ have the same distribution, they, also have the same variance. The requirement thus simplifies to $\text{Cov}[C_i, \tilde{C}_i] < 0$. In other words, each sample C_i should be negatively correlated with its antithetic mate $\tilde{C}_i$. Since they were generated from Z_i and $-Z_i$, respectively, it is reasonable to expect that they are indeed negatively correlated.

Control Variates

Recall that when we simulate stock prices to value an option, we simulate the prices in a risk-neutral world; i.e., we simulate the stocks with rate of return r, the riskless rate. This means that

$$S_0 = E[e^{-rT}S_T]; \tag{55}$$

i.e., the current stock price is the expected present value of the terminal stock price. In carrying out the simulation, we generate a collection $S_T^1, S_T^2, \ldots, S_T^n$ of samples of the terminal stock price, given the known initial price S_0. Equation (55) indicates that the sample mean

$$\bar{S} = \frac{1}{n}\sum_{i=1}^{n} S_T^i$$

should be close to $e^{rT}S_0$. Of course, on any particular set of runs $\bar{S}$ will not exactly equal $e^{rT}S_0$. Now here is the key observation: *the error $\bar{S} - e^{rT}S_0$ we observe in the simulated stock price contains information about the unknown error in the simulated option price.* Intuitively, if $\bar{S} > e^{rT}S_0$, the simulation has probably over-valued call options and under-valued put options. If $\bar{S} < e^{rT}S_0$, the errors should go in the opposite direction.

The control variate method exploits this information in a precise way. To be concrete, consider the estimate $\hat{C}$ defined in (43), and recall that this estimate is unbiased. For any choice of coefficient b, another unbiased estimate is provided by

$$\hat{C}_{\text{cv}} = \hat{C} + b(e^{rT}S_0 - \bar{S});$$

the term we have added has zero expected value. Among all such estimators, the most precise one minimizes the variance

$$\text{Var}[\hat{C}_{\text{cv}}] = \text{Var}[\hat{C}] + b^2\text{Var}[\bar{S}] - 2b\text{Cov}[\hat{C}, \bar{S}].$$

The optimal coefficient is therefore

$$b^* = \frac{\text{Cov}[\hat{C}, \bar{S}]}{\text{Var}[\bar{S}]}.$$

We recognize this coefficient as the slope of a regression line[14] with the simulated option values $C_1, C_2, \ldots, C_n$ as dependent variable and the simulated terminal stock prices S_T^1, S_T^2, $\ldots$, S_T^n as explanatory variable. We can therefore find the variance-minimizing coefficient b^* by carrying out a regression on these simulated data points. Using this coefficient we get the adjusted estimate

$$\hat{C}^*_{\text{cv}} = \hat{C} + b^*(e^{rT}S_0 - \bar{S}), \tag{56}$$

which is *guaranteed*[15] to have lower variance than the original estimate $\hat{C}$. Figure 17 illustrates a spreadsheet implmentation of the control variate technique.

Figure 18 shows the output of a simulation that illustrates the performance of these variance reduction techniques. Because the simulation is pricing a standard call option, we can compare the simulation estimates with the true value, also displayed in the output. Notice that the simulated stock price is slightly higher than the true initial price of 55. As predicted, this means that the straightforward estimator has overvalued the option. The control variate method adjusts this

[14]Recall that a similar ratio also gives the optimal hedge ratio for regression hedging. See *Introduction to Hedging.*

[15]Strictly speaking, this guarantee holds only if we know b^*. In practice, we estimate b^* from simulated data. This makes it theoretically possible that the variance will increase, though this is very unlikely. Also, using the same data to estimate b^* and to evaluate $\bar{S}$ in (56) introduces a bias in the estimator $\hat{C}^*_{\text{cv}}$. This bias vanishes as the number of observations increases.

	A	B	C	D
	$S_0 =$	$\sigma =$	$r =$	
	$K =$	$T =$	$b^* =$	
1	Z_1	S_T^1	C_1	
2	Z_2	S_T^2	C_2	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
n	Z_n	S_T^n	C_n	
		$\bar{S}$ = AVERAGE(B1:Bn)	C_{cv} = AVERAGE(C1:Cn)+$b^*(e^{rT}S_0 - \bar{S})$	

Figure 17: Spreadsheet for simulation with control variate. Column B is calculated from (51), column C from (52), and b^* from a regression of column C against column B.

```
annual volatility              0.35
riskless interest rate         0.06
strike price                   50
current stock price            55
option expiration (in years)   0.5

number of simulation trials        1000
simulated stock price              55.0083 +/- 0.433175
black-scholes call option value    8.97191
simulated call price               8.98124 +/- 0.348425
simulated call price with control  8.97485 +/- 0.105586
antithetic call price              9.06188 +/- 0.225048
```

Figure 18: Illustration of variance reduction techniques

estimate downward slightly. Most important is the impact on the standard error. The control variate has reduced the standard error by roughly a factor of three. This is as good as increasing the number of replications from 1000 to 9000 with no significant additional computing time. So, in this example, knowing how to use the control variate method is worth as much as a machine that works 9 times as fast. How much would it cost you to upgrade your computer to one that runs 9 times as fast?

In the example of Figure 18, the antithetic variates method also reduces variance but not as much as the control variate method. This is usually the case, at least if the control chosen (in this case the stock price) is closely related to the quantity estimated (the option price).

3.5. Summary

We conclude by summarizing some important considerations in pricing methods for options.

1. When a formula is available for an option price (like the Black-Scholes formula for standard European options), it is the best method for evaluating the price. However, the class of options for which formulas are available is quite limited.

2. In most cases in which a formula is not available, binomial methods are faster than simulation (for the same level of accuracy).

3. The early exercise feature of American options is easily incorporated in a binomial tree, but not so easily in a simulation.

4. Pricing path-dependent options on a binomial tree requires modification of the basic method to deal with specific types of path-dependence. This sometimes requires keeping track of additional information at each node. The greater the complexity of the path-dependence, the more unwieldy the tree becomes and the more attractive simulation becomes.

5. Pricing options that depend on multiple state variables on a binomial lattice requires branching in as many dimensions as there are state variables. This generally renders binomial methods infeasible for more than 4 or 5 state variables. In contrast, simulation is not fundamentally more difficult with multiple state variables than with just one.

6. Binomial methods do not provide information about the likely error in approximated prices. Simulation provides information about errors through confidence intervals. These statistical errors can be reduced by using antithetic variates or control variates.

Table 2 gives some indication of the relative speed and precision of the binomial method and simulation. These results are for a sample of 1000 randomly generated European calls with σ ranging between 0.1 and 0.6; r between 0 and 0.1; T between 0.1 and 5; and the ratio S/K between 0.7 and 1.3. The binomial results show that the computing time required increases as the *square* of the number of time steps n. (The time units have been scaled so that 50 steps is 1 time unit.) The precision, as measured by the *root-mean-square (RMS) relative error* over all sampled options drops roughly in half with every four-fold increase in computing time. To put the errors in perspective, interpret a 0.1% error as penny-accuracy on a \$10 option. Using simulation, computing time increases in proportion with the number of replications n. Halving the error again requires roughly a four-fold increase in computing time, and this is in fact an inevitable consequence of the $\sqrt{n}$ in (44). In Table 2, for the same amount of computing time the RMS relative error using simulation is about 50 times as large as that using the binomial method. The simulation results were generated using the control variate method. The error without a control variate would be substantially larger.

These results should be taken only as a broad indication of how the precision of these methods increases with computing time and of how their time requirements compare. Different types of options and different computing platforms can lead to dramatically different run times. Even among the options sampled for Table 2, the *maximum* relative error is approximately five to ten times as large as the RMS relative error. With these qualifications in mind, the overall pattern reflected in Table 2 can still be expected to hold quite generally.

Binomial			Simulation		
n	Time	RMS (%)	n	Time	RMS (%)
50	1	0.53	35	1	23.7
100	4	0.26	100	3	14.0
200	16	0.13	500	15	6.4
400	64	0.07	2500	73	2.8
800	256	0.03	10000	290	1.4

Table 2: Root-mean-square relative error using the binomial method and simulation to price a random sample of 1000 standard European calls. In the binomial method, n is the number of time steps; in simulation it is the number of replications.

Homework Problems

1. In the example of Section 1.1, suppose the securities B_1 and B_2 are replaced with the following payoffs and prices:

$$\begin{tikzcd}[row sep=small] & 3 & & 2 \\ B_1 = 2.50 \arrow[ur] \arrow[dr] & & B_2 = 1.75 \arrow[ur] \arrow[dr] & \\ & 2 & & 3 \end{tikzcd}$$

(a) What is the risk-free interest rate?

(b) What are the risk-neutral probabilities?

(c) What must be the price of security D to preclude arbitrage?

(d) Now suppose the payoffs on B_1 and B_2 are given by the following diagram:

$$\begin{tikzcd}[row sep=small] & x_1 & & x_2 \\ B_1 \arrow[ur] \arrow[dr] & & B_2 \arrow[ur] \arrow[dr] & \\ & y_1 & & y_2 \end{tikzcd}$$

Under what condition on (x_1, y_1) and (x_2, y_2) can the price of D be determined? (Hint: What condition would make one of these securities redundant?)

2. A *cash-or-nothing* call pays \$10 if the underlying asset price is above the strike at expiration and 0 otherwise. Price the call in terms of σ, K, S_0, r, and T under the Black-Scholes assumptions. An *asset-or-nothing* call delivers the underlying asset if the asset price is above the strike at expiration and 0 otherwise. Price this call too.

3. *Minimum-variance hedging of an option.* In Section 1, we discussed option replication as a step in option pricing. The same idea can be applied to hedge an option: selling a call and

buying the replicating portfolio creates a position perfectly hedged against fluctuations in the price of the option and the underlying. This is called *delta hedging.* The drawback to delta hedging is that it requires continuous rebalancing of the hedge (the replicating portfolio) to keep the position neutralized. This can result in substantial transactions costs. In practice, one might decide to update the hedge ratio (delta) only at fixed intervals, and thus get by with only an approximate hedge between updating intervals.

If the hedging horizon is fixed, an alternative is to construct a *minimum-variance* hedge for the horizon (e.g., one or two weeks). Call the current time 0, the option expiration T, and the hedging horizon t, with $0 < t < T$. In *Introduction to Hedging*, we saw how to find the variance-minimizing hedge ratio from a set of scenarios. In the present context, scenarios are pairs $(S_t^{(i)}, C_t^{(i)})$, $i = 1, \ldots, n$, of future stock prices and option prices. We can generate scenarios in a spreadsheet using simulation. We can then use these to find the number of shares b minimizing the variance of the portfolio $\Pi_t = bS_t - C_t$ by regressing the $C_t^{(i)}$ against the $S_t^{(i)}$.

One subtle issue arises in carrying this out. The variance we want to minimize is the *real-world* variance of Π_t, not the risk-neutral world variance. So, our stock price scenarios $S_t^{(i)}$ should be generated using real-world probabilities. This means using the real rate of return μ rather than r in (42). The real-world option price $C_t^{(i)}$ associated with the stock price $S_t^{(i)}$ is then obtained by applying the Black-Scholes formula to $S_t^{(i)}$, keeping in mind that the time to expiration is $T - t$ at time t.

Figure 19 illustrates a spreadsheet calculation of the minimum-variance hedge and a comparison with delta hedging.

- Column A contains independent samples Z_i, $i = 1, \ldots, n$, from the normal distribution; $n = 1000$ is appropriate.
- These are transformed to time-t stock price scenarios in Column B. Notice that these prices use the real rate of return μ.
- For each stock price in Column B we need an option price in Column C. The notation `B-S(...)` is short for an evaluation of the Black-Scholes formula. An actual spreadsheet would probably spread this over several columns, since a different d_1 and d_2 must be calculated for each row.
- A regression of Column C against Column B yields an *in-sample* estimate of the optimal hedge ratio b.
- To evaluate the hedging effectiveness, we generate new scenarios in columns D–F.
- From the new scenarios we construct portfolio values. Column G generates portfolio values based on the minimum-variance b. Column H generates them based on a static implementation of the delta hedge ratio. In each row of Column H, Δ is the Black-Scholes delta at time 0.
- Finally, we can compare the hedging strategies by finding the standard deviations of Columns G and H.

	A	B	C	D	E	F	G	H
1	x	$= (\mu - 0.5 * \sigma^2)t$						
2	y	$= \sigma * \sqrt{t}$				regr. coeff. =	b	
3	K	= strike				B-S hedge =	Δ	
4								
5	Normals	S_t	C_t	Normals	S_t	C_t	RegHedg	DelHedg
6	Z_1	S_0*EXP(x+y*A6)	B-S(B6)	Z_1'	S_0*EXP(x+y*D6)	B-S(E6)	b*E6-F6	Δ*E6-F6
7	Z_2	S_0*EXP(x+y*A7)	B-S(B7)	Z_2'	S_0*EXP(x+y*D7)	B-S(E7)	b*E7-F7	Δ*E7-F7
8	Z_3	S_0*EXP(x+y*A8)	B-S(B8)	Z_3'	S_0*EXP(x+y*D8)	B-S(E8)	b*E8-F8	Δ*E8-F8
$\vdots$				$\vdots$				$\vdots$

Figure 19: Spreadsheet for hedging simulation

Implement the approach described above. Suppose $\sigma = 0.30$, $r = 0.05$, $\mu = 0.12$, $K = 50$, $S_0 = 50$, $T = 0.24$, and $t = 0.04$. Report your mimimum-variance hedge ratio and Δ. Compare the standard deviations of portfolio values using the two approaches. Also, compare the estimated hedging effectiveness under b with that predicted by the R^2 in your regression.

4. Do Problem 3 using a binomial tree instead of simulation. Notice that the scenarios extracted from a binomial tree have associated probabilities. So, you cannot use regression to get the minimum-variance hedge ratio.

5. Consider the pricing of a down-and-out call by simulation. Suppose the "true" price is based on continuous monitoring of the barrier, but the simulation generates finitely many steps along each path. Is the price estimate from simulation biased low, biased high, or unbiased. Explain. Answer the same question for a down-and-out put, a down-and-in call, and an up-and-in call.

6. Price a lookback put using a four-step binomial tree. Use the same parameters used in Figure 8. (From the figure you can get the stock price at every node and then price the option.)

7. Price an American put on the spread between two assets using a 10-step spreadsheet implementation of the two-dimensional binomial tree. Use the following parameters: $(S^1, S^2) = (100, 120)$, $K = 20$, $(\sigma_1, \sigma_2) = (0.30, 0.20)$, $\rho = 0.50$, $T = 0.50$, $r = 0.05$. The intrinsic value is $\max\{0, K - (S^2 - S^1)\}$. Compare your answer with one obtained from `binom_2d.m`.

8. XYZ Inc. wants to implement an automatic stock buy-back if its stock falls relative to the A&B Index. To do this, it plans to sell out-of-the-money American puts on its stock, to be knocked out if the index falls below a barrier. Thus, if the stock falls but the index doesn't, the puts will be exercised, resulting in stock purchases by the corporation.[16] Currently, the stock is at 39 and the index is at 94. The strike is 35 and the barrier is at 85. Assume a risk-free rate of $r = 0.05$, volatilities of $\sigma_{\text{XYZ}} = 0.30$, $\sigma_{\text{A\&B}} = 0.25$, and a correlation of $\rho = 0.40$. For the index, assume a continuous dividend yield of 3%; the stock pays no dividends. The option expires in six months.

(a) Price the option in a 2-dimensional binomial tree using 12 steps in a spreadsheet or a larger number using MATLAB. To incorporate the dividend yield, subtract 0.03 in (38) from the ν_i for the index. (This option is easy to price by modifying `binom_2d.m`. If S_1 is the index, define a variable `alive_check = (smat1 > barrier)` and use it to knock-out the option the way it is used in `doc_bin.m`.)

[16]This strategy was described in *Barron's*, February 6, 1995.

(b) What are good values for the number of time steps in this tree?

9. Consider a down-and-out call with $S_0 = 48$, $K = 50$, $H = 40$, $\sigma = 0.30$, $r = 0.05$, $T = 0.25$. Suppose knock-outs can occur only at daily closings.

(a) Price the option by simulation (e.g., by modifying `dic_sim.m`). Assuming 252 trading days per year, there are 63 trading days in the life of the option. Since only daily closing prices matter, your simulation time step should be one day. Report a confidence interval for the price.

(b) Repeat (a) using the underlying as a control variate.

(c) What price does the formula give based on continuous monitoring of the barrier? (You can use `doc.m`.)

10. The payoff of an *Asian* option depends on the average price of the underlying over some period, rather than just the terminal price. Consider an Asian option with payoff

$$\max\{0, \frac{1}{20}\sum_{i=1}^{20} S_{t_i} - K\},$$

where $t_1, \ldots, t_{20}$ are the last 20 closing prices in the life of the option. Set $S_0 = 50$, $K = 50$, $\sigma = 0.30$, $r = 0.05$, $T = 0.25$. Assume 252 trading days per year. Price the option by simulation, using either MATLAB or a spreadsheet.

11. Price the following options. For each one, provide an interval that you believe contains the true price. (If you use a binomial method, this won't be a confidence interval in the usual sense.) You will be allocated points based on two considerations:

- Whether or not your interval contains the true value (significant penalty if not).
- The square of the width of your interval (narrower intervals get more points).

In some cases, there is more than one way to get a price and an interval. Clearly indicate what method you used and include information about the number of time steps, number of replications, etc., wherever appropriate. You may use more than one method if you like.

(a) A European put option on the minimum of two stocks, with $S_1 = S_2 = K = 100$, $r = 0.05$, $\sigma_1 = \sigma_2 = 0.25$, $\rho_{12} = 0.5$, and $T = 1$.

(b) Same as (a) but American.

(c) Same as (a) but with three stocks, and $S_3 = 100$, $\sigma_3 = 0.25$, and $\rho_{13} = \rho_{23} = 0.5$.

(d) A European call spread option with the parameters in (a) except $K = 5$.

(e) An American up-and-out put option with $S = 47$, $K = 45$, $B = 55$, $\sigma = 0.25$, $r = 0.05$, $T = 0.10$. Assume continuous barrier monitoring.

12. A standard American put should be exercised early if the price of the underlying drops sufficiently low. How low is necessary depends on the option parameters. For each time t in the life of the option there is a critical value S_t^*, such that early exercise is optimal if the price of the underlying is below S_t^* and not optimal if it is above S_t^*. The graph of S_t^* is called the *exercise boundary.* Use a binomial tree with at least 20 steps to determine the exercise boundary of an American put with $S_0 = 50$, $K = 45$, $\sigma = 0.30$, $r = 0.05$, and $T = 0.20$. Graph the exercise boundary.

13. The *marginal* probability of an upmove for S^1 in a two-dimensional tree for (S^1, S^2), is the probability that S^1 moves up in a single time step, regardless of whether S^2 moves up or down.

(a) Find the marginal probability of an upmove for S^1 and for S^2 in a two-dimensional tree, based on $r = 0.05$, $\sigma_1 = 0.30$, $\sigma_2 = 0.20$, $\rho = 0.50$, $T = 1$, and $n = 50$ time steps.

(b) Now suppose we were to construct separate 50-step trees for S^1 and for S^2. What would be the upmove probabilities in each tree? To what extent is the two-dimensional tree consistent with the separate one-dimensional trees?

14. A *shout* option is a type of path-dependent option that has features in common with both American and European options. Consider a shout call struck at K. By shouting at time t, the holder locks in a terminal payoff equal to at least the time-t intrinsic value. Thus, the terminal payoff on a shout call is $\max\{0, S_T - K, H - K\}$, where the *shout level* H is the price of the underlying at the time of the shout. Pricing a shout option entails determining the optimal time to stop, just as pricing an American option entails determining the optimal time to exercise.

Consider the pricing of a shout call in a binomial tree. At each node, the value of the option is

$$\max(\text{value if shout now, value if shout later}). \tag{57}$$

"Value if shout later" is simply the present value of the prices at the two successor nodes. "Value if shout now" will be found through a formula similar to the Black-Scholes formula.

(a) For fixed H, construct a portfolio of European puts and calls that replicates the payoff $\max\{0, S_T - K, H - K\}$. Each of the options in the replicating portfolio can be priced using the Black-Scholes formula. Use these prices to give a formula for "value if shout now" when the shout time and shout level are known.

(b) Construct a 10-step binomial tree to price a shout call with the following parameters: $\sigma = 0.30$, $r = 0.05$, $K = 50$, $S_0 = 47$ and $T = 0.5$. Your tree should evaluate (57) at each node.

Appendix A: Miscellaneous Option Pricing Formulas

In this appendix, we list formulas for some of the options discussed in these notes. We already encountered formula for standard European calls and puts. Define

$$\mathrm{BSC}(\sigma, r, K, S_0, T) = S_0 N(d_1) - Ke^{-rT} N(d_2),$$

with d_1, d_2 as in (20) and (21), to be the Black-Scholes call price. Another way to write the put price is

$$P = \mathrm{BSC}(\sigma, r, K, S_0, T) - S_0 + e^{-rT} K.$$

This is an instance of a general relation called *put-call parity.*

The price of a down-and-in call with barrier H is given by

$$\mathrm{DIC} = \left(\frac{H}{S_0}\right)^{\frac{2r}{\sigma^2}-1} \cdot \mathrm{BSC}(\sigma, r, K, H^2/S_0, T).$$

In other words, we can price the down-and-in call with the Black-Scholes formula by using H^2/S_0 for the initial asset price and then multiplying by an additional factor. This factor is related to the probability of reaching the barrier. Since the down-and-in and a down-and-out add up to an ordinary call, we also have

$$\mathrm{DOC} = \mathrm{BSC}(\sigma, r, K, S_0, T) - \mathrm{DIC}.$$

The price of a lookback call is given by

$$S_0 N(a_1) - S_0 \frac{\sigma^2}{2r} N(-a_1) - S_{\min}\left[N(a_2) - \frac{\sigma^2}{2r} e^{y_1} N(-a_3)\right],$$

where

$$\begin{aligned}
a_1 &= \frac{\log(S_0/S_{\min}) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \\
a_2 &= a_1 - \sigma\sqrt{T} \\
a_3 &= \frac{\log(S_0/S_{\min}) + (-r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \\
y_1 &= -\frac{2(r - \frac{1}{2}\sigma^2)\log(S_0/S_{\min})}{\sigma^2},
\end{aligned}$$

and $S_{\min}$ is the minimum price observed to date in the life of the option. The price of a lookback put is

$$S_{\max} e^{-rT}\left[N(b_1) - \frac{\sigma^2}{2r} e^{y_2} N(-b_3)\right] - S_0 N(b_2) + S_0 \frac{\sigma^2}{2r} N(-b_2),$$

where

$$b_1 = \frac{\log(S_{\max}/S_0) + (-r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$\begin{aligned} b_2 &= b_1 - \sigma\sqrt{T} \\ b_3 &= \frac{\log(S_{\max}/S_0) + (r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \\ y_2 &= \frac{2(r - \frac{1}{2}\sigma^2)\log(S_{\max}/S_0)}{\sigma^2}, \end{aligned}$$

and $S_{\max}$ is the maximum price observed to date in the life of the option.

Why discuss numerical techniques like the binomial method and simulation for options that can be priced in using formulas? There are several reasons:

- There are no formulas for American versions of the options above, except for cases where the American option degenerates to the European counterpart because early exercise is known to be suboptimal. In contrast, the binomial method handles American options just as easily as European options.
- The formulas for barrier and lookback options assume continuous price path. In practice, these types of contracts often specify discrete price fixings, requiring the use of a numerical method.
- The formulas assume the underlying pays no dividends. Typically, formulas can be extended to account for continuous *dividend yields*, and these are appropriate when the underlying is an exchange rate or a stock index. Pricing options on stocks with discrete dividends requires a numerical method. Discrete dividends are easily modeled in simulation through price drops at the ex-dividend dates; the required modification of the binomial is similar but slightly more involved.

Appendix B: MATLAB Listings

The MATLAB program `op_main.m` illustrates the calls to various option-pricing functions. In Figure 20, it is set up to call `binom.m` (which appears in Figure 21), `bs_sim.m` (which appears in Figure 32), and `bs.m` (which appears in Figure 28). The parameters are defined at the top; these can be edited to price different options. Not all parameters are required for all functions. For example, `rseed` and `nsim` are for the simulation programs only. Notice how `bs_sim.m` is called to return four values. Lines [14]–[22] echo the parameters to the screen before displaying the answer. This makes it easier to interpret the results.

```
 [1]  % op_main.m
 [2]  % program to test various binomial programs and formulas
 [3]
 [4]  % input parameters
 [5]  sig        = 0.30;        % annual volatility
 [6]  r          = 0.05;        % riskless rate (continuous compounding)
 [7]  strike     = 50;          % strike price
 [8]  s          = 47;          % current stock price
 [9]  t          = 0.5;         % option maturity (in years)
[10]  n          = 50;          % number of binomial time steps
[11]  rseed      = 4347;        % random number seed
[12]  nsim       = 1000;        % number of simulation trials
[13]
[14]  disp('call option results:');
[15]  disp1('annual volatility             ', sig);
[16]  disp1('riskless interest rate        ', r);
[17]  disp1('strike price                  ', strike);
[18]  disp1('current stock price           ', s);
[19]  disp1('option expiration (in years)  ', t); disp(' ');
[20]  disp1('number of binomial steps      ', n);
[21]  disp1('random number seed            ', rseed);
[22]  disp1('number of simulation trials   ', nsim); disp(' ');
[23]
[24]  bin_val  = binom(sig, r, strike, s, t, n);
[25]  disp1('binomial call option value       ', bin_val);
[26]
[27]  [call_c, se_c, call, se] = bs_sim(sig, r, strike, s, t, n, rseed);
[28]  disp1('simulated call price             ', call, '+/-', se);
[29]  disp1('simulated call price with control ', call_c, '+/-', se_c);
[30]
[31]  call_val = bs(sig, r, strike, s, t);
[32]  disp1('black-scholes call option value   ', call_val); disp(' ');
[33]
[34]  % end of op_main.m
```

Figure 20: MATLAB program op_main.m

```
function [opt_val] = binom(sig, r, strike, s, t, n);
% BINOM  binomial call option pricing routine
% input parameters: sig, r, strike, s, t, n
% (n is the number of time steps in the binomial tree)

% parameters for binomial tree
dt = t/n;
u  = exp(sig*sqrt(dt));
d  = 1/u;
p  = (exp(r*dt)-d) / (u-d);
q  = 1-p;

% create s_vec with m = 2*n+1 elements:
% s_vec = [ u^n s; u^(n-1) s; ... s; ds; ...; d^n s ]
s_vec   = cumprod([u^n*s; d(ones(2*n,1),:)]);

% create a sparse present value matrix, pv
m       = 2*n + 1;
pv      = spdiags([q(ones(m,1),:) p(ones(m,1),:)], [1 -1], m, m);
pv(1,1) = p;
pv(m,m) = q;
pv      = exp(-r*dt)*pv;

% create op_vec, a vector of terminal option values
op_vec  = max(s_vec-strike, 0);

% values call by working backward from time n-1 to time 0
for i = n-1 : -1 : 0;
    op_vec = pv*op_vec;
    end;

opt_val = op_vec(n+1);

% end of binom.m
```

Figure 21: MATLAB program binom.m

Figure 22 shows the MATLAB program `bin_plot.m` for plotting the binomial option price as a function of the number of time steps. This program can be used to create the binomial convergence plot in Figure 5 or can be easliy modified to create the binomial barrier option convergence plot in Figure 9.

Most of `cap_bin.m`, displayed in Figure 23 is identical to `binom.m`. Line 28 implements the capped intrinsic value formula. Line 18 defines a vector of 1s and 0s: the i^{th} entry of `capchk` is 1 if the i^{th} layer of nodes (counting from the top) is above $K + L$. These are the node layers that trigger automatic exercise. Triggering is implemented in line 32: above $K + L$, `op_vec` gets the value `cap`, and below it gets the present value at the successor nodes.

Figure 24 shows a MATLAB program `doc_bin.m` for pricing a knock-out option on a binomial tree. It is similar to `cap_bin.m`. The variable `alive_check`, defined in line 25, is a column of 0s and 1s; the 1s appear in rows corresponding to nodes above the barrier and the 0s in rows corresponding to nodes below the barrier. This variable is used to set the terminal payoffs in line 28 and to implement a step back through the tree in line 32.

```
function bin_plot(sig, r, strike, s, t, n1, n2);
% BIN_PLOT  function to plot binomial call option
% values versus number of binomial steps.
% The number of steps runs from n1 to n2.
% input parameters: sig, r, strike, s, t, n1, n2

op_vec = [];
for i = n1 : 1 : n2;
    call_val = binom(sig, r, strike, s, t, i);
    disp1('i = ', i, ' call = ', call_val);
    op_vec = [op_vec call_val];
    end;

call_val = bs(sig, r, strike, s, t);
disp1('exact value = ', call_val);

plot([n1 : 1 : n2], op_vec);
title('Option Value vs. Number of Binomial Steps');
xlabel('Number of Binomial Steps'); ylabel('Option Price');

% end of bin_plot.m
```

Figure 22: MATLAB program bin_plot.m

```
function [opt_val] = cap_bin(sig, r, strike, s, t, cap, n);
% CAP_BIN  binomial capped-call option pricing routine
% input parameters: sig, r, strike, s, t, cap, n
% (cap is the maximum payout; n is the number of time steps)

% parameters for binomial tree
dt = t/n;
u  = exp(sig*sqrt(dt));
d  = 1/u;
p  = (exp(r*dt)-d) / (u-d);
q  = 1-p;

% create s_vec with m = 2*n+1 elements:
% s_vec = [ u^n s; u^(n-1) s; ... s; ds; ...; d^n s ]
s_vec   = cumprod([u^n*s; d(ones(2*n,1),:)]);

% create vector to indicate automatic exercise
cap_chk  = (s_vec >= strike+cap);

% create a present value matrix, pv
m       = 2*n + 1;
pv      = spdiags([q(ones(m,1),:) p(ones(m,1),:)], [1 -1], m, m);
pv(1,1) = p;
pv(m,m) = q;
pv      = exp(-r*dt)*pv;

% create op_vec, a vector of terminal option values
op_vec  = min(cap, max(s_vec-strike, 0));

% value option by working backward from time n-1 to time 0
for i = n-1 : -1 : 0;
    op_vec = (cap_chk)*cap + (1-cap_chk).*(pv*op_vec);
    end;

opt_val = op_vec(n+1);

% end of cap_bin.m
```

Figure 23: MATLAB program cap_bin.m

```
function [opt_val] = doc_bin(sig, r, strike, s, t, barrier, n);
% DOC_BIN   down-and-out call pricing using the binomial method
% input parameters: sig, r, strike, s, t, barrier, n
% (barrier is the barrier level; n is the number of time steps)

% parameters for binomial tree
dt = t/n;
u  = exp(sig*sqrt(dt));
d  = 1/u;
p  = (exp(r*dt)-d) / (u-d);
q  = 1-p;

% create s_vec with m = 2*n+1 elements:
% s_vec = [ u^n s; u^(n-1) s; ... s; ds; ...; d^n s ]
s_vec   = cumprod([u^n*s; d(ones(2*n,1),:)]);

% create a present value matrix, pv
m       = 2*n + 1;
pv      = spdiags([q(ones(m,1),:) p(ones(m,1),:)], [1 -1], m, m);
pv(1,1) = p;
pv(m,m) = q;
pv      = exp(-r*dt)*pv;

% create alive_check indicator: 1 if s > barrier, 0 otherwise
alive_check = (s_vec > barrier);

% create op_mat, a matrix of terminal option values
op_vec = max(s_vec-strike,0).*alive_check;

% values call by working backward from time n-1 to time 0
for i = n-1 : -1 : 0;
    op_vec = (pv*op_vec).*alive_check;
    end;

opt_val = op_vec(n+1);

% end of doc_bin.m
```

Figure 24: MATLAB program doc_bin.m

Figure 25 shows a MATLAB program `dic_bin.m` for pricing a knock-in option on a binomial tree. The matrix `op_mat` has two columns, the first recording B^+ values, the second recording B^- values. Each row of the matrix corresponds to a layer of nodes. Lines 30–31 generate two columns of terminal payoffs with the "knocked-in" and "not-knocked-in" interpretation given to B^+ and B^- above. In particular, the first column (the B^+ value) is the usual terminal payoff, and the second column (the B^- value) can be nonzero only for terminal stock levels below the barrier. The variable `in_check`, defined in line 25, is 1 below the barrier and 0 above. Lines 35–36 work backwards through the tree. At each step, both columns are updated according to the usual present value calculation; this corresponds to Figure 10(a). For nodes below the barrier, we must then set B^- equal to B^+ to be consistent with Figure 10(b). This is done in line 36.

```
 [1]  function [opt_val] = dic_bin(sig, r, strike, s, t, barrier, n);
 [2]  % DIC_BIN   down-and-in call pricing using the binomial method
 [3]  % input parameters: sig, r, strike, s, t, barrier, n
 [4]  % (barrier is the barrier level and n is the number of time steps)
 [5]
 [6]  % parameters for binomial tree
 [7]  dt = t/n;
 [8]  u  = exp(sig*sqrt(dt));
 [9]  d  = 1/u;
[10]  p  = (exp(r*dt)-d) / (u-d);
[11]  q  = 1-p;
[12]
[13]  % create s_vec with m = 2*n+1 elements:
[14]  % s_vec = [ u^n s; u^(n-1) s; ... s; ds; ...; d^n s ]
[15]  s_vec   = cumprod([u^n*s; d(ones(2*n,1),:)]);
[16]
[17]  % create a present value matrix, pv
[18]  m       = 2*n + 1;
[19]  pv      = spdiags([q(ones(m,1),:) p(ones(m,1),:)], [1 -1], m, m);
[20]  pv(1,1) = p;
[21]  pv(m,m) = q;
[22]  pv      = exp(-r*dt)*pv;
[23]
[24]  % create in_check indicator: 1 if s <= barrier, 0 otherwise
[25]  in_check = (s_vec<=barrier);
[26]
[27]  % create op_mat, a matrix of terminal option values
[28]  % first column:  payoff if barrier previously crossed
[29]  % second column: payoff if barrier not previously crossed
[30]  op_mat              = [max(s_vec-strike,0) zeros(m,1)];
[31]  op_mat(in_check, 2) = op_mat(in_check, 1);
[32]
[33]  % values call by working backward from time n-1 to time 0
[34]  for i = n-1 : -1 : 0;
[35]      op_mat = pv*op_mat;
[36]      op_mat(in_check, 2) = op_mat(in_check, 1);
[37]      end;
[38]
[39]  opt_val = op_mat(n+1, 2);
[40]
[41]  % end of dic_bin.m
```

Figure 25: MATLAB program dic_bin.m

Figure 26 shows a MATLAB implementation of the two-dimensional binomial method. Most of

```
function [opt_val] = binom_2d(sig1, sig2, rho, r, strike, s1, s2, t, n);
% BINOM_2D  two-asset max call option pricing routine
% input parameters: sig1, sig2, rho, r, strike, s1, s2, t, n

% parameters for tree
dt   = t/n;
u1   = exp(sig1*sqrt(dt));
u2   = exp(sig2*sqrt(dt));
d1   = 1/u1;
d2   = 1/u2;
mu1  = r-0.5*sig1*sig1;
mu2  = r-0.5*sig2*sig2;
p_uu = 0.25*(1+rho+sqrt(dt)*((mu1/sig1)+(mu2/sig2)));
p_ud = 0.25*(1-rho+sqrt(dt)*((mu1/sig1)-(mu2/sig2)));
p_dd = 0.25*(1+rho-sqrt(dt)*((mu1/sig1)+(mu2/sig2)));
p_du = 0.25*(1-rho+sqrt(dt)*(-(mu1/sig1)+(mu2/sig2)));

% create s_mat1,s_mat2 matrices with (m = 2*n+1)^2 elements:
% s_mat1(2) repeats rows(columns) of terminal prices for stock 1(2)
svec1 = cumprod([u1^n*s1 ; d1(ones(2*n,1),:)])';
svec2 = cumprod([u2^n*s2 ; d2(ones(2*n,1),:)]);
m     = 2*n+1;
smat1 = svec1(ones(m,1),:);
smat2 = svec2(:,ones(1,m));

% create weights for present value calculation
pvec   =  exp(-r*dt)*[p_du p_dd p_ud p_uu];

% create op_mat, a matrix of terminal option values
op_mat = max(max(smat2,smat1)-strike,0);

% values call by working backward from time n-1 to time 0
for i = n-1 : -1 : 0;
    op_mat = pv2(op_mat,pvec);
    end;

opt_val = op_mat(n+1,n+1);

% end of binom_2d.m
```

Figure 26: MATLAB program binom_2d.m

```
function V = pv2(U,pvec)
% PV2   Four-point 2-dimensional present value operator
%       V is a matrix the same size as U with each element
%       equal to the average of its four diagonal neighbors
%       with weights pvec (in the order NE, SE, SW, NW).
%       Edges get no special treatment.

[p,q] = size(U);
e     = [2:q q];
w     = [1 1:q-1];
n     = [1 1:p-1]';
s     = [2:p p]';

V = pvec(1)*U(n,e) + pvec(2)*U(s,e) + pvec(3)*U(s,w) + pvec(4)*U(n,w);

% end of pv2.m
```

Figure 27: MATLAB program pv2.m

the program is similar to previous ones. Only two lines require comment. Line 30 determines the type of option; in the figure, it is set to price a call max option. It should be clear how to modify this line to get other payoffs and thus other types of options. Line 34 implements the present value calculation. This is no longer simply a matrix multiplication, because we now have an array of nodes at each step, rather than just a vector of nodes. The function `pv2.m` is displayed in Figure 27. Line 34 also determines whether the option is American or European. In the figure, the program is set to price a European option. To price an American option, replace `pv2(op_mat,pvec)` with

`max(pv2(op_mat,pvec),` immediate exercise value`);`

where, in the case of a call max option, the immediate exercise value is given by the expression `max(max(smat2,smat1)-strike,0);`. The immediate exercise value is always whatever payoff appears in line 30.

```
function [opt_val] = bs(sig, r, strike, s, t);
% BS  black-scholes call option pricing formula
% input parameters: sig, r, strike, s, t

d1      = (log(s/strike) + (r + 0.5*sig*sig)*t) / (sig*sqrt(t));
nd1     = nmlz2p(d1);
d2      = d1 - sig*sqrt(t);
nd2     = nmlz2p(d2);

opt_val = s*nd1 - strike*exp(-r*t)*nd2;

% end of bs.m
```

Figure 28: MATLAB program bs.m

```
function [opt_val] = doc(sig, r, strike, s, t, barrier);
% DOC  down-and-out call option pricing formula
% input parameters: sig, r, strike, s, t, barrier

opt_val = bs(sig, r, strike, s, t) ...
            - dic(sig, r, strike, s, t, barrier);

% end of dic.m
```

Figure 29: MATLAB program doc.m

```
function [opt_val] = dic(sig, r, strike, s, t, barrier);
% DIC  down-and-in call option pricing formula
% input parameters: sig, r, strike, s, t, barrier

opt_val = ((barrier/s)^((2*r/sig^2)-1))* ...
             bs(sig, r, strike, barrier^2/s, t);

% end of dic.m
```

Figure 30: MATLAB program dic.m

```
function [opt_val] = lookback(sig, r, s, t, s_min);
% LOOKBACK  lookback call option pricing formula
% input parameters: sig, r, s, t, s_min
% (s_min is the current minimum stock value)

a1      = (log(s/s_min) + (r + 0.5*sig*sig)*t) / (sig*sqrt(t));
a2      = a1 - sig*sqrt(t);
a3      = (log(s/s_min) + (-r + 0.5*sig*sig)*t) / (sig*sqrt(t));
y1      = -2*(r - 0.5*sig*sig)*log(s/s_min)/(sig*sig);

na1     = nmlz2p(a1);
nma1    = nmlz2p(-a1);
na2     = nmlz2p(a2);
nma3    = nmlz2p(-a3);
s22r    = sig*sig/(2*r);

opt_val = s*na1 - s*s22r*nma1 - s_min*exp(-r*t)*(na2 - s22r*exp(y1)*nma3);

% end of lookback.m
```

Figure 31: MATLAB program lookback.m

```
function [call_c, se_c, call, se] = bs_sim(sig, r, strike, s, t, n, rseed);
% BS_SIM  simulation call option pricing routine
% input parameters: sig, r, strike, s, t, n, rseed
% (n is the number of simulation trials, rseed is the random number seed)
% outputs: call_c, se_c, call, se ( _c indicates control variate result)

randn('seed', rseed);
s_term    = s*exp((r-0.5*sig*sig)*t + sig*sqrt(t)*randn(n,1));
call_vec  = exp(-r*t)*max(s_term-strike,0);
call      = mean(call_vec);
se        = std(call_vec)/sqrt(n);

% adjust using initial stock price as control variate
[call_c, se_c] = controlv(call_vec, exp(-r*t)*s_term, s);

% end of bs_sim.m
```

Figure 32: MATLAB program bs_sim.m

```
function [call, se] = bs_ant(sig, r, strike, s, t, n, rseed);
% BS_ANT  simulation call option pricing routine using antithetics
% input parameters: sig, r, strike, s, t, n, rseed
% (n is the number of simulation trials, rseed is the random number seed)

randn('seed', rseed);
n2        = n/2;
z_vec     = randn(n2,1);
z         = [z_vec -z_vec];
s_term    = s*exp((r-0.5*sig*sig)*t + sig*sqrt(t)*z);
call_vec  = exp(-r*t)*0.5*(max(s_term(:,1)-strike,0) ...
                            + max(s_term(:,2)-strike,0));
call      = mean(call_vec);
se        = std(call_vec)/sqrt(n2);

% end of bs_ant.m
```

Figure 33: MATLAB program bs_ant.m

```
function [call_c, se_c, call, se] = dic_sim(sig, r, strike, s, t, ...
                 barrier, n, nt, rseed);
% DIC_SIM  simulation down-and-in call option pricing routine
% input parameters: sig, r, strike, s, t, barrier, n, nt, rseed
% (n is the number of simulation trials, nt is the number of time steps)
% outputs: call_c, se_c, call, se ( _c indicates control variate result)

dt = t/nt;
randn('seed', rseed);

% s_mat(i,j) = stock price at time i, simulation trial j
% s_mat has nt+1 rows (row 1 is the current time) and n columns
s_mat  = exp((r-0.5*sig*sig)*dt + sig*sqrt(dt)*randn(nt,n));
s_mat  = cumprod([s(:,ones(1,n)); s_mat]);
s_term = s_mat(nt+1,:);

% evaluate discounted knock-in payoff, mean and standard error
call_vec = exp(-r*t)*(min(s_mat) <= barrier).*(max(s_term-strike,0));
call     = mean(call_vec);
se       = std(call_vec)/sqrt(n);

% adjust using initial stock price as control variate
[call_c, se_c] = controlv(call_vec, exp(-r*t)*s_term, s);

% end of dic_sim.m
```

Figure 34: MATLAB program dic_sim.m

```
function [call_c, se_c, call, se] = lb_sim(sig, r, s, t, n, nt, rseed);
% LB_SIM   lookback simulation pricing routine
% input parameters: sig, r, s, t, n, nt, rseed
% (n is the number of simulation trials, nt is the number of time steps)
% outputs: call_c, se_c, call, se ( _c indicates control variate result)

dt = t/nt;
randn('seed', rseed);

% s_mat(i,j) = stock price at time i, simulation trial j
% s_mat has nt+1 rows (row 1 is the current time) and n columns
s_mat  = exp((r-0.5*sig*sig)*dt + sig*sqrt(dt)*randn(nt,n));
s_mat  = cumprod([s(:,ones(1,n)); s_mat]);
s_term = s_mat(nt+1,:);

% evaluated discounted lookback payoff, mean and standard error
call_vec = exp(-r*t)*(s_term-min(s_mat));
call     = mean(call_vec);
se       = std(call_vec)/sqrt(n);

% adjust using initial stock price as control variate
[call_c, se_c] = controlv(call_vec, exp(-r*t)*s_term, s);

% end of lb_sim.m
```

Figure 35: MATLAB program lb_sim.m

```
function [call, se] = multisim(sig, corr, r, strike, s, t, n, rseed);
% MULTISIM   multi-asset max-call option pricing simulation
% input parameters: sig, corr, r, strike, s, t, n, rseed
% (sig and s are vectors; corr is a matrix)
% outputs: call, se

% factor correlation matrix
cov_mat  = diag(sig)*corr*diag(sig);
chol_mat = chol(cov_mat);

% simulate terminal asset values and option payoffs
% s_term is k x nsim matrix of terminal values, where k = no. assets
randn('seed', rseed);
k        = length(sig);
x_mat    = chol_mat'*randn(k,n);
s_term   = kron(ones(1,n), (log(s)+(r-0.5*sig.*sig)*t)');
s_term   = exp(s_term + sqrt(t)*x_mat);
call_vec = exp(-r*t)*max(max(s_term)-strike,0);
call     = mean(call_vec);
se       = std(call_vec)/sqrt(n);

% end of multisim.m
```

Figure 36: MATLAB program multisim.m

```
function [new_est, se] = controlv(sim_dat, ctl_dat, ctl_true);
% CONTROLV   function to use control variate technique to improve
% simulation estimates.  sim_dat and ctl_dat must be vectors of the
% same length.  ctl_true is the true expected value of the control.

c        = polyfit(ctl_dat, sim_dat, 1);
new_est  = polyval(c, ctl_true);
yhat     = polyval(c, ctl_dat);
err_vec  = sim_dat - yhat;
se       = std(err_vec) / sqrt(length(sim_dat));

% end of controlv.m
```

Figure 37: MATLAB program controlv.m

```
function disp1(s1, s2, s3, s4, s5, s6, s7, s8, s9, s10);
% DISP1
% displays up to ten arguments on the screen
% e.g., x = 2.3;
%       disp1('x =', x, '; 2*x =', 2*x);
% will print 'x = 2.3 ; 2*x = 4.6' on the screen

s = [];
for i = 1 : nargin;
    x = eval(['s',int2str(i)]);

    % num2str.m code follows with '%.6g' replacing '%.4g'
    if isstr(x)
            t = [x ' '];
    else
            t = sprintf('%.6g ',x);
    end

    s = [s t];
    end;

disp(s);

% end of disp1.m
```

Figure 38: MATLAB program disp1.m