1 Modeling risky assets and pricing options: The Binomial Lattice Model (BLM)

In these notes and what follows, we will present a probabilistic model for stock (risky asset) prices known as the *binomial lattice model*, and then learn how to price options (derivatives) of the asset. Along the way we will derive the famous Black-Scholes-Merton option pricing formula in discrete time. We will also learn how to simulate this model on your computer, and how to use such simulation to numerically estimate the price of options; a method known as *Monte Carlo Simulation*.

1.1 BLM

Consider a stock that has a price per share each day; S_n denotes the price at the end of the $n^{th} \geq 0$ day with S_0 the initial price. Suppose that given we know the price on a given day S_n, the price the next day S_{n+1} is determined by an independent Bernoulli (p) rv B_{n+1} as follows: If $B_{n+1} = 1$, then $S_{n+1} = uS_n$, whereas if $B_{n+1} = 0$, then $S_{n+1} = dS_n$. The parameters $0 < d < u$ are called 'UP' and 'DOWN' factors, where we view UP as a success and DOWN as a failure.

Thus, given our initial price S_0, the price S_n at any fixed time $n \geq 1$ in the future, is of the form $u^k d^{n-k} S_0$, for some k with $0 \leq k \leq n$, which has the binomial (n, p) probability of occurring

$$P(S_n = u^k d^{n-k} S_0) = p(k) = \binom{n}{k} p^k (1-p)^{n-k},$$

referring to the stock having gone up k times and down $n-k$ times; k successes out of n trials.

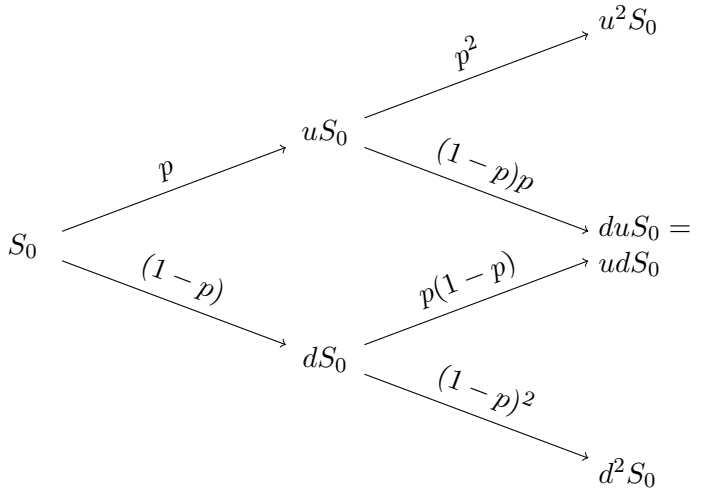
By introducing independent up/down rvs $Y_1, Y_2, \ldots$ defined by $Y_n = u$ if $B_n = 1$ and $Y_n = d$ if $B_n = 0,\ n \geq 1$, then $P(Y_n = u) = p = P(B_n = 1)$ and $P(Y_n = d) = q = 1 - p = P(B_n = 0)$ and we can define our sequence of stock prices recursively over time by

$$S_{n+1} = S_n Y_{n+1},\ n \geq 0. \tag{1}$$

We can expand this to obtain the price at any time n as

$$S_n = S_0 Y_1 \times Y_2 \times \cdots \times Y_n. \tag{2}$$

Below we draw this as a branching out tree with S_0 as the base and out to time $n = 2$:



We see that, as should be, $P(S_2 = u^2S_0) = p^2 = p(2)$, $P(S_2 = udS_0) = 2p(1-p) = p(1)$, $P(S_2 = d^2S_0) = (1-p)^2 = p(0)$: the binomial $(2, p)$ probabilities.

1.2 Simulating the binomial lattice model.

If we want to simulate a Y rv such that $P(Y = u) = p$, $P(Y = d) = 1 - p$, we can do so with the following simple algorithm:

(i) Enter p, u, d

(ii) Generate a U

(iii) Set

$$Y = \begin{cases} u & \text{if } U \leq p \ , \\ d & \text{if } U > p. \end{cases}$$

Similar to how we simulate a Bernoulli (p) rv, this works because $P(Y = u) = P(U \leq p) = F(p) = p$, and $P(Y = d) = P(U > p) = 1 - F(p) = 1 - p$ exactly as is required. Indeed Y has the distribution as desired. Repeating this algorithm's steps (ii)-(iii) using independent copies of U then yields independent copies of Y, say $Y_1, Y_2, \ldots$ from which we can simulate the BLM up to any time n via first generating n iid copies of Y, $Y_1, Y_2, \ldots, Y_n$, and then setting

$$S_n = S_0 Y_1 \times \cdots \times Y_n.$$

Here is the algorithm:

(i) Enter p, u, d, and S_0, and n.

(ii) Generate $U_1, U_2, \ldots, U_n$

(iii) For $i = 1, \ldots n$, set

$$Y_i = \begin{cases} u & \text{if } U_i \leq p \ , \\ d & \text{if } U_i > p. \end{cases}$$

(iv) For $i = 1, \ldots n$, set $S_i = S_0 Y_1 \times \cdots Y_i$.

This would yield the values $S_0, S_1, \ldots, S_n$.

One can also do this recursively using the recursion $S_{i+1} = S_i Y_{i+1},\ 0 \leq i \leq n-1$. The advantage (computationally) is that this allows you to generate the U_i, hence the Y_i, sequentially without having to save all n values at once. Here is the algorithm:

(i) Enter p, u, d, and S_0, and n. Set $i = 1$

(ii) Generate U

(iii) Set

$$Y = \begin{cases} u & \text{if } U \leq p\ , \\ d & \text{if } U > p. \end{cases}$$

(iv) Set $S_i = S_{i-1}Y$. If $i < n$, then reset $i = i + 1$ and go back to (ii); otherwise stop.

Note that equivalently, one can generate a Y rv by using a generated Bernoulli (p) rv B (which itself was generated by using a U):

(i) Enter p, u, d

(ii) Generate a Bernoulli (p) denoted by B

(iii) Set

$$Y = \begin{cases} u & \text{if } B = 1\ , \\ d & \text{if } B = 0. \end{cases}$$

Python code for the above algorithms

Let us first recall our basic Bernoulli (p) algorithm (from the previous Lecture Notes on Probability):

```
import random
#Bernoulli (p) algorithm
def Bernoulli(p):
    #Generate a uniform RV  U
    U = random.random()
    #Check if it is <=p
    if U <=p:
        return 1
    else:
        return 0
```

Generating a Y rv is done similarly:

```
import random
#Algorithm for generating a Y rv used in the binomial lattice model
def Y(p,u,d):
#Generate a uniform RV U
    U = random.random()

    if U <=p:
        return u
    else:
        return d
```

Generating a Y rv by using the Bernoulli (p) algorithm to do so:

```
#Algorithm for generating a Y rv used in the binomial lattice model but
#using a generated Bernoulli(p) to do it
def Y(p,u,d):
    #1 evaluates to true and 0 to false
    if Bernoulli(p):
        return u
    else:
        return d
```

Generating the BLM up to time n, $S_0, \ldots, S_n$:

```
def BLM(S0,n,p,u,d):
    #Initialize an empty array
    BLMpath = [0 for k in range(n+1)]
    #Set the first entry to our initial stock value S_0=S0
    BLMpath[0] = S0
    #Generate all $n$ of our iid Y samples Y_1,Y_2,...Y_n at once
    Ysamples = [Y(p,u,d) for k in range(n)]

    S = S0
    for k in range(1,n+1):
        S *= Ysamples[k-1]
        BLMpath[k] = S

    return BLMpath
```

Generating the BLM up to time n, $S_0, \ldots, S_n$ sequentially (recursively) using $S_{i+1} = S_i Y_{i+1}$:

```
#A more efficient algorithm that requires less storage
def BLMpath_recursive(S0,n,p,u,d):
    #Initialize an empty array
    BLMpath = [0 for k in range(n+1)]
     #Set the first entry to our initial stock value S_0=S0
    BLMpath[0] = S0
    #Generate our path by sampling from Y one at a time
    for k in range(1,n+1):
        BLMpath[k] = BLMpath[k-1]*Y(p,u,d)

    return BLMpath
```

1.3 The parameters u, d, p for the BLM

When used in practice, the BLM must take into account basic principles of economics, such as the premise that one can't make a fortune from nothing (arbitrage). Let $r > 0$ denote the interest rate that we all have access to if we placed our money in a savings account. If we started with an initial amount x_0 placed in the account, then one day later it would be worth $x_1 = (1+r)x_0$ and in general, n days later it would be worth $x_n = (1+r)^n x_0$. Thus the *present value* of x_n (at time n) is $x_0 = x_n/(1+r)^n$.

The following is the basic inequality that u, d, r must satisfy (theoretically) in practice

$$0 < d < 1 + r < u. \tag{3}$$

The idea is that if the stock goes up, we mean that is does better than the bank account interest rate, whereas if it goes down, we mean it does worse than the bank account interest rate: $uS_0 > (1+r)S_0$ and $dS_0 < (1+r)S_0$

Note that the expected value of a Y rv is $E(Y) = pu+(1-p)d$ and thus, $E(S_1) = S_0E(Y_1) = S_0[pu + (1-p)d]$ and in general.

$$E(S_n) = E(S_0Y_1 \times \cdots \times Y_n) = S_0[pu + (1-p)d]^n,\ n \geq 0.$$

In real applications, the amount $[pu + (1-p)d]$ would be considerably larger than $1 + r$, meaning that *on average*, investing in the stock is better than placing your money in the bank: $E(S_n) >> (1+r)^n S_0$.

That is why people buy stocks. But of course it is *risky* in that there is a chance (probability) that the stock will go down many times causing you to lose your money as compared to placing it (safely) in the bank at interest rate r.

Values of u, d and p for a given stock are determined by data and other considerations; you would learn more about how that is done in a financial engineering course.