Quantitative Research Case Study

Validus Risk Management

We consider a multiperiod binomial asset model for an asset S with N periods. Under this particular model, we have the following assumptions:

- the initial price of the asset is $S_0 = 1$;
- under the risk-neutral measure, the asset price at period j is $S_j = (1+v)S_{j-1}$ with probability $1/2$, and $S_j = (1+v)S_{j-1}$ with probability $1/2$, where $0 < v < 1$; and
- the interest rate associated with borrowing/lending currency for a single time period (i.e. the risk-free interest rate) is 0.

Question 1

We consider a European call option on the asset S, expiring after N periods at maturity time T, and with strike price K. According to the binomial model, during the lifetime of the option, the asset price at time t can either move up from S_t to $S_{t+1} = S_t u$, where $u = 1 + v > 1$, or move down to $S_{t+1} = S_t d$, where $d = 1 - v < 1$. If we denote the probability that the asset price will move upwards by p, where $p = 1/2$, then the probability of a downward move is given by $1 - p$. Also, the risk-free interest rate is denoted by r.

To calculate the current price V of the option, we will use the binomial pricing model, which assumes no arbitrage opportunities and a risk-neutral world. The idea of valuating the option using the binomial method is based on the following three steps:

- Generate the price tree for the underlying asset S
- Calculate the expected payoff from the option at maturity
- Work backwards in the tree to calculate the value of the option at each node

Before proceeding to the option valuation, we present a general formula for the parameter p, which denotes the probability that the asset will move upwards. Due to the assumption of absence of arbitrage opportunities and to the risk-neutral valuation principle, the probability p is given by:

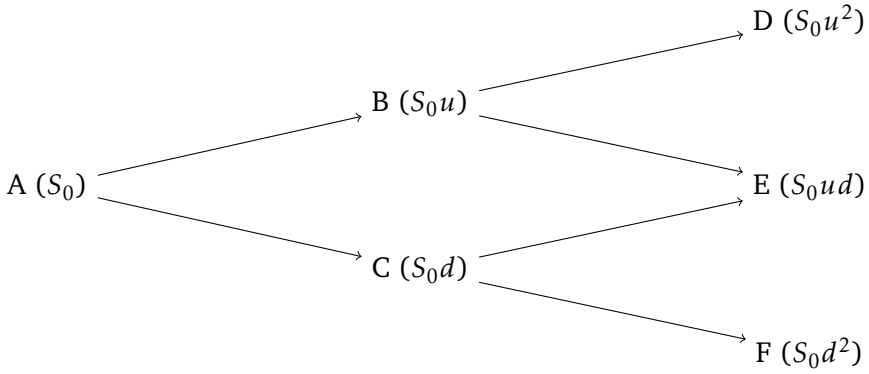
$$p = \frac{e^{rT} - d}{u - d} \tag{1}$$

In Python, we define the function `european_call` that receives as inputs the characteristics of the asset as well as of the European call option on this asset; return v of each upward move of the asset, strike price K, number of periods N until expiration, maturity time T in years, current price S_0 of the asset, and risk-free interest rate r per annum. Please note that two of the inputs in this function are fixed, due to the assumptions of $S_0 = 1$ and $r = 0$.

As shown in the code, the first step in the function is to calculate the risk-neutral probability using equation (1). For the given problem, where r is zero, $u = 1 + v$, and $d = 1 - v$, the risk-neutral probability will be equal to $1/2$, which is in line with our assumptions.

However, we present the general formula to make the function more flexible.

The second step in the function is to create the price tree for the underlying asset. For example, a two-step binomial tree can be found below. The price of the asset S is also displayed in the parenthesis of each node:



We initialise a NxN array of zeros to save the asset prices, and work from left to right in the binomial tree to calculate the asset price at each node. The (i, j) element of this array will correspond to the price of the asset at period j, where $0 \leq j \leq N$, and with i number of downward moves preceded.

The third step in the function is the calculation of the option payoff at maturity. At each of the final nodes of the binomial tree, the value of the option is equal to its exercise value. For a call option, this exercise value is equal to $\max\{S_T - K, 0\}$, where S_T is the price of the underlying asset on the maturity date.

The final step in the function is the calculation of the option prices at each node of the binomial tree. We initialise again a $(N+1) \times (N+1)$ array of zeros to save the option prices. To find these prices, we work backwards in the tree, by starting from the final nodes, where we have already found the option values, in the previous step.

For example, in the two-step binomial tree above, we consider that we have found the option value at nodes D and E, as they are the payoffs from the option. We will use these nodes to calculate the option price at node B. According to the principle of risk-neutral valuation, the option price is equal to its expected payoff in a risk-neutral world, discounted at the risk-free interest rate. Using this principle of the binomial pricing model, we can calculate the option value at node B, as follows:

$$V_B = e^{-rTN}[p * V_D + (1-p) * V_E] \tag{2}$$

By working backwards in the binomial tree and applying this formula to each node, we obtain the option price at the initial node A. The risk-neutral valuation principle holds for binomial trees with any number of steps, so the idea and the method to find the option price at the current time remains the same for any number of N periods. Having saved the option prices at our $(N+1) \times (N+1)$ array, the function returns finally the value V of the call option at time $t = 0$.

Question 2

We consider as before an asset S, whose price evolution over N time periods is described by the multiperiod binomial model given at the beginning of this document. This time, we consider a European call option on the asset S, whose value V at time $t = 0$ is known. We create a function named `move_calibration`, which will use the known option value V to calibrate the parameter v of the pricing model described in Question 1. In other words, the function will try to adjust the parameter v in order to achieve the best fit between the result of the pricing model and the observed option price.

Therefore, the function `move_calibration` in the Python code, takes as inputs the value V of the European call option, the strike price K, the number of periods N until maturity, the expiration date T, the initial price of the asset S_0 and the risk-free interest rate r.

The idea of calibrating the pricing model, i.e. of finding the optimal parameter v so that the pricing model `european_call` built in Question 1 will give a result as close as possible to the value of V, is described by the following optimisation problem:

$$\min_{0<v<1} |V - C(v)| \tag{3}$$

where $C(v)$ is the option price obtained from the pricing function `european_call`, given v. As mentioned earlier, the risk-neutral valuation principle of the binomial model indicates that the option price is equal to its expected payoff, at a risk-neutral world, discounted at the risk-free interest rate.

If we denote as P_{iN} the payoff from the option at the final node (i, N), where i denotes the number of upward moves that have been preceded to reach to this node, then:

$$P_{iN} = \max\{S_0 u^j d^{N-j} - K, 0\}, \qquad i = 1, \ldots, N$$

In a binomial model, the total number of upward movements follows a binomial distribution with parameters N and p. Given the properties of a binomial distribution, the expected payoff of the option, discounted at the risk-free rate is given by:

$$C(v) = e^{-rT} \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} \max\{S_0 u^i d^{N-i} - K, 0\}, \qquad u = 1 + v,\ d = 1 - v$$

In this framework, the function `move_calibration` starts by defining the objective function of the optimisation problem, called `option_value_diff`, which is given by:

$$f(v) = \left| V - e^{-rT} \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} \max\{S_0 (1+v)^i (1-v)^{N-i} - K, 0\} \right|$$

The object function will be minimised with respect to the parameter v, so an initial value for the variable $v \in (0, 1)$ should first be set before performing the optimisation. Thus, the function `move_calibration` initialises the calibration parameter to v_0 and then performs the optimisation, using `scipy.optimize.fmin` minimisation function.

In the final step, the function returns the value $\widehat{v}$ that solves the optimisation problem (3), which is therefore the result of the calibration.

Question 3

In Question 1, we considered the pricing of a European call option. We will now consider the valuation of an American call option. The main difference between an American and a European option, is that a European option can be exercised only at maturity, while an American option can be exercised at any time prior to maturity. Thus, the pricing method to be applied to the American call options is the same as the one in Question 1, with the only difference being that when working backwards in the binomial model, we have to check at each node whether an early exercise is optimal.

Therefore, the function `american_call` in the Python code receives the same inputs as the function `european_call` of Question 1 and implements the same first three steps, as described earlier:

1. Calculate the risk-neutral probability.
2. Generate the binomial price tree for the underlying asset.
3. Calculate the expected payoffs from the option at maturity.

In the fourth step, the function works backwards on the binomial tree and calculates the option price at each node, starting with the final nodes of the tree and reaching the initial one. At each node of the tree, the function checks if an early exercise is optimal, by comparing the following two quantities:

- The option price on the node given by equation (2);
- The payoff from an early exercise of the call option. This equals to $S_{ij} - K$, where S_{ij} is the price of the asset at the given node (i, j), with $0 \leq i, j \leq N$.

The maximum of these two quantities will be the value of the option on this node. Repeating this process until the beginning of the tree, the function finds and returns the current value of the option. In addition, the function prints the time $t \in [0, T]$ at which the exercise of the American call option is optimal.

Question 4

We now want to calculate the expectation of the maximum of the stock prices over the N time periods, given by $\mathbb{E}[\max_{0\leq j\leq N} S_j]$. The expectation of a variable X with possible outcomes $x_1,\ldots,x_n$, each of which has probability $p_1,\ldots,p_n$ of occurring, is defined as:

$$\mathbb{E}(X) = \sum_{k=0}^{n} x_k p_k \tag{4}$$

The idea is to use the above definition to calculate $\mathbb{E}[\max_{0\leq j\leq N} S_j]$, and in order to do that we have to calculate:

- The possible outcomes of variable $\max_{0\leq j\leq N} S_j$.
- The probability that each of these outcomes will occur.

To achieve the first part, the idea is to find all possible paths of the binomial tree, and to calculate the maximum asset price in each of these paths. Then, the possibility that this maximum asset price will be achieved is equal to the possibility that this path will be generated. In a binomial tree with N steps, the total number of possible paths is 2^N.

In terms of steps, the `expected_max` function in Python starts by generating in its first two steps, the binomial price tree for the asset S. So, it calculates the risk-neutral probability and the asset prices at each node of the tree, in the same way represented in functions `european_call` and `american_call` earlier.

In the third step, the function calculates the probability of occurrence of each of the 2^N possible paths, and saves all these probabilities in the `prob_path` array. The process of calculating these probabilities is:

1. Express the pair (up,down) of the asset move at each node in the form $(1,0)$,
2. Use the Cartesian product of the N pairs $(1,0)$ to find all the possible combinations of movements in the asset price,
3. Calculate the number of upward moves at each possible path,
4. Calculate the possibility of occurrence of each path as the product of $p^u \times (1-p)^{N-u}$, where u is the total number of upward moves

In the fourth step, the function also creates an array, named `path_array`, where each row represents a possible path of the binomial tree. Once the `path_array` is completed, the function finds the maximum asset price in each path. The 2^N maximum asset prices are stored in the `max_path` array. Finally, using the definition (4), the function calculates and returns the value of $\mathbb{E}[\max_{0\leq j\leq N} S_j]$.