Options

2024-03-01

Most of these will be option pricing algorithms.

Binomial Lattice

Consider the $N-$period binomial model with $0 < d < e^{r\Delta t} < u$. Suppose the derivative payout at maturity V_N is a random variable with known distribution.

Idea: Starting from the leaves (known V_N), we will recursiveley go down levels $n = N-1, \ldots, 0$ computing discounted expected values.

- Algorithm 2.1: If we wish to price **path-dependent** options, we must keep track of **exact** path evolution at each time n:$(\omega_1, \ldots, \omega_n)$

$$V_n(\omega_1, \ldots, \omega_n) = e^{-r\Delta T}\mathbb{E}^{\mathbb{Q}}(V_{n+1}|\mathcal{F}_n)$$

- Algorithm 2.2: To price **path-independent** option, we can simplify Algorithm 2.1 and just count the number of ups. This uses the fact that V_t^j is the same in the binomial lattice no matter independent of the path and only dependent on the number of ups, ie. trajectories $(up, down, up)$, $(down, up, up)$ and $(up, up, down)$ have the same stock value V_3^2.

Algorithm 2.1

1. Compute all possible payouts at maturity. $V_T(\omega_1, \ldots \omega_n)$ for $\omega_1, \ldots \omega_n \in \Omega = \{up, down\}$
2. For $n = N-1, \ldots, 0$ do: for all 2^n states $W = \omega_1, \ldots \omega_n$ compute:

$$V_n(W) = e^{-r\Delta t}(q^u V_{n+1}(W, up) + q^d V_{n+1}(W, down))$$

This has a complexity of 2^T when implemented in a naive way because it goes through all possible 2^T evolution. Depending on the option this can be improved.

Algorithm 2.2

To find the price of an option

1. Given S_0, d, u, p^u we will compute the lattice evolution up to time N where:

$$S_t^j = S_0 u^j d^{t-j}$$

   for all $t = 1, 2, \ldots, N$, $j = 0, \ldots, t$. Note that S_t^j stands for "stock price at time t with state $j-$ups from t=0"

2. Compute the payouts at maturity (the leaves of the lattice). V_T^j

```
for (j in 1:T) {
  V[T][j] = max((S[T][j]), 0)## for European call for instance
}
```

3. Compute $V[t][j]$ backwards in time until you get to $V[0][0]$:

```
# d, u are down and up steps
algo21 <- function(u,d,r, V, T){
  q_u = (e^r-d)/(u-d) # risk-free probability of up
  q_d = 1-q_u
  for (t in T-1:0){
    for (j in 0:t){
      V[t][j] = (e^(-r)*(q_u* V[t+1][j+1] + q_d*V[t+1][j])
    }
  }
  return V[0][0]
}
```