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.

Initialization

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"

Algorithm 2.1

To find the price of any option

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

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

4. Return V_0

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 a path-independent option

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]
}
```

American Option pricing

In american options we can exercise at any time up until maturity. We must keep track of two values at time t:

1. V_t^{ex}: The current exercise value, payout at time t.
2. V_t^{cont}: The continuation value, the value of the option if not exercised at time t (present value of what it is expected to give us).

Assuming rational agents, at any time t in an American option, we exercise if $V_t^{ex} > V_t^{cont}$ and hold otherwise.

We can adapt **Algorithm 2.1** to price this:

Algorithm 2.3:

1. Compute all possible payouts at maturity $V_T(\vec{w})$ corresponsing to all possible states $\vec{w} = (\omega_0, \ldots, \omega_T) \in \Omega = \{up, down\}$
2. Go backwards in time. For $n = N-1, \ldots, 0$ do:

- for all states $\vec{w}$ do:
    a. Compute the continuation value $V_n^{cont}(\vec{w}) = e^{-r\Delta t}(q^u V_{n+1}(\vec{w}, up) + q^d V_{n+1}(\vec{w}, down))$
    b. Compute the execution value $V_n^{ex}(\vec{w}) = max\{S_n(\vec{w}) - K, 0\}$
    c. Compute the rational value $V_n(\vec{w}) = max\{V_n^{ex}(\vec{w}), V_n^{cont}(\vec{w})\}$

3. Return V_0

Just like with Algorithm 2.2, if we have a path-independent option we can simplify this algorithm by expressing values in terms of number of up states V_t^j as opposed to expressing them in terms of entire state evolutions $V_t(\vec{\omega})$

Black-Scholes Model

We assume the stock price S_t is an Ito-process. Ito-processes satisfy Geometric Brownian Motion dynamics.

$$dS_t = \mu S_t dt + \sigma S_t dB_t$$

This line encapsulates alll our asusmptions.

Also we can show using Ito's lemma that the natural logarithm of an ito process satisfies Arithmetic Brownian motion dynamics:

$$dln(S_t) = (\mu - \frac{\sigma^2}{2})dt + \sigma dB_t$$

And we can solve this without Ito Integrals because the coefficients of the infinitesimal changes (dt, dB_t) are constant.

The solution is $S_t = S_0 \exp\{(\mu - \frac{\sigma^2}{2})t + \sigma B_t\}$

For Monte Carlo Algorithms, we will use the aforementioned GBM solution as well as the fact that:

$$V_0 = e^{rT}\mathbb{E}^{\mathbb{Q}}(\text{payout}((S_t)_{t\in[0,T]}))$$

Also under $\mathbb{Q}$, $\mu = r$ in our GBM solution (**Not sure how to prove this**), so we can say:

$$S_t = S_0 \exp\{(r - \frac{\sigma^2}{2})t + \sigma B_t\}$$

Monte Carlo Algorithms

Simulation algorithms in which we sample from a distribution. We will sample n different paths and then compute $n-$different payouts based on the realizations. We will then return the average.

Algorithm 3.1

Given n: number of simulation runs, N : number of time discretizations suth that for a maturity time horizon T, $T/N = \Delta t$.

1. Discretize time s.t. we only simulate at $0 < t_1 < t_2 < \cdots < t_N$
2. For $i = 1, \dots n$ (number of simulation runs):
   - Sample a discretized path $S_1, \dots S_N$ corresponding to $t_1, \dots t_N$.
   - Compute the payout in simulation run i, say V_N^j. Unlike with binomial models we don't need to know V_t for $t < N$.
3. Return the estimate $\hat{V}_0 = e^{-rT}\frac{1}{n}V_N^i$ (the discounted expected value of the average payout at maturity.)

This can be used for path-dependent (Asian) or independent options (European, rainbow?) but, as the algorithm is, the options should be exercised at maturity i.e. no american-type options (you probably just modify it a bit like with algo 2.3)

Sources of error for Monte Carlo:

- Discretization error as we sample only at $t_1 < t_2 < \cdots < t_N$ and not continuosuly (impossible on a finite resource computer)
- Sampling error or MC error, due to only sampling n paths. We can't ever sample all uncountably many paths.