TD5

October 2020

1 Black-Scholes model

We build a class that allows to compute the price of a European vanilla option, and also its Δ.
A European vanilla option has the following characteristics:

- Type: Call or Put (to be modelled with an enum)
- Strike price: K
- Expiry date: T

Its price depends on the following market data:

- Underlying price: S
- Interest rate: r

The following parameter is also required in order to price the option:

- Volatility: σ

Write a method that computes the price and the delta of the options. The Black-Scholes formula can be found on the internet.

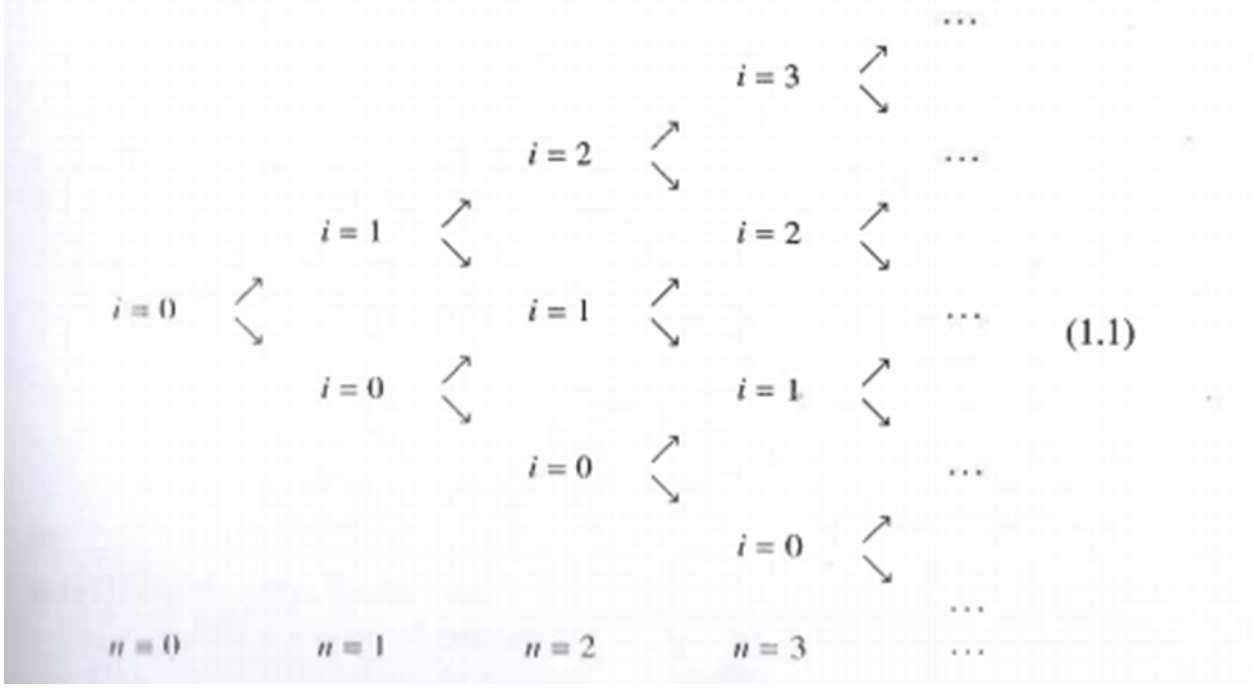
2 The CRR model

Implement a class that allows to compute the price of an option using the CRR method.

In the **binomial model** the prices of assets evolve in discrete time steps $n = 0, 1, 2, \ldots$. There is a **stock** whose price evolves randomly by moving up by a factor $1 + U$ or down by $1 + D$ independently at each time step, starting from the spot price $S(0)$. As a result, the stock price becomes

$$S(n, i) = S(0)(1 + U)^i(1 + D)^{n-i}$$

at step n and node i in the binomial tree



(1.1)

where $S(0) > 0$, $U > D > -1$ and $n \geq i \geq 0$. There is also a risk-free security, a **money market account**, growing by a factor $1 + R > 0$ during each time step. The model admits no arbitrage whenever $D < R < U$.

Within the binomial model the price $H(n, i)$ at each time step n and node i of a **European option** with expiry date N and payoff $h(S(N))$ can be computed using the **Cox–Ross–Rubinstein (CRR) procedure**, which proceeds by backward induction:

- At the expiry date N

$$H(N, i) = h(S(N, i)) \tag{1.2}$$

  for each node $i = 0, 1, \ldots, N$.
- If $H(n + 1, i)$ is already known at each node $i = 0, 1, \ldots, n + 1$ for some $n = 0, \ldots, N - 1$, then

$$H(n, i) = \frac{qH(n + 1, i + 1) + (1 - q)H(n + 1, i)}{1 + R} \tag{1.3}$$

  for each $i = 0, 1, \ldots, n$.

Here

$$q = \frac{R - D}{U - D}$$

is the **risk-neutral probability**. In particular, for a **call option** the payoff function is

$$h^{\text{call}}(z) = \begin{cases} z - K & \text{if } z > K, \\ 0 & \text{otherwise.} \end{cases} = (z - K)^+$$

and for a **put option** it is

$$h^{\text{put}}(z) = \begin{cases} K - z & \text{if } z < K, \\ 0 & \text{otherwise.} \end{cases} = (K - z)^+$$

for all $z > 0$, where K is the **strike price**.

TD6

November 2020

1 The CRR method - Closed-form formula

The CRR method provides also a closed-form formula for option pricing, instead of the iterative procedure seen in TD5:

$$H(0,0) \quad = \quad \frac{1}{(1+R)^N} \sum_{i=0}^{N} \frac{N!}{i!(N-i)!} q^i (1-q)^{N-i} h\left(S\left(N,i\right)\right).$$

Extend the CRRPricer created in TD5, so that it offers the possibility of pricing an option also by using the closed-form version for option pricing.

2 Option payoffs

Define an abstract Option class with the attributes common to all the option types and with a payoff method, so that separate classes (children of the parent Option class) can be defined for each option type.

Each option class has to exhibit its own payoff method depending on the option type (design is up to you).

Here are some option types to be implemented:

- European Call: $h\left(z\right) = \left(z-K\right)^+$
- European Put: $h\left(z\right) = \left(K-z\right)^+$
- European Digital Call: $h\left(z\right) = 1_{z \geq K}$
- European Digital Put: $h\left(z\right) = 1_{z \leq K}$

where K is the strike (a.k.a. exercise price).

3 BinLattice

Implement a class *BinLattice* that represents the data structure (path tree) used for the CRR method:

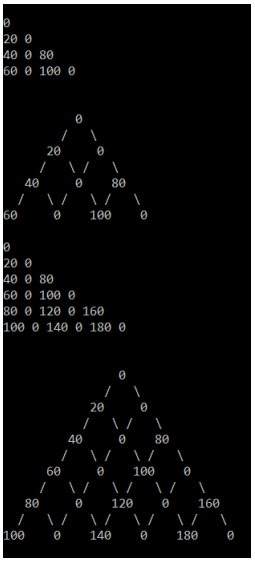
The class contains two variables:

2. `N` to store the number of time steps in the binomial tree,
3. `Lattice`, a vector of vectors to hold data of type `double`.

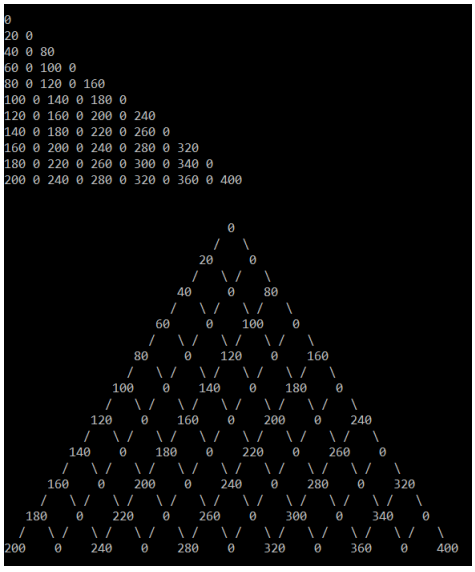
The `BinLattice` class also contains the following functions:

4. The `SetN()` function takes a parameter of type `int`, assigns it to `N`, sets the size of the `Lattice` vector to `N+1`, the number of time instants `n` from `0` to `N`, and then for each `n` sets the size of the inner vector `Lattice[n]` to `n+1`, the number of nodes at time `n`.
5. `SetNode()` to set the value stored at step `n`, node `i`,
6. `GetNode()` to return the value stored at step `n`, node `i`,
7. `Display()` to print the values stored in the binomial tree lattice.

The class has to be implemented as a template class, where the type of the data held by the vector of vectors Lattice can be a custom one (the double type mentioned here above is an example).



(a) Examples with N=3 and N=5



(b) Examples with N=10

Figure 1: Examples of output by the display function

TD7

November 2020

1 Some option pricing theory

1.1 European options and path-dependent option

We consider a risky asset with the Black-Scholes dynamics:

$$S_t = S_0 e^{\left(r-\frac{\sigma^2}{2}\right)t+\sigma W_t},$$

where $\sigma \in \mathbb{R}^+$ is the volatility and W_t a Wiener process under the risk neutral probability $\mathbb{Q}$.
We denote the price (at time 0) of an option by H_0.
This price can be determined by computing the expected discounted payoff under $\mathbb{Q}$:

$$H_0 = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[H_T\right],$$

where H_T denotes the payoff of the option at its expiry date T.

1.1.1 European options

In the case of a European option, $H_T = h(S_T)$, where $h : \mathbb{R}^+ \to \mathbb{R}$ is the payoff function of the option (see TD6), it only depends on the price of the risky asset at maturity.

1.1.2 Path dependent options

For more complex options, the payoff H_T also depends on the price of the risky asset at dates prior to the maturity.
These are called path dependent options.
Let $t_k = \frac{k}{m}T$, for $k = 1, \cdots, m$. A path-dependent option is a financial derivative with payoff at expiry date T:

$$H_T = h\left(S_{t_1}, \cdots, S_{t_m}\right),$$

where $h : (\mathbb{R}^+)^m \to \mathbb{R}$ is the payoff function.
For instance, the arithmetic Asian Call has the following payoff function:

$$h\left(z_1, \cdots, z_m\right) = \left(\left(\frac{1}{m}\sum_{k=1}^{m} z_k\right) - K\right)^+.$$

1.2 Random paths

The Wiener process W has independent increments, with $W_t - W_s \sim \mathcal{N}(0, t-s)$ for $0 \leq s < t$. S_{t_k} can be expressed as

$$S_{t_k} = S_{t_{k-1}} e^{\left(r - \frac{\sigma^2}{2}\right)(t_k - t_{k-1}) + \sigma\sqrt{t_k - t_{k-1}} Z_k}$$

Where $Z_1, \cdots, Z_m$ are i.i.d. random variables with distribution $\mathcal{N}(0,1)$.
Let the seqence $\widehat{Z}_1, \cdots, \widehat{Z}_m$ be a i.i.d. sample of $Z_1, \cdots, Z_m$. We refer the sequence $\left(\widehat{S}_{t_1}, \cdots, \widehat{S}_{t_m}\right)$ defined by:

$$\begin{aligned}
\widehat{S}_{t_1} &= S_0 e^{\left(r - \frac{\sigma^2}{2}\right) t_1 + \sigma\sqrt{t_1}\widehat{Z}_1}, \text{ for } k = 2, \cdots, m, \\
\widehat{S}_{t_k} &= \widehat{S}_{t_{k-1}} e^{\left(r - \frac{\sigma^2}{2}\right)(t_k - t_{k-1}) + \sigma\sqrt{t_k - t_{k-1}}\widehat{Z}_k}, \text{ for } k = 2, \cdots, m,
\end{aligned}$$

as a sample path.

1.3 Monte Carlo

Let $\left(\widehat{S}^i_{t_1}, \cdots, \widehat{S}^i_{t_m}\right)$, for $i \in \mathbb{N}$, be a sequence of independent sample paths. By the law of large numbers

$$\mathbb{E}^{\mathbb{Q}}\left[h(S_{t_1}, \cdots, S_{t_m})\right] = \lim_{N \to \infty} \frac{1}{N} \sum_{i=0}^{N-1} h\left(\widehat{S}^i_{t_1}, \cdots, \widehat{S}^i_{t_m}\right).$$

This means that for sufficient large N, we can approximate H_0 using

$$H_0 \approx e^{-rT} \frac{1}{N} \sum_{i=0}^{N-1} h\left(\widehat{S}^i_{t_1}, \cdots, \widehat{S}^i_{t_m}\right)$$

2 Programming

Design a class that price path dependent options and European options using the Monte Carlo method. Design the class for it to be able to refine its estimate and to provide with interval of confidence (as in TD2).

TD8

December 2020

1 American option in the binomial model

In addition to pricing European options, we want to include the ability to price American options in the binomial model.

The holder of an American option has the right to exercise it at any time up to and including the expiry date N. If the option is exercised at time step n and node i of the binomial tree, then the holder will receive payoff $h(S(n,i))$.

The price $H(n,i)$ of an American option at any time step n and node i in the binomial tree can be computed by the following procedure, which proceeds by backward induction on n:

- At the expiry date N: $H(N,i)$ has the same value as for the option's European counterpart. Financial interpretation: if not exercised before the expiry, there is no advantage holding an American option over holding a European option.
- If $H(n+1,i)$ is already known at each node $i \in \{0,\cdots,n+1\}$ for some $n < N$, then

$$H(n,i) = \max\left\{\underbrace{\frac{qH(n+1,i+1)+(1-q)H(n+1,i)}{1+R}}_{\text{continuation value}}, \underbrace{h(S(n,i))}_{\text{intrinsic value}}\right\}$$

  for each $i \in \{0,\cdots,n\}$.
  Financial interpretation: the option holder chooses the maximum between the continuation value (expected gain if they do not exercise, under the risk-neutral measure) and the intrinsic value (the value of the option if exercised immediately).

In particular, $H(0,0)$ at the root node of the tree is the price of the American option at time 0.

We would like to compute and store the price of an American option for each time step n and node i in the binomial tree. In addition, we want to compute the early exercise policy, which should be of Boolean type and tells if the American option should be exercised or not for each state of the tree. The early exercise policy should also be stored using the class BinLattice.

Programming: extend your Binomial tree class to price American options as well.

2 Black-Scholes as limit of the binomial tree

The binomial model can be used to approximate the Black-Scholes model if N is large.

One of the scheme is to divide the time interval $[0, T]$ into N steps of length $h = \frac{T}{N}$, and set the parameters of the binomial model to be:

$$\begin{aligned} U &= e^{\left(r+\frac{\sigma^2}{2}\right)h+\sigma\sqrt{h}} - 1, \\ D &= e^{\left(r+\frac{\sigma^2}{2}\right)h-\sigma\sqrt{h}} - 1, \\ R &= e^{rh} - 1, \end{aligned}$$

where σ is the volatility and r is the continuously compounded interest rate in the Black-Scholes model.

Implement a method to initialize a Binomial tree as a Black-Scholes approximation (using the Black-Scholes parameters). Compare option prices with the Monte Carlo method and the closed form method for European options.

TD4 - Random walks

October 2020

A random walk is a stochocastic process, that describes a path consisting of a succession of random steps on some mathematical space (e.g. $\mathbb{R}$).

- Define an object RandomWalk, which represents a random walk path of n time steps. The object has to:
  - store the values of the walk at the different time steps $\{0, \cdots, n\}$,
  - store the distribution type, out of which the walk has been drawn (see the following),
  - possess the specific getters for retrieving the information (n, distribution, other attributes),
  - expose methods to compute average and variance/standard deviation of the current walk instance,
  - have an operator [] that allows to retrieve the value of the walk at an instant t in the interval $\{0, \cdots, n\}$; a specific time_index_invalid_exception has to be thrown if an invalid index is specified.
- Define an object RandomWalkGenerator, which is a singleton, with a method *generate()* that generates a random walk of n timesteps drawn from a distribution (chosen from a set (enum)) with specific parameters.
  - Random walks have to be generated by considering a random initial value (it can be an optional parameter of the *generate()* method),
  - Succession of values has to be generated in the following way:

    $$v(t+1) \quad = \quad v(t) + \xi_{t+1},$$

    where $(\xi_t)_t$ is the innovation process assumed to be i.i.d., it will be generated from a specific distribution, upon the call of the method *generate()*.
- Design something to enable to output the values of several instances of RandomWalk (with the same generation parameters) into a .csv file. Then plot the file in Excel to check your code.