TD5-6

October 2022

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.
Hint: use *std::erfc*.

2 The Cox-Ross-Rubinstein model

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

In the CRR model the price of an asset evolves in discrete time steps ($n = 0, 1, 2, \cdots$). Randomly, it can move up by a factor $1+U$ or down by $1+D$ independently at each time step, starting from the spot price S_0 (see Figure below).

$$
\begin{array}{ccccccccc}
 & & & & & & & & \cdots \\
 & & & & & & & \nearrow & \\
 & & & & & & i=3 & & \\
 & & & & & \nearrow & & \searrow & \\
 & & & & i=2 & & & & \cdots \\
 & & & \nearrow & & \searrow & & \nearrow & \\
 & & i=1 & & & & i=2 & & \\
 & \nearrow & & \searrow & & \nearrow & & \searrow & \\
i=0 & & & & i=1 & & & & \cdots \\
 & \searrow & & \nearrow & & \searrow & & \nearrow & \\
 & & i=0 & & & & i=1 & & \\
 & & & \searrow & & \nearrow & & \searrow & \\
 & & & & i=0 & & & & \cdots \\
 & & & & & \searrow & & \nearrow & \\
 & & & & & & i=0 & & \\
 & & & & & & & \searrow & \\
 & & & & & & & & \cdots \\
n=0 & & n=1 & & n=2 & & n=3 & & \cdots
\end{array}
$$

Figure 1: Binary Tree

As a result, the stock price at step n and node i is:

$$S(n, i) \quad = \quad S_0 (1+U)^i (1+D)^{n-i},$$

where $S_0 > 0, U > D > -1$ and $0 \leq i \leq n$. There is also a risk-free asset which grows by the factor $1 + R > 0$ at each time step (starting at 1 at step 0).

The model admits no arbitrage iif $D < R < U$.

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

- At the expiry date N:

$$H(N,i) = h(S(N,i))$$

  for each node $i = 0, \cdots, N$.

- If $H(n+1,i)$ is already known for all nodes $i = 0, \cdots, n+1$ for some $n = 0, \cdots, N-1$, then

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

  for each $i = 0, \cdots, n$; and where q is defined by

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

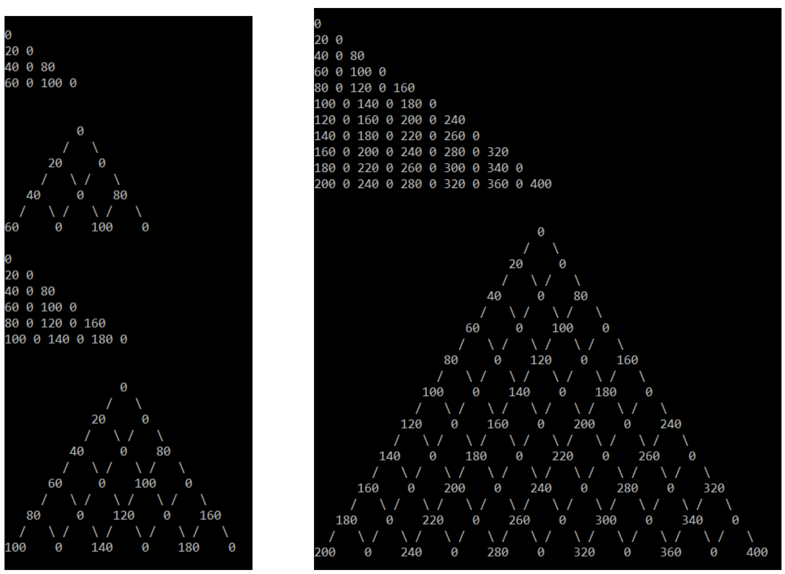
  is called the **risk-neutral probability**.

3 Starting the project!

1. Implement the abstract class *Option:*
   - with a private member *double _expiry*, along with a getter method *getExpiry()*
   - with a pure virtual method *double payoff(double)*, *payoff()* represents the function h
   - write a constructor that initialize *_expiry* with an argument
2. Derive *Option* into another abstract class *VanillaOption*:
   - with private attributes *double _strike*
   - write a constructor which initialize *_expiry* and *_strike* with arguments (call the base constructor)
   - the constructor should ensure that the arguments are nonnegative
   - write a **classe enum** *optionType* that has two values: *call* and *put*
   - write an pure virtual method *GetOptionType()* which should return an *optionType* enum
3. Derive *VanillaOption* into two classes: *CallOption* and *PutOption.*
   - They should use the constructor of *VanillaOption*
   - For a Call option with strike K, the payoff is given by $h(z) = \begin{cases} z - K & \text{if } z \geq K \\ 0 & \text{otherwise.} \end{cases}$
   - For a Put option with strike K, the payoff is given by $h(z) = \begin{cases} K - z & \text{if } K \geq z \\ 0 & \text{otherwise.} \end{cases}$
   - Override the *GetOptionType()* accordingly in the derived classes
4. Create the class *BlackScholesPricer*
   - With constructor *BlackScholesPricer(VanillaOption* option, double asset_price, double interest_rate, double volatility)*
   - Declare *BlackScholesPricer* as a friend class of *VanillaOption* in order for the former to access the strike of the latter
   - Write the *operator()* which returns the price of the option
   - Write the method *delta()* which returns the Delta of the option

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

   - It should be a template class *BinaryTree*<*T*>
   - It should have a member *_depth*, representing N
   - It should contain a private member *_tree*, a vector of vectors (STL) to hold data of type T
   - Implement the setter method *setDepth(int)* a setter for *_depth*, that resizes *_tree* and allocate/deallocate properly the vectors in *tree*
   - Implement the setter method *setNode(int, int, T)* which sets the value stored in *_tree* at the given indices
   - Implement the getter method *getNode(int, int)* which retrives the corresponding value
   - Implement the method *display()* which prints the all the values stored



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

(b) Examples with N=10

Figure 2: Examples of output by the *display()* function

6. Create the class CRRPricer
   - With constructor *CRRPricer(Option* option, int depth, double asset_price, double up, double down, double interest_rate)*
     - *depth*: N
     - *asset_price*: S_0
     - *up*, *down*, *interest_rate*: U, D, R respectively
   - In the constructor, check for arbitrage
   - Create the tree structure to store the tree of the desired depth (hint: use *BinaryTree* with an appropriate type)
   - Write the method *void compute()* that implements the CRR procedure
   - Write the getter method *get(int, int)* that returns $H(n, i)$.
   - Write the *operator()* which returns the price of the option, it must call *compute()* if needed
   - The CRR method provides also a closed-form formula for option pricing:

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

     Put an optional argument *bool closed_form* that defaults to *false* to the *operator()*. When it is set to true, the above formula should be used instead of the CRR procedure.

7. Similarly to *VanillaOption*, design *DigitalOption* and its derived classes (*DigitalCallOption* and *DigitalPutOption*) in order to take into account the following type of options:
   - Digital Call with payoff: $h(z) = 1_{z \geq K}$
   - Digital Put with payoff: $h(z) = 1_{z \leq K}$
   - Enable *BlackScholesPricer* to price digital options as well (closed form formulas also exist for Black-Scholes prices and deltas for digital options)