TD 5 - C++ for Finance

1. Build up a class modelling a European Option, and its pricing and delta.

A European option has the following attributes:

- Interest rate: *r*
- Volatility: *sigma*
- Strike price: *K*
- Expiry date (or maturity): *T*
- Underlying price: *S*
- Cost of carry: *b*

Options have also a type, that can be *call* or *put* (to be modelled with an enum).

Black-Scholes formula is an analytical method used for option pricing:

Type	**Price**
Call	$C(S,t) = S_t N(d_1) - Ke^{-r(T-t)} N(d_2)$
Put	$P(S,t) = Ke^{-r(T-t)} N(-d_2) - S_t N(-d_1)$

where:

$$d_1 = \frac{\ln \frac{S_t}{K} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

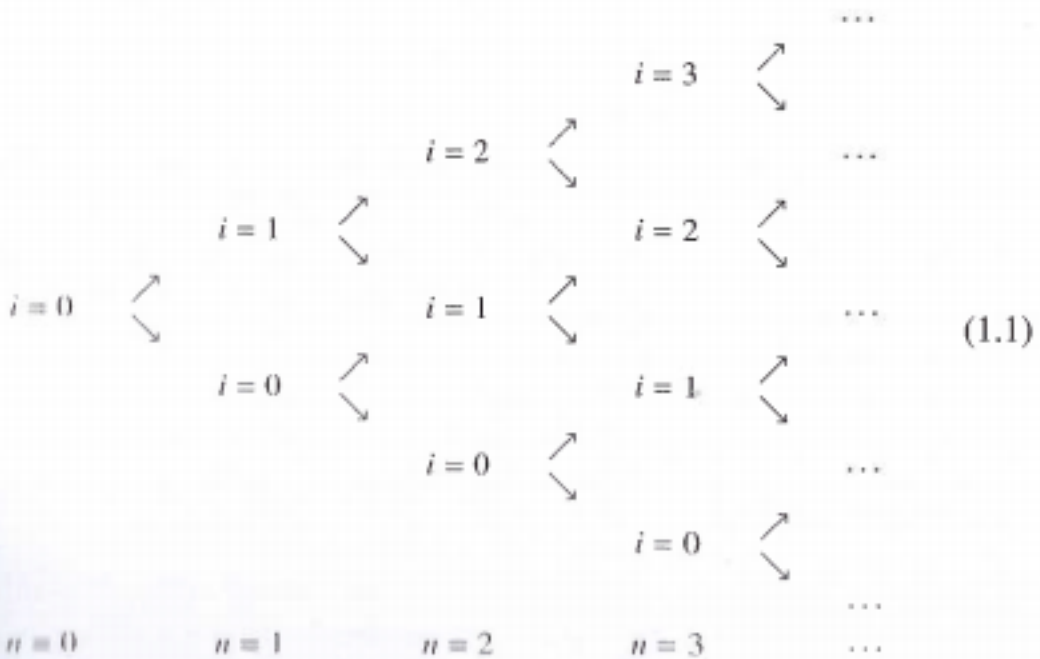
and σ^2 is the percentage variance of the logarithm of the underlying price, on an yearly basis.

2. Implement an option pricer according to 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**.

TD 6 - C++ for Finance

1. The CRR method provides also a closed-form formula for option pricing, instead of the iterative procedure seen in TD #5.

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

Extend the *CRRPricer* developped in TD #5, so that it offers the possibility of pricing an option also by using the closed-form version for option pricing.

2. Option payoff depends on option type, for example:

Option Type	Payoff (Call)	Payoff (Put)
European option	$h^{\text{call}}(z) = \begin{cases} z-K & \text{if } z > K, \\ 0 & \text{otherwise.} \end{cases} = (z-K)^+$	$h^{\text{put}}(z) = \begin{cases} K-z & \text{if } z < K, \\ 0 & \text{otherwise.} \end{cases} = (K-z)^+$
Digital option[1]	$h^{\text{digit call}}(z) = \begin{cases} 1 & \text{if } K < z, \\ 0 & \text{otherwise.} \end{cases}$	$h^{\text{digit put}}(z) = \begin{cases} 1 & \text{if } K > z, \\ 0 & \text{otherwise.} \end{cases}$

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).

[1] A **digital option** is an option whose payout is fixed after the underlying stock exceeds the predetermined threshold or strike price. It is also referred to as a **binary** or **all-or-nothing option**.

3. Implement a class *BinLattice* that represents the data structure (path tree) used for the Binomial 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).

TD 7 - C++ for Finance

General recall on theory for option pricing

We consider a risky asset

$$S(t) = S(0)e^{\left(r-\frac{\sigma^2}{2}\right)t+\sigma W_Q(t)},$$

where $\sigma \in \mathbb{R}$ is the volatility and $W_Q(t)$ is a Wiener process under the risk-neutral probability Q.

We denote the price of an option by $H(0)$. The price can be determined by computing the expected discounted payoff under Q

$$H(0) = e^{-rT}\mathbb{E}_Q(H(T)).$$

1. **The Monte Carlo method for option pricing**

A) The generation of random paths for underlying prices

The Wiener process W_Q has independent increments, with $W_Q(t) - W_Q(s)$ having normal distribution $N(0, t-s)$ for any $t > s \geq 0$.
$S(t_k)$ can therefore 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 $Z_1, \ldots, Z_m$ are independent and identically distributed (i.i.d.) random variables with distribution $N(0, 1)$.

Let $\hat{Z}_1, \ldots, \hat{Z}_k$ be a sequence of independent samples of $Z_1, \ldots, Z_k$, respectively. We refer to the sequence

$$\left(\hat{S}(t_1), \ldots, \hat{S}(t_m)\right),$$

defined by

$$\hat{S}(t_1) = S(0)e^{\left(r-\frac{\sigma^2}{2}\right)t_1+\sigma\sqrt{t_1}\hat{Z}_1},$$

$$\hat{S}(t_k) = \hat{S}(t_{k-1})e^{\left(r-\frac{\sigma^2}{2}\right)(t_k-t_{k-1})+\sigma\sqrt{t_k-t_{k-1}}\hat{Z}_k}, \quad \text{for } k = 2, \ldots, m,$$

as a **sample path**

B) The computation of option price

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

$$\mathbb{E}_Q(h(S(t_1), \ldots, S(t_m))) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} h(\hat{S}^i(t_1), \ldots, \hat{S}^i(t_m)).$$

This means that for sufficiently large N, we can approximate $H(0)$ using

$$H(0) \approx \hat{H}_N(0) = e^{-rT} \frac{1}{N} \sum_{i=1}^{N} h(\hat{S}^i(t_1), \ldots, \hat{S}^i(t_m)). \qquad (5.9)$$

Sample paths can be generated on a computer. This makes (5.9) a practical tool for the pricing of options.

- **Implement a Monte Carlo pricer to price a European option** (call or put)

2. **Path-dependent options**

Let $t_k = \frac{k}{m}T$ for $k = 1, \ldots, m$. A **path-dependent option** is a financial derivative with payoff at expiry date T

$$H(T) = h(S(t_1), \ldots, S(t_m)),$$

where $h : \mathbb{R}^m \to \mathbb{R}$ is a given deterministic payoff function. A typical example of a path-dependent option is the **arithmetic Asian call**, where the payoff function is given by

$$h^{\text{arithm Asian call}}(z_1, \ldots, z_m) = \left(\frac{1}{m} \sum_{k=1}^{m} z_i - K \right)^+ .$$

We denote the price of a path-dependent option by $H(0)$. The price can be determined by computing the expected discounted payoff under Q

$$H(0) = e^{-rT} \mathbb{E}_Q(H(T)).$$

In the majority of cases, including the arithmetic Asian call, such expectation cannot be computed analytically.

- **Extend the Monte Carlo pricer so that it can price an arithmetic Asian call option**

TD 8 - C++ for Finance

A) Pricing an American option with the Binomial method

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 (1.1), 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

$$H(N,i) = h(S(N,i)) \tag{3.1}$$

  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

$$H(n,i) = \max\left(\frac{qH(n+1,i+1) + (1-q)H(n+1,i)}{1+R}, h(S(n,i))\right) \tag{3.2}$$

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

In particular, $H(0)$ at the root node of the tree is the price of the American option at time 0. Note that the discounted price process $(1+R)^{-n}H(n,i)$ is the **Snell envelope** of the discounted payoff process $(1+R)^{-n}h(S(n,i))$.

We would like to compute and store the price of an American option not only at time 0, but also for each time step n and node i in the binoial tree. In addition, we want to compute the **early exercise policy** for an American option. The time steps n and nodes i at which the option should be exercised are characterised by the condition

$$H(n,i) = h(S(n,i)) > 0.$$

We are going to encode this information as data of type `bool`, taking just two possible values, 0 if the option should not be exercised at a given node, or 1 otherwise, depending on whether the above condition is violated or not.

The natural structure for the price data is that of a lattice indexed by the time steps $n = 0, 1, \ldots, N$ and nodes $i = 0, 1, \ldots, n$.

A convenient way to store the option prices will be a vector indexed by the time variable `n` consisting vectors of type `double` indexed by the nodes `i` at each time `n`. We wrap this vector of vectors in a class called `BinLattice`, and provide related functionality such as setting the number of time steps `N` or setting and retrieving a value at time `n` and node `i`.

- **Extend the Binomial method pricer in order to make it able to price an American option**, according to the explanations here above.

B) Pricing an American option with Black-Scholes and Binomial method

Equivalence between Black-Scholes and Binomial method

The binomial model can be employed to approximate the Black–Scholes model. One of several possible approximation schemes is the following. 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

$$U = e^{(r+\sigma^2/2)h+\sigma\sqrt{h}} - 1,$$
$$D = e^{(r+\sigma^2/2)h-\sigma\sqrt{h}} - 1,$$
$$R = e^{rh} - 1,$$

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

- **Extend the Binomial method pricer in order to make it able to price an American option using the Black-Scholes model through its approximation by the Binomial tree**