TD7

November 2022

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 TD5), 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 $h : \left(\mathbb{R}^+\right)^m \to \mathbb{R}$ 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 Black-Scholes 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:

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

as a Black-Scholes 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

- Augment the *Option* class with *payoffPath* method, taking a *std::vector<double>* as argument, returning $h(S_{t_1}, \cdots, S_{t_m})$.
- The non-overriden version of this function should return $h\left(S_{t_m}\right)$ (calling *payoff(double)*)
- Create a derived class from *Option*: *AsianOption*
  - The constructor takes a *std::vector<double>* as argument, representing $(t_1, \cdots, t_m)$
  - The argument should be stored in a private member, with a getter method *getTimeSteps()*
  - Override *AsianOption::payoffPath(std::vector<double>)* so that

$$h(S_{t_1}, \cdots, S_{t_m}) \quad = \quad h\left(\frac{1}{m}\sum_{k=1}^{m} S_{t_k}\right),$$

    where h on the right hand side is *payoff(double)*. *AsianOption::payoffPath(std::vector<double>)* should **not** be virtual.
- Created *AsianCallOption* and *AsianPutOption*, derived from *AsianOption*.
  - In addition to *std::vector<double>*, their constructor takes a double as argument, defining the strike.
  - They have to have proper implementations of *payoff()*.
- Augment the *Option* class with *bool isAsianOption()*, returning *false* in its non-overriden version, override it in *AsianOption*.
- In *CRRPricer*'s constructor, check if the option is an Asian option, if it is the case, throw an exception.
- Design a singleton class *MT*, encapsulating a *std::mt19937* object. Two public static methods are implemented: *double rand_unif()* and *double rand_norm()*, returning a realization of $\mathcal{U}([0,1])$ and $\mathcal{N}(0,1)$ respectively. Ensure that only one instance of *std::mt19937* can be used in all the program through *MT*.
- Write the *BlackScholesMCPricer* class:
  - The constructor must have signature *(Option* option, double initial_price, double interest_rate, double volatility)*
  - The class should have a private attribute that counts the number of paths already generated **since the beginning of the program** to estimate the price of the option, a getter named *getNbPaths()* needs to give a read access to this attribute.
  - The method *generate(int nb_paths)* generates *nb_paths* new paths of $(S_{t_1}, \cdots, S_{t_m})$ (for European Option, $m = 1$), and **UPDATES** the current estimate of the price of the option (the updating process is the same as in TD2).
  - The operator *()* returns the current estimate (throw an exception if it is undefined).
  - The method *confidenceInterval()* returns the 95% CI of the price, as a *std::vector<double>* containing the lower bound and the upper bound.
  - The random generation have to be handled by calling *MT::rand_norm()*.
  - No path should be stored in the object
  - Check the prices given by *BlackScholesMCPricer* are in line with those given by *BlackScholesPricer* on vanilla options.