Projet C++

Sujet "Finance", Pricer

Jacques Thomas Tien-Thinh

vendredi 12 janvier 2024

Section 1

Introduction

Consignes

[1] Créer un programme pour déterminer le prix d'une option financière à l'aide de l'équation de Black, Scholes et Merton, puis à l'aide de la simulation de Monte Carlo.

Section 2

La formule de de Black-Scholes

La formule

Supposons que r et vol sont constants, alors nous pouvons obtenir à l'aide du lemme d'Îto le prix d'un call européen selon l'équation:

$$C_0 = S_0 N(d_1) - K \exp(-rT) N(d_2)$$

où $d_1 = \frac{\ln(\frac{S_0}{K}) + (r + \frac{vol^2}{2})T}{vol\sqrt{T}}$ et $d_2 = d_1 - vol\sqrt{T}$

et on en déduit le prix d'un put européen à l'aide de la formule de parité call/put:

$$C_0 - P_0 = S_0 - K \exp(-rT)$$

Les Hypothèses

Soit un actif dont le prix est un processus stochastique $S = (St)_t$, les principales hypothèses du modèle sont:

- $(St)_t$ suit un mouvement brownien géométrique d'écart-type *vol*
- pas d'opportunité d'arbitrage
- $(St)_t$ est un processus à temps continu

Structure de code

Création de la classe BlackScholes qui calcule le prix d'une option européenne en fonction de :

- S: prix du sous-jacent à l'instant initial
- vol: volatilité du sous-jacent en années, supposée constante
- r: taux d'intérêt risque neutre en années, supposé constant
- K: strike de l'option
- T: date de maturité de l'option en années

Utilisation de cette classe dans main.cpp

Section 3

La méthode de Monte Carlo

Présentation de la méthode de Monte Carlo

La méthode de Monte Carlo consiste en l'utilisation de la loi forte des grands nombres, si l'on connait la loi du processus stochastique $(St)_t$, alors nous pouvons simuler informatiquement plusieurs trajectoires de ce processus et obtenir le payoff de l'option européenne dans chaque de ces cas, alors en calculant la moyenne, on peut en déduire le prix.

Création d'un graphique en C++

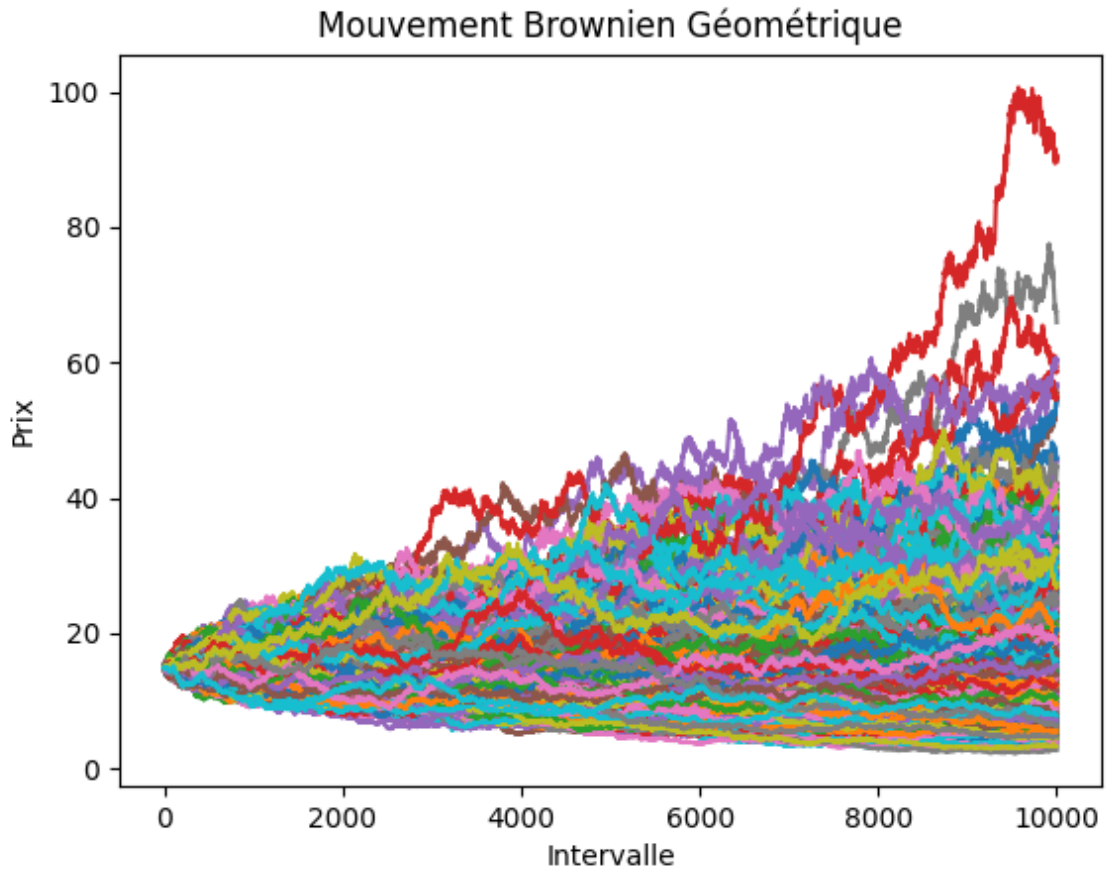

Figure 1: Notre simulation de prix

Difficulés rencontrées

coucou

Section 4

Le pricing d'options complexes

Un code modulable

Structure de code

Notre code repose sur l'utilisation de classe (POO) permettant une grande modularité :
- La classe permet de garder tous les paramètres de simulation - La fonction calculate permet d'effectuer les simulations

Penser le code pour des options plus complexes

$dS_t = (rdt + \sigma dW_t)S_t$ avec $dW_t \sim \mathcal{N}(0, dt)$
On obtient alors $S_{t+\Delta t} = S_t exp(dB_t)$ avec $dB_t \sim \mathcal{N}(rdt, \sigma^2 \Delta t)$

Un code modulable

mainplot.cpp

```
normal_distribution<> d(
    (r - 0.5 * vol * vol) * T/nb_delta_T,
    vol * sqrt(T/nb_delta_T)
); // Génère des valeurs suivant une loi normale

std::vector<double> prices(nb_delta_T+1);
prices[0] = price_path;

for (int j = 0; j < nb_delta_T; ++j) {
    // Divise [0, T] en nb_delta_T sous-intervalles
    price_path *= exp(d(gen));
    prices[j+1] = price_path;
}
```

Difficulés rencontrées

Difficultés

- Unifier les strucutres de classe *BlackScholes* et *MonteCarlo*
- Penser *MonteCarlo* :: *calculate* pour qu'il soit modulable
- Travailler avec les générateurs de nombres aléatoires

Pistes d'amélioration

- Utilisation de la méthode Quasi Monte-Carlo qui permettrait de gagner du temps
- Les méthodes de réduction de la variance
- Implémenter des calculs en parallèle sur carte graphique

Générer la présentation

```
pandoc -t beamer Presentation.md  -V theme:Warsaw -o Preser
```