# Projet : Simulation de Monte-Carlo – Pricing par le modèle de Black-Scholes

## Sommaire



**Baptiste VERMEERSCH**

## 1) Introduction

Dans ce projet, nous allons nous intéresser à la méthode de simulation de Monte-Carlo, une technique numérique incontournable dans le domaine de la finance de marché. Cette approche repose sur l'utilisation de tirages aléatoires pour approximer la valeur d'actifs ou de produits dérivés dont la valorisation analytique est complexe, voire inexistante. En pratique, elle consiste à simuler un grand nombre de trajectoires possibles de l'évolution d'un actif financier, puis à exploiter ces trajectoires pour estimer le prix d'un produit dérivé.

La méthode de Monte-Carlo est particulièrement précieuse lorsqu'il s'agit de pricer des options exotiques ou de traiter des produits financiers aux structures de paiement complexes. Dans ces cas, les formules fermées n'existent pas, ou bien elles sont trop lourdes à manipuler, ce qui rend l'approche numérique plus adaptée. Son avantage principal réside dans sa adaptabilité : elle peut être appliquée à un ensemble varié de modèles stochastiques et de produits financiers, à condition de disposer d'une puissance de calcul suffisante.

Dans le cadre de ce projet, nous limiterons notre analyse à un cas plus simple mais tout aussi instructif : l'option européenne. Pour ce produit, le modèle de Black-Scholes fournit une solution analytique bien connue et largement utilisée comme référence en finance quantitative. L'intérêt de l'exercice est double : d'une part, mettre en œuvre concrètement la méthode de Monte-Carlo pour simuler les trajectoires de l'actif sous-jacent et calculer le prix de l'option ; d'autre part, comparer ce prix simulé au prix théorique donné par la formule fermée de Black-Scholes. Cette comparaison servira de validation de notre approche numérique, tout en illustrant les atouts et les limites de la simulation Monte-Carlo.

## 2) Monte-Carlo

Commençons par importer les packages nécessaires

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Le modèle de Black-Scholes suppose que la dynamique suivie par l'actif sous-jacent est la suivante :

$$dS_t = \mu S_t d_t + \sigma dW_t$$

- $S_t$ : prix de l'actif sous-jacent à l'instant $t$
- $\mu$ : drift (taux de croissance instantané)
- $\sigma$ : volatilité
- $W_t$ : mouvement brownien standard

Fixons de façon arbitraire les paramètres dont nous avons besoin

In [20]:
```
S0 = 100  # Prix du sous-jacent à t = 0
K = 100  # Strike (prix d'exercice)
sigma = 0.2  # Volatilité
r = 0.05  # Taux sans risque
T = 1  # Maturité (en année)
N = 252  # Pas de temps, ici : discrétisation journalière (jours bourse)
M = 10000  # Nombre de trajectoire à simuler
```

Nous avons maintenant besoin de discrétiser le processus continu afin de le simuler numériquement :

In [21]:
```
dt = T / N
```

Nous aurons alors une discrétisation journalière sur $1$ an.

On crée ensuite un vecteur temps allant de $0$ à $T$ avec $N$ intervalles et donc, $N+1$ points :

In [22]:
```
t = np.linspace(0, T, N + 1)
```

Puis, nous stockons les trajectoires simulées dans une matrice initialisée de zéros. chaque ligne représentera une trajectoire de prix et chaque colonnes un instant $t$ (pas de temps) :

In [23]:
```
S = np.zeros((M, N + 1))
```

On pose ensuite une condition initiale, nous voulons que toutes les trajectoires de prix commencent à $S_0$ :

In [24]:
```
S[:,0] = S0
```

Nous pouvons à présent, créer une boucle qui va simuler $M$ trajectoires de prix de l'actif sous-jacent :

In [25]:
```
for i in range(1, N + 1):
    Z = np.random.randn(M)
    S[:,i] = S[:,i-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) *
```

Avec :

$$Z \sim \mathcal{N}(0,1)$$

La formule suivante :

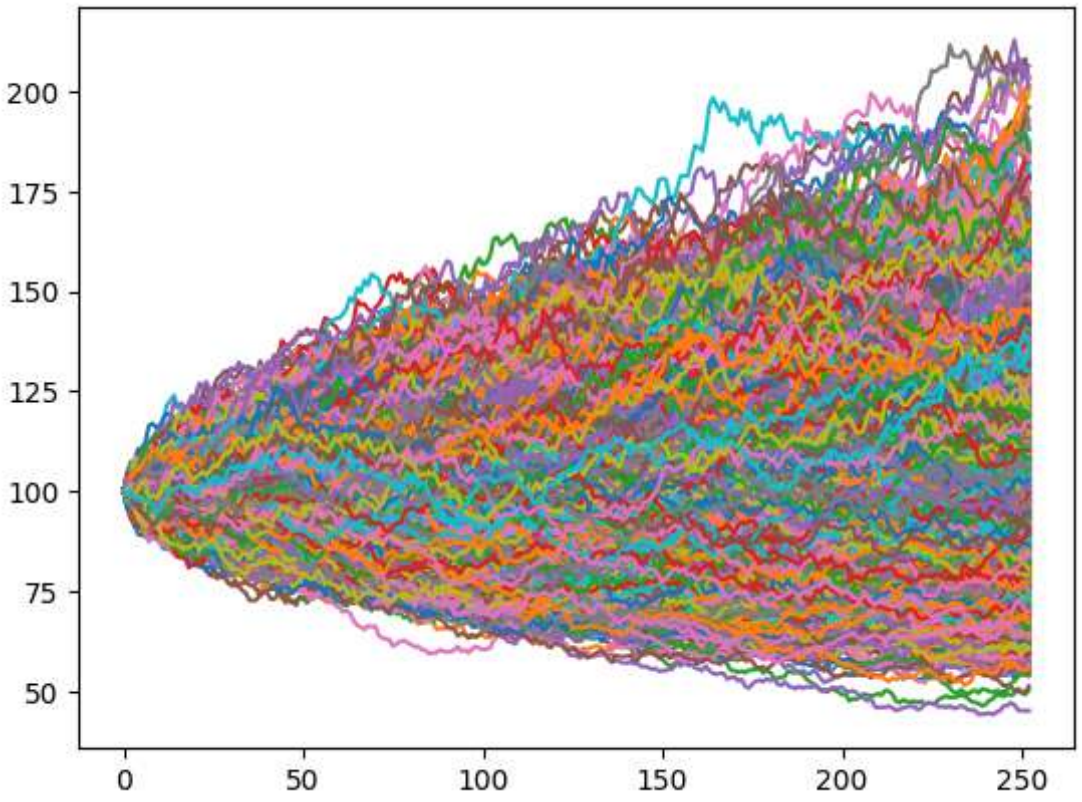
$$S_t = S_0 \exp\left((\mu - \tfrac{1}{2}\sigma^2)t + \sigma W_t\right)$$

est la solution exacte discrétisée, directement issue de l'application du Lemme d'Ito (qu'on ne détaillera pas dans ce projet) à l'équation différencielle stochastique (EDS) de

Black-Scholes.

Nous avons maintenant nos $M$ trajectoires de prix stochées dans une matrice, et pouvons donc très facilement les afficher sur un graphique à l'aide du code suivant :

In [26]:
```
for j in range(M):
    plt.plot(S[j])
```




L'objectif est de calculer le prix de cette option, sur la base de ces simulations. Le code suivant permet de pricer le call en calculant la moyenne des payoffs et en l'actualisant au taux sans risque :

In [27]:
```
ST = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)
payoff = np.maximum(ST - K, 0)
price = np.exp(-r * T) * np.mean(payoff)

print(price)
```

```
10.804515798799155
```

On obtient un prix égal à $10.321$

Afin vérifier la qualité de cette estimation, nous allons procéder au pricing de cette même option, à l'aide de la formule exacte du modèle de Black-Scholes.

## 3) Black-Scholes

Comme indiqué précédemment, le modèle de Black-Scholes nous donne une formule exacte pour pricer une option vanille, comme le call européen considéré ici. Le calcule du prix théorique de ce call se fait donc en appliquant la formule suivante :

$$C(S_0, K, T) = S_0\, N(d_1) - K e^{-rT}\, N(d_2)$$

avec

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right) T}{\sigma\sqrt{T}}, \qquad d_2 = d_1 - \sigma\sqrt{T}$$

- $S_0$ : prix de l'actif sous-jacent à l'instant $0$
- $K$ : strick (prix d'exercice)
- $r$ : taux d'intérêt sans risque
- $T$ : maturité de l'option (en année)
- $\sigma$ : volatilité
- $N$ : fonction de répartition de la loi normale centrée réduite

En appliquant cette formule du call à nos données, cela donne :

In [18]:
```
from scipy.stats import norm

d1 = (np.log(S0/K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
d2 = d1 - sigma * np.sqrt(T)

Nd1 = norm.cdf(d1)
Nd2 = norm.cdf(d2)

Price = S0 * Nd1 - K * np.exp(-r * T) * Nd2

print(Price)
```

```
10.450583572185565
```

Nous obtenons un prix de $10.45$

## 4) Conclusion

Les résultats nous montrent que notre simulation est cohérente et fonctionne bien. Même avec un nombre de simulations fixé à M=10 000, ce qui reste assez faible, le prix estimé par la méthode de Monte-Carlo se rapproche déjà de celui calculé avec la formule exacte de Black-Scholes.

Cependant, il faut garder en tête que la précision dépend fortement du nombre de simulations. Plus on en fait, plus le résultat converge vers la valeur exacte. En contrepartie, cela demande davantage de temps et de puissance de calcul.

En résumé, la méthode de Monte-Carlo est un outil simple mais extrêmement puissant. Elle permet d'obtenir une bonne approximation de la valeur de produits financiers, y compris lorsqu'on n'a pas de formule toute faite. Dans notre cas, elle a permis de valider son efficacité en la comparant à la formule de Black-Scholes. Son principal atout reste sa flexibilité, mais sa limite est la lourdeur des calculs nécessaires lorsque l'on veut atteindre une très grande précision.

Pour améliorer l'efficacité des simulations, plusieurs approches existent. On peut par exemple utiliser des méthodes de réduction de variance (antithetic variates...), qui permettent d'obtenir de meilleurs résultats sans multiplier le nombre de trajectoires. Une autre solution est de recourir à des séquences quasi-aléatoires comme les suites de Sobol, qui accélèrent la convergence par rapport aux tirages aléatoires classiques. Enfin, les progrès en calcul parallèle (GPU, clusters, cloud computing) offrent également la possibilité d'exécuter un grand nombre de simulations en un temps réduit.

Ainsi, la méthode de Monte-Carlo reste incontournable pour le pricing des produits financiers complexes : ses performances peuvent être améliorées à la fois par des raffinements mathématiques et par l'exploitation des nouvelles technologies de calcul.