Projet de C++
Finance
Sujet Pricing

Thomas ROUSSAUX, Tien-Thinh TRAN-THUONG et Jacques ZHANG

1- Modèle de Black, Scholes et Merton

Modèle de Black, Scholes et Merton

$$dS_t = (rdt + \sigma dW_t)S_t$$

où

St est le prix du sous jacent est le prix du sous-jacent à t

r est le taux d'intérêt sans risque

vol est la volatilité de l'option

dWt est un incrément de mouvement brownien

Prix d'un call et d'un put européen:

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

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

Et on en déduit le prix d'un put avec:

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

Implémentation de Black Scholes sur C++

```cpp
class BlackScholes {
public:
    BlackScholes(double vol, double S, double K, double r, double T);
    double calculate(bool is_call);
protected:
    double vol; // Volatilité de l'option
    double S; // Prix actuel de l'option
    double r; // Taux d'intérêt de l'option
    double K; // Prix d'exercice de l'option
    double T; // Échéance de l'option en années
    double standard_normal_cdf(double x);
    bool is_call;
};
```

Difficulté rencontrée

- Trouver une méthode pour approximer la fonction de répartition de la loi normale sur C++

$$\mathcal{N}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}u^2} du$$

```
double BlackScholes::standard_normal_cdf(double x) {
    return 0.5 *  erfc(-x / sqrt(2.0));
}
```

2- Méthode de Monte Carlo

Méthode de Monte Carlo

Une méthode algorithmique visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes.

$$dS_t = (rdt + \sigma dW_t)S_t \text{ avec } dW_t \sim \mathcal{N}(0, dt)$$

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

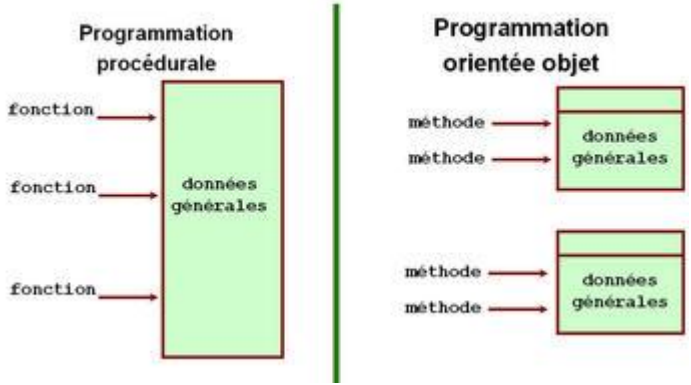
Un code modulable

```
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;
}
```

Un code modulable

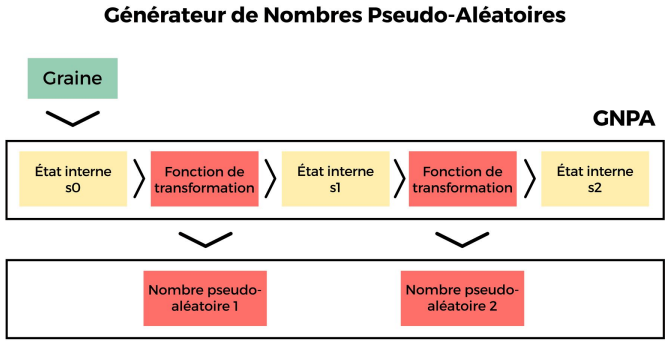
Notre code repose sur l'utilisation de classe (**POO**) permettant une grande modularité :

- La classe permet de garder tous les **paramètres** de simulation
- La fonction calculate permet d'effectuer les **simulations**



Difficultés

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



3- Création d’un graphique en C++

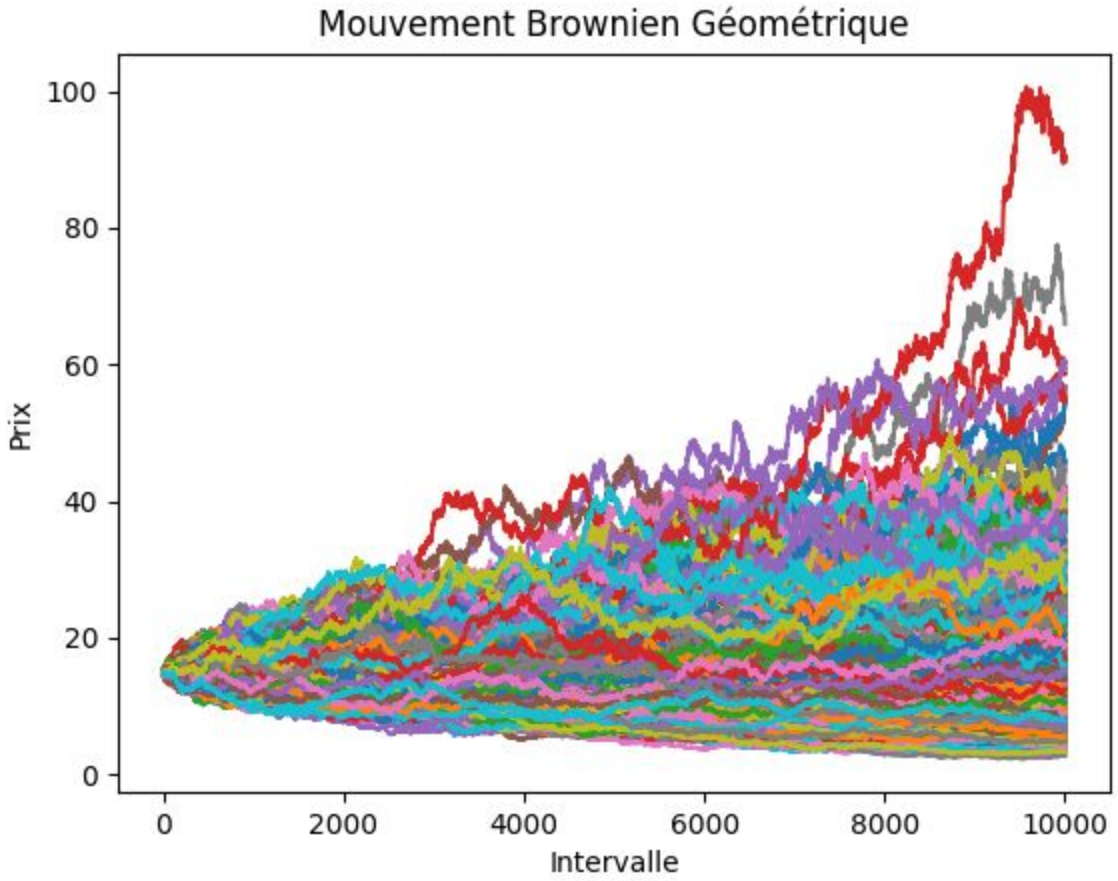
Création d’un graphique en C++

Idée: utiliser la librairie python matplotlib pour tracer les différentes trajectoires du processus

Etapes

- Installer python
- Installer matplotlib et numpy
- Copier le header matplotlib pour C++ et l'ajouter dans un fichier matplotlib.h
- Trouver le chemin des différentes librairies pour les rajouter aux variables environnementales
- Compiler le code à l'aide de g++

```
g++ mainplot.cpp -o mainplot src/MonteCarlo.cpp
-I "C:\Program Files\Python312\include"
-I "C:\users\thoma\appdata\roaming\python\python312\site-packages\numpy\core\include"
-L  "C:\Program Files\Python312\libs" -lpython312
```


Mouvement Brownien Géométrique
Prix
Intervalle
0
20
40
60
80
100
0
2000
4000
6000
8000
10000

difficultés rencontrées

_ mettre à jour g++

_ trouver les chemins des différentes librairies

_ trouver le fichier python.h

Conclusion

1. Pricer par formule de Black-Scholes
2. Pricer par méthode de Monte Carlo
3. Visualisation de la simulation Monte Carlo

Pistes d'optimisation de Monte Carlo:

- Utilisation de la méthode Quasi Monte-Carlo
- Les méthodes de réduction de la variance
- Implémenter des calculs en parallèle sur GPU