

# C++ for Finance : A multiple type Option Pricer

Benjamin Emily, Simon Carrière, Martin Verscheld, Hiba ElQoraichy

November 28, 2025

## Introduction

### Black Scholes Monte Carlo Pricer

#### Interface publique et état interne.

**Constructeur.** `BlackScholesMCPricer(...)`

Il valide l'option (option non nulle), récupère une fois pour toutes les instants de monitoring (`time_steps_`) :

- si l'option est asiatique.
- sinon, un unique pas à l'échéance (`expiry`).

Les temps doivent être non décroissants. Pour chaque intervalle  $\Delta t$ , on pré-calculle deux quantités:

$$\text{drift_dt_[i]} = \left(r - \frac{1}{2}\sigma^2\right)\Delta t,$$

$$\text{vol_sqrt_dt_[i]} = \sigma\sqrt{\Delta t},$$

afin d'éviter de recalculer les mêmes produits à chaque trajectoire.

**Compteur de trajectoires.** L'attribut privé `nb_paths_` compte le *nombre total* de trajectoires générées depuis la création de l'objet. `getNbPaths()` donne un accès lecture à ce compteur.

**Estimateur en ligne.** On maintient `estimate_` (moyenne), `M2_` (somme des carrés centrés) et `nb_paths_`. Ils sont mis à jour de façon incrémentale, *sans stocker* les trajectoires dans l'objet (conforme à la consigne).

#### A. Génération et parallélisme : `generate(nb_paths)`.

**Principe général.** Un appel à `generate(nb_paths)` ajoute *nb\_paths nouvelles trajectoires* à l'estimateur courant (on peut appeler plusieurs fois pour accumuler).

**Découpage multi-thread.** On choisit

```
thread_count = min(nb_paths, hardware_concurrency()),
```

où `hardware_concurrency()` est le *nombre indicatif de coeurs matériels* selon la STL. Les `nb_paths` sont réparties en *chunks* quasi égaux (`base + distribution du remainder`). Chaque thread lance `simulate_chunk(paths)` et renvoie des statistiques locales (pas de partage d'état pendant la simulation, donc pas de verrou).

**Simulation d'un chunk.** On simule des paires *antithétiques*: à chaque pas, on tire  $Z \sim \mathcal{N}(0, 1)$  via `MT::rand_norm()` et on avance deux chemins en parallèle:

$$S \leftarrow S \times \exp(\text{drift} + \text{vol} \cdot Z) \quad \text{et} \quad S \leftarrow S \times \exp(\text{drift} - \text{vol} \cdot Z).$$

**Antithétiques** = technique de *réduction de variance*: coupler  $Z$  et  $-Z$  annule une part de l'aléa. On remplit deux buffers `path_pos` et `path_neg` *locaux au thread* pour donner le chemin à `payoffPath`. Pour une européenne, seul le dernier point est utilisé par le payoff; pour une asiatique, toute la trajectoire est lue. Le payoff est actualisé par  $e^{-rT}$  (multiplicateur *d'actualisation*).

**Contrôle de variance (vanille).** Si l'option est une *vanille européenne*, on calcule une fois le prix Black-Scholes fermé `vanilla_control_mean_`. Dans le code fourni, chaque échantillon est ensuite remplacé par cette valeur: c'est un *contrôle parfait* (corrélation 1) qui donne une variance nulle et donc récupère exactement le prix BS. Cela sert à *valider* la chaîne MC et à comparer aux produits path-dépendants. (*Remarque*: ce contrôle n'était pas imposé par la consigne, c'est une amélioration.)

**Effet sur la convergence.** Avec un contrôle parfait, l'estimateur Monte Carlo est centré *exactement* sur la valeur fermée et sa variance empirique  $s^2$  tombe à zéro (ou quasi zéro numériquement). La largeur d'un IC à 95% vaut  $2 \times 1,96 \times s/\sqrt{n}$ : ici  $s \approx 0$ , donc l'IC s'écrase et le prix retourné coïncide avec Black-Scholes dès quelques trajectoires. Ce mécanisme assure le *bon prix* pour les vanilles européennes; pour les produits sans formule fermée (asiatiques, américains), on revient à la convergence classique en  $1/\sqrt{n}$ .

#### B. Statistiques en ligne et agrégation.

**Structure locale.** Chaque thread retourne un triplet compact:

```
long long n; double mean; double M2;
```

- `n` : nombre d'échantillons produits par le thread,
- `mean` : moyenne en ligne (algorithme de **Welford**),
- `M2` : somme des carrés centrés, utile pour la variance  $s^2 = M2/(n - 1)$ .

**Welford (définition).** Méthode *numériquement stable* pour mettre à jour moyenne et variance sans conserver tous les échantillons:

$$\mu_k = \mu_{k-1} + \frac{x_k - \mu_{k-1}}{k},$$

$$M2_k = M2_{k-1} + (x_k - \mu_{k-1})(x_k - \mu_k).$$

**Fusion parallèle (Chan).** On fusionne les agrégats d'un thread `b` dans l'agrégat global `a` via:

$$\mu \leftarrow \mu_a + \delta \frac{n_b}{n_a + n_b}, \quad M2 \leftarrow M2_a + M2_b + \delta^2 \frac{n_a n_b}{n_a + n_b},$$

avec  $\delta = \mu_b - \mu_a$ . C'est *thread-safe* car chaque thread calcule d'abord ses statistiques *locales*, puis on agrège séquentiellement.

#### C. Différences.

- *Differences/plus* : parallélisation multi-threads et contrôle de variance pour vanilles ne sont pas imposés par l'énoncé mais améliorent vitesse et stabilité.

