Construction d'un pricer d'options panier

Rémi Schmitt et Théo Verdelhan

Janvier 2026

1 Données de marché

1.1 Courbe de taux sans risque

Élément	Valeur
Référence	€STR (Euro Short-Term Rate)
Source	BCE
Date de pricing	23 janvier 2026
Taux overnight	**1.933%**

Remarque. €STR est un taux overnight. Pour des maturités ≤ 1 an et un environnement de taux stable, l'approximation par taux constant est acceptable.

1.2 Volatilités

Sous H1, les volatilités sont estimées à partir d'historiques de prix (vol. réalisée). Sous H2, les volatilités déterministes $\sigma_i(t)$ sont définies par interpolation linéaire entre points de maturité (proxy d'une calibration implicite).

1.3 Composition du panier

On considère n actifs ($1 \leq n \leq 10$) avec poids (a_i) normalisés ($\sum a_i = 1$). La matrice de corrélation (ρ_{ij}) est supposée symétrique et valide.

2 Théorie et implémentation technique

2.1 Cadre général

Panier pondéré et payoff :

$$A(t) = \sum_{i=1}^{n} a_i S_i(t), \qquad \Pi(T) = \begin{cases} (A(T) - K)^+ & \text{Call} \\ (K - A(T))^+ & \text{Put.} \end{cases}$$

Sous Q :

$$V_0 = E^Q\left[\exp\left(-\int_0^T r(s)\,ds\right)\Pi(T)\right], \qquad dW_i\,dW_j = \rho_{ij}\,dt.$$

Comme $A(T)$ est une somme de lognormales corrélées, on utilise : (i) *Moment Matching* (approximation rapide), (ii) Monte Carlo H2 (référence numérique).

2.2 Moment Matching : principe

On approxime $A(T)$ par une lognormale $\bar{A}(T)$ calibrée sur les deux premiers moments :

$$E[\bar{A}(T)] = M_1, \qquad E[\bar{A}(T)^2] = M_2,$$

d'où

$$\hat{\sigma}^2 = \frac{1}{T}\ln\left(\frac{M_2}{M_1^2}\right), \qquad d_1 = \frac{\ln(M_1/K) + \frac{1}{2}\hat{\sigma}^2 T}{\hat{\sigma}\sqrt{T}}, \quad d_2 = d_1 - \hat{\sigma}\sqrt{T}.$$

Prix type Black :

$$V_0 = P(0,T)\begin{cases} M_1 N(d_1) - K N(d_2) & \text{Call} \\ K N(-d_2) - M_1 N(-d_1) & \text{Put.} \end{cases}$$

Point numérique. Si $M_2 \leq M_1^2$ (arrondis), on force $M_2 > M_1^2$ (epsilon safeguard) pour éviter $\hat{\sigma}^2 \leq 0$.

2.2.1 Représentation des actifs et du panier sous H1

Chaque actif est représenté par la classe `Stock` comprenant les attributs suivants: `SpotPrice` : prix initial $S_i(0)$,`Volatility` : volatilité constante σ_i,`DividendRate` : taux de dividende continu q_i.
Les paniers sont représentés par `Basket`.Chaque panier contient : la liste des actifs,le vecteur de poids (a_i),la matrice de corrélation (ρ_{ij}),le taux sans risque.

Choix technique : Les contrôles suivants sont implémentés: somme des poids égaux à 1, dimensions cohérentes de la matrice de corrélation, symétrie de la matrice et $\rho_{ii} = 1$, $\rho_{ij} \in [-1, 1]$.
Ce choix correspond exactement au cadre Black–Scholes multi-actifs.

2.3 Pricer Moment Matching H1 (paramètres constants)

Représentation des actifs et du panier sous H1 Chaque actif est représenté par la classe `Stock` comprenant les attributs suivants: `SpotPrice` : prix initial $S_i(0)$,`Volatility` : volatilité constante σ_i,`DividendRate` : taux de dividende continu q_i.
Les paniers sont représentés par `Basket`.Chaque panier contient : la liste des actifs,le vecteur de poids (a_i),la matrice de corrélation (ρ_{ij}),le taux sans risque.

Choix technique : Les contrôles suivants sont implémentés: somme des poids égaux à 1, dimensions cohérentes de la matrice de corrélation, symétrie de la matrice et $\rho_{ii} = 1$, $\rho_{ij} \in [-1, 1]$.
Ce choix correspond exactement au cadre Black–Scholes multi-actifs.

Modèle.

$$dS_i(t) = (r - q_i)S_i(t)\,dt + \sigma_i S_i(t)\,dW_i(t).$$

Moments.

$$F_i(0,T) = S_i(0)e^{(r-q_i)T}, \qquad M_1 = \sum_i a_i F_i(0,T),$$

$$M_2 = \sum_{i,j} a_i a_j F_i(0,T) F_j(0,T) \exp\big(\rho_{ij}\sigma_i\sigma_j T\big).$$

Lien avec le code. La classe `MomentMatchingPricer` implémente l'approximation de Brigo et al. sous H1. Cela correspond à la méthode `CalculateFirstMoment()`, qui somme les forwards pondérés. M_2 est implémentée dans `CalculateSecondMoment()` via une double boucle et un terme de covariance exponentiel.

2.4 Pricer Moment Matching H2 (taux et volatilités déterministes)

Actifs en H2 : volatilités déterministes En H2, les actifs sont représentés par la classe `StockH2` aux caractéristiques suivantes:`SpotPrice` : $S_i(0)$, `VolatilityModel` : objet de type `DeterministicVolatilityModel`,`DividendRate` : q_i (constant).
Les paniers sont représentés par `BasketH2`.Chaque panier contient :la liste des actifs, le vecteur de poids (a_i), la matrice de corrélation (ρ_{ij}), le taux sans risque (H1) ou le modèle de taux (H2).

Choix technique : Les contrôles suivants sont implémentés: somme des poids égaux à 1, dimensions cohérentes de la matrice de corrélation, symétrie de la matrice et $\rho_{ii} = 1$, $\rho_{ij} \in [-1, 1]$.

La volatilité $\sigma_i(t)$ est définie par une courbe interpolée linéairement à partir de points (t, σ) :

$$\sigma_i(t) = \text{InterpLin}\big((t_k, \sigma_k)\big).$$

Deux méthodes essentielles sont fournies : `GetVolatility(t)` : retourne $\sigma_i(t)$ par interpolation, `IntegrateVariance(T)` : calcule $\int_0^T \sigma_i(t)^2\,dt$ par la règle des trapèzes.

Choix technique : L'interpolation linéaire a été retenue pour sa stabilité numérique, sa simplicité d'implémentation et sa cohérence avec l'approximation en volatilité constante par morceaux utilisée dans la littérature (Brigo et al.).

Modèle.

$$dS_i(t) = (r(t) - q_i)S_i(t)\,dt + \sigma_i(t)S_i(t)\,dW_i(t).$$

Moments.

$$R(0,T) = \int_0^T r(s)\,ds, \qquad P(0,T) = e^{-R(0,T)}, \qquad F_i(0,T) = S_i(0)e^{R(0,T)-q_iT},$$

$$M_1 = \sum_i a_i F_i(0,T), \qquad M_2 = \sum_{i,j} a_i a_j F_i F_j \exp\left(\rho_{ij}\int_0^T \sigma_i(t)\sigma_j(t)\,dt\right).$$

Dans le code , $R(0,T)$ et $C_{ij}(0,T)$ sont évaluées numériquement (par la règle des trapèzes) à partir des courbes déterministes $r(\cdot)$ et $\sigma_i(\cdot)$. Ainsi, $R(0,T)$ est calculé dans la méthode `IntegrateRate()` dans la classe `DeterministicRateModel()`; $C_{ij}(0,T)$ est calculé dans la méthode `IntegrateVariance()` dans la classe `DeterministicVolatili`

2.5 Pricer Monte Carlo H2 et réduction de variance

La classe `MonteCarloPricerH2` fournit une estimation numérique de référence du prix de l'option sous l'hypothèse H2 (taux $r(t)$ et volatilités $\sigma_i(t)$ déterministes). Elle renvoie un objet `MonteCarloResultH2` contenant le prix estimé, la variance et l'erreur standard, ainsi que (le cas échéant) des informations liées à la réduction de variance (`ControlVariateAdjustment`, `VarianceReduction`).

Corrélations (Cholesky).

$$\mathbf{Z}_c = L\mathbf{Z}, \qquad LL^\top = \rho, \qquad \mathbf{Z} \sim \mathcal{N}(0, I).$$

Dans le code, cette étape correspond à : `MathUtils.CholeskyDecomposition(basket.CorrelationMatrix)` pour obtenir L, `GenerateCorrelatedRandomNumbers(choleskyMatrix, numAssets)` pour calculer $\mathbf{Z}_c$ via multiplication triangulaire. **Choix technique :** la décomposition de Cholesky est la méthode standard pour corréler des normales ; elle est simple, robuste et efficace pour des paniers de taille modérée.

Schéma (log-Euler exponentiel) et discrétisation.

$$S_i(t + \Delta t) = S_i(t)\exp\Big((r(t) - q_i - \tfrac{1}{2}\sigma_i(t)^2)\Delta t + \sigma_i(t)\sqrt{\Delta t}\,Z_i\Big).$$

Dans le code, l'évolution est implémentée dans `SimulatePaths()` :pas temporel : `numSteps = max(252, (int)(maturity*365))` soit au moins 252 pas par an (convention de marché) et au moins un pas par jour ; $\Delta t = T/$`numSteps`, `rate = RateModel.GetRate(t)` et `volatility = VolatilityModel.GetVolatility(t)` ; mise à jour :

$$S \leftarrow S\exp\Big((r - q)\Delta t - \tfrac{1}{2}\sigma^2\Delta t + \sigma\sqrt{\Delta t}\,Z\Big).$$

Estimateur et incertitude.

$$\widehat{V}_0 = \frac{1}{N}\sum_{k=1}^{N} X^{(k)}, \quad X^{(k)} = P(0,T)\Pi^{(k)}(T), \quad \mathrm{SE} = \sqrt{\widehat{\mathrm{Var}}(X)/N}.$$

Dans le code, ceci correspond à l'accumulation de `sum` et `sumSquared` dans `Price()`, puis au calcul :

`price = sum/N, variance = sumSquared/N - price2, standardError = sqrt(variance/N).`

Ces statistiques sont stockées dans `MonteCarloResultH2`.

Réduction de variance par variable de contrôle (moyenne géométrique) Afin d'améliorer la précision, une option sur la moyenne géométrique du panier est utilisée comme variable de contrôle. On définit :

$$G(T) = \prod_{i=1}^{n} S_i(T)^{a_i}, \qquad Y = \begin{cases} (G(T)-K)^+ & \text{Call,} \\ (K-G(T))^+ & \text{Put.} \end{cases}$$

Dans le code, $G(T)$ et Y sont calculés par `CalculateControlVariate()` :

$$\texttt{geometricMean} = \prod_i \big(S_i(T)\big)^{a_i}.$$

La théorie du control variate propose l'estimateur :

$$\widehat{V}_0^{CV} = \widehat{V}_0 - \beta\big(\bar{Y} - E[Y]\big), \qquad \beta^\star = \frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}(Y)}.$$

Dans l'implémentation, β est estimé empiriquement à partir des sommes `controlVariateSum`, `controlVariateSumSquared` et `covarianceSum`, puis appliqué dans `ApplyControlVariateReduction()` via :

$$\beta = \frac{\widehat{\mathrm{Cov}}(X,Y)}{\widehat{\mathrm{Var}}(Y)}.$$

Spécificité de l'implémentation. Dans le code actuel, l'ajustement est de la forme

$$\widehat{V}_0^{CV} = \widehat{V}_0 - \beta\,\bar{Y},$$

c'est-à-dire sans injection explicite de $E[Y]$. Cette version peut réduire la variance lorsque X et Y sont fortement corrélés, mais elle n'est pas exactement la version canonique centrée sur $E[Y]$ (et peut donc introduire un biais si $E[Y] \neq 0$). Ce point est cohérent avec la remarque d'implémentation et doit être mentionné dans le rapport comme une approximation pratique.

3 Résultats numériques et paramètres détaillés

Cette section présente les résultats obtenus avec l'exécutable `dotnet run` (menu : démonstration automatique, mode interactif, tests unitaires et fonctionnels). Les valeurs reportées ci-dessous correspondent directement aux sorties console de l'application. **Note :** Tous les calculs utilisent le taux sans risque €STR = 1.933% (source : BCE, 23/01/2026).

3.1 Démonstration automatique : Moment Matching (H1) et Moment Matching (H2)

3.1.1 Cas H1 : paramètres constants

Paramètres. Le panier (2 actifs) et les paramètres de marché utilisés dans la démonstration H1 sont :Valeur initiale du panier : $A_0 = 102.00$ €, Maturité : $T = 1$ an, Taux sans risque constant : $r = 1.933\%$ (€STR BCE 23/01/2026), Strikes : $K_{\text{call}} = 107.10$ € (105% de A_0), $K_{\text{put}} = 96.90$ € (95% de A_0)

Résultats (Moment Matching H1). Les prix obtenus par `MomentMatchingPricer` sont :

Méthode	Call	Put
Moment Matching (H1)	4.8888 €	3.7491 €

Validation en dimension 1 (comparaison Black–Scholes). Dans le cas limite d'un panier à 1 actif, l'algorithme Moment Matching se rapproche de la formule Black–Scholes. La console affiche :

Méthode	Prix
Moment Matching (cas 1 actif)	8.266336 €
Black–Scholes	8.433327 €
Différence	1.669911×10^{-1}

3.1.2 Cas H2 : paramètres déterministes

Paramètres. La démonstration H2 illustre l'impact de $r(t)$ et $\sigma_i(t)$ non constants : Courbe de taux €STR (BCE 23/01/2026) :

$$r(0) = r(0.5) = r(1) = 1.933\%.$$

, Volatilités déterministes (interpolation linéaire entre $t = 0$ et $t = 1$) :

$$\sigma_A(0) = 20.0\%,\ \sigma_A(1) = 22.0\%, \qquad \sigma_B(0) = 18.0\%,\ \sigma_B(1) = 28.0\%.$$

Valeur initiale du panier : $A_0 = 108.00$ €, Strikes : $K_{\text{call}} = 110.00$ €, $K_{\text{put}} = 105.00$ €, Maturité : $T = 1$ an

Résultats (Moment Matching H2). Les prix obtenus par `MomentMatchingPricerH2` sont :

Méthode	Call	Put
Moment Matching (H2)	7.6120 €	5.8318 €

Test de convergence H2 → H1 (paramètres constants). Lorsque les courbes déterministes sont plates (taux et volatilité constants), la théorie impose que H2 se réduise à H1. La console confirme :

Prix H1	Prix H2	Erreur relative
6.172548 €	6.172548 €	0.0000%

3.1.3 Monte Carlo H2 et réduction de variance (démonstration automatique)

La démonstration automatique inclut également une comparaison Monte Carlo (H2) avec et sans réduction de variance :

Méthode	Prix	Écart-type estimateur (σ)
MC standard	7.5827 €	0.0568
MC avec variable de contrôle	7.3949 €	0.0066

La réduction de variance reportée est de **98.6%**. Ceci illustre le gain apporté par l'utilisation d'une variable de contrôle basée sur la moyenne géométrique du panier (cf. implémentation `CalculateControlVariate` et ajustement `ApplyControlVariateReduction`).

4 Annexe

4.1 Validation logicielle : tests unitaires et tests fonctionnels

Afin d'assurer la fiabilité des résultats numériques et la cohérence globale de l'implémentation, le projet intègre deux niveaux complémentaires de validation :

- **tests unitaires** (`UnitTests`) : vérification de briques élémentaires (fonctions mathématiques, construction des objets, cohérence locale du pricing)
- **tests fonctionnels** (`FunctionalTests`) : scénarios end-to-end reproduisant des cas d'usage représentatifs et vérifiant des propriétés économiques (sensibilités, cohérences entre méthodes, convergence H1/H2).

Ces tests sont exécutés via des méthodes statiques `RunAllTests()` qui parcourent un dictionnaire de fonctions booléennes `Func<bool>` et produisent un rapport lisible (succès/échec) en console.

4.1.1 Tests unitaires (`UnitTests`)

Les tests unitaires se concentrent sur la validité des composants fondamentaux.

Vérification de la CDF normale. Le test `TestNormalCdf` contrôle des valeurs de référence :

$$\mathcal{N}(0) = 0.5, \qquad \mathcal{N}(-1.96) \approx 0.025, \qquad \mathcal{N}(1.96) \approx 0.975,$$

avec des tolérances numériques compatibles avec l'approximation implémentée dans `MathUtils.NormalCdf`.

Vérification de Black–Scholes et de la parité put-call. Le test `TestBlackScholes` valide `MathUtils.BlackScholesPrice` en contrôlant la parité put-call :

$$C - P = S - Ke^{-rT}.$$

Tests de construction des objets `Stock` et `Basket`. `TestStockConstruction` vérifie l'intégrité des paramètres injectés (nom, spot, volatilité, dividende). `TestBasketConstruction` vérifie la valeur initiale du panier :

$$A_0 = \sum_{i=1}^{n} a_i S_i(0),$$

et la cohérence avec `Basket.GetBasketValue()`.

Test de cohérence du pricer Moment Matching. `TestMomentMatchingPricer` teste un cas $n = 1$ (option sur un seul actif). Le test vérifie des bornes naturelles :

- prix strictement positif,
- pour un call : $C < A_0$ (borne simple),
- pour un put ITM : $P < K$ (borne simple).

Validation des modèles déterministes H2. `TestH2Models` valide :

- **l'interpolation linéaire** du taux instantané $r(t)$ dans `DeterministicRateModel` :

$$r(0.5) = \frac{r(0) + r(1)}{2},$$

- **l'interpolation linéaire** de la volatilité $\sigma(t)$ dans `DeterministicVolatilityModel`,
- la construction de `StockH2`.

Cohérence de monotonicité en strike. `TestPricingConsistency` vérifie un principe économique : pour un call européen de même maturité,

$$K_{\text{ITM}} < K_{\text{ATM}} < K_{\text{OTM}} \quad \Rightarrow \quad C(K_{\text{ITM}}) > C(K_{\text{ATM}}) > C(K_{\text{OTM}}) > 0.$$

Cela permet de détecter des erreurs de signe dans les d_1, d_2, ou une mauvaise actualisation.

4.1.2 Tests fonctionnels (`FunctionalTests`)

Les tests fonctionnels couvrent des scénarios réalistes et des propriétés globales du système.

Scénario 1 : panier 2 actifs ATM. `TestTwoAssetBasketATM` vérifie :

- prix call et put strictement positifs,
- cohérence relative (avec $r > 0$, le call ATM tend à être supérieur au put ATM dans ce cadre),
- bornes simples ($C < A_0$),
- écart raisonnable entre call et put (contrôle de plausibilité).

Scénario 2 : panier 3 actifs diversifié. `TestThreeAssetDiversified` construit un panier de trois secteurs avec volatilités et dividendes différents, puis teste un call **OTM**. Le prix est contrôlé par une borne de plausibilité (prix inférieur à une fraction du spot panier), afin de détecter les erreurs d'ordre de grandeur.

Scénario 3 : convergence H2 vers H1. `TestH1H2Convergence` impose des paramètres H2 constants (courbes plates) et compare :

$$\text{MM-H1} \quad \text{vs} \quad \text{MM-H2}.$$

La théorie impose que H2 se réduise à H1 lorsque $r(t) = r$ et $\sigma(t) = \sigma$; le test exige une erreur relative inférieure à 1%. Cette validation est structurante pour garantir la cohérence de l'implémentation H2.

Scénario 4 : Monte Carlo vs Moment Matching. `TestMonteCarloVsMomentMatching` compare le prix Moment Matching (H2) et une estimation Monte Carlo (H2). Les critères retenus sont :

- écart relatif inférieur à 5%,
- erreur standard Monte Carlo inférieure à un seuil (contrôle de précision).

Ce test ne vise pas une égalité stricte, mais une validation empirique que l'approximation par moment matching est dans un ordre de grandeur cohérent avec une référence stochastique.

Scénario 5 : réduction de variance. `TestVarianceReduction` compare l'erreur standard Monte Carlo avec et sans variable de contrôle. Le critère impose :

$$\text{SE}_{CV} < \text{SE}_{standard} \quad \text{et} \quad \text{réduction} > 30\%.$$

Choix technique : ce test a pour but de vérifier l'efficacité empirique de l'algorithme d'ajustement (estimation de β) et de détecter des erreurs de covariance/variance.

Scénario 6 : sensibilité à la corrélation. `TestCorrelationSensitivity` vérifie une propriété qualitative importante : à volatilités identiques, une corrélation plus forte augmente la volatilité du panier, ce qui tend à augmenter la valeur d'un call :

$$\rho_{\text{haut}} > \rho_{\text{bas}} \quad \Rightarrow \quad C_{\text{haut}} > C_{\text{bas}}.$$

Le test impose également un écart minimal en pourcentage pour s'assurer que la sensibilité est significative.

Scénario 7 : impact de paramètres déterministes (non constants). `TestDeterministicParametersH2` construit :

- une courbe de taux croissante (ex. $1.5\% \to 2.5\%$),
- une courbe de volatilité croissante (ex. $15\% \to 25\%$),

et compare le prix obtenu à celui d'un modèle constant "moyen". Le test impose que la différence soit non négligeable (au moins 1%), ce qui valide que l'intégration des courbes $r(t)$ et $\sigma(t)$ impacte réellement le pricing.

Scénario 8 : relation Call/Put (cohérences économiques). `TestPutCallRelationship` effectue plusieurs vérifications qualitatives :

- call ATM $>$ put ATM lorsque le taux est positif (règle empirique cohérente dans ce cadre simple),
- monotonie en strike : call ITM $>$ call ATM, put OTM $<$ put ATM,
- cohérence relative : call ITM $>$ put OTM.