Construction of a Basket Option Pricer

Rémi Schmitt and Théo Verdelhan

January 2026

1 Market Data

1.1 Risk-Free Rate Curve

Element	**Value**
Reference	€STR (Euro Short-Term Rate)
Source	ECB
Pricing Date	January 23, 2026
Overnight Rate	**1.933%**

Remark. €STR is an overnight rate. For maturities ≤ 1 year and in a stable rate environment, the constant-rate approximation is acceptable.

1.2 Volatilities

Under H1, volatilities are estimated from historical prices (realized volatility). Under H2, deterministic volatilities $\sigma_i(t)$ are defined by linear interpolation between maturity points (proxy for implied calibration).

1.3 Basket Composition

We consider n assets ($1 \leq n \leq 10$) with normalized weights (a_i) ($\sum a_i = 1$). The correlation matrix (ρ_{ij}) is assumed symmetric and valid.

2 Theory and Technical Implementation

2.1 General Framework

Weighted basket and payoff:

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

Under Q:

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

Since $A(T)$ is a sum of correlated lognormals, we use: (i) *Moment Matching* (fast approximation), (ii) Monte Carlo H2 (numerical benchmark).

2.2 Moment Matching: Principle

We approximate $A(T)$ by a lognormal $\bar{A}(T)$ calibrated on the first two moments:

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

hence

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

Black-type price:

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

Numerical point. If $M_2 \leq M_1^2$ (rounding issues), we enforce $M_2 > M_1^2$ (epsilon safeguard) to avoid $\hat{\sigma}^2 \leq 0$.

2.2.1 Representation of Assets and Basket under H1

Each asset is represented by the class `Stock` with attributes: `SpotPrice`: initial price $S_i(0)$, `Volatility`: constant volatility σ_i, `DividendRate`: continuous dividend rate q_i.

Baskets are represented by `Basket`. Each basket contains: the list of assets, the weight vector (a_i), the correlation matrix (ρ_{ij}), the risk-free rate.

Technical choice: The following checks are implemented: weights sum to 1, consistent correlation matrix dimensions, matrix symmetry and $\rho_{ii} = 1$, $\rho_{ij} \in [-1, 1]$.

This exactly corresponds to the multi-asset Black–Scholes framework.

2.3 Moment Matching Pricer H1 (constant parameters)

Model.

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

Moments.

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

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

Link with the code. The class `MomentMatchingPricer` implements the Brigo et al. approximation under H1. This corresponds to `CalculateFirstMoment()`, which sums weighted forwards. M_2 is implemented in `CalculateSecondMoment()` via a double loop and an exponential covariance term.

2.4 Moment Matching Pricer H2 (deterministic rates and volatilities)

Assets in H2: deterministic volatilities Under H2, assets are represented by the class `StockH2` with attributes: `SpotPrice`: $S_i(0)$, `VolatilityModel`: object of type `DeterministicVolatilityModel`, `DividendRate`: q_i (constant).

Baskets are represented by `BasketH2`. Each basket contains: the list of assets, the weight vector (a_i), the correlation matrix (ρ_{ij}), the risk-free rate (H1) or the rate model (H2).

Technical choice: The following checks are implemented: weights sum to 1, consistent correlation matrix dimensions, matrix symmetry and $\rho_{ii} = 1$, $\rho_{ij} \in [-1, 1]$.

The volatility $\sigma_i(t)$ is defined by a linearly interpolated curve from points (t, σ):

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

Two essential methods are provided: `GetVolatility(t)`: returns $\sigma_i(t)$ by interpolation, `IntegrateVariance(T)`: computes $\int_0^T \sigma_i(t)^2\, dt$ using the trapezoidal rule.

Technical choice: Linear interpolation was selected for numerical stability, implementation simplicity, and consistency with the piecewise-constant volatility approximation commonly used in the literature (Brigo et al.).

Model.

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

Moments.

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

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

In the code, $R(0,T)$ and $C_{ij}(0,T)$ are evaluated numerically (trapezoidal rule) from deterministic curves $r(\cdot)$ and $\sigma_i(\cdot)$. Thus, $R(0,T)$ is computed in `IntegrateRate()` in class `DeterministicRateModel()`, and $C_{ij}(0,T)$ in `IntegrateVariance()` in class `DeterministicVolatilityModel()`.

2.5 Monte Carlo Pricer H2 and Variance Reduction

The class `MonteCarloPricerH2` provides a numerical benchmark estimate of the option price under H2 (deterministic $r(t)$ and $\sigma_i(t)$). It returns a `MonteCarloResultH2` object containing the estimated price, variance, and standard error, as well as (if applicable) variance reduction information (`ControlVariateAdjustment`, `VarianceReduction`).

Correlations (Cholesky).

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

In the code: `MathUtils.CholeskyDecomposition(basket.CorrelationMatrix)` computes L, `GenerateCorrelatedRandomN` `numAssets)` computes $\mathbf{Z}_c$.

Technical choice: Cholesky decomposition is the standard method to correlate Gaussian variables; it is simple, robust, and efficient for moderate basket sizes.

Scheme (exponential log-Euler) and discretization.

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

In the code, evolution is implemented in `SimulatePaths()`: time step: `numSteps = max(252, (int)(maturity*365))`, $\Delta t = T/$`numSteps`, update:

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

Estimator and uncertainty.

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

In the code:

```
price = sum/N,  variance = sumSquared/N - price2̂,  standardError = sqrt(variance/N).
```

Variance reduction via control variate (geometric mean)

$$G(T) = \prod_{i=1}^n S_i(T)^{a_i}.$$

Control variate estimator:

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

Implementation specificity. In the current code, the adjustment is:

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

without explicit injection of $E[Y]$. This may reduce variance when X and Y are highly correlated, but it is not the canonical centered version and may introduce bias if $E[Y] \neq 0$.

3 Numerical Results and Detailed Parameters

This section presents the results obtained with the executable `dotnet run` (menu: automatic demonstration, interactive mode, unit and functional tests). The values reported below correspond directly to the console outputs of the application.

Note: All calculations use the risk-free rate €STR = 1.933% (source: ECB, 01/23/2026).

3.1 Automatic Demonstration: Moment Matching (H1) and Moment Matching (H2)

3.1.1 Case H1: Constant Parameters

Parameters. The basket (2 assets) and market parameters used in the H1 demonstration are:

Initial basket value: $A_0 = 102.00$ €, Maturity: $T = 1$ year, Constant risk-free rate: $r = 1.933\%$ (€STR ECB 01/23/2026), Strikes: $K_{\text{call}} = 107.10$ € (105% of A_0), $K_{\text{put}} = 96.90$ € (95% of A_0)

Results (Moment Matching H1). The prices obtained with `MomentMatchingPricer` are:

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

Validation in dimension 1 (Black–Scholes comparison). In the limiting case of a one-asset basket, the Moment Matching algorithm converges toward the Black–Scholes formula. The console displays:

Method	Price
Moment Matching (1 asset case)	8.266336 €
Black–Scholes	8.433327 €
Difference	1.669911×10^{-1}

3.1.2 Case H2: Deterministic Parameters

Parameters. The H2 demonstration illustrates the impact of non-constant $r(t)$ and $\sigma_i(t)$:

€STR rate curve (ECB 01/23/2026):

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

Deterministic volatilities (linear interpolation between $t = 0$ and $t = 1$):

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

Initial basket value: $A_0 = 108.00$ €, Strikes: $K_{\text{call}} = 110.00$ €, $K_{\text{put}} = 105.00$ €, Maturity: $T = 1$ year

Results (Moment Matching H2). The prices obtained with `MomentMatchingPricerH2` are:

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

H2 → H1 Convergence Test (constant parameters). When deterministic curves are flat (constant rate and volatility), theory implies that H2 reduces to H1. The console confirms:

H1 Price	H2 Price	Relative Error
6.172548 €	6.172548 €	0.0000%

3.1.3 Monte Carlo H2 and Variance Reduction (Automatic Demonstration)

The automatic demonstration also includes a Monte Carlo (H2) comparison with and without variance reduction:

Method	Price	Estimator Standard Deviation (σ)
Standard MC	7.5827 €	0.0568
MC with Control Variate	7.3949 €	0.0066

The reported variance reduction is **98.6%**. This illustrates the benefit of using a control variate based on the geometric mean of the basket (see implementation `CalculateControlVariate` and adjustment `ApplyControlVariateReduction`).

4 Appendix

4.1 Software Validation: Unit Tests and Functional Tests

To ensure the reliability of numerical results and the overall consistency of the implementation, the project integrates two complementary validation levels:

- **Unit tests** (`UnitTests`): verification of elementary building blocks (mathematical functions, object construction, local pricing consistency)
- **Functional tests** (`FunctionalTests`): end-to-end scenarios reproducing representative use cases and verifying economic properties (sensitivities, consistency across methods, H1/H2 convergence)

These tests are executed via static methods `RunAllTests()` that iterate through a dictionary of boolean functions `Func<bool>` and produce a readable console report (success/failure).

4.1.1 Unit Tests (`UnitTests`)

Normal CDF verification. `TestNormalCdf` checks reference values:

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

with numerical tolerances consistent with the approximation implemented in `MathUtils.NormalCdf`.

Black–Scholes and put-call parity verification. `TestBlackScholes` validates `MathUtils.BlackScholesPrice` by checking put-call parity:

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

Object construction tests (`Stock` and `Basket`). `TestStockConstruction` verifies parameter integrity (name, spot, volatility, dividend).

`TestBasketConstruction` verifies the initial basket value:

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

and consistency with `Basket.GetBasketValue()`.

Moment Matching pricer consistency test. `TestMomentMatchingPricer` tests the case $n = 1$ (single-asset option). The test verifies natural bounds:

- strictly positive price,
- for a call: $C < A_0$,
- for an ITM put: $P < K$.

Validation of deterministic H2 models. `TestH2Models` validates:

- **linear interpolation** of instantaneous rate $r(t)$ in `DeterministicRateModel`

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

- **linear interpolation** of volatility $\sigma(t)$ in `DeterministicVolatilityModel`,
- construction of `StockH2`.

Strike monotonicity consistency. `TestPricingConsistency` verifies an economic principle: for a European call with identical maturity,

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

This detects sign errors in d_1, d_2 or incorrect discounting.

4.1.2 Functional Tests (`FunctionalTests`)

Scenario 1: 2-asset ATM basket. `TestTwoAssetBasketATM` verifies:

- strictly positive call and put prices,
- relative consistency (with $r > 0$, ATM call tends to exceed ATM put),
- simple bounds ($C < A_0$),
- reasonable call-put spread.

Scenario 2: diversified 3-asset basket. `TestThreeAssetDiversified` builds a three-sector basket with different volatilities and dividends, then tests an **OTM** call. The price is checked against a plausibility bound to detect order-of-magnitude errors.

Scenario 3: H2 to H1 convergence. `TestH1H2Convergence` imposes constant H2 parameters (flat curves) and compares:

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

The test requires relative error below 1%.

Scenario 4: Monte Carlo vs Moment Matching. `TestMonteCarloVsMomentMatching` compares H2 Moment Matching and Monte Carlo prices. Criteria:

- relative difference below 5%,
- Monte Carlo standard error below a threshold.

Scenario 5: variance reduction. `TestVarianceReduction` compares Monte Carlo standard errors with and without control variate. Criterion:

$$\text{SE}_{CV} < \text{SE}_{standard} \quad \text{and reduction} > 30\%.$$

Scenario 6: correlation sensitivity. `TestCorrelationSensitivity` verifies:

$$\rho_{\text{high}} > \rho_{\text{low}} \Rightarrow C_{\text{high}} > C_{\text{low}}.$$

Scenario 7: impact of deterministic (non-constant) parameters. `TestDeterministicParametersH2` builds:

- increasing rate curve (1.5% $\to$ 2.5%),
- increasing volatility curve (15% $\to$ 25%),

and requires at least 1% price difference versus a constant "average" model.

Scenario 8: Call/Put relationship (economic consistency). `TestPutCallRelationship` verifies:

- ATM call > ATM put when rate is positive,
- strike monotonicity,
- relative consistency: ITM call > OTM put.