Documentation de la librairie de pricing

1 Présentation générale

Cette bibliothèque de pricing, écrite en C++17 et orientée objet, illustre à la fois :

— des **concepts financiers** (options vanilles, produits de taux, exotiques, modèles de diffusion) ;

— une **architecture logicielle propre** (SOLID, design patterns, séparation claire des responsabilités) ;

— des **bonnes pratiques C++ modernes** (smart pointers, exceptions, tests unitaires, CMake).

L'objectif du projet est double :

1. Montrer une compréhension opérationnelle des **méthodes de valorisation** standards en finance de marché.
2. Montrer une capacité à **concevoir et structurer** une librairie C++ maintenable et extensible.

2 Périmètre financier

2.1 Produits actions

Les produits actions sont valorisés sous un modèle de type Black–Scholes.

2.1.1 Options vanilles européennes

Options de type call et put européens, avec payoff standard :

$$\text{Call} : (S_T - K)^+, \qquad \text{Put} : (K - S_T)^+.$$

2.1.2 Options digitales (cash-or-nothing)

Options digitales, dont le payoff est de la forme :

$$Q \cdot \mathbf{1}_{\{S_T > K\}} \quad \text{(call digital)}, \qquad Q \cdot \mathbf{1}_{\{S_T < K\}} \quad \text{(put digital)}.$$

Le pricing est réalisé en **formule fermée** dans le modèle de Black–Scholes (options digitales de type cash-or-nothing).

2.1.3 Options asiatiques (arithmétiques)

Options asiatiques sur la moyenne arithmétique :

$$\overline{S} = \frac{1}{n} \sum_{i=1}^{n} S_{t_i}, \qquad \text{Call asiatique} : (\overline{S} - K)^+.$$

La valorisation est effectuée par **Monte Carlo** :

— simulation de chemins sous Black–Scholes ;

— calcul de la moyenne arithmétique le long de chaque trajectoire ;

— actualisation de la moyenne des payoffs.

2.1.4 Options barrières

Options barrières de type :
— `UpAndOut`, `DownAndOut` ;
— `UpAndIn`, `DownAndIn`.

Le payoff est celui d'une option vanille, conditionné par le franchissement (ou non) d'une barrière (activation ou désactivation). Le pricing est réalisé par **Monte Carlo** avec suivi de la trajectoire du sous-jacent et test de franchissement.

2.2 Produits de taux

Les produits de taux sont modélisés avec une courbe de rendement plate et une volatilité de type Black pour les options de taux.

2.2.1 Caplet / Floorlet

— **Caplet** : option call sur un taux forward ;
— **Floorlet** : option put sur un taux forward ;

Payoff typique d'un caplet à la date de paiement :

$$\text{Payoff} = N \cdot \delta \cdot (L_T - K)^+,$$

où N est le nominal, δ la fraction d'année, L_T le taux libor/forward et K le strike.

La valorisation utilise le **modèle de Black** sur le taux forward.

2.2.2 Cap / Floor

— Un **Cap** est un portefeuille de caplets.
— Un **Floor** est un portefeuille de floorlets.

Le prix du Cap/Floor est obtenu en sommant les prix des caplets/floorlets individuels.

2.2.3 Interest Rate Swap (IRS)

Swap de taux simple, de type jambe fixe vs jambe flottante avec taux forward constant. Pour un swap *payer* (payeur fixe, receveur flottant), le prix est donné par :

$$PV = \text{sign} \times N \sum_i \alpha_i \, DF(t_i) \, (F - K),$$

où :
— N : notional ;
— α_i : accrual factor de la période i ;
— $DF(t_i)$: facteur d'actualisation ;
— F : taux forward ;
— K : taux fixe du swap ;
— sign $= +1$ pour un payer swap (pay fixed / receive float, avec $F - K$), -1 pour un receiver swap.

2.2.4 Swaption européenne

Une swaption européenne donne le droit d'entrer dans un swap (payer ou receiver) à une date d'exercice donnée. Dans le cadre simplifié du projet :
— la swaption est vue comme une option sur le taux de swap S ;
— l'*annuité* associée est $A = \sum_i \alpha_i DF(t_i)$;
— le pricing utilise la **formule de Black** sur le taux de swap.

2.3 Modèles et techniques de pricing

2.3.1 Modèle actions : Black–Scholes

Sous-jacent S_t suivant une dynamique log-normale :

$$dS_t = S_t \left((r - q)\, dt + \sigma\, dW_t \right),$$

avec :
- r : taux sans risque (constant) ;
- q : dividende continu ;
- σ : volatilité constante.
- Vanilles / digitales : **formules fermées** de Black–Scholes ;
- Exotiques (asiatiques, barrières) : **Monte Carlo**.

2.3.2 Modèle de taux : Black pour les options de taux

- Courbe de taux plate via une `YieldCurve` ;
- Volatilité σ constante pour les taux forwards / taux de swap ;
- Pricing :
  - caplets, caps, swaptions : formule de Black sur un taux forward ou un taux de swap ;
  - swaps : somme des flux actualisés de la jambe nette (float – fixe ou l'inverse).

2.3.3 Techniques numériques

- Formules de Black factorisées dans une unité utilitaire (`BlackFormula`) ;
- Monte Carlo pour les options exotiques :
  - schéma exponentiel exact sous GBM ;
  - paramétrage du nombre de chemins et de pas de temps ;
  - pas de techniques avancées de réduction de variance (projet pédagogique).

3 Architecture du projet

L'architecture suit une logique de couches et met en avant les abstractions clefs de la modélisation de produits financiers.

3.1 Arborescence

Listing 1 – Structure simplifiée du projet

```
include/
  core/        # abstractions g n riques (Instrument, PricingEngine,
     Payoff, Factories)
  market/      # donn es de march  (courbes, spot, dividendes)
  models/      # mod les stochastiques / de taux (Black-Scholes, Black IR)
  products/    # instruments financiers (equity, taux, exotiques)
  engines/     # moteurs de pricing (Black, Monte Carlo, swap PV)
  utils/       # outils math matiques (BlackFormula)

src/
  core/
  market/
  models/
  products/
  engines/
  utils/
```

```
examples/
  main_basic.cpp          # d monstration simple (equity, caplet, cap,
     swap, swaption, exotiques)
  main_factory.cpp        # d monstration avec InstrumentFactory et
     EngineFactory

tests/
  test_main.cpp           # point d' entr e doctest
  test_payoff.cpp
  test_black_formula.cpp
  test_european_option.cpp
  test_rates.cpp

third_party/
  doctest/doctest.h       # framework de tests header-only
```

3.2 Couche `core/` : abstractions génériques

Instrument Interface abstraite représentant un produit financier. Elle contient un `std::shared_ptr<Pric` et expose :

— `double NPV() const` : délègue le calcul de la valeur au moteur de pricing.

PricingEngine Classe de base abstraite pour tous les moteurs de pricing. Elle implémente un **Template Method** :

— `double calculate(const Instrument&)` : structure du calcul ;
— méthode virtuelle protégée `priceImpl(const Instrument&)` à surcharger dans les engines concrets.

Payoff Représente la fonction de payoff $f(S_T)$. Implémentations :

— `PlainVanillaPayoff` : call/put standard ;
— `DigitalPayoff` : digitale cash-or-nothing.

InstrumentFactory Simplifie la création d'instruments :

— `makeEuropeanOption(type, K, T)` ;
— `makeDigitalOption(...)` ;
— `makeAsianOption(...)` ;
— `makeUpAndOutOption(...)` ;
— `makeCaplet(...)` ;
— `makeSwap(...)` ;
— `makeSwaption(...)`.

EngineFactory Crée dynamiquement le moteur de pricing adapté à un `Instrument` concret, à partir de modèles (`BlackScholesModel`, `BlackIRModel`) :

— `EuropeanOption` ⇒ `EuropeanOptionBSEngine` ;
— `DigitalOption` ⇒ `DigitalOptionBSEngine` ;
— `AsianOption` ⇒ `AsianOptionMCEngine` ;
— `BarrierOption` ⇒ `BarrierOptionMCEngine` ;
— `Caplet`, `Cap`, `Floor` ⇒ engines Black IR ;
— `InterestRateSwap` ⇒ `SwapEngine` ;
— `Swaption` ⇒ `SwaptionBlackEngine`.

3.3 Couche `market/` : données de marché

YieldCurve
- Taux sans risque constant ;
- Méthodes : `rate()`, `discount(T)`.

EquityCurve
- Spot initial, dividende continu ;
- Méthodes : `spot()`, `dividendYield()`.

3.4 Couche `models/` : modèles de pricing

BlackScholesModel
- Dépend de `YieldCurve` (taux) et `EquityCurve` (spot, dividendes) ;
- Expose : `spot()`, `sigma()`, `discount(T)`, `forward(T)`, `rate()`, `dividendYield()`.

BlackIRModel
- Dépend d'une `YieldCurve` (taux) et d'une volatilité Black ;
- Expose : `discount(T)`, `rate()`, `sigma()`.

3.5 Couche `products/` : instruments financiers

Equity
- `EuropeanOption` : option européenne avec payoff générique et maturité ;
- `DigitalOption` : option digitale cash-or-nothing ;
- `AsianOption` : option asiatique (payoff sur moyenne arithmétique) ;
- `BarrierOption` : option barrière (up/down, in/out).

Taux
- `Caplet`, `Floorlet`, `Cap`, `Floor` ;
- `InterestRateSwap` : swap de taux simple ;
- `Swaption` : option européenne sur swap.

3.6 Couche `engines/` : moteurs de pricing

Equity, formules fermées
- `EuropeanOptionBSEngine` : formule fermée de Black–Scholes sur le forward ;
- `DigitalOptionBSEngine` : digitale cash-or-nothing en BS.

Equity, Monte Carlo
- `AsianOptionMCEngine` :
  - simulation de chemins sous GBM ;
  - moyenne arithmétique des spots simulés ;
  - calcul du payoff moyen actualisé.
- `BarrierOptionMCEngine` :
  - simulation sous GBM avec suivi de barrière ;
  - payoff conditionnel au franchissement/non-franchissement.

Taux, modèle de Black
- `CapletBlackEngine` : Black sur taux forward ;
- `CapBlackEngine` / `FloorBlackEngine` : agrégation de caplets/floorlets ;
- `SwaptionBlackEngine` : Black sur taux de swap avec annuité.

Taux, déterministe
— `SwapEngine` : somme des flux actualisés de la jambe nette.

3.7 Couche `utils/` : outils mathématiques

BlackFormula
— `normalCdf(x)` : CDF de la loi normale standard ;
— `blackForward(F, K, stdDev, type)` : formule de Black pour une option sur forward (call/put) ;
— `blackDigitalForward(F, K, stdDev, type, payout)` : formule de Black pour une digitale cash-or-nothing.

4 Concepts de programmation

4.1 Design patterns

Strategy
— `Payoff` : stratégie de payoff (forme de $f(S_T)$) ;
— `PricingEngine` : stratégie de pricing (formule fermée, Monte Carlo, etc.).

Template Method
— Implémenté dans `PricingEngine::calculate()` ;
— Encapsule le squelette du calcul (vérifications, appel à `priceImpl`) ;
— Les engines concrets définissent uniquement `priceImpl`.

Factory / Simple Factory
— `InstrumentFactory` : création de produits cohérents (bons payoffs, paramètres) ;
— `EngineFactory` : mapping instrument → moteur, en fonction des modèles.

Dependency Inversion Principle
— Le code utilisateur manipule surtout des abstractions (`Instrument`, `PricingEngine`) ;
— Le choix des implémentations concrètes (BS vs MC, Black vs déterministe) est encapsulé dans les factories.

4.2 Principes SOLID

— **Single Responsibility** :
  — `Instrument` décrit un produit, sans savoir comment le pricer ;
  — `PricingEngine` sait pricer, sans connaître la structure interne du produit.
— **Open/Closed** :
  — ajout d'un nouveau produit (e.g. option bermudienne) en ajoutant un nouvel instrument et un engine ;
  — pas besoin de modifier les produits existants.
— **Liskov Substitution** : tout `Instrument` concret doit pouvoir être utilisé via un pointeur/référence vers `Instrument`.
— **Interface Segregation** : interfaces simples, ciblées, sans "gros objet".
— **Dependency Inversion** : le main dépend d'abstractions, pas des implementations concrètes.

4.3 C++ moderne

— Standard : C++17 ;
— **Smart pointers** :

  - `std::unique_ptr` pour la propriété exclusive (payoffs) ;
  - `std::shared_ptr` pour le partage des modèles (courbes, modèles BS/Black IR).
- Utilisation maîtrisée de `dynamic_cast` pour vérifier les types d'instruments dans les engines ;
- `const`-correctness sur les méthodes n'altérant pas l'état ;
- Exceptions explicites pour les cas d'erreur (type d'instrument non supporté, tailles incohérentes, etc.).

5 Tests et qualité

Les tests unitaires sont écrits avec **doctest** (framework header-only, https://github.com/doctest/doctest).

5.1 Organisation des tests

Un exécutable `pricing_tests` est défini, avec les fichiers :

- `test_main.cpp` : point d'entrée doctest ;
- `test_payoff.cpp` : tests des payoffs (`PlainVanillaPayoff`, `DigitalPayoff`) ;
- `test_black_formula.cpp` : tests de `normalCdf` et `blackForward` (comportement à volatilité nulle) ;
- `test_european_option.cpp` : pricing BS d'un call européen avec ordre de grandeur attendu ;
- `test_rates.cpp` : tests de pricing d'un caplet, d'un swap 5Y, d'une swaption payer.

Objectif :

- garantir la **cohérence numérique** des briques de base ;
- faciliter la **non-régression** lors de l'ajout de nouveaux produits ou modèles.

6 Extensions possibles

Quelques pistes d'évolution pour enrichir la librairie :

Côté finance

- Swaptions bermudiennes, modèles de taux multi-facteurs ;
- Smile de volatilité (surface de volatilité, interpolation) ;
- Produits FX : options, cross-currency swaps, TARF, autocalls, etc.

Côté C++ / architecture

- Interfaces pour le calcul des sensibilités (Greeks) ;
- Gestionnaire de calendrier (jours ouvrés, conventions de marché) ;
- Paramétrage avancé des méthodes Monte Carlo (antithetic variates, Sobol, etc.) via des stratégies.

Ce projet a pour but d'illustrer ma capacité à manipuler des modèles et produits financiers standard tout en respectant les bonnes pratiques de **conception C++ moderne** et de **qualité logicielle** (architecture claire, patterns, tests).