Moteur CVA/DVA — Démo illustrative

Jugurtha Boudarene

Janvier 2026

Table des matières

1 Présentation rapide du projet et exécution locale **1**
1.1 Objectif . . . 1
1.2 Pré-requis . . . 2
1.3 Installation (machine quelconque) . . . 2
1.4 Lancer l'application Streamlit . . . 2
1.5 Lancer un run en ligne de commande (sans UI) . . . 2

2 Structure du projet (arborescence) **2**

3 Guide utilisateur (workflow) **3**
3.1 Workflow recommandé (via l'UI Streamlit) . . . 3
3.2 Workflow « batch » (CLI) + lecture UI . . . 3

4 Organisation des outputs d'un run **4**

5 Fichiers importants (référencés dans `docs_registry.json`) **4**
5.1 Courbe initiale (term structure) : point d'ancrage modèle . . . 4
5.2 Taux : modèle HW1F++ et pricer ZC affine . . . 4
5.3 Crédit : entités et intensité Log-OU . . . 4
5.4 Produits : IRS vanilla (pricing, par rate, roll) . . . 5
5.5 Exposition : MTM paths → EPE/ENE . . . 5
5.6 Orchestration : moteur de scénarios portefeuille (taux + crédit + xVA) . . . 5
5.7 xVA : legs bucketés, agrégation et explicabilité Shapley . . . 5

Cliquez pour accéder à la démo Streamlit (CVA/DVA)

1 Présentation rapide du projet et exécution locale

1.1 Objectif

Ce projet est une **démo technique** reproduisant une chaîne de calcul **CVA/DVA** dans un cadre **reproductible** :

- **Taux** simulés via **Hull–White 1F++** ajusté à une courbe initiale (Nelson–Siegel).
- **Crédit** simulé via **intensité log-OU** (contreparties *et* banque), avec calcul de **Survie** et **PD marginales**.
- **Expositions EPE/ENE** à partir de la **MTM** scénario par scénario d'IRS vanilla.
- **CVA/DVA** via agrégation des *legs* (actualisation, PD, exposition).
- **Explicabilité** : décomposition **Shapley** (contributions DF / EPE(ENE) / PD / Survie).
- **Traçabilité** : exports CSV/JSON/PNG, runs historisés dans `./data/`.

1.2 Pré-requis

— Python **3.10**+ recommandé.
— Dépendances principales : `streamlit`, `numpy`, `pandas`, `altair`, `matplotlib`, `pyyaml`.

1.3 Installation (machine quelconque)

Dans un terminal à la racine du projet :

```
# 1) (Optionnel) crer et activer un environnement virtuel
python -m venv .venv

# Windows
.venv\Scripts\activate


# 2) installer les dpendances
pip install -r requirements.txt
```

1.4 Lancer l'application Streamlit

Deux points d'entrée existent :
— **Interface multi-pages** (recommandé) : `Home.py`
— **Landing page « marketing »** : `app_streamlit.py`

```
# Landing page
streamlit run app_streamlit.py
```

1.5 Lancer un run en ligne de commande (sans UI)

```
python main.py
```

Où sont les résultats ? Chaque run crée un dossier `data/run_...` contenant :
— `totals/` : totaux CVA/DVA + séries (legs agrégés, DF).
— `per_counterparty/` : expositions/crédit/legs par contrepartie + méta JSON.
— `shapley_per_counterparty/` : Shapley CVA/DVA par contrepartie (si activé).
— `figs/` : PNG (si activé).

2 Structure du projet (arborescence)

NB : dans le dépôt, certaines pages Streamlit ont des noms contenant des emojis. Dans ce guide, on affiche des noms simplifiés pour éviter les problèmes de compilation LaTeX.

```
.
 app_streamlit.py
 main.py
 export_projet.py
 app_lib/
  io.py
  runner.py
  state.py
  style.py
  __init__.py
 pages/
```

```
 1_Dashboard.py
 2_Contreparties.py
 3_Sensibilites.py
 4_Simulation.py
 5_Documentation.py
 6_Portfolio_Tracking.py
 code_docs.py
 docs_registry.json
src/
 core/ (TimeGrid, utils)
 rates/ (Nelson-Siegel, HW1F++, ZC pricer, DF curve)
 credit/ (Log-OU intensity, entits banque/contrepartie)
 products/ (Schedule, Swap, roll_swap)
 exposure/ (ExposureEngine : MTM -> EPE/ENE)
 sim/ (Simulator : run_portfolio)
 io/ (exports CSV/JSON + plots)
 xva/ (CVA/DVA legs + Shapley explain)
 test.py (mini test de cohrence)
data/
    run_YYYYMMDD_HHMMSS/
    run_YYYYMMDD_HHMMSS_MAR/ (snapshot Mar si activ)
```

3 Guide utilisateur (workflow)

3.1 Workflow recommandé (via l'UI Streamlit)

1. **Démarrer l'app** :

   ```
   streamlit run app_streamlit.py
   ```

2. **Créer un run** (page *Simulation*) :
   — Choisir **Nombre de contreparties** (ex : 20).
   — Choisir **Nombre de scénarios** `N` (ex : 5000).
   — Choisir une **seed** (reproductibilité).
   — Option **PNG** (exports graphiques).
   — Option **Snapshot Mar + compare + Shapley** (recommandé pour la démo).
   — Lancer **Run**.

3. **Explorer les résultats** (pages de lecture) :
   — **Dashboard** : CVA/DVA totaux, legs agrégés, downloads rapides.
   — **Contreparties** : EPE/ENE, PD/Survie, legs CVA/DVA par contrepartie.
   — **Sensibilités** : Shapley CVA/DVA + (optionnel) comparatif Jan/Mar (PV Jan).
   — **Portfolio Tracking** : ranking (CVA/DVA/Net), compare run-vs-run, exports CSV.

4. **Exporter / tracer** :
   — Boutons `download` (CSV/ZIP) dans les pages.
   — Dossier `data/run_...` conservé comme **snapshot** d'audit.

5. **Lire la documentation du code** (page *Documentation*) :
   — La page limite la navigation aux fichiers listés dans `docs_registry.json`.
   — Le panneau de droite affiche une **fiche manuelle** (titre, résumé, usage, notes).

3.2 Workflow « batch » (CLI) + lecture UI

1. Exécuter un run via `main.py`.
2. Ouvrir l'UI et sélectionner le run créé dans la sidebar (sélecteur `Run`).

4 Organisation des outputs d'un run

Soit un run `data/run_X`. Les éléments structurants sont :

- `totals/` :
  - `totals.csv` : `CVA_total`, `DVA_total`.
  - `cva_legs_sum.csv` / `dva_legs_sum.csv` : séries temporelles agrégées (somme sur contreparties).
  - `df_curve.csv` : DF(0,t) sur la grille.
- `per_counterparty/` :
  - `exposures_{cid}.csv` : colonnes `k, EPE, ENE`.
  - `credit_{cid}.csv` : colonnes `k, PD_cpty, S_cpty`.
  - `xva_{cid}.csv` : colonnes `k, CVA_leg, DVA_leg` (+ ligne TOTAL).
  - `meta_{cid}.json` : `LGD_cpty`, `LGD_bank`, CVA/DVA par contrepartie.
- `shapley_per_counterparty/` (si activé) :
  - `shapley_cva_{cid}.csv` : `phi_DF`, `phi_EPE`, `phi_PD_cpty`, etc.
  - `shapley_dva_{cid}.csv` : `phi_DF`, `phi_ENE`, `phi_PD_bank`, `phi_S_cpty`, etc.
  - `xva_compare_pvjan.csv` : comparatif Jan/Mar (PV Jan).

5 Fichiers importants (référencés dans `docs_registry.json`)

Principe. La page *Documentation* lit `pages/docs_registry.json` : chaque clé correspond au chemin relatif d'un fichier, et chaque entrée fournit `title`, `tags`, `summary`, `usage`, `notes`. Dans l'UI, ces fiches sont rendues par `pages/code_docs.py` (panneau droit : fiche manuelle).

5.1 Courbe initiale (term structure) : point d'ancrage modèle

- `src/rates/termstructure/nelson_siegel.py`
  Rôle : courbe paramétrique **Nelson–Siegel** (compounding continu) fournissant : taux zéro $y(0,T)$, facteur d'actualisation $P(0,T)$ et forward instantané $f(0,t)$.

5.2 Taux : modèle HW1F++ et pricer ZC affine

- `src/rates/hw1f.py`
  Rôle : implémente **Hull–White 1F++** avec $\theta(t)$ dépendant du temps, **fit** pour recoller la courbe initiale, puis **simulation stable** du short rate $r(t)$ sur une grille uniforme.

- `src/rates/zc_pricer.py`
  Rôle : pricer analytique de **zéro-coupons** sous HW1F++ :

$$P(t,T) = A(t,T)\exp(-B(t,T)r_t).$$

5.3 Crédit : entités et intensité Log-OU

- `src/credit/entities.py`
  Rôle : définit les **entités crédit** (`Counterparty` et `Bank`) avec **LGD** et **spread initial**, et fabrique des paramètres d'intensité compatibles Log-OU.

- `src/credit/log_ou_intensity.py`
  Rôle : modèle d'intensité **Log-OU** : $\lambda(t) = \exp(x(t))$ avec $x(t)$ OU, garantissant $\lambda(t) > 0$.

5.4 Produits : IRS vanilla (pricing, par rate, roll)

— `src/products/swap.py`
Rôle : instrument **swap IRS vanilla** (fixed vs float) et ses méthodes de pricing sous HW1F++ via le pricer ZC affine.

5.5 Exposition : MTM paths → EPE/ENE

— `src/exposure/exposure_engine.py`
Rôle : moteur d'exposition pour un IRS vanilla **sans CSA** : à partir de chemins de taux courts $r[n,k]$, il reconstruit $P(t,T)$, calcule les **MTM** $V[n,k]$, puis agrège en **EPE** et **ENE** sur la grille.

5.6 Orchestration : moteur de scénarios portefeuille (taux + crédit + xVA)

— `src/sim/scenario_engine.py`
Rôle : orchestrateur de bout en bout de la simulation portefeuille :

1. simulation des taux HW1F++ (chemins $r[n,k]$) ;
2. simulation crédit Log-OU (banque + contreparties) ;
3. calcul exposition (EPE/ENE) via `ExposureEngine` ;
4. calcul legs et agrégats **CVA**/**DVA**.

5.7 xVA : legs bucketés, agrégation et explicabilité Shapley

— `src/xva/cva_dva.py`
Rôle : moteur **CVA**/**DVA bucketé** : calcule les *legs* par date puis agrège en montants totaux.

— `src/xva/shapley_explain.py`
Rôle : explicabilité : décomposition **Shapley exacte** (par permutations) de la variation des legs CVA/DVA, **bucket par bucket**.