

FRTB SA / SBM — 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
2	Structure du projet (arborescence)	2
3	Guide utilisateur (workflow)	3
3.1	Workflow recommandé (via l'UI Streamlit)	3
4	Organisation des outputs d'un run	3
5	Fichiers importants (référéncés dans docs_registry.json)	4
5.1	Entrées UI (Streamlit)	4
5.2	Cœur de calcul (FRTB)	4
5.3	Couche UI commune et persistance	5

[Cliquez pour accéder à la démo Streamlit \(FRTB SA / SBM\)](#)

1 Présentation rapide du projet et exécution locale

1.1 Objectif

Ce projet est une **démo technique** illustrant une chaîne de calcul **FRTB SA / SBM** dans un cadre **reproductible** :

- **Portefeuille** éditable via l'UI : **Equity Calls** et **GIRR** (*Swaps* et *Bonds*).
- **Equity** : calculs **Delta** / **Vega** / **Curvature** sur options call européennes (pricing Black-Scholes).
- **GIRR** : **Delta** via **bump-and-reprice** sur courbe zéro, agrégation **intra-bucket** (tenors) et **inter-bucket** (devises).
- **Traçabilité** : sauvegarde des *runs* (snapshot portfolio + marché + configs + résultats) dans une base **SQLite**.
- **Exports** : extraction CSV/JSON depuis l'UI (selon pages) pour conserver les résultats et les inputs.

1.2 Pré-requis

- Python **3.10+** recommandé.
- Dépendances principales (selon ton `requirements.txt`) : `streamlit`, `numpy`, `pandas`, `altair`, `matplotlib` (etc.).

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

Point d'entrée :

- **App multi-pages** : `app.py`

```
streamlit run app.py
```

Accès. Streamlit ouvre généralement l'application à :

`http://localhost:8501`

Où sont les résultats ? Les *runs* (inputs + outputs) sont historisés dans une base SQLite (par ex. `frtb_history.sqlite3`), et des exports peuvent être déclenchés depuis l'UI (CSV/JSON selon la page).

2 Structure du projet (arborescence)

```
.
├── app.py
├── ui_common.py
├── history_db.py
├── requirements.txt
├── __init__.py
├── engine.py
├── equity.py
├── girr.py
├── curves.py
├── market.py
├── portfolio.py
├── demo.py
├── pages/
│   ├── 1_Overview.py
│   ├── 2_Portfolio.py
│   ├── 3_Market.py
│   ├── 4_Configs.py
│   └── 5_Run_Results.py
```

```
6_Export.py
7_Historique.py
8_Documentation.py
code_docs.py
docs_registry.json
```

3 Guide utilisateur (workflow)

3.1 Workflow recommandé (via l'UI Streamlit)

1. Démarrer l'app :

```
streamlit run app.py
```

2. Charger / éditer le portefeuille (page *Portfolio*) :

- Charger un portefeuille **démo** (*reset/demo*) ou saisir/éditer manuellement.
- Vérifier la séparation par onglets : **Equity Calls**, **GIRR Swaps**, **GIRR Bonds**.
- Contrôler les champs clés (notional, maturité, devise, bucket, etc.).

3. Définir le marché (page *Market*) :

- Courbes zéro (par devise), FX, devise de reporting.
- Vérifier la cohérence : devises, tenors disponibles, interpolation.

4. Définir les configurations (page *Configs*) :

- **Equity** : risk weights, corrélations, paramètres vega/curvature.
- **GIRR** : tenors, taille du bump (bp), règles de corrélations et scénario (low/med/high), éventuellement *specified currency* si présent dans la démo.

5. Lancer un run (page *Run / Results*) :

- Cliquer *Run*.
- Le moteur exécute :
 - Equity : Delta / Vega / Curvature,
 - GIRR : bump-and-reprice swaps + bonds, puis agrégation SBM.

6. Lire les résultats et exporter :

- KPIs : K_{eq} , K_{swaps} , K_{bonds} , total (selon convention de l'app).
- Exports : CSV/JSON depuis la page *Export* ou via les boutons de téléchargement.

7. Historiser / restaurer (page *Historique*) :

- Chaque run est stocké avec un snapshot (portfolio + marché + configs) et ses résultats.
- Possibilité de recharger un run pour comparaison / reproductibilité.

8. 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).

4 Organisation des outputs d'un run

Principe. Un *run* correspond à l'exécution du moteur sur un triplet **Portfolio / Market / Configs** et produit :

- des **résultats** (KPI et détails par bucket/tenor),
- un **snapshot** des inputs (auditabilité),
- des **exports** (CSV/JSON) et une **trace** dans l'historique.

Persistance (SQLite). Le module `history_db.py` gère l'historique dans une base SQLite (souvent un fichier du type `frtb_history.sqlite3` stocké au niveau projet). Typiquement, on y retrouve :

- un identifiant de run (timestamp / UUID),
- la sauvegarde des inputs (portfolio, market, configs),
- les résultats agrégés et détaillés,
- des logs / erreurs éventuelles.

Exports. Selon l'implémentation des pages Streamlit, des boutons permettent d'exporter :

- le portfolio courant,
- le snapshot marché/configs,
- les résultats (KPI + breakdowns).

5 Fichiers importants (référéncés 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`.

Ci-dessous, une **liste de fichiers typiquement documentés** pour ce projet.

5.1 Entrées UI (Streamlit)

- `app.py`
Rôle : point d'entrée de l'app, mise en place de la sidebar globale, KPIs et navigation.
Usage : `streamlit run app.py`
- `pages/2_Portfolio.py`
Rôle : édition du portfolio (Equity Calls / GIRR Swaps / GIRR Bonds), chargement démo, upload CSV, exports.
Usage : page *Portfolio* → édition et sauvegarde en session.
- `pages/3_Market.py`
Rôle : saisie/édition du marché : courbes, FX, devise de reporting.
- `pages/4_Configs.py`
Rôle : paramétrage Equity/GIRR (RW, bumps, corrélations, scénarios low/med/high).
- `pages/5_Run_Results.py`
Rôle : déclenchement du run + lecture des KPIs et breakdowns.
- `pages/6_Export.py`
Rôle : exports des inputs/résultats (CSV/JSON selon le design).
- `pages/7_Historique.py`
Rôle : listing des runs, lecture détaillée, restauration d'un snapshot.
- `pages/8_Documentation.py` + `pages/code_docs.py`
Rôle : explorateur de code *restreint* aux fichiers déclarés dans `docs_registry.json`, et affichage de la documentation manuelle.

5.2 Cœur de calcul (FRTB)

- `frtb/engine.py`
Rôle : façade d'orchestration (`run`) qui appelle Equity et GIRR, assemble les outputs pour l'UI.
- `frtb/equity.py`
Rôle : pipeline Equity SA/SBM : pricing + greeks (Delta/Vega) + Curvature, RW et agrégation par bucket.

- `frtb/girr.py`
Rôle : GIRR Delta : pricing swaps/bonds, sensis bump-and-reprice, RW, corrélations et agrégations SBM.
- `frtb/curves.py`
Rôle : courbe zéro (DF, interpolation, bump sur nœuds/tenors).
- `frtb/market.py`
Rôle : snapshot marché : devise de reporting, FX, courbes par devise, conversion.
- `frtb/portfolio.py`
Rôle : structures de données (trades + conteneur portfolio), formats attendus pour l'UI et les exports.
- `frtb/demo.py`
Rôle : fixtures démo (market/config/portfolio) pour un run reproductible.

5.3 Couche UI commune et persistance

- `ui_common.py`
Rôle : helpers UI partagés : styles, parsing CSV, conversion portfolio \leftrightarrow tables, exports.
- `history_db.py`
Rôle : persistance des runs (SQLite) : stockage snapshot + résultats + logs, restauration et listing.