Documentation du pipeline `NN`

Répertoire `RepoFinal/NN`

November 13, 2025

1 Objectif général

Ce dossier implémente une chaîne complète pour apprendre des paramètres du modèle de Heston directement à partir de cotations d'options vanille. Toutes les briques reposent sur `PyTorch` en `float64` pour embarquer un pricer Carr–Madan FFT dans la boucle d'entraînement. Les fichiers décrits ci-dessous suivent l'ordre logique de la donnée marché jusqu'aux tests de non-régression.

2 Architecture globale

Étape 1: Chargement (`io_csv.py`) : lecture du CSV marché et validation des colonnes minimales (S_0, K, C_{mkt}, T).

Étape 2: Étiquetage Black–Scholes (`bs_iv.py`) : calcul du prix call analytique et inversion robuste pour récupérer la volatilité implicite de chaque ligne.

Étape 3: Features & split (`dataset.py`) : assemblage des tenseurs $[S_0/K, T, \sigma_{BS}]$, cibles C_{mkt} et découpage déterministe train/validation.

Étape 4: Paramétrisation Heston (`heston_torch.py`) : projection des sorties réseau vers $(\kappa, \theta, \sigma, \rho, v_0)$ physiquement valides, calcul de la fonction caractéristique et du pricer Carr–Madan.

Étape 5: Modèle neuronal (`model.py`) : MLP $3 \to 64 \to 64 \to 5$, mapping vers les paramètres Heston, pricing FFT par échantillon et perte $RMSE$.

Étape 6: Entraînement (`train.py`) : boucle AdamW, monitoring des pertes, tableau de validation et sauvegarde optionnelle des poids.

Étape 7: Tests (`tests/test_minimal.py`) : garde-fous unitaires (normalisation $\phi(0) = 1$, inversion BS/IV, flux autograd).

3 Détails par fichier

3.1 `bs_iv.py`

- `bs_call_torch`: implémente la formule fermée du call Black–Scholes avec contrôles numériques ($\sqrt{T}$ borné, $\log(S/K)$ clampé, volatilité minimale).
- `iv_call_brent_torch`: inversion de prix via bissection sécurisée ; initialise une borne supérieure σ_{high} élargie dynamiquement jusqu'à 5 si besoin.
- Utilitaires privés `_to_tensor` et `_norm_cdf` uniformisent les entrées en `torch.float64` et réutilisent une loi normale standard pré-instanciée.

3.2 `dataset.py`

- `make_dataset_from_csv`: convertit un `DataFrame` pandas en dictionnaire de tenseurs et ajoute la volatilité implicite comme feature guidant l'apprentissage.
- `split_train_val`: mélange déterministe via `torch.randperm` et fraction de validation bornée pour éviter les splits dégénérés.

3.3 `io_csv.py`

- `read_market_csv`: lit le fichier marché et lève une erreur descriptive si des en-têtes obligatoires manquent.

3.4 `heston_torch.py`

- `HestonParams`: dataclass qui applique `softplus` (paramètres positifs) et sigmoïde bornée (corrélation $\rho \in (-1, 1)$).
- `heston_cf`: implémente la version "Little Heston Trap" de la fonction caractéristique, avec correctifs numériques (ε complexes, forçage $\phi(0) = 1$).
- `carr_madan_call_torch`: calcule les prix via FFT (poids de Simpson, amortissement α, interpolation linéaire sur le log-strike).

3.5 `model.py`

- `HestonParamNet`: MLP compact dont la sortie est mappée vers `HestonParams`.
- `price_with_params`: reconstruit des lots cohérents (S_0, K, T, r), applique Carr–Madan par échantillon et remet la sortie à la forme d'origine.
- `rmse_loss`: métrique unique d'entraînement et de validation.

3.6 `train.py`

- Parsing des hyperparamètres (chemin CSV, taux, α, N_{FFT}, η, fraction de validation, seed, sauvegarde).
- Boucle d'entraînement AdamW avec journaux `RMSE` et tableau comparant prix marché/prédits, erreurs absolues et relatives.
- Sauvegarde optionnelle des poids via `torch.save`.

3.7 `tests/test_minimal.py`

- Vérifie la normalisation de la fonction caractéristique en zéro.
- Teste la bijection prix $\leftrightarrow$ volatilité via Black–Scholes.
- Lance une itération avant/arrière du réseau pour confirmer la présence de gradients non nuls.

4 Flux d'entraînement

Le script `train.py` orchestre les modules :

1. Lecture du CSV (`read_market_csv`) puis conversion en tenseurs (`make_dataset_from_csv`).
2. Split cohérent en sous-ensembles train/val.
3. Passage des features dans `HestonParamNet` pour obtenir des paramètres valides.
4. Pricing rapide via `price_with_params` qui délègue à `carr_madan_call_torch`.
5. Calcul des `RMSE`, rétropropagation sur la partie entraînement et suivi métrique sur validation.
6. (Optionnel) sauvegarde du modèle entraîné.

5 Remarque sur le format

LaTeX offre un rendu soigné pour des documents longs ou exportables en PDF. Pour des notes rapides ou collaboratives, un équivalent en Markdown (voir déjà le fichier `README`) reste plus léger à maintenir. Les deux formats peuvent coexister selon que l'on vise une diffusion interne (Markdown) ou un livrable formel (LaTeX/PDF).