Pair-Trading

Kerlaouezo Erwan

Février 2025

L'objectif de ce document est de présenter l'ensemble des étapes de notre stratégie de trading de paires de cryptomonnaies. Cette stratégie repose sur la détection de paires cointégrées, la calibration des paramètres optimaux, et l'exécution de trades basés sur les écarts de prix entre les actifs. La première étape consiste en la **récupération des données**, où nous collectons les prix historiques des cryptomonnaies depuis Binance. Ces données sont essentielles pour analyser la relation entre différentes paires d'actifs. Ensuite, nous effectuons un **prétraitement des données** afin de garantir leur qualité. Cela inclut le nettoyage des valeurs manquantes, l'uniformisation des formats et l'adaptation des séries temporelles. Une fois les données préparées, nous procédons à la **sélection des paires cointégrées**. Cette étape consiste à identifier les paires de cryptomonnaies qui présentent une relation statistique stable dans le temps, ce qui est essentiel pour le fonctionnement de notre stratégie. Après cela, nous passons à la **calibration des paramètres**, en optimisant les seuils d'entrée et de sortie des trades (par exemple, bound et stopLoss). Cette optimisation permet de maximiser la rentabilité tout en maîtrisant le risque. L'étape suivante est l'**exécution de la stratégie**, où nous appliquons notre modèle sur les données en temps réel afin d'ouvrir et fermer des positions en fonction des signaux générés. Enfin, nous réalisons un **suivi et une évaluation** de la stratégie. Nous analysons les performances des trades, ajustons les paramètres si nécessaire et vérifions la robustesse du modèle sur différentes périodes de marché. Dans ce document, on impose un code couleur qui est le suivant : les Objets python en vert, les Fonctions python utilisées en jaune et les Variables en bleu.

1 BinanceData.py

Ce document décrit le fonctionnement du code Python utilisé pour récupérer, traiter et stocker des données de différentes crypto-monnaies à partir de l'API Binance. La classe BinanceDataFetcher est conçue pour extraire des données de marché de Binance. Ses principales fonctionnalités comprennent :

- days : Nombre de jours de données à récupérer (par défaut 3 jours).
- tickers : Liste des crypto-monnaies concernées.
- timeframe : Périodicité des données (1 minute).
- output_dir : Répertoire de stockage des données.
- test_output_dir et backtest_output_dir : Répertoires pour les tests et backtests.

La fonction fetch_historical_data permet de récupérer les données OHLCV (Open, High, Low, Close, Volume) pour une crypto-monnaie donnée. On envoie des requêtes successives à l'API Binance, on stocke les résultats dans une liste et on gère les erreurs éventuelles liées aux requêtes API. Un délai est introduit pour éviter d'être bloqué par les restrictions de requêtes API.

La fonction run permet de télécharger ou mettre à jour les données existantes. On détermine la date de début des données à récupérer, on charge les fichiers existants pour éviter les doublons. On stocke les nouvelles données et les fusionne avec les anciennes et pour finir on sauvegarde les fichiers mis à jour.

La fonction adjustLenghtCSV harmonise la longueur des fichiers CSV afin qu'ils contiennent tous la même période d'historique. Pour cela on identifie le fichier contenant le moins de données, on tronque les autres fichiers à cette longueur minimale et on sauvegarde les fichiers modifiés.

La fonction split_data divise les données en deux jeux distincts :

- **Test** : Correspond aux premières 48 heures de données.
- **Backtest** : Correspond aux données historiques restantes.

Cette séparation permet d'utiliser les données récentes pour valider les stratégies de trading et les données plus anciennes pour entraîner ou backtester des modèles.

2 TestResiduals.py

Une fois les données récupérées, elles deviennent essentielles pour analyser la relation entre différentes paires d'actifs. Ensuite, nous effectuons un **prétraitement des données** afin de garantir leur qualité. Cela inclut le nettoyage des valeurs manquantes, l'uniformisation des formats et l'adaptation des séries temporelles. Une fois les données préparées, nous procédons à la **sélection des paires cointégrées**.

- dfs: Dictionnaire contenant les séries temporelles de prix pour chaque crypto-monnaie.
- level : Seuil de significativité utilisé pour le test de stationnarité des résidus, défini par défaut à 0.01.
- residuals_dict : Dictionnaire stockant les résidus de la régression linéaire entre chaque paire sélectionnée.
- alpha_beta_dict : Dictionnaire contenant les coefficients et des régressions linéaires effectuées pour chaque paire de crypto-monnaies analysée.
- numberOfPairs : Nombre total de paires de crypto-monnaies analysées.

La fonction analyze_and_find_cointegrated_pairs permet qu'à partir d'un ensemble initial $\mathcal{P}$ qui correspond à toutes les paires possibles entre les crypto-monnaies disponibles, nous analysons leur relation statistique. Pour chaque paire $(X, Y) \in \mathcal{P}$, nous identifions l'actif le plus cher X et l'actif le moins cher Y en fonction de leur prix moyen historique. Nous effectuons ensuite une régression linéaire où :

$$\forall t \in \{1, \ldots, T\}, \quad Y_t = \alpha + \beta X_t + \varepsilon_t \tag{1}$$

où $\varepsilon_t \in \mathcal{N}(0, \sigma_t)$ représente les résidus du modèle. La cointégration est validée si le test de stationnarité sur (test de Dickey-Fuller) donne une p-valeur inférieure à un seuil . Seules les paires vérifiant cette condition sont retenues dans $\mathcal{P}^* = (P_1, \cdots, P_N)$, l'ensemble des paires cointégrées. Nous expliquons systématiquement l'actif le moins cher Y par l'actif le plus cher X, ce qui a plusieurs implications importantes sur la stratégie de pair-trading. Premièrement, en choisissant toujours Y comme l'actif le moins cher, nous obtenons systématiquement des coefficients inférieurs à 1:

$$0 < \beta < 1.$$

Un $\beta < 1$ implique que les investissements sont en pratique plus réalisable.

3 BackTestResiduals.py

Cette partie permet uniquement de calculer les résidus des paires selectionnées par la partie de test à l'aide des coefficiants alpha et beta estimés dans la selections des bonnes paires. On organise le tout dans plusieurs dictionnaires initialisés dans l'objet nommé ResidualCalculator qui hérite de la classe BinanceDataFetcher pour avoir accès aux paths pour récupérer les données et qui s'initialise avec :

- alphas_betas_dict le dictionnaire des alphas et des betas estimés lors de la sélection des paires qui sont cointégrées.
- backtest_data les données de back test utilisées.
- backtestResiduals les résidus pour le back test calculés par la fonction calculate_residuals.

Avec tous ces objets, nous pouvons donc résumer ce que nous avons fait de la manière suivante : Soit N le nombre total de paires sélectionnées. Chaque paire $P_i \in \mathcal{P}^*$, avec $i \in \{1, \ldots, N\}$, est définie par le triplet :

$$P_i := (Y_i, X_i, \alpha_i, \beta_i, \varepsilon_i, Y_i^{'}, X_i^{'}, \varepsilon_i^{'}),$$

Où α_i et β_i sont les coefficients estimés de la régression linéaire associée à la paire P_i, $\varepsilon_i = (\varepsilon_{i,1}, \ldots, \varepsilon_{i,T})$ représente les résidus associés à la paire P_i, $X_i = (X_{i,1}, \cdots, X_{i,T})$ et $Y_i = (Y_{i,1}, \cdots, Y_{i,T})$ les valeurs des actifs sur la période T. On a alors que les résidus $\varepsilon_i^{'}$ sont calculés selon la relation :

$$\forall t \in \{1, \ldots, T^{'}\}, , \quad \varepsilon_{i,t}^{'} = Y_{i,t}^{'} - (\alpha_i + \beta_i X_{i,t}^{'}) \tag{2}$$

où $X_i^{'}$ et $Y_i^{'}$ désignent respectivement les valeurs des actifs pour le back test de notre stratégie sur une période $T^{'}$, et les $\varepsilon_i^{'}$ désignent les résidus des données utilisées pour le back test de notre stratégie à l'aide de α_i, β_i pour la paire P_i.

4 Strategy.py

Cette partie du code implémente la stratégie de *pair trading* en gérant l'ouverture et la fermeture des positions ainsi que l'évolution du capital et du profit. La classe Strategy hérite de BinanceDataFetcher pour accéder aux données de marché et aux fichiers de back-test. Elle est initialisée avec plusieurs paramètres clés permettant d'exécuter la stratégie sur un ensemble de paires sélectionnées. La classe inclut les éléments suivants :

- pairs : Dictionnaire contenant les paires d'actifs cointégrées sélectionnées.
- residuals : Dictionnaire des résidus des paires cointégrées.
- alpha_betas : Dictionnaire contenant les coefficients α et β associés aux régressions des paires.
- isBackTest : Booléen indiquant si la stratégie est exécutée en mode backtest.
- stopLoss : Seuil de stop-loss déterminant le niveau maximal de perte toléré avant clôture d'une position.
- initial_capital : Capital total alloué à la stratégie.

En complément, plusieurs structures de données sont mises en place pour suivre l'évolution des positions et des performances de la stratégie lors de l'initiation de l'objet Strategy:

- capital_per_pair : Dictionnaire contenant la répartition du capital sur chaque paire.
- pnl : Dictionnaire stockant l'évolution du PnL pour chaque paire.
- pnl_global : Liste contenant le PnL agrégé sur toutes les paires.
- open_trades : Dictionnaire des positions ouvertes en cours.
- trading_fee : Paramètre définissant les frais de transaction appliqués aux opérations de marché (0.1%).

La récupération des données se fait en fonction du mode d'exécution, si isBackTest est activé, les données sont chargées depuis le répertoire de backtest. Enfin, les résidus sont normalisés (normResidus) pour chaque paire afin d'être comparables entre elles :

$$\forall (i,t) \in \{1,\dots,N\} \times \{1,\dots,T\}, \quad \hat{\varepsilon}_{i,t} = \frac{\varepsilon_{i,t} - \mu_i}{\sigma_i}$$

où μ_i et σ_i sont respectivement la moyenne et l'écart-type des résidus historiques (données de test). Ainsi, la classe Strategy structure l'ensemble des paramètres et variables nécessaires à l'exécution d'une stratégie de *pair trading*, en assurant un suivi détaillé du capital et du PnL. L'objectif de la fonction StrategyForCalibration est d'évaluer la rentabilité globale de la stratégie de trading en sommant les profits et pertes (PnL) de toutes les paires sélectionnées dans $\mathcal{P}^*$. Cette fonction est utilisée pour calibrer les seuils *bound* et *stopLoss* afin d'optimiser la performance globale du portefeuille.

4.1 StrategyForCalibration()

Soit P^* l'ensemble des paires de cryptomonnaies sélectionnées après l'analyse de cointégration. Pour chaque paire $P_i \in P^*$, nous suivons une stratégie de réversion à la moyenne où nous entrons en position lorsque le résidu normalisé $\hat{\varepsilon}_{i,t}$ dépasse un seuil ***bound***, et nous clôturons la position lorsque :

$$\hat{\varepsilon}_{i,t} \approx 0, \text{ c'est-à-dire } |\hat{\varepsilon}_{i,t}| \leq 10^{-4}.$$
$$|\hat{\varepsilon}_{i,t}| \geq \textit{stopLoss}, \text{ perte maximale tolérée atteinte.}$$

Le capital total alloué à la stratégie est I_{total}, et il est réparti uniformément entre les N paires :

$$I_0 = \frac{I_{\text{total}}}{N}$$

Lorsque nous ouvrons une position, nous déterminons si nous devons acheter ou vendre en fonction de la variable ω :

$$\omega = \begin{cases} -1, & \text{si } \hat{\varepsilon}_{i,t} \geq \textit{bound} \\ +1, & \text{si } \hat{\varepsilon}_{i,t} \leq -\textit{bound} \end{cases}$$

Un $\omega = -1$ signifie que l'on va être vendeur sur Y_i et un $\omega = 1$ signifie que l'on va être acheteur de Y_i. Dans notre stratégie on choisit de manière totalement arbitraire d'investir dans chaque 100% du capital disponible pour chaque paire $P_i \in \mathcal{P}^*$, donc à chaque trade on souhaite investir I_0. La quantité investie dans chaque actif est normalisée par $(1+\beta_i)$ pour assurer que la somme des investissements correspond exactement au capital alloué:

$$q_X = \frac{\omega \beta_i I_0}{(1+\beta_i) X_{i,t}}, \quad q_Y = \frac{\omega I_0}{(1+\beta_i) Y_{i,t}}.$$

En considérant donc ces deux quantités d'investient dans chaque trade on a la garantie que l'on investi bien I_0, en effet on a :

$$|q_X X_{i,t}| + |q_Y Y_{i,t}| = \left|\frac{\omega\beta_i I_0}{(1+\beta_i)}\right| + \left|\frac{\omega I_0}{(1+\beta_i)}\right| = I_0.$$

À chaque fois que nous sommes dans un trade et qu'un $\hat{\varepsilon}_{i,t}$ satisfait une des deux conditions de sorties défnies au-dessus, alors on ferme la position. Si l'on suppose que pour une paire P_i on a k trades alors on note $\forall j \in \{1, \cdots, k\}$ le j PnL réalisé sur une paire P_i est calculé comme suit et on actualise le capital :

$$\begin{cases} \mathrm{PnL}_{j,i} = q_X(X_{i,t} - X_{i,\text{entrée}}) + q_Y(Y_{i,t} - Y_{i,\text{entrée}}), \\ I_{i,t+1} = I_{i,t} + \mathrm{PnL}_{j,i}. \end{cases}$$

Finalement, par itération sur les paires $P_i \in \mathcal{P}^*$, la fonction StrategyForCalibration retourne, pour une ***bound*** (b) et un ***stopLoss*** (s) qui forment un couple $\Theta = (b, s)$ donné, la somme des profits sur toutes les paires P_i avec k trades par paires :

$$C(\Theta) = \sum_{P_i \in P^*} \sum_{j=1}^{k} \mathrm{PnL}_{j,i}(\Theta). \tag{3}$$

4.2 Calibration()

La fonction Calibration cherche à déterminer les valeurs optimales de ***bound*** et ***stopLoss*** qui maximisent le PnL global, que l'on note $\Theta^* = (b^*, s^*)$. Ces paramètres définissent les seuils d'entrée et de sortie de la stratégie de trading, et leur optimisation est cruciale pour maximiser la performance de la stratégie. Nous définissons l'espace $\mathcal{D}$ des paramètres calibrables comme l'ensemble des couples (b, s) respectant les contraintes suivantes :

$$\mathcal{D} = \{(b, s) \in \mathbb{R}^2 \mid 0.5 \leq b \leq 4, \quad 0.6 \leq s \leq 5.5, \quad b < s\}.$$

Cet espace $\mathcal{D}$ représente toutes les combinaisons possibles de ***bound*** et ***stopLoss*** qui sont réalistes et applicables dans le cadre de notre stratégie. La contrainte $b < s$ garantit que le seuil de stop-loss est toujours supérieur au seuil d'entrée, ce qui est essentiel pour limiter les pertes. L'objectif de la fonction Calibration est donc de chercher à maximiser le PnL global $C(\Theta)$, où $\Theta = (b, s)$. En pratique, les algorithmes d'optimisation sont généralement conçus pour minimiser une fonction plutôt que pour la maximiser. Par conséquent, nous reformulons le problème en minimisant l'opposé du PnL :

$$\Theta^* = \underset{\Theta \in \mathcal{D}}{\operatorname{argmin}} \; -C(\Theta). \tag{4}$$

Pour résoudre ce problème d'optimisation, nous utilisons l'algorithme **Nelder-Mead**. Cet algorithme est particulièrement adapté à notre cas pour plusieurs raisons :

- Il ne nécessite pas de calcul de gradient, ce qui est avantageux car la fonction $C(\Theta)$ peut être complexe.
- Il est robuste et efficace pour des problèmes d'optimisation non linéaires comme le nôtre.
- Il permet de travailler directement dans l'espace des paramètres $\mathcal{D}$ tout en respectant les contraintes définies.

L'algorithme **Nelder-Mead** explore l'espace des paramètres $\mathcal{D}$ en utilisant un simplex (un ensemble de points) pour évaluer la fonction objectif $-C(\Theta)$ et converger vers le minimum. Une fois l'optimisation terminée, le couple optimal $\Theta^* = (b^*, s^*)$ est retourné, et ces valeurs sont utilisées pour exécuter la stratégie de trading. En résumé, la fonction Calibration permet de trouver les paramètres optimaux Θ^* qui maximisent le PnL global de la stratégie, tout en respectant les contraintes définies par l'espace $\mathcal{D}$. Cette étape est essentielle pour garantir que la stratégie est performante et adaptée aux conditions du marché. Après convergence de l'optimisation, les valeurs optimales $bound^*$ et $stopLoss^*$ sont obtenues et appliquées à la stratégie de trading. L'utilisation d'une calibration globale sur toutes les paires permet d'optimiser la rentabilité tout en garantissant une approche robuste et cohérente. Il est préférable de calibrer une unique *bound* et un unique *stopLoss* pour l'ensemble des paires de trading, plutôt que d'optimiser ces paramètres individuellement pour chaque paire. En effet, une calibration paire par paire conduit à une exécution très limitée des trades, ce qui rend l'optimisation des paramètres inefficace. Lorsqu'on optimise les paramètres *bound* et *stopLoss* individuellement, certaines paires effectuent très peu de transactions. Par conséquent, la calibration repose sur un échantillon de trades trop restreint, ce qui fausse l'estimation des paramètres et peut conduire à des valeurs inadaptées. En calibrant une *bound* et un *stopLoss* communs à toutes les paires, on considère l'ensemble des entrées et sorties de positions pour une meilleure estimation des paramètres en s'appuyant sur un volume plus important de données.

5 Résultats

Dans cette section, nous présentons les résultats de la stratégie pour différentes fenêtres de temps. Supposons que les données soient récupérées entre le **07-01-2025** et le **10-01-2025**. Nous procédons alors comme suit avec un capital total à **100**:

- Les données du **07-01-2025 au 08-01-2025** sont utilisées comme **données de test**. À partir de cet échantillon, nous sélectionnons les paires de cryptomonnaies cointégrées et calibrons le couple optimal de paramètres $\Theta^* = (b^*, s^*)$ en maximisant le PnL de la stratégie sur cette période.
- Les données du **09-01-2025 au 10-01-2025** sont utilisées comme **données de backtest**. La stratégie y est appliquée avec les bornes calibrées précédemment sur les paires cointégrées identifiées.

Nous considérons un portefeuille composé initialement de 20 cryptomonnaies. En prenant plusieurs fenêtres de données, nous observons que le pourcentage de paires cointégrées varie généralement entre 7% et 15%. Pour chaque période de temps étudiée, une page d'annexe est dédiée aux graphiques correspondant : La première partie de chaque page montre les performances de la stratégie sur les données de test et la seconde partie affiche les mêmes graphiques appliqués aux données de backtest.

Date		Θ^*		**Test**			**BackTest**		
Début	Fin	b	s	rdm(%)	Sharpe	MDD(%)	rdm(%)	Sharpe	MDD(%)
07-01-2025	10-01-2025	1.8344	3.8289	7.09	0.91	-0.86	2.39	0.86	-0.18
14-01-2025	17-01-2025	1.5468	3.1312	5.41	0.61	-0.62	-0.17	-0.03	-0.73
21-01-2025	24-01-2025	1.6426	4.8978	5.65	0.59	-1.59	2.15	0.58	-0.62
04-02-2025	07-02-2025	1.6018	4.6218	6.05	0.57	-0.78	1.21	0.24	-0.89
11-02-2025	14-02-2025	1.3984	4.1434	14.44	0.62	-0.25	1.15	0.61	-0.21
18-02-2025	21-02-2025	1.7045	3.3743	1.82	0.31	-0.61	0.36	0.21	-0.25
24-02-2025	27-02-2025	1.2111	3.5959	4.86	0.43	-1.69	0.15	0.04	-1.58

Table 1: Résultats de la stratégie pour différentes fenêtres temporelles

6 Conclusion

Dans cette étude, nous avons exploré une approche de pair trading appliquée aux cryptomonnaies, basée sur la cointégration de certaines paires d'actifs. Nous avons décrit l'ensemble du processus, depuis la récupération et la préparation des données, jusqu'à la sélection des paires cointégrées, la calibration des seuils de trading et l'évaluation des performances de la stratégie sur des données de test et de backtest.

Nos résultats montrent que la stratégie est capable de générer des rendements intéressants dans certaines configurations de marché. En optimisant les seuils b et s, nous avons pu améliorer la rentabilité globale en contrôlant mieux les entrées et sorties des positions. Toutefois, plusieurs axes d'amélioration peuvent être envisagés afin d'optimiser davantage cette approche.

- Tout d'abord, le choix des cryptomonnaies incluses dans l'analyse peut jouer un rôle crucial dans la performance de la stratégie. Actuellement, nous limitons notre univers d'investissement à seulement 20 cryptos, mais en élargissant ce choix à un plus grand nombre d'actifs, nous pourrions potentiellement identifier davantage de paires cointégrées et ainsi diversifier les opportunités de trading.
- Ensuite, un aspect important à considérer est la proportion de paires cointégrées identifiées lors de la phase de test. Nos analyses montrent que ce ratio varie entre 7% et 15% selon la période étudiée. Un problème majeur pourrait se poser si ce pourcentage devient trop faible : dans ce cas, le nombre de trades exécutés serait insuffisant, rendant la stratégie inefficace. Une solution envisageable serait d'introduire un seuil minimal de paires cointégrées sous lequel nous décidons de ne pas trader, afin d'éviter de baser la stratégie sur un échantillon trop restreint.
- Une autre piste d'amélioration concerne l'adaptation dynamique des seuils b et s. Actuellement, nous calibrons ces paramètres sur une fenêtre de test fixe et les appliquons directement au backtest. Cependant, il pourrait être intéressant d'ajouter une mise à jour progressive des seuils en fonction de l'évolution des résidus et des conditions de marché.
- Par ailleurs, la gestion du risque pourrait être renforcée. L'introduction de mécanismes supplémentaires, tels que des stop-loss dynamiques ajustés en fonction de la volatilité ou des limites de drawdown maximales, pourrait permettre de réduire les périodes de sous-performance.
- Enfin, il serait intéressant d'explorer d'autres critères de sélection des paires. Actuellement, nous utilisons un test de cointégration basé sur les résidus de régressions linéaires. D'autres méthodes, comme les modèles de séries temporelles avancés (VAR, Kalman filter) ou des tests de cointégration plus robustes, pourraient être envisagées pour améliorer la pertinence des paires sélectionnées.

En conclusion, bien que notre stratégie de *pair trading* sur les cryptomonnaies ait montré des résultats encourageants, plusieurs axes d'amélioration existent pour la rendre plus robuste et plus efficace. L'augmentation de l'univers d'investissement, l'introduction de filtres de sélection plus stricts, l'adaptation dynamique des seuils et l'amélioration des mécanismes de gestion des risques sont autant de pistes à explorer dans de futures recherches. Ces évolutions pourraient permettre d'optimiser davantage la rentabilité et la stabilité de la stratégie, tout en minimisant les risques liés aux fluctuations de marché inhérentes aux cryptomonnaies.

Annexe 1

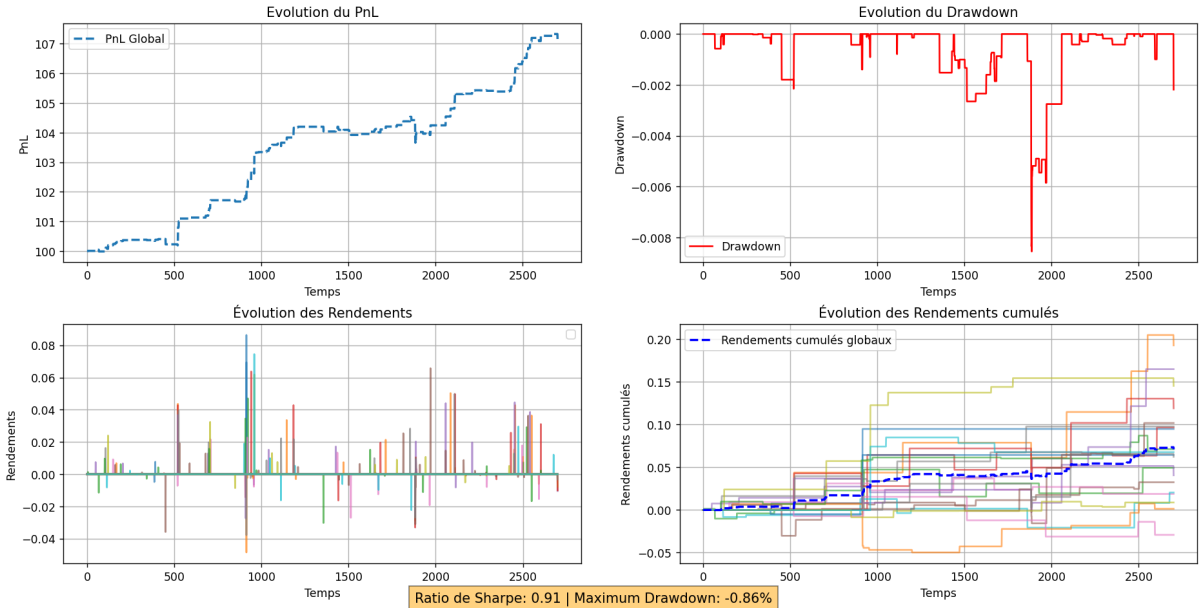

Figure 1: Schémas pour les données de test du **07-01-2025** au **08-01-2025**.

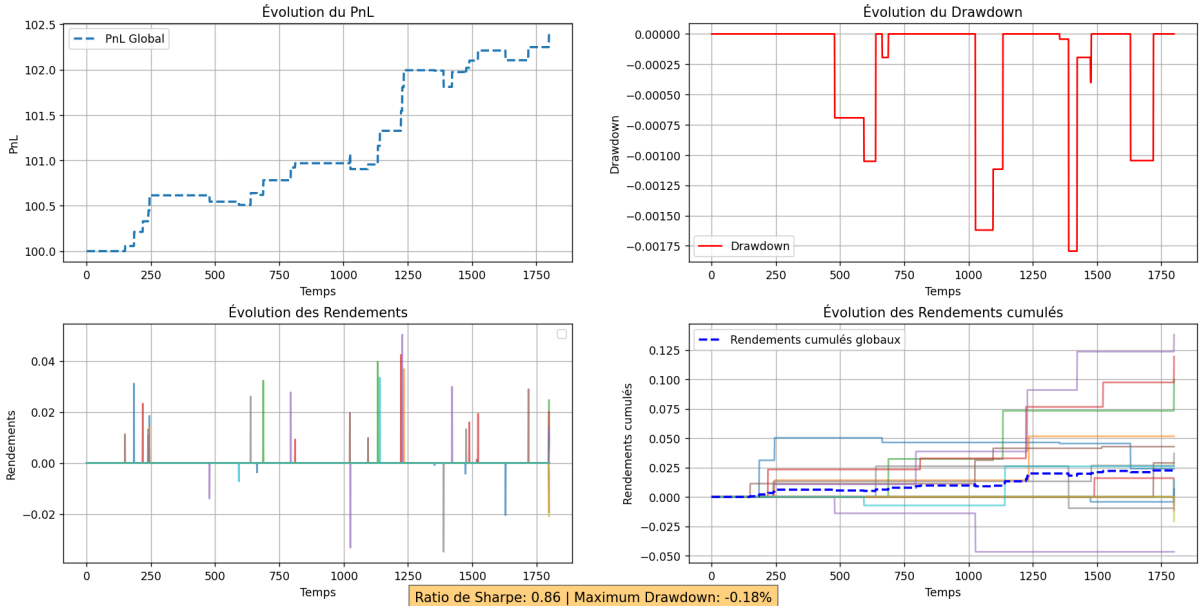

Figure 2: Schémas pour les données de back-test du **09-01-2025** au **10-01-2025**.

Annexe 2

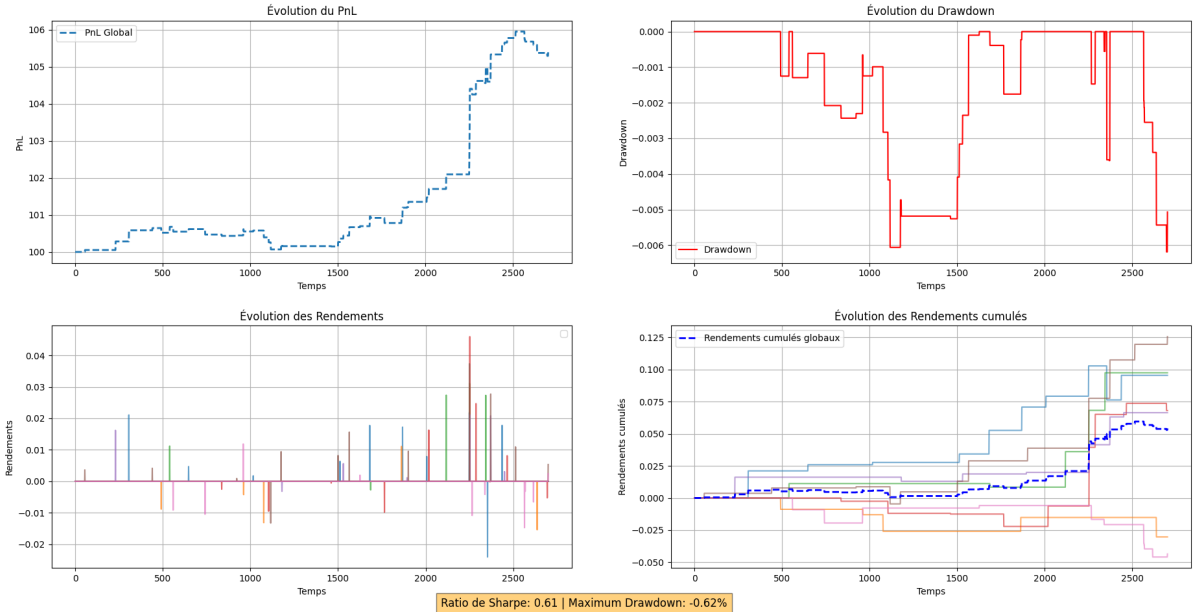

Figure 3: Schémas pour les données de test du **14-01-2025** au **15-01-2025**.

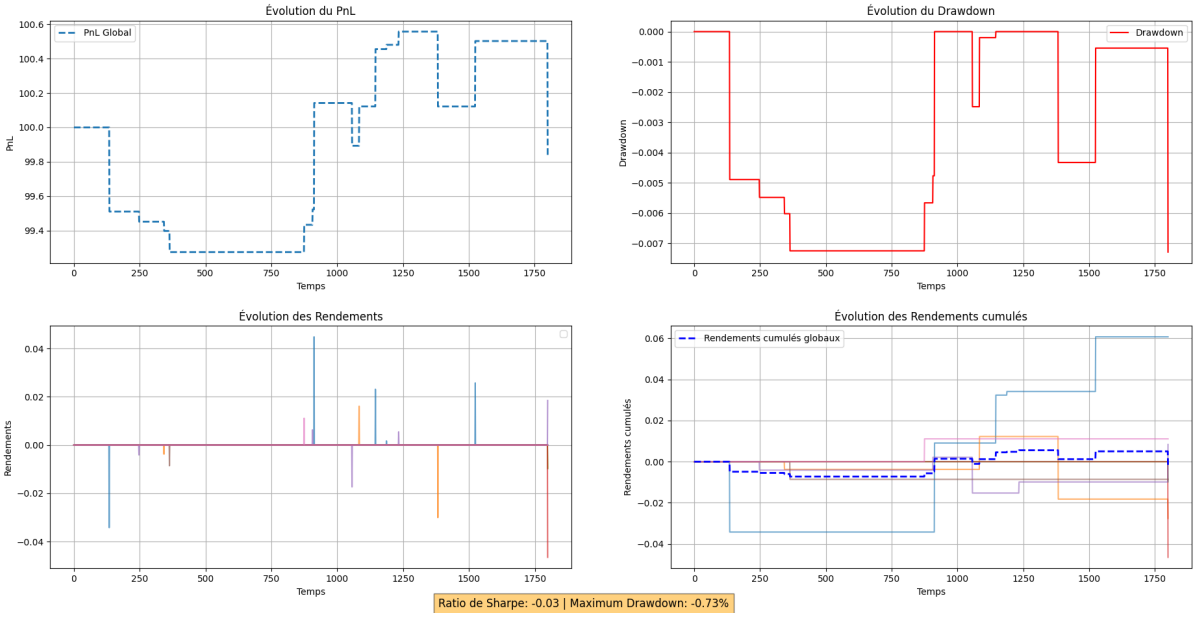

Figure 4: Schémas pour les données de back-test du **16-01-2025** au **17-01-2025**.

Annexe 3

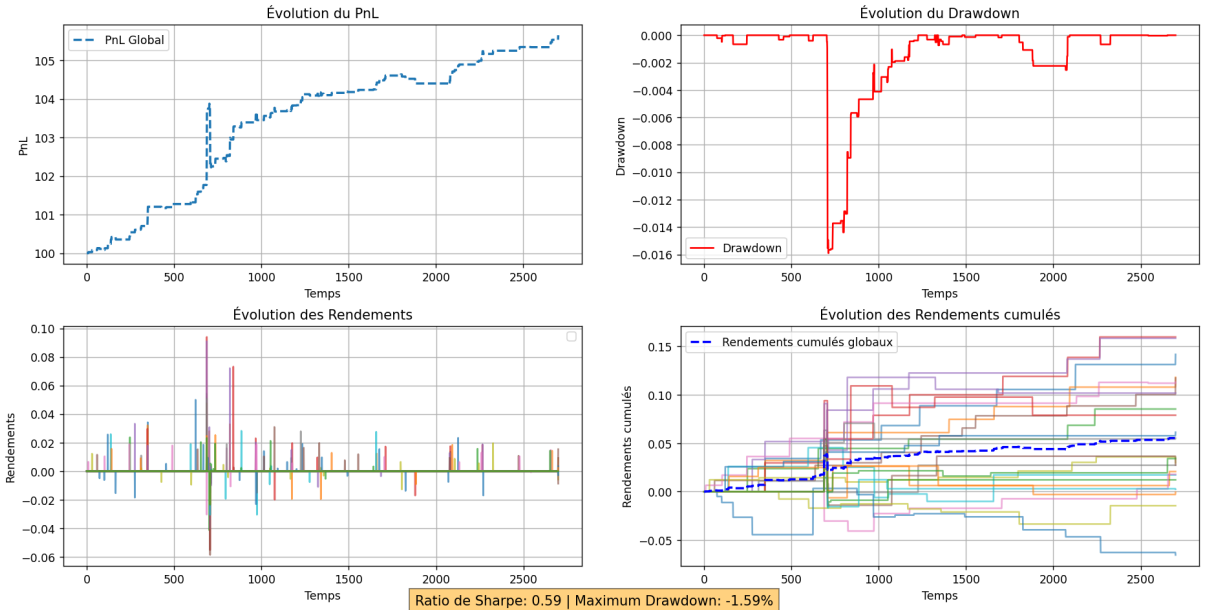

Figure 5: Schémas pour les données de test du **21-01-2025** au **22-01-2025**.

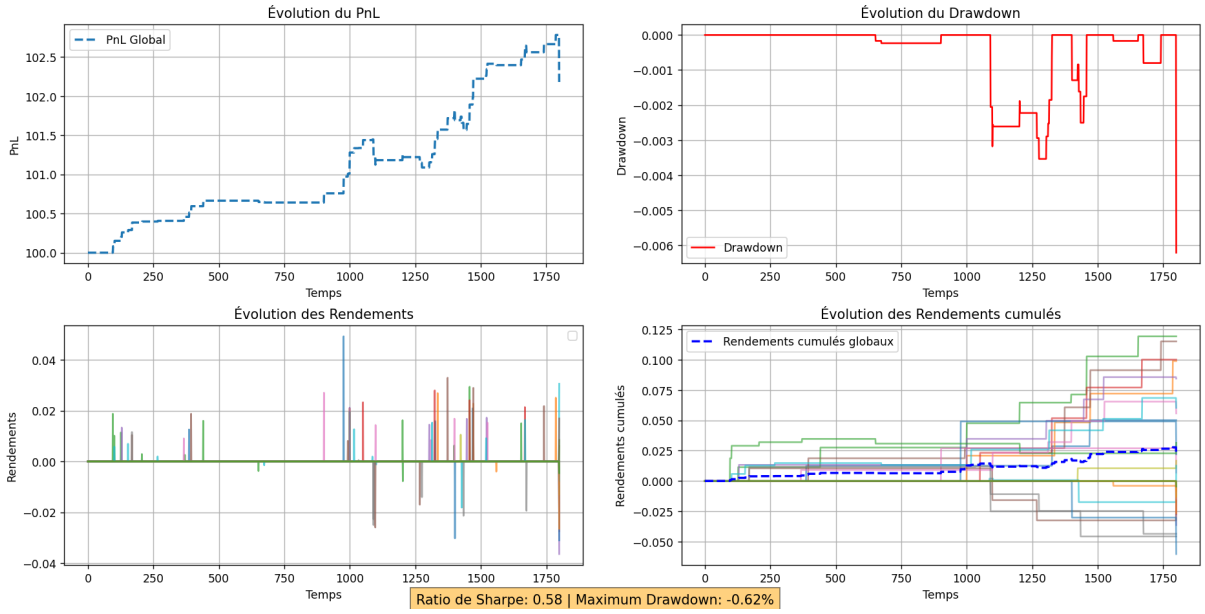

Figure 6: Schémas pour les données de back-test du **23-01-2025** au **24-01-2025**.

Annexe 4

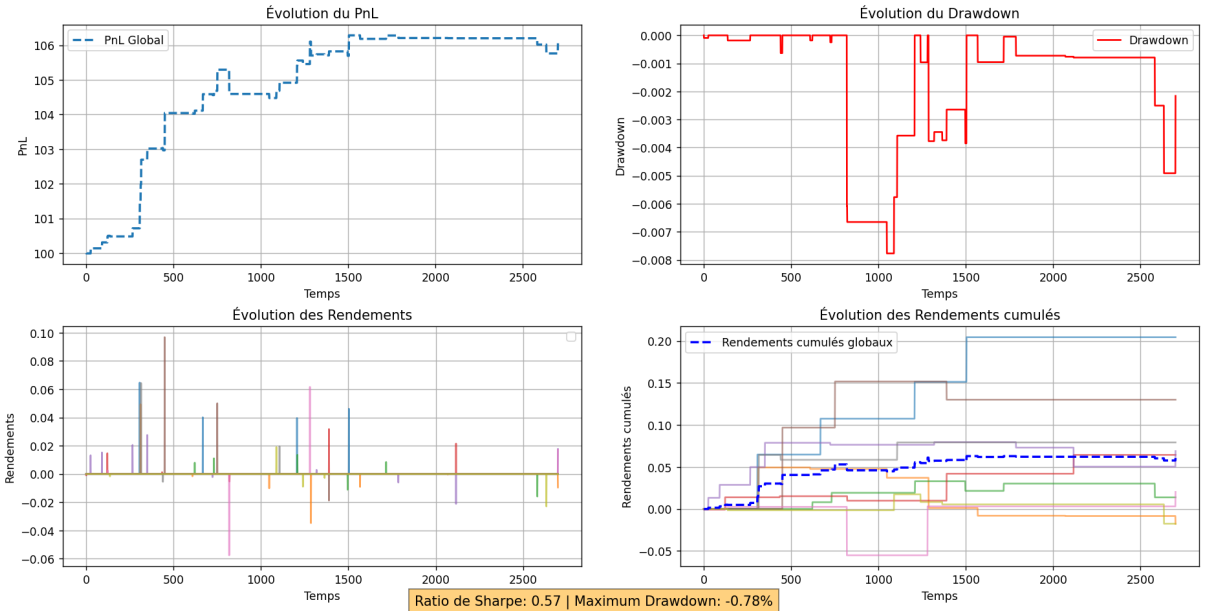

Figure 7: Schémas pour les données de test du **04-02-2025** au **05-02-2025**.

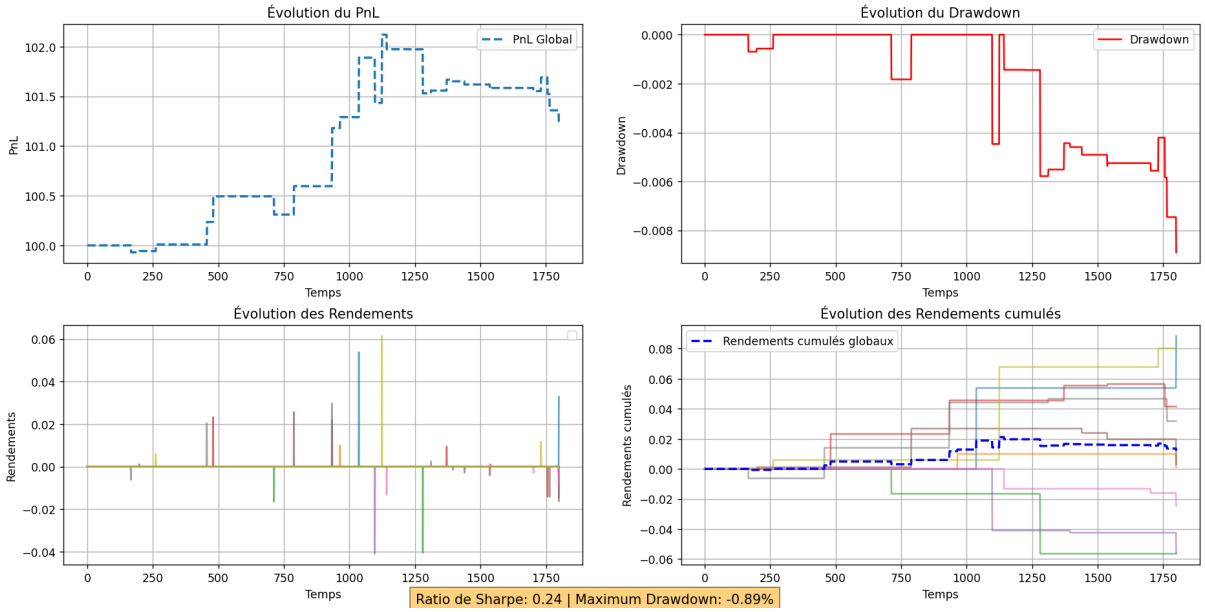

Figure 8: Schémas pour les données de back-test du **06-02-2025** au **07-02-2025**.

Annexe 5

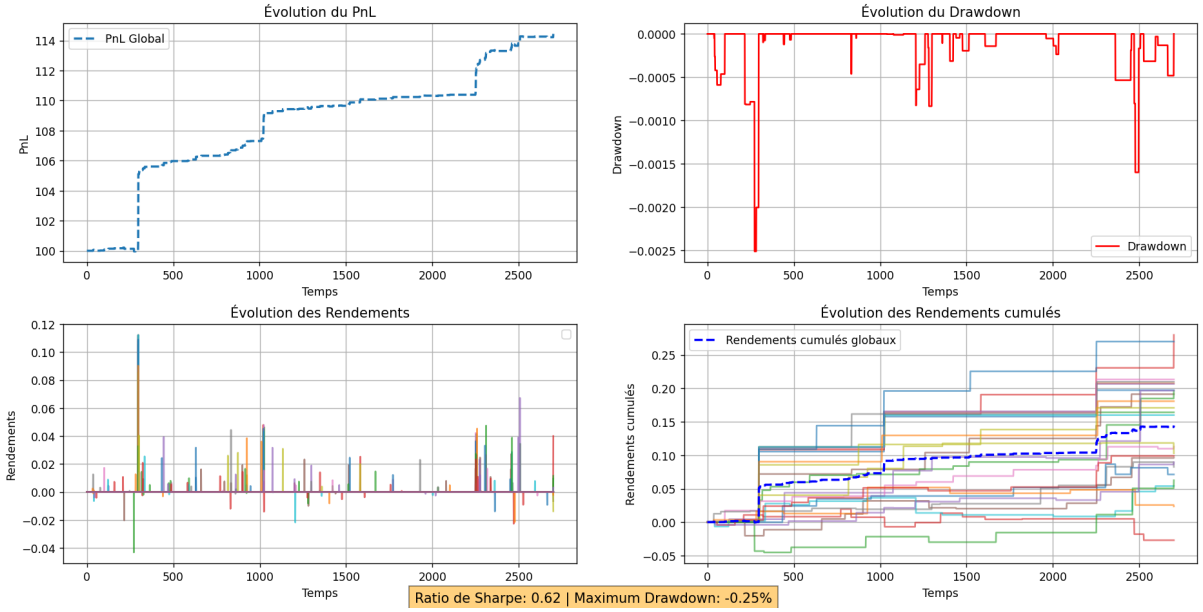

Figure 9: Schémas pour les données de test du **11-02-2025** au **12-02-2025**.

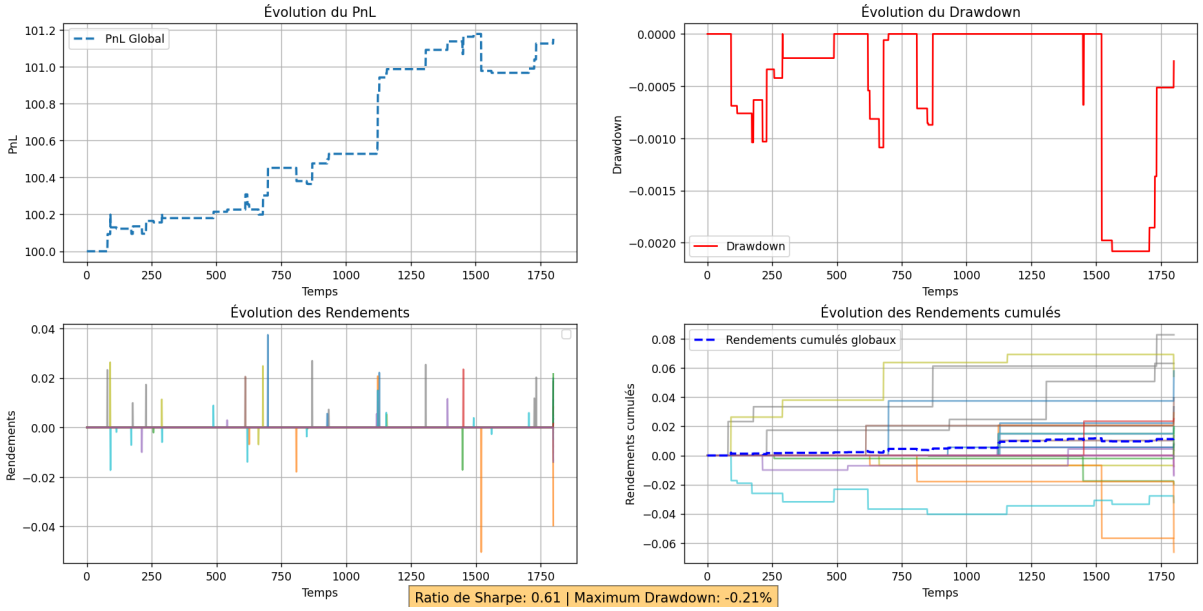

Figure 10: Schémas pour les données de back-test du **13-02-2025** au **14-02-2025**.

Annexe 6

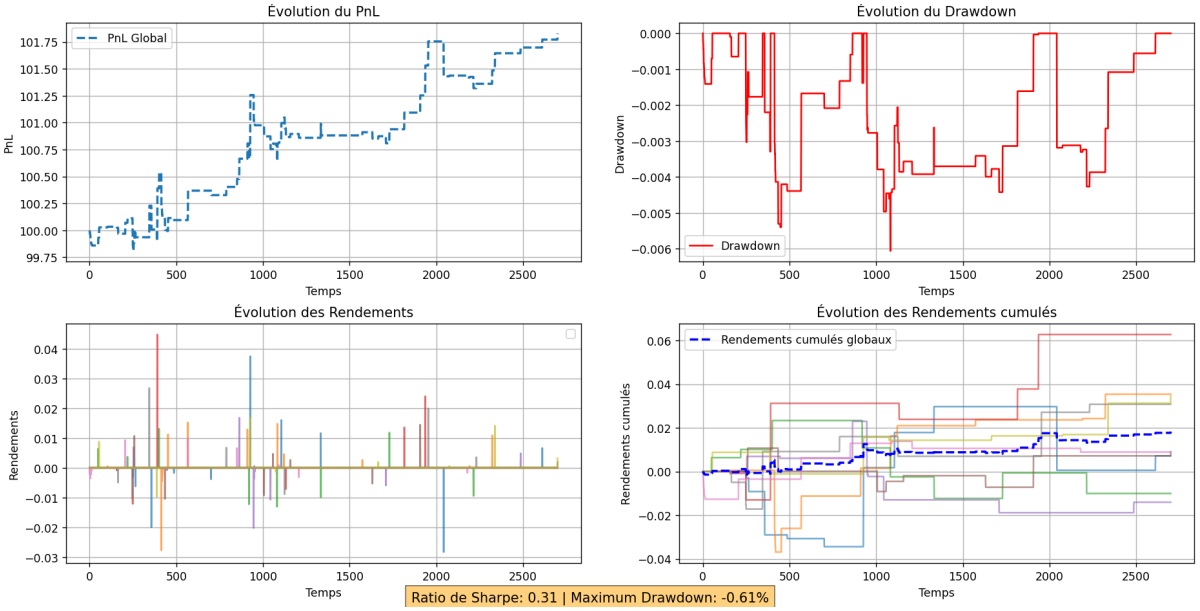

Figure 11: Schémas pour les données de test du **18-02-2025** au **19-02-2025**.

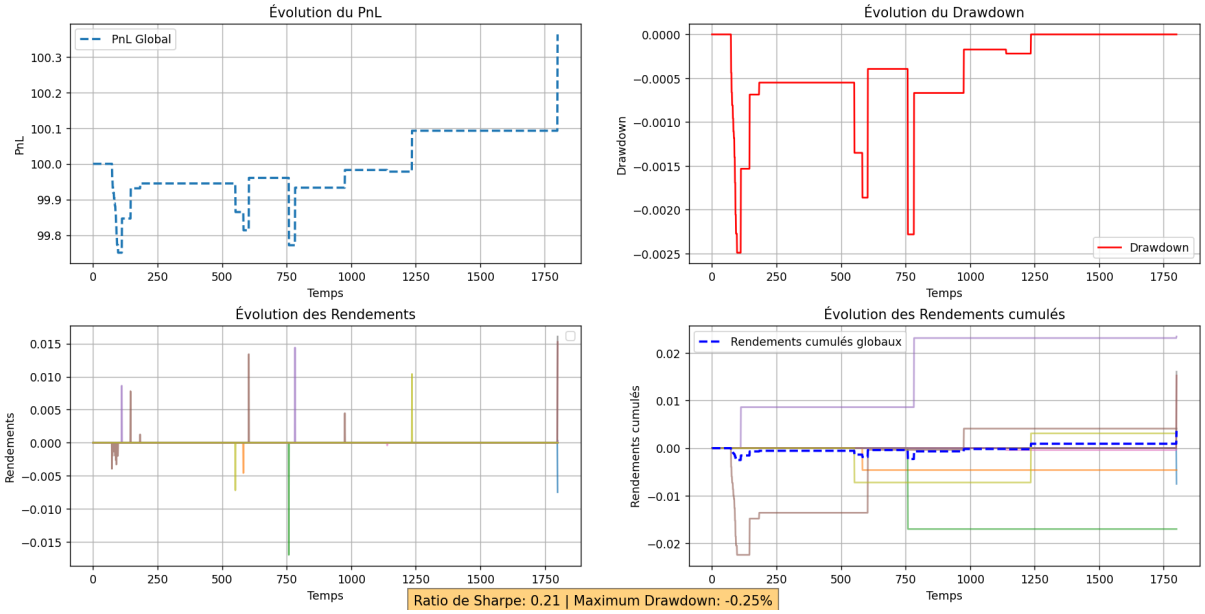

Figure 12: Schémas pour les données de back-test du **20-02-2025** au **21-02-2025**.

Annexe 7

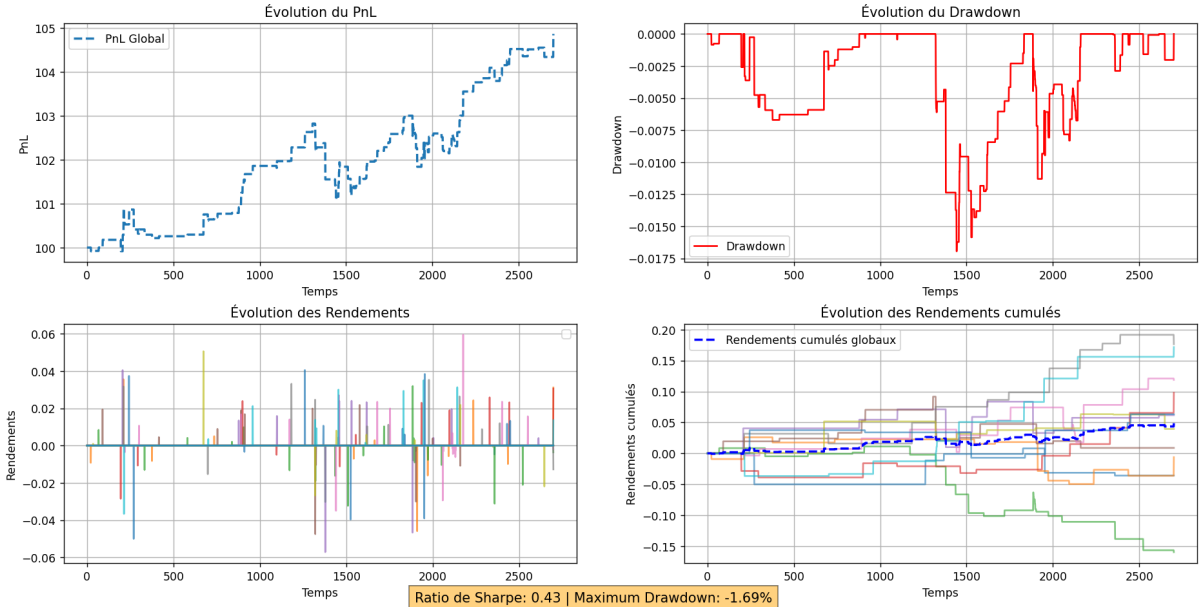

Figure 13: Schémas pour les données de test du **24-02-2025** au **25-02-2025**.

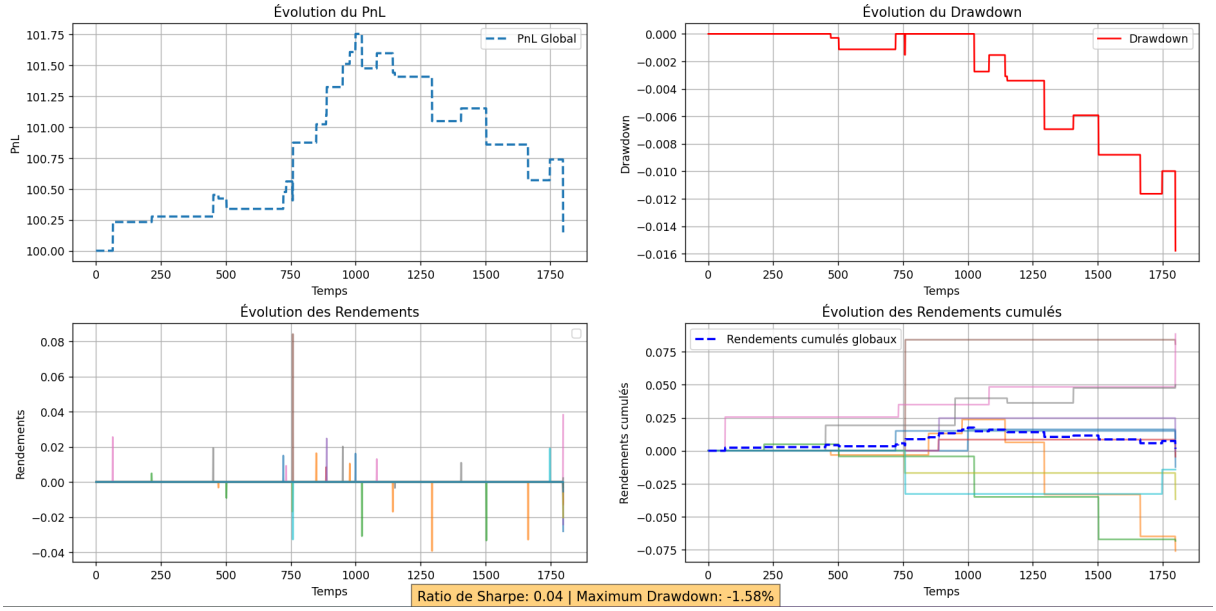

Figure 14: Schémas pour les données de back-test du **26-02-2025** au **27-02-2025**.