**Document Technique – Structures du Programme et Algorithmes**

**Alexandre Remiat – Jules Mourgues-Haroche – Walid Boudounit**

---

## 1. Architecture du Programme

Le projet est organisé en plusieurs modules interconnectés qui assurent le chargement des données, le calcul des indicateurs, l'application des stratégies d'investissement et la visualisation des résultats. Les principales classes et leurs responsabilités sont les suivantes :

**Database :**
Cette classe gère l'importation et le stockage des données historiques à partir d'un fichier CSV. Elle convertit la première colonne en date (format "yyyy-MM-dd HH:mm:ssK") et stocke les prix dans une structure de type dictionnaire (clé : Date, valeur : dictionnaire {Ticker → Prix}). Elle offre également des méthodes pour filtrer les données selon des intervalles temporels et pour calculer les log-returns.

**StrategyBase et ses dérivées (MomentumStrategy et ValueStrategy) :**
La classe abstraite *StrategyBase* définit la méthode `CalculateWeight(DateTime startDate)`, servant de base pour l'implémentation de stratégies d'investissement.

*MomentumStrategy* calcule les rendements sur une période de 12 mois (moins le dernier mois), sélectionne les 10 actifs les plus performants et les 10 moins performants, puis attribue des pondérations proportionnelles à ces rendements.

*ValueStrategy* évalue chaque actif en calculant un coefficient de valorisation, défini comme le prix actuel divisé par la différence entre le prix actuel et le prix d'il y a cinq ans. Les actifs sont classés selon ce coefficient, et les 10 actifs les plus faibles (indiquant une sous-évaluation) sont sélectionnés pour les positions longues, tandis que les 10 plus élevés sont retenus pour les positions courtes.

**Backtest :**
Cette classe orchestre l'exécution du backtesting en combinant les données issues de *Database* et les pondérations générées par les stratégies. Elle procède au filtrage des données par période (cycles économiques, phases d'inflation), détermine les dates de rebalancement mensuelles ( le premier jour de chaque mois) et simule quotidiennement l'évolution du portefeuille à partir des log-returns, en partant d'un capital initial fixe.

**Results :**
La classe *Results* calcule les indicateurs de performance du portefeuille, notamment le rendement annualisé, la volatilité annualisée (calculée à partir de l'écart-type des rendements journaliers multiplié par $\sqrt{252}$), le ratio de Sharpe et le rendement total sur la période.

**Plotter :**
Utilisant la librairie ScottPlot, cette classe génère et sauvegarde des graphiques illustrant la performance du portefeuille (courbes de performance, histogrammes, scatter plots). Elle inclut également une fonction d'export des résultats dans un fichier CSV.

---

## 2. Structures de Données et Algorithmes

### Structures de Données

- **Dictionnaire pour les Données Historiques :**
  Le stockage des données se fait sous la forme d'un dictionnaire dont la clé est une date et la valeur est elle-même un dictionnaire associant chaque ticker à son prix pour cette date. Cette structure permet un accès rapide et une manipulation aisée des données lors du filtrage et du calcul des log-returns.
- **Listes et Dictionnaires pour les Rendements et Pondérations :**
  Les log-returns sont stockés dans des dictionnaires indexés par date, facilitant le calcul des indicateurs de performance. Les pondérations calculées par les stratégies sont également stockées sous forme de dictionnaires {Ticker → Poids}, normalisés pour que la somme des valeurs absolues soit égale à 1.

### Algorithmes de Calcul

- **Calcul des Log-Returns :**
  Pour chaque date, si une date précédente est disponible, le log-return est calculé selon

  Log-return= ln(Pt/Pt_1) avec pt le prix à l'instant t et pt_1 le prix à l'instant t-1.

  Le résultat est stocké dans le dictionnaire de log-returns pour une utilisation ultérieure dans la simulation.

- **Calcul des Pondérations pour Momentum :**
  Les rendements sur une période de 12 mois (moins le dernier mois) sont calculés pour chaque actif. Les actifs sont ensuite triés par ordre décroissant de rendement. Les 10 meilleurs et 10 pires actifs sont sélectionnés. Chaque actif se voit attribuer un poids proportionnel à son rendement, avec une normalisation finale pour garantir que la somme des poids absolus est égale à 1.
- **Calcul des Pondérations pour Value :**
  Pour chaque actif, le coefficient de valorisation est calculé par la formule :
  "Coefficient = prix actuel divisé par (prix actuel moins le prix d'il y a cinq ans)".
  Les actifs sont triés par ce coefficient et les 10 plus faibles (long) et 10 plus élevés (short) sont sélectionnés. Les pondérations sont attribuées proportionnellement aux coefficients puis normalisées.
- **Optimization des poids sous contrainte min et max :**
  Cette optimisation applique des contraintes aux poids d'un portefeuille d'investissement. Elle sépare d'abord les positions longues et courtes, puis applique une méthode itérative pour limiter chaque poids à une valeur minimale et maximale spécifiée. Si certains poids dépassent ces bornes (0.5% et 20% en valeur absolue), l'excédent est redistribué aux autres actifs non contraints. Une fois ces ajustements effectués, les positions courtes retrouvent leur signe négatif et l'ensemble des poids est renormalisé pour que la somme de leurs valeurs absolues soit égale à 1. Cela garantit un portefeuille bien équilibré, évitant des pondérations extrêmes tout en conservant l'exposition globale.
- **Simulation du Backtest :**
  La simulation repose sur un algorithme de walk-forward, qui met à jour quotidiennement la valeur du portefeuille. À chaque date de rebalancement, les

pondérations sont recalculées, et la valeur du portefeuille est ajustée en appliquant les log-returns de la journée correspondante.

---

## 3. Intégration et Visualisation

- **Interface de Reporting :**
  Les résultats du backtest (indicateurs de performance et évolution du portefeuille) sont affichés en console et exportés dans un fichier CSV pour une analyse détaillée.
- **Génération de Graphiques :**
  Grâce à la classe *Plotter* et à la librairie ScottPlot, des graphiques illustrant la performance quotidienne, la distribution des rendements et d'autres analyses visuelles sont générés et sauvegardés dans un répertoire prédéfini.

## 3. Tests unitaires de validations

Les tests unitaires, implémentés avec NUnit, garantissent la robustesse du programme. Ils couvrent plusieurs aspects critiques du système :

- **Tests de la classe `Database`** :
  - Vérification du chargement des données (`LoadData`) avec un fichier valide et gestion des exceptions en cas de fichier introuvable.
  - Extraction correcte des données (`GetData`) sur un intervalle de dates donné.
  - Vérification des rendements logarithmiques (`CalculateLogReturns`), s'assurant que les données en sortie ne sont pas vides.
  - Test de récupération de la date la plus proche (`GetClosestDate`), garantissant que la date retournée est bien un jour ouvré.
- **Tests des stratégies d'investissement (`MomentumStrategy`, `ValueStrategy`)** :
  - Validation que `MomentumStrategy` sélectionne correctement les actifs en fonction des performances passées et que la somme des poids alloués est normalisée à 1.
  - Vérification que `ValueStrategy` gère correctement les périodes de données manquantes.
- **Tests du moteur de simulation `Backtest`** :
  - Vérification que la simulation fonctionne sans erreurs sur une période valide.
  - Gestion correcte des cas où les données sont insuffisantes ou absentes.
  - L'algorithme d'optimisation des poids doit réussir à clipper aux bornes les actifs sur où sous-pondérés.
- **Tests des résultats (`Results`)** :
  - Validation du calcul des rendements totaux et annualisés.
  - Vérification de la volatilité annualisée et de son calcul cohérent.
  - Test des cas limites où l'historique contient un nombre insuffisant de points de données.