Projet de Backtesting en C++ : Implémentation d'un Backtester de Trading Quantitatif

1 Contexte

Dans le cadre de la gestion quantitative, les stratégies systématiques nécessitent une validation rigoureuse avant leur déploiement. Le backtesting permet de simuler ces stratégies sur des données historiques pour évaluer leur performance potentielle.

2 Objectif du Projet

Développer un backtester en C++ permettant de tester une stratégie de trading. Le système devra être modulaire et extensible pour permettre l'ajout futur d'autres stratégies.

3 Structure du Projet

3.1 Organisation des Répertoires

Le projet sera organisé selon une structure de répertoires hiérarchique permettant une séparation claire entre les fichiers d'en-tête et les fichiers d'implémentation. La structure détaillée est présentée en **Annexe A**.

3.2 Makefile

Un Makefile est un fichier de configuration utilisé par l'utilitaire `make` pour automatiser le processus de compilation d'un projet. Son utilisation présente plusieurs avantages essentiels (documentation) :

— **Automatisation** : Évite d'avoir à taper manuellement les commandes de compilation
— **Compilation sélective** : Recompile uniquement les fichiers modifiés et leurs dépendances
— **Reproductibilité** : Garantit que le processus de build est identique pour tous les développeurs
— **Configuration centralisée** : Centralise les paramètres de compilation (flags, chemins, etc.)

Un exemple complet de Makefile adapté à la structure du projet est disponible en **Annexe B**.

4 Implémentation du Backtester

4.1 Lecture des Données de Marché

4.1.1 Format des Données d'Entrée

Les données de marché seront fournies au format CSV avec la structure suivante :

```
Date,Open,High,Low,Close,Volume
2024-01-01,150.25,152.30,149.80,151.20,1000000
2024-01-02,151.50,153.40,150.90,152.80,1200000
```

La récupération des données et leur mise en forme n'est pas fournie. Le script est libre d'implémentation (langage et packages utilisés).

4.1.2 Lecture des Données dans le Backtester

Pour implémenter la lecture des fichiers CSV, nous utiliserons les outils de base fournis par la bibliothèque standard C++ :

- Lecture de fichiers avec `std::ifstream`
- Manipulation de chaînes avec `std::string`
- Conteneurs appropriés de la STL
- Des structures ou classes afin de faire transiter les données dans le backtester

Ne pas hésiter à consulter CPPReference.

4.2 Le cœur du backtester

L'objectif est d'injecter les données à la stratégie et de récupérer ses décisions de trading à chaque étape.

Afin de faciliter la gestion des positions, la stratégie pourra être Flat, Short ou Long. Le backtester devra gérer le portfolio, la gestion de la position et générer les trades résultants. Ces informations seront utilisées à la prochaine étape afin de calculer les métriques.

N'oubliez pas de concevoir les stratégies selon un contrat d'interface uniforme pour garantir leur compatibilité avec le backtester. La programmation orientée objet est particulièrement adaptée à ce cas d'usage, en exploitant les concepts d'héritage et de classes abstraites pour créer un système modulaire et extensible.

4.3 Calcul des métriques

Afin d'évaluer la qualité de notre stratégie, nous allons calculer un certain nombre de métriques :

- PnL
- Turnover
- Ratio de Sharpe
- Ratio de Sortino
- Drawdown

5 Axes d'amélioration

— Gestion plus fine des positions (ex : pouvoir short ou long un certain nombre de contrats).

— Ajout d'un fichier de configuration '.ini' afin de paramétrer le backtester.

— Pouvoir rejouer plusieurs paramètres de stratégies en même temps (cf. multithreading).

— Stocker les résultats afin de pouvoir les interpréter ultérieurement (ex par un jupyter notebook et plotly).

A Structure des Répertoires

La structure suivante présente l'organisation proposée pour les fichiers et répertoires du projet (les fichiers ne sont qu'à titre d'exemple) :

```
backtester/
|-- include/
|   |-- core/
|   |   |-- Asset.h
|   |   |-- Portfolio.h
|   |   `-- Types.h
|   |-- data/
|   |   |-- DataLoader.h
|   |   `-- CSVReader.h
|   |-- strategy/
|   |   |-- Strategy.h
|   |   `-- MomentumStrategy.h
|   `-- utils/
|       |-- Logger.h
|       `-- Constants.h
|-- src/
|   |-- core/
|   |   |-- Asset.cpp
|   |   `-- Portfolio.cpp
|   |-- data/
|   |   |-- DataLoader.cpp
|   |   `-- CSVReader.cpp
|   |-- strategy/
|   |   |-- Strategy.cpp
|   |   `-- MomentumStrategy.cpp
|   `-- utils/
|       `-- Logger.cpp
`-- Makefile
```

Cette organisation assure une séparation claire entre les interfaces (fichiers .h dans le répertoire include/) et les implémentations (fichiers .cpp dans le répertoire src/), facilitant ainsi la maintenance et la lisibilité du code.

B Exemple de Makefile

Voici un exemple de Makefile simplifié, adapté à un projet avec un seul fichier source `main.cpp` :

```
# Compilateur et options
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -O2
INCLUDES = -Iinclude

# Fichier source et executable
SRC = src/main.cpp
TARGET = backtester

# Regle par defaut
all: $(TARGET)

# Compilation de l'executable
$(TARGET): $(SRC)
	$(CXX) $(CXXFLAGS) $(INCLUDES) -o $@ $<

# Nettoyage
clean:
	rm -f $(TARGET)

# Execution
run: all
	./$(TARGET)

# Indique que ces cibles ne sont pas des fichiers
.PHONY: all clean run
```

Ce Makefile simplifié inclut :

— Configuration de base du compilateur C++ avec les options standard

— Compilation directe du fichier source principal en un exécutable

— Commandes pour nettoyer et exécuter le projet

Lors du développement du projet, vous pourrez enrichir ce Makefile en fonction de l'évolution de la structure des fichiers.