Basic Stock Trading Strategy Program

1.1 Basic Strategy Overview

This program implements a basic stock trading strategy based on the momentum of stock prices over the past `n` days. The algorithm operates as follows:

- **Buy**: If the stock price has been monotonically increasing for the last `n` days, the program buys one share.
- **Sell**: If the stock price has been monotonically decreasing for the last `n` days, it sells one share.

Constraints and Assumptions

- **Short Selling**: The program assumes the ability to short-sell stocks.
- **Position Limits**: The maximum and minimum positions are `+x` and `-x` respectively, ensuring the position always stays within this range.
- **Data Preparation**: The program requires past `n` days of data even for the start date. This data must be pre-written to the file.
- **Parameters**:
  - `n`: Number of past days to consider for momentum calculation.
  - `x`: Maximum position allowed.

Intuition Behind the Strategy

The strategy is based on the expectation that a continuous increase in price over the last `n` days indicates a further increase, prompting a buy action. Conversely, a continuous decrease suggests a future decline, leading to a sell action.

Implementation Details

StockData Class

- **Purpose**: Represents individual stock data with date, price, and trading action.
- **Key Methods**:
  - `buy()`: Executes a buy action.
  - `sell()`: Executes a sell action.
  - `readFromFile()`: Reads stock data from a file.

Main Functionality

- **File Reading**: Reads stock data from a specified file.
- **Trading Logic**: Implements the buy/sell strategy based on price momentum.
- **Output**:
  - "order_statistics.csv": Contains order details.
  - "daily_cashflow.csv": Records daily cash flow.

Auxiliary Functions

- `getLastNElements()`: Retrieves the last `n` elements from a vector of `StockData`.

Execution

Run the program with the following command format:

```
   make strategy=BASIC symbol=SBIN n=5 x=2 start_date="01/01/2023"
end_date="01/01/2024"
```

Lab Report: Implementation of Trend-based Strategy using n-Day Moving Average (DMA)

Introduction

In this project, we extend the basic stock trading strategy by incorporating a trend-based strategy using the n-Day Moving Average (DMA). This approach involves calculating the mean price of the past `n` days, along with the standard deviation, to make informed trading decisions.

Implementation

Class Design: `StockData`

- **Purpose**: Represents individual stock data, including date, price, and calculated DMA.
- **Key Features**:
  - `buy()` and `sell()` methods to record trading actions.
  - `setdma()` to update DMA for each stock data instance.
  - Additional methods for data management and retrieval.

File Reading

- Implemented in `readFromFile()` method.
- Reads stock data from a CSV file and stores it in a vector of `StockData` objects.

DMA Calculation

- For each data point, calculates the mean (DMA) and standard deviation of the past `n` days' prices.
- These calculations are pivotal for the decision-making process in the trading strategy.

Trading Logic

- **Buying Criterion**: If the current price is greater than DMA by ≥ `p` standard deviations, buy one share.
- **Selling Criterion**: If the current price is less than DMA by ≥ `p` standard deviations, sell one share.
- **Position Constraints**: Ensures the position always stays within the range [-x, +x].

Output Generation

- Generates "order_statistics.csv" and "daily_cashflow.csv".

- These files record the details of each trade and the daily cash flow, respectively.

Constraints and Assumptions

- The strategy assumes the capability to short-sell stocks.
- The maximum and minimum positions are capped at `+x` and `-x`.

Intuition Behind the Strategy

- This strategy is based on the premise that crossing the DMA is indicative of a potential trend.
- Using the standard deviation threshold `p` adds a layer of confidence in trend identification.

Parameters

- `n`: Number of days for calculating the DMA.
- `x`: Maximum position allowed.
- `p`: Standard deviation threshold for decision making.

Execution Command

```
make strategy=DMA symbol=SBIN n=50 x=3 p=2 start_date="01/01/2023"
end_date="01/01/2024"
```

Lab Report: Enhancing the DMA Strategy with Stop-Loss and Smoothing Factor

Introduction

This project aims to refine the DMA-based stock trading strategy by introducing two new aspects: a stop-loss mechanism and an Adaptive Moving Average (AMA) calculation. These enhancements are designed to mitigate risks and adapt to market volatility more effectively.

Implementation Details

`StockData` Class Enhancements

- **New Attributes**: Efficiency Ratio (ER), Smoothing Factor (SF), and Adaptive Moving Average (AMA).
- **Methods**: Updated to handle the new attributes and trading logic based on AMA.

File Processing

- Continues to read and process stock data from a file using `readFromFile()`.

Adaptive Moving Average (AMA) Calculation

- **Efficiency Ratio (ER)**: Measures price change efficiency over `n` days.

- **Smoothing Factor (SF)**: Dynamically adjusts based on ER to weigh recent price changes more heavily.
- **AMA Calculation**: Combines SF with price data to create a more responsive moving average.

Trading Logic

- **Buy**: When the current price exceeds AMA by ≥ `p` percent.
- **Sell**: When the current price is below AMA by ≥ `p` percent.
- **Stop-Loss**: Forcefully closes positions after `max hold days` if they haven't been closed otherwise.

Output Files

- "order_statistics.csv": Details each trade executed.
- "daily_cashflow.csv": Records the daily cash flow.

Algorithm Explanation

1. **ER Calculation**: Measures the directional movement efficiency of stock prices.
2. **SF Updating**: Adjusts dynamically based on ER, starting from an initial value of 0.5.
3. **AMA Computation**: Provides a more nuanced moving average by incorporating recent price trends.
4. **Trade Decisions**: Based on AMA and price deviation, along with position management considering max holding days.

Constraints and Assumptions

- Short-selling is possible.
- Position limits are between `-x` and `+x`.
- The strategy aims to respond to market trends while minimizing risks through stop-loss.

Command for Execution

```
make strategy=DMA++ symbol=SBIN x=4 p=5 n=14 max_hold_days=28 c1=2 c2=0.2
start_date="01/01/2023" end_date="01/01/2024"
```

Lab Report: Implementation of MACD Trading Strategy

Introduction

This project involves implementing the Moving Average Convergence Divergence (MACD) trading strategy in C++. MACD is a trend-following momentum indicator that shows the relationship between two moving averages of a security’s price.

Implementation Details

`StockData` Class Enhancements

- **Attributes**: Includes Short and Long Exponential Weighted Mean (EWM), MACD value, and Signal line.
- **Trading Actions**: Updated to make decisions based on MACD and Signal line.

MACD Calculation

- **Short EWM**: Exponential Weighted Mean for the past 12 days.
- **Long EWM**: Exponential Weighted Mean for the past 26 days.
- **MACD Value**: Calculated as Short EWM - Long EWM.
- **Signal Line**: EWM of MACD for the past 9 days.

Trading Logic

- **Buy Signal**: If MACD is greater than the Signal line.
- **Sell Signal**: If MACD is less than the Signal line.

File Processing

- Data read from a file using `readFromFile()`.
- Data processed to calculate Short EWM, Long EWM, MACD, and Signal line.

Output Files

- "order_statistics.csv": Records details of each trade.
- "daily_cashflow.csv": Tracks daily cash flow.

Algorithm Explanation

1. **EWM Calculation**: For 12 and 26 days to calculate Short EWM and Long EWM.
2. **MACD Computation**: Difference between Short EWM and Long EWM.
3. **Signal Line Calculation**: EWM of MACD for past 9 days.
4. **Trading Decisions**: Based on the comparison between MACD and Signal line.

Constraints and Assumptions

- Consistent with the MDA strategy, allowing short selling and having position limits.

Command for Execution

```
make strategy=MACD symbol=SBIN x=3 start_date="01/01/2023"
end_date="01/01/2024"
```

Lab Report: Implementation of the Relative Strength Index (RSI) Trading Strategy

Introduction

The Relative Strength Index (RSI) is a popular momentum indicator used in stock trading. This project involves implementing the RSI strategy in C++, which helps in identifying overbought and oversold conditions in the stock market.

Implementation Details

`StockData` Class Enhancements

- **New Attributes**: Includes Average Gain, Average Loss, Relative Strength (RS), and RSI.
- **Methods**: Updated to calculate RSI and make trading decisions based on RSI thresholds.

RSI Calculation

- **Average Gain and Loss**: Calculated over the last `n` days.
- **Relative Strength (RS)**: Ratio of Average Gain to Average Loss.
- **RSI Value**: Computed using the formula `RSI = 100 - (100 / (1 + RS))`.

Trading Logic

- **Buy Signal**: Generated when RSI crosses below the `oversold_threshold`.
- **Sell Signal**: Generated when RSI crosses above the `overbought_threshold`.

File Processing

- Data is read from a file using `readFromFile()` and processed for RSI calculation.

Output Files

- "order_statistics.csv": Records trade details.
- "daily_cashflow.csv": Tracks the daily cash flow.

Algorithm Explanation

1. **Gain and Loss Calculation**: Determines the average gain and loss over the specified period.
2. **RSI Computation**: Calculates the RSI based on RS, indicating overbought or oversold conditions.
3. **Trading Decisions**: Based on RSI crossing predefined thresholds.

Constraints and Assumptions

- Similar to the DMA strategy, allowing for short selling and position limits.
- The `overbought_threshold` is set to be greater than or equal to the `oversold_threshold`.

Command for Execution

```
make strategy=RSI symbol=SBIN x=3 n=14 oversold_threshold=30
overbought_threshold=70 start_date="01/01/2023" end_date="01/01/2023"
```

Lab Report: Linear Regression for Stock Price Prediction

Introduction

This project applies a Linear Regression model to predict the closing price of a stock on day `t` and make trading decisions based on the prediction. The model uses historical stock data, considering various factors like previous closing prices, opening prices, volume-weighted average price (VWAP), and the number of trades.

Linear Regression Model

Equation

The Linear Regression equation used is:

$$Closet = \beta0 + \beta1Closet-1 + \beta2Opent-1 + \beta3VWAPt-1 + \beta4Lowt-1 + \beta5Hight-1 + \beta6(No\ of\ Trades)t-1 + \beta7Opent$$

Variables

- `Closet`: Closing price on day `t`.
- `β0` to `β7`: Coefficients determined by the regression model.
- `Opent`, `VWAPt`, `Lowt`, `Hight`, `(No of Trades)t`: Various stock metrics on day `t`.

Key Functions and Their Roles

1. `storeData`

- **Purpose**: Reads historical data from CSV files for training and testing.
- **Inputs**: File paths for training (`file_train_data`) and testing data (`file_test_data`).
- **Operation**: Parses CSV files and stores data into global vectors like `OPEN_TRAIN`, `HIGH_TRAIN`, `LOW_TRAIN`, `CLOSE_TRAIN`, `VWAP_TRAIN`, `NO_OF_TRADES_TRAIN`, and their corresponding `TEST` vectors.

2. `readData`

- **Purpose**: Prepares the data matrices (`x`, `y`, `x_test`, `y_test`) for regression analysis.
- **Operation**:
  - Fills `y` and `y_test` with closing prices from training and testing data.
  - Constructs `x` and `x_test` matrices with independent variables (like previous day's close, open, VWAP, low, high, number of trades, and current open price).
  - Utilizes data points from `DATE_TRAIN` and `DATE_TEST` to align data correctly.

3. `transpose`

- **Purpose**: Calculates the transpose of a given matrix.
- **Inputs**: Matrix to be transposed.

- **Returns**: Transposed matrix.

4. `determinant`

- **Purpose**: Computes the determinant of a matrix.
- **Inputs**: Matrix for which the determinant is to be calculated.
- **Returns**: Determinant value.

5. `inverseMatrix`

- **Purpose**: Calculates the inverse of a given matrix.
- **Inputs**: Matrix to be inverted.
- **Returns**: Inverse of the matrix.

6. `multiply`

- **Purpose**: Performs matrix multiplication.
- **Inputs**: Two matrices to be multiplied.
- **Returns**: Resultant matrix after multiplication.

7. `solveNormalEqns`

- **Purpose**: Solves the normal equations to compute the regression coefficients (Beta values).
- **Inputs**: Independent variables matrix `X`, dependent variable vector `y`, and a reference to the Beta vector `B`.
- **Operation**: Utilizes matrix operations (transpose, multiply, inverse) to solve for `B` using the formula `B = (X^T*X)^-1*X^T*Y`.

8. `Decide`

- **Purpose**: Predicts stock prices and decides on trading actions.
- **Inputs**: Beta coefficients `B`, percent difference `P`, maximum position `X`, and test data matrices.
- **Operation**:
  - Predicts prices using the regression equation.
  - Compares predicted prices with actual prices and decides whether to buy or sell based on the `P` threshold.
  - Manages the portfolio within the limits of `[-X, X]`.

9. `writeCSV`

- **Purpose**: Writes the trading decisions and cashflow to CSV files.
- **Operation**: Outputs the daily cashflow and order statistics into `daily_cashflow.csv` and `order_statistics.csv`, respectively.

Execution Flow

1. **Data Reading**: `storeData` reads the training and testing data.
2. **Data Preparation**: `readData` prepares the regression matrices.
3. **Model Training**: `solveNormalEqns` calculates the Beta coefficients.

4. **Prediction and Decision Making**: `Decide` uses the model to predict prices and make trading decisions.
5. **Output Generation**: `writeCSV` writes the results to CSV files.

Conclusion

This detailed breakdown of the implementation showcases a comprehensive approach to applying Linear Regression in stock price prediction. Each function plays a critical role, from data handling to model training and making informed trading decisions.

Note: This document elaborates on the specific functions and their roles in the Linear Regression strategy for stock price prediction as implemented in the provided C++ code.

Command for Execution

```
make strategy=LINEAR_REGRESSION symbol=SBIN x=3 p=2
train_start_date="01/01/2020" train_end_date="30/12/2022"
start_date="01/01/2023" end_date="01/01/2024"
```

Lab Report: Implementation of Mean-Reverting Pairs Trading Strategy

Overview

This project involves implementing a Mean-Reverting Pairs Trading Strategy using C++. The strategy is based on the price spread of a pair of correlated stocks, rather than on a single stock. We assume pre-tested correlation/cointegration between the stocks.

Key Functions and Their Roles

1. `StockData` Class

- **Purpose**: Represents the data for a pair of stocks, including their individual prices, calculated spread, rolling mean, rolling standard deviation, z-score, and trading actions.
- **Methods**:
  - `getPrice1`, `getPrice2`: Return individual stock prices.
  - `setspread`, `setRollingMean`, `setRollingStdDev`, `setz_score`: Calculate and set spread, rolling mean, rolling standard deviation, and z-score.
  - `buy`, `sell`: Set trading actions based on the strategy.

2. `readFromFile`

- **Purpose**: Reads stock price data from a file.
- **Input**: Filename of the CSV file containing stock data.
- **Operation**: Parses the CSV file to extract stock price data for each trading day.

3. `getLastNElements`

- **Purpose**: Retrieves the last `n` elements from a vector of `StockData`.
- **Input**: Vector of `StockData` and an integer `n` representing the number of elements.
- **Returns**: A vector containing the last `n` elements.

4. `main` Function

- **Initialization**: Parses command-line arguments for strategy parameters (`x`, `n`, `threshold`).
- **Data Preparation**: Reads data from files and calculates spreads for each pair of stock data.
- **Calculating Rolling Statistics**:
  - Computes the rolling mean and standard deviation of the spread over a given lookback period (`n` days).
  - Calculates the z-score for each day.
- **Trading Logic**:
  - Generates buy or sell signals based on the z-score crossing predefined thresholds.
  - Manages the portfolio within the limits of `[-x, +x]`.
- **Output Generation**:
  - Writes the trading actions to `order_statistics_1.csv` and `order_statistics_2.csv`.
  - Records the daily cash flow in `daily_cashflow.csv`.

Strategy Implementation

Spread Calculation

- **Spreadt = PS1,t - PS2,t**: The spread between the prices of the two stocks is calculated for each day.

Rolling Mean and Standard Deviation

- For each day `t`, the mean and standard deviation of the spread over the past `n` days are calculated, including the current day.

Z-Score Calculation

- **Z-Score**: `(Spread - Rolling Mean) / Rolling Std Dev`
- This score measures the deviation of the current spread from its historical average, normalized by the standard deviation.

Trading Decisions

- **Sell Signal**: If `z_score > threshold`, indicating the spread is significantly higher than its historical average.
- **Buy Signal**: If `z_score < -threshold`, suggesting the spread is much lower than usual.
- The strategy involves selling Stock 1 and buying Stock 2 for a sell signal, and vice versa for a buy signal.

Constraints and Assumptions

- The strategy allows short selling and maintains positions within [-x, +x] for each stock.
- The stocks are assumed to be correlated or cointegrated, as pre-tested.

Execution

The strategy is executed using the command:

```
make strategy=PAIRS symbol1=SBIN symbol2=ADANIENT x=5 n=20 threshold=2
start_date="01/01/2023" end_date="01/01/2024"
```

Lab Report: Implementation of Mean-Reverting Pairs Trading Strategy with Stop-Loss

Introduction

This project enhances the Mean-Reverting Pairs Trading Strategy by incorporating a Loss-based Stop-Loss mechanism. The strategy is applied to a pair of correlated stocks, focusing on the spread between their prices and utilizing a stop-loss threshold to manage risks.

Key Components of the Implementation

1. `StockData` Class

- **Attributes**: Stores stock data such as date, individual prices, spread, rolling mean, rolling standard deviation, z-score, action, quantity, and a change flag.
- **Methods**: Includes getters and setters for each attribute, and methods to set buy/sell actions.

2. `readFromFile` Function

- **Purpose**: Reads and parses stock data from a CSV file.
- **Operation**: Stores the extracted data (date, prices of two stocks) in a vector of `StockData` objects.

3. `getLastNElements` Function

- **Purpose**: Fetches the last `n` elements from a vector of `StockData`.
- **Inputs**: The `StockData` vector and the number of elements `n`.
- **Returns**: A vector containing the last `n` elements.

4. `main` Function

- **Initialization**: Parses command-line arguments to extract strategy parameters (`x`, `n`, `threshold`, `stop_loss_threshold`).
- **Data Preparation**: Reads and combines data from the main and extra data files.
- **Spread Calculation**: Calculates the price spread between the two stocks for each day.
- **Rolling Statistics Calculation**: For each day, computes the rolling mean and standard deviation of the spread over the past `n` days.
- **Z-Score Calculation**: Calculates the z-score based on the current spread, rolling mean, and standard deviation.
- **Trading Logic with Stop-Loss**:

    - Generates buy/sell signals based on z-score thresholds.
    - Implements stop-loss by closing positions when the z-score crosses the `stop_loss_threshold`.
    - Manages positions within `[-x, +x]` limits.
- **Output Generation**:
    - Records trading actions for each stock in `order_statistics_1.csv` and `order_statistics_2.csv`.
    - Writes daily cash flow to `daily_cashflow.csv`.

Detailed Strategy Logic

Spread Calculation

- **Spreadt = PS1,t - PS2,t**: Calculates the daily spread between the prices of the two stocks.

Rolling Mean and Standard Deviation

- Computes these values for the spread over the past `n` days, including the current day.

Z-Score and Stop-Loss Mechanism

- **Z-Score**: `(Spread - Rolling Mean) / Rolling Std Dev`.
- **Stop-Loss**: Closes positions when the z-score crosses the predefined `stop_loss_threshold`, indicating that the spread is moving unfavorably.

Trading Decisions

- **Sell Signal**: Triggered when `z_score > threshold`.
- **Buy Signal**: Triggered when `z_score < -threshold`.
- **Stop-Loss Action**: Executed based on the `stop_loss_threshold`.

Constraints and Assumptions

- Strategy assumes the possibility of short selling.
- Positions for each stock are maintained within the range of `[-x, +x]`.

Execution

The strategy is executed using the command:

```
make strategy=PAIRS_WITH_STOP_LOSS symbol1=SBIN symbol2=ADANIENT x=5 n=20
threshold=2 stop_loss_threshold=4 start_date="01/01/2023"
end_date="01/01/2024"
```