Coding Assignment

Objective

- Make a simulator program that calculates the history of holding position (i.e. the dollar amount held per stock) according to various portfolio trading algorithm codes provided by the user.
- The purpose of this assignment is not just to make a program that returns corre output, but we want to see how candidates design the structure of the program that it can handle various possible issues that can arise in practice, without rewriting entire program code. Also we want candidates to write documentatior their design and code for better communication with co-workers and users of tl program in the future.
- It is strongly recommended to use C++, but it is also possible to use other languages if there are no constraints on performance or memory usage due to t size of the input data or expansion of the search space.

Possible Program Flow

- Read config
- Read input data
- Iterate some calculation for generating position from start time point to end tim point
- Write output data

Requirements

- Input data
    - The simulator calculates the position using numeric 2D arrays of a predetermined shape as data.
    - This numeric 2D data is a time-series data for each stock given in the form csv file.
    - The shape of the array is T x N, where T is the number of time points for the entire simulation run and N is the total number of stocks available for use in portfolio.
    - Example of data
        - The closing price data of the ii-th stock between the ti-th and (ti+1)-th t points: Close[ti][ii]
        - The change in price of the ii-th stock between the ti-th and (ti+1)-th tim points: Returns[ti][ii]
        - Sample input data

```
Time,AAPL,GOOGL ... 2022-01-01,0.01,-0.005,... 2022-01-
02,0.0011,-0.015,...
```

- Output data
    - The output data is in csv format and represents the amount of stock held fo each stock at each time point.
    - The shape is the same as the input data, T x N.
    - The time point and stock order are matched to the input data.
    - Sample output data

```
Time,AAPL,GOOGL ... 2022-01-01,1234,512... 2022-01-
02,-5441,144...
```

- Trading Algo
    - The trading algo refers to an algorithm that calculates the amount of stock l for each stock at a specific point in time from input time-series data.

- User's various trading ideas are implemented as an algorithm, and each trac algo has a unique name. Trading algo is represented in the program as code (functions or classes, etc.).
- The same algo can behave differently depending on the input parameters. Parameter values can be specified in the config file.
- Component of trading algo
    - input data
    - output
    - parameter : algo config of current simulation
    - state variable : values generated during calculation, which can be passe next iteration, or it can be ignored
    - calculation logic : code to process input data and generate output
- The trading algo code includes the following:
    - Names of the data to be read from the file and used in the algorithm
    - Variables that hold the 2D arrays of data read
    - Specific variables used in the algorithm and their initial values (i.e. parameters)
    - void run(int ti) function
        - Contains an algorithm that calculates the position of each stock, position[ii], that you want to hold between the ti-th and (ti+1)-th tim points for all ii-th stocks.
        - Must be able to pass the variable values calculated during the ti-th iteration so that they can be used in the subsequent (ti+1)-th iteratio
    - Example of algo code in pseudo-code format. It is not necessary to follo this format, this is just a reference to give you the idea.

```
procedure algo1( time_index : current time index input_dat
: (T x N) matrix of close prices, T=number of time points,
N=number of instruments state_variable : variable for
storing intermediate caculation results. output : 1D array
of current dollar positions ) for ii = 1 to
number_of_instruments do: output[ii] = state_variable[ii]
input_data[time_index][ii] end procedure return output
```

- Simulation
  - When the user executes the simulation, they specify the contents of the simulation in the config file. The configuration file contains the following information:
    - StartTime and EndTime of the simulation
    - Algorithm config
      - trading algo name to be used
      - Initial values for specific variables to be used in the trading algorithr code (i.e. parameter values)
        - For example, suppose an algorithm calculates the point at which two moving averages of prices cross, and uses the lengths of th first and second moving averages as variables named w1 and w this case, you can receive the values of w1 and w2 from the con file and initialize them.
      - The config file may contain multiple algorithm configs.
    - Example config files.
      - There are three algorithm configs SimSample1, SimSample2, SimSample3, respectively.
      - Below examples are written in yaml format and csv format, but it is r necessary to follow, and you can make your format freely for your convenience.
        - Yaml Format

```
Simulation: Info: StartTime: 2020-01-01 EndTime:
2022-12-31 Algo: SimSample1: Algoname: Algo1 param1
1.0 param2: 24 SimSample2: Algoname: Algo1 param1:
1.5 param2: 24 SimSample3: Algoname: Algo2 var1: 36
var2: 20 var3: 2.0
```

        - csv format

```
2020-01-01,2022-12-31
```

```
SimSample1,Algo1,param1,1.0,param2,24
SimSample2,Algo1,param1,1.5,param2,24
SimSample3,Algo2,var1,365,var2,20,var3,2.0
[simulation name],[algo name],[param1_name],
[param1_value],[param2_name],[param2_value]...
```

- Simulator should calculate position vectors by iterating run(ti) from ti_start ti_end, where ti_start and ti_end are the time indices corresponding to StartTime and EndTime read from the config file.
  - The calculated positions are stacked in the form of a 2D array, a historic positions.
  - The results is saved in {simulation_name}.csv.
- The config file may contain multiple algorithm configs. In this case, each trading algo is simulated using the given algorithm config, and each result i saved separately.
  - Since the same input data can be used in multiple trading algo codes, a efficient simulator structure should be designed.
  - (Extra credit) Allow multiple simulations to be performed simultaneousl using multi-threading.

Guideline

- C++ is recommended, but using other languages is also possible if there are no constraints on speed and memory usage.
- In the case of C++, the use of modern C++ is recommended, and the use of libraries is free.
- Be mindful of the speed and memory usage of the program.
- It is not necessary to strictly follow the format of the codes mentioned above, a is allowed to implement them freely.
- If there are parts of the instructions that are not specific, assumptions can be added and implemented accordingly.

- Design the program so that the user can add new logic frequently without caus frequent changes to the entire program.
    - Ultimately, the logic developed by the user is compiled into a shared library dynamically loaded according to the simulation settings, but in this assignm it is assumed that it is written in advance and compiled with the simulator.
- Write a program design document. Describe the structure and precautions of th program in detail to make communication with other collaborators or users easi
- When trying to create a simulator with minute or sub-minute resolution, list possible issues that can arise using the above simulator structure, and describe how they can be resolved.
    1. Issues regarding memory usage, simulation speed, and others that should b considered for large-scale simulations
    2. Issues with capability in implementing various strategies.

Sample Data, Algo Code (pseudo-code), and Position

- Sample Data

    Close.csv 17920.1KB

- Sample Algo pseudo-code

```
procedure algo1( time_index : ti close_price : (T x N) matrix of
close prices, T=number of time points, N=number of instruments
position : 1D array of current dollar positions, size=number of
instruments ) for ii = 1 to number_of_instruments do: position[i
= -(close_price[ti-1][ii] - close_price[ti-2][ii])/close_price[t
2][ii] * 10000 end procedure return position
```

- Output Position of the above Sample Algo

    Position.csv 47439.8KB