ISyE 6767 Sys-Computation Finance
S. Deng

Numpy Exercise
Oct 22, 2025

1 Task 1: Trading Strategy

You are a Quantitative Financial Analyst looking for a new trading strategy for an investment firm. You decide to use Python's Numpy library to analyze historical stock price data to discover potential trading signals. Your task is to implement a function that takes in historical price data for a stock and returns a decision array for the trading strategy.

1.1 Input

A csv input file that contains the closing prices of the stock for n days (n consecutive trading days), where $0 \leq n \leq 10^5$. The number of trading days: n. Initial account value w (in day 0): \$10000. Number of shares to buy each time: sh = 10.

1.2 Output

A Numpy array `signals` of length n, where each element indicates:

- 1: Represents buy to open a position
- 0: Represents holding/no action
- -1: Represents sell to close open-position

An array containing the positions in the stock over the n days. An array containing the total account value (namely, stock value plus the remaining cash value) in each of the n periods. Screen output the cumulative trading profit-and-loss by the end of n days. Save the above 3 arrays in a csv file with number "trading_results.csv", with the first column being the dates given in the input csv file.

1.3 Strategy description

1. If the stock's price rises for the past 3 consecutive days, one decides to: buy sh shares if the current shares of stock is less than 2*sh, otherwise do nothing.
2. If the stock's price falls for the past 2 consecutive days, one decides to sell everything if current number of shares > 0, otherwise do nothing.
3. If the current period is the last day of the period, sell everything if there are positive shares, otherwise do nothing.
4. In any other case, one decides not to take any action.

1.4 Example and Explanation

Example:

$$\begin{aligned} \texttt{prices} &= [98, 100, 102, 104, 103, 101, 99, 100, 102, 104, 106, 107, 105] \\ \texttt{signals} &= [0, 0, 0, 1, 0, -1, 0, 0, 0, 1, 1, 0, -1] \\ \texttt{positions} &= [0, 0, 0, 10, 10, 0, 0, 0, 0, 10, 20, 20, 0] \end{aligned}$$

Explanation: For the given `prices`, we observe the following:

- On days 1-4, the prices are $98, 100, 102, 104$, which is an increasing sequence, hence the signal on day 4 is 1 (buy).
- On days 5-6, the two prices are $103, 101$, which is a decreasing sequence comparing the price of 104 on day 4. As we have 10 shares of stock to sell, the signal on day 6 is -1 (sell).
- On days 7-10, the prices are $99, 100, 102, 104$, another increasing sequence, hence the signal on day 10 is 1 (buy).
- On days 11, the past 3 prices and the current price form another increasing sequence, and the current position is $10 < 20$, hence the signal on day 11 is 1 (buy).
- On days 12, the current price along with previous 3 prices form yet another increasing sequence, but the current position is 20 which is no longer < 20, hence the signal on day 12 is 0 (do nothing).
- On day 13, this is the last day and the position is $20 > 0$, thus the signal is to sell everything: -1 (note the shares to sell is not 10 but all shares in the account, which is 20).
- All other days have a signal of 0 (hold/no action).

1.5 Requirements

Implement a python function with name **trading_strategy** which takes the **csv file name** as input variable. The function outputs are described above. You must utilize Numpy array to solve this problem.

2 Task 2: Energy Maze

Imagine entering a maze where each cell has an energy value associated with it. These energy values can either be positive (indicating you gain that much energy) or negative (indicating you lose energy). Your task is to traverse the maze from the top-left corner to the bottom-right corner with the highest net energy possible. However, there's a catch: at no point during the traversal can your energy go below 1.

2.1 Input

A Numpy array `maze` representing the maze where each cell contains an energy value. The dimensions of the maze are $m \times n$ where $1 \leq m, n \leq 1000$ and the energy value at each cell $-100 \leq$ `maze[i][j]` ≤ 100.

2.2 Output

An integer `initial_energy`, which represents the minimum initial energy required to traverse the maze without the total energy dropping below 1 at any point.

2.3 Constraints

1. You can only move either to the right or downwards in the maze.
2. The traversal starts at the top-left corner and ends at the bottom-right corner of the maze.
3. The initial energy should be such that at no point during the traversal does the energy go below 1.

2.4 Example and Explanation

Example:

$$\texttt{maze} = \begin{bmatrix} -2 & -3 & 3 \\ -5 & -10 & 1 \\ 10 & 30 & -5 \end{bmatrix}$$

$$\texttt{initial\_energy} = 7$$

Explanation: For the given `maze`, starting with an initial energy of 7 is the optimal choice. This is because, by the time we reach the bottom-right corner of the maze, our energy is 1. At no point during the traversal does our energy fall below this value.

2.5 Requirements

Implement a python function named **min_initial_energy** which takes the **maze** as an input variable. The function should output the minimum initial energy required to traverse the maze as described above. You must utilize Numpy array to solve this problem.