Comity Coding Challenge: Virtual Bid Portfolio Optimizer

Background

Wholesale electricity markets in the US operate with a two settlement mechanism that consists of a real-time (RT) and day-ahead (DA) market. In both markets, participants make supply offers and demand bids to deliver or consume power at a particular location (i.e. node on the electrical grid) and time interval (e.g. hour of day). The RT market accepts bids on a rolling basis for immediate delivery, while the DA market accepts bids during a fixed time window every day for each hour of the following day (the "operating day"). The markets also allow for purely financial participation via virtual bidding, in which participants arbitrage price differences across the RT and DA markets at a particular node. These bids are either virtual supply offers (INCs), with which power is sold in the DA market and purchased in the RT market, or virtual demand bids (DECs), with which power is purchased in the DA market and sold in the RT market.

Goals

The goal of this challenge is to develop a simple virtual bidding strategy based on Harry Markowitz's Modern Portfolio Theory, and to backtest the strategy on some provided historical data. You will be provided with skeleton Python code for your solution, with stubs for all of the functions that must be implemented to complete a working solution, as well as the historical price data you should use for your backtest. Your task is to write a program that will execute the strategy and backtest it over the last year of data, outputting the bids produced by the strategy during the backtest period. Although the skeleton code is written in Python, you are welcome to use any language of your choice provided you can find a package with an optimization capability similar to the one from SciPy used in the skeleton.

Mean-Variance Optimization for Virtual Bidding

Modern Portfolio Theory (MPT) is a classic mathematical framework from the field of quantitative finance for constructing portfolios of assets that maximize expected returns for a given level of risk. In MPT, risk is measured using the variance of the portfolio return. The "efficient" portfolio (i.e. the portfolio with the greatest return) for a given risk level is found by minimizing the following expression:

$$w^T \Sigma w \; - \; q \times R^T w$$

where

- w is the vector of portfolio weights, which can be positive or negative, and $\sum_i w_i = 1$.
- Σ is the covariance matrix for the returns of the assets in the portfolio.

- $q \geq 0$ is the "risk tolerance" factor, where 0 gives a portfolio with minimal risk and larger values give riskier portfolios.
- R is the vector of expected returns.
- $w^T \Sigma w$ is the variance of the portfolio return.
- $R^T w$ is the expected return of the portfolio.

In the typical setting, the assets in the portfolio would be financial assets such as stocks or bonds, but we can apply the same framework to virtual bidding in electricity markets by taking power delivered at a certain node and hour of day to be the "asset" and using the difference (or spread) between the DA and RT prices as the "return". The "weights" then become the quantities of our bids (in MW) with the sign determining the direction (buy/sell or DEC/INC). Rather than summing to 1, the weights instead sum to a maximum volume (in MW) that we wish to trade on any day.

Concretely, in the virtual bidding setting:

- w is the vector of bid quantities, with one entry per node and hour combination, which can be positive or negative, and $\sum_i |w_i| = V$, where V is the daily volume limit.
- R is the vector of expected "DART spreads" or differences between DA and RT prices at each location and hour.
- Σ is the covariance matrix for the DART spreads.

Note that for simplicity we'll assume that our strategy is "self-scheduling": it will always take whatever price the market offers for a bid, and the entire quantity of the bid will clear. This is analogous to a market order in other financial markets.

Your solution should use fixed values of 500 for the risk tolerance factor, q, and 100 for the daily volume limit, V. These are defined as constants for you in the skeleton code.

Simplifying the Covariance Matrix

One of the biggest challenges with mean-variance portfolio optimization is coming up with good estimates for the covariance matrix, Σ. To simplify this step, for our strategy we'll estimate only the diagonal entries of the covariance matrix, which represent the variance of the DART spread at a single node and hour combination. We'll use 0 for all other entries, which is equivalent to the assumption that DART spreads are not correlated across nodes and hours. In general this is a poor assumption but it works reasonably well in practice and vastly simplifies the problem.

Constrained Optimization with SciPy

To perform the minimization of the above mathematical expression, the skeleton solution uses the optimization package included with SciPy, specifically its minimization routine of an arbitrary objective function with an inequality constraint. Part of your task is to define the objective

function corresponding to this expression as well as a function describing the constraint over the bid quantities, which must take a non-negative value if and only if the constraint is satisfied. The objective function should accept as arguments the bid quantities (w), expected spreads (R), and spread variance estimates (diagonal entries of Σ), while the constraint function should accept the bid quantities (w) and maximum daily volume limit (V).

Point-In-Time Mean and Variance Estimates

To estimate the expected DART spreads and their variances, we'll take the sample mean and variance observed over all days in our historical data. To produce an accurate backtest, however, we need to censor any data that would not have been observable at the time we would have placed our bids. Since virtual bids are placed in the day-ahead market, we must exclude any observations later than one day prior to the operating day for which we are bidding. Concretely, when calculating the expected spreads and spread variances for bids for operating day d, exclude all observations after day $d - 2$ (although technically some observations for day $d - 1$ are available in time for trading, it's simpler to use the same cutoff for all hours).

Data

The input data consists of historical price data across 8 nodes from 2021 to 2022. It is provided in a CSV file called "prices.csv" with the following schema:

Column	Type	Description
`operating_day`	date	The operating day (i.e. day of power delivery) in YYYY-MM-DD format
`hour_beginning`	int	The hour of day that marks the beginning of the delivery hour
`node`	string	A string identifier of the node where power is delivered
`da_price`	float	The clearing price (per MW) in the day-ahead market
`rt_price`	float	The clearing price (per MW) in the real-time market

Deliverables

Your program should produce two outputs: a CSV file with the bids produced by the strategy for the last year of historical data (2022) and the total profit/loss (PnL) of the strategy over that year. The bid file should be called "bids.csv" and have the following schema:

Column	Type	Description
`operating_day`	date	The operating day (i.e. day of power delivery) in YYYY-MM-DD format

`hour_beginning`	int	The hour of day that marks the beginning of the delivery hour
`node`	string	A string identifier of the node where power is delivered
`bid_mw`	float	The bid quantity (in MW). This must be greater than or equal to 0.1
`bid_type`	string	The type of bid (“INC” or “DEC”)

Note that there is a **minimum bid size of 0.1 MW,** so please filter your output to include only bids that reach the minimum size.

Recall that we assume our strategy is self-scheduling, so to compute the PnL we simply multiply the bid quantity by the DART spread for the corresponding node, hour, and operating day. The direction of the spread will depend on the bid type. Your solution should print the total PnL number to the console (as the skeleton code does).

Furthermore, if you make any changes to the project setup, please update the provided README file with instructions for how to build (if necessary) and run your solution.

Finally, please include a write-up answering the following questions:

1. How long did you spend working on the problem? What difficulties, if any, did you run into along the way?
2. The slowest step in your program is most likely the loop where we run the portfolio optimization for each operating day in the backtest period. How could we speed this up to accelerate research and development velocity?
3. For this challenge we provided you with a fixed value of the risk tolerance factor. How would you go about setting the risk tolerance value if this was not provided to you?
4. What happens to the PnL when you increase the maximum daily volume limit while keeping the risk tolerance factor fixed? What does this imply about how you would choose the risk tolerance parameter?
5. How would you productionize your code so that the strategy could run on a daily basis and submit portfolios to the market? How can you structure the code and what other precautions can you take to ensure that the core strategy code which generates your backtest results is the same as the code that runs on production?

Bonus Features

If you feel so inclined, here are some bonus features that you can add to your solution after completing the core requirements. We highly recommend finishing all other requirements, including the writeup, before working on these.

- Output more metrics quantifying the strategy backtest performance
- Output charts of the strategy backtest performance

- Add functionality to tune the risk tolerance factor
- Computational performance improvements/speedups for the backtest

Evaluation

Your solution will be evaluated based on the following criteria:

- Correctness - Is the strategy implemented as defined above? Are the spread statistics point-in-time accurate?
- Maintainability - Is the code well-organized, readable, and easy to modify?
- Communication - Do the answers in your writeup clearly communicate your ideas about the problem and solution?
- Bonus features - Which bonus features, if any, have you implemented?

Notes and Suggestions

- As a reminder, although the provided skeleton code is written in Python, you don't have to use Python for your solution. You'll have to port the skeleton code and find a comparable optimization package, but you're welcome to use any language you feel comfortable with.
- Running the backtest over all of 2022 should take ~5 minutes. While you're still iterating on your solution, we recommend reducing the date range so that you don't spend as much time waiting for the program to run.
- The program structure outlined by the skeleton code is only a suggestion. Feel free to refactor your solution as much as you like, as long as the same outputs are produced.
- To help with readability, type hints are included in the skeleton code. We use them in our codebase and do find them helpful, but given how spotty the coverage is for many of Python's scientific computing libraries (including SciPy), your mileage may vary. You are not required to include type hints in your solution, but if you choose to do so, mypy is included in the project requirements for your convenience.