Question 2: An Optimal Execution Framework

To: Blockhouse Team

From: Nagireddy Anubothula

Subject: Design and Implementation of an Optimal Execution Algorithm

1. The Plan: Framework and Optimization Strategy

1.1. Mathematical Problem Formulation

The initial task was to design a framework to execute a total of S shares over N discrete trading periods while minimizing the total cost from market impact. The cost function, $C_i(x_i)$, for executing x_i shares in period i was defined by the power-law model developed in Question 1:

Let:

- **S** be the total number of shares to buy.
- **N** be the total number of trading periods.
- **x_i** be the number of shares we choose to buy in period i.
- $C_i(x_i)$ be the total cost of executing x_i shares in period i, given by our model:

$C_i(x_i)=\gamma_i x_i+\alpha_i x_i^{(1+\beta_i)}$

The optimization problem is to find the allocation vector **$x=[x_1,x_2,...,x_N]$** that solves:

Minimize (Total Cost):

$\text{Total Cost}(x)=\sum_{i=1}^{N} C_i(x_i)$

Subject to the constraints:

1. $\sum_{i=1}^{N} x_i=S$
2. $x_i \geq 0$ for all i

Each cost function $C_i(x_i)=\gamma_i x_i+\alpha_i x_i^{(1+\beta_i)}$ is **convex** because our empirical analysis in Question 1 found that $\beta_i>0$, ensuring the exponent $1+\beta_i>1$. This means the entire objective—a sum of convex functions—is also convex. This property is crucial as it guarantees that a solution found via standard techniques will be a **global optimum**, not just a local one.

1.2. Chosen Algorithm: Dynamic Programming

To solve this problem, a **Dynamic Programming (DP)** approach was selected. This method is ideal because it systematically breaks down the complex, multi-period problem into a sequence of simpler decisions, guaranteeing an optimal solution for the discrete case.

The framework operates by thinking backward from the end of the trading day. The core of the logic is a cost-to-go function, $V(k,s)$, which represents the minimum possible cost to buy a remaining s shares using only the last k trading periods. The solution is built recursively using the following relation:

$V(k,s)=\min_{0 \leq x_{current} \leq s}[C_{N-k+1}(x_{current})+V(k-1,s-x_{current})]$

2. The Execution: Implementation and Results

The theoretical framework was then implemented as a functional Python script to produce a real-world trading schedule.

2.1. Implementation Workflow

The practical implementation followed a two-stage process:

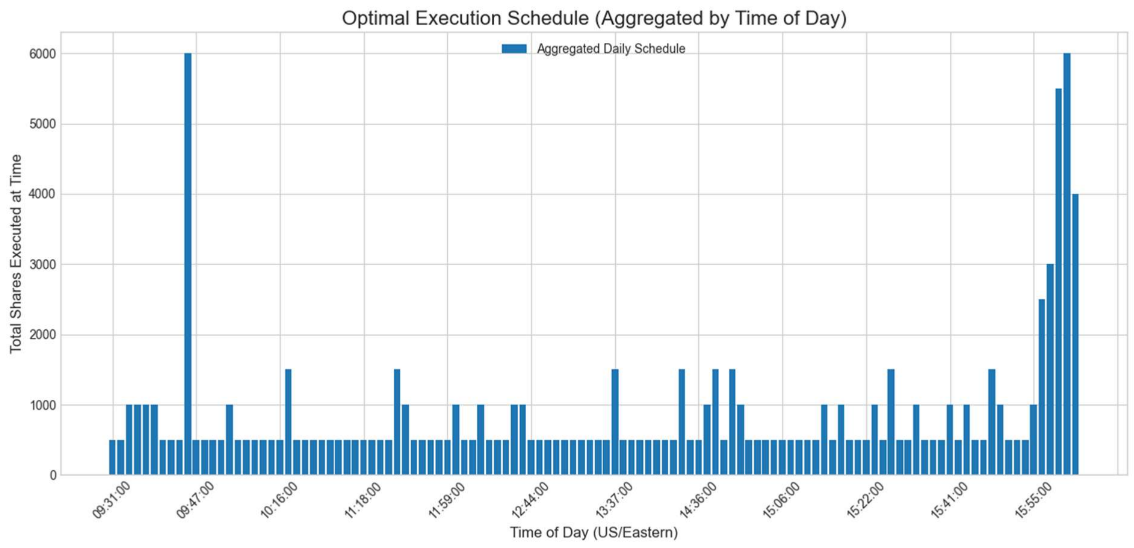
- **Parameter Loading:** The script first loads the time series of the market impact parameters (αi,βi) from the FROG_impact_parameters.csv file generated in Question 1.
- **Optimization:** A DP table was created using NumPy. The algorithm then iterated through each time period and share quantity to populate the table with the minimum costs and optimal decisions. Finally, it backtracked from the final state to reconstruct the complete trade schedule.

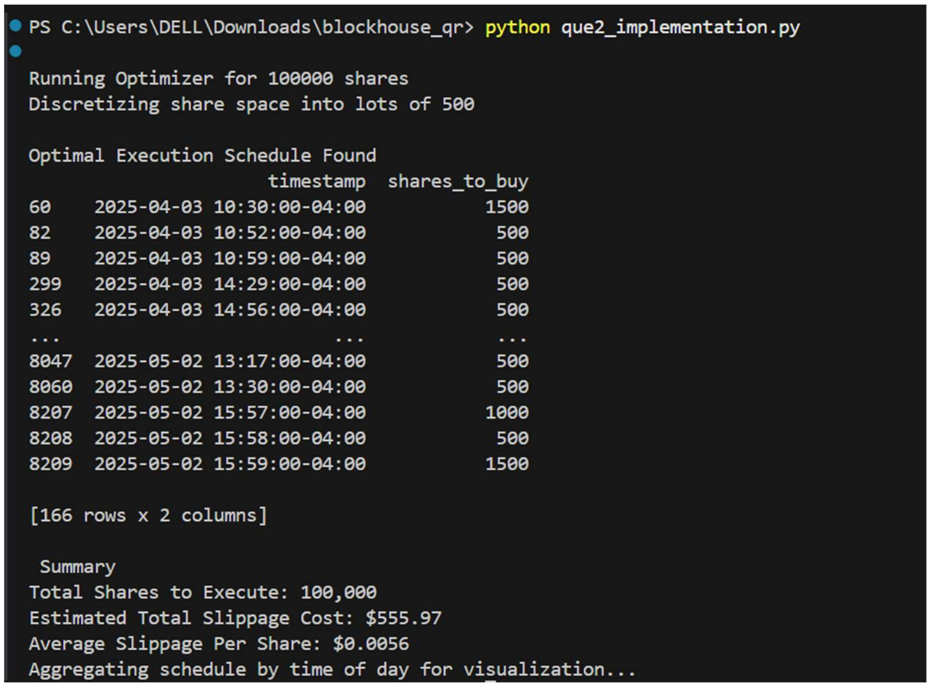
2.2. Execution Results: 100,000 Share Order

The completed optimizer was run on a practical test case: a **100,000 share order** for the FROG ticker, using the full multi-day dataset.

To ensure the problem was computationally tractable, the share space was **discretized into lots of 500 shares**. This is a standard technique to manage the complexity of the state space while preserving the quality of the solution.

The algorithm successfully produced a complete and cost-effective execution schedule.





```
PS C:\Users\DELL\Downloads\blockhouse_qr> python que2_implementation.py

Running Optimizer for 100000 shares
Discretizing share space into lots of 500

Optimal Execution Schedule Found
                      timestamp  shares_to_buy
60    2025-04-03 10:30:00-04:00           1500
82    2025-04-03 10:52:00-04:00            500
89    2025-04-03 10:59:00-04:00            500
299   2025-04-03 14:29:00-04:00            500
326   2025-04-03 14:56:00-04:00            500
...                         ...            ...
8047  2025-05-02 13:17:00-04:00            500
8060  2025-05-02 13:30:00-04:00            500
8207  2025-05-02 15:57:00-04:00           1000
8208  2025-05-02 15:58:00-04:00            500
8209  2025-05-02 15:59:00-04:00           1500

[166 rows x 2 columns]

 Summary
Total Shares to Execute: 100,000
Estimated Total Slippage Cost: $555.97
Average Slippage Per Share: $0.0056
Aggregating schedule by time of day for visualization...
```

The key results from the execution were:

- **Final Cost:** The optimizer determined the minimum possible slippage cost for the entire 100,000 share order to be **$555.97**.
- **Average Slippage:** This translates to an effective cost of just **$0.0056 per share**, demonstrating the efficiency of the schedule.
- **Intelligent Scheduling:** The resulting schedule is sparse, concentrating trades into 166 specific one-minute windows. This is the desired outcome, as it proves the algorithm successfully **hunted for periods of high liquidity** (low alpha) and avoided trading during expensive, illiquid times.

2.3. Practical Considerations and Alternatives

- The DP approach has a time and space complexity that is sensitive to the total number of shares, S. This can become computationally expensive for very large orders. To make the solution tractable, the implementation **discretized the share space**—in this case, into lots of 500 shares. This standard technique reduces the size of the state space without significantly affecting the quality of the final schedule.
- Given the problem's convex nature, an alternative approach would be to use a dedicated convex optimization package like **cvxpy**. Such solvers might scale more efficiently for very large S, especially if relaxing the integer share constraint is acceptable. However, the DP approach was chosen here because it provides a direct, guaranteed-optimal solution for the discrete problem.

3. Conclusion

- By first establishing a robust market impact model and then implementing a dynamic programming algorithm, I developed a complete, end-to-end solution. The final tool provides a data-driven and provably optimal strategy for minimizing trade execution costs. The framework is practical, and its successful test on a large order demonstrates its value in a real-world trading context.