Modeling Temporary Impact Functions from Limit Order Book Data

Devansh Srivastava

July 31, 2025

Introduction

This report addresses the modeling of the temporary market impact function $g_t(x)$, defined as the slippage (relative to the prevailing mid-price) incurred by a market order of size x at time t. Using depth-10 limit order book data for the three stocks SOUN, FROG, and CRWV, we empirically estimate the temporary impact function and provide a mathematical framework for optimally allocating trades over time to minimize total execution cost.

1 Empirical Modeling of Temporary Impact (Problem 1)

The temporary impact function $g_t(x)$ describes the cost penalty for executing x shares at time t, relative to the prevailing mid price. Our analysis involved:

1. **Order Book Reconstruction:** At one-minute intervals, a full snapshot (10 levels per side) of the LOB was constructed for each ticker.
2. **Market Order Simulation:** For a grid of order sizes x, simulated aggressive market orders consume liquidity, tracking fill prices.
3. **Slippage Calculation:** For each simulated trade, slippage is calculated as

$$\text{Slippage} = \text{VWAP}^{1} - \text{Pre-trade Mid Price}$$

4. **Curve Fitting:** To the average slippage versus x, we fit both linear models ($g(x) = \beta x$) and power-law models ($g(x) = kx^{\alpha}$).
5. **Model Evaluation:** R^2 scores are computed for fit quality.

Empirical Results

Interpretation and Model Choice

- **Power-law superiority:** For all tickers, the power-law model $g(x) = kx^{\alpha}$ fits better than linear (all R^2_{lin} are negative: linear is worse than fitting a constant mean).

[1] VWAP (Volume Weighted Average Price): $\text{VWAP} = \frac{\sum(\text{price}\times\text{size})}{\sum \text{size}}$

Ticker	k (Power Law)	α (Power Law)	R^2_{pl}	β (Linear)	R^2_{lin}
SOUN	0.0236	0.07	0.047	0.00001	-13.6
FROG	0.0165	0.35	0.143	0.00007	-0.68
CRWV	0.0757	0.14	0.046	0.00002	-1.56

Table 1: Empirical model fits: mean slippage power-law and linear parameters for three tickers. R^2_{pl}: power law fit, R^2_{lin}: linear fit quality.

- **Sublinear impact:** The exponents α are all in $(0, 1)$, indicating that the impact function is concave—i.e., average slippage rises with order size, but at a decreasing rate.
- **Implication:** Real LOBs for these names and this day are sufficiently deep and liquid; for the tested range, marginal cost per share decreases as size rises. This is typical for high-liquidity environments or when most simulated order sizes are much less than total displayed liquidity.
- **Model recommendation:** For further modeling and optimization, use $g_t(x) = k_t x^{\alpha_t}$, with k_t, α_t locally estimated from LOB data in each time period as needed.

2 Mathematical Framework for Optimal Scheduling (Problem 2)

Suppose you must buy exactly S shares over a trading day split into N periods, purchasing x_i shares in period i ($i = 1, \ldots, N$). The goal is to minimize total expected slippage using the fitted cost functions.

Setup

$$\begin{aligned} &\text{Variables:} && \mathbf{x} = (x_1, x_2, \ldots, x_N) \\ &\text{Constraints:} && \sum_{i=1}^{N} x_i = S, \quad x_i \geq 0 \end{aligned}$$

For each trading period t_i, you model the cost of executing x_i shares as:

$$g_{t_i}(x_i) = k_{t_i} x_i^{\alpha_{t_i}} \qquad (0 < \alpha_{t_i} < 1,\ k_{t_i} > 0)$$

Total cost to minimize:

$$\text{Total Cost} = \sum_{i=1}^{N} g_{t_i}(x_i) = \sum_{i=1}^{N} k_{t_i} x_i^{\alpha_{t_i}}$$

Optimization Problem

$$\boxed{\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{N} k_{t_i} x_i^{\alpha_{t_i}} \\ \text{s.t.} \quad & \sum_{i=1}^{N} x_i = S, \quad x_i \geq 0 \; \forall i \end{aligned}}$$

Solution Approach and Interpretation

1. Concave Costs $(0 < \alpha < 1)$**:**

- Here, $g_{t_i}(x_i)$ is sublinear (*diminishing marginal cost*), so the cost-minimizing solution for a purely concave objective *without further constraints* is to execute the entire order in the period where k_{t_i} is lowest.
- This "bang-bang" solution is not realistic in practice due to liquidity, market risk, and execution constraints.

2. Lagrangian and First-Order Conditions: Define the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \lambda) = \sum_{i=1}^{N} k_{t_i} x_i^{\alpha_{t_i}} + \lambda \left(\sum_{i=1}^{N} x_i - S \right)$$

First-order condition for x_j:

$$k_{t_j} \alpha_{t_j} x_j^{\alpha_{t_j} - 1} + \lambda = 0 \implies x_j^{\alpha_{t_j} - 1} = -\frac{\lambda}{k_{t_j} \alpha_{t_j}}$$

With $0 < \alpha_{t_j} < 1$, $x_j^{\alpha_{t_j} - 1}$ is decreasing in x_j, so the solution concentrates allocation in the "cheapest" periods.

3. Realistic Execution: To generate robust and practical schedules, augment the base cost minimization with additional constraints:

- **Liquidity caps:** $0 \leq x_i \leq L_{t_i}$
- **Risk penalties:** Penalize variance of execution price or non-uniform schedules, e.g. as in Almgren-Chriss:

$$\min_{\mathbf{x}} \sum_{i=1}^{N} k_{t_i} x_i^{\alpha_{t_i}} + \gamma \text{Var}(\text{Final Price})$$

  where $\gamma > 0$ is risk aversion.

Numerical Solution

Because (i) α_{t_i}, k_{t_i} may vary, (ii) additional constraints matter, and (iii) objective is not (jointly) convex, you must solve numerically. Use packages like `scipy.optimize.minimize` or convex/concave solvers with your empirically fit $g_{t_i}(x_i)$. A typical workflow:

1. Fit k_{t_i}, α_{t_i} from recent LOB at each t_i.

2. Set L_{t_i} by observed max depth/liquidity.
3. Solve minimization for $\mathbf{x}$ under constraints.
4. Recalibrate schedule as new information or fills occur.

3 Summary

- For your data, empirical cost functions are *concave* ($0 < \alpha < 1$): slippage increases with order size, but sublinearly.
- The base optimal scheduling, in this regime, would push most (or all) volume into the periods with the smallest k_{t_i}, but real-world execution adds risk, liquidity, and operational constraints.
- Correct practice is to continually update $g_{t_i}(x)$, encode all practical execution limitations, and solve numerically for the allocation vector $(x_1, ..., x_N)$ at each step.

Note: All analysis, simulation code, and fit results were computed as shown in the attached notebook. Refer to the github repo at link for all code and analysis.