Market Temporary Impact Modeling & Execution Strategy

Part One: $g_t(X)$ Modeling

Overview

This project explores the modeling of market impact through the slippage function $g_t(X)$, which quantifies the execution cost of a market order of size S at time t. We analyze order book data from three ticker symbols, FROG, CRWV, and SOUN, and evaluate the efficacy of linear, polynomial, and nonlinear models in predicting slippage. A key objective is to design a data-driven allocation strategy that dynamically distributes large parent orders throughout the trading day in a way that minimizes cumulative slippage.

$g_t(X)$ and the Limitations of Linear Approximations

The temporary market impact function $g_t(X)$ is defined as:

$g_t(X) = [(\text{Total Execution Cost}_t(X))/X] - \text{Mid Price}_t$

This represents the per-share price deviation from the mid-price when executing a market order of size S at time t. A common simplification is the **linear approximation**:

$g_t(X) \approx \beta_t \cdot X$

where β_t is often estimated as the spread over the aggregated top-of-book liquidity. While computationally efficient and simple, this linearization fails in several ways:

1. **Order Size Nonlinearity**: Slippage does not scale linearly with size. Larger orders walk deeper into the book, triggering disproportionate cost.

2. **Static Spread Metrics**: β_t based on L1 data fails to adapt to liquidity shifts or intraday dynamics.

3. **Lack of Depth Structure**: Linear estimates ignore multi-level quote depth, leading to poor estimates especially during low-liquidity periods.

Simulation-Based Slippage Estimation

To overcome these limitations, we simulate the execution of market orders using LOB snapshots from the dataset CSV files. For each 1-minute snapshot:

- We simulate market buys of varying sizes (e.g., 10, 25, 50 shares).

- Execution price is computed by consuming available ask levels until the order is filled.

- Slippage is calculated as the difference between the volume-weighted execution price and the mid-price.

This simulation provides ground-truth values of $g_t(X)$ and allows for empirical modeling of its behavior throughout the day.

Modeling Approach for β_t

To explore whether real-time LOB features can predict market impact, we model the slippage slope β_t as a function of spread, imbalance, and mid-price.

Model Performance Summary

FROG

Model	MSE	R^2 Score
Linear Regression	9.40e-08	0.021
Polynomial	9.27e-08	0.035
Random Forest	6.74e-08	**0.298**

Note: The Random Forest captures nonlinear dependencies well in this high-liquidity name. Polynomial features provide marginal improvement over linear ones.

CRWV

Model	MSE	R^2 Score
Linear Regression	3.59e-07	0.029
Polynomial	3.57e-07	0.036
Random Forest	3.58e-07	0.032

Note: The model explains limited variance due to sparse liquidity and volatile spreads. All methods perform similarly.

SOUN

Model	MSE	R^2 Score
Linear Regression	1.85e-09	-0.003
Polynomial	1.86e-09	-0.006
Random Forest	1.92e-09	-0.042

Note: None of the tested models generalize well on SOUN due to its erratic, low-liquidity nature.

All Datasets Combined

Model	MSE	R^2 Score
Linear Regression	8.23e-10	0.019
Polynomial	8.10e-10	**0.034**
Random Forest	8.17e-10	0.026

Note: The polynomial model marginally outperforms others, though all exhibit modest predictive power. This suggests weak but consistent relationships between LOB features and execution cost across symbols.

Cross-Ticker Analysis

This modeling pipeline was applied across all three tickers and revealed several patterns:

1. **Nonlinearity in $g_t(X)$**: Slippage increases superlinearly with order size in all cases, confirming the limitations of linear β approximations.

2. **FROG shows signal; SOUN does not**: The Random Forest model achieved ~30% R^2 in FROG but negative R^2 in SOUN, reflecting differences in liquidity structure.

3. **Polynomial Regression is most stable**: It consistently improved performance modestly across datasets, without overfitting like the Random Forest.

4. **Inverse-$g_t(X)$ Allocation is Effective**: Even with weak βt_t predictability, directly computed gt(X)g_t(X)gt(X) enables a dynamic execution plan that avoids peak impact times.

Key Findings

- **Linear β_t models are too simplistic**: Spread/depth ratios fail to capture intraday dynamics or order size effects.

- **Empirical slippage estimation is necessary**: Direct computation of $g_t(X)$ from multi-level LOBs is a more accurate approach.

- **Modeling β_t is feasible for liquid assets**: Especially in FROG, where LOB features correlate well with observed slippage.

- **Inverse-slippage allocation outperforms naive methods**: It adapts to market conditions and reallocates risk dynamically throughout the trading session.

Part Two: Mathematical Algorithm

Allocation Strategy Based on $g_t(X)$

$x_t = S \cdot [1/g_t(X)] / \sum 1/g_i(X)$

This formulation prioritizes time periods with lower slippage, adapting to intraday changes in liquidity. Unlike TWAP or linear spread-based schedules, this method avoids high-slippage windows, shifts volume to periods where the book is deep and spreads are tight, and accommodates sudden liquidity changes in a non-parametric way.

Goal

The optimization problem can be expressed as:

Minimize:$\sum$i=1N $g_t(x_i)$
Subject to:$\sum$i=1N x_i=S

Here, $g_{ti}(x_i)$ denotes the **temporary slippage cost function** for trading x_i shares at time t_i.

Inverse Slippage-Based Allocation Strategy

To address this, we adopt a **greedy adaptive strategy** that leverages real-time estimates of execution cost. Rather than modeling slippage from first principles or relying on static weights, we simulate market order execution at each time t_i, compute empirical slippage $g_{ti}(X)$ for a fixed hypothetical order size X, and use the inverse of these values to determine share allocation.

This results in the following closed-form allocation:

$x_t = S \cdot [1/g_t(X)] / \sum 1/g_i(X)$

This ensures that **periods with low slippage receive higher weight**, concentrating execution when liquidity is abundant. Additionally, the constraint $\sum x_i = S$ is always satisfied. Finally, the method remains **computationally efficient and interpretable**.

Implementation Workflow

1. **Inputs**:
   - Minute-level snapshots of Level-1 order book data (bid/ask prices and sizes).
   - Total order size S
   - A fixed order size X for slippage simulation
2. **For each time step tit_iti**:
   - Extract top-of-book values: bid_px_00, ask_px_00, bid_sz_00, ask_sz_00
   - Compute **mid-price**:
     $mid_i = [bid\_px\_00 + ask\_px\_00] / 2$
   - Simulate a market order of size XXX by walking the book and compute slippage:
     $g_{ti}(X) = VWAP(X) - mid_i$
3. **Aggregate Inverse Slippage**:
   sum_inverse=∑j=1N (1 / g*tj*(X))
4. **Allocate Shares**:
   $x_i = S \cdot [1/gt_i(X)] / sum\_inverse$
5. **Output**: Allocation vector x\mathbf{x}x

Tools & Techniques

- **Python Libraries**: pandas, numpy, matplotlib
- **Modeling**: Simulated slippage curves derived from real-time order book depth
- **Benchmarking**: Linear, Polynomial, and Random Forest regressions for modeling slippage coefficients β_t

Github Repo: https://github.com/Kenny9075/Blockhouse_Work_Trial_Task