TASK1: Modeling Temporary Impact Function gt(X) and Execution Strategy

Introduction

In modern trading, the distinction between market orders and limit orders defines not only execution certainty but also the cost incurred in terms of slippage. The task was to analyze order book dynamics and model the temporary impact function gt(X), which quantifies slippage as a function of order size X. A core objective was to develop a data-driven allocation strategy that minimizes total market impact when executing large order sizes across a trading day.

Given three ticker datasets (FROG, SOUN, and CRWV), we conducted a comprehensive study to model gt(X), compare common linear approximations with actual computed slippage, and design an allocation strategy based on real-time order book conditions.

Understanding gt(X) and the Limitations of Linear Models

The temporary impact function, gt(X), represents the slippage incurred when executing an order of size X at a particular time t. Traditionally, slippage is approximated using a linear model where gt(X) ≈ β_t X, where β_t is computed as the spread divided by the total depth (sum of bid and ask sizes). While simple and computationally light, this linear approximation has significant limitations:

1. **Static Nature**: β_t does not capture dynamic order book shifts, especially when liquidity is thin.
2. **Lack of Depth Consumption Modeling**: For larger order sizes, linear models fail to account for order book levels beyond Level 1, thus underestimating slippage.
3. **Order Size Non-linearity**: Real-world slippage increases disproportionately with larger X due to walking the book into worse price levels.

Our Modeling Approach

Instead of relying on β_t, we directly computed gt(X) using data available in the given CSV files. For every time snapshot (minute-wise granularity), we calculated slippage for varying hypothetical order sizes X. The slippage was defined as:

gt(X) = (Average Execution Price for X - Mid-Price)

Since Level 1 data limits depth to the best bid and ask, we constrained our X values to sizes within available liquidity (ask_sz_00 for buys). For each snapshot, we simulated market orders of size X, computed the execution price, and derived slippage by comparing it to the mid-price.

This computation was repeated for multiple order sizes (X = 10, 50, 100, 200, 500 shares), and we plotted gt(X) functions to observe how slippage grows with order size dynamically across the day.

Regression Modeling of β_t

To benchmark the conventional β_t estimation approach, we trained three regression models to predict β_t based on features like spread, imbalance, and mid-price:

1. **Linear Regression**
2. **Polynomial Regression (Degree 2)**
3. **Random Forest Regression**

Model Performance (FROG Ticker Example)

- **Linear Regression**: MSE = 4.328e-07, R^2 = 0.190
- **Polynomial Regression (deg=2)**: MSE = 4.251e-07, R^2 = 0.204
- **Random Forest**: MSE = 8.392e-07, R^2 = -0.570

Polynomial Regression marginally outperformed Linear Regression, but both models captured limited variance (R^2 ~0.20). Random Forest performed poorly, suggesting overfitting or insufficient feature representation for the non-linear model. This reinforced our hypothesis that β_t linearizations are oversimplifications of market impact.

Allocation Strategy Based on gt(X)

Given the task constraint to execute a total of S = 60,000 shares across N = 390 trading periods (1-minute intervals), we devised an allocation strategy proportional to the inverse of gt(X):

x_t = S * (1 / gt(X)_t) / ∑(1 / gt(X)_t)

This strategy inherently prioritizes periods with lower slippage (flattened gt(X)), thereby minimizing total impact. Unlike the naive β_t-based allocation, gt(X)-based allocation adapts to real-time liquidity conditions, ensuring that larger orders are executed when the order book can absorb them with minimal price disruption.

Analysis Across All Three Tickers

The same methodology was applied to the three tickers provided (FROG, SOUN, CRWV). Despite variations in liquidity profiles and intraday volatility, similar patterns were observed:

1. **gt(X) Non-Linearity**: Across all tickers, gt(X) increased non-linearly with order size, especially during illiquid periods.
2. **Model Performance Consistency**: Polynomial Regression consistently outperformed Linear Regression, albeit marginally (R^2 ranging between 0.18 and 0.22 across tickers).
3. **Random Forest Instability**: Random Forest models failed to generalize, producing negative R^2 values across datasets.
4. **Allocation Adjustments**: gt(X)-based allocation curves varied across tickers, highlighting the necessity of real-time adaptive strategies. Periods with wider spreads or thinner ask sizes received lower allocations, effectively mitigating slippage risk.

Key Findings

1. **β_t Linearizations are Insufficient**: Simple spread/depth ratios do not capture the true market impact, especially for varying order sizes and changing liquidity profiles.
2. **gt(X) Provides a More Accurate Impact Measure**: By directly computing slippage based on available liquidity, gt(X) models real-world conditions more effectively.
3. **Polynomial Regression Offers Marginal Gains**: While polynomial models offer slight improvements over linear regression, their predictive power remains limited in the absence of deeper order book data.
4. **Adaptive Allocation Minimizes Impact**: An inverse-g(X)-based allocation strategy dynamically responds to market conditions, distributing order flow optimally throughout the trading day.

Task 2: Mathematical Framework for Allocation Strategy (xi at ti)

Problem Formulation

Given a total order size S (e.g., 60,000 shares) that must be fully executed within N = 390 trading periods (1-minute intervals), the task is to allocate the order across the timeline while minimizing total temporary market impact. This is equivalent to choosing allocation vector **x ∈ R^N**, where x_i represents the quantity of shares to be executed at time t_i.

Objective: Minimize total slippage: **Total Impact = Σ gt_i(x_i)**

Subject to Constraint: **Σ x_i = S** (All shares must be bought by end of day)

Allocation Strategy Using Inverse gt(X)

We propose a greedy but effective approach:

1. For each time period t_i, compute gt(X) based on the prevailing order book snapshot.
2. Allocate a proportion of S inversely proportional to gt(X). Lower slippage periods get higher allocations.

Mathematically:

x_i = S * (1 / gt_i(X)) / Σ (1 / gt_j(X)) for all j ∈ [1, N]

This ensures that:

- Σ x_i = S (sum of allocations equals total shares)
- x_i is larger when gt_i(X) is small (cheaper execution periods)

Algorithm Outline

1. **Input**: Order book snapshots per minute, total shares S, N periods.
2. **For each t_i**:
    - Extract bid_px_00, ask_px_00, bid_sz_00, ask_sz_00.
    - Compute gt_i(X) = (ask_px_00 - mid_price).
3. **Compute sum_inverse_gtX = Σ (1 / gt_i(X))**
4. **For each t_i**:

- Compute x_i = S * (1 / gt_i(X)) / sum_inverse_gtX

5. **Output**: Allocation vector x = [x_1, x_2, ..., x_N]

Techniques and Tools Used

- Pandas and Numpy for data processing.
- Matplotlib for visualizing gt(X) slippage curves and allocation distributions.
- Linear and Polynomial Regression for benchmarking β_t models.

Observations

- This method dynamically adapts to real-time liquidity conditions.
- Allocation concentrates on high-liquidity periods with minimal market impact.
- Unlike static VWAP or TWAP, this approach reacts to microstructure nuances

Conclusion

Through this project, we demonstrated that while linear β_t models offer a simplistic view of market impact, direct computation of slippage via gt(X) provides a far more accurate and actionable representation of execution costs. Our allocation strategy effectively minimized temporary impact by prioritizing periods of high liquidity and low slippage.

While the models were constrained by Level 1 data and a small sample of tickers, the methodology is robust and can be scaled to larger datasets with deeper order book snapshots. The analysis validates the necessity of real-time, data-driven execution strategies over static linear approximations.

All code and analyses have been provided in separate Python notebooks for each ticker, detailing the modeling process, impact visualizations, and allocation strategies.