A Limit Order Book Simulation with an Advanced Glosten–Milgrom Model

IB9JH0 Programming for Quantitative Finance

Authors: Panyu Chen, Piotr Kurek, Naksh Patel, Mohammed Ali Yasin

Supervisor: Dr. Aurash Karimi

Student IDs: u5638368, u5676609, u2105616, u2105843

March 11, 2025

Abstract

This report describes our group's development and implementation of a limit order book trading system in C++. We incorporate an advanced variation of the Glosten–Milgrom model to simulate informed trading, noise trading, and a market maker updating beliefs about a latent fundamental value. Our work explores how limit and market orders interact in price-formation processes under different microstructural assumptions. We describe the design, data structures used, testing, and the parameters chosen for the model. We finally present our simulation results, including best bid/ask prices, spreads, the evolution of the market maker's belief, and a comparison with the underlying true asset value.

Contents

1 Introduction **1**

2 Methodology **2**

2.1 Project Workflow and Collaboration 2

2.2 Glosten–Milgrom Model and Extensions 3

2.3 Implementation 3

2.3.1 Object-Oriented Design 3

2.3.2 Data Structures 5

2.3.3 Matching Algorithm 6

2.4 Unit Tests 6

2.5 Simulation Approach and Parameter Choices 6

3 Results and Discussion **8**

4 Conclusion **10**

Bibliography **11**

List of Figures **12**

List of Tables **13**

Chapter 1

Introduction

Modern financial markets generally rely on a limit order book (LOB) structure to match buy and sell orders in real-time. By operating continuously, a LOB directly shapes the liquidity and price discovery processes across many venues. Our project sets out to achieve four primary goals. First, we implement a robust LOB in C++ for an order-driven market. Second, we integrate an advanced variant of the Glosten–Milgrom framework to handle informed and noise traders interacting with a market maker. Third, we conduct simulations exploring how prices, spreads, and belief updates evolve over time under uncertain fundamentals. Finally, we provide an in-depth discussion of our chosen parameters and reflect on their real-world relevance.

This project unites object-oriented principles with the Glosten–Milgrom model from the work of Glosten and Milgrom [1985]. By combining a C++ limit order matching engine with a model in which a market maker infers the true asset value in the presence of informed and uninformed orders, we gain insights into the ways liquidity and price formation arise and fluctuate. Our simulation traces the best bid/ask quotes, the spread, and a hidden true value that switches between “high” and “low” regimes.

Chapter 2

Methodology

2.1 Project Workflow and Collaboration

At the start, the team discussed individual interests in algorithmic trading and decided on the Glosten–Milgrom framework, motivated by the common experience with the contents of Cartea et al. [2015]. Subsequently, we considered each member's comfort with C++ and mathematical modelling. We decided on roles that would match a person's enthusiasm for a specific aspect of the project, alongside their existing skillset. One member implemented the `Order` class - covering both `LimitOrder` and `MarketOrder` - and wrote all unit tests for these constructs. Another contributed by creating the `Exchange` class and its respective tests, and also took responsibility for writing the Glosten–Milgrom simulation logic. A third member built the `Trader` class and associated tests, introduced exception handling to the entire codebase, and developed the Python script for plotting simulation outcomes. The final member focused on the `OrderBook` class, authored tests for it, implemented a detailed logging mechanism for the simulation and integrated all components into a cohesive system.

This arrangement ensured each contributor concentrated on distinct classes aligned with their strengths, while also learning from the collective review process. By apportioning responsibilities based on interest and expertise, the team maintained high standards of code quality and consistency.

2.2 Glosten–Milgrom Model and Extensions

Central to our simulation is an adaptation of the Glosten–Milgrom model, which offers a theoretical basis for how a market maker updates beliefs about the true price of an asset when facing buy or sell orders that may come from informed or uninformed traders. The model includes several key ingredients. First, the market maker's belief about whether the true asset value is high, denoted p_t, changes whenever a trade occurs. If a buy order arrives, the probability is revised upward, whereas a sell order triggers a downward revision. Second, the best bid/ask quotes depend on p_t, a parameter α that encodes the proportion of informed versus noise trading, and two possible values for the fundamental, v_{High} and v_{Low}. Third, the fundamental value itself is assumed to follow a two-state Markov chain. Transitions between the high and low states occur at specified probabilities, meaning the market maker's earlier conclusions may become obsolete if the latent state flips.

To keep markets liquid in our simulation, the market maker stands ready with large buy and sell limit orders at the best quotes. Whenever trades happen, these orders are cancelled and reposted at updated prices reflecting the new belief about the fundamental value. Such a design captures the essential spirit of the Glosten–Milgrom model, where the market maker's quotes continually adapt to order flow that may or may not be informed.

2.3 Implementation

2.3.1 Object-Oriented Design

We emphasised a clear object-oriented structure, presented in Figure 2.1, for both readability and maintainability. In particular, classes such as `OrderBook`, `Exchange`, and `Trader` enclose their data, with well-defined methods to add or remove orders, register traders, or modify internal states. We also employ inheritance by making `Order` a base class and `LimitOrder` and `MarketOrder` its specialised subclasses. This approach reduces duplication because we can handle distinct order types through a shared interface, relying on polymorphism to call the appropriate methods.

Aggregation and composition further appear in the way `Exchange` holds a registry of `Trader` objects and manages an `OrderBook`. Although a trader can issue orders, the primary responsibility for matching and record-keeping rests with the exchange. By encapsulating these functionalities, we preserve a clean boundary among the system's components.

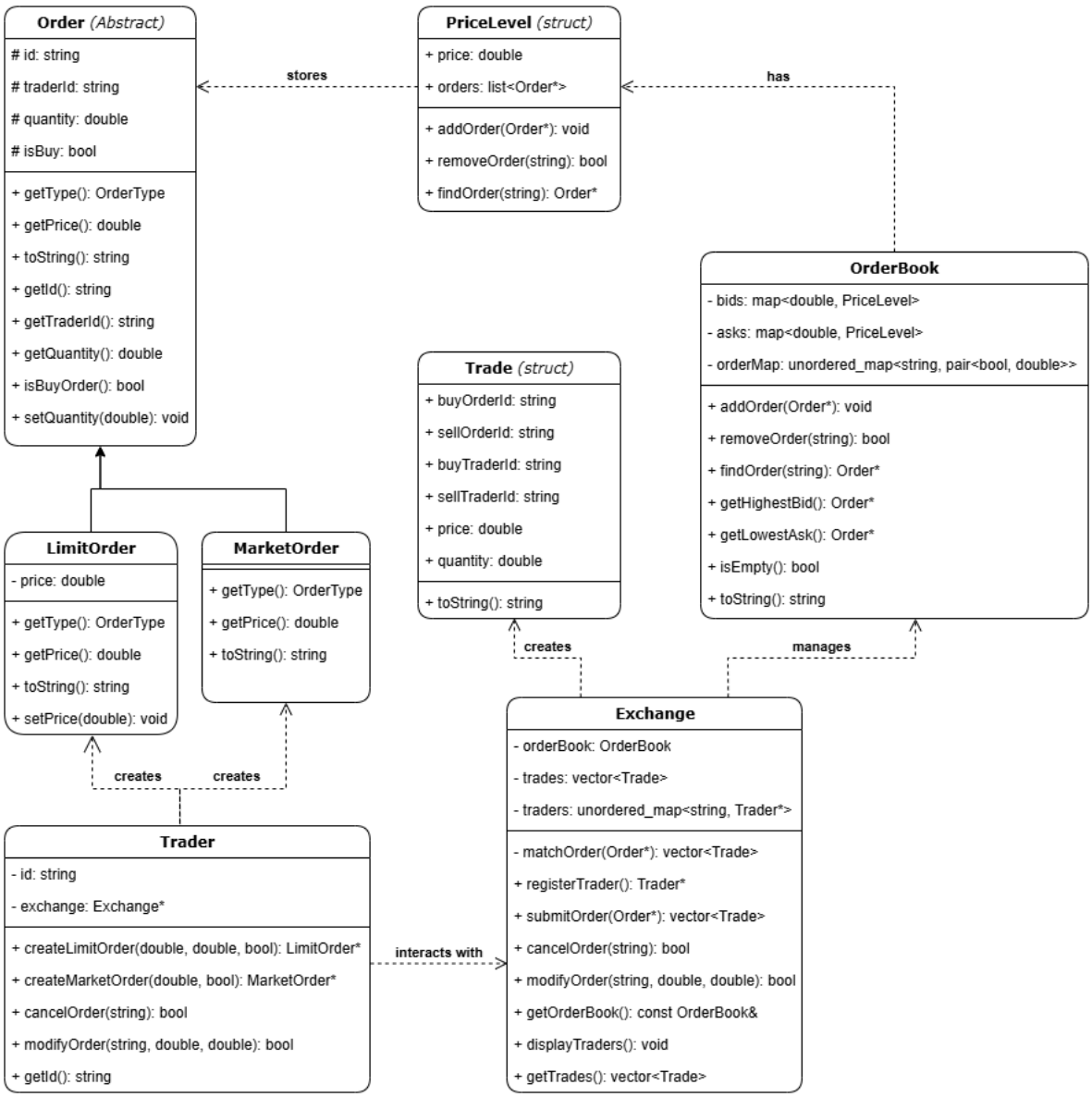

Figure 2.1: Diagram of the class structure.

2.3.2 Data Structures

A combination of data structures underlies our Limit Order Book (LOB). Within each price level, we use doubly linked lists (DLLs) to hold orders in the order of arrival. This helps facilitate FIFO matching, where the earliest order at a given price is executed first. Meanwhile, an AVL tree (`std::map`) tracks the price levels themselves and keeps them sorted so that the best bid and best ask can be found efficiently. This hybrid structure is essential for handling large numbers of orders without sacrificing performance.

Doubly Linked List for Order Storage

Each price level is backed by a DLL, which makes it straightforward to append new arrivals, remove filled or cancelled orders, and quickly traverse open orders at a specific price. Insertion and deletion each operate in constant time. Traversal is linear in the number of orders at that price, but in a typical LOB scenario, this approach is well-suited to prioritising time-based matching.

AVL Tree (Ordered Map) for Price Levels

The top-level organisation of distinct prices uses a self-balancing AVL tree. Prices serve as keys, mapped to their corresponding DLL of orders. Inserting a new price level or removing an obsolete one follows logarithmic complexity. We can then discover the lowest ask or highest bid in $\mathcal{O}(\log n)$ time. This structure prevents pathological slowdowns and handles dynamic changes gracefully.

Comparison of Data Structures

By uniting DLLs for orders within each price level and an AVL tree for the overall set of prices, we achieve an efficient LOB that can handle large volumes of operations. Table 2.1 provides a summary of key complexities.

Operation	**Doubly Linked List**	**AVL Tree (Ordered Map)**
Insert Order	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Delete Order	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Find Order	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Find Best Bid/Ask	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$

Table 2.1: Overview of complexity for key operations in our LOB data structures.

2.3.3 Matching Algorithm

The system enforces price-time priority. When a new order arrives, it targets the opposite side of the book (bids if the new order is a sell, asks if it's a buy). If it is a limit order that does not cross the best quote, it is simply placed in the order book. If, however, its limit price is equal to or better than the prevailing quote, the order will be matched and partially or wholly filled against existing orders, possibly removing them from the book. Market orders ignore pricing constraints and attempt immediate execution, draining as much of the best available liquidity as necessary to fulfil their quantity or until the book is exhausted.

2.4 Unit Tests

Thorough unit testing, performed via the Catch2 framework, underpins our confidence in the correctness of this LOB system. Tests focus on each core segment. The `Order` tests check that the creation of limit and market orders is valid and that exceptions are thrown for invalid inputs. The `OrderBook` tests verify insertion, removal, and retrieval logic and ensure the best bid/ask detection works properly. We examine the `Exchange` code for correct trader registration, order matching, partial and full fills, order cancellation, and modification. Finally, the `Trader` tests confirm that traders can create, and cancel orders successfully, reflecting changes in the book and trade records.

Empirical coverage analyses show that these tests capture the major functionality and edge cases, including error-handling scenarios. By rigorously evaluating each step of order creation, submission, matching, and removal, we find the system stable and well-prepared for our simulation experiments.

2.5 Simulation Approach and Parameter Choices

For our main experiments, we simulate up to $T = 200.0$ units of time in increments $\Delta t = 0.01$. A Poisson process with rate $\lambda = 80$ determines whether new orders arrive in each step. We suppose that with probability $p_{\text{Informed}} = 0.25$, an informed trader arrives, whereas a noise trader arrives otherwise. The fundamental value ($v_{\text{High}} = 120$ or $v_{\text{Low}} = 80$) flips randomly at a 2% chance per time step from one state to the other, forcing the market maker to reassess its quotes.

Market maker belief updates rely on $\alpha = 0.85$, which strongly weighs the possibility that any given trade is from an informed participant. Additionally, the belief p_t undergoes mean

reversion toward 0.5 at a rate of 0.1, ensuring that prolonged inactivity or contradictory order flow can gradually reset the maker's confidence. Order sizes average around 5.0 units (with a maximum of 50.0), and roughly 70% of noise traders use limit orders. A minimal transaction fee of 0.001 (0.1%) is also imposed for realism.

These parameters reflect a market that is quite busy, with frequent arrivals and a significant share of potentially informed trades. The wide gap between 80 and 120 is chosen to highlight how strongly the quotes can move when the maker updates its views, while the 2% flip rate produces occasional shifts in the underlying state and ensures that no single regime dominates for too long. The mechanics behind the simulation follow a Bayesian updating scheme that updates the probabilities at each time step [Touzo et al., 2020].

Chapter 3

Results and Discussion

We carried out the simulation for a total of 200 time units. The Python script generated Figure 3.1, revealing the major dynamics of the simulation. The top panel shows how the best bid (blue) and ask (red) evolve and reflects flips in the fundamental state. The second panel tracks the spread, which at times widens significantly when the maker's uncertainty about the true value grows. The third panel plots an exponential moving average (EMA) for the bid and ask, in addition to the actual underlying value (orange), showing that once the market detects a persistent buy (or sell) flow indicative of the higher (or lower) fundamental, prices drift toward that value. Finally, the bottom panel displays the market maker's belief probability, which oscillates amidst incoming trades. Whenever repeated orders come in on one side, the maker pushes p_t closer to 1 or 0, only to revert if contradictory trades later appear or if time passes without new updates.

These observations demonstrate that the LOB and matching engine remain responsive under high usage, the Glosten–Milgrom mechanism adapts beliefs in a plausible manner, and random flipping of the fundamental engenders dynamic variations in prices and spreads. Notably, the interplay of informed and noise orders forces the maker to keep revising quotes, generating realistic price fluctuations.

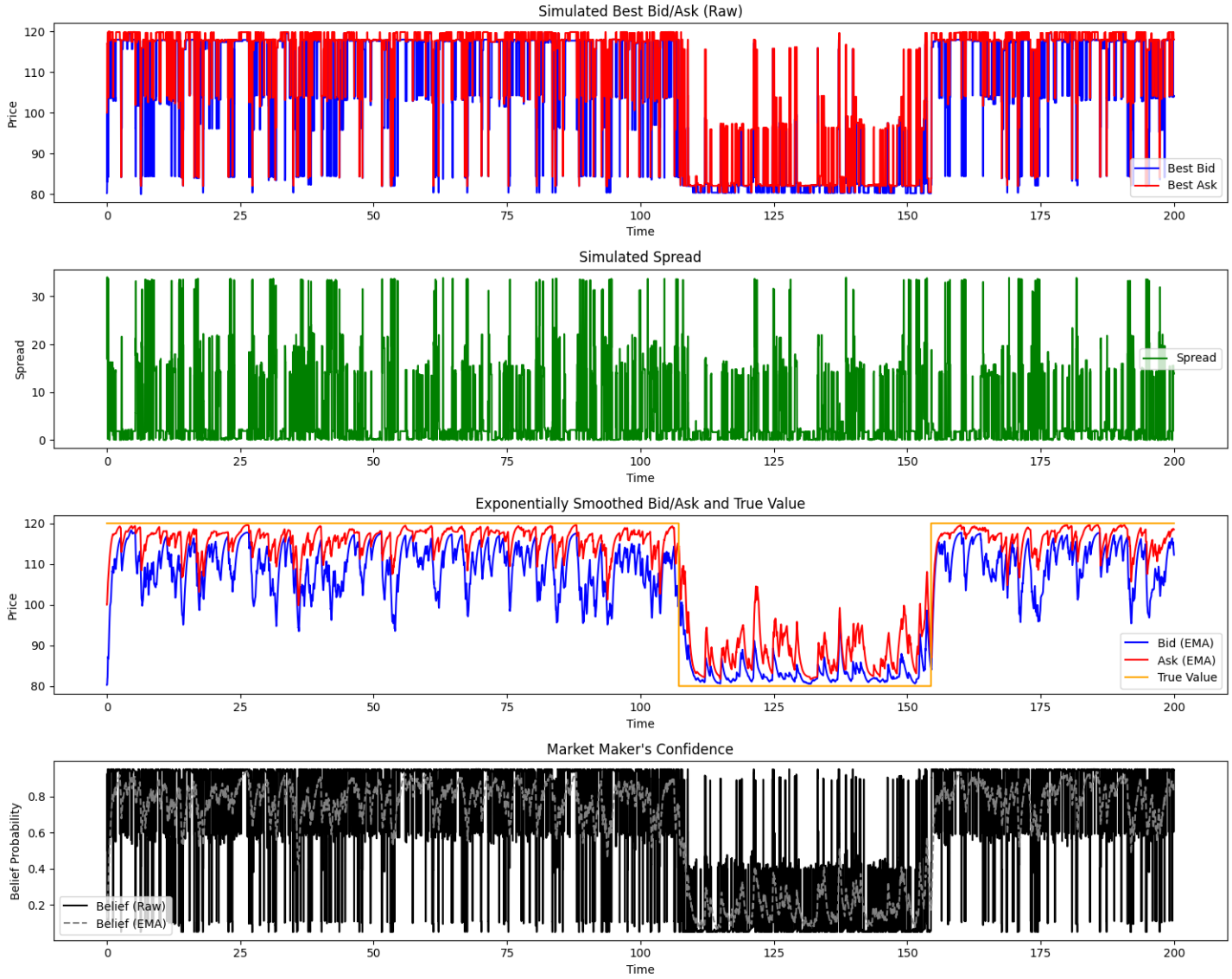

Figure 3.1: Simulation outputs, moving from top to bottom: (1) raw best bid (blue) and ask (red); (2) spread; (3) exponential moving average (EMA) of best quotes with the true value (orange); (4) market maker's belief (solid black) with an EMA (dashed grey).

Chapter 4

Conclusion

In summary, we have presented a C++ implementation of a limit order book integrated with an extended Glosten–Milgrom model. Our experiments highlight how a market maker, faced with potentially informed orders, adjusts quotes and beliefs over time. The price-time priority matching rules allow for partial and complete fills, and the presence of random arrivals, shifts in the true value, and transaction fees leads to spreads, volatility, and liquidity levels that mirror real-world phenomena in simplified form.

In the future, one could allow the share of informed traders or the fundamental states to vary more flexibly, implement more sophisticated noise-trader order strategies, or investigate how different parameter regimes alter short-term volatility. Adjusting the transaction fee structure or analyzing patterns of order cancellations are also relevant directions. Overall, this project lays out a framework that can be built upon to explore further aspects of market microstructure within an object-oriented, modular design.

References

Á. Cartea, S. Jaimungal, and J. Penalva. *Algorithmic and High-Frequency Trading*. Cambridge University Press, illustrated, reprint edition, 2015. ISBN 1107091144,9781107091146.

L. R. Glosten and P. R. Milgrom. Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. *Journal of Financial Economics*, 14:71–100, 1985.

L. Touzo, M. Marsili, and D. Zagier. Information thermodynamics of financial markets: the glosten–milgrom model, 2020. URL `https://doi.org/10.48550/arXiv.2010.01905`.

List of Figures

2.1 Diagram of the class structure. 4

3.1 Simulation outputs, moving from top to bottom: (1) raw best bid (blue) and ask (red); (2) spread; (3) exponential moving average (EMA) of best quotes with the true value (orange); (4) market maker's belief (solid black) with an EMA (dashed grey). 9

List of Tables

2.1 Overview of complexity for key operations in our LOB data structures. 5