University of Cape Town

Masters Dissertation

Market Simulations with a Matching Engine

Author:
Ivan Jericevich

Supervisor:
Prof. Tim Gebbie

A thesis submitted in fulfillment of the requirements
for the degree of Master of Statistics

in the

Advanced Analytics
Department of Statistical Sciences

August 21, 2022

Declaration of Authorship

I, Ivan Jericevich, declare that this thesis titled, Market Simulations with a Matching Engineand the work presented in it are my own. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University.
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
5. I have acknowledged all main sources of help.
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date:

UNIVERSITY OF CAPE TOWN

Abstract

Science

Department of Statistical Sciences

Master of Statistics

Market Simulations with a Matching Engine

by Ivan Jericevich

We demonstrate the CoinTossX Java web-application as a low-latency, high-throughput, open-source matching engine/artificial exchange/simulation platform and deploy it to a cloud environment for asynchronous order matching and submission in a controlled framework via two seperate simulation techniques — Hawkes processes and agent-based modelling. A 10-variate Hawkes model stress tests the software whilst measuring the extent to which a matching engine can cloud the modelling of underlying order submission and management processes in a continuous-double auction. Estimation and calibration to the subsequent trade-and-quote data results in a model specification statistically different from the original — providing insight into the limits of the software, inference conducted on HFT models and future market microstructure modelling considerations. An asynchronous ABM with interacting low-frequency liquidity takers and high-frequency liquidity-providers is subsequently formulated with the aim of producing realistic trading scenarios/price action without relying on restrictive modelling assumptions or additional sources of noise. The resulting simulations are shown to replicate many stylized facts along with non-trivial price-impact curves and we use this to argue for future simple, reactive/actor-based financial model specifications that mimics real-world work-flow and system implementation.

Acknowledgements

This dissertation is made possible through the continuous support of my supervisor, Associate Professor Tim Gebbie, who granted me the freedom to pursue challenging and ambitious research avenues. I would like to express my sincere gratitude for his support, insight, patience and, more importantly, for instilling in me a passion for statistical finance research and finance in general throughout my undergraduate and postgraduate carreer.

To Patrick Chang for his support and continuous devotion to the project. Without his work ethic, insight and huge contributions, this project would not have been possible.

My appreciation towards Dharmesh Sing's time spent helping me get familiar with CoinTossX and for being extremely helping and patient during our discussions. The skills gained from our discussions and work have proved to be invaluable to me.

Furthermore, I would like to thank members of the Statistical Finance Research Group at UCT — in particular Lionel Yelibi, Marcus Gawronsky, and Dr Etienne Pienaar — for their helpful feedback, probing questions, and constructive criticism.

We thank Turgay Celik and Brian Maistry WITS Mathematical Science Support and the TW Kambule Mathematical Sciences Laboratories for allowing us to use 4 former TACC Ranger blades for some of the simulation work. We thank Patrick Chang, Dieter Hendricks and Diane Wilcox for various discussions and advice with regards to the project. The historic snap-shot of part of the A2X transaction message data was kindly provided to us by A2X.

Lastly, I am grateful to my family Adrie, John and Ricky whose unfailing years of encouragement brought me to this stage of my carreer.

Related Publications and Pre-prints

Here the key papers, pre-prints and software archives authored by myself, Patrick Chang and Tim Gebbie upon which parts of the work are based:

1. Jericevich, Ivan, Patrick Chang, and Tim Gebbie (2020). *Comparing the market microstructure betweentwo South African exchanges.* arXiv:2011.04367 [q-fin.ST]. url:https://arxiv.org/abs/2011.04367.
   Reproducable from: https://github.com/CHNPAT005/PCIJAPTG-A2XvsJSE
2. Jericevich, Ivan, Dharmesh Sing, and Tim Gebbie (2021). *CoinTossX: An open-source low-latencyhigh-throughput matching engine.* arXiv:2102.10925 [cs.DC]
   Reproducable from: https://github.com/dharmeshsing/CoinTossX
3. *Simulation and estimation of a point-process market-model with a matching engine.* arXiv:2105.02211 [q-fin.TR].
   Reproducable from: https://github.com/IvanJericevich/IJPCTG-HawkesCoinTossX
4. *Simulation and estimation of an agent-based market-model with a matching engine.* arXiv:2108.07806 [q-fin.TR].
   Reproducable from: https://github.com/IvanJericevich/IJPCTG-ABMCoinTossX

Contents

List of Figures

List of Tables

List of Abbreviations

LOB	**L**imit **O**order **B**ook
TOB	**T**op **O**f **B**ook
BBO	**B**est **B**id and **O**ffer
ABM	**A**gent-**B**ased **M**odel
MMS	**M**odel **M**anagement **S**ystem
OMS	**O**rder **M**anagement **S**ystem
EMS	**E**xecution **M**anagement **S**ystem
HPC	**H**igh-**P**erformance **C**luster
TCP	**T**ransmission **C**ontrol **P**rotocol
JDK	**J**ava **D**evelopment **K**it
UDP	**U**ser **D**atagram **P**rotocol
SBE	**S**imple **B**inary **E**ncoding
JSE	**J**ohannesburg **S**tock **E**xchange
BDA	**B**roker **D**ealer **A**ccounting
TIF	**T**ime-**I**n-**F**orce
JVM	**J**ava **V**irtual **M**achine
FIX	**F**inancial **I**nformation e**X**change
FAST	**F**ix **A**dapted for **ST**reaming
MRS	**M**inimum **R**eserve **S**ize
MES	**M**inimum **E**xecution **S**ize
MO	**M**arket **O**rder
LO	**L**imit **O**rder
HL	**H**idden **L**imit Order
CO	**C**ancel **O**rder
SO	**S**top Market **O**rder
SL	**S**top **L**imit Order
OPG	At the **OP**ening**G**
GFA	**G**ood **F**or **A**uction
GFX	**G**ood **F**or Intraday(**X**) Auction
ATC	**A**t **T**he **C**lose
DAY	**DAY**
IOC	**I**mmediate **O**r **C**ancel
FOK	**F**ill **O**r **K**ill
GTC	**G**ood **T**ill **C**ancel
GTD	**G**ood **T**ill **D**ate
GTT	**G**ood **T**ill **T**ime
CPX	**C**losing **P**rice Cross(**X**)
DPR	**D**ynamic **P**rice **R**eference
SPR	**S**tatic **P**rice **R**eference
EMH	**E**fficient **M**arket **H**ypothesis
GPH	**G**eweke and **P**orter-**H**udak estimator
ADF	**A**ugmented **D**ickey-**F**uller test
GARCH	**G**eneralized **A**uto**R**egressive **C**onditional **H**eteroskedastic model
SMD	**S**imulated **M**inimum **D**istance

List of Symbols

p_x	Price of an order x
ω_x	Size/volume of an order x
t_x	Submission/execution time of an order x
$\mathcal{L}(t)$	Limit order book at time t
$\mathcal{A}(t)$	Ask side of a limit order book at time t
$a(t)$	Best ask price at time t
$n^a(p,t)$	Ask-side depth available at price p and time t
$\mathcal{B}(t)$	Bid side of a limit order book at time t
$b(t)$	Best bid price at time t
$n^b(p,t)$	Bid-side depth available at price p and time t
$s(t)$	Bid-ask spread at time t
$m(t)$	Mid-price at time t
$\psi(t)$	Micro-price at time t
π	Tick size of a limit order book
$\lambda^m(t)$	Conditional intensity function at time t for the mth process of a multivariate Hawkes process
$N^m(t)$	Counting process at time t for the mth process of a multivariate Hawkes process
μ^m	Deterministic base/background/exogenous intensity for the mth process of a multivariate Hawkes process
$\alpha^{m,n}$	mnth element of the matrix-valued scale/jump parameter of a multivariate Hawkes process for the influence of the nth event on the mth process
$\beta^{m,n}$	mnth element of the matrix-valued decay parameter of a multivariate Hawkes process for the influence of the nth event on the mth process
$\phi^{m,n}(t)$	mnth element of the matrix-valued kernel or excitation function of a multivariate Hawkes process
$\Lambda^m(t,t')$	Integrated intensity function or compensator for the mth process between time t and t' of a multivariate Hawkes process
$\Gamma^{m,n}$	mnth element of the matrix-valued branching factor/ratio of a multivariate Hawkes process
$\mathcal{L}(\theta)$	—
$S(\theta)$	The score function with respect to parameter θ
$\mathcal{I}(\theta)$	Fisher information with respect to parameter θ
$I(\theta)$	Observed Fisher information with respect to parameter θ
N^f_{LT}	ABM parameter — number of fundamentalist liquidity taker agents
N^c_{LT}	ABM parameter — number of chartist liquidity taker agents
N^{LP}	ABM parameter — number of liquidity provider agents
δ	ABM parameter — volume size aggression multiplier
κ	ABM parameter — placement depth multiplier
ν	ABM parameter — scaling factor for power-law volume order size
σ	ABM parameter — fundamentalists' value perception uncertainty for the trading day
$\boldsymbol{W}$	Inverse covariance matrix of empirical moments
$G(\boldsymbol{\theta})$	Matrix-valued mean distance between empirical and simulated moments
$\boldsymbol{m^e}$	Vector of empirical moments
$\boldsymbol{m^s}$	Vector of simulated moments
ω_{ij}	Daily-normalised volume for trade i on day j

Δp	Price change
α	Tail-index parameter for a power-law distribution

1 Introduction

This thesis considers order submission models in the context of an experimental order matching engine — CoinTossX (Sing 2017; Sing and Gebbie 2017; Jericevich, Sing, and Gebbie 2021; Jericevich, Chang, and Gebbie 2021a; Jericevich, Chang, and Gebbie 2021b). This is a modular, lightweight software/web-application designed by Sing and Gebbie (2017) and modified/maintained by Jericevich, Sing, and Gebbie (2021) to replicate traditional market-microstructure as closely as possible whilst maintaining high-throughput and low-latency. Simulations are carried out asynchronously using two popular methods — point processes and agent-based models.

Although each chapter hereafter has distinct research goals, the content is developed in such a way that each chapter may build on elements researched in the chapters that precede it. These chapters are structured as follows.

Some background to market microstructure, Hawkes processes and agent-based modelling is given in Chapter 2. If background knowledge is not required, the sections in Chapter 2 can be avoided without loss of intelligibility. Latest developments and the applications of topics covered, as well as the crucial literature informing this dissertation are outlined in Chapter 3.

Chapter 4 is dedicated to an outline of CoinTossX. More specifically, Section 4.2 gives a detailed description of the architecture and software construction. Section 4.4 presents the capabilities and features of CoinTossX. Section 4.5 gives the details of the extensive tests conducted to ensure the system meets the stringent latency and throughput requirements. Supplementary to the testing framework, Section 4.6 shows the results of the simulation of a simple Hawkes process for generating large volumes of market and limit orders.

Chapter 5 introduces a slightly more sophisticated multivariate Hawkes process for simulating HFT activity. Experimentation focuses on defining two slightly different models (Section 5.2) with the aim of determining the extent to which the matching engine distorts the underlying data generating process. Estimation of the output process is conducted in Section 5.3, afterwhich extensive statistical tests are applied in Section 5.4 to identify and explain the distortions of each process.

The implementation of an agent-based model in Chapter 6 demonstrates an approach close to an event/actor-based paradigm for modelling the LOB. The system implementation and agent rules are provided in Sections 6.3 and 6.2 respectively. Using this set-up, we conduct a sensitivity analysis (Section 6.4) on parameters and calibrate them against JSE Naspers data (Section 6.5). The resulting model parameters produce simulations that are able to replicate a number of constraining empirical facts (Section 6.6). More importantly, Section 6.7 outlines how this may set some of the ground work for concurrent/reactive/actor-based financial agent-based models.

Finally, Section 7 ends with some concluding remarks. In the appendix, Section A provides instructions for users to deploy and use the application on their local machine or on a remote server. Supplementary Pseudocode for both the Hawkes models and the agent-based model is provided in Section B. Section C gives additional sensitivity analysis results from the the agent-based model experiment.

2 Background

This chapter begins by introducing the key concepts relating to financial markets as complex systems as well as their underlying technological infrustructure, building up to an introduction to two widely used simulation techniques — agent-based modelling and point processes. The notational definitions provided here will be consistently referred to throughout the rest of the thesis.

2.1 Limit order books and order-matching systems

Electronic exchanges, at their most basic, are best described by a limit order book (LOB) and a matching engine. In what follows a basic description of the limit order book is provided along with some notational definitions following Gould et al. (2013).

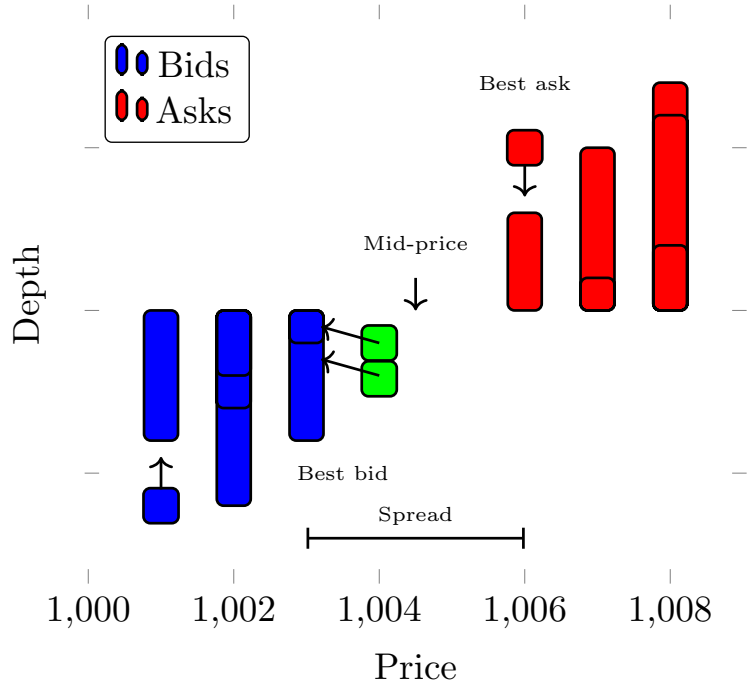


FIGURE 2.1: A basic visualisation of a limit order book snapshot.

An Exchange is a place where traders come together to exchange goods or services. Securities traded on a stock exchange include stock issued by listed companies, unit trusts, derivatives, pooled investment products and bonds. Orders submitted to an exchange are managed by a matching engine and a limit order book (a record/queue of outstanding orders to buy and sell stock in particular security). Order matching is the process by which an exchange pairs one or more unsolicited buy orders to one or more sell orders to make trades — which depends on the rules of the exchange. The continuous double auction is the most widely used method of price formation in modern financial markets. The auction is called "double" because traders can submit orders to both buy and sell, and "continuous" because they can do so at any time.

Definition 1 (Order)**.** An order $x = (p_x, \omega_x, t_x)$ submitted at time t_x with price p_x and size $\omega_x > 0$ is a commitment to buy (sell) up to ω_x units of the traded asset at a price no greater (less) than p_x.

The two types of markets that exist are the dealer market and the limit order market. The latter is most common and most electronic exchanges use limit order books to match orders placed by buyers and sellers. A limit order book is an example of a complex system rich in detailed, high-quality data. The LOB is defined on a fixed discrete grid of prices (the price levels). With reference to Figure 2.1, bids to buy and asks to sell are stored in the LOB as stacks of price levels where each price level comprises one or more orders with the same price and side. Bids are arranged in descending order of price while asks are arranged in ascending order of price. Therefore, the price of a traded security is not given by a unique price. It is actually a collection of prices, which are given by all the available orders in the LOB.

Definition 2 (Limit order book)**.** A LOB $\mathcal{L}(t)$ is the set of all active orders in a market at time t. The active orders in a LOB can be partitioned into the set of active buy orders $\mathcal{B}(t)$ and the set of active sell orders $\mathcal{A}(t)$. The LOB can then be considered as a set of queues at specified prices.

Definition 3 (Depth)**.** The bid-side and ask-side depth available at price p and time t are given by $n^b(p,t) := \sum_{x\in\mathcal{B}(t)|p_x=p} \omega_x$ and $n^a(p,t) := \sum_{x\in\mathcal{A}(t)|p_x=p} \omega_x$, respectively.

The size of the step (the difference between one price level and the next) is called the tick (*e.g.* 1 cent). Similarly, the units of order size may only occur in multiples of an asset's lot size. The lot size and tick size of a LOB are collectively called its resolution parameters.

Definition 4 (Tick)**.** The tick size π of a LOB is the smallest permissable price interval between different orders within it. All orders must arrive with a price that is specified to the accuracy of π.

Definition 5 (Lot)**.** The lot size σ of a LOB is the smallest ammount of the asset that can be traded within it. All orders must arrive with a size $\omega_x \in \{\pm k\sigma | k = 1, 2, \ldots\}$.

The first level or Top-Of-Book (TOB) shows the best bid and offer (BBO) available[1]. That is, the highest price at which a trader is willing buy and the the lowest price at which a trader is willing to sell, respectively.

Definition 6 (BBO)**.** The best bid is the highest stated price among all active buy orders at time t: $b(t) := \max_{x\in\mathcal{B}(t)} p_x$. The best ask is the lowest stated price among all active sell orders at time t: $b(t) := \min_{x\in\mathcal{A}(t)} p_x$.

The difference between the ask and the bid price[2] is called the quoted spread — measuring the cost to transact in that security. Alternatively, it can be considered as a measure of how highly the market values the immediacy and certainty associated with market orders versus the waiting and uncertainty associated with limit orders (Gould et al. 2013). The larger the spread, the larger the transaction costs.

Definition 7 (Spread)**.** The bid-ask spread at time t is $s(t) := a(t) - b(t)$

Another common object used when describing the LOB is the mid price — the arithmetic average of the best bid and the ask. It is often used to proxy for the true underlying price of the asset — the price for the asset if there were no explicit or implicit trading costs (and hence no spread).

Definition 8 (Mid-price)**.** The mid-price at time t is $m(t) := [a(t) + b(t)]/2$

Alternatively, the micro-price is used as a more subtle proxy for the asset's transaction cost-free price, as it measures the tendency that the price has to move either towards the bid or ask side as captured by number of shares posted, and hence indicates the buy (sell) pressure in the market. If there are a lot of buyers (sellers), then the micro-price is pushed toward the best ask/bid price to reflect the likelihood that prices are going to increase (decrease).

[1] Sometimes referred to as the inside quotes.

[2] The notation will often replace "best bid price" and "best ask price" simply with "bid" and "ask".

Definition 9 (Micro-price)**.** The micro-price at time t is a volume weighted average of the best bid and ask $\psi(t) := a(t)\frac{n^a(a(t),t)}{n^a(a(t),t)+n^b(b(t),t)} + b(t)\frac{n^b(b(t),t)}{n^a(a(t),t)+n^b(b(t),t)}$

A number of other important notational definitions that will be refernced throughout this paper are as follows.

Definition 10 (Order imbalance)**.** We define limit order imbalance $\rho(t)$ at time t as the ratio of the quoted volume imbalance between the bid side and the ask side to the total quoted volume: $\rho(t) := \frac{\sum_{x\in\mathcal{B}(t)}\omega_x - \sum_{x\in\mathcal{A}(t)}\omega_x}{\sum_{x\in\mathcal{L}(t)}\omega_x}$

Most distribution properties and stylized facts of financial market times-series are computed on the returns or log-returns of the price series (usually mid-price or micro-price). It is important to note that that exchange rate time series are not ergodic. This property states that averaging a stochastic process over time or over the ensemble are completely different operations which can rarely be interchanged. However, it was noted by Bernoulli (1954) that applying a specific transformation to a price time series can make it ergodic. This transformation is given by the lag-1 difference of log-prices.

Definition 11 (Log-returns)**.** Log returns are given by the tick-by-tick price fluctuations or lag-1 difference between consecutive prices $r(t_k) = \ln(\psi(t_{k+1})) - \ln(\psi(t_k))$

The two most common order types for buying or selling a stock in an exchange are limit orders, which specify prices and match against orders better than or equal to the specified price, and market orders, which executes/matches immediately against the best available price according to the engine's matching rules. The price on a limit order is known as the limit price and specifies the worst price at which the trader is willing to trade (highest for bids and lowest for offers). A limit order is not gauranteed to execute but ensures that an investor does not pay more than a pre-determined price for a stock. How an order behaves when it is not immediately filled varies between order types and time-in-force specification. An unmatched (standard) limit order rests in the LOB until its price is reached or until it is cancelled. Order types that cannot rest in the order book are either immediately filled or cancelled. In a special case, limit orders may be executed immediately if they cross the spread — that is, if the limit price is set to be equal to or better than the best on the contra side.

Market orders, on the other hand, are used when certainty of execution is a priority over the price of execution and are therefore generally classified as an aggressive order type. In fast-moving markets, the price paid or received may be quite different from the last price quoted before the order was entered. A trade happens when a buy or sell order is matched with one or more pre-existing orders in the LOB. Each trade transaction comprises two matched orders. An order may also execute against multiple orders at multiple price levels if its volume is large —- a behaviour described as "Walking the LOB". In this manner (for a matching engine adopting price-time priority) the incoming order will be matched with the LOs that offer the best price, then, if the quantity demanded is less than what is on offer at the best price, the matching algorithm selects the oldest LOs (the ones that were posted earliest) and executes them in order until the quantity of the MO is executed completely. If the MO demands more quantity than that offered at the best price, after executing all standing LOs at the best price, the matching algorithm will proceed by executing against the LOs at the second-best price, then the third-best and so on until the whole order is executed. The seperate order quantities executed at each price level constitute "split" trades whose execution prices are the corresponding price levels. The new bid (ask) price $b(t_x)$ immediately after the arrival of a sell (buy) market order x, respectively, are (Gould et al. 2013):

$$\begin{aligned} &\max(p_x, q), \quad \text{where} \quad q = \operatorname*{argmax}_{k'} \sum_{k=k'}^{b(t)} |n^b(k,t)| > \omega_x, \\ &\min(p_x, q), \quad \text{where} \quad q = \operatorname*{argmin}_{k'} \sum_{k=k'}^{a(t)} |n^a(k,t)| > \omega_x. \end{aligned} \tag{2.1}$$

Orders can be classified both in terms of a buyer and a seller or a maker and a taker. In general, orders that cross the spread and "take" liquidity[3] from the order book are known as "aggressive" orders. By contrast, orders that do not cross the spread are known as "passive" orders. Traders that submit aggressive orders are known as "takers". Traders that submit passive orders are known as "makers". Market-makers play an important role in providing liquidity to the market, because, under normal market conditions, they are willing to quote two-way prices on a continual basis, thus ensuring that there is always a maker available to take the opposing side of a taker's trade.

Exchanges also adopt different trading session at different time points, each of which may employ alternative trading rules. It is quite typical to have an initial auction and/or a closing auction, that is, an auction at the start of the trading day and/or an auction to close the market. An exchange can also use an auction after a market trading halt (*e.g.* after a volatility limit has been triggered) so as to smooth the transition back to active trading. Volatility auction calls have the effect of preventing large drops or spikes and stabalising prices.

The matching engine/system of an exchange is an electronic system that uses a well-defined algorithm that establishes when a possible trade can occur, and if so, which criterion is going to be used to select the orders that will be executed. The matching algorithms decide the efficiency and robustness of the order matching system. Exchanges aim to prioritize trades in a way that benefits buyers and sellers equally so as to maximize order volume — the lifeblood of the exchange. The computerized, order-matching systems of different exchanges use a variety of methods to prioritize orders for matching.

1. *First-in-first-out* or *price-time priority.* This means that orders are matched first by price, and then by time (order age). Generally, matches happen when compatible buy orders and sell orders for the same security are submitted in close proximity in price and time. For example, the earliest active buy (sell) order at the highest (lowest) price takes priority over any subsequent order at that price, which in turn takes priority over any active buy order at a lower (higher) price. This strategy motivates to narrow the spread, since by narrowing the spread the limit order is the first in the order queue. Similarly it discourages other orders to join the queue since a limit order that joins the queue is the last. It can however be more computationally demanding than pro-rata since market participants might want to place more small orders in different positions in the order queue.

2. *Pro-rata.* The system prioritizes active orders at a particular price proportional to the relative size of each order. In other words, MOs are matched against the posted LOs available at the best price, in proportion to the quantities posted. This strategy motivates other orders to join the queue with large limit orders. As a consequence, the cumulative quoted volume at the best price is relatively large. It does not motivate to narrow the spread in the natural way. This weakness is partially offset by introducing the time priority element for the first order that makes a new price.

3. *Constant-product automatic market maker (AMM).* Very recently in the decentralized finance space, many decentralized exchanges have been developed for peer-to-peer transactions of crypto-currencies. Briefly, the basis behind the dynamics and price-formation in this type of market is an equation for the ratio of liquidity available for one currency compared to another in a large "liquidty pool". As with a traditional order-book, the interaction of supply and demand that affects the relative amount of liquidity in the pool is what dictates the exchange rate.

The priority algorithms change the way traders behave. Traders will submit orders earlier if a price-time priority algorithm is used. Price-size and pro-rata priority will encourage traders to place large limit orders. These large orders will provide greater liquidity to the market

[3]Liquidity is a measure of the ability to cheaply transact a security without moving the inside quotes by much.

The role of time is fundamental in the usual price-time priority electronic exchange and, in a fragmented market[4], the issue becomes even more important. Traders need to be able to adjust their trading positions fast in response to or in anticipation of changes in market circumstances, not just at the local exchange but at other markets as well. Investors, particularly active investors and day traders, will look for ways to minimize inefficiencies in trading from every possible source. A slow order-matching system may cause buyers or sellers to execute trades at less-than-ideal prices, eating into investors' profits. The race to be the first in or out of a certain position is one of the focal points of the debate on the benefits and costs of high-frequency trading (Cartea, Jaimungal, and Penalva 2015).

The importance of speed permeates the whole process of designing trading algorithms, from the actual code, to the choice of programming language, to the hardware it is implemented on, to the characteristics of the connection to the matching engine, and the way orders are routed within an exchange and between exchanges. Exchanges, being aware of the importance of speed, have adapted and, amongst other things, moved well beyond the basic two types of orders (MOs and LOs). Some examples of the types of orders that you may find are: non-routable, pegged, hide-not-slide, hidden, iceberg, immediate-or-cancel, fill-or-kill, good-till-time, discretionary (Cartea, Jaimungal, and Penalva 2015). Some of these order types and more implemented in CoinTossX are discussed in more detail in Section 4.4.

Exchanges thus also control the amount and degree of granularity of the information you receive (*e.g.* you can use the consolidated/public feed at a low cost or pay a relatively much larger cost for direct/proprietary feeds from the exchanges). They also monetise the need for speed by renting out computer/server space next to their matching engines — a process called colocation. Through colocation, exchanges can provide uniform service to trading clients at competitive rates. Having the traders' trading engines at a common location owned by the exchange simplifies the exchange's ability to provide uniform service as it can control the hardware connecting each client to the trading engine, the cable (so all have the same cable of the same length), and the network. This ensures that all traders in colocation have the same fast access, and are not disadvantaged (at least in terms of exchange-provided hardware). Naturally, this imposes a clear distinction between traders who are colocated and those who are not. Those not colocated will always have a speed disadvantage. It then becomes an issue for regulators who have to ensure that exchanges keep access to colocation sufficiently competitive (Cartea, Jaimungal, and Penalva 2015).

Another dimension of importance when characterising an exchange is the degree (and cost) of transparency. Traders genercally distinguish between lit (open order book) from dark markets based on whether limit book information is publicly available or not (Cartea, Jaimungal, and Penalva 2015).

In summary, there are numerous benefits to understanding the LOB, namely, to better inform actions to take in given market situations, optimal execution strategies, market impact minimization, designing better trading algorithms, assessing market stability, etc (Gould et al. 2013). A key benefit of the LOB to traders in general is the fact that they allow some traders to demand immediacy, while simultaneously allowing others to supply it to those who later require it. They are also an effective way for patient traders to provide liquidity to less patient traders, even when liquidity is scarce.

2.2 Hawkes processes

In statistics and probability theory, a point process or point field is a collection of mathematical points randomly located on some underlying mathematical space such as the real line, the Cartesian plane, or more abstract spaces. Point processes can be used as mathematical models of phenomena or objects representable as points in some type of space (Daley and Vere-Jones 2003). If the space is time then the arrival times $T_1, T_2, \ldots$ will be realizations of this point process on the non-negative real line (where

[4]Orders submitted may be on one exchange, or aggregated across many exchanges. A fragmented market refers to the case where no single exchange/organization has enough influence to move the industry in a single direction

$\mathbb{P}[0 < T_1 \leq T_2 \leq \ldots] = 1)$. This provides a way of describing the timing and properties of events (however these events may be defined). The equivalent counting process $N(t)$ satisfying $N(0) = 0$ is a random, finite, right-continuous step function on time $t \geq 0$ whose value is the number of events of the point process by some time t (jump size 1)[5].

The Hawkes process, defined by Hawkes (1971) to model seismic events, is a class of multivariate counting/point processes which allow for event-occurrence clustering through a stochastic intensity vector. Fundamental to this point process is the simplest class of point processes, the Poisson process — either homogenous or non-homogenous and having the property of being memoryless. Poisson process events arrive randomly based on a constant intensity λ. The events of a Non-homogenous Poisson process, on the other hand, arrive randomly based on an intensity that is a function of time ($\lambda = \lambda(t)$). So, the Hawkes process is an extension of a non-homogenous Poisson process where events do not arrive independently. The reasoning behind this is that, in some applications, the arrival of an event increases the likelihood of observing events in the future. In other words, some types of events will cluster in time. Therefore a Hawkes process is simply a counting process having a random intensity function. More specifically, the stochastic intensity is made up from an *exogenous* component where the current intensity is not influenced by prior events and an *endogenous* component where prior events lead to an increased intensity. Within the endogenous component, *self-excitation* refers to an event type leading to more of the same event and *mutual-excitation* is an event driving the occurrence of other event types. Hence, the Hawkes model provides a way of modelling processes that are self exciting and/or mutually exciting in the multivariate case. That is, each event increases the likelihood of subsequent events of the same or different type for a period of time.

In many stochastic process models, a point process arises not as the primary object of study but as a component of a more complex model; often, the point process is the component that carries the information about the locations in time or space of objects that may themselves have a stochastic structure and stochastic dependency relations. From the point of view of point process theory, many such models can be subsumed under the heading of marked point processes (Daley and Vere-Jones 2003).

The Hawkes process is interesting for, among other things, its analytical advantages, simplicity, and for the fact that its likelihood is known in closed form. There do however exist other stochastic processes possessing this clustering property which typically involve more complex dynamics (refer to Hautsch (2011)). In the context of financial markets, high-frequency trading activity leads to time series of irregularly spaced points that show a specific type of clustering behaviour. So, because market participants exhibit herding behaviour, events tend to excite other events. This suggests the use of a Hawkes process which will be the first simulation model applied to CoinTossX for demonstration and stress testing purposes.[6]

[5] The "point" and "counting" process terminology is used interchangeably.

[6] For an extensive list of notable studies of Hawkes processes in finance refer to Bacry, Mastromatteo, and Muzy (2015).

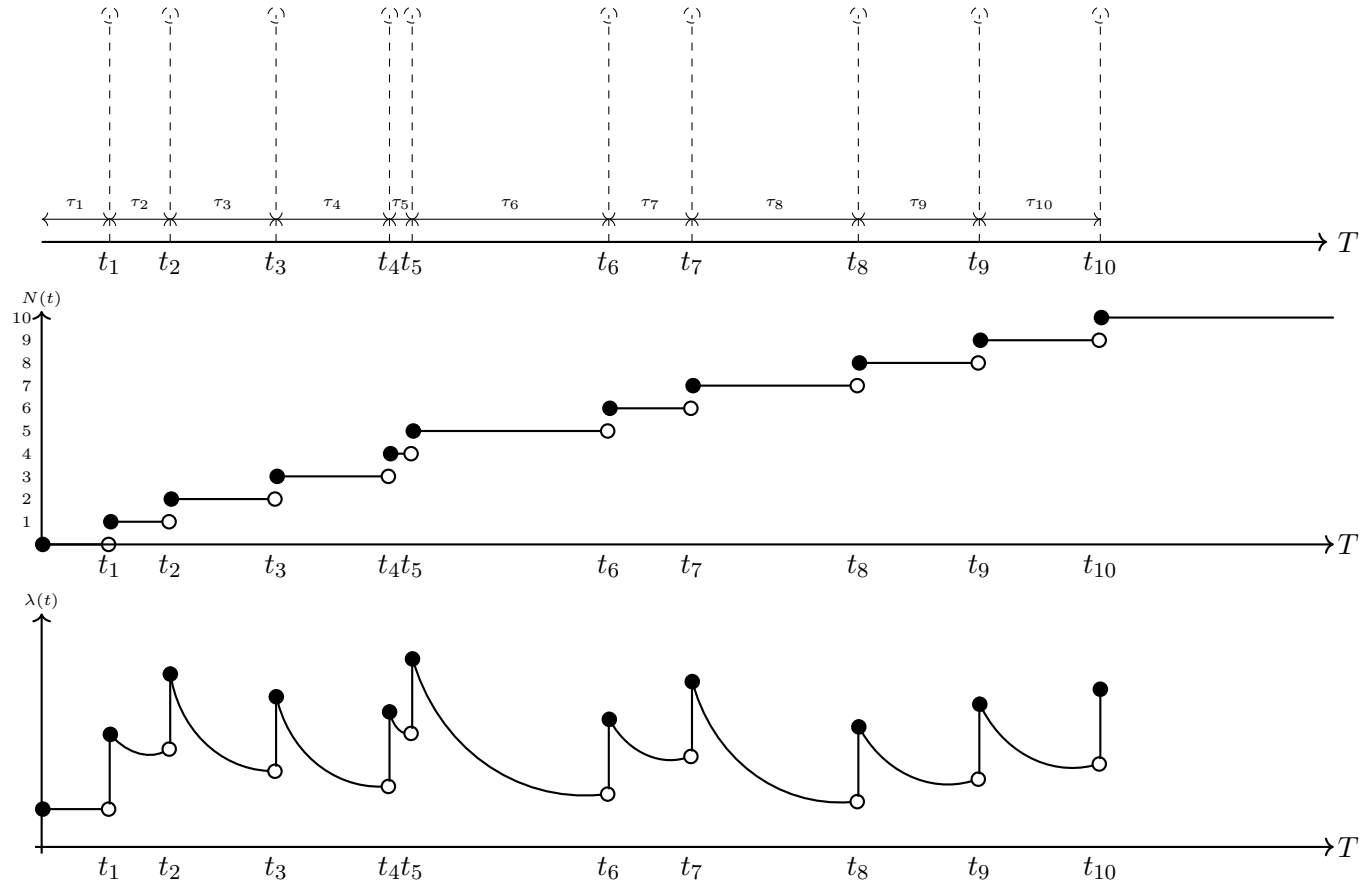


FIGURE 2.2: Demonstration of the arrival of 10-events from a self-exciting Hawkes process with an intensity-based representation.

2.2.1 Analytical framework

This section provides a basic definition of intensity-based Hawkes processes (which is by no means extensive) with a short description of cluster-based Hawkes processes. A M-variate Hawkes counting process $\boldsymbol{N}(t) = \{N^m(t)\}_{i=1}^M$ is characterised by a vector of conditional intensities (random variables) $\boldsymbol{\lambda}(t) = \{\lambda^m(t)\}_{m=1}^M$ ($t \in \mathbb{R}^+$), dependent on the history of the process up until point t, and defined as

$$\lambda^m(t) = \lim_{h \to 0} \mathbb{E}[N^m(t+h)) - N^m(t) | \mathcal{F}_t] \tag{2.2}$$

for the mth event where the filtration $\mathcal{F}_t$ (on the underlying probability space $(\omega, \mathcal{F}, \mathbb{P})$) stands for the information up to, but not including, time t[7]. The counting process $N^m(t)$ is determined by the intensity process $\lambda^m(t)$ and satisfies

$$\mathbb{P}[N^m(t+h) - N^m(t) = m | \mathcal{F}_t] = \begin{cases} \lambda^m(t)h + o(h) & m = 1, \\ o(h) & m > 1, \\ 1 - \lambda^m(t)h + o(h) & m = 0, \end{cases} \tag{2.3}$$

[7]For notational purposes, $\lambda^m(t)$ is substituted for $\lambda^m(t|\mathcal{F}_t; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ are the parameters defining the process.

where $o(h)$ is the Landau "little-o." The mutually and self-exciting intensity has the functional form

$$\begin{aligned}\boldsymbol{\lambda}(t) &= \boldsymbol{\mu}(t) + \int_0^t \boldsymbol{\phi}(t-s)\partial \boldsymbol{N}(s) \\ \lambda^m(t) &= \mu^m(t) + \sum_{n=1}^{M} \int_{-\infty}^{t} \phi^{mn}(t-s)\partial N^n(s) \\ &= \mu^m(t) + \sum_{n=1}^{M} \sum_{t_k^n < t} \phi^{mn}(t - t_k^n),\end{aligned} \tag{2.4}$$

where t_k^n is the arrival time of the nth event type for the kth process, $\boldsymbol{\mu}(t) = \{\mu^m(t)\}_{m=1}^M$ is a vector of deterministic base/background/exogenous intensities that are functions of time and $\boldsymbol{\phi}(t) = \{\phi^{mn}(t)\}_{m,n=1}^M$ is a matrix-valued kernel or excitation function which governs the dependency of prior events from the nth component to the mth component at the current time t.[8] The background intensities describe the arrival of events triggered by external sources. The kernel expresses the positive influence of the past events on the current value of the intensity process. So processes excite themselves and each other via the excitation function. Typically, the function ϕ is taken to be monotonically decreasing so that more recent events have higher influence on the current event intensity compared to events having occured further away in time. This requires the specification of the excitation function(s) of which there are many, to name a few:

1. Exponential kernel: $\phi(t) = \alpha e^{-\beta t}$ where $\alpha < \beta$; $\alpha \geq 0$ and $\beta > 0$. Preferred for it's analytical advantages and simplicity — in particular, the recursive relation when computing the likelihood function.
2. Power-law kernel: $\phi(t) = \frac{\alpha}{(t+\beta)^{\eta+1}}$ where $\alpha \geq 0$; $\beta > 0$; $\eta > 0$ and $\alpha < \eta\beta^\eta$. Popularised by Ozaki (1979) within the seismology literature.
3. Piecewise linear: Benefiting from being more computationally efficient than either of the above (Yosihiko Ogata and Akaike 1982).

In what follows, the exponential kernel is adopted. Consider first a univariate Hawkes process with a single event type. The mathematical representation of a Hawkes process with exponentially decaying intensity is derived from the following stochastic differential equation (SDE).

$$\partial\lambda(t) = \beta(\mu - \lambda(t))\partial t + \alpha \partial N(t) \tag{2.5}$$

where λ is the intensity of the process at $t = \infty$. In words, a jump of $N(t)$ at time t will increase the intensity which will inturn increase the probability of another jump (from Equation 2.3). Additionally, jumps cluster but do not "blow-up" because the drift ($\beta > 0$) becomes negative whenever the intensity $\lambda(t)$ is above $\mu > 0$, thereby preventing any explosion (Da Fonseca and Zaatour 2014). Applying Ito's lemma yields

$$\lambda(t) = e^{-\beta t}(\mu_0 - \mu) + \mu + \int_0^t \alpha e^{-\beta(t-s)} \partial N(s) \tag{2.6}$$

As $t \to \infty$ the impact of μ_0 diminishes. Similarly, setting $\mu_0 = \mu$ yields $\lambda(t) = \mu + \int_0^t \alpha e^{-\beta(t-s)}\partial N(s)$. For the multidimensional case this becomes

$$\begin{aligned}\lambda^m(t) &= \mu^m + e^{-\beta^{mn} t}(\mu_0^m - \mu^m) + \sum_{n=1}^{M} \sum_{t_k^n < t} \alpha^{mn} e^{-\beta^{mn}(t - t_k^n)} \\ &= \mu^m + \sum_{n=1}^{M} \sum_{t_k^n < t} \alpha^{mn} e^{-\beta^{mn}(t - t_k^n)}\end{aligned}$$

[8] Setting $\boldsymbol{\phi}(t) = 0$ is equivalent to a homogenous Poisson process.

where $\boldsymbol{\mu} = \{\mu^m\}_{m=1}^M$ is a vector of constant background intensities, and $\alpha^{mn} \geq 0$ and $\beta^{mn} > 0$ ($\alpha^{mn} < \beta^{mn}$) are the scale and decay influence parameters, respectively, of the nth event on the mth process.

Another equivalent view of the Hawkes process refers to the Poisson cluster process interpretation or immigration-birth representation (Hawkes and Oakes 1974). Although only the intensity-based formulation is adopted in Chapter 5, the cluster-based formulation proved to be very insightful for the point-process analysis conducted in Sections 5.4.1 and 5.4.2. As opposed to the intensity-based representation, a cluster based Hawkes representation can be constructed as a marked process which has a recursive branching structure. This representation benefits by avoiding simulating intensity paths and discretization bias but may only be applied to linear Hawkes processes. In this case the process can be visualised as having cluster centers (immigrants) which give rise to branching events (offspring). The offspring events are triggered by existing (previous) events in the process, while the immigrants arrive independently and thus do not have an existing parent event. The offspring are said to be structured into clusters, associated with each immigrant event — called the branching structure. The cluster representation states that the immediate offspring events associated with a particular parent arrive according to a non-homogeneous Poisson process with intensity ϕ. For example, events T_3 and T_6 in Figure 2.2 are realizations of a non-homogenous Poisson process with intensity $\phi(t - T_2)$ for $t > T_2$. Event T_2 is then said to be the immediate root/ancestor of those offspring. Similarly, T_2, T_3, T_4 form the cluster of offspring of T_1.

The key quantity describing this process is its branching factor/ratio, defined as the expected number direct offspring spawned by a single event/immigrant (shown in Equation 2.7). This intuitively describes the amount of events to appear in the process. This branching factor is defined and computed as the integral of the kernel over time.

$$\Gamma = \int_0^\infty \phi(s)\partial s \tag{2.7}$$

Define Z_j to be the random variable representing the number of offspring in the jth generation with $Z_0 = 1$ (since each immigrant represents generation zero and each cluster only has one immigrant). We have that $Z_1 \sim Poi(\Gamma)$ where the mean Γ is the branching ratio. Knowledge of this branching ratio can inform simulation algorithms since, for example, for each immigrant i, the times of the first-generation ($j = 1$) offspring arrivals — conditioned on knowing the total number of them Z_1 — are each i.i.d. with density $\phi(t - t_i)/\Gamma$.

The branching ratio also gives an indication about whether the expected number of offspring associated with each offspring is finite ($\Gamma < 1$ — bounded set of offspring) or infinite ($\Gamma > 1$ — unbounded set of offspring). To see this, the expected number of total events in the cluster is given by

$$\mathbb{E}[\sum_{i=0}^{\infty} Z_i] = \sum_{i=0}^{\infty} \mathbb{E}[Z_i] \tag{2.8}$$

Noting that each previous generation A_{i-1} has on average Γ children events leads to the following recursive relationship:

$$\mathbb{E}[Z_i] = \mathbb{E}[Z_{i-1}]n = \mathbb{E}[Z_{i-2}]n^2 = \ldots = \mathbb{E}[Z_0]\Gamma^i = \Gamma^i \tag{2.9}$$

Therefore,

$$\mathbb{E}[\sum_{i=0}^{\infty} Z_i] = \sum_{i=0}^{\infty} \Gamma^i = \begin{cases} \frac{\Gamma}{1-\Gamma} & \Gamma < 1 \\ \infty & \Gamma \geq 1 \end{cases}. \tag{2.10}$$

So for $\Gamma \geq 1$ an immigrant would generate infinitely many descendents on average.

The branching ratio is also a probablitly and is intuitively understood as the ratio of total offspring to the size of the entire family (*i.e.*, the toal offspring plus the original immigrant):

$$\frac{\mathbb{E}[\sum_{i=0}^{\infty} Z_i]}{1 + \mathbb{E}[\sum_{i=0}^{\infty} Z_i]} = \frac{\frac{\Gamma}{1-\Gamma}}{1 + \frac{\Gamma}{1-\Gamma}} = \Gamma \tag{2.11}$$

Therefore any Hawkes process arrival selected at random was either generated endogenously with probability Γ or exogenously with probability $1-\Gamma$ (Laub and Phil 2014).

Estimation and calibration

Common calibration[9] procedures include maximum likelihood estimation (Ozaki 1979; Muni Toke and Pomponio 2011) and method of moments estimation (Da Fonseca and Zaatour 2014). This section considers only likelihood estimation.

We wish to obtain estimates for the parameters $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$ where $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ are matrices and $\hat{\boldsymbol{\lambda}}$ is a vector. Here we have a single exponential kernel ($P = 1$ in Muni Toke and Pomponio (2011)) and $\hat{\boldsymbol{\lambda}}$ is a vector of constants (can be genralized to be a function of time). Let $\{t_n\}_{n=1}^N$ be the ordered pool of arrival times for all events (total of N arrivals from all events) with $t_1 = 0$ and $t_N = T$ and let $\{N(t_n)\}_{n=1}^N$ be the corresponding counting process. Then the log-likelihood of a multidimensional Hawkes process can be computed as the sum of the likelihoods for each process.

$$\begin{aligned}
\ell(\boldsymbol{\theta}; \{t_n\}_{n=1}^N) &= \ln \mathscr{L}(\boldsymbol{\theta}; \{t_n\}_{n=1}^N) \\
&= \sum_{m=1}^M \ell^m(\boldsymbol{\theta}, \{t_n\}_{n=1}^N) \\
&= \sum_{m=1}^M \ln \left[\left(\prod_{n=1}^N \lambda^m(t_n) \right) \exp\left(-\int_0^T \lambda^m(s) \partial s \right) \right]
\end{aligned}$$

The last line follows from Daley and Vere-Jones (2003). It follows that the log-likelihood for process/event m is

$$\begin{aligned}
\ell^m(\boldsymbol{\theta}; \{t_n\}_{n=1}^N) &= \int_0^T (1 - \lambda^m(s)) \partial s + \int_0^T \ln \lambda^m(s) \partial N^m(s) \\
&= \int_0^T (1 - \lambda^m(s)) \partial s + \sum_{n=1}^N \ln \lambda^m(t_n) \\
&= T - \Lambda^m(0,T) + \sum_{n=1}^N z_n^m \ln \left[\mu^m + \sum_{j=1}^M \sum_{t_k^j < t_n} \alpha^{mj} e^{-\beta^{mj}(t_n - t_k^m)} \right]
\end{aligned}$$

where $\Lambda^m(0,T) = \int_0^T \lambda^m(s) \partial s$ is the integrated intensity or compensator and $z_n^m = \begin{cases} 1 & \text{event } t_n \text{ is of type } m \\ 0 & \text{Otherwise} \end{cases}$.

Prior to the breakthrough from Ozaki (1979), Hawkes processes were exteremely computationally intensive to callibrate. A recursion relation for computing the likelihood of a multivariate Hawkes process was found by Ozaki which reduces its computation time significantly. Let

$$\begin{aligned}
R^{mj}(l) &= \sum_{t_k^j < t_l^m} e^{-\beta^{mj}(t_l^m - t_k^j)} \\
&= \begin{cases} e^{-\beta^{mj}(t_l^m - t_{l-1}^m)} \sum_{t_k^j < t_{l-1}^m} e^{-\beta^{mj}(t_{l-1}^m - t_k^j)} + \sum_{t_{l-1}^m \le t_k^j < t_l^m} e^{-\beta^{mj}(t_l^m - t_k^j)} & m \neq j \\ e^{-\beta^{mj}(t_l^m - t_{l-1}^m)} (1 + \sum_{t_k^j < t_{l-1}^m} e^{-\beta^{mj}(t_{l-1}^m - t_k^j)}) & m = j \end{cases} \\
&= \begin{cases} e^{-\beta^{mj}(t_l^m - t_{l-1}^m)} R^{mj}(l-1) + \sum_{t_{l-1}^m \le t_k^j < t_l^m} e^{-\beta^{mj}(t_l^m - t_k^j)} & m \neq j \\ e^{-\beta^{mj}(t_l^m - t_{l-1}^m)} (1 + R^{mj}(l-1)) & m = j \end{cases}
\end{aligned}$$

[9] Calibration refers to a reverse process to regression, where a known observation of the dependent variable (price) is used to determine the corresponding explanatory variables (parameters).

Substituting back, the log-likelihood can be written as

$$\ell^m(\boldsymbol{\theta}; \{t_n\}_{n=1}^N) = T - \Lambda^m(0,T) + \sum_{t_l^m} \ln \left[\lambda^m + \sum_{j=1}^M \alpha^{mj} R^{mj}(l) \right] \tag{2.12}$$

and $R^{mj}(0) = 0$. z_n^m falls away since we only sum over the arrivals which are from event m (t_l^m). $\Lambda^m(0,T)$ can be calculated by simplifying the integrated instensity between two consecutive events (t_{n-1}^m and t_n^m) for the mth process.

$$\begin{aligned}
\Lambda^m(t_{n-1}^m, t_n^m) &= \int_{t_{n-1}^m}^{t_n^m} \lambda^m(s) \partial s \\
&= \int_{t_{n-1}^m}^{t_n^m} \mu + \sum_{j=1}^M \sum_{t_k^j < s} \alpha^{mj} e^{-\beta^{mj}(s - t_k^j)} \partial s \\
&= \int_{t_{n-1}^m}^{t_n^m} \mu \partial s + \int_{t_{n-1}^m}^{t_n^m} \sum_{j=1}^M \sum_{t_k^j < t_{n-1}^m} \alpha^{mj} e^{-\beta^{mj}(s - t_k^j)} \partial s + \int_{t_{n-1}^m}^{t_n^m} \sum_{j=1}^M \sum_{t_{n-1}^m \le t_k^j < t_n^m} \alpha^{mj} e^{-\beta^{mj}(s - t_k^j)} \partial s \\
&= \int_{t_{n-1}^m}^{t_n^m} \mu \partial s + \sum_{j=1}^M \sum_{t_k^j < t_{n-1}^m} \frac{\alpha^{mj}}{\beta^{mj}} \left[e^{-\beta^{mj}(t_{n-1}^m - t_k^j)} - e^{-\beta^{mj}(t_n^m - t_k^j)} \right] + \\
&\quad \sum_{j=1}^M \sum_{t_{n-1}^m \le t_k^j < t_n^m} \frac{\alpha^{mj}}{\beta^{mj}} \left[1 - e^{-\beta^{mj}(t_n^m - t_k^j)} \right] \\
&= \int_{t_{n-1}^m}^{t_n^m} \mu \partial s + \sum_{j=1}^M \frac{\alpha^{mj}}{\beta^{mj}} \left[R^{mj}(n-1)(1 - e^{-\beta^{mj}(t_n^m - t_{n-1}^m)}) + \sum_{t_{n-1}^m \le t_k^j < t_n^m} (1 - e^{-\beta^{mj}(t_n^m - t_k^j)}) \right]
\end{aligned}$$

By summing the above result over $(0,T)$ and noting that $R^{mj}(0) = 0$ we have that

$$\begin{aligned}
\Lambda^m(0,T) &= \sum_{n=2}^N \Lambda^m(t_{n-1}^m, t_n^m) \\
&= T\mu + \sum_{n=1}^N \sum_{j=1}^M \frac{\alpha^{mj}}{\beta^{mj}} (1 - e^{-\beta^{mj}(T - t_n)})
\end{aligned} \tag{2.13}$$

This leads to the final form of the partial log-likelihood.

$$\ell^m(\boldsymbol{\theta}; \{t_n\}_{n=1}^N) = T - T\mu - \sum_{n=1}^N \sum_{j=1}^M \frac{\alpha^{mj}}{\beta^{mj}} (1 - e^{-\beta^{mj}(T - t_n)}) + \sum_{t_l^m} \ln \left[\mu^m + \sum_{j=1}^M \alpha^{mj} R^{mj}(l) \right] \tag{2.14}$$

Therefore the maximum likelihood estimates are obtained by minimizing Equation 2.14 with respect to $\hat{\boldsymbol{\theta}}$ over the parameter space $\boldsymbol{\Theta}$ ($\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmin}}\ \ell(\boldsymbol{\theta}; \{t_n\}_{n=1}^N)$). This negative log-likelihood objective, however, is associated with a couple of common optimization problems, namely: non-linearity, non-convexity and multiple local minima. This implies that maximum likelihood estimation is unlikely to reach the global minimum and that the objective can only be minimized with non-linear optimization routines such as L-BFGS or Nelder-Mead. Many of these routines require the specification of a set of initial values. If the different optimization routines produce a consistent set of parameters then one can conclude with high certainty that the solution is infact the global minimum. An additional issue commonly associated with Hawkes processes is that of edge effects. The underlying process, observed in the interval $[0,T]$, may have started sometime prior to time $t = 0$. Since events cluster,

it may be the case that there were events before time $t = 0$ which could have generated offspring in the interval $[0, T]$ and had a significant impact on the process in the period $[0, T]$. In that case the events' intensities are not considered in the calibration process — referred to as edge effects (Daley and Vere-Jones 2003). Thirdly, the log-likelihood, requiring the calculation of a double summation, can be particularly computationally intensive. To summarise, issues related to the estimation of Hawkes process parameters include local maxima, edge effects, bias, and computational bottlenecks.

Simulation

This section demonstrates the intensity-based thinning method (Lewis and Shedler 1979; Yosihiko Ogata 1981; Dassios, Zhao, et al. 2013) for simulating random inter-arrival times $\tau_1, \tau_2, \ldots$ according to callibrated parameters and intensity function $\boldsymbol{\lambda}(t)$. The Thinning Algorithm 1 for Hawkes processes is derived by first considering the properties of a homogenous Poisson process. The inter-arrival times of a homogenous Poisson process with rate λ follow an exponential distribution.

$$\tau \sim Exp(\lambda) \implies f_\tau(t) = \lambda e^{-\lambda t} \implies F_\tau(t) = 1 - e^{-\lambda t} \tag{2.15}$$

The above closed form expressions imply that one can use the inverse transform method to sample waiting times. That is, we can sample from $u \sim U(0, 1)$ and compute $\tau = F_\tau^{-1}(u) = \frac{-\ln(u)}{\lambda}$. The thinning property of a Poisson process with rate λ states that it can be split into two independent processes with intensities λ_1 and λ_2 so that $\lambda = \lambda_1 + \lambda_2$. Given this, sampling from a non-homogenous Poisson process with intensity function $\lambda(t)$ is a matter of thinning a homogenous Poisson process with intensity $\lambda^* \geq \lambda(t)\ \forall\ t$. This can be applied to a Hawkes process by noting that for any λ_t we can find a constant λ^* such that $\lambda^* \geq \lambda(t)$ on a given time interval. For a monotonically decreasing kernel function, the upper bound of event intensity between two consecutive event times $[t_i, t_{i+1})$ is $\lambda(t_i)$. Having already sampled events $t_1, t_2, \ldots, t_i$ we set the start time counter to $T = t_i$ and sample an inter-arrival time τ using equation 2.15 with $\lambda^* = \lambda(T)$. The time counetr is reset to $T = T + \tau$. This inter-arrival time is accepted according to the ratio of the true event rate to the thinning rate λ^* and the event time is recorded as $t_{i+1} = T$. Otherwise the above process is repeated until one is accepted. Note that in the case of rejection, the time counter is still updated.

Algorithm 1: D-variate Hawkes process simulation by thinning algorithm as introduced by Lewis and Shedler (1979) and modified by Yosihiko Ogata (1981).

Input: $\boldsymbol{\lambda}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, t_N
Result: Arrival times $\{t_n\}_{n=1}^N$ of N jumps
Initialize $T = 0$, $\lambda^j(T) = \lambda^j\ \forall\ j \in 1, \ldots, D$
while $i \leq N$ **do**
  Set the upper bound of Poisson intensity $\lambda^* = \lambda_T$
  Sample the inter-arrival time: draw $u \sim U(0, 1)$ and compute $\tau = F_\tau^{-1}(u) = \frac{-\ln(u)}{\lambda}$
  Update the current time: $T = T + \tau$
  Draw $s \sim U(0, 1)$
  if $s \leq \frac{\lambda_T}{\lambda^*}$ *and* $T < t_N$ **then**
    Accept the sampled arrival time: set $T_i = T$ and $i = i + 1$
  end
end

Validation

Standard tests can be applied to check the goodness-of-fit of a Hawkes process calibrated to empirical data based on the result that the integrated intensity $\Lambda^i(t^i_{n-1}, t^i_n)$ is a time interval of a homogenous Poisson process (Random Time Change Theorem 1). This is a result of the multivariate random time change theorem of Bowsher (2007).

Theorem 1. Theorem 4.1 of Bowsher (2007) (rephrased):
Consider the M sequences of generalised residuals $\{e_i^{(m)}(\theta)\}_{m=1}^{M}$, where

$$e_i^{(m)}(\theta) = \int_{t_i^m}^{t_{i+1}^m} \lambda^m(s|\theta)\partial s. \tag{2.16}$$

When θ is the true set of parameters, then each sequence $e_i^{(m)}(\theta)$ will be an independently distributed exponentially random variable with unit mean.

The inter-arrivals $\{\Lambda^i(t_{n-1}^i, t_n^i)\}_{n=2}^{N}$ can thus be tested for being exponentially distributed using a QQ-plot and Kolmogorov-Smirnov test. The Kolmogorov-Smirnov test is based on the maximal discrepancy between the empirical cumulative distribution and the exponential cumulative distribution with unit rate. Additionally, the memoryless property of the process must be checked by a Ljung-Box test for independence. The Ljung-Box test examines the null hypothesis of an absence of auto-correlation in a given time-series. The independence property can be checked qualitatively by plotting the pairs (U_k, U_{k+1}), as a check for autocorrelations, where $U_k = F_{Exp(1)}(\Lambda^i(t_n^i, t_{n-1}^i))$. If integrated intesnities are independent then these plots should show a random scatter. These intensities are calculated recursively according to the recursive integrated intensity equation (2.13).

2.3 Agent-based modelling

Financial Agent-Based Models (ABMs) are characterised by a bottom-up approach to modelling complex systems by attributing market dynamics and emergent phenomena to relatively simple interactions of trader agents without the need to explicitly program or mechanistically define a vast web of complex generative models to capture all the nuances of real financial markets. ABMs benefit from not assuming agents are able to solve complex problems in order to determine their rational response, but instead assume that different agents follow simple rules in order to cope with their overly complex environments. The concept of bounded rational agents has been introduced by Simon (1955) and Simon (1957). Under uncertainty, bounded rational agents may act irrationally and do not solve an optimization problem, but rather look for a satisfactory solution which is near the optimum. This concept is heavily influenced and supported by behavioral finance. Thus, the deviations of financial agents to the optimal solution can be accounted for behavioral biases of agents (Kahneman and Tversky 1979).

Successfully calibrating ABMs to financial time series (Platt 2020) can allow for inference about the factors determining the price behaviour observed in the real world, provided that parameter estimates are sufficiently robust. These models are appealing for, among other things, being able to produce reasonably realistic financial time series that can be used to inform market regulation decisions, explain market phenomena, and to provide test-beds for testing trading strategies in an artificial exchange. In recent years, ABMs became popular as a tool to study macroeconomics — specifically, the impact of trading taxes, market regulatory policies, quantitative easing, and the general role of central banks. ABMs can also play an important role in analysis of the impact of the cross-market structure (Lussange et al. 2018).

ABMs are, however, not without their pitfalls (LeBaron 2006; Lussange et al. 2018; Platt 2020): computational cost, validation and calibration challenges, a bias towards inductive decision making, endemic parameter degeneracies, and their tendency to focus on mechanistic behaviours and rules, rather than interaction dynamics and learning. More specifically, a major underlying tenet of some closed form agent-based modelling philosophies is that studying every single element is sufficient to understand the system as a whole (Wang et al. 2018). While this attempt is able to derive analytical solutions for subsequent analysis, it frequently fails to account successfully for stylized facts. This is largely because various models in this framework rely heavily on many unrealistic assumptions, such as market clearing, market convergence to equilibrium prices, perfect information, and rationality, whilst ignoring the emergent characteristics of agents' interactions and their diverse strategies (Wang et al. 2018).

Despite these challenges, their potential to link the micro-level rules of investors behaviour with the macro-behaviour of asset prices in real market is compelling, in part due to the apparent expansive amount of data that is collected from financial markets. However, a key concern remains the scientific costs or impact of losing the realism of high-concurrency adaptive interactions between strategic agents in a sufficient reactive framework, for the computational convenience of time-synchronised bottom-up rule based approaches. Here we have decided to first focus on the matching engine framework, and separate it entirely from the agent generation framework; this may ensure that high-concurrency and low latency features of real markets are not lost.

2.3.1 Calibration

The three most common calibration strategies are *(i)* the maximum likelihood approach which is applicable to models with closed form solutions, *(ii)* the sequential Monte-Carlo approximate Bayesian computation method,[10] and *(iii)* the method-of-moments (MM) which often employs variations of simulated minimum distance (SMD) methods. The MM approach has been criticised for requiring the selection of an arbitrary set of moments and for creating a rough objective surface with many local minima and for only generating point estimates as opposed to credibility intervals and posterior distributions.[11]

The task of calibrating intraday ABMs has been proven to be challenging for a number of reasons (Platt and Gebbie 2017). Firstly, model parameters often demonstrate degeneracies and have a haphazard effect on the objective function which leads to consecutive calibration experiments with significantly different parameters. Certain parameters tend to dominate all the others and it is often difficult to identify the role and dependencies between the various parameters. For the models we have considered the degeneracies are hypothesised to find much of their origin in the realistic matching processes, and in the particular model and its price setting mechanism (Platt and Gebbie 2018).

Also, any solution for the optimum is only ever an approximate solution due to the stochastic nature of ABMs. For this reason, the use of heuristic methods can be preferable for ABMs, as a heuristic method can be better at obtaining an approximation of the global optimum. Threshold accepting is a heuristic search method which can be used in conjunction with the Nelder-Mead simplex algorithm to efficiently calibrate ABMs. Gilli and Winker (2003) present a global optimisation heuristic using this Nelder-Mead with threshold accepting technique. One downside of a heuristic approach, however, is the long computation times required to effectively calibrate the model. To offset this, it is often necessary to resort to very coarse approximations of the objective function, which hinders the quality of the results.

2.3.2 Validation and expected stylised facts

We present a selection of stylised facts useful to constrain simple order book agent-based models. For more extensive lists of the empirical facts of financial markets we suggest readers refer to Bouchaud, Mézard, and Potters (2002), Chakraborti et al. (2011a), Cont (2001), Gould et al. (2013), Lillo and Farmer (2007), and Pagan (1996), and Schmitt and Westerhoff (2017). One key attribute of agent-based modelling is the idea that simple relational rules and interactions between agents can generate a variety of complex and dynamic in-sample distributional features — features that do not need to be directly (and distributionally) encoded into the model representation. Rather the distributional properties and various features emerge from the aggregate interaction between agents and their environment.

[10] SMC-ABC was applied to the Preis et al. (2006) model in Goosen and Gebbie (2020) with the surprising result that frequentist methods were sufficient.

[11] For an extensive comparison of the many different calibration techniques refer to Platt (2020).

Feature	Description
Return auto-correlations	The absence of auto-correlations in price fluctuations is sometimes cited as support for random walk models and the Efficient Market Hypothesis (EMH) because the presence of auto-correlations in the returns from financial markets would imply the existence of very simple strategies for making money using only knowledge of past share prices co-movements (Cont 2001). Given the great incentives people have to make money, it follows that such opportunities would quickly be snatched upon by traders up until the point where the opportunity no longer exists through the cancelling out of auto-correlations by the behaviour of the traders. Thus, the existence of this stylised fact, when considering returns over a sufficient period of time is intuitive. More importantly as related to the order book, returns on bid-ask prices exhibit a significant negative first lag auto-correlation (Cont 2001), suggesting fast mean reversion at the tick resolution. The absence of auto-correlations does not hold for time-scales greater than a week (Cont 2001).
Heavy/Fat tails	Mandelbrot (1963): The distribution of returns being leptokurtic, where a distribution is considered leptokurtic if its kurtosis is greater than that of the Normal distribution (kurtosis = 3).[12] Returns at the extreme percentiles or tails are found to exhibit a power-law distribution with tail index $2 < \alpha < 5$ — even after correcting returns for volatility clustering via GARCH-type models, the residual time series still exhibit fat tails. Suggested distributions include the generalized hyperbolic Student-t, the normal inverse Gaussian, the exponentially truncated stable, etc (Chakraborti et al. 2011a; Pagan 1996).
Aggregational Gaussianity	Kullmann et al. (1999): The distribution of returns as the time-scale at which they are calculated Δt increases, tends to normality. This feature becomes more apparent when calculating log-returns in trade time (Chakraborti et al. 2011a).
Volatility clustering	Biais, Hillion, and Spatt (1995) and Mandelbrot (1963): The slow decay in the auto-correlations of the absolute value of price fluctuations is indicative of long memory, and large changes in financial markets returns tend to be grouped together.
Gain/Loss asymmetry	Large drops in prices tend to be observed more frequently than large price increases (Kahneman and Tversky 1979).
Trade-sign auto-correlation	Order-flow in equity markets are persistent. That is, buy orders tend to be followed by more buy orders and sell orders tend to be followed by more sell orders. This feature is usually attributed to agent herding behaviour and trade splitting (Lillo and Farmer 2007).
Depth profiles	Mean relative depth profiles exhibit a hump shape in a wide range of markets. The maximal mean depth available is most often reported to occur at $b(t)$ and $a(t)$ (at a relative price of 0) (Biais, Hillion, and Spatt 1995; Bouchaud, Mézard, and Potters 2002; Gould et al. 2013).
Volume-volatility correlation	Trading volume is correlated with all measures of volatility. It has been shown that the variance of log-returns after N trades is proportional to N (Plerou et al. 2000).
Other desirable features	The *bimodality of a security's price distortion* (Schmitt and Westerhoff 2017) — a feature that is difficult to explain by the EMH[13], the *Epps effect* (Chang, Pienaar, and Gebbie 2020; Epps 1979), the *leverage effect* — where most measures of volatility are correlated with returns, the *power-law placement of limit orders around the best quotes*, and the *Power-law distribution of trade inter-arrival times, order volumes, and limit order average lifetime* are other desirable features we would hope to be able to recover or conform to with a reasonably representative model of financial market interactions.

TABLE 2.1: Summaries of some useful financial market stylised facts used to inform the model and parameter constraints.

2.4 Summary

The financial market represents a prime example of an observable complex adaptive system which cannot be easily explained by standard aggregate economic models or modelled by simple time series models. Many heterogeneous adaptive agents, such as traders, portfolio managers, market makers and regulatory authorities, interact non-linearly overtime with each other and the electronic exchange allowing for the emergence of complex behaviours beyond that expected based on intrinsic agent characteristics. For this reason studying market microstructure and the task of building an abstract representation of a financial market is challlenging. We start to address these challenges by first replicating the financial market from the most granular level and then apply our modeling abstractions.

3 Literature review

This chapter provides a background and context to the relevant literature that relates to the investigation of LOB simulation techniques and their extensions. We also present key insights into the engineering abstractions of financial exchanges to better inform necessary design principles and considerations.

3.1 On the design of matching engine software systems

A complete study of the market microstructure[1] of the Johannesburg Stock Exchange (JSE) is not possible without access to their matching engine[2]. Studying market microstructure is challenging due to the various changes in the market, regulation and technology. However, most of the current literature focuses on analyzing existing exchanges and the building of agent based models. The importance of order matching engines in the trading infrastructure makes these systems of interest not only to computer scientists but also to computational finance and risk management, while the non-linear impact of event-driven processes relating to order matching may provided an impenetrable calibration boundary for agent-based models attempting to empirically relate temporal dynamics with specific agent behaviours (Chang, Pienaar, and Gebbie 2020; Platt and Gebbie 2018; Goosen and Gebbie 2020).

A trade matching engine is the core software and hardware component of an electronic exchange. It matches up bids and offers to complete trades. A matching engine is the key piece of technology that intermediates agents submitting orders with the aggregation of orders and then the emergence of prices, correlations and order book dynamics. Electronic order matching was introduced in the early 1980s in the United States to supplement open outcry trading[3]. Before this, stocks where traded on exchange floors and transaction costs were high. Failures in these systems increased as the frequency and volume on the electronic networks increased.

Modern matching engines are fully automated and use one or several algorithms to allocate trades among competing bids and offers at the same price. They typically support different order types and have unique APIs or use standard ones[4]. As it pertains to trading, latency[5] directly influences the amount of time it takes for a trader to interact with the market.

Traditionally, to achieve low latency, high-frequency trading has required powerful server hardware appropriately networked in a data center, scaled to accommodate worst-case network traffic scenarios on the busiest trading days. These trading systems must be resilient in the face of network or power failures, requiring expensive redundant hardware as well as offsite data retention (Addison et al. 2019). For example, firms would use co-location, fibre-optic network cables, optimized hardware architectures

[1] Market microstructure refers to a branch of finance concerned with the details of how exchange occurs in markets. While much of economics abstracts from the mechanics of trading, microstructure literature analyzes how specific trading mechanisms affect the price formation process.

[2] A matching engine is a component of an exchange that matches buy and sell orders according to the rules of the exchange.

[3] A method of communication between professionals on a stock exchange or futures exchange typically on a trading floor

[4] Vendors include Connamara Systems, Cinnober (acquired by Nasdaq), **Aquis Technologies (A2X)**, SIA S.p.A., Nasdaq, Match-Trade, **MillenniumIT (JSE)**, GATElab Ltd (acquired by London Stock Exchange), Eurex, LIST, Stellar Trading Systems, Quodd, Baymarkets, Market Grid, ARQA Technologies, Kappsoft, Thesys Technologies

[5] Latency refers to the ability of a system to handle data messages with minimal delay.

and other technology to get as close to zero latency as possible (Sing 2017). On the other hand, cloud-based software solutions benefit by being resilient and offering cost-effective, easy scaling — an advantage that is not offered by traditional trading systems. However, the biggest challenge for latency in a cloud-based environment, and one of the greatest barriers to building high-frequency trading systems in this environment, is the fact that hardware is not co-located within a data center (Addison et al. 2019). That said, a combination of the low latency of traditional matching engines and the resiliency, scalability and availability of cloud-based environments is something that is yet to come about[6].

A low latency high throughput[7] matching engine does not exist for academics to further their understanding in this field (Sing and Gebbie 2017). CoinTossX provides an environment for the application of agent-based modelling experiments which would otherwise be prohibitive to undertake in real financial markets due to cost, complexity and other factors. It was for this reason that CoinTossX was developed and why its applications are studied further here. Here we hope to provide an easy-to-use, openly available, realistic, real-time, simulated trading system that is straight-forward to set-up on multiple platforms. Hence, this paper benefits institutions, academics and others seeking to take advantage of an open-source, resilient, scalable matching engine simulator that may avoid a variety of conflicts of interests related to research or insights derived from commercial alternatives.

3.1.1 Market simulation for testing

Simulating the entire financial market ecosystem for trade strategy testing and risk management is appealing because of the mechanistic complexity of the market structure and costs and the nonlinear feed-backs and interactions that multiple interacting agents bring to the market ecosystem. This type of simulation can be both financially costly, as well as computational expensive exercise; yet it appears tractable, and subsets of the ecosystem are used for system verification *e.g.* in vendor provided test market venues.

The rapid evolution of software and hardware and the increasing need to reduce transaction costs, system failures and transaction errors have led to important changes in market structure and market architecture. These changes have supported the rise of electronic, algorithmic, and high frequency trading. The specifics are unique to each and every market and regulatory environment. For example, in the South African context the JSE uses the MillenniumIT trading systems following the approach of the London Stock Exchange with the BDA broker-dealer clearing system (Johannesburg-Stock-Exchange 2020a) with rules and regulations set by the Financial Markets Act (2012), the JSE rules and directives, and the Financial Intelligence Centre (FIC) Act (2001). However, CoinTossX is fully configurable. Although implemented and tested here using the published and publicly available JSE market rules and test-cases, it can be configured for a rich variety of market structures.

Many researchers developing trading strategies do not have access to vendor provided testing environments, while those that do, often do not want to expose their strategies to competitors in shared testing environments. However, sufficiently realistic multi-agent simulation environments remain illusive, particularly for low liquidity, and collective behaviour based risk-event scenarios (Jericevich, Chang, and Gebbie 2020) because a key component remains the realism of the underlying trading agents and their interactions within test markets.

Most agent-based artificial exchanges and models are over simplified to the extent of not being relevant for realistic science and risk management. Some examples include using a global calendar time to sequentially order trading events, or providing intentional (and sometimes unintentional) market clearing events that synchronise price discovery and information flow with agent interactions in terms

[6]There is however a strong argument to place the Order Management System (OMS), that decides what to trade — the selection of parent orders and execution strategies, in the Cloud; while retraining the Execution Management System (EMS), that implements child orders, in close proximity to the matching engine. Here we are moving the matching engine into the Cloud.

[7]High throughput refers to the ability of a system to process a very high volume of data messages.

of a unique global time — the loss of high-concurrency and cohesion in favour of tight-coupling for computational convenience.

There are many examples, in the context of South African markets. Nair (2015) prototyped a simple matching engine using a single stock and adopting the standard order types and mechanisms associated with a continuous double auction market as specified in the JSE. Although works like this seem to provide a simplified but realistic framework demonstrating the over-all principles of coupling a matching engine with agent-based modeling of a stock market; such frameworks are unlikely to recover realistic dynamics or lead to useful insights when interrogated at the level of asynchronous but high concurrency order-book events and dynamics. High-concurrency and a careful management of the concept of time (Chang, Pienaar, and Gebbie 2020) seem a prudent requirement related to both the stylised facts of the market, as argued for in many South African market examples (Jericevich, Chang, and Gebbie 2020), but more importantly, for realistic strategy testing and risk management.

Here like-for-like message delays, order-book rules, and both asynchronous order matching as well as reactive, asynchronous, and high-concurrency agent (for the rules and behaviours) and actor (for the computational representations and causation models) generation can affect model outcomes and estimation. It is the asynchrony and order-splitting at the agent level that can dominate the emergence of auto-correlations and cross-correlations in various traded assets because there is no single calendar time related mechanism that generates equilibrium prices, can synchronise information and order-flow, or can co-ordinate machine time events, with sequential calendar time (Chang, Pienaar, and Gebbie 2020).

3.1.2 Realistically simulating high-concurrency

The LMAX Exchange (London Multi Asset Exchange) (Thompson et al. 2011) is an FX exchange with an ultra low latency matching engine. Thompson et al. (2011) developed the Disruptor ring buffer for inter-thread concurrency as an alternative to storing events in queues or linked lists. This was in response to the problem that linked-lists could grow and increase the garbage in the system — causing significant costs to latency and jitter[8]. At the heart of the disruptor mechanism sits a pre-allocated bounded data structure in the form of a ring-buffer. The Disruptor preallocates memory in a cache-friendly manner which is shared with the consumer. The system was also developed on the JVM and can process up to 25 million messages per second on a single thread with latencies lower than 50 nanoseconds (Thompson et al. 2011).

Recently, Addison et al. (2019) implement a simple foreign exchange (FX) trading system and deploy it to cloud environments from multiple cloud providers (Amazon Web Service, Microsoft Azure and Oracle Cloud Infrastructure), recording network latency and overall system latency in order to assess the capability of public cloud infrastructure in performing low-latency trade execution under various configurations and scenarios. They conclude that sufficiently low latency and controlled jitter can be achieved in a public cloud environment to support security trading in the public cloud (Addison et al. 2019). More specifically, they demonstrate the ability to achieve sub-500 microsecond roundtrip latency — therefore concluding that it is currently feasible to build a production low-latency, high-frequency trading system in the cloud.

For research in finance, economics and computer science, the importance of having a realistic, flexible, high performance matching engine deployable to different environments can be found in a wide range of tradings projects. In particular, given the whole new range of realism that such a software provides, agent-based computational finance and financial models in general are some of those that may show the greatest potential in terms of the insights to complex systems that can be gained from their application. This is not to mention the additional class/layer of causation that is introduced in the modelling framework by having the complex set rules of the environment/system/architecture be separate/independent from the modelling and decision making processes (at the agent level). In

[8] Jitter is the deviation from true periodicity of a presumably periodic signal, often in relation to a reference clock signal. This variation in the time between data packets arriving is caused by network congestion.

this way more emphasis can be placed on top-down actions and states — a potential step towards hierarchical causality (Wilcox and Gebbie 2014).

On this note, CoinTossX is a simulation environment and so the task of producing realistic simulated market dynamics, comparable to those observed in empirical investigations, is left to the user(s). For this purpose the two popular methods usually adopted are mutually exciting Hawkes processes (see Sections 3.3, 4.5 and Chapter 5) and agent-based models (ABMs) (see Section 3.4 and Chapter 6) (Lussange et al. 2018).

3.2 Market microstructure and modelling the LOB

The markets dealt with here are electronic order books of lit equity markets with no official market maker, in which orders are submitted in a continuous double auction and executions follow price/time priority. This type of exchange has now been adopted nearly all over the world, but this was not obvious when computerization was incomplete. Different market mechanisms have been widely studied in the market microstructure literature (see, *e.g.* Hasbrouck (2007), O'Hara (1997), and Garman (1976)).

The information content and dynamics of the limit order book (LOB) in order-driven markets have been studied extensively over the years with various stochastic, agent-based and Markov models being the most common representations (Bonabeau 2002; Chakraborti et al. 2011b; Farmer, Patelli, and Zovko 2005; Zheng, Roueff, and Abergel 2014). These studies point to a number of important challenges associated with modelling an electronic limit order book. The first concerns the assumptions made — in particular, whether order-flow exists due to the actions of "perfectly rational" (scrutinised for their inconsistency with direct observation of individual traders), or "zero-intelligence" traders or agents. The latter approach assumes order-flow is a result of aggregate behaviour that can be specified by a stochastic process. As a starting point, the "zero-intelligence" approach shows appeal for leading to easily quantifiable models that can yield falsifiable predictions (Gould et al. 2013). A second challenge regards capturing key features relating to the feedbacks and nonlinear couplings that naturally occur in a market ecosystem mediated by an electronic limit order book.

Traders' actions depend on the state of the LOB and the state of the LOB also clearly depends on traders' actions. The estimation and identification of statistical properties with LOB data remains problematic — several properties are observed to have power-law distributions *e.g.* order volumes, relative limit prices or the LOB depth profile, but there is no clear consensus on the best fitting procedure to adopt, nor the generation processes involved. Related to this is the stylised-fact centric validation of models with high-frequency data that ignore the problems of: *(i.)* degenerate model parameters, and *(ii.)* the inherently asynchronous and event driven nature of market clearing when modelling market in continuous trading (Platt and Gebbie 2017). Other challenges include: the complexity of a LOB's state-space, hidden liquidity, bilateral trade agreements and opening/closing auctions (Gould et al. 2013).

3.3 Simulating high-frequency trading using Hawkes processes

Recently, Zheng, Roueff, and Abergel (2014) used a multivariate marked Hawkes process for the joint spread-price dynamics. This is made possible by incorporating a constraint on intensities that preserves the natural ordering of best bid and ask prices. Four event types are considered to model the best bid or ask moving up or down by one tick. The constraint preventing the downward (upward) movement of the best ask (bid) comes into effect when the spread is one tick.

Bacry, Mastromatteo, and Muzy (2015) provide an extensive list of applications for point processes in finance which gives valuable insights into the price, spread and top-of-book dynamics. The key problem with an order-flow Hawkes model in a matching engine environment is that the arrival rates and the types of events do not depend on the state of the limit order book — specifically relating new orders to the existing orders in order book and the spread. This is part of the motivation for the use

of a minimally intelligent simulation model to better understand how Hawkes process modelled events can be distorted by realistic matching rules.

3.4 Agent-based artificial exchanges

ABMs successfully link the micro-level rules of investor interactions with the macro-behaviour of asset prices in real markets. However, not all complex behaviour in markets can be replicated using this bottom-up approach. There appears to be a hierarchy in the levels of abstractions which lead to different nuances in overall behaviour (Platt and Gebbie 2017; Wilcox and Gebbie 2014) when observed from different averaging scales and from different market participants. For example, the concerns of an equity risk trader are not the same as those of a commodity trader, or a mutual fund manager or a fixed income manager, or that of a central bank market regulator and economic policy makers. Nonetheless, ABM's are interesting as a tool for replicating complex behaviour.[9]

One of the first attempts at modelling the LOB in financial markets came from Stigler (1964). The approach taken at the time was to constrain prices within 10 ticks and randomly draw bid or ask orders (with equal probability) with prices uniformly distributed on the price grid. Each time an order crosses the opposite best quote, it becomes a market order. All orders are of size one. Orders not executed $N = 25$ time steps after their submission are cancelled. Thus, N is the maximum number of orders available in the order book.[10]

A "zero intelligence" model of the LOB was implemented by Farmer, Patelli, and Zovko (2005) showing that constraints imposed by market institutions can often dominate strategic agent behaviour. They test a simple model with minimally intelligent agents who place orders to trade at random — essentially dropping agent rationality altogether — which is demonstrated to produce good quantitative predictions. More specifically, two types of agents place order randomly in both price (limit orders only) and time (both market and limit orders). Patient agents submit limit orders that arrive according to the same Poisson rate and where the price is uniformly distributed in $-\infty < p_t < a(t)$ for buy orders and $b(t) < p_t < \infty$ for sell orders (where p_t, $a(t)$, and $b(t)$ are the price, best ask, and best bid respectively on the log-scale). Impatient agents submit market orders according to a constant Poisson rate. The sizes of market and limit orders are kept constant. Rates of buying and selling are kept equal. Based on this, order arrivals will alter the best and in turn change the boundary conditions of the limit order price distributions. Despite its simplicity, the model predicts the spread and average price diffusion rate well by dropping strategic choice and only considering the constraints imposed by the continuous double auction. The importance of these findings lies in the simple laws relating prices to order flow.

In reality, agents behave strategically, but these findings suggest that there are often circumstances where agent behaviour is dominated by considerations other than strategy. It then seems reasonable to approach agent-based modelling by first gaining a good understanding of market institutions using minimally intelligent agents and then work upwards towards more strategic agents. The basic approach taken by Farmer, Patelli, and Zovko (2005) can be considered similar to the one that will be taken here, with the key difference being that we use a multivariate Hawkes process to generate order types and their timings.

At the present, there are several agent-based artificial exchanges, with varying functionalities and architectures, addressing different problems (Brandouy, Mathieu, and Veryzhenko 2013; Kumar et al. 2013; LeBaron 2002; Raman, Jochen L. Leidner, et al. 2019). Most of the present artificial market platforms suffer from a lack of flexibility and must be viewed as software frameworks and tool libraries rather than Advanced Programming Interfaces (API's). This is because they are mainly implemented

[9] For extensive reviews of heterogeneous agent models and computational agent models refer to Bonabeau (2002), Chakraborti et al. (2011b), Chiarella, Dieci, and He (2009), Dieci and He (2018), Hommes (2006), LeBaron (2000), LeBaron (2006), Wang et al. (2018), and Iori and Porter (2012).

[10] Other early models of the continuous double auction include Garman (1976), Gode and Sunder (1993), Gould et al. (2013), and Mike and Farmer (2008).

for solving a specific problem confined to a particular choice of programming tools, and (most of the time) cannot easily be used to explore a wide range of financial issues that more closely mimic how software systems are designed and used for real trading and investment management (Brandouy, Mathieu, and Veryzhenko 2013). Most similar to our implementation, Brandouy, Mathieu, and Veryzhenko (2013) present the ArTificial Open Market Java API (ATOM) through which one can build a variety of experiments on order-driven markets. They define software engineering abstractions that provide a generic framework for stock market simulations by outlining key principles governing the development of Agent-Based financial market APIs as well as key issues such as local interaction, distributed knowledge and resources, heterogeneous environments, agents' autonomy, artificial intelligence, speech acts, discrete or continuous scheduling and simulation. Brandouy, Mathieu, and Veryzhenko (2013) demonstrate that the choices made for agent-based modelling in this context deeply impact the resulting market dynamics. At the present moment, it is realized based on the architecture close to the Euronext-NYSE Stock Exchange where orders sent by virtual traders are matched in a continuous double auction. Similar to CoinTossX, it allows for "humans in the loop" by mixing human beings with artificial traders. Key features of ATOM include high-throughput and low latency; limit, market, stop and iceberg orders; different adaptive/learning agent types with their own behaviour and intelligence; multiple assets; human interaction/trading; a "replay engine" for re-executing whole trading log-files and agent-server communication over a network. Additionally, the scheduling system for order submissions is parametric in that the user can choose to simulate sequentially or in parallel (where agent decisions can be made simultaneously). In the sequential approach, agents evaluate their positions every "round." In the parallel aproach, agents evaluate their positions in real-time. ATOM differs from CoinTossX in that the agents or scheduling system are highly coupled to the matching system and does not implement different trading sessions. Additionally, only the clients can be separated from the rest of the system and be connected by a network. As designed, ATOM is essentially more of a financial market agent-based modelling API/simulation environment and less of an exchange matching engine.

The difference here is that the implementation in this paper focuses on building agent-based models around well defined market mechanics. We first focus on defining the market rules and mechanics, and then allowing interaction via the rules and market mechanics. We believe that our implementation of separating the Model Management System (MMS) from the order matching system, the Matching Engine (ME), offers the desired flexibility for exploring different issues in financial markets without constraining the nature of the agent-based model (see Figure 6.1). This is because the order matching is then entirely decoupled from what agents do, and agent interactions do not take place in some globally synchronised calendar time.[11]

3.4.1 Towards actor/event-based reactive systems

The foundation of actor-based reactive systems is a set of individual components called actors or processes. These run autonomously in parallel and explicitly interact according to their local behaviour and reaction rules. The main fundamental differences between ABMs and reactive systems are (Crafa 2021): *(i)* a different management of time, and *(ii)* a fully decentralised decision control-logic. Both have significant effects on simulation dynamics. An agent-based model is typically iterated over stochastic or deterministic calendar time steps with aspects of the agents updated at each time step, whereas actors autonomously run in parallel and occasionally interact by exchanging messages. The sending of a message can be thought of as a signal that an event has occurred so that the receiver can react by modifying its behaviour and responding with another message (Hewitt, Bishop, and Steiger 1973; Hewitt 2011). Actors communicate asynchronously and may be "blocked" from further action until a reply or triggering event occurs. This asynchronicity is particularly well suited for financial markets and it can have significant effects on the model output. For agent-based models we argue that this may bring about a greater degree of realism, heterogeneity and computational efficiency. It also has the advantage of allowing modellers to simplify the models that specify how agents interact,

[11] A discussion on the nature of time in the trading environment can be found in Chang, Pienaar, and Gebbie (2020).

i.e., allow for models with fewer parameters that are still able to achieve high levels of realism. This may also be an important step towards realising hierarchical causality within simulations of financial market systems (Wilcox and Gebbie 2014) so that both top-down and bottom-up sources of causality can be included within the agent-based modelling design.

3.4.2 Intraday agent-based modelling

While building our model in Chapter 6, our thinking was informed by a number of different traditional intraday ABM approaches. We had to adapt them for an asynchronous continuous double auction implementation because of our departure from sequential market clearing.

In the Leal et al. (2016) model, low frequency agents adopt trading rules based on chronological or calendar time, and can switch between fundamentalist and chartist strategies. High Frequency (HF) trader activation is event-driven, in the sense of a stochastic time, and depends on price fluctuations. These traders use directional strategies to exploit market information produced by Low-Frequency (LF) traders. The Leal et al. (2016) model is our reference model and provides the basis for the ABM formulation in this paper — not only because it is a simple model with relatively realistic properties, but because we have a well understood model calibration that is useful for our use case (Platt and Gebbie 2017). However, this model presents a number of challenges for integration with a matching engine. It uses sequential market clearing and thus it is not an asynchronous market. Second, the limit price placements follows more closely to a random walk setup. On the other hand, this is a useful specification because the model does not require a single market maker providing liquidity, rather the numerous high-frequency agents act as a proxy to achieve a similar effect. A general problem with market making that we carefully avoid is the need to repeatedly optimise and solve control problems to find optimal order placements; solving these type of problems generally only give placement for one level of the book.

With the same model, Raman, Jochen L. Leidner, et al. (2019) develop an artificial exchange for data generation at scale. This heterogeneous model setup takes a traditional agent-based approach by specifying the same three agent types and parameters as provided by Leal et al. (2016). However, as an alternative to this, their software provides a library of models addressing specific use cases to produce realistic transaction price dynamics and trading scenarios. The four models in this library are: a heterogeneous model (the Leal et al. (2016) model), a semi synthetic model, an asset interaction model and a bond pricing model. In these cases the synthetic market matches orders posted, publishing the bid, ask, execution prices and the volumes at all levels of the Limit-Order Book (LOB) at consecutive (but synchronous) time steps. Agents in the semi-synthetic model can either be "real" which is the case where order time, side and quantity are the same as that of historical data but the price is relative to prevailing market prices, or "synthetic" where the orders are completely defined by stochastic decision rules. The asset interaction model considers trading in multiple assets to induce correlated price dynamics where order prices follow a random walk and the sides and quantities are determined by a weighted combination of expected returns. Our approach is similar to that of the heterogeneous model but with asynchronous order submission times in a continuous double auction. In constructing an asynchronous continuous double auction ABM we draw from a number of noteworthy sources.

Raman and Jochen L Leidner (2019) apply deep learning methods to determine the directional bet. However, their limit order placement remains similar to a random walk. Their model allows for volume sizes to also partially depend on the strength of the directional signal predicted by the deep learning method. In this case the issue of implementation is the lack of a continuous double auction. However, some care is prudent when using machine learning methods in the presence of strategic agents as feature selection can induce feed-backs that change the target function(s).

Very similar and subsequent to the work done in later chapters, Balch, Veloso, and Mandic (2021) demonstrate an agent-based model for trading in a continuous double auction for real world application with a strong focus on opponent modelling. They show how *(i)* opponent modelling techniques can be applied to classify trading agent archetypes and how *(ii)* behavioural cloning can be used to imitate

these agents in a simulated setting. They compare a number of techniques for both tasks and evaluate their applicability and use in real-world scenarios. They consider the most common four agent types: Background (noise) trader agents, Market maker agents (liquidity providers), market taker agents (liquidity takers) and directional trader agents (fundamentalists and chartists).

Preis et al. (2006) provide a limit order placement strategy that retains some of the ideas from Farmer, Patelli, and Zovko (2005) and also encapsulates the feedback dynamics from the market into the agents decisions. The calibration of this model has been explored in the literature (Platt and Gebbie 2018). We draw inspiration from their ideas to the limit order placement for our HF agents. In their model, liquidity-provider-agents submit limit orders, and liquidity-taker-agents submit market orders only, but to the same LOB through a series of Monte-Carlo steps. During each Monte-Carlo step liquidity providers first submit limit orders based on the current state of the LOB, after which liquidity takers submit market orders. During each Monte-Carlo step liquidity-providers may also cancel previously placed orders based on uniform sampling and a probability parameter. Limit prices are set to be a certain amount of ticks away from the best on the contra side, the size of the tick is dependent on state variables and comes from a random variable. This point ensures that our HF agents will place LO's at different levels in the LOB while incorporating feedback dynamics. The additional modification required is that we need to have different volumes sizes.

Mandeş (2015) is one of the few ABM setups that computes the resulting market impact arising from trading and order-book activity. Their setup requires some optimisation for decision making but is different from traditional optimal control and focuses on the appropriate market behaviour based on states of the LOB. Their work highlights the importance of agent decisions needing to depend on the state of the LOB to be realistic. In summary: The size of market orders should have some dependency on the spread, order-imbalance and depth of the LOB, and an overall supply-demand imbalance drives traders to price their orders more aggressively when their side of the book is crowded in order to increase their order execution probability (competition effect). Conversely, traders become less aggressive when the opposite side is thicker, forecasting a favourable short-term order flow (strategic effect).

The question that remains is how do we determine volume size? Mandeş (2015) uses an independent log-normal distribution for the volume size. If we do something similar then we need to provide additional rules or constraints to ensure liquidity takers can remove sufficient liquidity because the current Leal et al. (2016) model will not remove liquidity fast enough to maintain stability. Here we strive to avoid additional trading rules and constraints to force empirical realism, *e.g.*, by forcing HF agents to have entry thresholds as an additional parameter to be tuned to force empirical realism.

3.4.3 Market making

One key characteristic of the simulation environment that makes decision rules like order price setting challenging is that the market does not sequentially clear and agent decisions are made asynchronously. This, when coupled with the fact that agent decisions depend on the LOB and the LOB depends on agent decisions (there is a feedback loop) means that we require a mechanism to maintain market liquidity whilst at the same time maintaining a realistic spread, imbalance, etc. In other words, we need a market maker.

Kumar et al. (2013) implement an agent-based artificial exchange using the extended Glosten and Milgrom (1985) model of Das (2005). This implementation cannot be considered truly reactive and bundles order matching with the model management in a sequential system thereby exhibiting high coupling, low concurrency but high cohesion. The Glosten and Milgrom (1985) model is a widely used and well understood market making model. However, we require a distribution for the true value of the stock while having a proportion of informed traders and a proportion of uninformed traders. The idea is that the market maker wants to know how to place the bids and asks based on the order flow from the traders. The model can then only work for toy cases where one can maintain the probability

density estimate over the true value of the stock, thus allowing expectation to be taken. This is not possible in our simulation setting.

Das (2005) extends the Glosten-Milgrom model by using an online algorithm that does not require tracking the true value of the stock. Rather, it applies Bayesian non-parametric methods to place Limit Orders (LOs) appropriately into the LOB. There is still a problem with the Glosten-Milgrom model, because it is a model with only one market maker that only provides liquidity at one level, and there is no method in determining the volume it should place. This needs to be written in by hand.

Chakraborty and Kearns (2011) provide an insightful derivation on the profitability of market making. This is central to the decision making made by any market maker. The setup is simple: at each time point the market maker places bids and asks around the mid-price up to a depth of $n^b(p,t)$ for bids and $n^a(p,t)$ for asks at each time point t. The work provides an intuition on how market making can be profitable and thus self-sustaining. They show that when the local price movements is larger than the volatility from the change in opening and closing prices then market making can be profitable, *i.e.* market making is profitable when there is a large amount of local price movement, but only a small net change in the price. They are quite clear in articulating the important difference between market making and statistical arbitrage. Market making places no directional bets and tries to minimises directional risk. This is a fundamental point that should not be lost when building financial market agent-based models.

Avellaneda and Stoikov (2008) provide a stochastic control setup for market making. They make strong assumptions about the mid-price dynamics and utility function. The problem is their model only places orders with a volume size of one and it will only place limit prices on one level. This later example provides a natural conceptual split between the building of models for a single market making agent when designing trading algorithms to interact with a given market and its existing participants, as opposed to providing a minimalist specification of a market maker like interaction that can provide reasonably stable and realistic collective market dynamics. Here we are concerned with the later.

3.5 Summary

This chapter has outlined the relevant literature informing the work done in the chapters that follow. Literature covering the design and architecture of financial market systems/software/infrastructure is scarce and mostly proprietary. In the chapter that follows we provide our own design abstractions.

On the topic of modeling the continuous double auction, we draw from some of the many notable publications that address agent behaviour and rationality, market dynamics, price formation, microstructure, *etc.* — including Leal et al. (2016), Mandeş (2015), Farmer, Patelli, and Zovko (2005), Raman, Jochen L. Leidner, et al. (2019), and Zheng, Roueff, and Abergel (2014). These findings provide a basis for testing/demonstrating the implemention of simple asynchronous models with a matching engine.

Of particular interest for future research is the class of models described by Lussange et al. (2018) who outline a computational research study of collective economic behavior via agent-based models, where each agent would be endowed with specific cognitive and behavioral biases known to the field of neuroeconomics, and at the same time autonomously implement rational quantitative financial strategies updated by machine learning (and reinforcement learning (RL)).

4 CoinTossX

We deploy and demonstrate the CoinTossX low-latency, high-throughput, open-source matching engine with orders sent using the Julia and Python languages. We show how this can be deployed for small-scale local desktop testing and discuss a larger scale, but local hosting, with multiple traded instruments managed concurrently and managed by multiple clients. We then demonstrate a cloud based deployment using Microsoft Azure, with large-scale industrial and simulation research use cases in mind. The system is exposed and interacted with via sockets using UDP[1] SBE[2] message protocols and can be monitored using a simple web browser interface using HTTP. We give examples showing how orders can be be sent to the system and market data feeds monitored using the Julia and Python languages. The system is developed in Java with orders submitted as binary encodings (SBE) via UDP protocols using the Aeron Media Driver as the low-latency, high throughput message transport. The system separates the order-generation and simulation environments *e.g.* agent-based model simulation, from the matching of orders, data-feeds and various modularised components of the order-book system. This ensures a more natural and realistic asynchronicity between events generating orders, and the events associated with order-book dynamics and market data-feeds. We promote the use of Julia as the preferred order submission and simulation environment.

4.1 Introduction

CoinTossX is a an open-source, high-frequency, low latency, high throughput matching engine for simulating the JSE (Sing and Gebbie 2017; Sing 2017; Jericevich, Sing, and Gebbie 2021) (or any market that has well established rules and test cases). The software was developed with Java and open-source libraries and is designed to maximize throughput, minimize latency, and accommodate rapid development of additional functionality. It can be configured for multiple clients, stocks and trading sessions (continuous trading, opening auction, closing auction, intraday auction and volatility auction). It may allow traders, organizations and academic institutions to test market structure, fragility and dynamics without the cost of live test trading. The software can provide a platform to study price formations in stock exchanges and the interplay between regulators, market structure and dynamics (Sing and Gebbie 2017). This work also addresses some aspects of the unavailability of data and direct data-feed access from industry, by providing a framework that can be compared to recorded transaction data arising from the actual market system interfaces.

The system requirements we implemented were obtained from JSE's publicly available technical documentation and test cases (Johannesburg-Stock-Exchange 2020b; Johannesburg-Stock-Exchange 2018; Johannesburg-Stock-Exchange 2019)[3].

The eight main components of the simulator are (Sing and Gebbie 2017):

1. *Stocks* are objects for which clients can send orders and limit order books can be constructed and kept track of.

[1] With the User Datagram Protocol, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths.

[2] The Simple Binary Encoding is an OSI layer 6 presentation for encoding and decoding binary application messages for low-latency financial applications.

[3] Technical documentation can be found at https://www.jse.co.za/services/technologies/equity-market-trading-and-information-technology-change

2. *Clients* are computer algorithms that send order events to the simulator. Clients will send order events to the trading gateway (refer to Section 4.3 and A.2.4 for more detail).
3. The *trading gateway* receives the client request, validates the request and then sends it to the matching component to be processed. It sends updates to the client to indicate the status of the event.
4. The *matching engine* processes the events from the Trading Gateway. It manages one or more limit order books. If there is an update to the LOB, it sends updates to the market data gateway.
5. The *market data gateway* receives updates from the matching engine component and sends market data updates to all connected clients.
6. The *website* receives updates from the market data gateway. It displays the LOBs for each security and allows the user to configure the stocks and clients.
7. The *web-event listener* was specifically incorporated for the purpose of reducing pauses to the website due to garbage collection. This single, dedicated writer principle would improve performance of the website as well (refer to the discussion on the website in Section 4.2 and A.2.5). This component also saves each event to the database.
8. The *database* stores the LOBs of each stock and is designed to have low memory overhead whilst being efficient in searching, updating and deleting orders (refer to the discussion on off-heap data structures in Section 4.2).

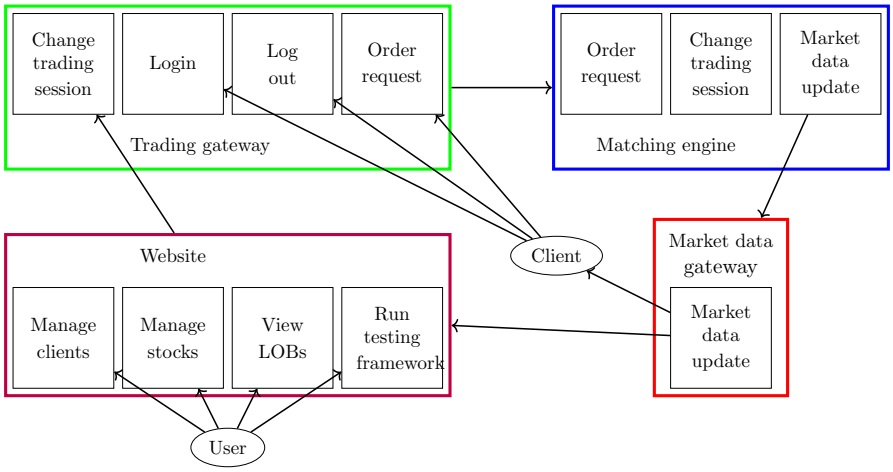


FIGURE 4.1: A High level diagram of the relationship between the components of CoinTossX in terms of end-user functionality (Sing and Gebbie 2017).

A website is required to monitor and configure the stocks and clients. The relationships between these components are also summarised in Figure 4.1. The client sends a login request message to the trading gateway which then validates the client. If the client is already logged in or the username or password is invalid it responds with a reject code message. The client can also send a log out request to the trading gateway. In this case the trading gateway removes the client from the list of connected clients and sends a response to the client to indicate if the log out was successful. The trading gateway logs out clients when it shuts down. When there is a change in the trading session, the website sends a message to the trading gateway which then sends the message to the matching engine. Clients can send an order request or order message to the trading gateway which then validates the message. If the message is valid, it then sends the message to get matched. Valid order types are market, limit, hidden, stop and stop limit. Each order has a time in force (TIF) value, which cannot be amended, that determines how long an order is active until it is executed, deleted or expired (which ever comes first). The engine matches the orders using the matching algorithms based on the trading session. Market data updates are sent to the market data gateway to indicate the changes in the LOB of each

stock. Additionally, the website and all clients receive market data updates and internal messages (to monitor the application) from the gateway. A user can create, update and delete clients and stocks. A user can view the limit order book for each stock as a bar chart of all active orders. Tables will show the details of these active orders. A user can start, stop and configure the parameters of the testing framework.

4.2 Architecture

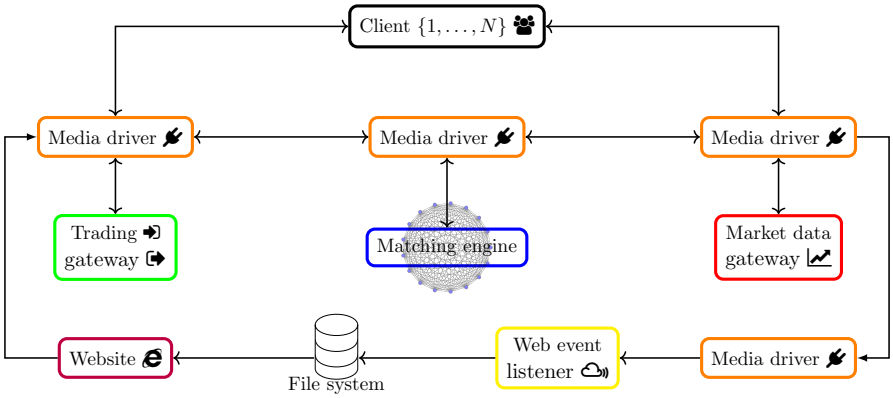


FIGURE 4.2: A High level architecture diagram visualising the technical relationship between the software components (Sing and Gebbie 2017).

This software is open source and was built to run on different operating systems as well as on a single server or on multiple servers. Implementation in Java means that CoinTossX can be deployed on different hardware configurations (due to the JVM). The goal was to build a matching engine that could achieve low-latencies using industry standard technology. This ensures that the software is portable and can be deployed confidently. The high level architecture of CoinTossX is summarised in Figure 4.2.

A single market event goes through the following process. A simple binary encoding (SBE) message[4] will be sent to the trading gateway. The trading gateway will forward the message to the matching engine which will then process it. It will send a message back to the trading gateway to indicate the status of the message. The trading gateway will forward the message back to the client. The matching engine will send an update to the market data gateway if there is a change in the limit order book. The market data gateway will forward the updates to all connected clients and will forward some of the updates to the web event listener. The web event listener saves each event to the file system. The website then reads and displays the data from the file system.

The communication between the website and the file system is done by transmitting data over a network using only User Datagram Protocol (UDP)[5] as opposed to Transmission Control Protocol (TCP). TCP is a heavyweight protocol because it requires three packets to set up a connection and requires a connection to be set up between two applications before data can be transmitted[6]. TCP is not suitable for low latency (especially for financial market use cases) due to the tradeoff between bandwidth usage after loss and responsiveness.

UDP, on the other hand, was chosen for being lightweight protocol as it does not check the connection or the order of the messages and does not require a connection to be setup before data is transmitted.

[4] JSE uses text messages between the clients and their matching engine which is inefficient since characters take up more memory and are slower to transmit across the network. From the comparisons made in Sing and Gebbie (2017), the SBE message protocol was found to be the fastest way to encode and decode messages; and had the greatest throughput.

[5] JSE uses TCP for their trading gateway and UDP for their market data gateway.

[6] Nonetheless, TCP can be reliable because if a message is not received, it will try multiple times to deliver the message. TCP will drop the connection if there are multiple timeouts.

Data is transmitted irrespective of whether the receiver is ready to receive the message or not. UDP may, however, be unreliable because the sender does not know if the message was delivered. The messages that are received (read as a byte stream) will always be in the order that it is sent. Modularisation is achieved by designing each of the above components independently of one other while allowing communication between them to occur only via exposed ports using these high speed SBE message protocols. This functionality was introduced by assigning each component an IP address and port on which to listen. These ports are specified in the "properties" files in the root project directory. Furthermore, each client/user may submit orders from a remote server. So if CoinTossX is deployed to the cloud, one should ensure that the virtual machine allows for these types of inbound port communications.

After an extensive comparison of message transport software, the Aeron media driver library was chosen for supporting the above protocols as well as for being an efficient and reliable UDP unicast, UDP multicast, and IPC message transport for communicating between all the components. Each component has its own media driver to shovel events to and from the component. Aeron is designed to be the highest throughput with the lowest latency possible of any messaging system (Real-Logic 2021) (see Figure 4.3). This software ensures the application is separated from the protocol driver logic via core-to-core communication over lock-free and wait-free data structures. Simply put, Aeron is the component of the exchane that allows for the transmit of messages (*e.g.* orders, market data, etc.) to and from the client or other exchange components.

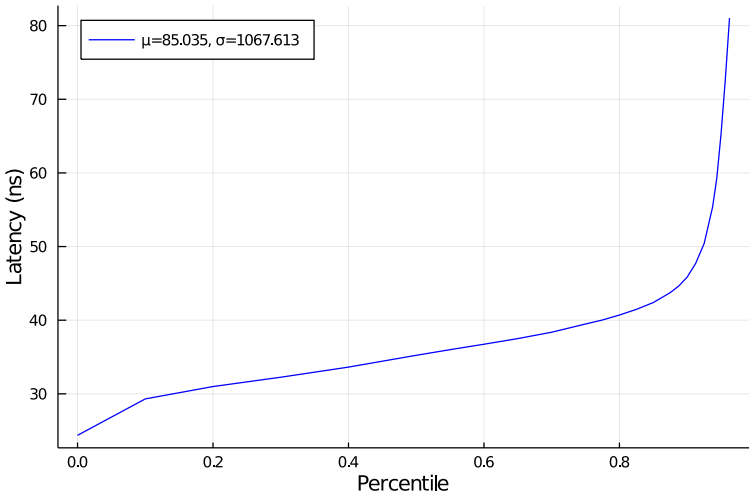


FIGURE 4.3: A High Dynamic Range (HDR) Histogram graph showing the latency percentiles of Aeron's PingPong test. This provides a lower-bound for the achievable latency of CoinTossX as well as for open-source message transport libraries in general (Real-Logic 2021).

Given this modular design, each component and client may be started and run on seperate, independent servers. As mentioned, components communicate with each other through the media driver via UDP SBE message packets. Therefore, it is essentially possible to have an industrial matching engine by providing each component with its own high performance server.

The matching engine is the component which relies heaviest on the latency and throughput of the system and is therefore designed with these features in mind whilst sticking to the JSE matching engine's architecture. The software was designed such that different matching logic algorithms are in separate Java classes. This allows the logic to be changed to test any variations of the matching logic. The matching engine adopts the price-time priority algorithm during the continuous trading session. That is, for multiple orders occurring at the same price, a market order will be matched with the order having the earliest submission time. The matching strategy used is configurable to any other strategy (*e.g.* pro-rata matching strategy).

Limit and market orders submitted during a call auction are not matched immediately. These orders are matched at the end of the call auction at a single price using a price discovery process. The Volume Maximizing Auction Algorithm finds the price that will match the most number of buy and sell orders. More specifically, during both the continuous trading and auction call sessions, the matching engine processes orders as follows. The filter and uncross algorithms run each time the Best Bid or Offer (BBO) changes or every 30 seconds. The filter algorithm uses the heuristic Hill Climber

search/optimization algorithm[7] to find the optimal volume of hidden limit orders that can be executed. The search will filter out hidden limit orders with MES constraints that are not eligible. After the filtering, specific rules are used to select the orders and price to executed in the crossed region.

As in the JSE, the matching engine uses native message protocols[8] to communicate with clients as opposed to FIX[9] (Financial Information eXchange), FAST[10] (FIX Adapted for STreaming), ITCH[11] or OUCH[12]. The matching engine component stores the active orders in memory since storing the active orders on disk would increase the I/O and reduce performance. The data structure for storing the LOB is designed to have a low memory overhead (lists, hash tables, trees, B-tree, B+tree) and be efficient in searching, updating and deleting orders (Sing and Gebbie 2017). More importantly, the low latency is achieved through the efficient use of the CPU cache. The simulator only uses main memory without the use of virtual memory[13]. The time taken for data to move from the CPU to main memory is reduced by using caches that contain copies of frequently used data from main memory.

The website was developed using Spring Boot and Apache Wicket and only has the permission to read from the file system. The original design had the event listener and website in the same Java process (Sing and Gebbie 2017). When the website was used or paused because of garbage collection, it affected the receiving and saving of events. Therefore this logic was split into a separate component — the web event listener. The listener could keep the received events in memory or save it to the file system. Saving the data in memory would be fast, but would require an unknown maximum memory setting. Therefore the data needs to be saved to the file system. Using the MapDB library[14], an off heap hashmap is used to save the events to memory mapped files. An off heap hashmap stores data outside the Java heap space and is not affected by garbage collection. Off heap data is suited for storing data larger than the current memory and allows sharing of data between JVMs. Memory mapped files allow Java programs to read and write files using only memory while the operating system reads and writes to the file system. This significantly improves performance. The entire file or a part of the file can be loaded into memory. The values in memory will still be written to the file system even if the JVM crashes. The web event listener has read and write permissions. It saves events to the file system but receives events faster than it can save it to file. This issue is solved with the Disruptor by having two threads: one to receive and store events, and the other to save events to the file system (Thompson et al. 2011).

4.3 Clients

Important to the design of this trading system are the clients who send orders to the trading gateway and receive updates from the website and market data gateway (see Section A.2.4 for more details on the submission of orders). Each client is assigned input and output URLs for both the Native Gateway and Market Data Gateway. These URLs specify the IP addresses (in this case localhost — the user's local machine) to/from which messages will be sent/received as well as the ports on which these components are listening. Therefore each client takes up a number of threads on the machine it

[7] The Hill Climber algorithm is an optimization technique that iteratively searches for the solution to a problem by changing one element in each iteration

[8] Message protocols are the methods by which the exchange communicates with market participants. Native protocols are custom built protocols.

[9] FIX is an open standard non-propriety protocol that is used by buy and sell firms, trading platforms and regulators to transmit trade data. The protocol allows organizations to easily communicate domestically and internationally with each other.

[10] Greater numbers of market participants and therefore volume increased the network latency and lead to market participants not receiving updates in an acceptable time. The FAST protocol reduces latency by encoding and compressing the data before transmission.

[11] ITCH is used to publish market data only.

[12] OUCH is used to place, amend and cancel orders.

[13] Virtual memory is space on the hard disk which is used by an operating system when it requires more memory. The time to fetch data from virtual memory is very slow and is therefore not used.

[14] MapDB is an open-source, embedded Java database engine and collection framework. It provides Maps, Sets, Lists, Queues, Bitmaps with range queries, expiration, compression, off-heap storage and streaming.

runs on — which can be the same or different from the machine running the other components of the matching engine. So, having clients send orders to the server remotely would free up hardware and improve performance on the matching engine server.

It is also important to note that each client here is a unique trader having its own unique ID, password, ports and login credentials. However, as was done in the testing framework, a single client may still be used to simulate multiple traders. This would be preferable in the case where the client(s) and other components run on the same machine, thereby limiting the computational resources required for simulations. In that case, the client is simply a tool/object for submitting orders to the gateway. In this way one can have a self-contained simulation framework be independent from the actual implementation or sending of orders by, for example, defining an ABM in Julia with each agent holding a reference to the client object that sends orders.

The number of client-stock pairs that can be supported in total is only limited to the performance capabilities of the server on which the matching engine is running (see Section 4.5 for hardware recommendations and performance results with differing number of client-stock pairs). The limits of the number of clients that can be supported by each stock, however, still needs to be explored and measured with respect to different hardware configurations.

4.4 Functionality

When the user starts CoinTossX, through the website, four main screens can be switched between and displayed. The stock screen shows: the stocks configured, the trading session that is active for each stock, and a button to view the limit order book of the stock. In the limit order book screen for each stock a bar chart will be generated to display the active orders. The market data is shown on the LOB pages for each stock as snapshots representing the current state of the limit order book. Tables show the details of the bids, offers, trades and all submitted orders. All data on these pages can be exported. The Hawkes configuration page (see Section 4.5) allows the user to change the values of the Hawkes input data before running the simulation. The simulation page allows the user to stop and start the warm-up process and the Hawkes simulation. It shows the status of each client and the active trading session. The clients screen shows the clients that are configured and allows the user to create, update and delete clients. Clients not configured will not be able to log in to the trading gateway. In order for a client to submit orders, a login request, with the relevant username and password, must first be sent to the trading gateway. Similarly, log-out request must be sent when the client logs out. Thereafter the client can submit orders by publishing order messages to the trading gateway via UDP ports. Traders cannot submit orders that are smaller than the LOB's tick size (controls the smallest order size). The allowable order types that are not conditioned on time are (Johannesburg-Stock-Exchange 2020b):

1. **Market orders (MO)** — contains the quantity of shares to trade, but not the price, and executes against each order in the LOB on the contra side until it is fully filled. Orders submitted during the auction call will remain in the LOB until uncrossing is done. All market orders that are not filled will expire. Market orders submitted when the contra side of the LOB is empty do not park in the LOB. Instead they are ignored.

2. **Limit order (LO)** — displays the quantity and price which may execute against a trade or expire based on it's time-in-force (TIF).

3. **Cancel order (CO)** — requires the corresponding order identification number and its limit price. Cancel orders specifying an ID that does not exist will be ignored by the matching engine. At the request of the client, CoinTossX can be configured to automatically cancel all open and parked orders submitted under an user ID (CompID — see Section A.2.4) whenever it disconnects from the server.

4. **Hidden limit orders (HL)** — allows traders to submit orders which are completely hidden from the market (price and volume). These have a lower priority than all visible active orders.

These orders can execute against other visible and hidden orders. Hidden limit orders that are submitted during an auction call are rejected. The quantity of a hidden order must meet the minimum reserve size (MRS)[15]. Each hidden order also has a minimum execution size (MES)[16]. After a trade executes against a hidden order, if the remaining quantity is less than the MES (¿ MRS) then the order will expire. If the remaining quantity is greater than the MRS and MES then the order will only expire when it executes or based on it's TIF (whichever comes first). Hidden orders may only be submitted during the continuous trading session.

5. **Stop market order and stop limit order (SO & SL)** — a stop order is a market order with a stop price. These orders do not enter the order book (remain unelected) until their stop price is reached. When the stop price is reached, the stop order becomes a market order. Similarly, a stop limit order is a limit order with a stop price. These orders do not enter the order book until their stop price is reached. When the stop price is reached, the stop order becomes a limit order. Stop and stop limit buy orders will be elected if the last traded price is equal to or greater than the stop price. Similarly, stop and stop limit sell orders will be elected if the last traded price is equal to or less than the stop price. An incoming stop or stop limit order may be immediately elected on receipt if the stop price has already been reached. These orders will also only be elected at the end of the execution of an order.

All of the above orders require additional meta-data fields such as a client-assigned ID and a trader identification number. Some of the above orders can also have a time-in-force (TIF) which cannot be amended after the order is placed. Valid combinations of order types and TIF are shown in Table 4.1. The following time in force options are available (Johannesburg-Stock-Exchange 2020b):

TIF	Market	Limit	Hidden limit	Stop	Stop limit
IOC					
FOK					
DAY					
GFA					
GFX					
OPG					
ATC					
GTC					
GTD					
GTT					
CPX					

Accepted Rejected

TABLE 4.1: Valid order types and time-in-force (TIF) combinations for the submission of new orders to the trading gateway.

1. **At the opening (OPG)** — only accepted during the opening auction and are used to direct orders only towards this session. OPG orders that are not filled during the uncrossing will expire at the end of the opening auction. If there is no opening auction scheduled, OPG orders will be rejected.

2. **Good for auction (GFA)** — orders of this type that are submitted will be parked[17] until the next auction and are injected at the start of the auction. GFA orders that are not filled during the uncrossing will be parked for the next auction and are only removed when they are filled or cancelled. If there is no auction session scheduled, GFA orders will be rejected.

3. **Good for intraday auction (GFX)** — These orders are parked until the next intraday auction and are injected at the start of the auction. As opposed to GFA orders, GFX orders that are not filled during uncrossing will expire at the end of the intraday auction. If there is no intraday auction scheduled, GFX orders will be rejected.

[15]The minimum order quantity for orders to qualify as hidden limit orders.
[16]The minimum quantity of the hidden limit order which is permitted to execute
[17]While orders are parked they cannot be executed.

4. **At the close (ATC)** — submitted orders of this type are parked until the next closing auction. They are injected at the start of the auction and if they are not filled, will expire at the end of the closing auction. If there is no closing auction scheduled, ATC orders will be rejected.
5. **Day (DAY)** — orders that expire at the end of the trading day. If an order does not specify a TIF, it will default to DAY.
6. **Immediate or cancel (IOC)** — can be partially or fully filled. An IOC that is only partially filled will cancel immediately after execution. IOC orders are rejected during auction calls.
7. **Fill or kill (FOK)** — this order type differs from IOC orders in that they are either fully filled or expired. Partially filled orders are not allowed.
8. **Good till cancel (GTC)** — can remain in the order book for a maximum of 90 calendar days or until the order is filled or canceled. If a GTC order is amended, the re-submission date is not updated and the expiry will still be calculated on the original order submission date irrespective of the change made.
9. **Good till date (GTD)** — as opposed to GTC orders, these orders remain in the order book until the order is filled, cancelled or a specified expiration date is reached (maximum active time is 90 days). GTD orders do not allow for the specification of an expiration time (only date).
10. **Good till time (GTT)** — remain in the order book until the specified expiry time is reached in the trading day. GTT orders that have an expiry time during an auction will not expire until uncrossing is finished.
11. **Closing price cross (CPX)** — these orders are parked until the start of the closing price cross session. Unexecuted CPX orders are expired at the end of the closing price cross session. Stop and stop limit orders are not allowed to have a TIF of type CPX.

Table 4.2 lists all compulsory and optional order attributes (depending on the order type) that the user can specify when sending orders to the matching engine. Note that some of these attributes only apply to certain order types.

Field	Required	Description
Security ID	✓	Unique identifier of the security.
Side	✓	Whether the order is to buy or sell.
Order type	✓	Limit, stop limit, stop, cancel, market or hidden.
Order quantity	✓	A whole number that is greater than zero and must be a multiple of the instrument's Lot size. This field is required for cancel orders.
Price	✓	This value should be greater than zero and a multiple of the instrument's tick size. This field is not required for market orders.
Capacity	✓	Denotes if the order is entered as an agency (on behalf of a client) or principal (own account)
Trader Mnemonic	✓	The group that the trader belongs to
Client ID	✓	Unique client identifier field
Client order ID	✓	User-assigned order ID
Time-in-force	✗	The duration the order is valid for (defaults to DAY).
Expiry date/time	✗	The time/date at which an order with GTT/GTD Time in Force should expire. Only required for orders with TIF = GTT/GTD.
Stop price	✗	The price at which the order may be elected. This field is required for stop orders.
Visible size	✗	Non-zero for hidden orders but zero for pegged hidden and pegged hidden limit orders.
MES	✗	The mandatory Minimum Execution Size for a hidden order. This field is required for hidden orders.
MRS	✗	The minimum order volume for orders to qualify as hidden limit orders. This field is required for hidden orders.

TABLE 4.2: A list of all compulsory and optional order attributes to be specified by the user when submitting an order to CoinTossX

Lastly, CoinTossX also allows for multiple trading session types which correspond to that of the JSE. The trading sessions below and the rules adopted by each are configurable in the `data/tradingSessionsCron.properties` file (see Section A.2.2 for more details on the configuration of trading sessions). The times shown are those of the JSE, provided for context (Johannesburg-Stock-Exchange 2020b). All trading session start and end times can be specified by the user through the cron expressions in the data directory. During the auction call sessions orders are not matched immediately, rather, they are matched at the end of the call auction at a single price using a volume maximizing price-discovery process. This process finds the price that will match the most number or buy and sell orders. Auction and continuous trading sessions also rely on a static and dynamic reference price[18]. The static price reference (SPR) is the price at which all orders in an auction are executed and is updated at the end of each auction session (Opening, Re-opening, Intraday, Volatility, Closing). The dynamic price reference (DPR), on the other hand, is updated continuously throughout the day and serves as a reference point for calculating the price of a share during sessions. In the continuous trading session the dynamic price reference at any one point is the last traded price. The static and dynamic price reference at the start of each day is the closing price of the previous day. The DPR and SPR are also important for the activation of volatility auctions. A volatility auction is triggered when the percentage change in a security's price from the DPR or DPR is greater than the circuit breaker tolerance[19] (Typically set to be 10%). In other words, during the day, if a stock moves beyond a certain predetermined range, it will automatically go into a volatility auction. All these auctions work in the same manner and are designed to enable price discovery.

1. **Start of trading** (*07:00 — 08:30*) — No orders can be submitted or executed executed during this session. Traders will be able to cancel, but not submit, orders during this session.

[18]The static/dynamic reference price is the last auction or automated trade price or the previous closing price, whichever is the most recent.

[19]The circuit breaker tolerance is the maximum allowed percentage change an equity's next price can move from the Static or Dynamic Reference Price. This prevents unnatural price movements in an instrument by triggering a volatility auction call session

2. **Opening auction call sessions** (*08:30 — 09:00*)
3. **Continuous trading session** (*09:00 — 16:50*) — The system will continuously match incoming orders against those in the order book according to the price-time priority execution rule.
4. **Volatility auction call session** (*triggered*) — This session will only trigger when a stock's circuit breaker tolerance level (10%) has been breached. Volatility auction call sessions last for a scheduled period of 5 minutes. The orders accumulated during this session will be executed at the uncrossing based on the volume maximizing algorithm.
5. **Intraday auction call session** (*12:00 — 12:15*)
6. **Closing auction call session** (*16:50 — 17:00*)
7. **Closing price publication** session (*17:00 — 17:05*) — No orders can be submitted or executed executed during this session. Traders will be able to cancel, but not submit, orders during this session.
8. **Closing price cross session** (*17:05 — 17:10*) — Trading will only take place at the closing price that was published during the closing price publication session.
9. **Post close session** (*17:05 — 18:15*) — Traders will be able to cancel, but not submit, orders during this session.
10. **Halt** (*manually envoked*) — A halt session may be activated by the user during which time no order equests may be executed. Traders will however be able to cancel orders.
11. **Halt and close** (*manually envoked*) — The behaviour is the same as the halt session except closing price calculations will be performed.
12. **Pause** (*manually envoked*) — No executions will take place during the pause session. Traders will be able to submit, amend or cancel orders during this session, however, market orders will expire at the end of the session.
13. **Re-opening auction call** (*manually envoked*) — The user may manually invoke the re-opening auction call session when resuming from a manual trading halt or a trading pause.
14. **Trade reporting** (*08:00 — 18:15*) — Trades and statistics, where applicable, will be published through the market data gateways.

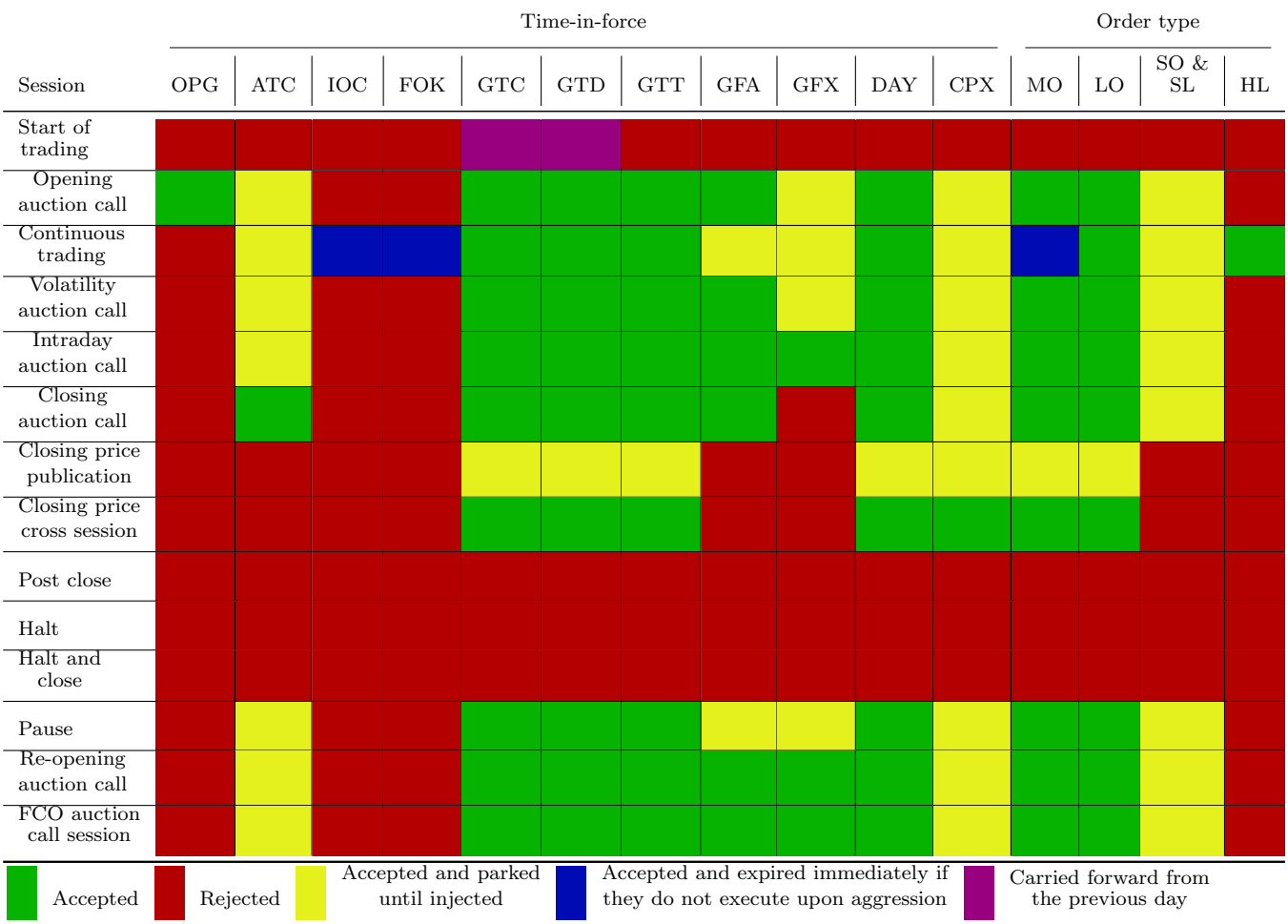


TABLE 4.3: Valid trading session and order type/TIF combinations for the submission of new orders to the trading gateway.

4.5 Testing framework

CoinTossX has been successfully deployed locally (see Section A.1 for more details on how to install and deploy CoinTossX) as well as to remote servers such as Microsoft Azure, CHPC and TW Kambule Mathematical Sciences Laboratories provided servers (using 4 re-purposed legacy TACC Ranger blades, see Table 4.5). Deployment to high-performance compute solutions such as UCT HPC was found to be infeasible for a number of reasons: First, facilities such as UCT HPC often perform computations by relying on an MPI approach using one of a variety of different job management systems where the submission of "jobs" to "worker nodes" is highly constrained by the preferences of systems administrators managing many different use cases; the worker nodes are not equivalent to virtual machines, rather they receive tasks from the "head node" to be executed in parallel. Simplistically this means that, for example, web interfaces will not be easily accessible from worker nodes due to the system and network architecture — but this more generally impacts any client worker interactions when agents interact via clients through some centralised architecture — here the matching engine; but this can be any interaction landscape. In general, such systems are poorly suited for real-time and reactive use cases. This is because, by design, many high-performance computing facilities, such as UCT HPC or CHPC, are not high-throughput facilities. They are typically not well suited for large scale high-concurrency, low latency market and agent-based simulation problems that necessarily need to be highly reactive[20] in nature. We believe that this is an important design perspective that is often

[20]Reactive is used here in the broad sense of system architectures that require the fast propagation of data changes and relationships within a system with many clients, but require high cohesion with low coupling.

not well considered when designing advanced market (or social science) simulators.

4.5.1 Unit testing

The functionality of the software was tested using the test cases made available online from the JSE. The requirements were taken from Johannesburg-Stock-Exchange (2020b). Testing of the trading sessions were restricted to the continuous and intraday auction trading sessions using only the DAY TIF. Unit tests were implemented to cover the testing of the functional requirements of the software while individual throughput and latency performance tests were implemented in conjunction with the Java Microbenchmark Harness[21] (JMH) to test methods whose performance were critical. These tests, covering the majority of the application, provide a safety net to allow changes to be made to the code without breaking existing functionality.

The outputs of these tests were not compared to the JSE or another exchange as the data is not available by the industry. The JSE's test environment is also closed and does not provide realistic order-book dynamics. This paper focuses mainly on the results of latency and throughput tests, however, the types and results of the extensive list of tests performed can be found by referring to Sing and Gebbie (2017).

4.5.2 Order-flow testing

Matching engine integrity was evaluated using unit tests and, instead of an agent-based approach, simulations and tests aim to understand throughput and latency were carried out with a 8-variate marked Hawkes process (Hawkes 1971; Yosihiko Ogata 1988). This provides a flexible framework to simulate a market data feed with varying throughput, with full control over the trade and quote conditional intensities. The software and Hawkes client processes were deployed on one server which affected the performance of all components. The mutually exciting processes correspond to 8 different order types that are considered in the testing framework. The testing framework only considers basic aggressive and passive market and limit orders as in Large (2007). Refer to Table 4.4 for the exact event-classification/types.

Type #	Event type	Ask or Bid	Immediate execution	Moves prices	Descritpion
1	Trade	Ask	✓ (MO)	✓	Market buy that moves the ask
2	Trade	Bid	✓ (MO)	✓	Market sell that moves the bid
3	Add	Bid	✗ (LO)	✓	Bid between quotes
4	Add	Ask	✗ (LO)	✓	Ask between quotes
5	Trade	Ask	✓ (MO)	✗	Market buy does not move ask
6	Trade	Bid	✓ (MO)	✗	Market sell does not move bid
7	Add	Bid	✗ (LO)	✗	Bid at or below best bid
8	Add	Ask	✗ (LO)	✗	Ask at or above best ask

TABLE 4.4: Order event types used for the 8-variate Hawkes process in the testing framework simulation following the approach of Large (2007).

The simulation is conducted using the intensity-based thinning algorithm as introduced by Lewis and Shedler (1979) and modified by Yosihiko Ogata (1981) and is used to define the time at which orders arrive. Each thinning algorithm submitting large numbers of orders may be thought of as clients. So, in the case of the testing framework, a single client is associated with each stock and is meant to represent a large group of traders investing in that stock. The prices and volumes are generated based on the order type in a fairly random manner.

First, a maximum LOB depth $M = 10$ is specified. A lower limit for the price at which buy limit orders are generated is set to $L_b = 25000$. Similarly, an upper limit for the price at which sell limit orders are generated is set to $H_s = 25057$. The LOB is initialised with an initial limit and buy $I_b = 25034$ and sell $I_s = 25057$ order. Thereafter prices and volumes are generated according to a

[21] JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targetting the JVM

random normal distribution in these bounds. According to this bids and asks can cross each other — in which case limit orders become effective market orders. The VWAP is used to calculate the price for an aggressive buy/sell trade (not the same as the execution price) that executes against the LOB. That is, if an aggressive trade affects the first k highest/lowest levels of the order book then the price is calculated as:

$$\frac{\sum_{i=1}^{k} \text{Price}_i \times \text{Volume}_i}{\sum_{i=1}^{k} \text{Volume}_i}$$

For the next section, test scenarios were created to test the performance of the software by evaluating the impact of multiple clients-stock pairs. That is, each stock is assigned a unique client who sends high volumes of orders. By increasing the number of stocks and clients after each subsequent run, the volume of messages being processed increases as well — during which time the throughput per second and latency are automatically recorded. These performance tests have been conducted on multiple computers, all with different operating systems and hardware specifications. Here the performance results are documented by running the application on the two machines listed in Table 4.5. The average start-up time of the web application is approximately 50 seconds.

Machine	Operating System	Memory	Processors
WITS MSS Server	64-bit Linux Ubuntu 16.04-LTS	32GB	4× Quadcore AMD Opteron 8356 @ 2.3GHz (16 cores)
Standard A4m V2 Azure VM	64-bit Linux Ubuntu 18.04-LTS	32GB	4× Intel Xeon CPU E5-2673 v3 @ 2.4GHz (32 cores)

TABLE 4.5: Hardware specifications of machines used for throughput and latency testing used to compare the cloud solution (Azure) to the dedicated physical hardware solution (WITS MSS Server blade).

Although not presented here, limit order book storage testing was also conducted on the WITS Mathematical Science Support provided server hardware and demonstrated the ability to store thousands of orders at each price point. The design of the LOB also allows orders to be added and removed easily

4.5.3 Latency tests

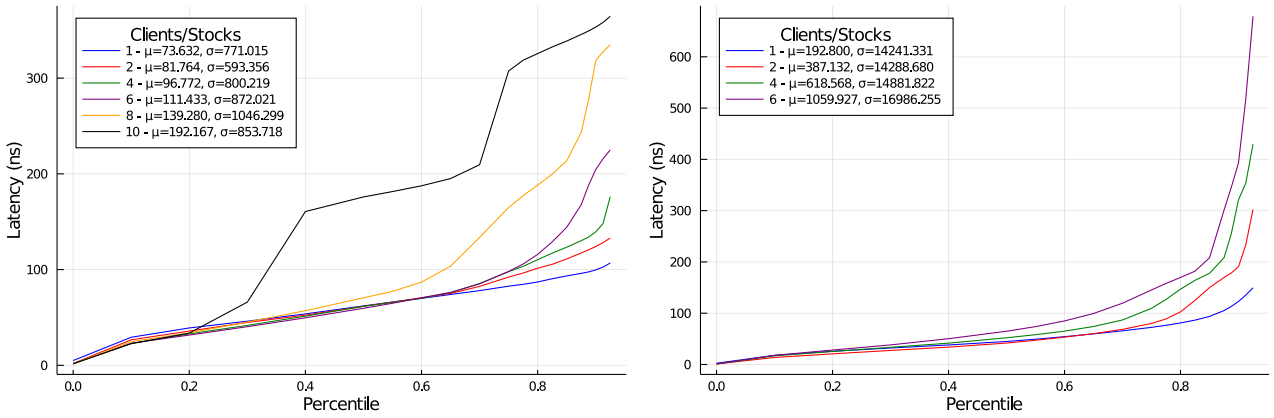
The latency was tested and visualized using the HdrHistogram library[22]. With every run, each client submits approximately 110 000 orders. The time it takes to process all these orders is then measured and compared between machines. To reproduce Figure 4.4 simply re-run the Hawkes simulations for the system to write latency and throughput results to file.

Figure 4.4 shows the latency results, in nanoseconds, for runs with differing numbers of active client-stock pairs on each machine. The latency increases as the number of clients-stock pairs increase — since each client is associated with multiple threads running processes in parallel. Due to there being only 4 CPUs on the Azure VM, running more than 6 clients simultaneously was found to be infeasible. For this reason the user should ensure that the hardware on the server is capable of supporting the desired number of clients. For the high-spec Wits Server machine (Figure 4.4a) the minimum and maximum latency (measured up to 10 clients) at the 90th percentile is 106ns and 248ns, respectively. Similarly, for the medium-spec machine (Figure 4.4b) (with a maximum of 6 clients) the minimum and maximum latency at the 90th percentile is 123ns and 393ns, respectively. Therefore, on a high

[22]HdrHistogram is designed for recording histograms of value measurements in latency and performance sensitive applications. Measurements show value recording times as low as 3–6 nanoseconds on modern (circa 2012) Intel CPUs

end machine one can expect sub 250ns latency (with 10 clients), while with a medium spec machine one can expect sub 400ns latency (with 6 clients)[23].

Figure 4.5 considers the scenario where high volumes of orders are submitted to the trading gateway. One million orders are submitted from a single client and compared across the two machines. For the Wits server, the latency is 735ns at the 90th percentile but maintains a significantly lower latency on average. On the other hand, the Azure server has higher latency's on average with a latency of 964ns at the 90th percentile.



(A) Wits Server high specification machine

(B) Azure medium specification virtual machine

FIGURE 4.4: A High Dynamic Range (HDR) Histogram graph showing the latency percentiles of the matching engine for increasing numbers of stocks and clients on both WITS (4.4a) and Azure (4.4b) servers. In the testing framework each client submits approximately 110000 market/limit orders.

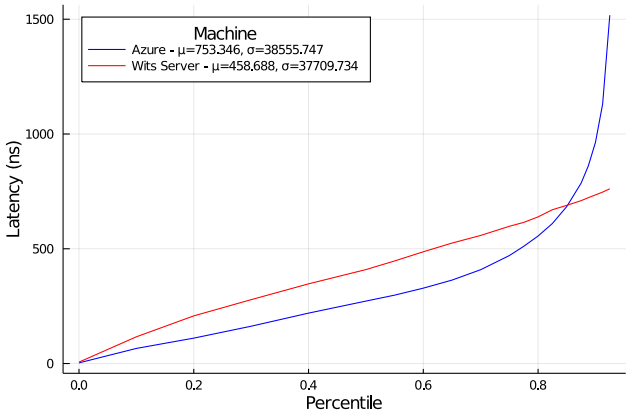


FIGURE 4.5: A comparison of latencies during a high volume scenario. One million orders were submitted by a single client on both the WITS and Azure servers.

4.5.4 Throughput tests

Table 4.6 shows the throughput per second as the number of clients-stock pairs increase on each machine. To reproduce the data in Table 4.6 simply re-run the Hawkes simulations for the system to write latency and throughput results to file. It should be noted that each client waits for market data updates before calculating the next order to send. The client also waits a few nanoseconds as part of the Hawkes simulation. These delays reduced the throughput of orders sent. The results show that as the number of clients-stock pairs increase, the throughput decreases. The most significant decrease

[23] JSE's average round-trip colocation network latency is sub 100 microseconds and it's matching engine has a latency of 50 microseconds (Johannesburg-Stock-Exchange 2015).

in throughput is when more than 4 clients and stocks are used. Beyond 6 clients the hardware configuration of the Azure VM was found to be insufficient. In total, the high spec machine was shown to be capable of processing more than one million orders in less than a 19 minute simulation period with significantly low latencies. On the other hand, the medium spec machine was capable of processing 450000 and 560000 orders in approximately 44 minute and 2 hour simulation periods for 4 and 6 client-stock pairs, respectively.

	Wits Server			Microsoft Azure		
client-stock pairs	Duration	Number of orders	Throughput per second	Duration	Number of orders	Throughput per second
1	00:00:48.820	111646	2287	00:06:44.917	110825	274
2	00:04:23.981	224562	850	00:19:38.576	222390	189
4	00:12:02.204	448774	621	00:43:55.734	447878	170
6	00:15:12.386	669331	733	02:01:27.494	564070	77
8	00:20:27.010	895080	729	-	-	-
10	00:18:52.289	1120514	989	-	-	-

TABLE 4.6: Tabulated measurements of throughput per second with increasing numbers of client-stock pairs. Based on the Hawkes process parameters each client submits approximately 110000 orders. After each simulation the start and end times are published and used to calculate the throughput per second. The hardware configuration on the Azure VM was found to be insufficient when defining more that 6 clients. All data generated in this table can be acquired at the end of each simulation in the data directory of the start-up folder.

4.6 Simulation results

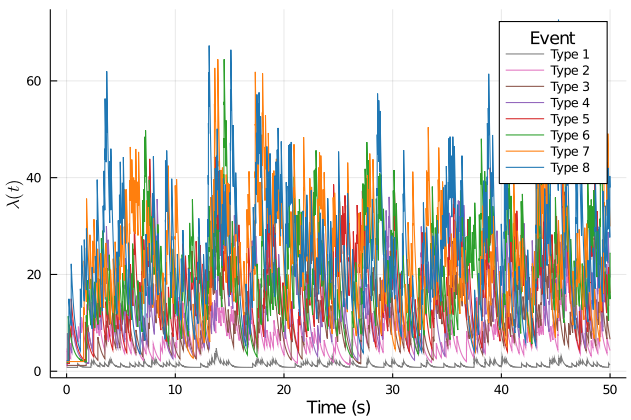
The first set of results in this section relate to the simulation for a single stock using a simple Hawkes process as a proof of concept. Simulation results are contained in two `.csv` files for each security: one for a history of all orders submitted and the last for a snapshot of the limit order book at then end of the simulation. The data output follows the format below and is only written out to file once all clients are logged out at the end of the simulation so that latency is not compromised during trading sessions. There are some caveats regarding the format and way in which orders are printed to file.

Execution times are printed at millisecond precision and are given by the UTC standard. Each order is associated with the ID of the trader who submitted the order ("TraderMnemonic"), the ID assigned to the order ("ClientOrderId" used to track cancelled and executed orders) as well as the price, volume, side and type of the order.[24] In this way, all orders can be assigned unique identification numbers by the client. Orders labeled with type "New" refer to limit orders. Trades with a large volume submitted will walk the LOB. These will execute at multiple price levels and are split into multiple trades corresponding to the the different orders against which they are executed in the reported data. In this way, as shown below, market orders are also duplicated in the data output. The initial occurrence of a market order specifies the total volume (full "unbroken" market order) submitted and is given a price of 0 as well as a unique client submission ID. This represents the un-broken/un-split market order before it is aggressed against the contra side of the LOB. Further occurrences of orders marked "Trade" represent the broken/split trades which have executed at one or multiple levels of the order book. These entries have IDs that correspond to the limit orders against which they were executed. Similarly, cancel orders are assigned the ID of the corresponding limit orders. In addition to order identification numbers, orders in the output also carry a trader ID identifier specifying the ID of the trader who submitted the order. These IDs should match those specified in the "Trader.csv" file (see A.1.4). This data is only printed to file at the end of the simulation once all logged-in clients have logged out. The format of these files are shown in snippet 4.6 below.

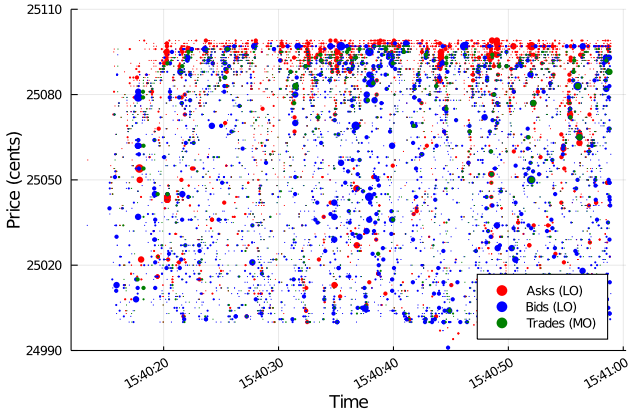
[24]Note that a single "client" logged in to CoinTossX need not be associated with a single trader. The "clients" in CoinTossX may be interpreted as a group of traders or an investment firm.

```
"DateTime","TraderMnemonic","ClientOrderId","Price","Volume","Side","Type"
"2021-05-19 11:35:34.827","1","1","20","100","Buy","New"
"2021-05-19 11:35:43.405","1","3","0","150","Sell","Trade"
"2021-05-19 11:35:43.406","1","2","49","100","Sell","Trade"
"2021-05-19 11:35:43.415","1","1","20","50","Sell","Trade"
"2021-05-19 11:35:47.027","1","2","49","0","Buy","Cancelled"
"2021-05-19 11:35:59.973","1","4","50","100","Buy","New"
```

FIGURE 4.6: A snippet of the file format for trade data output from CoinTossX.



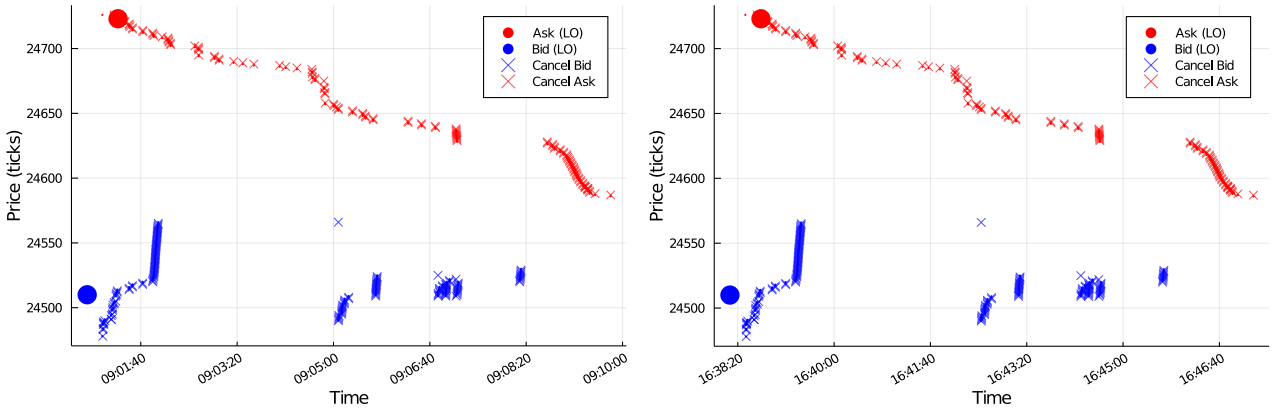
(A) A Graph of the intensities of the 8-variate Hawkes process for 50 seconds of the Hawkes simulation.



(B) A Graph of the limit and market orders submitted to the trading gateway. As a proof of concept, the Hawkes simulation assigns volumes and prices in a fairly random and unrealistic manner as the bid and ask are allowed to cross (negative spread). The size of the points are proportional to the volume of the order.

FIGURE 4.7: A visualisation of a single simulation period obtained from the Hawkes process defined in the testing framework (Section 4.5) to be used for latency, throughput and unit tests.

To provide context, and with the aim of validating and stress testing the system, we provide a visual comparison of order submission results between a real commercial exchange and CoinTossX. In doing so, given detailed TAQ data from A2X, we submit a subset of the exact same orders to CoinTossX by preserving limit prices, volumes, cancellations and inter-arrival times — as some artificial exchange softwares refer to as a "replay engine" (Brandouy, Mathieu, and Veryzhenko 2013). The TAQ data from the A2X exchange and CoinTossX are visualised in Figures 4.8a and 4.8b, respectively. Other simulations adopting agent-based modelling and point-process methods have been demonstrated in Jericevich, Chang, and Gebbie (2021a) and Jericevich, Chang, and Gebbie (2021b) (see Chapters 5 and 6).



(A) Original A2X orders

(B) A2X orders re-submitted to CoinTossX

FIGURE 4.8: Visual comparison of historical A2X TAQ data re-submitted to CoinTossX. A2X orders preserve their prices, IDs, volumes and inter-arrival times when submitted to CoinTossX.

4.7 Summary

CoinTossX is a low latency high throughput artificial stock exchange. It is configurable and allows users to view the limit order book in real time as well as for multiple clients to connect and send orders to the exchange. The exchange supports multiple stocks and a variety of different trading session. Here Hawkes processes are used to provide a simple but robust simulation of order arrivals for infrastructure testing. The CoinTossX website can be enhanced to analyze the data that is processed. We also recommend further improvements to latency and simulation quality by adopting the actor framework of computation[25]. The exchange provides a realistic platform for agent based models exploration. The software was designed so that different matching logic algorithms are in separate Java classes. This allows the logic to be changed to test any variations of the matching logic or market rules. Hence, additional work can be done to change the matching rules on the engine to test the impact of rules and regulation changes on the limit order book and its dynamics. Since the components are de-coupled, their implementation language can be continually and easily changed to support the latest frameworks (Sing 2017).

The matching engine can provide a realistic platform for market model exploration and interaction. It may allow traders, organisations and academic institutions to test market structure, fragility and dynamics without the cost of live test trading, but with the realism of interaction via asynchronous market rules. Chapter 5 will be aimed at better understanding the interaction dynamics from increasingly realistic approaches to point-process based simulations *e.g.* with trading clients based on Hawkes processes using models such as those of Bacry and Muzy (2014) and Zheng, Roueff, and Abergel (2014) to investigate the interactions of limit-order and market-order trading agents through a realistic order matching process. Ultimately this type of work is aimed at better understanding high-concurrency agent-based perspectives for market modeling and their impact on model calibration (Platt and Gebbie 2018) and causation.

[25] By using the actor model of computation orders and clients can be associated unique behaviours that are specific to their types — allowing for improved speed, a smaller memory footprint and simplified use of distributed processes.

5 Point process model

The extent to which a matching engine can cloud the modelling of underlying order submission and management processes in a financial market and the importance of this with regards to the statistical robustness of models and resulting inference remains an unanswered concern for market models. Here we consider a 10-variate Hawkes process with simple rules to simulate common order types which are submitted to a matching engine. Hawkes processes can be used to model the time and order of events, and how these events relate to each other (see Section 2.2). However, they provide a freedom with regards to implementation mechanics relating to the prices and volumes of injected orders. This allows us to consider a reference Hawkes model and two additional models which have rules that change the behaviour of limit orders. The resulting trade and quote data from the simulations are then calibrated and compared with the original order generating process to determine the extent with which implementation rules can distort model parameters. Evidence from validation and hypothesis tests suggest that the true model specification can be significantly distorted by market mechanics to the extent that the models are no longer statistically equivalent. It can be argued that the matching mechanism merely blurs the estimation and identification of the process, and this is a triviality — we argue that this can lead to fundamental changes in the models as they are no longer statistically equivalent.

5.1 Introduction

The matching engine is independent of the modelling framework and order submission, and it directly exposes modelling to the vagaries of time-delays, message implementation rules and event driven asynchronicity. This implies that there can be a layer of inter-mediation that resides between the models, the data generation and observation processes whose impact on model decisions needs to be understood in terms of potential impacts on model identification, stability and interpretability. In other words, the extent to which the matching engine can cloud the modelling of the underlying order submission and management processes remains an unanswered concern with regards to market models.

This work demonstrates some aspects of this problem. Specifically the impact of the additional layer of complexity from the matching engine. There are two key aspects in this problem. First, given a model, can the model be reasonably identified and calibrated once model events are inject into a matching engine and then emerge from the data feeds? Second, how different can the identified model be from the actual model, or some reference model, when there are other practical considerations that come into play? In addition, this type of work can address some aspects of the unavailability of data and direct data feed access from industry to researchers, by providing a framework that can be compared to recorded transaction data arising from the actual market system interfaces using synthetic data (Raman and Jochen L Leidner 2019). However, the type of model used here can at best be used to inform thinking about infrastructure integrity under various stress test scenarios as it cannot recover all the required stylised facts. In particular, a power law dependency in the limit order placement and other aspects of measured price impact (Lillo, Farmer, and Mantegna 2003). The intention of this work is not to code for stylised facts but to demonstrate how implementation rules can interact with model specifications in a simulated environment.

The aim of this chapter is to determine the extent to which market mechanisms and practical considerations from the limit order book can distort standard Hawkes model parameters. This is accomplished

by specifying two minimally intelligent models for the submission of orders to a matching engine. Section 5.2 starts with a summary of the theory behind Hawkes processes together with a discussion on the simulation set-up performed with CoinTossX. This includes a discussion around the event-types considered, the rules for submitting events/orders, caveats of our simulations with CoinTossX, model parameter choice, order volume choice, and lastly basic results generated from each model implemented with CoinTossX. Section 5.3 demonstrates the estimation procedure followed together with results informing the extent to which simulation model parameters are distorted. The maximum likelihood calibration routine and the resulting parameter estimates are compared through a number of qualitative and quantitative tests for goodness-of-fit. Section 5.4 performs a likelihood-ratio test to confirm our results and the implications are discussed. Finally, Section 5.5 summarises our findings.

5.2 Simulation

Simulation is performed using the thinning procedure (Muni Toke and Pomponio 2011; Lewis and Shedler 1979; Yosihiko Ogata 1981) (see Section 2.2.1 and Algorithm 1) which can be summarised as follows: let $I^K(t) = \sum_{k=1}^{K} \lambda^k(t)$ and let t be the current time. We sample an exponential inter-arrival time with rate parameter $I^M(t)$, call it τ. A random uniform u is then sampled from $[0, I^M(t)]$, and if $u < I^M(t) - I^M(t+\tau)$, the arrival time $t+\tau$ is rejected. Otherwise, the arrival time $t+\tau$ is accepted and attributed to the ith component, where i is such that $I^{i-1}(t+\tau) < u \leq I^i(t+\tau)$.

5.2.1 Event types

The benefit of using Hawkes processes to model the continuous double auction is that each message sent to the exchange by a trader or institution can be thought of as an event. This means that we only need to identify the type of events and specify the corresponding rules for each event.

Type #	Event type	Side
1	Market Order	Bid
2	Market Order	Ask
3	Aggressive Limit Order	Bid
4	Aggressive Limit Order	Ask
5	Passive Limit Order	Bid
6	Passive Limit Order	Ask
7	Aggressive Cancellation	Bid
8	Aggressive Cancellation	Ask
9	Passive Cancellation	Bid
10	Passive Cancellation	Ask

TABLE 5.1: The various event types simulated using the Hawkes process.

Orders sent during a simulation are done in a similar manner as that of any other exchange. There are a wide variety of messages that a trader or institution can send to the exchange. These include but are not limited to: Limit Orders (LOs), Market Orders (MOs), cancellations, modifications, Hidden Orders (HOs) and many more. For the purposes of the simulation, we only use the common message types that can be easily identified (without raw message data) and do not result in an excessively complex Hawkes process. The message choices used are: MOs, LOs and cancellations. The LOs and cancellations are further split into aggressive and passive types, leaving us with a total of five message types. Each of these message types are further split into the bid side and ask side, resulting in a total of 10 event types. The various event types used are enumerated in Table 5.1.

Order modifications are also a commonly used message type. However, including these lead to too many possible event types which results in a Hawkes process that is too complex to reasonably calibrate with a day of data. Therefore, modifications messages are excluded from the simulation.

Hawkes reference model

Here we do not need any implementation rules to generate order messages because we do not inject messages into the matching engine. Hence there is no need for any implementation mechanics as we can directly calibrate to the simulated data using event types and their associated event times (see Section 5.3.3). The reference model does not produce any volumes or prices associated with the order

book events hence cannot be considered a physically meaningful model — it only generates order event types and their event times.

5.2.2 Implementation rules

Since a Hawkes process only produces event types and their timings, an associated action must be taken to send a message to the matching engine for each of the event types. We call these the implementation rules. The rules create models that largely drop agent rationality and instead focuses on the problem of understanding the practical implementation constraints. We focus on two separate approaches to managing limit orders via two cases: where limit orders are independent of the spread and where limit orders are dependent on the spread. Both of these cases will share the basic implementation rules for cancellations and market orders.

The basic rules are demonstrated in Pseudocode 2 in the Appendix. These include rules for the implementation of all the order event types in Table 5.1. We now discuss how these are implemented as actual messages.

Cancellations require an order identification number (Id.), a side and a price. The order Id is determined by the state of the order book and whether the cancellation is aggressive or passive. The side is determined by the event type. The price is determined by the corresponding order Id. For passive cancellations, we extract all the orders from the appropriate side of the order book depending on the side of the cancellation. A random order with a price that is not equal to the best is sampled and then cancelled. For aggressive cancellations, we extract all the orders from the appropriate side of the order book depending on the side of the cancellation. A random order with a price that is equal to the best is sampled and then cancelled.

Market order messages only require a volume and side. The choice of volume is discussed in Section 5.2.3. The side is determined by the event type. The resulting price is completely determined by the matching engine when it crosses the MO against existing LOs according to the price-time priority strategy. The MO is not sent if there are no orders on the contra side for the MO to cross with. This is to make things easier when cleaning data to identify events in Section 5.3.1. If the MO was sent regardless, the matching engine would simply ignore the MO and we still would not have a transaction. In the case when the MO volume is larger than the combined volume of LOs on the contra side, the MO is only partially filled and the remaining MO is ignored.

Limit order messages require a volume, a side and a price. The choice of volume is discussed in Section 5.2.3. The side is determined by the event type. The price is determined by the state of the order book and whether the LO is aggressive or passive. If the LOB is empty from the same side as the initiating LO, the price is then set to be a random integer from 1 to 10 ticks away from the best on the contra side. For example, if we have a LO from the bid side (either event type 3 or 5) and the bid side of the LOB is empty, we set the price to be 1 to 10 ticks below the best ask. In the case when both sides of the LOB are empty, we apply the same rule as above, with the exception that the reference price on the contra side uses the previous best bid or ask before the LOB became empty.

For passive LOs, we set the price to be one tick worse than the current best. For example, if we have a passive LO from the bid side (event type 5), we set the price to be one tick below the best bid. For aggressive LOs, we have two rules to set the price which results in two separate models. This implies that the Hawkes model based simulation here will lead to three distinct models: *i.)* the Hawkes reference model in Section 5.2.1 for the events and their times, *ii.)* Model 1 and *iii.)* Model 2 for the different implementations of aggressive limit orders.

Limit order model 1

This model has no mechanism controlling for the spread. Here the price is always set to be one tick better than the current best even if it results in the new best crossing the contra side. For example, if

we have an aggressive LO from the bid side (event type 3), we set the price to be one tick above the best. The rules are demonstrated in Pseudocode 3.

For this model, the matching engine handles the crossing of LOs by converting the crossing LO into a MO to execute as a trade. The excess volume that did not execute as a trade remains in the LOB as a limit order at its original crossing price. For example, consider the best ask being a single order at a price of 50 with a volume of 30. Suppose a new LO bid is placed at a price of 50 and a volume of 70. Then a volume of 30 from the new LO will be executed as a trade against the best ask resulting in a trade at a price of 50 with a volume of 30. The remaining volume of 40 from the LO bid now becomes the best bid with a price of 50.

Limit order model 2

The limit price for this model is dependent on the current spread. If the spread is greater than one tick, the price is set to be one tick better than the current best. Otherwise if the spread is equal to one tick, the price is set to be the same as the current best. This model controls the spread to ensure that LOs do not cross the order book. For example, if we have an aggressive LO from the bid side (event type 3), we set the price to be one tick above the best if the current spread is greater than one tick. Otherwise we set the price to be the same as the current best if the current spread is equal to one tick. The rules are demonstrated in Pseudocode 4.

5.2.3 Simulation initialisation

The simulations implemented are restricted to the continuous trading session with order types only being limit, market and cancel orders which have a DAY time-in-force.[1] A single client and stock is adopted for simplicity to represent the market as a whole. The simulation is performed in Julia by using the simulated 10-variate Hawkes and submitting the appropriate order to the matching engine at the correct time based on the event type and the rules defined. The simulation rules require that the client receive market data feedback from the matching engine in the form of a snapshot of the limit order book and the current best bid/ask.

Timeouts may occur if the inter-arrival times are less than the time it takes for the matching engine to process the data plus the time it takes for Julia to receive market data and process the next order. This occurs if there was a large build up of orders which affects processing time and round-trip latency. Cancellation orders are the most affected because this requires running through all the orders on one side of the LOB to determine if the price of an order is equal or not equal to the best, to then find the appropriate sample of orders from which we can cancel. It is therefore important that we have appropriate Hawkes parameters and volume choices to ensure that neither model has a large build up of orders.

Having a matching engine at our disposal means that simulations need not be conducted in real-time, *i.e.*, we can speed up the simulation time relative to real business or calendar time assuming that we do not force timeouts. Furthermore, data output can be modified when needed which can be convenient for debugging and analysis.

Parameter choice

The parameters for the Hawkes process used in the simulation were not calibrated to real data. This was a pragmatic choice. When calibrating with real data we need to worry whether we have the correct specification and then whether we have an indicative window of data with enough events to empirically capture the specification under estimation. Finding the right specification may inadvertently lead to a Hawkes process that is more complicated than necessary which may lead to estimation complications later on — for example using a sum of exponential kernels significantly increases the number of

[1] The DAY TIF is the default for all orders and implies that these orders will only be valid for the day they are submitted.

parameters required. This would not change the arguments made in this work — it would just add unnecessary complexity.

In tandem with model complexity is the issue of having sufficient observations of all event types of interest. Many datasets may have periods of insufficient order types to calibrate the process for particular scenarios that may have features of interest. For example, the A2X exchange has many days where there are less than 10 transactions per day. Therefore, it has insufficient market orders for calibration. Excluding market orders is not always an option because it is can be an essential feature of the model.

The problem is generic in that Hawkes process modelling seems effective with regards to creating a model snap-shot of a particular set of market conditions in a particular window of data but requires sufficiently representative data for identification. It can be a powerful visual diagnostic tool that can relate order types under a particular set of market conditions. However it becomes difficult to create simple models that capture the dynamics of interest with a sparse finite piece of data.

Another reason why we did not calibrate to real data is to ensure that we achieve a correct balance in liquidity. Incorrectly balancing liquidity can cause latency problems when receiving market data from the matching engine which can cause timeouts in our simulation.[2] For this reason extracting data snapshots of the LOB under simulation was a key area requiring some additional development work.[3][4]

The parameters we used were chosen to ensure that we maintain balance between events that provide liquidity (event types 3–6) and events that remove liquidity (event types 1–2 and 7–10). Obtaining this balance is important because if there are too many events that take away liquidity then the LOB will often be empty. If there are too many events that provide liquidity then there will be a large build up of orders, particularly for model 2, which can result in no movements in the price. The Hawkes process we implemented was then with a single exponential kernel with parameters given as: $\boldsymbol{\alpha} = \boldsymbol{\mu}\mathbf{1}^\top$, $\boldsymbol{\beta} = \{\beta^{mn}\}_{m=1,n=1}^{10}$ where $\beta^{mn} = 0.2$ for all m and n, and the baseline intensity $\boldsymbol{\mu}$ as the vector:

$$[0.01, 0.01, 0.02, 0.02, 0.02, 0.02, 0.015, 0.015, 0.015, 0.015]^\top.$$

The challenge of balancing liquidity is perhaps a key motivation for moving away from point-process type models to models that can capture a more strategic and dynamic interplay between market participants.

Volume choice

The final component required to conduct simulations is a choice for the volumes associated with MOs and LOs. There seems to be no consensus for a model specification for the volume of orders as the distributions varies widely with the product and market. However, it was empirically found that the unconditional distribution of orders follows a power-law behaviour (Chakraborti et al. 2011a).

Therefore, we will assume that the volume associated with each LO and MO is from an independent and identically distributed (IID) power-law where the probability density function is given as:

$$f(x) = \begin{cases} \frac{\alpha x_{\mathrm{m}}^{\alpha}}{x^{\alpha+1}} & x \geq x_{\mathrm{m}} \\ 0 & x < x_{\mathrm{m}}. \end{cases} \tag{5.1}$$

[2] This is a data monitoring restriction rather than an order book latency problem and required an enhancement of the requests associated with LOB snapshots. It is recommended that the LOB does not get too big, *i.e.*, > 100 orders on each side.

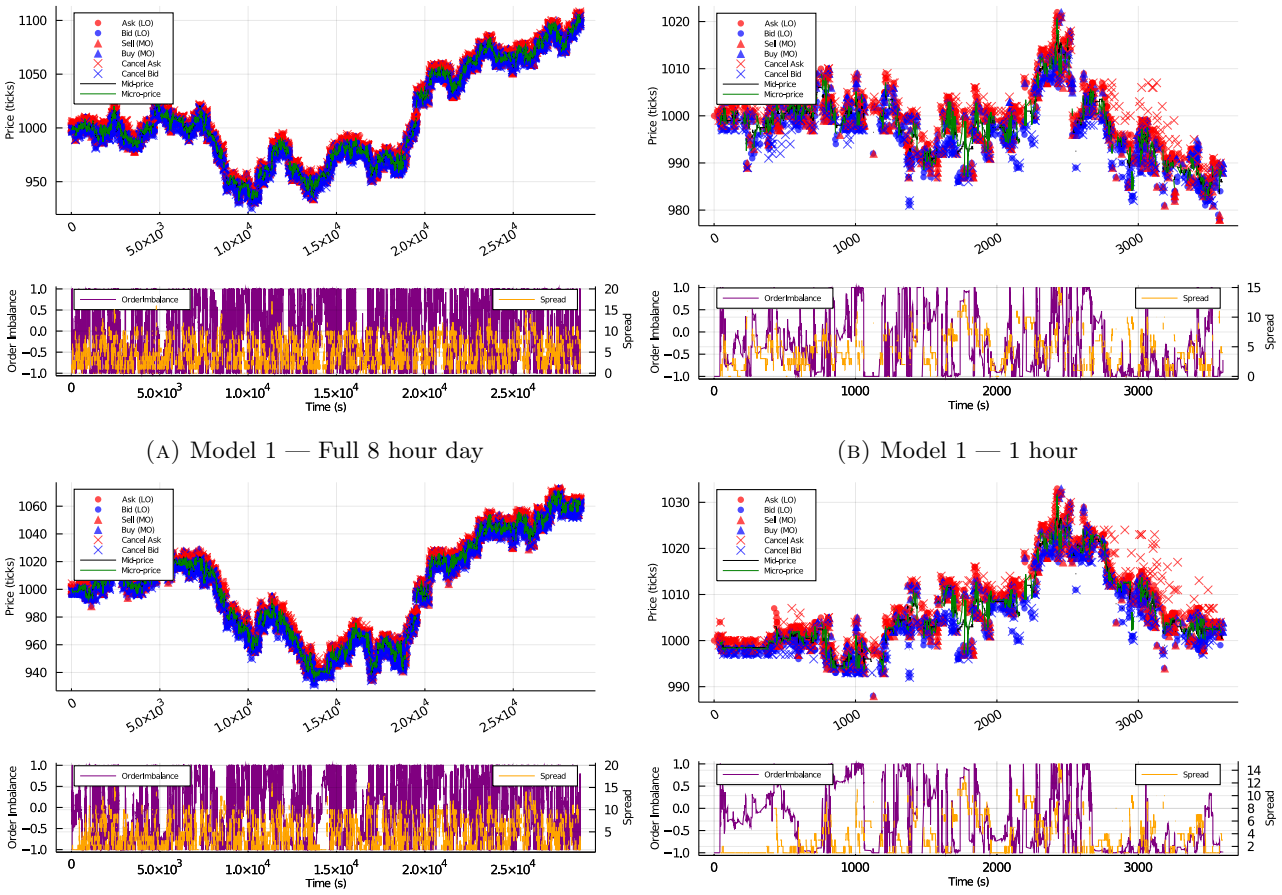
[3] A problem was also identified in the processing of LOs that crossed the spread, whereby the contra side would be executed against the crossed LO instead of vice versa. This had no affect on the matching dynamics but impacts the trade reporting.

[4] The reporting of trades and order cancels was improved by assigning their IDs that of the corresponding order that was executed.

We set x_{m} to be 20 for LOs and 50 for MOs, whereas we set $\alpha = 1$ for both MOs and LOs. These choices along with our parameter choices ensured that there was never a large build up of orders in model 2.

The volume sizes play an important role in ensuring that there is not a large build up of orders for model 2.[5] In particular, the total MO volume cannot be too small relative to the total LO volume. This is because even though there are cancellations to assist in removing liquidity, cancellations can also occur on partially filled LOs which can then still result in a build up of orders. Moreover, depending on the state of the order book, MOs and cancellations are sometimes lost because there is nothing to execute against or nothing to cancel, whereas LOs will always be placed.

5.2.4 Simulation results



(A) Model 1 — Full 8 hour day

(B) Model 1 — 1 hour

(C) Model 2 — Full 8 hour day

(D) Model 2 — 1 hour

FIGURE 5.1: The simulation of a 10 variate Hawkes process used to generate the different order types submitted to CoinTossX in Table 5.1 is visualised over an 8-hour and 1-hour period and compared across both models. Together with the plot of orders submitted (top), we visualise the evolution of the spread (bottom, right axis) and the order imbalance (bottom, left axis).

Simulations for both model 1 and model 2 are performed using the same realisation of the Hawkes reference model with the same event types, counts and volumes. The simulations were performed using Pseudocode 2 and the only difference between the two models are in the rules for limit order price placement.

[5] Model 1 does not suffer from a large build up of orders because the model permits LOs to cross the spread. Therefore, this model has a natural way to remove too much liquidity.

Figure 5.1 plots the resulting simulation of the two models. The first and second rows are model 1 and 2 respectively, and the first and second columns are the full 8 hour day and the first hour respectively. Additionally, the evolution of the mid-price, micro-price, spread and order-imbalance is tracked to gain a better view of the underlying dynamics. The following dynamics can be thought of being representative of a low-liquidity market, given the event counts in Table 5.2.

Despite there being significantly more market orders for model 1 (refer to Table 5.2), both models' price series were found to be fairly similar. Despite this similarity, model 1 exhibits a slightly greater price range by the end of the day. The spread remained erratic throughout the simulation period, reaching as high as 15 ticks. Over the shorter time-period, as expected, breaks on either side of the LOB would result in more aggressive price movements than would otherwise occur in a high liquidity market.

5.3 Estimation

5.3.1 Event identification

In order to compare and investigate how market mechanisms may affect our ability to infer the underlying process we first need to classify the resulting data from the models appropriately. We use the same set of event types in Table 5.1.

When classifying market orders we need to be careful. Trades with a large volume submitted will walk the LOB. These will execute at multiple price levels and are split into multiple trades corresponding to the the different orders against which they are executed in the reported data. These transactions will also all report the same timestamp. Therefore, these transactions must be aggregated and counted as one event.

Aggressive limit orders are any of those *(a)* whose limit price is equal to the current best, *(b)* whose limit price is better than the best, or *(c)* that occur in an empty LOB — thereby becoming the new best. Passive limit orders are those whose limit price is worse than the current best.

Cancel orders are classified as aggressive if the order cancelled was in the L1LOB (at the best). Otherwise the cancellation is classified as passive.

5.3.2 Distortions due to market mechanics

From practical market mechanic considerations dependant on the state of the order book we can explain why certain classified events arising from the matching engine deviate from the true event type injected into the matching engine. There are several cases that contribute towards this distortion.

First, from Pseudocode 3 and 4, both passive and aggressive LOs can be submitted to an empty LOB.[6] This order immediately becomes the best which means that passive limit orders can become aggressive.

Second, cancelled orders will not go through the matching engine and be reflected in the data feed when there are no appropriate orders to cancel. In particular, both passive and aggressive cancellations will not be executed when the LOB is empty on the side which the cancellation came from. Alternatively, passive cancellations cannot cancel orders when there is only price level (the best) in the LOB. Similarly, market orders are ignored when it arrives while the contra side of the LOB is empty.

Lastly, model 1 requires additional care for limit orders that cross the spread. Limit orders that cross the spread become an effective market order. Additionally, the crossing can spawn an additional event which results in an effective market order and an aggressive limit order. This only occurs when the

[6] This typically only occurs in markets with very little liquidity (*e.g.* A2X).

crossed limit order was only partially executed against the contra side, *i.e.*, when the incoming LO has a volume larger than the best on the contra side.

The case when an additional event is spawned becomes problematic because these two events will have the same timestamp. Therefore, to overcome the problem of concurrent events in a point process we add one millisecond to the additional aggressive limit order that was spawned. We can do this because we know that for the matching engine, limit orders that cross will first execute as a market order before it becomes a limit order with a reduced volume. Since we know the true ordering of events, we can make the small adjustment to ensure that the data remains a simple point process (see Large (2007) and the discussion around concurrent events).

Table 5.2 reports the event counts from the Hawkes reference model and the events classified from model 1 and 2. We see that the event counts from both models deviate from the Hawkes reference model used to create the models. The key question is: Do these market mechanisms and practical considerations affect our ability to infer the underlying process generating the models?

Type number	Hawkes count	Model 1 count	Model 2 count
1	1436	1533	1228
2	1380	1579	1268
3	2873	3071	3164
4	2875	3046	3152
5	2736	2356	2445
6	2726	2360	2448
7	2135	1841	1918
8	2075	1799	1866
9	2095	1379	1490
10	2068	1393	1517

TABLE 5.2: The various event counts for the three models.

5.3.3 Calibration

We use the popular parametric approach to calibrate a Hawkes process using the Maximum Likelihood Estimation (MLE) (Bacry, Mastromatteo, and Muzy 2015; Muni Toke and Pomponio 2011). To calibrate we maximise the log-likelihood (the log-likelihood and the optimisation details are given in Section 2.2.1)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \ \ln \mathcal{L}(\theta), \tag{5.2}$$

where θ refers to the parameters of the Hawkes process which include $(\mu^m, \alpha^{mn}, \beta^{mn})$, depending on the specification of the model. Here we do not enforce a specific design on the matrices $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$ and allow the calibration to determine the structure of the matrices. Based on the events identified from Table 5.2 we have 10 event types and therefore we have a total of $10 \times 10 \times 2 + 10 = 210$ parameters.

Here we do not enforce a specific design on the matrices $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$ and allow the calibration to determine the structure of the matrices. Therefore, we have a total of 210 parameters. We perform the calibration using MLE to find $\hat{\theta}_{\text{Hawkes}}$, $\hat{\theta}_{\text{M1}}$ and $\hat{\theta}_{\text{M2}}$ for the Hawkes reference model, and for model 1 and 2, respectively, with initial parameter values set to the true starting values θ_{true}.

Our optimisation of Equation 2.14 employs the gradient descent method L-BFGS from the optimisation package "Optim" (Mogensen and Riseth 2018). The method takes steps according to

$$\theta_{h+1} = \theta_h - \kappa P^{-1} \nabla \ln \mathcal{L}(\theta_h), \tag{5.3}$$

where κ is a scalar chosen by a linesearch algorithm to ensure that each step gives sufficient descent, P is an approximation to the Hessian using the latest m steps and $\nabla \ln \mathcal{L}$ is the first derivative of our log-likelihood found through numerical approximations using the package "ForwardDiff" (Revels, Lubin, and Papamarkou 2016). The search begins at the true parameter values θ_{true} defined in Section 5.2.3.[7]

The results are summarised by measuring the deviation between the MLE parameters $\hat{\theta}$ against the original Hawkes parameters θ_{true} from Section 5.2.3 using the mean absolute error (MAE)[8] and the root

[7] To the best of our knowledge, closed form solutions for the gradients and Hessian have not been found for a multivariate Hawkes process. Closed form solutions for the univariate case can be found in Ozaki (1979).

[8] $\text{MAE}(\hat{\theta}, \theta_{\text{true}}) = \frac{1}{210} \sum_{k=1}^{210} |\theta_{\text{true},k} - \hat{\theta}_k|$.

mean square error (RMSE)[9]. Table 5.3 reports the MAE and RMSE of the calibrated parameters for the Hawkes reference model and for model 1 and 2 against the original Hawkes parameters. We see that the calibrated parameters from the reference model present the smallest deviation from the original parameters. This is expected because there are no market mechanisms or practical considerations changing the event types, therefore the small deviation is an indication that our calibration procedure is correct. A more detailed visualisation of the parameters and their deviations between models is given in Section 5.4:

Measure	Hawkes	Model 1	Model 2
MAE	0.0542	16.3300	25.8408
RMSE	0.1439	107.0479	224.6096

TABLE 5.3: Measures of deviation between Hawkes input parameters and calibrated parameters for the Hawkes reference model as well as for model 1 and 2 simulations.

The calibrated parameters from model 1 and 2 both deviate significantly from the original parameters. This is expected because from Table 5.2 we see that model 1 and 2 are affected by market mechanisms and practical considerations which cause additional events to spawn, events being dropped and event types changing. Therefore, the large deviation is an indication that market mechanisms and practical considerations have changed the process such that it is no longer the same as the original process. We will corroborate this using a hypothesis test in Section 5.4. First we will ensure that our calibrated parameters have been correctly identified by using the multivariate random time change theorem proposed by Bowsher (2007).

5.3.4 Validation

Using the result from Theorem 1 and by computing Equation 2.16 with $\hat{\theta}$ from the calibration, we can perform statistical tests to determine whether the generalised residuals are in fact independent and exponentially distributed with unit rate (see Section 10). To this end, we first perform a simple Q-Q (quantile-quantile) plot for visual inspection.

We then perform the Kolmogorov-Smirnov test which is a non-parametric test based on the maximal discrepancy between the empirical cumulative distribution and the exponential cumulative distribution with unit rate. An *acceptance* of the null hypothesis H_0 for this test indicates that the hypothesis that the generalised residuals are exponentially distributed with unit rate cannot be rejected.

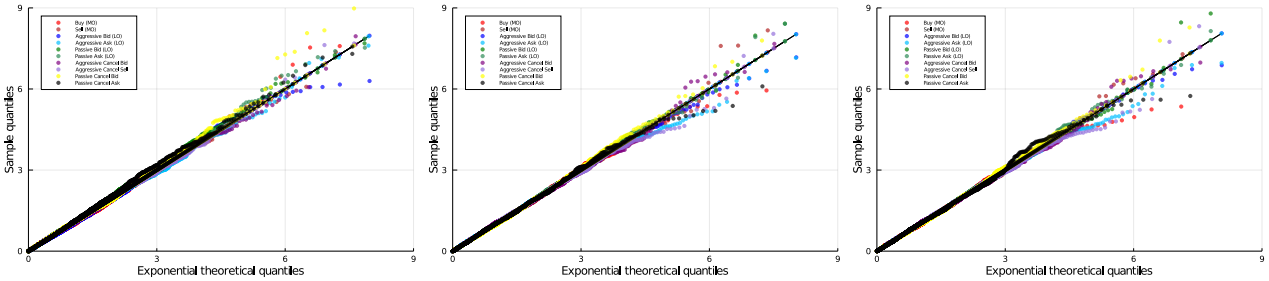
Finally, the Ljung-Box test can be used to test the independence. The test examines the null hypothesis of an absence of auto-correlation in a given time-series. For significance level α, the critical region for rejection of the hypothesis of randomness is

$$Q = n(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k} > \chi^2_{1-\alpha,h} \tag{5.4}$$

where n is the sample size, $\hat{\rho}_k$ is the sample auto-correlation at lag k, and Q asymptotically follows a χ^2_h distribution. An *acceptance* of the null hypothesis H_0 for this test indicates that hypothesis that the generalised residuals are serially independent cannot be rejected.

Figure 5.2 plots the Q-Q plot of the generalised residuals for each of the event types against the theoretical quantiles from an exponential with unit rate. From left to right, we have the generalised residuals from the reference model, model 1 and model 2 respectively. From a visual perspective, we see that the generalised residuals for all three calibrations seem to follow the quantiles from the theoretical exponential.

[9] $\text{RMSE} = \sqrt{\text{MSE}}$ and $\text{MSE}(\hat{\theta}, \theta_{\text{true}}) = \frac{1}{210}\sum_{k=1}^{210}\left(\theta_{\text{true},k} - \hat{\theta}_k\right)^2$.



(A) Hawkes (B) Model 1 (C) Model 2

FIGURE 5.2: As a qualitative measure of goodness-of-fit, generalised residuals for each event type (obtained from Equation 2.16) are compared against the quantiles of an Exponential distribution with unit rate.

For more quantitative results, Table 5.4 reports the p-values from the Ljung-Box and the Kolmogorov-Smirnov test for the generalised residuals from each event type for each calibration. We see that all of the $p-$values (except the Ljung-Box test on event type 7 from model 2) are larger than a 10% threshold. Therefore the various null hypotheses H_0 that our generalised residuals are independent and exponentially distributed with unit mean cannot be rejected.

	Hawkes		Model 1		Model 2	
Type number	Ljung -Box	Kolmogorov -Smirnov	Ljung -Box	Kolmogorov -Smirnov	Ljung -Box	Kolmogorov -Smirnov
1	0.116	0.5279	0.32051	0.87332	0.96014	0.5663
2	0.455	0.23593	0.57889	0.69011	0.98565	0.90626
3	0.51286	0.13925	0.86472	0.75525	0.41731	0.97167
4	0.14084	0.63515	0.51574	0.15786	0.85563	0.72569
5	0.54962	0.55145	0.73988	0.89928	0.85563	0.94349
6	0.64359	0.20044	0.3429	0.83269	0.58501	0.64426
7	0.16513	0.75415	0.81173	0.34173	0.00777	0.57761
8	0.92577	0.39957	0.49469	0.53315	0.59695	0.67609
9	0.1144	0.88623	0.5002	0.60972	0.90512	0.65956
10	0.56949	0.73368	0.33559	0.69256	0.39552	0.73026

TABLE 5.4: Mis-specification tests are performed by checking the null hypotheses that generalised residuals are serially independent and are exponentially distributed with unit rate. The values reported are $p-$values.

The successful application of Theorem 1 is an indication that the results obtained from the MLE are indeed true parameters for their respective data. Coupling this result along with the result from Table 5.3, it indicates that market mechanisms have indeed altered the underlying process such that it is no longer the same as the initial Hawkes process which we used to generate the various event types.

5.4 Discussion of results

5.4.1 Parameter uncertainty

We attempt to obtain confidence intervals for our parameters obtained through MLE using the result that the Fisher information can approximate the covariance matrix of the estimates.

The *score* function is the first derivative of the log-likelihood given as

$$S(\theta) = \frac{\partial}{\partial \theta} \ln \mathcal{L}(\theta). \tag{5.5}$$

From Theorem 4 of Yoshiko Ogata (1978), $\frac{1}{\sqrt{T}} S(\theta)$ converges in law to $\mathcal{N}(0, \mathcal{I}(\theta))$ as $T \to \infty$, where $\mathcal{I}(\theta)$ is the Fisher information. The Fisher information is therefore the variance of the score. Moreover, from Theorem 1 of Yoshiko Ogata (1978), the Fisher information can be computed as

$$[\mathcal{I}(\theta)]_{i,j} = \mathbb{E}\left[\frac{\partial \ln \mathcal{L}(\theta)}{\partial \theta_i} \frac{\partial \ln \mathcal{L}(\theta)}{\partial \theta_j}\right] = -\mathbb{E}\left[\frac{\partial^2 \ln \mathcal{L}(\theta)}{\partial \theta_i \partial \theta_j}\right], \quad i, j = 1, \ldots 210. \tag{5.6}$$

Instead of computing the Fisher information, we use the observed Fisher information given as

$$[I(\theta)]_{i,j} = -\frac{\partial^2 \ln \mathcal{L}(\theta)}{\partial \theta_i \partial \theta_j}. \tag{5.7}$$

We make this choice because Equation 5.6 is usually computed analytically or using Monte Carlo methods which is not feasible for the size of our problem. Additionally, Efron and Hinkley (1978) favour the use of the observed Fisher information over the Fisher information. Therefore, our observed Fisher information is simply the negative Hessian of the log-likelihood evaluated at the maximum likelihood estimate which we compute numerically using the package "ForwardDiff" (Revels, Lubin, and Papamarkou 2016). Combining the results, we have a 95% asymptotic confidence interval for each θ_i given by

$$\hat{\theta}_i \pm 1.96 \sqrt{I^{-1}_{i,i}\left(\hat{\theta}\right)/T}, \tag{5.8}$$

subject to a variety of conditions. This is the same approach Large (2007) used when constructing confidence intervals.

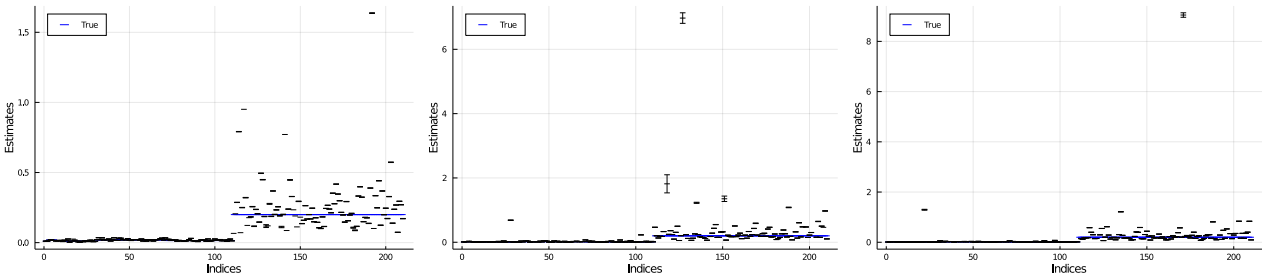
Interestingly, we found that 5 and 4 of our variance estimates were negative for model 1 and model 2 respectively. This was not a problem for the reference model. Freedman (2007) mentioned that the observed information can generate negative variances estimates when maximum likelihood estimates are obtained on a restricted parameter space imposed by the null hypothesis. However, our calibration routine was performed on the unrestricted space and our validation indeed points towards the fact that we have obtained appropriate parameters for the respective data. Morgan, Palmer, and Ridout (2007) demonstrated something interesting, which is that for many datasets, the observed information can lead to negative variance estimates as pointed out by Verbeke and Molenberghs (2007).

Looking further into this issue we found that the cause for our negative variance estimates was because the observed information was not positive definite. We had small negative eigenvalues. This result goes against Theorem 3 of Yoshiko Ogata (1978) where our observed information should have been positive definite. As of now, it is unclear which assumptions from the Theorem are violated. However, we suspect that the issue may be numerical errors or we have insufficient observations for certain event types (due to events being dropped) leading to a bad likelihood. It is for this reason that we do not consider hypothesis tests using the Score or Wald test. Since our observed Fisher information generates negative variances, the test will be inconsistent as pointed out by Freedman (2007).

Nonetheless, we visualise the estimated parameters along with their confidence intervals from Equation 5.8 in Figure 5.3. To this end, we plot the estimated parameter and its confidence interval for

$$\hat{\theta} = \left(\hat{\mu}^1, \ldots, \hat{\mu}^{10}, \hat{\alpha}^{1,1}, \ldots, \hat{\alpha}^{1,10}, \hat{\alpha}^{2,1}, \ldots, \hat{\alpha}^{2,10}, \ldots, \hat{\alpha}^{10,1}, \ldots, \hat{\alpha}^{10,10}, \hat{\beta}^{1,1}, \ldots, \hat{\beta}^{1,10}, \hat{\beta}^{2,1}, \ldots, \hat{\beta}^{2,10}, \ldots, \hat{\beta}^{10,1}, \hat{\beta}^{10,10}\right)$$

For parameters with a negative variance estimate, the confidence interval is excluded and only the estimate is visualised. Additionally, we exclude both the parameter value and confidence intervals for the parameters with extremely large deviation for better readability.



(A) Hawkes reference model (B) Model 1 (C) Model 2

FIGURE 5.3: Confidence intervals for parameter estimates for the three models computed from Equation 5.8. Outliers are removed for improved readability with the largest parameter values being those that were distorted the most in Figures 5.6 and 5.7. The top most data parameters from the reference model, model 1 and model 2 are 1.637 ± 0.000109, 6.972 ± 0.00119 and 9.049 ± 0.00154 respectively.

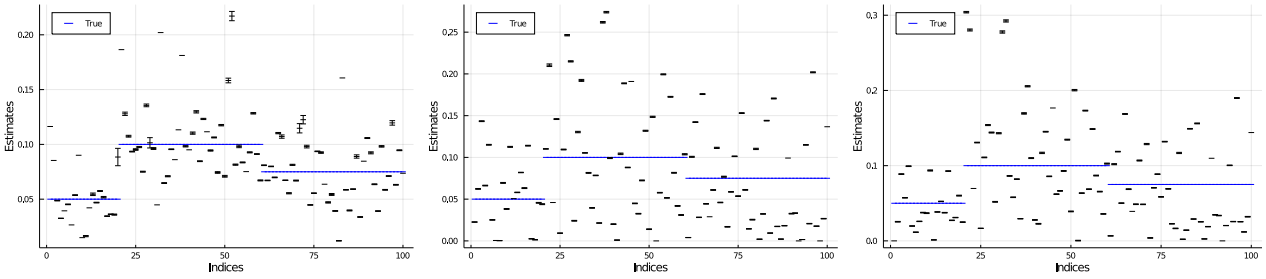
We also include estimates and confidence intervals for the branching ratios. The variance for the branching ratio is obtained using the delta method for error propagation. We need to obtain the variance for $\Gamma^{nm} = f(\alpha^{nm}, \beta^{nm}) = \alpha^{nm}/\beta^{nm}$ but we only have variances and covariances estimates for α^{nm} and β^{nm} (from the observed Fisher information). Therefore, we can approximate the variance of Γ^{nm} as

$$\operatorname{Var}\Gamma^{nm} \approx \left(\frac{\partial f}{\partial \alpha^{nm}}\right)^2 \operatorname{Var}\alpha^{nm} + \left(\frac{\partial f}{\partial \beta^{nm}}\right)^2 \operatorname{Var}\beta^{nm} + \left(\frac{\partial f}{\partial \alpha^{nm}}\right)\left(\frac{\partial f}{\partial \beta^{nm}}\right) 2\operatorname{Cov}(\alpha^{nm}, \beta^{nm}),$$

where $\frac{\partial f}{\partial \alpha^{nm}} = 1/\hat{\beta}^{nm}$ and $\frac{\partial f}{\partial \beta^{nm}} = -\hat{\alpha}^{nm}/\left(\hat{\beta}^{nm}\right)^2$. The resulting estimates and confidence intervals can be found in Figure 5.4. The indices are ordered as

$$\hat{\Gamma} = \left(\hat{\Gamma}^{1,1}, \ldots, \hat{\Gamma}^{1,10}, \hat{\Gamma}^{2,1}, \ldots, \hat{\Gamma}^{2,10}, \ldots, \hat{\Gamma}^{10,1}, \ldots, \hat{\Gamma}^{10,10}\right).$$

Finally, confidence intervals are excluded for Γ^{nm} which ended up with a negative variances or if α^{nm} and β^{nm} had negative variances.

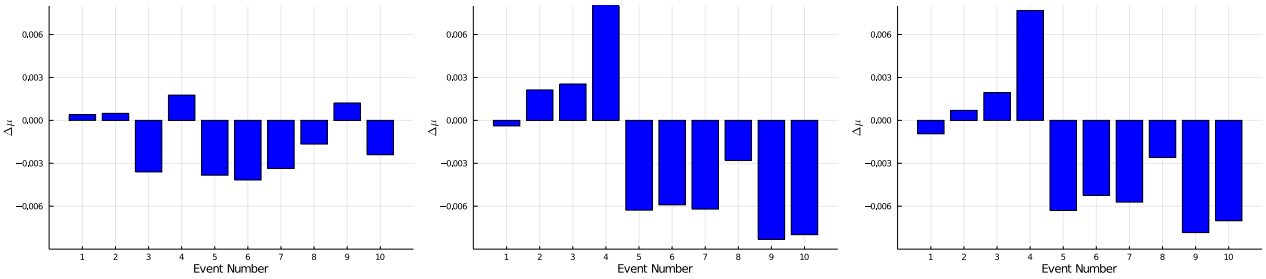


(A) Hawkes reference model (B) Model 1 (C) Model 2

FIGURE 5.4: Confidence intervals for branching ratios computed using the delta method for error propagation. Certain ratios exhibit larger intervals than others. Model 1 and 2 maintained a good balance of expected average event numbers but their branching ratios did not conform to that of the original process.

5.4.2 Parameter distortions

We visualise the deviation between parameters from the calibration against the true parameters from the underlying process. First, the deviation between the baseline intensities $\Delta\boldsymbol{\mu} = \hat{\boldsymbol{\mu}} - \boldsymbol{\mu}_{\text{true}}$ are visualised using bar plots in Figure 5.5. We see that the deviations are not particularly large.



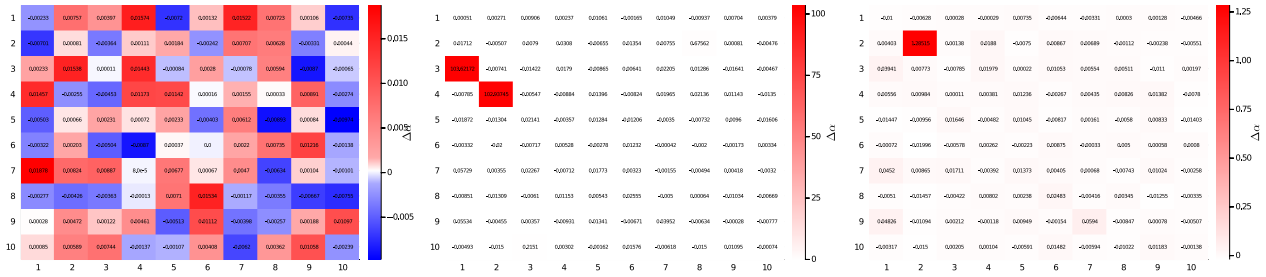
(A) $\boldsymbol{\mu}$ distortions: Reference model (B) $\boldsymbol{\mu}$ distortions: Model 1 (C) $\boldsymbol{\mu}$ distortions: Model 2

FIGURE 5.5: Parameter distortions in the baseline intensity $\Delta\boldsymbol{\mu}$ between the calibrated parameters and the true parameters from the underlying process. The baseline intensities in model 1 and model 2 are found to have similar distortion magnitudes. The largest of these came from aggressive asks (event 4), passive LOs (events 5 and 6) and passive cancels (events 9 and 10).

Second, the distortion between $\Delta\boldsymbol{\alpha} = \hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha}_{\text{true}}$ and $\Delta\boldsymbol{\beta} = \hat{\boldsymbol{\beta}} - \boldsymbol{\beta}_{\text{true}}$ are visualised using heatmaps in figures 5.6 and 5.7 respectively. The deviations in the reference model are minimal whereas the deviations in model 1 and 2 are relatively large. We will highlight and discuss a few notable changes that can be explained through the various market mechanics.

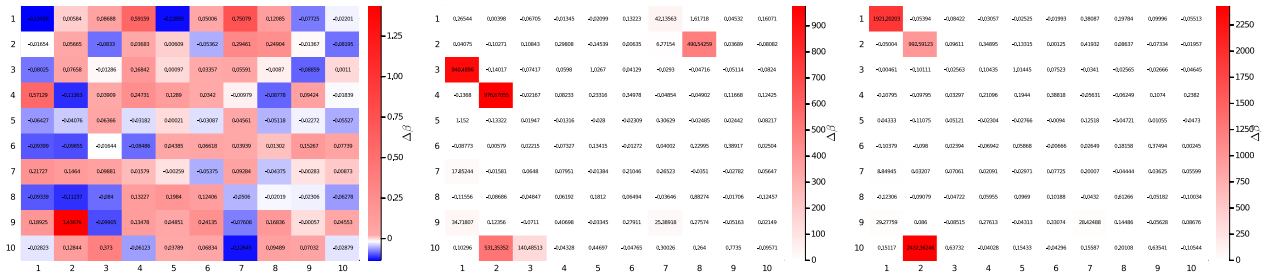
Looking at $\alpha^{3,1}, \alpha^{4,2}, \beta^{3,1}$ and $\beta^{4,2}$ for model 1, these deviations can be attributed towards the additional spawned event caused by aggressive limit orders crossing the book. This is because when an aggressive limit order crosses the book, it first becomes a market order and if it has a larger volume than the contra side then an additional limit order will be spawned as discussed in Section 5.3.2. Therefore, to capture an aggressive limit order that spawns immediately after a market order (which was converted), $\alpha^{3,1}$ and $\alpha^{4,2}$ has to significantly increase to ensure that an event is spawned immediately and the event will become an aggressive limit order (from the same side as the market order). However, this increased intensity should only last for a very short while since only one event is spawned. Thus to accommodate this, the rate of decay in $\beta^{3,1}$ and $\beta^{4,2}$ must also significantly increase to avoid spawning too many additional events.

Looking at $\beta^{1,1}$ and $\beta^{2,2}$ for model 2, these deviations can be attributed towards the dropping of events. This is because too many consecutive market orders coming through from the same side will deplete liquidity on the contra side and thus emptying the LOB. Once the contra side of the LOB is empty, any additional market orders will not go through. Therefore to capture this, the rate of decay must increase to capture the dropped orders.



(A) $\boldsymbol{\alpha}$ distortions: Reference model (B) $\boldsymbol{\alpha}$ distortions: Model 1 (C) $\boldsymbol{\alpha}$ distortions: Model 2

FIGURE 5.6: Parameter distortions in the excitation matrix $\Delta\boldsymbol{\alpha}$ between the calibrated parameters and the true parameters from the underlying process.

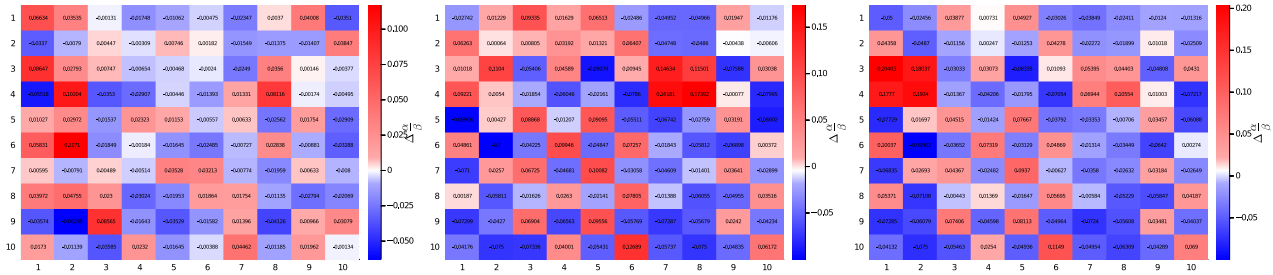


(A) $\boldsymbol{\beta}$ distortions: Reference model (B) $\boldsymbol{\beta}$ distortions: Model 1 (C) $\boldsymbol{\beta}$ distortions: Model 2

FIGURE 5.7: Parameter distortions in the decay matrix $\Delta\boldsymbol{\beta}$ between the calibrated parameters and the true parameters from the underlying process.

Next, the distortion of the branching ratios $\Delta\boldsymbol{\Gamma} = \hat{\boldsymbol{\Gamma}} - \boldsymbol{\Gamma}_{\text{true}}$ are visualised in Figure 5.8. In model 1 the events whose branching ratios are most significantly increased are that of aggressive cancels that spawn aggressive LOs on both sides of the LOB (the effect of events 7 and 8 on events 3 and 4). The reason for the increase in expected number of events in this case is that cancel orders often empty the LOB, resulting in any LO occuring thereafter being classified as aggressive irrespective of whether the original order was passive or not.

Similarly, for model 2 the most significant increase in branching ratios occur for market orders that spawn aggressive limit orders. This can be explained by the fact that trades emptying the LOB will cause limit orders occurring thereafter to always become aggressive — thus resulting in greater numbers of offspring from those events.



(A) $\boldsymbol{\Gamma}$ distortions: Reference model (B) $\boldsymbol{\Gamma}$ distortions: Model 1 (C) $\boldsymbol{\Gamma}$ distortions: Model 2

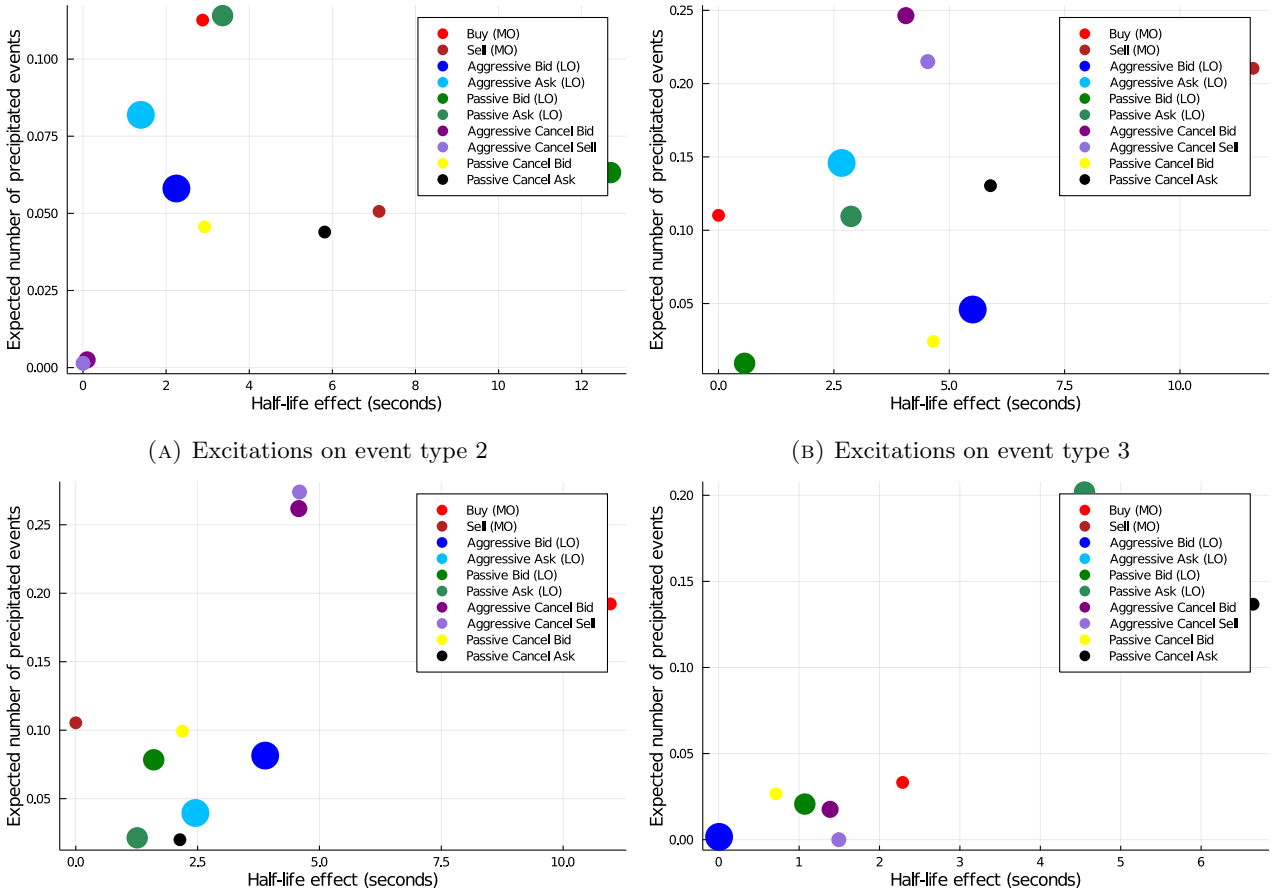
FIGURE 5.8: Branching ratio distortions $\Delta\boldsymbol{\Gamma}$ between the calibrated parameters and the true parameters from the underlying process. Despite the many distortions, the expected average number of events remain fairly well balanced.

Finally, Figures 5.9 and 5.10 investigates the dynamics of individual excitation effects on event types with some severely distorted parameters found in Figures 5.6 and 5.7. Each marker represents an individual effect with its branching ratio measured on the y-axis and its half-life[10] (seconds) measured on the x-axis. The diameter of the markers are proportional to the excitation effect's event count. To obtain the half-life of a given intensity effect α^{mn} we need to solve

$$\exp\left(-t_{1/2}\beta^{mn}\right) = 1/2 \quad \text{to get} \quad t_{1/2} = \log(2)/\beta^{mn}. \tag{5.9}$$

Branching ratios have the same interpretation as before, it is the average number of events of type m caused by a single event of type n. The half-lives give us a sense of how long a particular excitation will last for. Moving through each figure one gets a sense for how each event's intensities are affected by all the others.

[10]The half-life is the amount of time it takes for a given intensity effect α^{mn} to reduce by a half



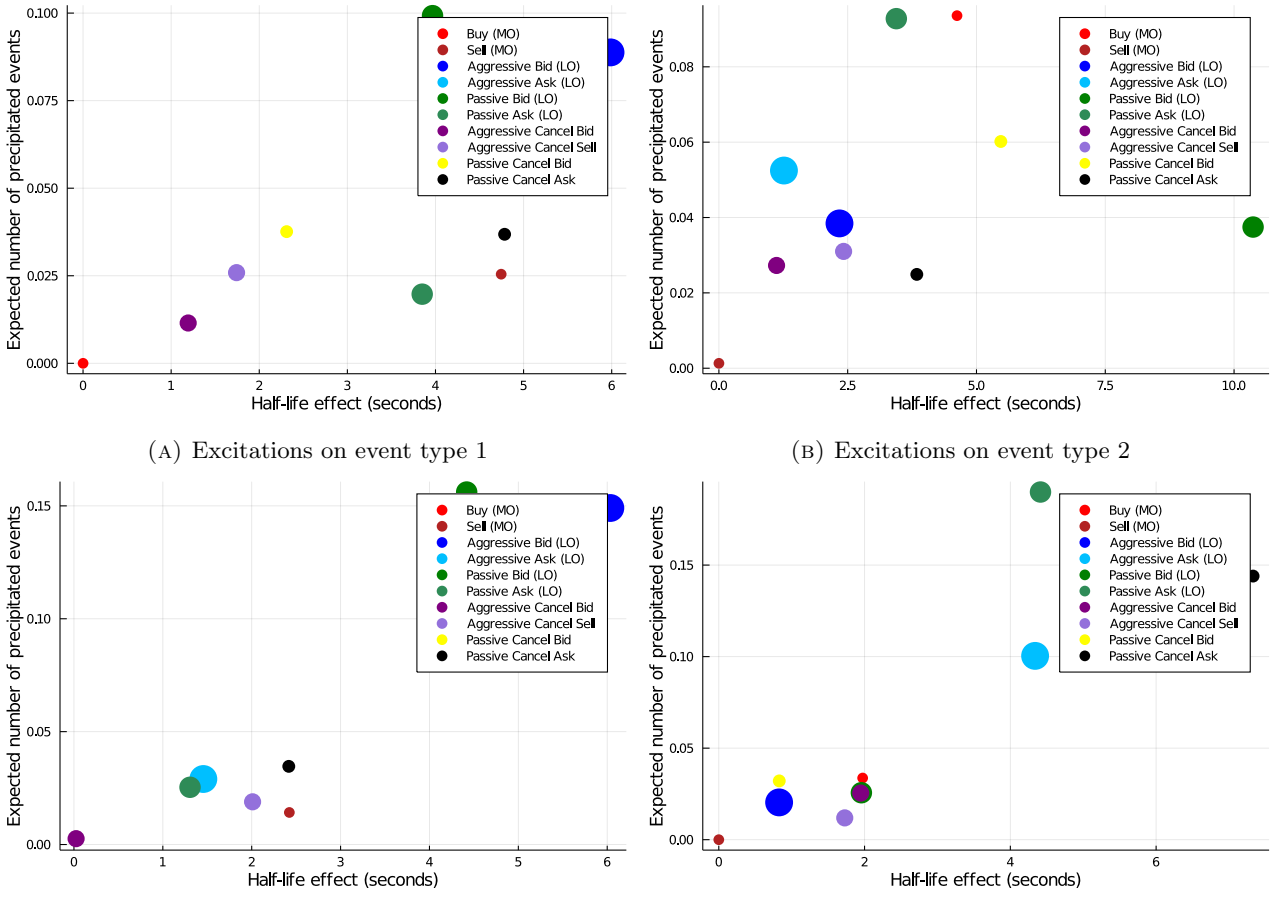
(A) Excitations on event type 2

(B) Excitations on event type 3

(C) Excitations on event type 4

(D) Excitations on event type 10

FIGURE 5.9: Model 1 bubble plot for event types with large distortions. The size of the markers are proportional to the number of events of each type. The expected number of precipitated events are given by branching ratios of each type while the half-lives are calculated as in Equation 5.9. Aggressive cancels had the smallest half-life effect on sell MOs while passive bids had the greatest (Figure 5.9a). Sell MOs had the greatest half-life effect on aggressive bids (Figure 5.9b) while buy MOs had the greatest half-life effect on aggressive asks (Figure 5.9c). Lastly, passive cancels of sell orders had the largest half life effect on itself with aggressive bids having the smallest (Figure 5.9d).



(A) Excitations on event type 1

(B) Excitations on event type 2

(C) Excitations on event type 9

(D) Excitations on event type 10

FIGURE 5.10: Model 2 bubble plot for event types with large distortions. The size of the markers are proportional to the number of events of each type. The expected number of precipitated events are given by the branching ratios of each type while the half-lives are calculated as in Equation 5.9. Buy MOs had the smallest half-life effect on itself while aggressive bids had the greatest (Figure 5.10a). Similarly, sell MOs had the smallest half-life effect on itself while passive bids had the greatest (Figure 5.10b). Aggressive cancels of bids had the smallest half-life-effect on passive cancels of bids with aggressive bids having the greatest (Figure 5.10c). Lastly, passive cancels of sell orders had the largest half life effect on itself with buy MOs having the smallest (Figure 5.10d).

5.4.3 Hypothesis tests

In order to statistically determine if market mechanisms have altered the underlying process, we can perform a simple vs composite hypothesis test using the likelihood-ratio test also known as Wilks test (Wilks 1938).

We can do this by testing

$$H_0 : \theta = \theta_{\text{true}} \quad \text{vs} \quad H_1 : \theta \in \Theta, \tag{5.10}$$

where Θ is the full unrestricted parameter space. Therefore, the test statistic for the hypothesis test is given by

$$\lambda_{\text{LR}} = -2 \ln \left[\frac{\mathcal{L}(\theta_{\text{true}})}{\sup_{\theta \in \Theta} \mathcal{L}(\theta)} \right],$$

which reduces to

$$\lambda_{\text{LR}} = -2 \left[\ln \mathcal{L}(\theta_{\text{true}}) - \ln \mathcal{L}\left(\hat{\theta}\right) \right], \tag{5.11}$$

where $\hat{\theta}$ are the estimates obtained through MLE. The test statistic in Equation 5.11, subject to regularity, will be approximately distributed as χ^2_{210} assuming the null hypothesis is true (Silvey 1975, p. 112–114).

Model	λ_{LR}	p-value
Hawkes	235.50	0.109457
Model 1	4636.38	0
Model 2	1197.95	0

TABLE 5.5: Test statistic and p-values for the likelihood-ratio test of Equation 5.10.

Table 5.5 reports the test statistic and p-values for the likelihood-ratio test on the reference model and on model 1 and 2. The results from the reference model indicate that it remains probable that the observed data came from θ_{true}. However, the results from model 1 and 2 indicate that is essentially impossible that the observed data came from θ_{true}.

This is expected since the reference model is simply a realisation of θ_{true}, whereas model 1 and 2 are affected by market mechanisms that drop events, change event types and spawn additional events. These results confirm that market design and its mechanisms can indeed affect our ability to infer the underlying process.

5.4.4 Implications

We have demonstrated that market mechanics and practical constraints can significantly cloud our ability to identify the true process used to generate orders when the given representation does not fit the interplay between orders and the matching engine. When this is the case, the process is changed in a statistically meaningful way by the matching process. The changes to the resulting orders and trades due to the market mechanics were significant enough, according to a likelihood-ratio test, to make it is practically impossible to conclude that the resulting data came from the same process used to generate the data. It may be argued that the clouding of the order generating process by a matching engine is trivial. However, this is not the case. We argue that it suggests that the data generating process may not always be faithfully represented using the model's framework itself.

These results suggest that care should be taken when applying alternative market models, such as agent-based models, to continuous double auctions. Specifically, that agent behaviour can also be dominated by practical considerations rather than strategic interactions alone. Therefore, models need to be able to account for this and cannot be independent of the market rules and mechanisms.

For the example that we used, this can possibly be ameliorated by using a constrained Hawkes process similar to what was done by Zheng, Roueff, and Abergel (2014). We can also introduce new processes to keep track of the spread and the number of orders on each side of the LOB. These processes can inform us which intensities to turn off so as to ensure that event types which would otherwise violate practical considerations and market rules do not occur. Ideally, these types of additional complexities should arise naturally from a model specification rather than added as constraints.

The considerations that one would need to account for will depend on the model and the set of actions that can be taken. Moreover, the considerations will also depend on the specific market and their specific rules. For example, in 2003, Chinese stock markets only allowed limit orders to be submitted and there were no "true" market orders (Zhou 2012). Therefore, the considerations will need to be tailored for the model and its application.

This does not detract from the value that a point-process model has in testing trading infrastructure as it can provide fast exploratory data analytics. The work here successfully demonstrates this use with the CoinTossX matching engine on test cases that have reasonable realism in terms of the volume and intensity of the order flow with different order types.

5.5 Summary

We investigate whether market microstructure and the mechanics of the order book can cloud our ability to identify the underlying process when it is known, and demonstrate the ability of the CoinTossX matching engine to manage realistic orders in an expected manner. This work is an important milestone to test the matching engine, and the initial development of the model management system that will be used to generate agent-based models predicated on a continuous double auction in future work. Here we use a 10-variate Hawkes process along with simple rules to generate order messages that are injected into a matching engine for order matching and limit order book aggregation and management.

From the simulation we compare the calibrated parameters for a Hawkes reference model (Section 5.2.1) with two additional models (Sections 5.2.2 and 5.2.2) with modified implementation rules against the true model parameters (see Table 5.3 in Section 5.3.3). We find that the calibrated parameters from the models deviate significantly from the true parameters (see Section 5.3.3). The calibration is then validated using the multivariate random time change theorem to ensure that the calibrated parameters are correct for the data obtained from the simulation (see Section 5.3.4). A likelihood-ratio test is then performed to see if the resulting observations could have been an extreme sample from the true process (see Section 5.4.3 and Table 5.5).

The results indicate that market mechanisms and practical considerations have significantly altered the process and can indeed affect our ability to infer the underlying process, as evidenced in Figures 5.6 and 5.7. Our findings suggest that when considering market models, such as agent-based models at least in the setting of continuous double auctions, it is prudent to account for practical considerations imposed by market rules, mechanism and asynchronicity. It may be inadequate to argue their impacts away using additional sources of noise or systematic model parameter uncertainties.

From a pragmatic perspective this suggests that focusing the modelling on event times separate from the event characteristics can lead to ambiguities in model interpretation. This suggests that more realistic market models need to combine event time models directly with processes that generate their sizes, order types and interactions between orders and their environment. These findings bring us to Chapter 6 — the implementation of a close-to event-based financial agent-based model with CoinTossX.

6 Agent-based model

An agent-based model with interacting low frequency liquidity takers inter-mediated by high-frequency liquidity providers acting collectively as market makers can be used to provide realistic simulated price impact curves. This is possible when agent-based model interactions occur asynchronously via order matching using a matching engine in event time to replace sequential calendar time market clearing. Here the matching engine infrastructure has been modified to provide a continuous feed of order confirmations and updates as message streams in order to conform more closely to live trading environments. The resulting trade and quote message data from the simulations are then aggregated, calibrated and visualised. Various stylised facts are presented along with event visualisations and price impact curves. We argue that additional realism in modelling can be achieved with a small set of agent parameters and simple interaction rules once interactions are reactive, asynchronous and in event time. We argue that the reactive nature of market agents may be a fundamental property of financial markets and when accounted for can allow for parsimonious modelling without recourse to additional sources of noise.

6.1 Introduction

The pairing of agent-based models with a matching engine in an artificial stock market framework that mimics the real-world work-flow and system implementation offers a controlled environment for exploring market complexity without forcing the market structure and constraints into the domain specification of agent interaction rules.

Here we replicate real equity markets at the level of the model components' design as well as at the model's output level. The recovery of the market orders' price impact being a crucial component (Farmer, Patelli, and Zovko 2005; Mandeş 2015). Farmer, Patelli, and Zovko (2005) argued that with the zero-intelligence approach one may be able to find macroscopic averaged laws, not dissimilar to those used in thermodynamics, that can capture universal relationships between various macroscopic averaged quantities derived from microscopic interactions, such as the average diffusion rates, spreads and so on. Although a zero-intelligence approach is desirable, there is some motivation for considering perfectly rational or at least boundedly rational interacting agents, as in this case. Here microscopic rules are directly related to macroscopic properties via simulating a large number of agent interactions. The motivation is that some empirical features may not be straightforward to achieve without agent intelligence or strategic behaviour, or at least without detailed external assumptions about empirical facts and the distributional assumptions that may be needed to encode the macroscopic results from strategic agent interactions.

In the financial ABM literature, significant attention has been devoted to the calibration of models involving closed-form solutions where markets are cleared at each iteration and prices are determined by a weighted average of trader expectations, *i.e.*, models with sequential market clearing which are analogous to closing auction trading sessions at the end of each day. This simplification is appropriate for mimicking the volume maximising mechanism typically used by an exchange. In contrast, intraday models such as those of Leal et al. (2016) and Preis et al. (2006) focus on recreating the double-auction dynamics, but with prices that are still formed sequentially in calendar time. We deviate here by introducing a framework for semi-asynchronous order submission and matching by pairing an ABM with a matching engine. We consider this an important step towards embracing the event time realities of financial markets at the level of the model design.

6.2 Agent specifications

There are several problems to overcome when shifting from sequential market clearing to a truly continuous double auction implementation. First, the agent rules can no longer depend on clearing prices from homogeneous time steps. The rules must be changed to an asynchronous framework where the agents decisions should depend on variables at any given time t. Second, there needs to be sufficient liquidity present, or at least interaction mechanisms in place to provide orders on both sides of the limit order book at any given time for the market to function. This is not required when applying sequential market clearing, but it is a necessary requirement in a continuous double auction.

Market makers usually fulfil the role of providing liquidity. The hallmark of market making is that they do not deliberately place directional bets but rather aim to profit off the spread while minimising directional risk (Chakraborty and Kearns 2011). The problem with market making models such as the Glosten and Milgrom (1985) model or more recent formulations into stochastic control problems (Avellaneda and Stoikov 2008; Cartea, Jaimungal, and Penalva 2015) is that the focus is only on the optimal placement of limit orders. These formulations do not guide the choice of the size of the order, nor do they cater for the placement of orders at multiple levels. Additionally, the formulation in a stochastic control problem setting does not fit easily within an agent-based model framework. Therefore, we avoid using market maker algorithms and follow Leal et al. (2016) who used multiple high-frequency agents to mimic the behaviour of market makers in collective.

Concretely, our agent design largely follows Leal et al. (2016) with several modifications to account for our continuous double auction setup. We also incorporate ideas from Preis et al. (2006) and Mandeş (2015) to create more realistic agent behaviours. The market consists of N_{LT} Liquidity Takers (LT) that only submit market orders and N_{LP} Liquidity Providers (LP) that only submit limit orders. The ratio of liquidity providers to liquidity takers is N:

$$N = \frac{N_{LP}}{N_{LT}}. \tag{6.1}$$

The volume of each agent's order will be sampled according to a power law distribution with a density function given as:

$$f(x) = \begin{cases} \frac{\alpha x_{\mathrm{m}}^{\alpha}}{x^{\alpha+1}} & \text{if } x \geq x_{\mathrm{m}} \\ 0 & \text{if } x < x_{\mathrm{m}}. \end{cases} \tag{6.2}$$

Here the parameters x_{m} and α will depend on the state of the LOB and activation rules. Mandeş (2015) samples the volume size according to an independent log-normal. However, from an empirical perspective, the unconditional distribution of the order size follows a power-law behaviour (Chakraborti et al. 2011a). For the power law distribution, x_{m} is the minimum size of the volume order and a small α generates larger samples while a larger α generates smaller samples. These properties will become important when we couple the parameters to the state of the LOB and activation rules to achieve our desired agent behaviour.

6.2.1 Liquidity takers

We have two types of LT agents: "fundamentalists" and "chartists" (or trend-following agents) with a population of $i = 1, \ldots, N_{LT}^{f}$ and $i = 1, \ldots, N_{LT}^{c}$ respectively where the total population of liquidity takers is: $N_{LT} = N_{LT}^{f} + N_{LT}^{c}$. Each agent makes a decision to either buy or sell according to a random inter-arrival time governed by a truncated exponential distribution with mean intensity λ^{T} where the intensity is bounded between $\lambda_{\min}^{T}$ and $\lambda_{\max}^{T}$ with time measured in seconds.

Each fundamentalist agents decision to buy (or sell) is given as

$$D_{it}^{f} = \begin{cases} \text{sell} & \text{if } f_i < m(t) \\ \text{buy} & \text{if } f_i > m(t). \end{cases} \tag{6.3}$$

Here, f_i is the fundamental value for the ith agent. The fundamental value for each of the fundamentalist is unique and fixed for the given trading day, specifically for that day's trading session. It is computed as

$$f_i = m_0 e^{x_i} \quad \text{where} \quad x_i \sim \mathcal{N}\left(0, \sigma^2\right). \tag{6.4}$$

This is different from Leal et al. (2016) where all agents have the same fundamental value at time t and the fundamental value fluctuates between trading sessions.

Each chartist agents decision to buy (or sell) is given by:

$$D_{it}^{c} = \begin{cases} \text{sell} & \text{if } m(t) < \bar{m}_i(t) \\ \text{buy} & \text{if } m(t) > \bar{m}_i(t), \end{cases} \tag{6.5}$$

where $\bar{m}_i(t)$ is the exponential moving average (EMA) of the mid-price for the ith agent at time t. The EMA for each chartist is unique to themselves and is computed each time they make a decision. The moving average is updated according to $\bar{m}_i(t) = \bar{m}_i(t') + \lambda\left(m(t) - \bar{m}_i(t')\right)$ where t' is the time point where the ith agent made their last decision and $\lambda = 1 - e^{-\Delta t/\tau}$. Here $\Delta t = t - t'$ is the inter-arrival of the ith agent's current decision and previous decision and τ is the time constant for the ith agent computed as the mean inter-arrival of the agent's decision time. This allows our agents to base their decisions on variables at time t.

To create the market orders for LF agents to submit, we need to know both the direction of the order and the quantity of the order. The direction of the order is determined by Equations 6.3 and 6.5, and the quantity of the order is sampled according to Equation 6.2 with parameters determined according the magnitude of the activation rule and the state of the order book.

The lower bound x_{m} of the volume size is determined by the deviation between the agent's valuation or trading strategy relative to the current mid-price. We set

$$x_{\mathrm{m}} = \begin{cases} 20 & \text{if } |f_{it} - m(t)| \leq \delta m(t) \\ 50 & \text{if } |f_{it} - m(t)| > \delta m(t), \end{cases} \tag{6.6}$$

for fundamentalists, and

$$x_{\mathrm{m}} = \begin{cases} 20 & \text{if } |m(t) - \bar{m}_i(t)| \leq \delta m(t) \\ 50 & \text{if } |m(t) - \bar{m}_i(t)| > \delta m(t), \end{cases} \tag{6.7}$$

for chartists. Here the cutoff is a δ percentage of the current mid-price. This setup increases the aggression of the agent depending on how favourable the current price is relative to their valuation or trading strategy.

The second factor that determines the size of an agent's volume is the order book imbalance $\rho(t)$ (Definition 10). The shape parameter α is then:

$$\alpha = \begin{cases} 1 - \rho(t)/\nu & \text{for sell MO} \\ 1 + \rho(t)/\nu & \text{for buy MO}, \end{cases} \tag{6.8}$$

where $\nu > 1$ to ensure that α is never equal to zero. The setup achieves two effects. First, if the contra side of the order book is thicker then an agent will submit larger orders to take advantage of the available liquidity. Second, if the same side of the order book is thicker then they will submit a smaller order to avoid excessive price impact.

6.2.2 Liquidity providers

Each LP agent makes a decision to place either a bid or ask limit order according to an random inter-arrival time governed by a truncated exponential distribution with mean intensity λ^P that is bounded between $\lambda^P_{\min}$ and $\lambda^P_{\max}$ seconds. At each decision point the agent places either a bid or ask depending on the current order book imbalance $\rho(t)$ at time t. An agent has a probability θ of placing

an ask, and probability of $1-\theta$ for the placing of a bid where $\theta = \frac{1}{2}(\rho(t)+1)$. This design ensures that liquidity providers will on average provide liquidity to the side with less liquidity and thus stabilise the order book.

Once an agent has made the decision on which side of the order book to place the limit order, the placement of the limit order is given as:

$$p_t = \begin{cases} b(t) + 1 + \lfloor \eta \rfloor & \text{for asks} \\ a(t) - 1 - \lfloor \eta \rfloor & \text{for bids,} \end{cases}$$

where $b(t)$ and $a(t)$ is the best bid and best ask at time t respectively (Definition 6). Here η is a random sample from a gamma distribution with a density given as

$$f(x) = \frac{\beta^k}{\Gamma(k)} x^{k-1} e^{-\beta x},$$

where k is the shape parameter and β is the rate parameter. The shape parameter k is set to be the spread $s(t)$ at time t (Definition 7), whereas the rate parameter is set as

$$\beta = \begin{cases} e^{-\rho(t)/\kappa} & \text{for asks} \\ e^{\rho(t)/\kappa} & \text{for bids,} \end{cases}$$

at time t. This means the size of η at time t will on average be

$$\bar{\eta} = k/\beta = \begin{cases} s(t)e^{\rho(t)/\kappa} & \text{for asks} \\ s(t)e^{-\rho(t)/\kappa} & \text{for bids.} \end{cases} \tag{6.9}$$

The above equation achieves several effects.

First, when the order book is relatively balanced, the order placement will on average be near the best bid or ask. Second, when the order book is thicker on the contra side of the order book then the limit price placement will be more passive and will on average be placed further away from the best bid price to capture the strategic effect (Mandeş 2015). Lastly, when the order book is thicker on the same side of the order book then the limit price placement will be more aggressive and will on average be placed between the spread to capture the competition effect (Mandeş 2015). Here $\kappa > 0$ is a parameter that intensifies or relaxes the strategic or competition effect. When κ is small the effects are intensified and when κ is large the effects are relaxed. All limit orders have a time in force of γ seconds.

Remark 1. Initially we sampled η from an exponential distribution following Preis et al. (2006) but with a mean of $\bar{\eta}$ from Equation 6.9. However, we found that the exponential distribution resulted in the spread closing too quickly even though the sample will on average be placed at the best bid/ask (provided $\rho(t) \approx 0$). This is because the exponential distribution has more mass below the mean so that the limit price placement is more likely to be between the spread. This will reduce the spread for the next limit price placement and results in a feedback loop that closes the spread. The gamma distribution slows down the collapse which gives liquidity takers a chance to push back the spread.

The volume of each limit order is sampled according to Equation 6.2 with the lower bound of the volume size as $x_n = 10$ and the shape parameter as

$$\alpha = \begin{cases} 1 - \rho(t)/\nu & \text{for asks} \\ 1 + \rho(t)/\nu & \text{for bids.} \end{cases} \tag{6.10}$$

This setup achieves two effects. First, when the order book is thicker on the contra side of the order then the volume of the limit order will be large. Second, when the order book is thicker on the same side as the order then the volume of the limit order will be small. The motivation behind this choice was to ensure that imbalances are swiftly addressed to ensure that we have a stable order book.

6.3 System implementation

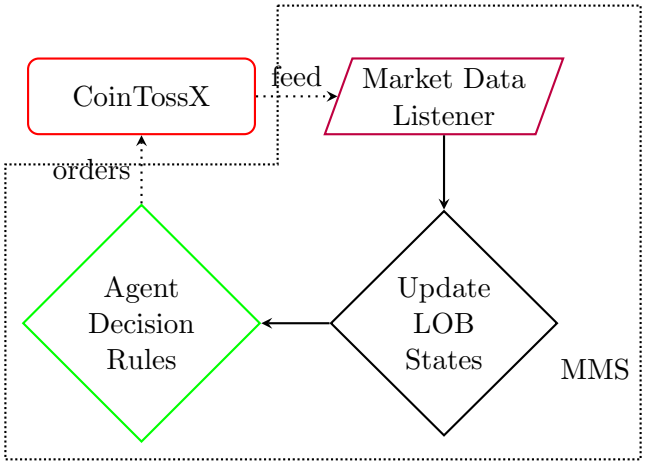


FIGURE 6.1: Schematic representation of the separation of the Model Management System (MMS) which replaces the functionality of the Execution Management Systems (EMS) and the Order Management System (OMS) in a conventional trading environment. The Matching Engine (ME) is implemented using CoinTossX (Sing 2017). The ME is loosely coupled to the MMS. The MMS accesses messages from the data feed by listening to a port using a socket. Orders are injected as message strings and pushed to a port.

Figure 6.1 outlines how the various components of the model fit together. The Matching Engine (ME) is its own independent piece of software which is coupled to the Model Management System (MMS) by the data feed and submission of orders. The model management system consists of the market data listener and the agent decision rules. Here the market data listener builds its own LOB by listening to the data feed from the matching engine and updating the LOB whenever a message comes through. The updated LOB can be accessed by all the agents which then make a decision and submit their order to the matching engine.

6.3.1 Matching engine

To achieve low-coupling and high-cohesion between the Matching Engine (ME) and our agent-based model we required further development of the matching engine code and the order management and model management framework.[1] In our previous work we had a high-coupling between the matching engine and our simulation. This was because the state of the LOB was observed by requesting snapshots using a back-end functionality from the matching engine (Jericevich, Chang, and Gebbie 2021a). The back-end functionality was designed to help debug the matching engine when it was being built and tested; it was not intended to be used as part of the simulation workflow functionality.

The system has now been separated with order submissions/confirmations and executions being forwarded in real time over a network via UDP from the matching engine, *i.e.*, we have a continuous data feed.[2] These messages are then continuously captured and managed by a dedicated order management shim on the modelling framework side.

The basic procedure for submitting orders to CoinTossX is demonstrated in Pseudocode 9 of Appendix B. We will discuss how different order type submissions are specified. Instead of specifying limit order expiry durations, we cancel orders manually once their active time limit is reached. Cancellations require an order identification number, a side and a price. These fields should match the

[1] Here cohesion refers to the degree to which a part of a code base forms a logically single, atomic unit. Here coupling refers to the degree to which a single unit is independent from others, *i.e.*, it is related to the number of connections between two or more units.

[2] User Datagram Protocol (UDP) is a lightweight, connectionless (unreliable) but fast protocol where data is transmitted irrespective of whether there is a listener to receive the message or not.

limit order to be cancelled and is determined by the decision rules of the liquidity providers. Limit orders require a volume, side and price which are similarly determined by liquidity providers. Market order messages only require a side and volume. These fields are determined by the decision rules of the liquidity takers.

For the continuous data feed we encode and decode messages in a binary format such that transmission of messages over the network remains fast. This is currently a prototype for more advancements to be made in the flexibility of CoinTossX's API with regards to other programming languages.

6.3.2 Market-data listener

The market data listener is focused on listening for messages on the data feed from CoinTossX and processing these messages to build a separate LOB for the agents that is independent of the matching engine. This mirrors the workflow of real market participants who collate and manage their own representations of the limit order book for decision making. Sample messages received from CoinTossX are illustrated below.

We deviate slightly from original CoinTossX architecture in order to receive the market data from CoinTossX. Since Julia currently cannot integrate with Aeron media driver, we open another port in the matching engine for the forwarding of byte encoded messages via UDP. This is in response to issues regarding the LOB snapshot functionality previously used in Jericevich, Chang, and Gebbie (2021a). First, requesting a snapshot of the LOB is not realistic from a trading/exchange point of view. Second, in the event of a large build of of orders in the LOB, requesting it often would compromise latency. Therefore, we assign a dedicated listener to the modelling framework in Julia that will continuously wait and receive market data messages as they occur. The LOB is built with each message update as a separate process and will form the "lit order book".

In terms of the feedback from each order submission, the market data feed is straight-forward to customise and currently adopts the following format. Each report is represented by a single string which is encoded into a vector of bytes and sent via UDP to a specified port and address. The message is decoded back into a string in Julia to be processed. The reason for adopting this process is so that the transmitting speed is as fast as possible. This method also automates the process of receiving market data as opposed to manually requesting execution reports.

The snippet below provides examples of messages transmitted. The first field provides the number of milliseconds since the unix epoch at which the event(s) arrived. The second specifies the event type which will either be: "New" (limit order), 'Trade" or "Cancelled". The third field specifies the side of the order. The fourth field provides the agent's identity. More importantly, the last field is given in vector form where each sub-vector contains the ID, price and volume of the order. In the second last case where a trade walked the LOB and executed against the first two limit orders, two different sub-vectors are recorded and corresponds to the single market order that was split into two and executed at two different prices.

```
1621159225272,New,Buy,HF|1,50,100
1621159230899,New,Buy,HF|2,49,100
1621159235548,Trade,Sell,LF|1,50,100|2,49,50
1621159242341,Cancelled,Buy,HF|2,0,0
```

The LOB is created from the messages using separate dictionaries for the ask side and bid side. Each order is stored as key-value pairs where the keys are given by the order IDs assigned from CoinTossX while the value for the key contains the limit price and volume. With this setup, updating the LOB is simply a matter of adding new orders and removing or cancelling orders by cross-referencing order IDs given in transactions or cancellations (see Pseudocode 5 for maintaining the LOB). Note that the LOB built in the MMS can only keep track of the price priority and not the time priority. The full price-time priority is maintained in the matching engine. Therefore, the order ID is the most

important component to help maintain the LOB in the MMS. This reflects what happens in real financial markets.

Once the LOB has been updated we extract some important quantities upon which our agent decisions depend on, such as: the best bid $b(t)$ and ask $a(t)$, the mid-price $m(t)$, the spread $s(t)$ and the order book imbalance $\rho(t)$. It should be noted that when the bid side of the LOB is empty, the best bid price $b(t)$ retains the price of the previous best bid before the bid side was emptied as a reference price. When the ask side of the LOB is empty, $a(t)$ retains the price of the previous best ask before the ask side emptied as its reference price.

The spread and mid-price is computed as in Definitions 7 and 8 respectively, even if the best bid or ask are merely indicative (from the last existing reference price). The imbalance can always be computed even if one side is empty as it will simply take on 1 for an empty ask side or -1 for an empty bid side. The same logic is applied when both sides of the LOB are empty, except that $\rho(t)$ will be set to 0 in this particular case. This "fail-safe" setup for an empty LOB scenario makes it easy to initialise the simulation because the simulation can begin with just an indicative best bid and ask. This can avoid unnecessary volatility auctions which are currently not accounted for by our agent specifications.

6.3.3 Agent decision rules

The order sent by the agent is determined by the decision rules discussed in Section 6.2. Since agent actions depend on state variables of the LOB at time t, all the agents have access to the same LOB constructed from the market data listener but maintained in the model management system that mimics the role of order management system in the traditional trader workflow.

The arrival of decision times for all agents follow a truncated exponential. Therefore, we can simulate and sort all the decision time points in chronological order before starting the simulation. This means we can login to CoinTossX with a single client to manage and submit the orders for all the agents. At each decision point we identify the type and the specific agent and create the order for the agent according to the appropriate rules using the current state of the LOB. This is again a model design convenience specific to our particular model as managed from the MMS.

The simulation does not have to occur in real time and can be significantly sped up because we know all the decision times beforehand and we do not have any reactive agents.[3] The appropriate event times can be appended to the trade and quote data after the simulation. The only requirement to achieve this is the appropriate ordering of events so that the price-time priority of orders can be maintained within the matching engine.

6.3.4 Compute times and hardware

The Microsoft Azure[4] hardware configuration used and indicative times for each of the key steps of the simulation framework are given in Table 6.1. All times mentioned in Table 6.1 are dependent on the number of orders submitted and therefore the number of trader agents.

[3] By reactive agents we mean agents who act on events, *i.e.* agents listen for events and then act on them to define an event time.

[4] For an overview of Linux VMs in Azure refer to https://docs.microsoft.com/en-us/azure/virtual-machines/linux/overview.

Hardware	Specification
Virtual Machine	Standard A4m V2 Azure VM
Operating System	64-bit Linux Ubuntu 18.04-LTS
RAM	32GB
CPU	4× Intel Xeon CPU E5-2673 v3 @ 2.4GHz (32 threads)
Computation	**Run-time**
1-hour of Trading	41.179s
Sensitivity Ana.	35.746hrs
Calibration	10.880hrs

TABLE 6.1: Hardware configuration and indicative computation times. The average hour of trading for a single stock results in between a total of 4800 and 6600 orders. This includes, on average, between 2200 and 3100 limit orders, between 2000 and 2800 cancellations and between 350 and 450 trades.

6.3.5 Miscellaneous considerations

In the Johannesburg Stock Exchange (JSE) a volatility auction call session can be triggered during a continuous double auction when an instrument's circuit breaker tolerance has been breached (Johannesburg-Stock-Exchange 2020b). This means that if the execution price is larger than a 10% deviation from the dynamic reference price then a volatility auction will be triggered. The execution price is the price of the best bid or ask that is crossed when a market order arrives while the dynamic reference price is the deepest price level executed from the last transaction (Sing 2017).

Our agent specification does not account for the appropriate agent behaviour during an auction. Therefore, we impose a restriction on all liquidity takers where they will not submit their market order if the order will lead to an execution price with a deviation larger than 10% of the current dynamic reference price in order to avoid triggering a volatility auction.

Lastly, liquidity takers will not submit their market order if the contra side of their order is empty. We do this to simplify the data cleaning process. This is because when a market order is submitted to an empty order book the data feed returns a string but with no orders executed. This means that sending or not sending the order will lead to the same result.

6.3.6 Simulation

The framework starts with the basic logic for the updating the Limit-Order Book (LOB) (Algorithm 5), agent can then be activated from one of the three classes of agents: fundamentalists (Algorithm 6), liquidity providers (Algorithm 7), and chartists (Algorithm 8), and finally the submission of orders into the Matching Engine (ME) (Algorithm 9 to complete the order and agent activation sequence. The MMS listens for activation events. The Matching engine (ME) is separated from the Model Management System (MMS) as shown in Figure 6.1). The relationships between the algorithms in the MMS are shown in the simulation state-flow diagram in Figure 6.2. The definitions of key variables and parameters used in the simulation state-flow are given in Section 2.1.

To initialise all simulations we fix the number of liquidity takers with $N_{LT}^{f} = 1$ and $N_{LT}^{c} = 1$ and only vary the number of liquidity takers during calibration and sensitivity analysis. The LOB state of the best ask, best bid, spread, imbalance $(a(t), b(t), s(t), \rho(t))$ is arbitrarily set as (9950, 10050, 100, 0) and we start with an empty LOB.

Simulations carried out on calibrated parameters had the most consistent behaviour compared to ideal parameters found through sensitivity analysis. That is, parameters selected based on qualitative evaluations of moments of simulated price time-series, produced simulations with little to no liquidity taker activation's. This resulted in almost no price movement for longer simulations.[5] Figure 6.3 provides the simulated price time-series obtained from calibrated parameters for a 30 second interval.

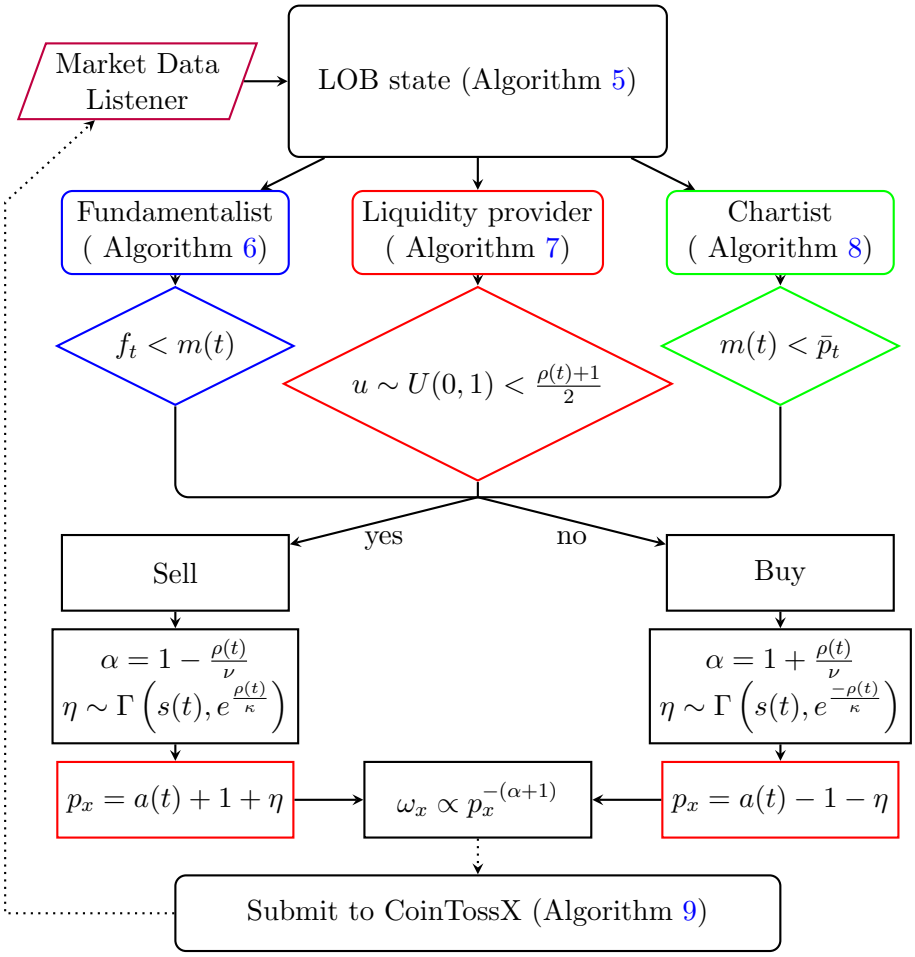


FIGURE 6.2: ABM simulation state flow diagram. See Section 2.1 for the definition of the variables and concepts used. The algorithm that updates the state of the Limit-Order Book (LOB) is given in Algorithm 5. The agent activation algorithms are given by Algorithms 6, 7, and 8 for the implementations for the Fundamentalist, Liquidity provider and Chartist agents, respectively. The order injection algorithm with the order submission logic is given in Algorithm 9.

[5]Due to the long simulation times with the Julia-Java wrapper, memory usage and speed became important. This meant JVM arguments for garbage collection and memory allocation were required.

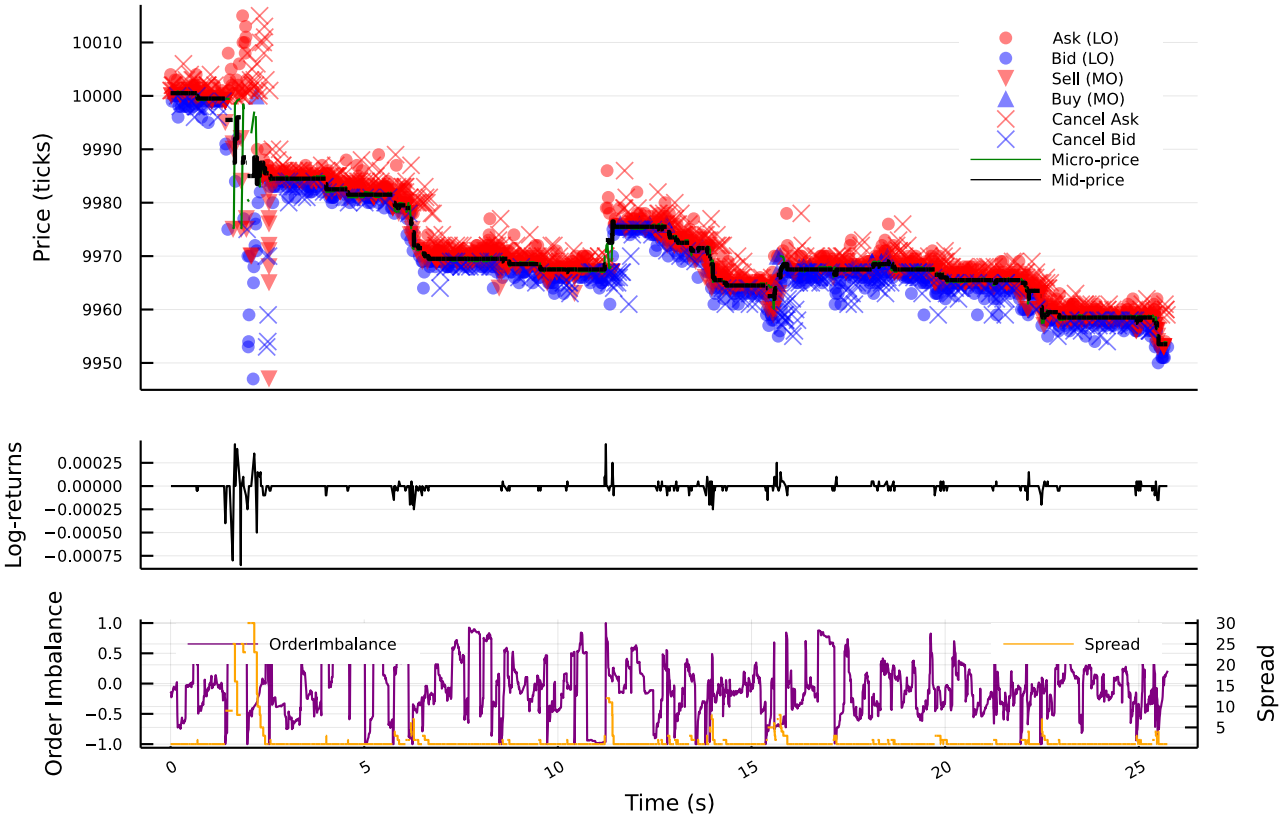


FIGURE 6.3: Half a minute of simulated price time-series from calibrated parameters. Orders placed at all levels of the LOB are visualised with markers for limit, market and cancellations of buy and sell orders. The progression of returns, spread and LOB volume imbalance are also visualised over the same time period.

6.4 Sensitivity analysis

We conduct a sensitivity analysis on the moments of both simulated mid-price and micro-price. This serves to demonstrate the effect of different price time-series on the estimates of simulated moments. Our choice of moments and statistics corresponds to those recommended by Gilli and Winker (2003) which are robust enough to reflect the properties of financial data and flexible enough to discriminate between parameter configurations (Winker, Gilli, and Jeleskovic 2007). Therefore, we adopt the following moments and statistics for being the most representative measures of the stylised facts of financial data: mean, standard deviation, kurtosis, Kolmogorov-Smirnov (KS) statistics, Geweke-Porter-Hudak estimator (GPD), augmented Dicky-Fuller (ADF), a combination of GARCH(1,1) parameters, and a Hill estimator.[6] These are given in Table 6.2.

[6]The modified Hill estimator is computed by numerically solving $\frac{1}{l-r+1}\sum_{j=r}^{l}\ln(X_j) = \frac{1}{\alpha} + \frac{\ln(X_l)X_l^{-\alpha} - \ln(X_r)X_r^{-\alpha}}{X_l^{-\alpha} - X_r^{-\alpha}}$ where X_l and X_r are the minimum and maximum return values respectively (Nuyts 2010).

Targeted Empirical Feature	Estimators and Motivation
Distribution shape	The mean μ, standard deviation σ, kurtosis μ_4, and Kolmogorov-Smirnov (KS) statistic of log-returns are chosen to represent the overall shape of the data distribution. The KS statistic compares the empirical distribution of the actual returns, to that of the simulated or bootstrapped returns obtained from the model. The actual returns obtain a KS statistic of 0 as they are identically distributed to themselves. In the context of both the calibration and the sensitivity analysis, the KS statistic was computed relative to the empirical distribution.
Long-range dependence	The Geweke and Porter-Hudak (GPH) estimator: Geweke and Porter-Hudak (1983) and Lillo and Farmer (2007) provide a measure of the long-range dependence of the absolute log returns by estimating the memory parameter $d \in [-0.5, 0.5]$ of the log-periodogram.
Serial auto-correlation	Augmented Dickey-Fuller (ADF) statistic: ADF measures of the extent of the random walk property of log returns by testing the null hypothesis that a unit root is present in the time series. A large negative test statistic is more evidence against the null hypothesis.
Long-memory	The empirical Hurst exponent (H): The Hurst exponent relates to the auto-correlations of the time series, and its rate of decay for increasing lags. A value in the range 0.5 and 1 indicates a time series with long-term positive auto-correlation (momentum) while a value in the range 0 and 0.5 is indicative of long-term negative auto-correlation (mean reversion). This is indicative of long-memory effects.
Short-range dependence	The sum of the two GARCH(1,1) parameters is used as a measure of short-range dependence. The reason for taking the sum is that this value is much more robust than using either one of the two parameters in the GARCH model alone.
Power-law tail	An "improved" Hill estimator (HE): Nuyts (2010) argue for the estimation of the tail index of a power-law distribution as an additional measure of the power behaviour and tailedness of the distribution over and above kurtosis. The use of this modified estimator is in response to the known issues and inefficiencies of the standard estimator.

TABLE 6.2: Estimators and moment parameters used for the calibration and sensitivity analysis along with the properties they are targeting. See Sections 6.4 and 6.5. The moments are used in the Nelder-Mead (NM) method with Simulated Minimum Distance (SMD) for the indicative calibration because of the methods simplicity, speed and reliability.

Overall, this results in a combination of 9 moments and statistics being used to characterise the statistical properties of the simulations.

To visualise the impact of pairs of parameters on the statistical properties of the simulated mid and micro price, we consider combinations of a range of reasonable parameter values. That is, we iteratively fix each parameter and vary the others in their ranges such that we obtain moment surfaces for all combinations of parameter values. We considered a range of five different values for each parameter, resulting in 3125 different combinations. Each unique parameter value thus has 625 replications over which to aggregate moments.

Given the large number of possible parameter configurations and that the model management system

is not coupled to the matching engine system, some care must be taken to ensure that separate back-to-back order submission periods are efficient, independent and do not result in unexpected behaviour (both at a software level and simulation output level). For example, we require that each simulation begins with an empty limit order book such that orders from previous simulations don't carry over. One way to achieve this would be to initiate new continuous trading sessions at set time intervals. Simulation time, however, is not fixed and this could cause orders to be submitted in the wrong trading session. For this reason we manually clear the off-heap storage and LOB each time the client logs out.

Figures C.1, C.2 and C.3 in Appendix C provide high-level summaries of the individual effects of parameter values on the moments of simulated log-return time-series.

Kurtosis was found to be the least reliable moment since extreme price changes during the initialisation period would have resulted in a large kurtosis value even if the price remained constant for the rest of the simulation.[7] Moments that were robust and good indicators of realistic price series include the GPH statistic, the Hurst exponent, and the Hill estimator. Due to the nature of the model, the Hurst exponent remained relatively constant and above 0.5 across parameters and simulations indicating a tendency towards momentum behaviour. Assigning a value of close to zero for the σ parameter of fundamentalists resulted in mostly mean-reverting behaviour with a Hurst exponent close to 0.5.

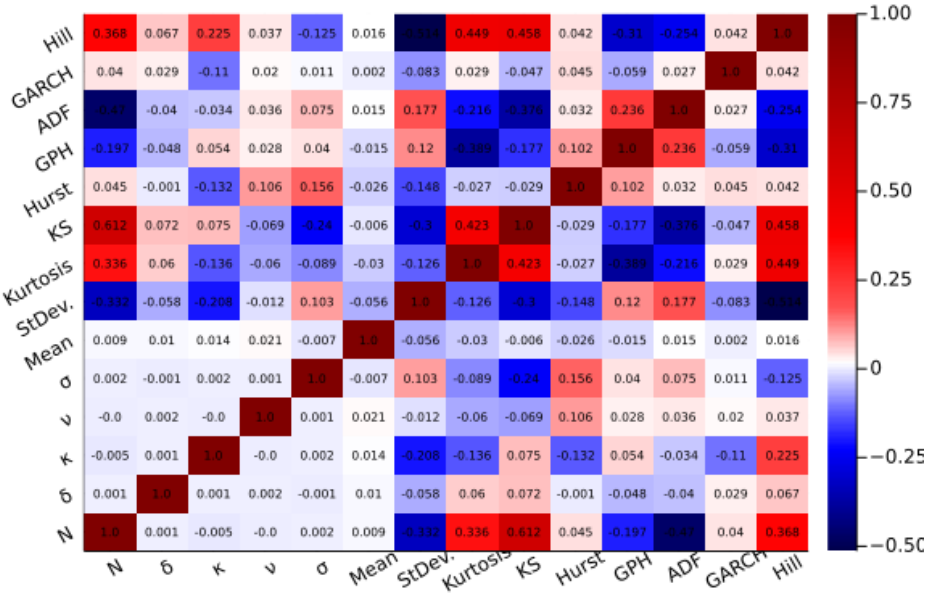


FIGURE 6.4: Bootstrapped correlations between individual parameter-moment combinations calculated on simulated micro-price time-series obtained from the sensitivity analysis. The parameter descriptions are provided in Table 6.3 and the moment descriptions in Table 6.2.

On average, most parameters by themselves do result in significant differences in simulated moments across their range of values. On the other hand, individual differences in parameters κ and N were found to result in slightly different simulated moments across their range of values on average. The strong dependence of simulated moments on N is expected as this parameter controls the number of liquidity providers. Smaller values of κ exhibited more leptokurtic log-returns which captures the intended effect in the model definition — small values of κ intensifies competition while larger values relax competition. Another observation worth noting is the relatively stable Hurst exponent which, with the exception of N, remains consistently above 0.5 — indicating a time-series exhibiting largely momentum behaviour.

[7] During calibration the instability of certain moments is accounted for by the inverse covariance matrix of empirical moments by down-weighting the effect of those moments on the objective function (refer to Sect. 6.5).

Considering different parameter combinations together produced simulated moments that were significantly different from those obtained by averaging the marginal effects of individual parameters over all other parameter combinations. This can been by the kurtosis, Hurst exponent and Hill estimator moment surfaces in Figure C.3 for combinations of δ & κ and N & ν. Certain moment surfaces in Figure C.3 differ depending on the use of mid-price or micro-price log-returns.

From Figure 6.4, as expected, the kurtosis, Kolmogorov-Smirnov statistic and Hill estimator are strongly correlated to the ratio of liquidity providers to liquidity takers while the ADF statistic and standard deviation of log-returns are negatively correlated. Similarly, with κ controlling the order-placement depth, it is positively correlated with the Hill estimator and negatively correlated with the standard deviation of log-returns.

6.5 Calibration

Ignoring parameter inference, we apply the MM SMD method for its simplicity, speed and reliability when there are more moments than parameters and we have significant computational time and resource constraints. As such our calibration should be considered indicative rather the robust. From the full set of model parameters a small subset of 5 free parameters was chosen: $\boldsymbol{\theta} = \{N, \delta, \kappa, \nu, \sigma\}$ (see Section 6.2 and Figure 6.2). This set of 5 parameters are described in Table 6.3 and are the parameters to be calibrated.

The list of fixed parameters that control other simulation characteristics are as follows. The inter-arrival rate parameter with minimum and maximum sampling intervals for liquidity takers (λ^T, $\lambda^T_{\min}$ and $\lambda^T_{\max}$ respectively) and liquidity providers (λ^P, $\lambda^P_{\min}$ and $\lambda^P_{\max}$ respectively) control for any low-frequency or high-frequency trading behaviour and affect, among other things, the amount of liquidity in the order book. These parameters are, however, redundant in the presence of parameters ν and δ which also control liquidity. Here simulations were not conducted in real time, but for computational convenience, event times were appended to the data after submission. Lastly, there was the power-law order-size cut-off parameter x_m as well as the time-to-arrival of cancellations that were held fixed which also affected the build up of liquidity in the order book. These components were kept fixed to alleviate calibration problems that may occur with redundant behaviour specifications as a result of conflicting and dependent parameters. In our case, agent decision rules are explicitly encoded, resulting in fewer parameters to calibrate and possibly a more unique set of calibrated parameters. This also ensures that the number of free parameters is less than the number of moments used to construct the objective function.

Param.	Description
N	The population ratio of liquidity takers to liquidity providers (see Equation 6.1).
δ	Volume size aggression multiplier as a percentage of mid-price (see Equations 6.6 and 6.7).
κ	Placement depth multiplier (see Equation 6.9).
ν	Scaling factor for power-law volume order size (see Equations 6.8 and 6.10).
σ	Fundamentalists' value perception uncertainty for the trading day (see Equation 6.4).

TABLE 6.3: Free model parameters (for calibrated values see Table 6.4).

For the problem of calibrating models with as few parameters as this one, we adopt the frequentist/method-of-moments simulated minimum distance (SMD) approach where the objective function is chosen to take into account the stylised facts of financial data (Winker, Gilli, and Jeleskovic 2007). The objective function to be minimised is thus the weighted sum-of-squares of estimation errors between simulated and empirical moments. The moments to be estimated are those mentioned in Section 6.4 along with the mean, standard deviation and Komogorov-Smirnov statistic which are estimated on the micro-price log-returns.[8] Denoting $\boldsymbol{m^e} = [m^e_1, \ldots, m^e_k]'$ and $[\boldsymbol{m^s} \mid \boldsymbol{\theta}] = [m^s_1, \ldots, m^s_k]'$ as the vector

[8] Tick-by-tick mid-prices were found to unreliable when trying to identify distributional properties through moments.

of empirical moments of real and simulated data respectively, we minimise the quadratic function

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta}) = G(\boldsymbol{\theta})'\boldsymbol{W}G(\boldsymbol{\theta}), \tag{6.11}$$

with

$$G(\boldsymbol{\theta}) = \frac{1}{I}\sum_{i=1}^{I}(\boldsymbol{m_i^e} - [\boldsymbol{m_i^s} \mid \boldsymbol{\theta}]), \tag{6.12}$$

where $\boldsymbol{\theta}$ is the vector of parameters, $\boldsymbol{\Theta}$ is the space of feasible parameters and $I = 5$ is the number of replications/simulations used in estimating the average moments and statistics. The matrix of weights W, given by the inverse covariance matrix of empirical moments ($\text{Cov}^{-1}[\boldsymbol{m^e}]$), takes into account the joint distribution of moments and assigns larger weights to those moments with the least uncertainty and that are more robust (Winker, Gilli, and Jeleskovic 2007).[9]

To alleviate the issue of a globally non-convex objective surface, we employ the Nelder-Mead with Threshold Accepting (NMTA) heuristic optimisation routine (Gao and Han 2012; Gilli and Winker 2003; Nelder and Mead 1965). The data on which the model is calibrated is chosen to be 1-hour of Naspers level-1 trade-and-quote data from the JSE.

6.5.1 Heuristic optimisation

Standard local optimisation methods are prone to failure because such an objective function does not always behave well to guarantee a global optimum solution. The objective function, built when taking into account the statistical properties of data, is not smooth or globally convex. There tends to be many local optima, so it is important to use an optimisation routine that is able to escape these local optima. Furthermore, due to the complexity of the problem and the nature of agent-based models, finding a global optimum is very computationally intensive, so any optimisation routine used should be able to efficiently explore the parameter space. Using the methodology proposed in the paper by Gilli and Winker (2003), we describe the Nelder-Mead with threshold accepting algorithm for the global optimisation of the objective function. In particular we follow the approach of Gao and Han (2012) when defining adaptive Nelder-Mead parameters that take into account the dimensionality of the optimisation problem.

In the Nelder-Mead (NM) simplex algorithm, each current solution has n free moving parameters which consists of $n + 1$ vertices of a simplex[10] in the parameter space $\boldsymbol{\Theta}$. Each vertex represents a set of values for the free moving parameters of the model. At each iteration, the algorithm shifts the simplex according to either reflection, expansion, contraction and shrinkage. These steps are repeated until the iteration limit is reached or the convergence criterion is met. For non-convex functions such as the one dealt with here, the simplex shrinks rapidly to local minima.

The standard threshold accepting (TA) algorithm considers only a single point in the parameter space. A single step consists of shifting the value of a randomly chosen parameter to a neighbouring value (however that may be defined). If the new point obtained represents an improvement in fitness it is accepted as the new current solution. However, even if the new point is slightly worse, it will still be accepted as the new current solution provided the drop in fitness does not exceed a predefined threshold. This improves the ability of the algorithm to escape local minima in the objective function. The fitness thresholds $\boldsymbol{\tau} = \tau_1, \ldots, \tau_r$ are decreased at set length intervals/rounds to allow for convergence to a more globally optimal solution at the end of the optimisation. Four different relative thresholds

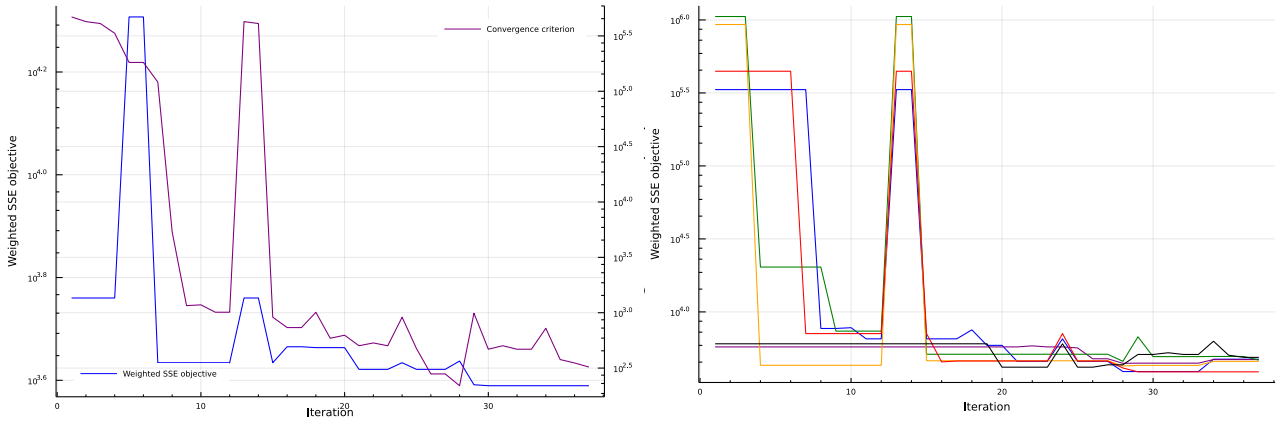
[9] The weight matrix $\boldsymbol{W}$ is estimated by applying a moving block bootstrap to the time series with a window of size 2000. As opposed to sampling individual returns, sampling bootstrap blocks preserves the auto-correlations of the time-series. For each block/window we re-sample with replacement until we obtain 1000 samples, each of which is equal to the length of the original data set. The moments and statistics are then calculated on each of these samples such that we can obtain the covariance between them. The condition number of the weight matrix $\boldsymbol{W}$ is $8.375811152908104 \times 10^{19}$ computed as $||W^{-1}|| \cdot ||W||$.

[10] A simplex is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions (*i.e.* an n-dimensional object with flat sides that has only $n + 1$ vertices).

are chosen to explore the local structure of the objective function along with a fixed number of steps $n_s = \{14, 12, 10, 8\}$ for each round respectively.

The Nelder-Mead simplex method provides an efficient way of identifying the search direction while threshold accepting helps to avoid local minima. Now considering a combination of the two algorithms, each iteration uses either the Nelder-Mead search or Threshold Accepting random shift with a probability of $(1 - \xi)$ and ξ respectively. The Nelder-Mead search will have a higher probability of being chosen. In the TA algorithm, the magnitude of the random shift is dependent on the mean value (over the simplex) of the parameter being shifted. In addition to the TA algorithm, threshold accepting is also applied to the simplex search by ensuring that reflection, expansion, contraction and shrinkage perturbations are evaluated against threshold adjusted fitness such that a worse fitness may be accepted in the simplex search as well.

The trace results of the optimisation routine are visualised in Figure 6.5. Close to convergence was reached near 40 iterations with no thresholds being applied in the last few iterations. Vertices in the simplex were close to converging to the best with no significant improvement in the best vertex for subsequent iterations.[11]



(A) Convergence criterion and objective fitness values of the best vertex

(B) Objective fitnesses of vertices in the simplex

FIGURE 6.5: NMTA optimization results are visualised through traces of the convergence criterion and simplex objective values.

6.5.2 Parameter and moment inference

From the sensitivity analysis we generate chains of parameters $\boldsymbol{\theta}$ from which we can compute the moments $\boldsymbol{m}^s$; each vector of moments is then associated with particular vectors of parameters. From the sensitivity analysis we can the compute the exposure $\boldsymbol{B}$ (here a 5×9 matrix) of parameters to the moments.[12] Then from the bootstrapped moment covariance matrix $\Sigma_{m^e} = \text{Cov}[\boldsymbol{m^e}] = \boldsymbol{W}^{-1}$ and the approximated parameter exposures to the moments $\boldsymbol{B}$, we can construct an indicative covariance for the parameters: $\Sigma_\theta = \boldsymbol{B}'\Sigma_{m^e}\boldsymbol{B}$. The diagonal of this gives us indicative sample variances: $\sigma^2_\theta = \text{diag}(\Sigma_\theta)$. Parameters obtained from calibration and sensitivity analysis are shown in Table 6.4.

[11] Optimisation with CoinTossX and the Julia-Java wrapper was initially prone to crashes in the Julia environment. It was suspected to be a Julia memory and garbage collection issue that came with very long consecutive simulations. It is noted that CoinTossX was designed purely as a matching engine for simulations, rather than a tool to be used for calibration.

[12] Here $B_{ij} = \text{Cov}[\theta_i, m^s_j]/\text{Var}[m^s_j]$ for ith parameter and jth moment in the exposure matrix $\boldsymbol{B}$.

Parameter	$\hat{\theta}_{0.025\%}$	$\hat{\theta}$	$\hat{\theta}_{0.975\%}$
N	3.609	5.992	8.376
δ	0.058	0.092	0.125
κ	1.716	2.555	3.394
ν	2.876	3.381	3.887
σ	0.08	0.418	0.756

TABLE 6.4: Calibrated parameters along with 95% confidence intervals where parameter variances are obtained from sensitivity analysis parameter chains. See Table 6.3 for the parameter descriptions.

To obtain the indicative error bars on simulated moments in Table 6.5, we obtain moment variances from the diagonal of the bootstrapped moment covariance matrix Σ_{m^e} and compute confidence intervals as $\boldsymbol{m}^s \pm 1.96 \times \sqrt{\text{diag}(\Sigma_{m^e})}$.

Moment	$\hat{m}^s_{0.025\%}$	$\hat{m}^s$	$\hat{m}^s_{0.975\%}$	$\hat{m}^e$
Mean	-0.0	-0.0	0.0	0.0
StDev.	0.0	0.0	0.0	0.001
Kurtosis	-1195.441	2323.204	5841.849	4.12
KS	0.424	0.456	0.488	0.0
Hurst	0.488	0.527	0.566	0.231
GPH	-0.199	0.055	0.309	0.729
ADF	-311.74	-296.876	-282.012	-176.529
GARCH	0.859	1.033	1.207	1.005
Hill	0.676	0.958	1.241	2.043

TABLE 6.5: Comparison between empirical moments and moments estimated on tick-by-tick simulated micro-price returns along with 95% confidence intervals. See Table 6.2 for the description of the calibration moments.

6.6 Exploratory Data Analysis

It is difficult to compare the "goodness" of one ABM to another. Such comparisons are usually based on the model's ability to replicate empirical observations which involves subjective comparisons given the qualitative nature of most stylised facts. There are a wide variety of financial ABM's that are able to replicate common empirically observed facts. However, Platt and Gebbie (2017) demonstrate possible inadequacies of a stylised fact-centric approach to model validation and stress the importance of rigorous calibration methods as well as qualitative measures relating to model parsimony and the ability of the model to make predictions or produce empirical phenomena not directly encoded in, or used in the estimation process. It is also usually the case that a unique set of parameters for the calibration of a specific model to a time series does not exist. Without a unique set of parameters allowing us to reproduce the properties of financial time series drawn from a particular market, we cannot argue that introducing and observing changes in the model truly reflects the same changes in the market. A good model should both be able to replicate the stylised facts of financial markets and have parameters that behave in clear ways that do not have insignificant effects on the resulting behaviour of the simulation.

That said, given the uniqueness of different markets, qualitative empirical-versus-simulation comparisons remain the best validation methods in computational agent-based models. We choose an extensive list of stylised facts that are constraining enough to demonstrate the realism of our simulations (see Section. 2.3.2 for a shortened list of financial market stylised and empirical facts).

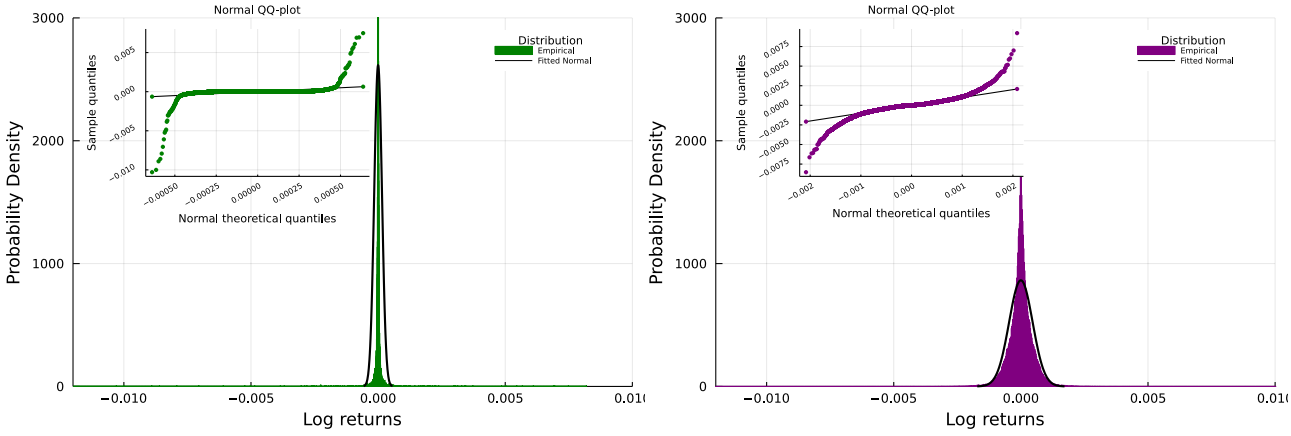
6.6.1 Stylised facts

The stylised facts presented hereafter are computed on micro-prices on a tick-by-tick basis and are compared between simulated and empirical data. The micro-price is updated whenever an event changes the best bid or ask. The returns are computed as price fluctuations (see Definition 11):

$$r(t_k) = \ln\left(\psi(t_{k+1})\right) - \ln\left(\psi(t_k)\right),$$

where $\psi(t_k)$ is the micro-price at time t_k (Definition 9). The stylised facts included are by no means exhaustive but rather include the most common of these: order-flow auto-correlation, log-return and extreme log-return distributions and log-return auto-correlation. In addition to these we compute the depth-profile curves on simulated LOB data.

First, we investigate the distribution of the tick-by-tick returns. Figure 6.6 plots the full distribution of the returns with quantile-to-quantile (QQ) plots fitted to a normal distribution provided as insets. Both distributions are highly leptokurtic with heavy tails. Note, however, that empirical returns are more volatile and has its density more evenly distributed away from the mean. This is in contrast to simulated returns which has a larger proportion of its density centred at zero.

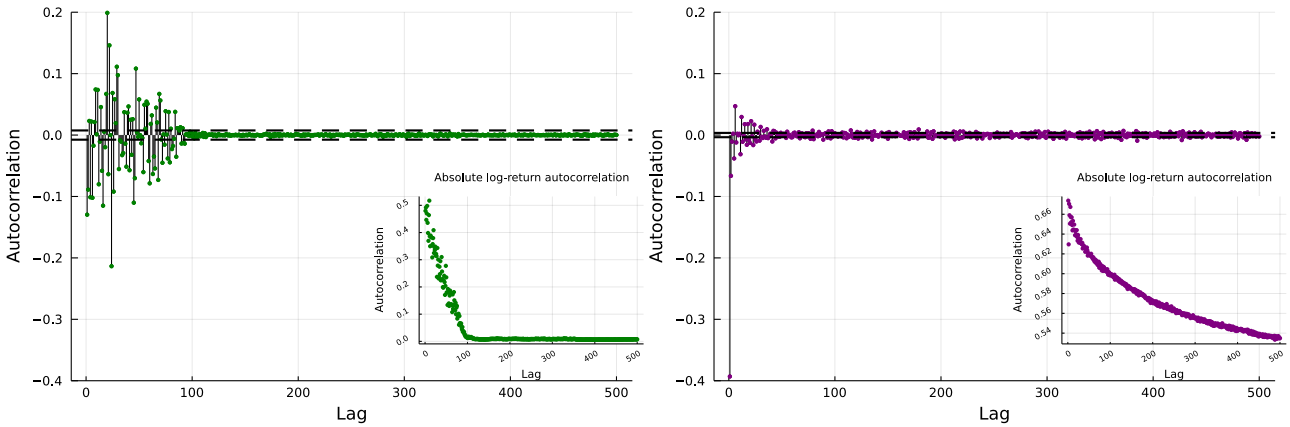


(A) CoinTossX — calibrated parameters

(B) JSE — empirical

FIGURE 6.6: Tick-by-tick simulated and empirical micro-price log-return distribution properties are depicted along with fitted Normal distributions as references. Simulated price-series were generated by applying the optimal calibration parameters on CoinTossX while empirical results are obtained from JSE level 1 trade-and-quote data. QQ-plots are provided as insets.

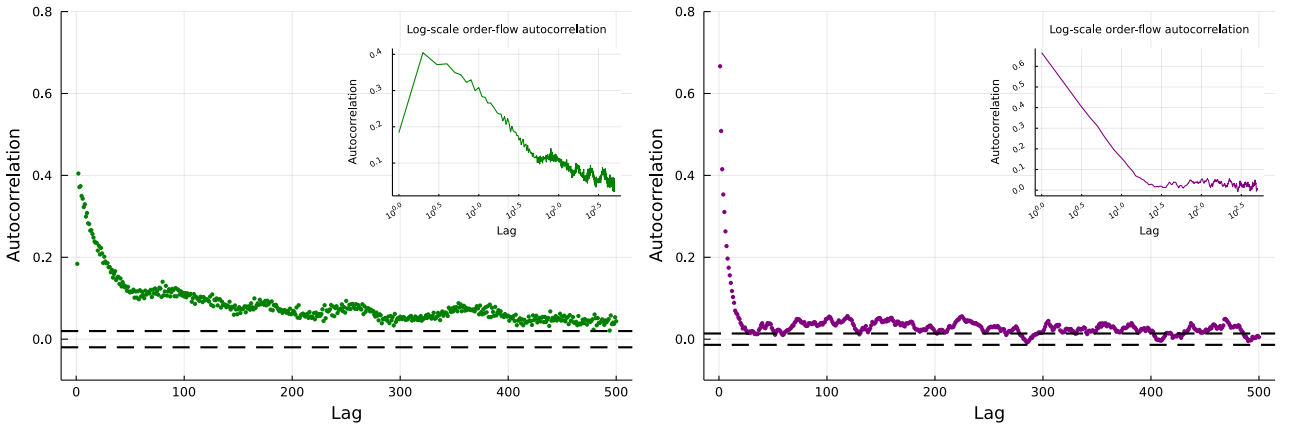
Referring to the log-return auto-correlation plots in Figure 6.7, the strong negative first order auto-correlation seen in empirical returns is indicative of a strong mean-reverting component. This is less prevalent in simulated returns which exhibit significant auto-correlations for the first few lags that decay to zero relatively quickly. We speculate that these differences in long-memory characteristics are due to the significant difference in relative liquidity and order book resilience that are produced by the calibrated parameters.



(A) CoinTossX — calibrated parameters (B) JSE — empirical

FIGURE 6.7: Tick-by-tick simulated and empirical microprice log-return auto-correlations plots. Simulated price-series were generated by applying the optimal calibration parameters on CoinTossX while empirical results are obtained from JSE level 1 trade-and-quote data. Absolute log-return auto-correlation plots are provided as insets as an indicative measure of long-memory.

The order-flow is represented by a time-series of values with $+1$ for buyer initiated trades, and -1 for seller initiated trades. Persistence in the order-flow is then in the sense that buy orders tend to be followed by more buy orders, and sell orders tend to be followed by more sell orders. Possible causes for this are herding behaviours (positive correlation between the behaviour of different investors) and order splitting (positive auto-correlation in the behaviour of single investors). Figure 6.8 plots the order-flow auto-correlations from the simulations and empirical data on JSE level 1 trade-and-quote data. The trade classification is performed using the Lee and Ready (1991) classification for the empirical data and we compute up to 500 lags between the trades. Furthermore, the same auto-correlation plots presented on a $\log_{10}$ scale are provided as insets. Despite the significant differences in liquidity and the number of trades, we see that both simulated and empirical data sets exhibit persistent order-flows.



(A) CoinTossX — calibrated parameters (B) JSE — empirical

FIGURE 6.8: Order-flow auto-correlation plots based on true and inferred trade signs for simulated and empirical results respectively. Simulated price-series were generated by applying the optimal calibration parameters on CoinTossX while empirical results are obtained from JSE level 1 trade-and-quote data. Provided as insets are the same auto-correlation plots with the lags presented on a $\log_{10}$ scale.

Figure 6.9 includes the distributions of the left tail (below the 5th percentile) and the right tail (above the 95th percentile) with QQ-plots fitted to a power-law distribution provided as insets (presented on

a log-log scale). The power-law distributions are fitted using maximum likelihood estimation (MLE) where the probability density function is as:

$$p(x) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x}{x_{\min}} \right)^{-\alpha},$$

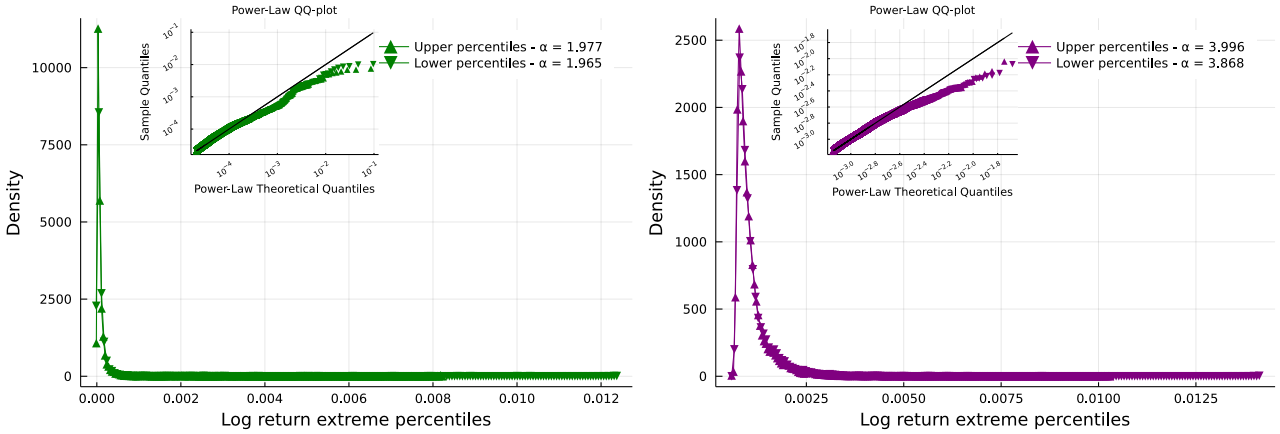
where $x \geq x_{\min} > 0$. The complementary cumulative distribution is:

$$1 - F(x) = \left(\frac{x}{x_{\min}} \right)^{-\alpha+1}.$$

Here we only estimate α because $x_{\min}$ is set as the cutoff percentile. The MLE for α is given as:

$$\hat{\alpha} = 1 + n \left[\sum_{i=1}^{n} \ln \frac{x_i}{x_{\min}} \right]^{-1}.$$

The tail distributions between data sets differ significantly with simulated returns having less of a power-law behaviour than empirical returns as indicated by the smaller estimated tail-index α.



(A) CoinTossX — calibrated parameters

(B) JSE — empirical

FIGURE 6.9: Distributions of upper and lower extreme log-returns above the 95th percentile and below the 5th percentile respectively computed on tick-by-tick simulated and empirical micro-price log-returns. Simulated price-series were generated by applying the optimal calibration parameter on CoinTossX while empirical results are obtained from JSE level 1 trade-and-quote data. The left and right tails have QQ-plots fitted to a power-law distribution provided as insets.

Our empirical data only has the level 1 LOB, so the depth profile is only computed on calibrated parameters in Figure 6.10. To construct these profiles we average the total volume available at the top 7 price levels of the LOB. The resulting curves exhibit a strong decay in the amount of liquidity available at each price level. We also observe asymmetry between the bid and ask mean relative depth profiles. The tendency for most of the volume to be placed at the first level of the LOB is inline with empirical observations (Biais, Hillion, and Spatt 1995; Gould et al. 2013) and provides a partial explanation for the power-law price-impact curves.

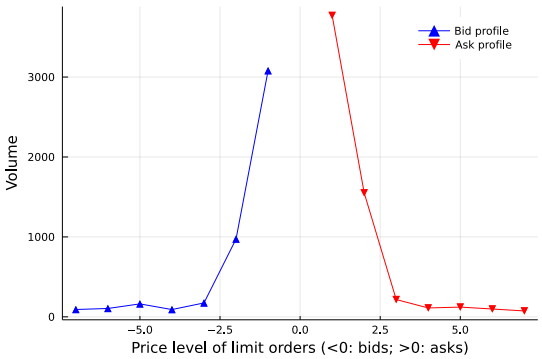


FIGURE 6.10: LOB depth profiles on simulated results from optimal calibration parameters as depicted by the average volume at each of the top seven price levels through time.

6.6.2 Price response and impact

Here we compare the immediate price impact functions, which are the immediate price responses that are due to transactions, between the empirical and simulated data. Price impact quantifies how a transaction of a given volume affects the best bid/ask price (Harvey et al. 2016; Jericevich, Chang, and Gebbie 2020). We define the impact or immediate price increment of a transaction occurring at time t_k as:

$$\Delta p_{t_k} = \ln\left(m(t_{k+1})\right) - \ln\left(m(t_k)\right),$$

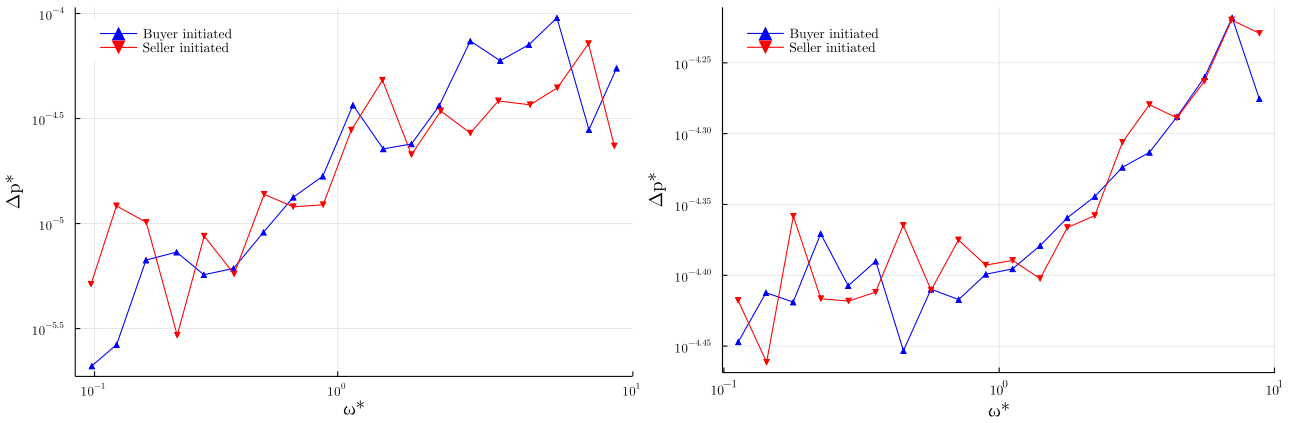
where t_k is the time of mid-price before the transaction and t_{k+1} is the time of mid-price immediately after the transaction.[13]

Additionally, we follow Harvey et al. (2016) and Jericevich, Chang, and Gebbie (2020) and normalise the trade volume as:

$$\omega'_{ij} = \frac{\omega_{ij}}{\sum_{k=1}^{T_j} \omega_{kj}} \left[\frac{\sum_{j=1}^{M} T_j}{M}\right],$$

where ω'_{ij} is the daily-normalised volume for trade i on day j, T_j is the number of trading events on the jth day and M is the total number of days. The relationship between price change and transaction size is investigated separately for buyer- and seller-initiated transactions. The transactions in the JSE data set are classified according to the Lee-Ready rule (Lee and Ready 1991).

Figure 6.11 investigates the price impact for a given volume size. We create 20 logarithmically spaced normalised volume bins between $[10^{-1}, 10^{1}]$. For each of these volume bins we compute the average price change Δp^* and the average normalised volume ω^* over the entire data set. The relationship between the average price change Δp^* and the average volume bin ω^* is then plotted on a log-log scale for buyer- and seller-initiated transactions. Consistent with previous findings (Harvey et al. 2016; Jericevich, Chang, and Gebbie 2020; Lillo, Farmer, and Mantegna 2003), we see that there is a power-law relationship (linear on a log-log scale) between the price change and normalised volume in both the simulated and empirical data.



(A) CoinTossX — calibrated parameters

(B) JSE — empirical

FIGURE 6.11: Individual buyer-initiated and seller-initiated price impact curves plotted on a log-log scale with average daily-normalised volumes ω^* on the x-axis and average price increments Δp^* on the y-axis. Simulated price-series were generated by applying the optimal calibration parameters on CoinTossX while empirical results were obtained from JSE level 1 trade-and-quote data.

[13] We make the strict assumption to only use the mid-price immediately after the transaction. For the JSE data set, all transactions are immediately followed by an updated quote. For CoinTossX, we also update the quote immediately following a transaction.

6.7 Discussion

6.7.1 Limitations

For the purpose of future improvements in market simulations with CoinTossX we mention two key shortcomings of the model in this work. The first and most important for dynamics and agent activation being that of the limit price placement.

As shown by the depth profiles, limit price placements are sampled from a Gamma distribution which will have most of their density at or just below the best bid/ask. This results in most of the volume being placed at the best bid or ask price level. Although this loosely conforms with empirical observations, this characteristic tends to result in little to no liquidity taker activation in the presence of too many liquidity providers because trade volumes were often insufficient to break through the first level. This can then result in very little price movement and thus little variation in log-returns.

This shortcoming is further aggravated, but not caused, by the choice of a power-law volume distribution. There is a tension because too many liquidity takers will mean too little liquidity even at level 1. Deeper levels in the order book are shallow with most volume directed at level 1. Clusters of trades or large individual trades breaking through the first level would subsequently break through all deeper levels and thereby emptying one side of the order book. With one side empty, subsequent limit prices placements sampled through the Gamma distribution would tend to place prices far from the contra best — resulting in a large spread. This behaviour would repeat as large price movements can increase the aggression of liquidity takers. The end result being unrealistically large price changes, sometimes into the negatives. Additionally, once large price changes occur, the fail-safe preventing volatility-auctions to trigger would prevent LT agents submitting further market orders.

The second feature that may have had adverse effects on simulation dynamics were agent activation rules. In particular, fundamentalists were assigned a fixed value perception for the day. If the perceived value was far from the starting mid-price, consistent upward/downward trends would persist throughout the day with little market pressure being applied to the opposite side. This may give evidence for the existence of contrarian strategies in real markets.

6.7.2 Future extensions and reactive agents

Having an independent matching infrastructure at our disposal means that greater attention can be applied to the model specification, configurations and attributes as the mechanics of order matching is externally provided as a given. To implement a truly asynchronous actor-based model, the modelling framework needs to be able to react to streaming data in an event based framework. This will require the model management system and its integration with the data feed to become reactive. This is how real financial markets function. This will require a reactive framework such as akka to implement a truly multi-agent model framework (as one example) or the extension of the existing code base.[14]

Doing so will allow the system to include additional classes of agents, *e.g.*, RL agents, learning agents, distressed agents and multi-asset agents that can potentially induce cross asset correlations. Each asset can then still be traded by at least the 3 basic agent formulations: the two LF and HF agents who will then be operating in event time. Additional agent classes can then inject trades into the matching engine and the various model management systems can then monitor the order book and trade events in order to make trading decisions.

Examples of multi-asset agent-based models can be found in Böhm and Chiarella (2005) and Ponta, Raberto, and Cincotti (2011). These models have historically been based on sequential order submission and market clearing. However, with a bit of work these can be extended to become compatible with an event driven asynchronous framework moderated by a matching engine.

[14]The requirement for reactive agents can be implemented using our Julia components by a simple migration into a framework such as Reactive.jl or Rocket.jl to replace our hybrid implementation.

6.8 Summary

We have presented an agent-based model in an artificial exchange framework that is able to recover price-impact while replicating the conventional stylised facts by directly interacting with a decoupled Matching Engine (ME) that manages orders away from the software relating to agent interactions and rules.

The key innovation here is that the model updates no longer occur in chronological or calendar time. In many models such as that of Leal et al. (2016), the activations are random, but the model takes place in calendar time with sequential market clearing. This means that the agent rules do not depend on market clearing prices and additional constraints are needed to create realism.

Here the Model Management System (MMS) polls the market data feed based on asynchronous model time, rather than be entirely driven by the event time of the market. This is a "hybrid framework" where we have not isolated the MMS from the ME entirely. This was a convenient choice made to reduce compute times for the work done here. However, the system was designed and setup for the "reactive framework" which is what future work should focus on.

We recall that the MMS plays the same roles as the OMS/EMS in a typical trading technology stack. The hybrid solution is computationally convenient here because we make use of the model specification that the only trade events experienced by the matching engine will be those triggered and injected from the MMS environment. This means that we do not need to implement a truly reactive framework as there are no other agents that can trigger events for which the agent-based model needs to react to, *i.e.*, there are no agents outside of those from the model specification.

The complexity of balancing a reactive agent-based modelling system with the convenience of fast calibration is central to the design decisions taken here. The use of Nelder-Mead (NM) with the heuristic Threshold-Acceptance (TA) approach was similarly pragmatic in as much as we have opted for a fast calibration methodology that will allow us to quickly get to indicative parameter values. Hence a practical framework that allows one to quickly get indicative parameters and generally replicate market stylised facts with relative effectiveness. The framework balances theoretical and statistical rigour, computational convenience and implementation realism. We also made the choice of having more moments than there are calibration parameters; here 5 parameters but 9 moments.

The agent-based model's optimal calibration parameters were found to provide more consistent time-series behaviour across seeds than parameter values selected based on the sensitivity analysis. It was noted that simulated prices obtained from sensitivity analysis parameters would occasionally either trigger the volatility-auction fail-safe or close out prices until they remained unchanged — thereby preventing any further actions from agents who depend on price movements.

In the instance where the MMS is one of many such frameworks connected to the market then the MMS needs to listen for externally triggered event types that are not the result of its own agent responses. This is computationally expensive. This is important if one is then going to implement alternative agents exogenous to those specified in this particular MMS, *e.g.*, distressed agents, learning agents, a specific implementation of a particular trading algorithm, or particular types of agent rules that one would like to investigate relative to the core framework. The system as it is currently designed and configured can accommodate this. However, this was not implemented here due to computational efficiency requirements and is left to future work.

7 Conclusion

Chapter 4 introduced CoinTossX as a low latency high throughput artificial stock exchange that would serve as a realistic platform with which to perform simulation experiments. The key characteristics of this software that make it advantageous for our use case includes: high-speed order matching, an architecture similar to that of commercial exchanges, asynchronous order submissions and flexible functionality. The rest of the thesis was dedicated to the implentation of a simple agent-based model and point-process model using the Julia language in a trading environment for: informing more advanced financial market modelling abstractions and creating a setting for robust and realistic trading scenario replications.

Applying a multivariate Hawkes processes, as a reference model for generating bid and ask prices and market orders to be submitted to the matching engine, served as a stress testing tool and provided the motivation for further applications such as: market making algorithms, agent-based models, optimal execution algorithms, general trading strategies and other stochastic models without requiring any modification of the simulation environment. Chapter 5 also adresses an independent market microstructure research goal — to determine the extent to which a matching engine can cloud the modelling of underlying order submission and management processes. The findings suggest that market mechanics blur the identification/estimation of the true order generating processes. This suggests that when considering market models, such as agent-based models at least in the setting of continuous double auctions, it is prudent to account for practical considerations imposed by market rules, mechanism and asynchronicity. It may be inadequate to argue their impacts away using additional sources of noise or systematic model parameter uncertainties.

The agent-based model provided evidence for the application of more sophisticated models with the intent of producing realistic financial market time-series as compared against empirical findings. Chapter 6 formulated a semi-asynchronous ABM submitting orders directly to the decoupled matching engine without relying on sequential market clearing in calendar time. This model recovers conventional stylized facts as well as price impact through interacting low-frequency liquidity takers and high-frequency liquidity providers. The key innovation here is the shift to a reactive agent based model for financial markets where agent activations can be specified independently of others and are completely determined by exogenoues market factors.

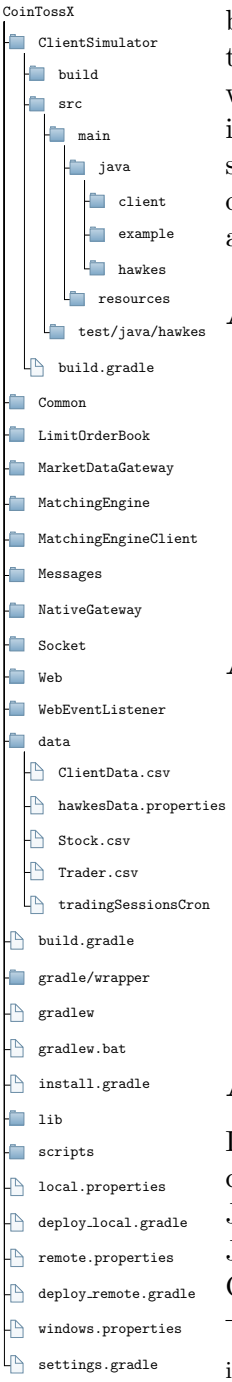
Having the complex set rules of the environment/system/architecture be separate or independent from the modelling and decision making processes could potentially overcome the calibration boundary for agent-based models attempting to empirically relate temporal dynamics with specific agent behaviours. The inclusion of asynchronous matching improves the existing state-of-the-art in agent-based modelling of financial markets, which typically relies on discrete, sequential time. These simple simulation models provide a framework and giudeline for more advanced models as well more realistic/robust trading scenarios in a development environment similar to those used in industry. This work may find applications both in industry as well as research. The work provided on order matching systems may be particularly insightful for commercial settings as it relates to trading, backtesting and technological concerns.

Future work on the CoinTossX matching engine simulator will involve significant upgrades with larger-scale use cases in mind. Of further interest in the modelling space is the implementation of alternative agents exogenous to those specified in this particular MMS, *e.g.*, distressed agents, learning agents, a

specific implementation of a particular trading algorithm, or particular types of agent rules that one would like to investigate relative to the core framework.

A Using CoinTossX

A.1 Installation and deployment



Below are the instructions for deploying CoinTossX to a user's local machine or virtual machine in a cloud environment. These instructions apply to Windows, Linux and OSX operating systems. CoinTossX is a Java web application and is built using the Gradle build tool. The user need not install Gradle or change the version of Gradle installed on their system as the project uses the Gradle wrapper to download and run the required Gradle version[a]. Currently CoinTossX is compatible with Java version 8 and Gradle version 6.7.1. The application can be started from the command line, however, it is recommended that the user make use of a Java IDE such as Eclipse or IntelliJ IDEA to automate the start-up process and simplify deployment.

A.1.1 Prerequisites

1. JDK version 8 (see Section A.1.3 for more detail)
2. A computer with 4 or more CPU cores and a sufficient amount of RAM (ideally 4 cores 32 GB but can be less depending on the number of clients and stocks used).
3. A Java IDE such as Eclipse or IntelliJ IDEA.
4. The user may be required to run commands using Command Prompt (Windows) or Bash (Linux and OSX).

A.1.2 Additional resources

1. Tutorial for building a java project with gradle: .
2. Tutorial for building a simple Java web application:
3. Introduction to Spring Boot
4. Introduction to Apache Wicket
5. Tutorial for setting up a student Azure account:
6. Instructions for the installation of Oracle JDK 8 and configuration of the system environment variables on Linux:
7. Calling Java in Julia , Python and R
8. Using the Aeron message transport:

A.1.3 Installation

If the correct version of Java is already installed and configured correctly, the user can skip this section. CoinTossX is currently only compatible with version 8 of Java. Therefore, the user should install version 8 of either the Oracle () or Open JDK (Java Development Kit)[b]. For simplicity, it is assumed that Windows and OSX users will install the Oracle JDK while Linux users will install the Open JDK.

[a] The Wrapper is a script that invokes a declared version of Gradle, downloading it beforehand if necessary. This standardizes a project on a given Gradle version and simplifies execution (especially when using an IDE)

[b] Note that the user should install the JDK (which includes the JRE), not only the JRE.

1. **Windows** — After installing Oracle JDK 8 using the link above, if not done so automatically by the java install wizard, ensure that `JAVA_HOME` is set and that the java executable is set in the path environment variable. To set/add java to the system's `JAVA_HOME` and `path` environment variables, go to `Settings > Advanced System Settings > Environment Variables > System Variables`. Then add the location of the java installation to a new variable called `JAVA_HOME` which points to the relevant java distribution (e.g. `C:\Program Files\Java\jdk1.8.0_271`). Thereafter append a new pointer in the `path` environment variable by adding `\%JAVA_HOME\%\bin`.

2. **Linux** — Note that Open JDK is not compatible with CoinTossX. Installation of Oracle JDK 8 is done by executing the commands below: Download the tarball java installation file (e.g. `jdk-8u271-linux-x64.tar.gz`) or use the `wget` command. Open the terminal and execute the following commands:

   ```
   sudo mkdir /usr/lib/jvm
   cd /usr/lib/jvm
   ```

   With the working directory set above, extract the distribution from the tarball into this folder using the first command below. This command assumes the tarball was downloaded to the `Downloads` folder and should be changed to match the directory the user downloaded it to. Next, open the system environment variables configuration file (Vim is used here but can be replaced with any other text editor). Change the commands according to the version downloaded and the download path.

   ```
   sudo tar -xvzf ~/Downloads/jdk-8u271-linux-x64.tar.gz
   sudo vi /etc/environment
   ```

   In the opened file add the following bin folders to the existing `PATH` variable:

   ```
   /usr/lib/jvm/jdk1.8.0_271/bin
   /usr/lib/jvm/jdk1.8.0_271/db/bin
   /usr/lib/jvm/jdk1.8.0_271/jre/bin
   ```

   Add the following environment variables at the end of the file:

   ```
   J2SDKDIR="/usr/lib/jvm/jdk1.8.0_271"
   J2REDIR="/usr/lib/jvm/jdk1.8.0_271/jre"
   JAVA_HOME="/usr/lib/jvm/jdk1.8.0_271"
   DERBY_HOME="/usr/lib/jvm/jdk1.8.0_271/db"
   ```

   Enter the following commands to inform the system about Java's location:

   ```
   sudo update-alternatives --install "/usr/bin/java" "java"
       ↪"/usr/lib/jvm/jdk1.8.0_271/bin/java" 0
   sudo update-alternatives --install "/usr/bin/javac" "javac"
       ↪"/usr/lib/jvm/jdk1.8.0_271/bin/javac" 0
   sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_271/bin/java
   sudo update-alternatives --set javac /usr/lib/jvm/jdk1.8.0_271/bin/javac
   ```

3. **OSX** — To set `JAVA_HOME` run

   ```
   export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_[version].jdk/Contents/Home
   ```

Finally, clone the CoinTossX repository (using either `git` in the terminal or Github Desktop). The user can clone the repository to any location of their choosing.

```
cd [directory to clone into]
git clone https://github.com/dharmeshsing/CoinTossX.git
```

A.1.4 Deployment

The next set of instructions provides guidelines for the deployment of CoinTossX. By this point the user will have cloned the repository to a location of their choosing for either local or remote deployment. Depending on the operating system, different deployment configurations would need to be employed — hence the multiple configuration files. The files with the ".properties" extensions define the required

ports for each component as well as user specific path definitions. To match the location to which the repository was cloned, the user would have to configure the `local.properties` file (for Linux) or the `windows.properties` file (for Windows) to correspond with the user's directories. For remote deployment the user would have to configure the `remote.properties` with the corresponding paths on the remote server. The path variables which need to be configured are:

1. `MEDIA_DRIVER_DIR` pointing to the Aeron Media Driver. This folder will only be created after the application is started. For Linux users it is recommended that the default path (`/dev/shm`) is not deviated from in order to achieve optimal performance. Otherwise the path can be amended as follows:

   ```
   MEDIA_DRIVER_DIR=[...]/aeron
   ```

   Note that if the user deviates from the default media driver path, they would have to make the same change in the `build.gradle` file.

2. `SOFTWARE_PATH` pointing to the start-up folder that will be created upon deployment. This path can be set to any valid path on the user's machine and may be given any name. For example:

   ```
   SOFTWARE_PATH=[...]/[folder name]
   ```

3. `DATA_PATH` points to the data folder within the software path. For example:

   ```
   DATA_PATH=[SOFTWARE_PATH]/data
   ```

Before the application can be started, we are required to change a few system settings to ensure that network performance and system memory are utilised correctly. Firstly, the receive and send UDP buffer sizes/limits need to be configured as follows.

```
sudo sysctl net.core.rmem_max=2097152
sudo sysctl net.core.wmem_max=2097152
```

Secondly, this Java application requires large pages to be enabled. By default Linux does not enable any HugePages (portions of memory set aside to be solely used by the application). To determine the current HugePage usage, run `grep Huge /proc/meminfo`. Furthermore, the default HugePage size is 2MB (2048kB). So if the user wishes to enable approximately 20GB of HugePages (dedicated memory), this would require 10000 HugePages — so the command to run would be

```
sudo sysctl vm.nr_hugepages=10000
```

The instructions below demonstrate both local and remote deployment. Users deploying the application to Microsoft Azure, CHPC, Wits Server or any other server may choose to do so locally or remotely. Remote deployment will require that the user specify the above paths to correspond with that of the remote server. For users deploying remotely, one must first ensure that SSH is enabled on the server and that port 22 is open for the transfer of files. Additionally, the username, IP address and password fields in the `deploy_remote.gradle` file should match that of the server.

1. Set the current directory to the location of CoinTossX:

   ```
   cd [path to]/CoinTossX
   ```

2. To build the project, use the provided gradle wrapper to run:

   ```
   ./gradlew -Penv=local build -x test # For local deployment
   ./gradlew -Penv=remote build -x test # For remote deployment
   ./gradlew.bat -Penv=windows build -x test # For windows deployment
   ```

3. To deploy the project to a start-up folder, execute one of the following.

   ```
   ./gradlew -Penv=local clean installDist bootWar copyResourcesToInstallDir copyToDeploy
       ↪deployLocal # For local deployment
   ./gradlew -Penv=remote clean installDist bootWar copyResourcesToInstallDir copyToDeploy
       ↪deployRemote # For remote deployment
   ./gradlew.bat -Penv=windows clean installDist bootWar copyResourcesToInstallDir
       ↪copyToDeploy deployLocal # For windows deployment
   ```

4. Finally, to run the program, execute the shell script:

```
sh [SOFTWARE_PATH]/scripts/startAll.sh # For local start-up
sh [SOFTWARE_PATH]/scripts/remote_startAll.sh # For remote start-up
[SOFTWARE_PATH]/scripts/startAll.bat # For Windows start-up
```

5. Access the web app by typing `localhost:8080` (if deployed locally) or `[server IP]:8080` (if deployed remotely) into the URL search bar of a browser.

A.2 Usage

A.2.1 JVM arguments

Argument	Description
`-Xms<heap size>`	Minimum amount of heap size (memory) allocated to the application
`-Xmx<heap size>`	Maximum amount of heap size (memory) allocated to the application
`-XX:+UseG1GC`	Default garbage collection algorithm
`-XX:+UseStringDeduplication`	Optimizes the heap memory by reducing duplicate String values to a single global `char` array
`-XX:+UseLargePages`	Use large page memory if it is supported by the system
`-XX:+OptimizeStringConcat`	Optimize String concatenation operations where possible
`-server`	Selects server application runtime optimizations. Results in longer warm-up time but optimized to produce higher throughput
`-XX:GuaranteedSafepointInterval=<ms>`	Guarantee a safepoint every `<ms>` milliseconds
`-d<OS bit>`	OS environment (32/64 bit)

TABLE A.1: JVM arguments.

A.2.2 Trading sessions

To configure the trading sessions to be fired during the simulation refer to `tradingSessionsCron.properties` file found in the `data` directory. The usage/syntax of the cron expressions within that file are as follows:[a]

```
[seconds] [minutes] [hours] [day of month]
    ↪[month] [day of week] [year]
```

A few examples, provided in `tradingSessionsCron.properties`, are shown on the right. The behaviour and types of trading sessins can be modified by referring to the scripts in the `crossing/tradingSessions` folder in the `MatchingEngine` module.

```
TRADING_SESSIONS=ContinuousTrading2

#StartOfTrading.name=ContinuousTrading
#StartOfTrading.cron=0 0 7 * * 1-7

#ContinuousTrading.name=ContinuousTrading
#ContinuousTrading.cron=0 0 9 * * 1-7

#IntraDayAuctionCall.name=IntraDayAuctionCall
#IntraDayAuctionCall.cron=0 0 17 * * 1-7

ContinuousTrading2.name=ContinuousTrading
ContinuousTrading2.cron=0 15 17 * * 1-7

#RandomAuction.name=VolatilityAuctionCall
#RandomAuction.cron=0 0/5 * * * 1-7
```

[a] A cron expression is a string consisting of six or seven fields, separated by white space, that describe individual details of the schedule

A.2.3 Start scripts

Each component of the system as well as other actions can be started independently via the shell scripts. The list of all the runnable shell scripts can be found in the `deploy/scripts` directory. Each shell script simply executes the Java byte code of a "main" method whose class is specified in the `build.gradle` file of the corresponding module. For example, the `startAll.sh` script starts each component consecutively (equivalent to clicking the "Start" button on the Hawkes simulation web page). Similarly, `stopAll.sh` stops all the components (equivalent to clicking the "Shut Down" button on the Hawkes simulation webpage). Furthermore, any other actions on the website can also be done from the command line. To start the Hawkes simulation for a single client and stock simply run `./startHawkesSimulation.sh 1 1`. This starts the simulation for client 1 and stock 1 by submitting the client ID and stock ID as the first and second argument, respectively.

A.2.4 Clients

The list of clients that can be activated can be found in the `data/ClientData.csv` file. Consider, for example, the first client shown below.

```
CompID,Password,NGInputURL,NGInputStreamId,NGOutputURL,NGOutputStreamId,MDGInputURL,
    ↪MDGInputStreamId,MDGOutputURL,MDGOutputStreamId,SecurityId
1,test111111,udp://localhost:5000,10,udp://localhost:5001,10,udp://localhost:5002,10,
    ↪udp://localhost:5003,10,1
```

Each client is assigned input and output URLs for both the Native Gateway and Market Data Gateway. These URLs specify the IP addresses (in this case localhost — the user's local machine) to/from which messages will be sent/received as well as the ports on which these components are listening.

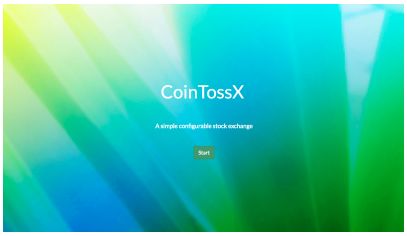
The "Trader.csv" file provides a list that cross references valid trader IDs with their mnemonics to be used in the submission of orders as identifiers for the client/agent who submitted the order. Note that multiple traders may be associated with a single client. Here the client is meant to represent a firm or group of traders submitting orders from the same gateway.

Any market data received by clients can be modified by referring to the `gateway` folder in the `MatchingEngineClient` module and the `data` folder in the `MatchingEngine` module.

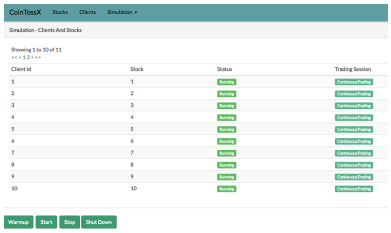
A.2.5 Website

From the website the user can: view/add/edit/delete clients, view the simulation status of clients and stocks, start the Hawkes simulation, edit the Hawkes simulation parameters, view/extract Hawkes simulation data as well as view a snapshot of the limit order books of each stock. With regards to the output of results, after all clients have submitted an end of trading session message, the orders along with their submission times are written to file and stored in the `deploy/data` directory. At the end of each Hawkes simulation the HdrHistogram latency results are written to a text file in the same directory.

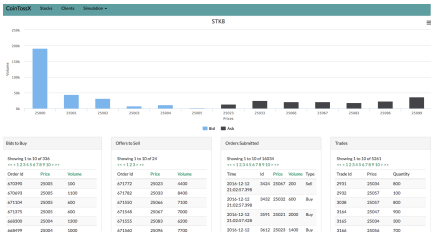
During simulations the `OffHeapStorage.java` script in the `Common` module controls the storage of orders submitted, transactions and other data that is no longer required in memory. The `LimitOrderBook` module implements the data structures required for efficient storage data kept in memory.



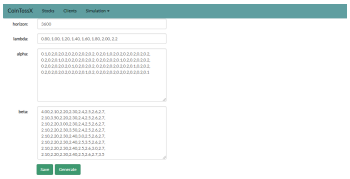
(A) Splash screen



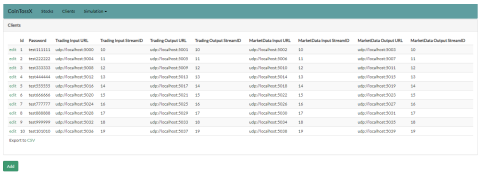
(B) All stocks and clients page snapshot



(C) Simulation page snapshot



(D) Hawkes configuration page snapshot



(E) Client page snapshot

FIGURE A.1: CoinTossX website

A.2.6 Submitting orders

The subsections below demonstrate the submission of orders to the trading gateway (on a user's local machine) in three programming languages: Java, Julia and Python. Note that currently CoinTossX only allows for Java implementations. Nonetheless, most programming languages include packages/libraries which provide interfaces for calling Java. Although it may be considered sub-optimal, this solution allows us to bypass some problems with a Java-only implementation without the need for a software shim. Even though the message interface is port based, and hence it should not matter what language or interface is used, to submit orders the choices made here are convenient as they still allow the separation of the matching engine for the components that may generate orders.

The below programming languages are only a few of those that provide these types of libraries. The code snippets below can be found in the CoinTossX project under the directory texttttClientSimulator/src/main/java/example. More specifically, all necessary static Java methods are defined in the `Utilities` class. Clients have the following functionality that will be demonstrated in the code snippets:

1. Submit the order combinations shown in table4.3.
2. Cancel orders.
3. Modify orders
4. Market data updates[1]
   (a) VWAP of buy/sell side of LOB.
   (b) Current best bid/ask price/quantity.
   (c) Active trading session.
   (d) Continuous order confirmation and execution feed.

Java

As opposed to the other implementations below, the Java implementation is shown in full without reference to the `Utilities` class to provide more detail for how the other implementations function.

[1]The information contained within market data updates are easily customizable.

```
/*
Example:
- Java version: 1.8.0_271
- Authors: Ivan Jericevich, Dharmesh Sing, Tim Gebbie
- Structure:
    1. Dependencies
    2. Supplementary methods
    3. Implementation
        - Login and start session
        - Submit orders
        - Market data updates
        - End session and logout
*/
//--------------------------------------------------------------------------------------------

//----- Import the necessary dependencies -----//
package example;

import client.*;
import sbe.msg.*;
import java.util.Properties;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import java.time.LocalDateTime;
import com.carrotsearch.hppc.IntObjectMap;
//--------------------------------------------------------------------------------------------

public class Example {
    private static final String PROPERTIES_FILE =  "simulation.properties"; // The name of the
        ↪file specifying the simulation configuration
    private static Properties properties = new Properties();

    //----- Supplementary method for extracting the simulation settings -----//
    private static void loadProperties(Properties properties, String propertiesFile) throws
        ↪IOException {
        try (InputStream inputStream =
            ↪Example.class.getClassLoader().getResourceAsStream(propertiesFile)) {
            if (inputStream != null) {
                properties.load(inputStream);
            } else {
                throw new IOException("Unable to load properties file " + propertiesFile);
            }
        }
    }
    //--------------------------------------------------------------------------------------------

    //----- Implementation -----//
    public static void main (String[] args) throws Exception {

        //----- Login and start session -----//
        int clientId = 1; int securityId = 1; // Define the client ID corresponding the client to
            ↪be logged in as well as the security ID corresponding to the security in which they
            ↪will trade
        // Load the simulation settings as well as all client data (ports, passwords and IDs)
        loadProperties(properties, PROPERTIES_FILE);
        String dataPath = properties.get("DATA_PATH").toString();
        IntObjectMap<ClientData> clientData = ClientData.loadClientDataData(dataPath);
        // Create and initialize client
        Client client = new Client(clientData.get(clientId), securityId);
        client.init(properties);
        // Start trading session by Logging in client to the gateways and connecting to ports
        client.sendStartMessage();
        System.out.println("Start at " + LocalDateTime.now());

        //----- Submit orders -----//
        // Arguments for "submitOrder": volume, price, side, order type, time in force, display
            ↪quantity, min execution size, stop price
        client.submitOrder(1000, 99, "Buy", "Limit", "Day", 1000, 0, 0); // Buy limit order
        client.submitOrder(1000, 101, "Sell", "Limit", "Day", 1000, 0, 0); // Sell limit order
        client.submitOrder(1000, 0, "Buy", "Market", "Day", 1000, 0, 0); // Buy market order
        client.submitOrder(1000, 0, "Buy", "StopLimit", "Day", 1000, 0, 0); // Stop buy limit
            ↪order
        client.submitOrder(1000, 0, "Buy", "Stop", "Day", 1000, 0, 0); // Stop buy market order
        client.cancelOrder("1", "Buy"); // Cancel limit order
```

```
69
70          //----- Market data updates -----//
71          client.calcVWAP("Buy"); // VWAP of buy side of LOB
72          client.getBid(); // Best bid price
73          client.getBidQuantity(); // Best bid volume
74          client.getOffer(); // Best ask price
75          client.getOfferQuantity(); // Best ask volume
76          client.waitForMarketDataUpdate(); // Pauses the client until a new event occurs
77          client.isAuction(); // Current trading session
78
79          //----- End trading session by logging out client and closing connections -----//
80          client.sendEndMessage();
81          client.close();
82          System.out.println("Complete at " + LocalDateTime.now());
83
84          System.exit(0);
85      }
86      //--------------------------------------------------------------------------------------------
87  }
```

Julia

The Julia-Java interface is provided in the JavaCall.jl package. This package works by specifying the paths to the compiled Java byte code (.jar files) as well as the path to the required dependencies. Thereafter the JVM is started and the utilities for logging a client in, starting the trading session, submitting an order and logging out are imported. With the execution of each command, the Java stacktrace is printed in the REPL as well.

```
1  #=
2  Example:
3  - Julia version: 1.5.3
4  - Java version: 1.8.0_271
5  - Authors: Ivan Jericevich, Dharmesh Sing, Tim Gebbie
6  - Structure:
7      1. Preliminaries
8      2. Login and start session
9      3. Submit orders
10     4. Market data updates
11     5. End session and logout
12 =#
13 #--------------------------------------------------------------------------------------------
14
15
16 #----- Preliminaries -----#
17 # Import the Java-Julia interface package
18 using JavaCall
19 cd(@__DIR__); clearconsole() # pwd()
20 # Add the path to java classes as well as the path to the ".jar" files containing the required
       ↪java dependencies
21 JavaCall.addClassPath("/home/ivanjericevich/CoinTossX/ClientSimulator/build/install/
       ↪ClientSimulator/lib/*.jar")
22 # Initialize JVM
23 JavaCall.init()
24 # Import the class containing the reqired methods
25 utilities = @jimport example.Utilities # JavaCall.listmethods(utilities)
26 #--------------------------------------------------------------------------------------------
27
28
29 #----- Login and start session -----#
30 # Define the client ID corresponding the client to be logged in as well as the security ID
       ↪corresponding to the security in which they will trade
31 clientId = 1; securityId = 1
32 # Load the simulation settings as well as all client data (ports, passwords and IDs) and
       ↪create/initialize client
33 client = jcall(utilities, "loadClientData", JavaObject{Symbol("client.Client")}, (jint, jint),
       ↪clientId, securityId)
34 # Start trading session by Logging in client to the gateways and connecting to ports
35 jcall(client, "sendStartMessage", Nothing, ())
36 #--------------------------------------------------------------------------------------------
37
38
39 #----- Submit orders -----#
```

```
# Arguments for "submitOrder": volume, price, side, order type, time in force, display quantity, min execution size, stop price
jcall(client, "submitOrder", Nothing, (jlong, jlong, JString, JString, JString, jlong, jlong, jlong), 1000, 99, "Buy", "Limit", "Day", 1000, 0, 0) # Buy limit order
jcall(client, "submitOrder", Nothing, (jlong, jlong, JString, JString, JString, jlong, jlong, jlong), 1000, 101, "Sell", "Limit", "Day", 1000, 0, 0) # Sell limit order
jcall(client, "submitOrder", Nothing, (jlong, jlong, JString, JString, JString, jlong, jlong, jlong), 1000, 0, "Buy", "Market", "Day", 1000, 0, 0) # Buy market order
jcall(client, "submitOrder", Nothing, (jlong, jlong, JString, JString, JString, jlong, jlong, jlong), 1000, 0, "Buy", "StopLimit", "Day", 1000, 0, 0) # Stop buy limit order
jcall(client, "submitOrder", Nothing, (jlong, jlong, JString, JString, JString, jlong, jlong, jlong), 1000, 0, "Buy", "Stop", "Day", 1000, 0, 0) # Stop buy market order
# Arguments for "cancelOrder": order id, side
jcall(client, "cancelOrder", Nothing, (jlong, jlong), "1", "Buy") # Cancel limit order
#-----------------------------------------------------------------------------------------


#----- Market data updates -----#
jcall(client, "calcVWAP", jlong, (JString,), "Buy") # VWAP of buy side of LOB
jcall(client, "getBid", jlong, ()) # Best bid price
jcall(client, "getBidQuantity", jlong, ()) # Best bid volume
jcall(client, "getOffer", jlong, ()) # Best ask price
jcall(client, "getOfferQuantity", jlong, ()) # Best ask volume
jcall(client, "waitForMarketDataUpdate", Nothing, ()) # Pauses the client until a new event occurs
jcall(client, "isAuction", jboolean, ()) # Current trading session
#-----------------------------------------------------------------------------------------


#----- End session and logout -----#
# End trading session by logging out client and closing connections
jcall(client, "sendEndMessage", Nothing, ()); jcall(client, "close", Nothing, ())
#-----------------------------------------------------------------------------------------
```

Python

The Python Java interface is provided in the JPype module. This code snippet follows a similar process as above but with different syntax. An alternative to using this Java wrapper class would be to directly interface Aeron using the aeron-python library to send and receive messages directly. The latter method is likely to allow for faster computation.

```
'''
Example:
- Python version: 3.8.5
- Java version: 1.8.0_271
- Authors: Ivan Jericevich, Dharmesh Sing, Tim Gebbie
- Structure:
    1. Preliminaries
    2. Login and start session
    3. Submit orders
    4. Market data updates
    5. End session and logout
'''
#-----------------------------------------------------------------------------------------


#----- Preliminaries -----#
# Import the Java-Python interface module
import jpype as jp
# Initialize/start the JVM
jp.startJVM(jp.getDefaultJVMPath(), "-ea", classpath = "/home/ivanjericevich/CoinTossX/ClientSimulator/build/classes/main") # Start JVM
# Add the path to the ".jar" files containing the required java dependencies
jpype.addClassPath("/home/ivanjericevich/CoinTossX/ClientSimulator/build/install/ClientSimulator/lib/*.jar")
# Import the class containing the reqired methods
utilities = jp.JClass("example.Utilities")
#-----------------------------------------------------------------------------------------


#----- Login and start session -----#
# Define the client ID corresponding the client to be logged in as well as the security ID corresponding to the security in which they will trade
clientId = 1
securityId = 1
```

```
# Load the simulation settings as well as all client data (ports, passwords and IDs) and create/initialize client
client = utilities.loadClientData(clientId, securityId)
# Start trading session by Logging in client to the gateways and connecting to ports
client.sendStartMessage()
#-------------------------------------------------------------------------------------------


#----- Submit orders -----#
# Arguments for "submitOrder": volume, price, side, order type, time in force, display quantity, min execution size, stop price
client.submitOrder(1000, 99, "Buy", "Limit", "Day", 1000, 0, 0) # Buy limit order
client.submitOrder(1000, 101, "Sell", "Limit", "Day", 1000, 0, 0) # Sell limit order
client.submitOrder(1000, 0, "Buy", "Market", "Day", 1000, 0, 0) # Buy market order
client.submitOrder(1000, 0, "Buy", "StopLimit", "Day", 1000, 0, 0) # Stop buy limit order
client.submitOrder(1000, 0, "Buy", "Stop", "Day", 1000, 0, 0) # Stop buy market order
# Arguments for "cancelOrder": order id, side
client.cancelOrder("1", "Buy") # Cancel limit order
#-------------------------------------------------------------------------------------------


#----- Market data updates -----#
client.calcVWAP("Buy") # VWAP of buy side of LOB
client.getBid() # Best bid price
client.getBidQuantity() # Best bid volume
client.getOffer() # Best ask price
client.getOfferQuantity() # Best ask volume
client.waitForMarketDataUpdate() # Pauses the client until a new event occurs
client.isAuction() # Current trading session
#-------------------------------------------------------------------------------------------


#----- End session and logout -----#
# End trading session by logging out client and closing connections
client.sendEndMessage()
client.close()
# Stop JVM
jp.shutdownJVM()
#-------------------------------------------------------------------------------------------
```

B Psuedocode

B.1 Point process model implementation

Algorithm 2: Basic logic for the submission of Hawkes-generated orders to CoinTossX

Input: Event (i) relative times t_i, volume ω_i, type, classification, and side generated from a 10-variate Hawkes process

Login a single client to CoinTossX to trade in a single security
Initialize the previous best bid $b(t_0)$ and best ask $a(t_0)$ at a price level of 1000 where $t_0 = 0$
Convert all order submission relative times to clock time (t): $t_i \; \forall \; i \in \{1, \ldots, N\}$
Submit the first order at a starting price of 1000
foreach *Order x occuring at time t_i with $i \in 2, \ldots, N$* **do**
    Retrieve the best bid $b(t_i)$ and best ask $a(t_i)$ from the market data feed
    if *x is a limit order* **then**
        if *The LOB is empty ($(L)(t) = \emptyset$): $b(t_i) == 0$ and $a(t_i) == 0$* **then**
            if *x is a bid* **then** Place the limit price p_x according to the rules in Algorithm 3 or 4 with input parameters 0 for the best bid and $a(t_{i-1})$ for the previous best ask
            else Place the limit price p_x according to the rules in Algorithm 3 or 4 with input parameters $b(t_{i-1})$ for the previous best bid and 0 for the previous best ask
        else
            Place the limit price p_x according to the rules in Algorithm 3 or 4 with input parameters $b(t_i)$ for the best bid and $a(t_i)$ for the best ask
        end
        if *If the current time is less than the arrival time of x: $t < t_i$* **then** Wait for the remaining duration until the arrival time to submit the limit order at price p_x with volume ω_x
        else The algorithm is lagging behind the times specified by the Hawkes process, so submit the limit order at price p_x with volume ω_x immediately
    else if *event i is a market order and the contra side is non-empty* **then**
        if *If the current time is less than the arrival time of order$_{t_i}$: $t < t_i$* **then** Wait for the remaining duration until the arrival time to submit the market order with volume v_{t_i}
        else The algorithm is lagging behind the times specified by the Hawkes process, so submit the market order with volume v_{t_i} immediately
    else if *Order x is a cancel order and the corresponding side of the LOB is non-empty* **then**
        Retrieve a LOB snapshot from the market data feed
        if *x is an aggressive order* **then**
            Extract the order IDs from the corresponding LOB side that are active at level 1: $\mathcal{L}^1(t_i)$
            Select an order x by randomly sampling from $\mathcal{L}^1(t_i)$
        else
            Extract the order IDs from the corresponding LOB that are active at levels > 2: $\mathcal{L}^{2+}(t_i)$
            Select an order x by randomly sampling from $\mathcal{L}^{2+}(t_i)$
        end
        if *If the current time is less than the arrival time of order$_{t_i}$: $t < t_i$* **then** Wait for the remaining duration until the arrival time to cancel order x
        else The algorithm is lagging behind the times specified by the Hawkes process, so cancel order o immediately
end
Logout the client from CoinTossX

Algorithm 3: Limit price placement in model 1

Input: The best bid $b(t_{i-1})$ prior to the submission of the ith event x, the best ask $a(t_{i-1})$ prior to the submission of the ith event, x

Result: Limit price for the ith order p_x

Randomly sample $u \in \{1, \ldots, 10\}$ with equal probability
if *x is a bid* **then**
  if *x is passive* **then**
    if *Contra side is empty* $a(t_{i-1}) = \emptyset$ **then**
      $p_x = b(t_{i-1}) - 1$
    else
      if *The bid side of the LOB is empty* $b(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = a(t_{i-1}) - u$
      else $p_x = b(t_{i-1}) - 1$
    end
  else
    if *The bid side of the LOB is empty* $b(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = a(t_{i-1}) - u$
    else Improve the best by 1 tick $p_x = b(t_{i-1}) + 1$
  end
else
  if *x is passive* **then**
    if *Contra side is empty* $b(t_{i-1}) = \emptyset$ **then**
      $p_x = a(t_{i-1}) + 1$
    else
      if *The ask side of the LOB is empty* $a(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = b(t_{i-1}) + u$
      else $p_x = a(t_{i-1}) + 1$
    end
  else
    if *The ask side of the LOB is empty* $a(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = b(t_{i-1}) + u$
    else Improve the best by 1 tick $p_x = a(t_{i-1}) - 1$
  end
end

Algorithm 4: Limit price placement in model 2

Input: The best bid $b(t_{i-1})$ prior to the submission of the ith event, the best ask $a(t_{i-1})$ prior to the submission of the ith event, x

Result: Limit price for the ith order p_x

Randomly sample $u \in \{1, \dots, 10\}$ with equal probability
if *x is a bid* **then**
  if *x is passive* **then**
    if *Contra side is empty* $a(t_{i-1}) = \emptyset$ **then**
      $p_x = b(t_{i-1}) - 1$
    else
      if *The bid side of the LOB is empty* $b(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = a(t_{i-1}) - u$
      else $p_x = b(t_{i-1}) - 1$
    end
  else
    if *The bid side of the LOB is empty* $b(t_{i-1}) = \emptyset$ **then**
      Place the price a random number of ticks opposite the contra-best $p_x = a(t_{i-1}) - u$
    else
      Calculate the spread as $s(t_{i-1}) = |a(t_{i-1}) - b(t_{i-1})|$
      if *The spread is greater than 1 tick* $s_i > 1$ **then** Improve the best by 1 tick $p_x = b(t_{i-1}) + 1$
      else Place the price at the best $p_x = b(t_{i-1})$
    end
  end
else
  if *$event_i$ is passive* **then**
    if *Contra side is empty* $b(t_{i-1}) = \emptyset$ **then**
      $p_x = a(t_{i-1}) + 1$
    else
      if *The ask side of the LOB is empty* $a(t_{i-1}) = \emptyset$ **then** Place the price a random number of ticks opposite the contra-best $p_x = b(t_{i-1}) + u$
      else $p_x = a(t_{i-1}) + 1$
    end
  else
    if *The ask side of the LOB is empty* $a(t_{i-1}) = \emptyset$ **then**
      Place the price a random number of ticks opposite the contra-best $p_x = b(t_{i-1}) + u$
    else
      Calculate the spread as $s(t_{i-1}) = |a(t_{i-1}) - b(t_{i-1})|$
      if *The spread is greater than 1 tick* $s_i > 1$ **then** Improve the best by 1 tick $p_x = a(t_{i-1}) - 1$
      else Place the price at the best $p_x = a(t_{i-1})$
    end
  end
end

B.1.1 Reproducibility

CoinTossX v1.1.0 was used in this work. All code and instructions for replication can be found at: https://github.com/IvanJericevich/IJPCTG-HawkesCoinTossX.

B.2 Agent-based model implementation

Algorithm 5: Algorithm for updating the state of the LOB with the latest execution message from CoinTossX

Input: A confirmation or execution report string message from CoinTossX
Output: The updated LOB state $\mathcal{L}(t)$

Extract order type, price, side, volume and trader identification fields
if *A new limit order has been confirmed* **then**
  if *Buy limit order* **then** Add limit order x_{t_i} to $\mathcal{B}(t)$
  else Add limit order x_{t_i} to $\mathcal{A}(t)$
else if *A trade has been executed* **then**
  if *Buy market order* **then** Remove the executed volume from the limit order having the corresponding ID in $\mathcal{A}(t)$
  else Remove the executed volume from the limit order having the corresponding ID in $\mathcal{B}(t)$
else if *A limit order has been cancelled* **then**
  if *Buy limit order* **then** Remove the limit order having the corresponding ID from $\mathcal{B}(t)$
  else Buy limit order
  Remove the limit order having the corresponding ID from $\mathcal{A}(t)$
end
if $\mathcal{A}(t) \neq \emptyset$ *and* $\mathcal{B}(t) \neq \emptyset$ **then**
  Update $a(t)$ and $b(t)$ with the current updated LOB state $\mathcal{L}(t)$
else
  if $\mathcal{A}(t) \neq \emptyset$ **then** Update $a(t)$ with the current updated LOB state $\mathcal{L}(t)$
  if $\mathcal{B}(t) \neq \emptyset$ **then** Update $b(t)$ with the current updated LOB state $\mathcal{L}(t)$
end
Update $s(t)$, $m(t)$ and $\rho(t)$ with the current or historic LOB state

Algorithm 6: Fundamentalist agent action algorithm

Input: LOB state $\mathcal{L}_t$, agent parameters f_t and simulation parameters $\boldsymbol{\theta}$
Output: Market order $x_i = (0, \omega_{x_i}, t_{x_i})$

if $(m(t) - f_t) > \delta m(t)$ **then** $x_m = 50$
else $x_m = 20$
if $f_t < m(t)$ **then**
  x_i is a sell order
  $\alpha = 1 - \frac{\rho(t)}{\nu}$
else
  x_i is a buy order
  $\alpha = 1 + \frac{\rho(t)}{\nu}$
end
if *The contra side of LOB* $\mathcal{L}_t$ *is non-empty* **then** Submit market order x_i with volume $\omega_{x_i} \sim f(x; 10, \alpha)$ Power Law distribution with tail index parameter α (6.8)

B.2.1 Reproducibility

The specific version of CoinTossX used can be found at: https://github.com/IvanJericevich/CoinTossX/tree/abm-experiment. Code and instructions for replication can be found at: https://github.com/IvanJericevich/IJPCTG-ABMCoinTossX.

Algorithm 7: Liquidity provider agent action algorithm

Input: LOB state $\mathcal{L}_t$ and simulation parameters $\boldsymbol{\theta}$
Output: Limit order $x_i = (p_{x_i}, \omega_{x_i}, t_{x_i})$

Sample $u \sim U(0,1)$
if $u < (\rho(t)+1)/2$ **then** x_i is a sell order
else x_i is a buy order
if *x_i is a sell order* **then**
    $\alpha = 1 - (\rho(t)/\nu)$
    Sample $\eta \sim \Gamma\left(s(t), e^{\frac{\rho(t)}{\nu}}\right)$
    Set the limit price as $p_{x_i} = b(t) + 1 - \lceil \eta \rceil$
    Set the volume as $\omega_{x_i} \sim$ Power Law distribution with tail index parameter α
else
    $\alpha = 1 + (\rho(t)/\nu)$
    Sample $\eta \sim \Gamma\left(s(t), e^{-\frac{\rho(t)}{\nu}}\right)$
    Set the limit price as $p_{x_i} = b(t) - 1 - \lceil \eta \rceil$
    Set the volume as $\omega_{x_i} \sim f(x; 10, \alpha)$ Power Law distribution with tail index parameter α (6.10)
end

Algorithm 8: Chartist agent action algorithm

Input: LOB state $\mathcal{L}_t$, agent parameters τ and $\overline{p}_t$, and simulation parameters $\boldsymbol{\theta}$
Output: Market order $x_i = (0, \omega_{x_i}, t_{x_i})$

Compute the agent's decision inter-arrival time Δt
$\lambda = 1 - e^{-\Delta t/\tau}$
Set the agent's EWMA $\overline{p}_t = \lambda(m(t) - \overline{p}_t)$
if $(m(t) - \overline{p}_t) > \delta m(t)$ **then** $x_m = 50$
else $x_m = 20$
if $m(t) < \overline{p}_t$ **then**
    x_i is a sell order
    $\alpha = 1 - \frac{\rho(t)}{\nu}$
else
    x_i is a buy order
    $\alpha = 1 + \frac{\rho(t)}{\nu}$
end
if *The contra side of LOB $\mathcal{L}_t$ is non-empty* **then** Submit market order x_i with volume $\omega_{x_i} \sim f(x; x_m, \alpha)$ Power Law distribution with tail index parameter α (6.8)

Algorithm 9: Basic logic for the submission of orders to CoinTossX

Input: Agent and simulation parameters $\boldsymbol{\theta}$

Initialise agents with their submission times, order types and order IDs
Set the initial LOB state $\mathcal{L}(0)$ with values for $s(0)$, $a(0)$, $b(0)$, $m(0)$
Login a single client to CoinTossX to trade in a single security
Open the market data feed socket by connecting to the correct port and IP address for the receival of UDP message packets
Convert agent relative decision times to calendar time
foreach $x_i = (p_{x_i}, \omega_{x_i}, t_{x_i})$ *with* $i \in 1, \ldots, T$ **do**
    if t_{x_i} *is the liquidity providers' decision time* **then**
        if x_i *is a limit order* **then**
            Set order fields p_{x_i} and ω_{x_i} according the liquidity provider decision rules
        else
            Cancel the limit order having the current order's pre-assigned order ID
        end
    else if t_{x_i} *is the liquidity providers' decision time* **then**
        if *The static price reference does not deviate from the bes by more than 10%* **then**
            if *Agent is a chartist* **then**
                Set order fields p_{x_i} and ω_{x_i} according the chartist decision rules
            else if *Agent is a fundamentalist* **then**
                Set order fields p_{x_i} and ω_{x_i} according the fundamentalist decision rules
            end
        end
    end
    Receive UDP confirmation/execution message from CoinTossX that corresponds to the current order submitted
    Update the LOB state $\mathcal{L}(t)$
end
Disconnect from the market data feed port
Log out from CoinTossX

C Sensitivity analysis results

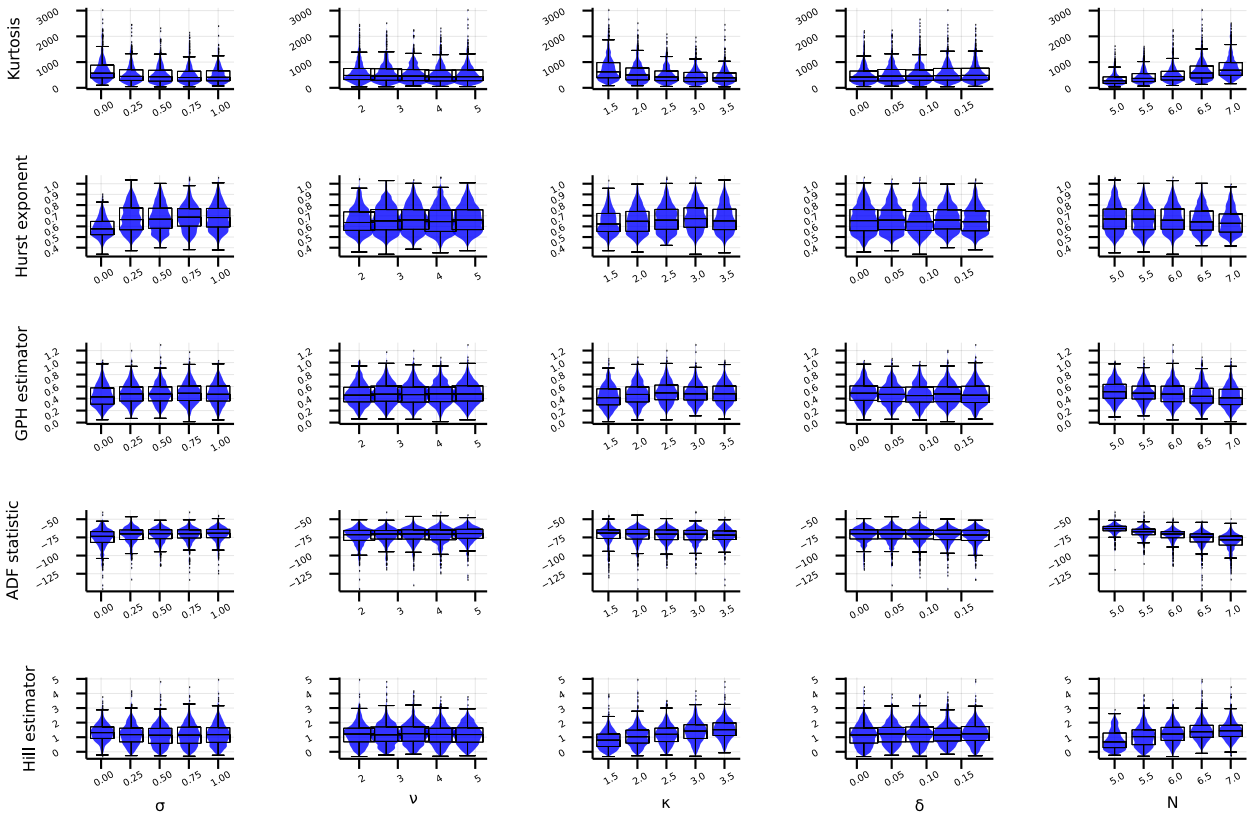


FIGURE C.1: Mid-Price Moments: Summary statistics box-plots for each individual parameter-moment combination calculated on both simulated mid-price time-series. Columns represent individual moments while rows represent individual parameters. The vertical axis of each plot gives the values of the moments and the horizontal axis gives the unique parameter values chosen.

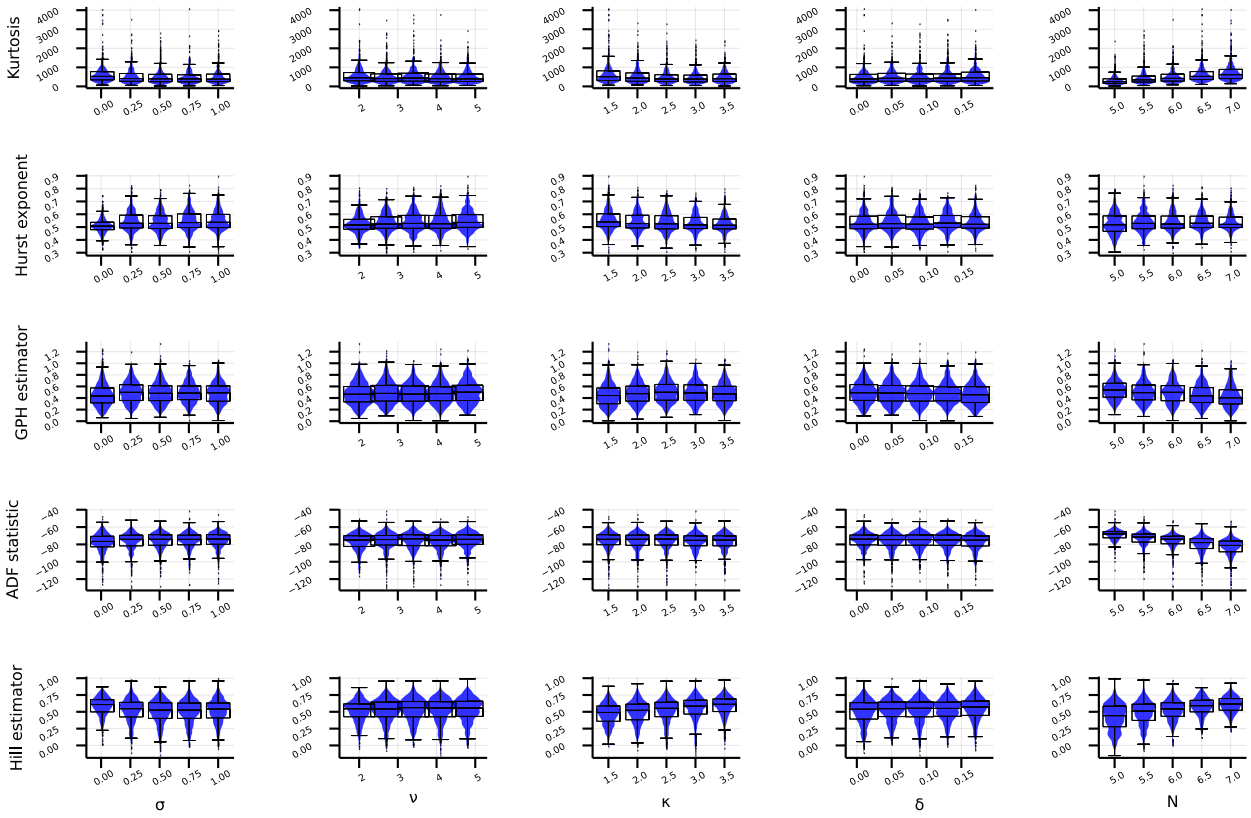
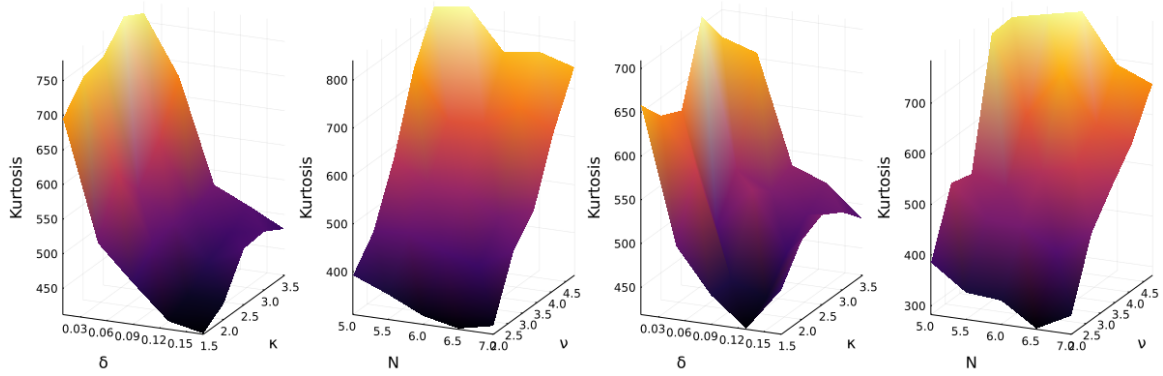
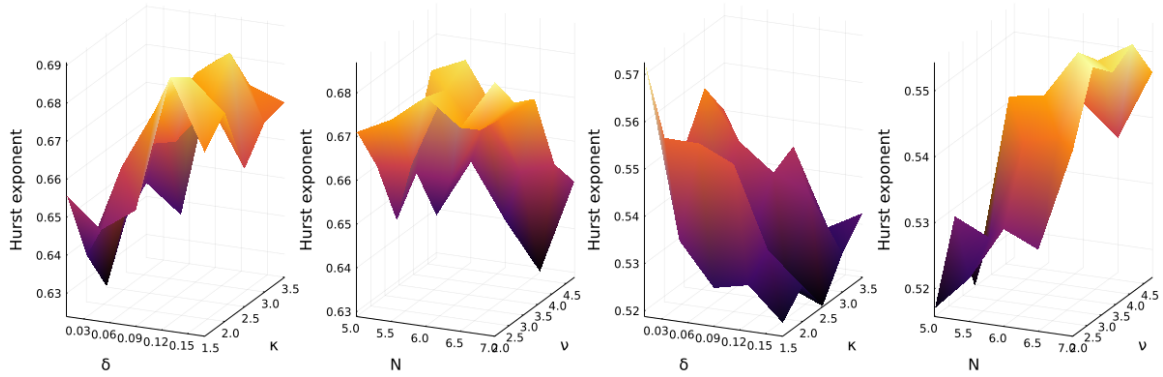


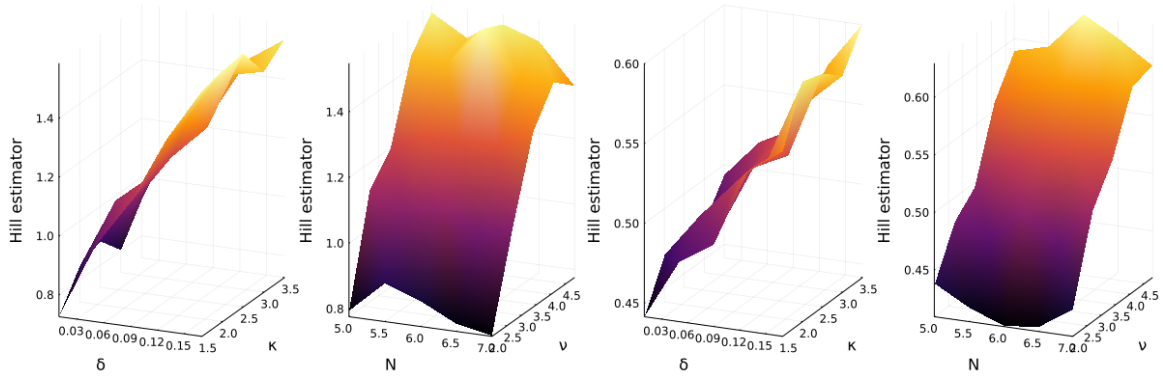
FIGURE C.2: Micro-price Moments: Summary statistics box-plots for each individual parameter-moment combination calculated on both simulated micro-price. Columns represent individual moments while rows represent individual parameters. The vertical axis of each plot gives the values of the moments and the horizontal axis gives the unique parameter values chosen.



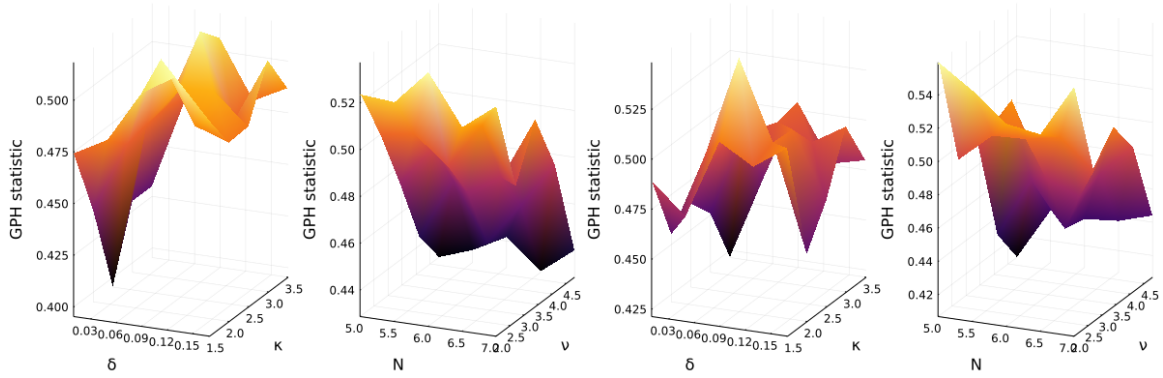
(A) Mid-price Kurtosis surfaces for δ & κ and N & ν (B) Micro-price Kurtosis surfaces for δ & κ and N & ν



(C) Mid-price Hurst exponent surfaces for δ & κ and N & ν (D) Micro-price Hurst exponent surfaces for δ & κ and N & ν



(E) Mid-price Hill estimator surfaces for δ & κ and N & ν (F) Micro-price Hill estimator surfaces for δ & κ and N & ν



(G) Mid-price GPH statistic surfaces for δ & κ and N & ν (H) Micro-price GPH statistic surfaces for δ & κ and N & ν

FIGURE C.3: Moment surfaces for pairwise combinations of parameters calculated on simulated log-returns and averaged over all combinations of other parameters

Bibliography

Addison, Andrew et al. (2019). "Low-Latency Trading in the Cloud Environment". In: *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, pp. 272–282. DOI: 10.1109/CSE/EUC.2019.00060. URL: https://ieeexplore.ieee.org/document/8919600.

Avellaneda, Marco and Sasha Stoikov (2008). "High-frequency trading in a limit order book". In: *Quantitative Finance* 8.3, pp. 217–224. DOI: 10.1080/14697680701381228.

Bacry, Emmanuel, Iacopo Mastromatteo, and Jean-François Muzy (2015). "Hawkes processes in finance". In: *Market Microstructure and Liquidity* 1.01, p. 1550005. DOI: 10.1142/S2382626615500057.

Bacry, Emmanuel and Jean-François Muzy (2014). "Hawkes model for price and trades high-frequency dynamics". In: *Quantitative Finance* 14.7, pp. 1147–1166. DOI: 10.1080/14697688.2014.897000.

Balch, Tucker, Manuela Veloso, and Danilo Mandic (2021). *Learning to Classify and Imitate Trading Agents in Continuous Double Auction Markets*. arXiv: 2110.01325 [q-fin.TR].

Bernoulli, Daniel (1954). "Exposition of a New Theory on the Measurement of Risk". In: *Econometrica* 22.1, pp. 23–36. ISSN: 00129682, 14680262. DOI: 10.2307/1909829. URL: http://www.jstor.org/stable/1909829.

Biais, Bruno, Pierre Hillion, and Chester Spatt (1995). "An empirical analysis of the limit order book and the order flow in the Paris Bourse". In: *the Journal of Finance* 50.5, pp. 1655–1689. ISSN: 00221082, 15406261. DOI: 10.2307/2329330.

Böhm, Volker and Carl Chiarella (2005). "Mean variance preferences, expectations formation, and the dynamics of random asset prices". In: *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics* 15.1, pp. 61–97. DOI: 10.1111/j.0960-1627.2005.00211.x.

Bonabeau, Eric (2002). "Agent-based modeling: Methods and techniques for simulating human systems". In: *Proceedings of the National Academy of Sciences* 99.suppl 3, pp. 7280–7287. DOI: 10.1073/pnas.082080899. eprint: https://www.pnas.org/content/99/suppl_3/7280.full.pdf. URL: https://www.pnas.org/content/99/suppl_3/7280.

Bouchaud, Jean-Philippe, Marc Mézard, and Marc Potters (2002). "Statistical properties of stock order books: empirical results and models". In: *Quantitative finance* 2, pp. 251–256. DOI: 10.1088/1469-7688/2/4/301.

Bowsher, Clive G (2007). "Modelling security market events in continuous time: Intensity based, multivariate point process models". In: *Journal of Econometrics* 141.2, pp. 876–912. URL: https://EconPapers.repec.org/RePEc:eee:econom:v:141:y:2007:i:2:p:876-912.

Brandouy, Olivier, Philippe Mathieu, and Iryna Veryzhenko (2013). "On the design of agent-based artificial stock markets". In: *Agents and Artificial Intelligence*. Springer, pp. 350–364. ISBN: 978-3-642-29966-7. DOI: 10.1007/978-3-642-29966-7_23.

Cartea, Álvaro, Sebastian Jaimungal, and José Penalva (Aug. 2015). *Algorithmic and high-frequency trading*. Cambridge University Press. ISBN: 9781107091146.

Chakraborti, Anirban et al. (2011a). "Econophysics review: I. Empirical facts". In: *Quantitative Finance* 11.7, pp. 991–1012. DOI: 10.1080/14697688.2010.539248.

– (2011b). "Econophysics review: II. Agent-based models". In: *Quantitative Finance* 11.7, pp. 1013–1041. DOI: 10.1080/14697688.2010.539249.

Chakraborty, Tanmoy and Michael Kearns (2011). "Market making and mean reversion". In: *Proceedings of the 12th ACM conference on Electronic commerce*, pp. 307–314. DOI: 10.1145/1993574.1993622.

Chang, Patrick, Etienne Pienaar, and Tim Gebbie (2020). *The Epps effect under alternative sampling schemes*. arXiv: 2011.11281 [q-fin.ST]. URL: https://arxiv.org/abs/2011.11281.

Chiarella, Carl, Roberto Dieci, and Xue-Zhong He (2009). "Heterogeneity, market mechanisms, and asset price dynamics". In: *Handbook of financial markets: Dynamics and evolution*. Ed. by Thorsten Hens and Klaus Reiner Schenk-Hoppé, pp. 277–344. ISSN: 15684997. DOI: 10.1016/B978-012374258-2.50009-9. URL: https://www.sciencedirect.com/science/article/pii/B9780123742582500099.

Cont, Rama (2001). "Empirical properties of asset returns: stylized facts and statistical issues". In: *Quantitative Finance* 1, pp. 223–236. DOI: 10.1080/713665670.

Crafa, Silvia (2021). "From agent-based modeling to actor-based reactive systems in the analysis of financial networks". In: *Journal of Economic Interaction and Coordination* 16.3, pp. 649–673. DOI: 10.1007/s11403-021-00323-8. URL: https://doi.org/10.1007/s11403-021-00323-8.

Da Fonseca, José and Riadh Zaatour (2014). "Hawkes process: Fast calibration, application to trade clustering, and diffusive limit". In: *Journal of Futures Markets* 34.6, pp. 548–579. DOI: 10.1002/fut.21644. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/fut.21644.

Daley, Daryl J and David Vere-Jones (2003). *An introduction to the theory of point processes: Elementary theory and methods*. 2nd ed. Vol. 1. Springer, pp. 1–471. ISBN: 978-0-387-21564-8. DOI: 10.1007/b97277. URL: https://www.springer.com/gp/book/9780387955414.

Das, Sanmay (2005). "A learning market-maker in the Glosten–Milgrom model". In: *Quantitative Finance* 5.2, pp. 169–180. DOI: 10.1080/14697680500148067.

Dassios, Angelos, Hongbiao Zhao, et al. (2013). "Exact simulation of Hawkes process with exponentially decaying intensity". In: *Electronic Communications in Probability* 18. DOI: 10.1214/ECP.v18-2717. URL: https://projecteuclid.org/download/pdf_1/euclid.ecp/1465315601.

Dieci, Roberto and Xue-Zhong He (2018). "Heterogeneous agent models in finance". In: *Handbook of computational economics* 4. Ed. by Cars Hommes and Blake LeBaron, pp. 257–328. ISSN: 1574-0021. DOI: 10.1016/bs.hescom.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S157400211830008X.

Efron, Bradley and David V Hinkley (1978). "Assessing the accuracy of the maximum likelihood estimator: Observed versus expected Fisher information". In: *Biometrika* 65.3, pp. 457–483.

Epps, Thomas W (1979). "Comovements in stock prices in the very short run". In: *Journal of the American Statistical Association* 74.366a, pp. 291–298. DOI: 10.1080/01621459.1979.10482508.

Farmer, J Doyne, Paolo Patelli, and Ilija I Zovko (2005). "The predictive power of zero intelligence in financial markets". In: *Proceedings of the National Academy of Sciences* 102.6, pp. 2254–2259. ISSN: 0027-8424. DOI: 10.1073/pnas.0409157102. URL: https://www.pnas.org/content/102/6/2254.

Freedman, David A (2007). "How can the score test be inconsistent?" In: *The American Statistician* 61.4, pp. 291–295.

Gao, Fuchang and Lixing Han (2012). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51.1, pp. 259–277. DOI: 10.1007/s10589-010-9329-3.

Garman, Mark B. (1976). "Market microstructure". In: *Journal of financial Economics* 3.3, pp. 257–275. ISSN: 0304-405X. DOI: 10.1016/0304-405X(76)90006-4. URL: https://www.sciencedirect.com/science/article/pii/0304405X76900064.

Geweke, John and Susan Porter-Hudak (1983). "The estimation and application of long memory time series models". In: *Journal of time series analysis* 4.4, pp. 221–238. DOI: 10.1111/j.1467-9892.1983.tb00371.x.

Gilli, Manfred and Peter Winker (2003). "A Global Optimization Heuristic for Estimating Agent Based Models". In: *Computational Statistics & Data Analysis* 42.3, pp. 299–312.

Glosten, Lawrence R. and Paul R. Milgrom (1985). "Bid, ask and transaction prices in a specialist market with heterogeneously informed traders". In: *Journal of Financial Economics* 14.1, pp. 71–100. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(85)90044-3.

Gode, Dhananjay K and Shyam Sunder (1993). "Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality". In: *Journal of political economy* 101.1, pp. 119–137. ISSN: 00223808, 1537534X. DOI: 10.2307/2138676. URL: http://www.jstor.org/stable/2138676.

Goosen, Kelly and Tim Gebbie (2020). "Calibrating high-frequency trading data to agent-based models using approximate bayesian computation". MA thesis. University of cape town. DOI: 10.25375/uct.12894005.v1. URL: https://github.com/KellyGoosen1/hft-abm-smc-abc.

Gould, Martin D et al. (2013). "Limit order books". In: *Quantitative Finance* 13.11, pp. 1709–1742. DOI: 10.1080/14697688.2013.803148.

Harvey, Michael et al. (Feb. 2016). "Deviations in expected price impact for small transaction volumes under fee restructuring". In: *Physica A: Statistical Mechanics and its Applications* 471, pp. 416–426. DOI: 10.1016/j.physa.2016.11.042.

Hasbrouck, Joel (2007). *Empirical market microstructure: The institutions, economics, and econometrics of securities trading*. Oxford University Press.

Hautsch, Nikolaus (2011). *Econometrics of financial high-frequency data*. Springer Science & Business Media, pp. 1–371. DOI: 10.1007/978-3-642-21925-2.

Hawkes, Alan G (1971). "Spectra of some self-exciting and mutually exciting point processes". In: *Biometrika* 58.1, pp. 83–90. DOI: 10.2307/2334319. URL: https://www.jstor.org/stable/2334319.

Hawkes, Alan G and David Oakes (1974). "A cluster process representation of a self-exciting process". In: *Journal of Applied Probability* 11.3, pp. 493–503. DOI: 10.2307/3212693. URL: http://www.jstor.org/stable/3212693.

Hewitt, Carl (2011). *Actor Model of Computation: Scalable Robust Information Systems*. arXiv: 1008.1459 [cs.PL].

Hewitt, Carl, Peter Bishop, and Richard Steiger (1973). "A universal modular actor formalism for artificial intelligence". In: *Advance Papers of the Conference*. Vol. 3. Stanford Research Institute Menlo Park, CA, p. 235.

Hommes, Cars H (2006). "Heterogeneous agent models in economics and finance". In: *Handbook of computational economics* 2. Ed. by L. Tesfatsion and K.L. Judd, pp. 1109–1186. ISSN: 1574-0021. DOI: 10.1016/S1574-0021(05)02023-X. URL: https://www.sciencedirect.com/science/article/pii/S157400210502023X.

Iori, Giulia and James Porter (2012). "Agent-based modelling for financial markets". In: DOI: 10.1093/oxfordhb/9780199844371.013.43.

Jericevich, Ivan, Patrick Chang, and Tim Gebbie (2020). *Comparing the market microstructure between two South African exchanges*. arXiv: 2011.04367 [q-fin.ST]. URL: https://arxiv.org/abs/2011.04367.

– (2021a). *Simulation and estimation of a point-process market-model with a matching engine*. arXiv: 2105.02211 [q-fin.TR].

– (2021b). *Simulation and estimation of an agent-based market-model with a matching engine*. arXiv: 2108.07806 [q-fin.TR].

Jericevich, Ivan, Dharmesh Sing, and Tim Gebbie (2021). *CoinTossX: An open-source low-latency high-throughput matching engine*. arXiv: 2102.10925 [cs.DC].

Johannesburg-Stock-Exchange (2015). *The lowest-latency connection to JSE markets*. URL: https://www.jse.co.za/content/JSETechnologyDocumentItems/3.%5C%20JSE%5C%20Colocation%5C%20Brochure%5C%202015.pdf.

– (Apr. 2018). *Equity Market Trading and Information Solution JSE Specification Document Volume 01 — Native Trading Gateway*. 3.05. URL: https://www.jse.co.za/content/JSETechnologyDocumentItems/Volume%5C%2001%5C%20-%5C%20Native%5C%20Trading%5C%20Gateway.pdf.

– (Jan. 2019). *Equity Market Trading and Information Solution JSE Specification Document Volume 05 — Market Data Gateway*. 3.09. URL: https://www.jse.co.za/content/JSETechnologyDocumentItems/Volume%5C%2005%5C%20-%5C%20Market%5C%20Data%5C%20Gateway%5C%20(MITCH%5C%20-%5C%20UDP).pdf.

– (2020a). *Broker Deal Accounting System*. URL: https://www.jse.co.za/services/technologies/bda-technical-specifications.

– (Apr. 2020b). *Equity Market Trading and Information Solution JSE Specification Document Volume 00E — Trading and Information Overview*. 4th ed. URL: https://www.jse.co.za/content/

JSEContractSpecificationItems/Volume%5C%2000E%5C%20-%5C%20Trading%5C%20and%5C%20Information%5C%20Overview%5C%20for%5C%20Equity%5C%20Market.pdf.

Kahneman, Daniel and Amos Tversky (1979). "Prospect theory: An analysis of decision under risk". In: *Econometrica* 47.2, pp. 363–391. ISSN: 00129682, 14680262. DOI: 10.2307/1914185.

Kullmann, L et al. (1999). "Characteristic times in stock market indices". In: *Physica A: Statistical Mechanics and its Applications* 269.1, pp. 98–110. ISSN: 0378-4371. DOI: https://doi.org/10.1016/S0378-4371(99)00084-9. URL: https://www.sciencedirect.com/science/article/pii/S0378437199000849.

Kumar, P et al. (2013). "Implementing an agent based artificial stock market model in jade-an illustration". In: *International Journal of Engineering & Technology* 5, pp. 2636–2648.

Large, Jeremy (2007). "Measuring the resiliency of an electronic limit order book". In: *Journal of Financial Markets* 10.1, pp. 1–25. ISSN: 1386-4181. DOI: 10.1016/j.finmar.2006.09.001. URL: http://www.sciencedirect.com/science/article/pii/S1386418106000528.

Laub, Patrick and Pollet Phil (2014). "Hawkes processes simulation, estimation, and validation". MA thesis. University of Queensland, pp. 1–76. URL: https://pat-laub.github.io/pdfs/honours_thesis.pdf.

Leal, Sandrine J. et al. (2016). "Rock Around the Clock: An Agent-Based Model of Low-and High-Frequency Trading". In: *Journal of Evolutionary Economics* 26.1, pp. 49–76.

LeBaron, Blake (2000). "Agent-based computational finance: Suggested readings and early research". In: *Journal of Economic Dynamics and Control* 24.5, pp. 679–702. ISSN: 0165-1889. DOI: 10.1016/S0165-1889(99)00022-6. URL: https://www.sciencedirect.com/science/article/pii/S0165188999000226.

– (2002). "Building the Santa Fe artificial stock market". In: *Physica A*, pp. 1–20.

– (2006). "Agent-based computational finance". In: *Handbook of computational economics* 2. Ed. by L. Tesfatsion and K.L. Judd, pp. 1187–1233. ISSN: 1574-0021. DOI: 10.1016/S1574-0021(05)02024-1. URL: https://www.sciencedirect.com/science/article/pii/S1574002105020241.

Lee, Charles M. C. and Mark J. Ready (1991). "Inferring Trade Direction from Intraday Data". In: *The Journal of Finance* 46.2, pp. 733–746. DOI: 10.1111/j.1540-6261.1991.tb02683.x.

Lewis, PA W and Gerald S Shedler (1979). "Simulation of nonhomogeneous Poisson processes by thinning". In: *Naval research logistics quarterly* 26.3, pp. 403–413. DOI: 10.1002/nav.3800260304. URL: https://onlinelibrary.wiley.com/doi/10.1002/nav.3800260304.

Lillo, Fabrizio and J Doyne Farmer (Feb. 2007). "The Long Memory of the Efficient Market". In: *Studies in Nonlinear Dynamics & Econometrics* 8, pp. 1–1. DOI: 10.2202/1558-3708.1226.

Lillo, Fabrizio, J Doyne Farmer, and Rosario Mantegna (Feb. 2003). "Master curve for price-impact function". In: *Nature* 421, pp. 129–30. DOI: 10.1038/421129a.

Lussange, Johann et al. (Jan. 2018). "A bright future for financial agent-based models". In: *SSRN*. DOI: 10.2139/ssrn.3109904. URL: https://ssrn.com/abstract=3109904.

Mandelbrot, Benoit B. (1963). "The Variation of Certain Speculative Prices". In: *The Journal of Business* 36.4, pp. 394–419. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2350970.

Mandeş, Alexandru (2015). "Microstructure-based order placement in a continuous double auction agent based model". In: *Algorithmic Finance* 4.3-4, pp. 105–125. DOI: 10.3233/AF-150049.

Mike, Szabolcs and J Doyne Farmer (2008). "An empirical behavioral model of liquidity and volatility". In: *Journal of Economic Dynamics and Control* 32.1, pp. 200–234. DOI: 10.1016/j.jedc.2007.01.025.

Mogensen, Patrick Kofod and Asbjørn Nilsen Riseth (2018). "Optim: A mathematical optimization package for Julia". In: *Journal of Open Source Software* 3.24, p. 615. DOI: 10.21105/joss.00615.

Morgan, BJ T, Karen J Palmer, and Martin S Ridout (2007). "Negative score test statistic". In: *The American Statistician* 61.4, pp. 285–288. DOI: 10.1198/000313007X242972.

Muni Toke, Ioane and Fabrizio Pomponio (2011). "Modelling trades-through in a limited order book using hawkes processes". In: *Economics: The Open-Access, Open-Assessment E-Journal* 6.32, pp. 1–23. DOI: 10.2139/ssrn.1973856. URL: http://www.economics-ejournal.org/dataset/PDFs/journalarticles_2012-22.pdf.

Nair, Preyen (2015). "Agent based modelling of a single-stock market on the JSE". MA thesis. University of the Witwatersrand. URL: http://wiredspace.wits.ac.za/handle/10539/16840.

Nelder, John A. and Roger Mead (1965). "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4, pp. 308–313.

Nuyts, Jean (2010). "Inference about the tail of a distribution: Improvement on the hill estimator". In: *International Journal of mathematics and mathematical sciences* 2010. ISSN: 0161-1712. DOI: 10.1155/2010/924013.

O'Hara, M (1997). *Market Microstructure Theory*. Wiley.

Ogata, Yoshiko (1978). "The asymptotic behaviour of maximum likelihood estimators for stationary point processes". In: *Annals of the Institute of Statistical Mathematics* 30.2, pp. 243–261. DOI: 10.1007/BF02480216.

Ogata, Yosihiko (1981). "On Lewis' simulation method for point processes". In: *IEEE transactions on information theory* 27.1, pp. 23–31. DOI: 10.1109/TIT.1981.1056305. URL: https://ieeexplore.ieee.org/document/1056305.

– (1988). "Statistical models for earthquake occurrences and residual analysis for point processes". In: *Journal of the American Statistical association* 83.401, pp. 9–27. ISSN: 01621459. DOI: 10.1080/01621459.1988.10478560. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1988.10478560.

Ogata, Yosihiko and Hirotugu Akaike (1982). "On linear intensity models for mixed doubly stochastic Poisson and self-exciting point processes". In: *Journal of the Royal Statistical Society. Series B*, pp. 269–274. DOI: 10.1007/978-1-4612-1694-0_20.

Ozaki, Tohru (1979). "Maximum likelihood estimation of Hawkes' self-exciting point processes". In: *Annals of the Institute of Statistical Mathematics* 31.1, pp. 145–155. DOI: 10.1007/BF02480272.

Pagan, Adrian (1996). "The econometrics of financial markets". In: *Journal of empirical finance* 3.1, pp. 15–102. ISSN: 0927-5398. DOI: 10.1016/0927-5398(95)00020-8. URL: https://www.sciencedirect.com/science/article/pii/0927539895000208.

Platt, Donovan (2020). "A comparison of economic agent-based model calibration methods". In: *Journal of Economic Dynamics and Control* 113, p. 103859. ISSN: 0165-1889. DOI: 10.1016/j.jedc.2020.103859. URL: https://www.sciencedirect.com/science/article/pii/S0165188920300294.

Platt, Donovan and Tim Gebbie (2017). *The Problem of Calibrating an Agent-Based Model of High-Frequency Trading*. arXiv: 1606.01495 [q-fin.CP].

– (2018). "Can agent-based models probe market microstructure?" In: *Physica A: Statistical Mechanics and its Applications* 503, pp. 1092–1106. ISSN: 0378-4371. DOI: 10.1016/j.physa.2018.08.055. URL: http://www.sciencedirect.com/science/article/pii/S0378437118309956.

Plerou, Vasiliki et al. (2000). "Economic fluctuations and anomalous diffusion". In: *Physical Review E* 62.3, R3023. DOI: 10.1103/PhysRevE.62.R3023.

Ponta, L, M Raberto, and S Cincotti (2011). "A multi-assets artificial stock market with zero-intelligence traders". In: *EPL (Europhysics Letters)* 93.2, p. 28002. DOI: 10.1209/0295-5075/93/28002.

Preis, Tobias et al. (2006). "Multi-agent-based order book model of financial markets". In: *EPL (Europhysics Letters)* 75.3, p. 510. DOI: 10.1209/epl/i2006-10139-0.

Raman, Natraj and Jochen L Leidner (2019). "Financial market data simulation using deep intelligence agents". In: *Advances in Practical Applications of Agents and Multi-Agent Systems*. Ed. by Yves Demazeau et al. Springer, pp. 200–211.

Raman, Natraj, Jochen L. Leidner, et al. (2019). *Synthetic Reality: Synthetic market data generation at scale using agent based modeling*. URL: https://www.simudyne.com/wp-content/uploads/2019/06/Simudyne-Refinitiv-Paper.pdf (visited on 05/04/2021).

Real-Logic (2021). *Aeron media driver*. URL: https://github.com/real-logic/Aeron/wiki.

Revels, Jarrett, Miles Lubin, and Theodore Papamarkou (2016). *Forward-Mode Automatic Differentiation in Julia*. arXiv: 1607.07892 [cs.MS]. URL: https://arxiv.org/abs/1607.07892.

Schmitt, Noemi and Frank Westerhoff (2017). "On the bimodality of the distribution of the S&P 500's distortion: Empirical evidence and theoretical explanations". In: *Journal of Economic Dynamics*

and Control 80, pp. 34–53. ISSN: 0165-1889. DOI: 10.1016/j.jedc.2017.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0165188917300994.

Silvey, Samuel David (1975). *Statistical inference*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis. ISBN: 9780412138201.

Simon, Herbert A (1955). "A behavioral model of rational choice". In: *The quarterly journal of economics* 69.1, pp. 99–118. ISSN: 00335533, 15314650. DOI: 10.2307/1884852. URL: http://www.jstor.org/stable/1884852.

– (1957). "Models of man; social and rational." In:

Sing, Dharmesh (2017). *CoinTossX Software*. DOI: 10.25375/uct.14069552. URL: https://github.com/dharmeshsing/CoinTossX.

Sing, Dharmesh and Tim Gebbie (2017). "JSE Matching Engine Simulator". MA thesis. University of the Witwatersrand. URL: http://wiredspace.wits.ac.za/handle/10539/25136.

Stigler, George J (1964). "Public regulation of the securities markets". In: *The Journal of Business* 37.2, pp. 117–142. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2351027.

Thompson, Martin et al. (2011). *Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads*. Tech. rep. LMAX. URL: https://lmax-exchange.github.io/disruptor/.

Verbeke, Geert and Geert Molenberghs (2007). "What can go wrong with the score test?" In: *The American Statistician* 61.4, pp. 289–290. DOI: 10.1198/000313007X243089.

Wang, L et al. (2018). "Agent-based models in financial market studies". In: *Journal of Physics: Conference Series* 1039.1, p. 012022. DOI: 10.1088/1742-6596/1039/1/012022.

Wilcox, Diane and Tim Gebbie (Aug. 2014). "Hierarchical causality in financial economics". In: *SSRN*. DOI: 10.2139/ssrn.2544327. URL: https://ssrn.com/abstract=2544327.

Wilks, Samuel S (1938). "The large-sample distribution of the likelihood ratio for testing composite hypotheses". In: *The annals of mathematical statistics* 9.1, pp. 60–62. DOI: 10.1214/aoms/1177732360. URL: https://doi.org/10.1214/aoms/1177732360.

Winker, Peter, Manfred Gilli, and Vahidin Jeleskovic (2007). "An Objective Function for Simulation Based Inference on Exchange Rate Data". In: *Journal of Economic Interaction and Coordination* 2.2, pp. 125–145. DOI: 10.1007/s11403-007-0020-4.

Zheng, Ban, François Roueff, and Frédéric Abergel (2014). "Modelling bid and ask prices using constrained Hawkes processes: Ergodicity and scaling limit". In: *SIAM Journal on Financial Mathematics* 5.1, pp. 99–136. DOI: 10.1137/130912980.

Zhou, Wei-Xing (2012). "Universal price impact functions of individual trades in an order-driven market". In: *Quantitative Finance* 12.8, pp. 1253–1263. DOI: 10.1080/14697688.2010.504733.