University of the Witwatersrand

MSc Dissertation

The Calibration of Financial Agent-Based Models

Author:
Donovan Platt

Supervisor:
Prof. Tim Gebbie

A dissertation submitted in fulfillment of the requirements of the degree of Master of Science

in the

School of Computer Science and Applied Mathematics

March 22, 2017

Declaration of Authorship

I, Donovan Frederick Platt, declare that this dissertation entitled 'The Calibration of Financial Agent-Based Models' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this university.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation 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: Donovan Platt

Date: 2017/03/22

Abstract

Agent-based models, particularly those applied to financial markets, demonstrate the ability to produce realistic, simulated system dynamics, comparable to those observed in empirical investigations. Despite this, they remain fairly difficult to calibrate due to their tendency to be computationally expensive, even with recent advances in technology. For this reason, financial agent-based models are frequently validated by demonstrating an ability to reproduce well-known log return time series and central limit order book stylized facts, as opposed to being rigorously calibrated to transaction data. We thus apply an established financial agent-based model calibration framework to a number of intraday agent-based models employing realistic order matching procedures and demonstrate that while the parameters of these models rooted in market microstructure can indeed be meaningfully calibrated, those exclusively related to agent behaviors and incentives remain problematic, due to the presence of parameter degeneracies not identified by stylized fact-centric validation. We further argue that the observed parameter degeneracies are likely a consequence of the realistic matching processes employed in these models, which suggests that alternative approaches to linking data, phenomenology and market structure may be necessary and that the stylized fact-centric validation of intraday agent-based models is insufficient.

Acknowledgements

This dissertation is made possible through the support, financial or otherwise, of a number of individuals, groups and organizations. I would therefore like to thank:

- My supervisor, Prof. Tim Gebbie, for the support and guidance given throughout the various stages of this investigation and for instilling in me a renewed passion for research and scientific discovery.
- The remainder of the QuERILab research group for the invaluable feedback and support provided over the past year and for the productive research environment facilitated by the group, with particular thanks to Prof. Diane Wilcox, Dieter Hendricks, Mike Harvey and Fayyaaz Loonat.
- All friends and family members who have supported me throughout the year and have ultimately made the completion of this dissertation possible.
- The various sources of funding that have made the undertaking of research of this nature possible. These include the National Research Foundation (NRF) of South Africa, for the provision of a generous MSc scholarship, and the University of the Witwatersrand, for the provision of a Postgraduate Masters Merit Award.

The conclusions herein are due to the author and the NRF accepts no liability in this regard.

Publications and Preprints

This dissertation makes extensive use of results, figures and discussions originally presented in the following publications and preprints:

1. Platt DF, Gebbie TJ (2016) The Problem of Calibrating a Simple Agent-Based Model of High-Frequency Trading. arXiv:1606.01495

2. Platt DF, Gebbie TJ (2016) Can Agent-Based Models Probe Market Microstructure? arXiv:1611.08510

Contents

List of Figures

List of Tables

List of Symbols

We restrict the list below to standard symbols that are used consistently throughout the literature:

f	Objective function
θ	Arbitrary ABM parameter set
Θ	Feasible parameter space
k	Number of objective function moments
$\mathbf{R}$	Measured (log price) time series
$\mathbf{m}$	Exact moments of the measured process
$\mathbf{m^e}$	Estimate of $\mathbf{m}$ based on $\mathbf{R}$
$\mathbf{m^s}(\theta)$	Simulated moment estimates for parameter set θ
$\mathbf{W}$	Objective function weight matrix
θ^R	Reflection of worst simplex vertex
θ^E	Expansion of worst simplex vertex
θ^O	Out-contraction of worst simplex vertex
θ^I	In-contraction of worst simplex vertex
ρ	Reflection parameter for the Nelder-Mead simplex algorithm
ξ	Expansion parameter for the Nelder-Mead simplex algorithm
ψ	Contraction parameter for the Nelder-Mead simplex algorithm
σ	Shrinking parameter for the Nelder-Mead simplex algorithm
n_R	Number of thresholds for the threshold accepting algorithm
τ_r	r-th threshold for the threshold accepting algorithm
$\mathcal{N}_\theta$	Neighborhood of θ
p_c	Crossover probability for the genetic algorithm
p_m	Mutation probability for the genetic algorithm
θ_{NM+TA}	Best parameter set obtained using the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic
θ_{GA}	Best parameter set obtained using the genetic algorithm

Chapter 1

Introduction

1.1 Investigation Background and Context

The application of agent-based modeling to financial markets has become an area of increasing interest in recent years within the financial modeling community and significant strides have been made in the discipline. Despite this, a number of key issues still remain.

While agent-based models (ABMs) show much promise by being able to replicate some of the empirically-observed stylized facts of return time series, such qualitative comparisons are not a robust form of validation (Barde (2016)). Among the possible solutions to address this concern is the calibration of candidate ABMs to transaction data, though this brings with it a number of significant challenges.

While the literature regarding the calibration of ABMs employing closed-form approximations to daily prices is relatively well-developed, the literature relating to the calibration of more sophisticated intraday ABMs employing realistic matching processes is incredibly sparse. This is likely due to the computational challenges associated with ABM calibration (Fabretti (2013)).

Therefore, qualitative comparisons between the stylized facts of return time series produced by a candidate ABM and those observed empirically remain the standard for the validation of intraday ABMs (Panayi et al. (2013)). For this and a number of other reasons, there still exist many critics of the agent-based modeling approach (Hamill and Gilbert (2016)).

This dissertation therefore presents a comprehensive investigation into the calibration of intraday ABMs by extending the scope of application of the calibration framework of Fabretti (2013), constructed specifically for daily ABMs, to a number of more sophisticated intraday ABMs.

Through this process, we also shed light on the inadequacies of qualitative stylized fact comparisons as a form of model validation and demonstrate their

inability to detect the emergence of parameter degeneracies, leading us to present a comprehensive argument for the centrality and necessity of calibration to ABM validation.

1.2 Literature Review

1.2.1 ABMs and Complex Systems

ABMs have become increasingly commonplace in recent years, a direct result of the increased emphasis on complexity science within the larger scientific community. In short, complexity science is a new paradigm for scientific enquiry that argues that some systems possess emergent properties that are not explainable by an understanding of the properties of their constituent parts. These emergent properties are usually attributed to the interactions and relationships between the components or agents in the system (Heylighen (2008)). These factors can also lead to self-organizing behavior, in which overall order emerges within a system in which there is no initial order, despite the fact that there is no central control optimizing the system as a whole (Heylighen (2008)). Systems demonstrating this behavior are described as complex systems within the discipline and many phenomena in the natural world, society and financial markets are studied within the field.

A key question to consider is how one could actually model a complex system. As demonstrated by Macal and North (2010), ABMs are well-suited to modeling complex systems and historically emerged as an attempt to capture complex behavior in a mathematical framework. These models are essentially constructed in a bottom-up fashion, where the micro-constituents of a system are represented as interacting agents within a simulation framework. Each agent is usually represented as a set of simple rules and properties that define its behavior and its interactions with other agents. Macal and North (2010) indicated that overall patterns and behaviors emerge within such models, despite not being explicitly programmed. This encapsulates the emergent behaviors described in complexity science literature fairly well.

One should, however, be skeptical of the idea that all complex behavior in a system can be replicated using a bottom-up modeling approach. Wilcox and Gebbie (2014) proposed the idea of hierarchical causality in the financial system, stating that there is in fact a hierarchy representing different levels of abstraction, each leading to different nuances in overall behavior. This would imply that there is both bottom-up and top-down causation within the financial system and that replicating complex behavior within it is more nuanced than simply replicating the bottom-up processes that may be present. Nevertheless, ABMs represent a

significant attempt at the replication of complex behaviors emerging in a variety of systems.

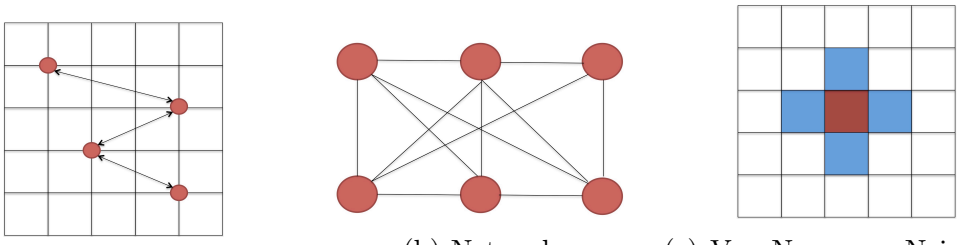
1.2.2 Components of ABMs

Macal and North (2010) stated that ABMs, regardless of application, require three components to be defined: agents, agent interactions and the agent environment.

Agents are described as having attributes, which are application specific properties, such as available wealth or risk preference in financial models, and behaviors, actions associated with a particular agent.

Macal and North (2010) defined agents as self-contained, modular, autonomous, state-dependent and social entities. Agents are self-contained in that one can easily tell which model components belong to a particular agent and which do not. Modularity implies that agents of a particular type, such as low-frequency traders in a financial model, have common features. Agents are required to be autonomous, such that they perform independent actions, not being directed by a central entity that aims to optimize the system, and agents should also be social, able to interact with one another in order to produce the overall system dynamics. State-dependency implies that agents have time-varying states, which result in different attribute values and behaviors, representing the change in agent activity as the overall system dynamics change.

Macal and North (2010) further stated that agents may also be adaptive, modifying their rule sets through processes such as machine learning, goal-directed, comparing the outcomes of interactions to a desired outcome and modifying their rule sets accordingly, and heterogeneous, with different attributes and behaviors among agents within a model.



(a) Euclidean Space

(b) Network

(c) Von Neumann Neighborhood

Figure 1.1: Three common ABM topologies, based on explanations by Macal and North (2010)

Agent interactions are usually described in terms of the neighbors of each agent and the overall topology of the model (Macal and North (2010)). An agent's set

of neighbors is simply the subset of the total agent population in the model with which the agent communicates and this subset changes as time progresses during simulation. In most ABMs, it is unusual for all agents to communicate with one another at all times and most agents possess and share local information through interactions. The way in which agents are connected in a model can be fully described through an understanding of the neighbors of each agent and the nature and flow of information. This is known as the model's topology. There exist many different topology standards and models may include multiple topologies. As an example, a Euclidean space topology could be used in a simple epidemic ABM. We could simulate susceptible and infected agents moving along a Euclidean space, with infections occurring should the Euclidean distance between a susceptible and infected agent be below a certain threshold. In contrast, network topologies may be used where we simply consider direct connections between an agent and its neighbors, rather than movement through some space. Example topologies are presented in Figure 1.1.

The agent environment is a fairly nuanced concept. Macal and North (2010) indicated that the environment may do little more than provide information to agents with regard to their relative positioning in some models, or provide a sophisticated stream of information and restrict agent behavior in other models. In some financial ABMs approximating a continuous double auction market, the limit order book may be seen as a component of the environment, providing a stream of information in the form of quotes and also constraining agent behavior by allowing interactions only through the posting of limit and market orders.

1.2.3 Pros and Cons of Agent-Based Modeling

Agent-based modeling offers a number of compelling advantages over more traditional approaches. ABMs, in most cases, do not require many assumptions regarding the overall dynamics of a system to be made, since these dynamics emerge as a result of agent interactions as opposed to being explicitly programmed (Macal and North (2010)). In an economic context, assumptions such as equilibrium states and market efficiency, which some consider dubious, are therefore not required (Hamill and Gilbert (2016)). Furthermore, ABMs are well-known for their ability to produce realistic behaviors, comparable to empirical observations, largely due to the decreased need for assumptions regarding overall system behavior (Topping et al. (2009)).

Mandes (2015) demonstrated how the simulation-based nature of financial ABMs can also be used to test new regulatory policy, investigating if the implemented policy has the desired effect or instead creates more problems. This is evidence for the powerful explanatory possibilities of ABMs, far exceeding that of many traditional approaches.

Despite the advantages offered by the agent-based modeling approach, it presents a number of disadvantages. Due to the fact that these models are typically simulation-based, there is a tendency for them to be computationally expensive, which can make them difficult to calibrate (Fabretti (2013)). A second issue relates to the existing simulation literature. Hamill and Gilbert (2016) stated that much of the literature relating to ABMs contains insufficiently documented simulation details, making the replication of results difficult, or in some cases, impossible. LeBaron (2005) also noted that there is a lack of standardization among models, making much of the literature very broad and hard to reconcile with other approaches when attempting to create new models.

1.2.4 Market Microstructure and the Limit Order Book

The ABMs that we consider in our investigation make use of a number of concepts from the domain of market microstructure. According to O'Hara (1995), "Market microstructure refers to the study of the process and outcomes of exchanging assets under a specific set of rules", which would also imply consideration of the design and procedures associated with the market.

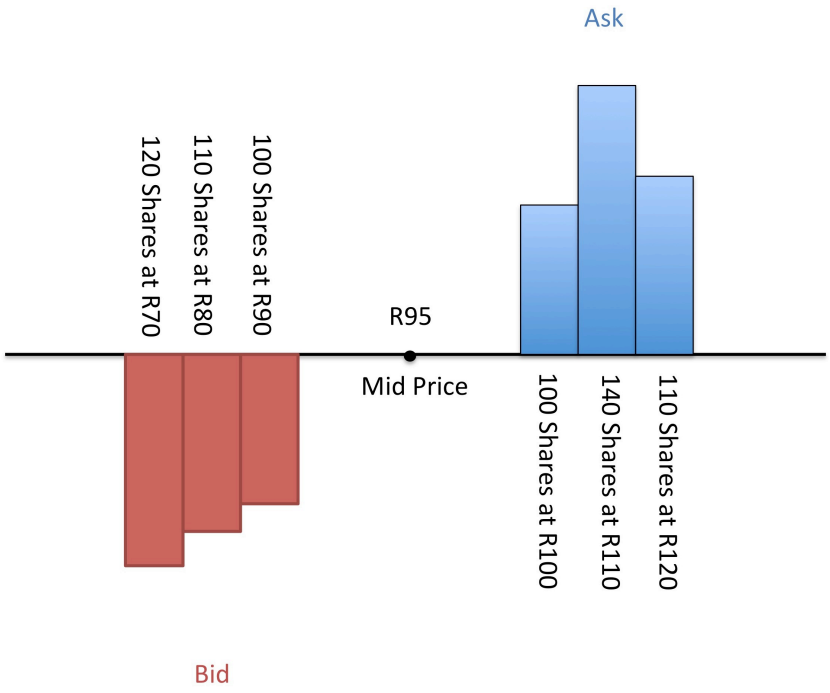


Figure 1.2: An illustration of a central limit order book

The market considered in the ABMs of interest is a continuous double auction market involving a central limit order book (LOB), in which any trader is capable

of posting limit or market orders to buy or sell an asset. This is in contrast to a dealer market where only market makers indicate the price at which they are willing to buy or sell an asset. Limit orders and the LOB were discussed extensively by Gould et al. (2013), with relevant concepts summarized as follows:

A limit order has an associated placement time, order price and order size and only executes if it is possible to achieve the associated order price when buying or selling. Upon arrival in the LOB, a matching algorithm is used to determine if the arriving buy (sell) order can be matched to a sell (buy) order, which is only possible if there is a sell (buy) order in the LOB with an equal or lower (higher) price. If an order can be matched, the order will execute and a trade will occur, otherwise the order will remain in the LOB until an order to which it can be matched arrives or the order is cancelled. It should be noted that orders need not be the same size to be matched. In the case of different order sizes, only a fraction of the volume of the larger matched order would be depleted by the corresponding trade and this order would remain in the LOB with a reduced order size. An illustration of a typical LOB configuration is presented in Figure 1.2.

We typically refer to a sell order in the LOB as an ask and we similarly refer to a buy order as a bid, with the best bid being the bid with the highest order price and the best ask being the ask with the lowest order price. Two key characteristics of the LOB at any given time include the bid-ask spread, which refers to the current difference between the best ask and best bid, and the mid price, which is the current average of the best bid and best ask.

A market order, in contrast to a limit order, simply executes at the best possible price currently available.

Clearly, the LOB and its associated dynamics have a profound effect on the evolution of prices in the market, hence it is a central theme throughout this investigation, with the vast majority of the work conducted being framed within the context of the placement and matching of limit and market orders.

1.2.5 High-Frequency Trading

In a number of the ABMs we consider, the interactions between high-frequency (HF) and low-frequency (LF) traders is a central concept. The distinction between these different groups, however, is often misunderstood.

It is commonly assumed that speed is the defining factor differentiating LF and HF traders, but in reality, HF trading merely benefits from speed and is fundamentally contrasted to LF trading by a change in paradigm. Extensive discussion into this paradigm shift was presented by Easley et al. (2012).

It is suggested that LF traders operate on calendar time, specifically traditional measures of time such as hours and minutes, with consideration given to traditional economic and financial concerns such as stock valuation and financial statement

analysis. HF traders, however, operate on what is known as event time. In this paradigm, orders are placed in response to certain events, such as a particular volume being traded, rather than at certain times of the day. This paradigm is well-suited to computational and algorithmic trading, as computers tend to follow a similar paradigm.

HF trading also differs from LF trading since consideration is given to factors such as the dynamics of the LOB, matching engine and the underlying microstructure of the market in general, particularly in the case of a continuous double auction market. These factors are typically taken as a given by LF traders.

Because of these fundamental differences, HF traders typically exploit the positions and possible mistakes of LF traders by responding to their particular actions, which may be considered events in the event time paradigm. For this reason, HF trading strategies can typically be seen as predatory in nature and tend to heavily exploit the weaknesses of LF trading.

Therefore, HF trading relies less on speed, but rather on making the best possible move based on revealed information before a competitor can make the same move (Easley et al. (2012)). It is clear, however, that speed is of benefit to a trader that employs this paradigm.

A particular type of HF trader that is of interest is called an electronic liquidity provision (ELP) trader. According to Mandes (2015), these traders seek to capture the bid-ask spread and rebates paid as incentives for posting liquidity. ELP traders have smaller order sizes and round-trip gains compared to human market makers, but a larger number of round-trips due to quoting at the best bid and ask more frequently, from which the profitability of the strategy is derived.

These traders feature prominently in the Mandes (2015) model and liquidity providers in the Preis et al. (2006) model engage in similar behaviors. Both models are discussed in Chapter 2.

1.2.6 Motivations for the Consideration of Agent-Based Modeling in Finance

When considering the financial applications of agent-based modeling, it is worth noting reasons why this relatively new approach would be worth investigating as opposed to more traditional approaches, such as stochastic calculus.

LeBaron (2005) suggested that agent-based modeling is appealing in that it does not require assumptions such as the Gaussianity of the distribution of log returns, market efficiency or rational expectations, assumptions which have become increasingly criticized due to their inability to replicate a number of known empirical features. A key example of this particular concern is the fact that log return time series typically have leptokurtic distributions (Cont (2001)), making an as-

sumption of Gaussianity particularly problematic. Therefore, the fact that ABMs typically do not require as many assumptions regarding overall system behavior as more traditional approaches provides some motivation for their use.

LeBaron (2005) further noted that financial markets provide rich datasets that could ultimately be used for robust calibration and validation procedures, a key issue relating to ABMs. Therefore, a financial framework also has the potential to overcome a number of the flaws relating to the discipline of agent-based modeling, particularly the lack of sufficient validation in many applications, a problem noted by Hamill and Gilbert (2016).

ABMs have also been used to provide explanations as to why certain phenomena emerge in empirical studies. Chiarella and Iori (2002) have indicated that fundamentalist trader agents seem to stabilize prices whereas chartist agents tend to cause jumps in prices, providing some form of explanation as to why we may observe certain price movements when considering financial data. Alternatives, such as traditional econometric time series models, can be extended to replicate features observed in empirical return distributions, such as volatility clustering (Cont et al. (1997)), but this would not necessarily explain why such a phenomenon emerges.

1.2.7 Existing Approaches to Financial Agent-Based Modeling

Agent-based modeling in a financial context usually involves the representation of traders as agents who interact through the placement of orders to buy or sell an asset, resulting in a simulated market and price time series. In most cases, multiple agent types exist within a particular model, each representing different trading strategies with different rule sets.

Earlier models, such as the Farmer and Joshi (2002) model, include two trader types, chartists, who observe a price history as a basis for decision making, and fundamentalists, who base their decisions on deviations from a perceived fundamental value of the asset. Each strategy may be implemented using separate agents, as in the Farmer and Joshi (2002) model, or agents may switch between certain strategies with a certain probability, as in the Jacob Leal et al. (2015) model, or agents may follow a weighted combination of strategies, as in the Chiarella and Iori (2002) model. More contemporary approaches, such as the Jacob Leal et al. (2015) and Mandes (2015) models, also include attempts to model more intricate strategies, such as HF trading.

Also worth noting is the work of Nair (2014), who developed an ABM that conforms to the market protocols of the Johannesburg Stock Exchange (JSE), which is the market from which we sample our datasets in this investigation.

Interactions in financial ABMs typically take place in one of two possible ways:

through an LOB or market maker. In the case of an LOB, trader agents submit limit or market orders to buy or sell an asset to an LOB, based on their programmed rule sets, after which orders are matched through a well-defined procedure. Any limit orders not matched remain in the LOB until they expire or are matched at a later stage.

In the case of a market maker, we usually consider a central entity who posts prices at which it will buy or sell an asset. Agents then submit orders to buy or sell at these prices, again according to their programmed rule sets, and the market maker adjusts its specified prices accordingly. Each of these interaction approaches has associated with it a number of possible price determination mechanisms. LeBaron (2005) identified four different price determination mechanisms currently in common use in financial ABMs.

We first consider price adjustment. When using this method, a central market maker announces a price and agents submit orders to buy or sell at this price, based upon their respective rule sets. The price then increases if there is an excess demand and decreases if there is an excess supply, with the exact price changes dependent on the model in question. A classical example of this particular methodology is present in the Farmer and Joshi (2002) model.

When we refer to ABMs with closed-form solutions for daily prices, we typically refer to models incorporating the above mechanism. This mechanism is generally appropriate for the daily time scale considered by these models, since it could be seen as analogous to the closing auctions found at the end of each trading day in real financial markets. Such a simplification would not, however, be appropriate for intraday time scales and hence HF trader activity (Platt and Gebbie (2016a)).

A second price determination mechanism involves numerically calculating a clearing price at the end of each trading session in a series (or using an analytical approximation). The key disadvantage of this particular framework is that the associated session structure fails to simulate continuous trading and periodic market clearing can be computationally expensive. An example of this particular methodology was presented by Brock and Hommes (1998).

The third mechanism discussed by LeBaron (2005) is that of an order book method, where agents post limit or market orders to an LOB, where orders are continuously matched as they arrive through the use of some well-defined procedure. The Chiarella and Iori (2002) and Chiarella et al. (2009) models are examples of this price determination mechanism.

When we refer to intraday ABMs involving realistic matching procedures, we typically refer to the second and third mechanisms. We focus on models incorporating these mechanisms, since they are capable of representing intraday activity, unlike price adjustment. It is worth noting that both of these mechanisms may be used in the context of a continuous double auction market involving an LOB, with

the key distinction being continuous matching as opposed to a session structure.

The final price determination mechanism discussed by LeBaron (2005) is that of random matching. Models involving this process, such as the Beltratti and Margarita (1992) model, usually simulate the random meeting of agents, where agents trade with one another if they perceive some benefit, as per their rule sets. These models typically simulate informal markets and floor trading and are hence not especially relevant in our context.

1.2.8 Replication of Empirically Observed Stylized Facts and Validation of ABMs

One of the more compelling features of financial ABMs, particularly apparent in earlier work in the field, is their ability to effectively replicate empirically-observed stylized facts of return time series, noted by both Fabretti (2013) and Barde (2016). At face value, this ability serves as an informal validation that a particular ABM is, to some extent, correct in principle. While stylized facts are indeed a good starting point in attempting to ascertain the plausibility of a simulated return time series, we would require a far more rigorous framework for true validation.

Hamill and Gilbert (2016) indicated that there is still much skepticism surrounding the inherent correctness and value of agent-based modeling, due to a variety of factors, including existing validation practices. Barde (2016) further noted that the sheer number of ABMs able to replicate empirically-observed stylized facts has become increasingly large and for that reason, more sophisticated methods of comparison and validation are needed.

As suggested by Fabretti (2013), validation can be performed by showing that the statistical properties of a price (or return) time series simulated by an ABM and those of a price (or return) time series derived from real data are sufficiently similar and that the two series can be said to share the same distribution.

This leads to the problem of obtaining a parameter set that allows the chosen ABM to produce a simulated price (or return) time series with the desired statistical properties. The issues of calibration and validation are therefore intrinsically linked.

Despite the fact that concerns have been raised in the financial agent-based modeling community, Fabretti (2013) indicated that calibration and validation literature is still relatively sparse, particularly for sophisticated intraday models. Therefore, it is not surprising that a stylized-fact centric view of model validation is still fairly prevalent (Panayi et al. (2013)), with recent models, such as the Jacob Leal et al. (2015) model, still relying on discussions of stylized facts without introducing rigorous calibration and validation methods.

1.2.9 The Problem of Calibration and the Effectiveness of Calibration Techniques

As indicated by Platt and Gebbie (2016a), a number of successful strategies have emerged for the calibration of models with closed-form solutions for prices, with the two most prominent being the method of simulated moments and maximum likelihood estimation. The work of Gilli and Winker (2003) and Fabretti (2013), on which we base our investigation, detailed the application of the method of simulated moments. Similar experiments involving maximum likelihood estimation were presented in Alfarano et al. (2005), Alfarano et al. (2006) and Alfarano et al. (2007). Also worth noting is the work of Amilon (2008), which presented a comparison of maximum likelihood and the efficient method of moments.

Despite the pervasiveness of the above two methods in ABM calibration literature, new methods and augmentations to existing approaches have also been proposed in recent years, with computational constraints remaining a key obstacle. Notable examples include the work of Recchioni et al. (2015), who employed gradient methods, the work of Guerini and Moneta (2016), who estimated vector autoregression models from simulated and real world data and compared their structure, and the work of Grazzini and Richiardi (2015), who presented a discussion on approaches able to estimate ergodic models (Grazzini (2012)) and in what circumstances such approaches are successful. A fairly exhaustive survey of existing ABM calibration methods was presented by Kukacka and Barunik (2016).

Despite the above attempts at robust and widely-applicable calibration methodologies, the calibration of ABMs is a non-trivial problem. In general, objective functions in an agent-based modeling context tend to lack smoothness and possess many local minima, meaning that many traditional optimization techniques, such as simplex methods, tend to find local minima rather than the true global minimum (Gilli and Winker (2003)). This leads to a need for heuristic methods.

Heuristic methods overcome a number of the challenges associated with traditional approaches, but tend to be stochastic in nature, producing stochastic approximations to the global minimum, and require many iterations for effective calibration. Despite this, Fabretti (2013) stated that a stochastic global minimum is still more desirable than a precise local minimum or no solution at all, with the number of iterations required being the greatest barrier to success. Because ABMs are, by their very nature, computationally intensive, the number of calibration iterations required to implement a number of heuristic methods could potentially lead to significant challenges.

The above problems are largely exacerbated by the use of more realistic matching processes at an intraday time scale, meaning models of this class remain largely uncalibrated, a key motivation for our investigation.

Many of the aforementioned calibration methodologies also suffer from a com-

mon drawback that makes them unsuitable for our purposes. More specifically, a large number of these methodologies assume a closed-form solution for prices, making them unsuitable when these closed-form assumptions are abandoned. Since the framework of Fabretti (2013) does not suffer from this drawback, it was our framework of choice.

It is important to be aware of what the idealistic sufficient and necessary conditions are for a model to be considered identifiable, namely the objective function having a reasonable and unique extremum at the true parameter values (Canova and Sala (2009)). Some caution is required in the context of the types of ABMs we consider as they can at best be approached via indirect estimation to provide consistency between moments and data as opposed to being rigorously estimated; hence we retain the terminology of calibration (Cooley (1997)) as opposed to estimation, in the sense that we attempt to provide consistency between the data and model parameters because we cannot directly test the structure of the models themselves.

1.3 Structure of this Dissertation

In Chapter 2, we provide a detailed discussion of the basic structure, advantages, and disadvantages of four financial ABMs, namely the Farmer and Joshi (2002), Jacob Leal et al. (2015), Preis et al. (2006) and Mandes (2015) models, and provide reasons for our eventual selection of the Jacob Leal et al. (2015) and Preis et al. (2006) models for use in our investigation.

In Chapter 3, we discuss the ABM calibration framework proposed by Fabretti (2013), including the associated objective function construction process and optimization algorithms. We also detail the data sampling strategy employed in this investigation.

In Chapter 4, we provide a brief overview of the software implementation strategy employed in this investigation, with a focus on the selected programming language, software paradigms, and computing hardware.

In Chapter 5, we demonstrate the ability of our implementations of the Jacob Leal et al. (2015) and Preis et al. (2006) models to replicate a number of empirically-observed return time series stylized facts.

In Chapter 6, we present the results of the conducted calibration experiments and provide a detailed analysis of the behavior of the objective function over the space of possible model parameter values.

In Chapter 7, we provide detailed analysis of the calibration results, link these findings to established literature in the field of market microstructure, and discuss the possible implications of these findings.

In Chapter 8, we summarize the key findings of this investigation and include concluding remarks.

Chapter 2

Financial ABM Selection

A key concern in the achievement of successful calibration experiments involving intraday ABMs is the selection of appropriate candidate models for calibration.

A balance must be struck between the sophistication of a particular model and the time taken to execute a single simulation. Very detailed models with more sophisticated and realistic mechanisms have the potential to result in more meaningful calibration results, but present the problem of being too difficult to calibrate, due to relatively long execution times for every simulation. Simpler models offer the advantage of being easier to calibrate, but models that are unrealistically simplified may result in calibration results that are not meaningful.

For this reason, we discuss a number of models, namely the Farmer and Joshi (2002), Jacob Leal et al. (2015), Preis et al. (2006) and Mandes (2015) models, and provide reasons for our eventual selection of the Jacob Leal et al. (2015) and Preis et al. (2006) models.

All flowcharts appearing in the following subsections make use of a consistent color key for improved readability. All LF trader events are presented in red, all HF trader events are presented in blue, and all LOB and other processing activities are presented in green.

2.1 The Farmer and Joshi Model

The first ABM we consider is not an intraday model of the class of models we are attempting to calibrate, but rather a closed-form, daily model that has already been successfully calibrated (Fabretti (2013)). This model is primarily presented in order to provide more concrete distinctions between the two classes of models we discuss, namely daily and intraday models.

2.1.1 Model Overview

Referring to Figure 2.1, the model consists of two trader agent types, chartists and fundamentalists, previously discussed in Section 1.2. During each of T discrete simulation steps, these trader agents indicate demands to buy or sell certain quantities of an asset to a risk-neutral market maker, who then aggregates these demands and fills the requested orders at a newly determined market price based on a closed-form equation aggregating trader agent demands. This ultimately results in a time series of market prices.

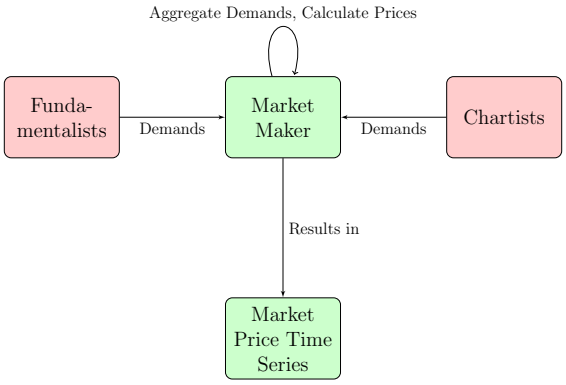


Figure 2.1: An illustration of the agent types and interactions within the Farmer and Joshi (2002) model

2.1.2 Simulation Event Flowchart

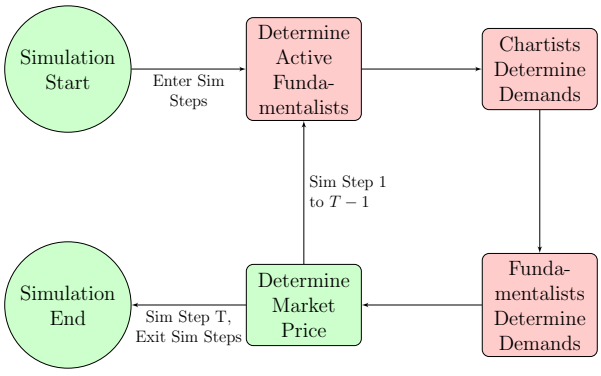


Figure 2.2: Flowchart describing a typical simulation of T simulation steps in the Farmer and Joshi (2002) model

The events occurring during each simulation step, presented in the flowchart in Figure 2.2, are elaborated upon in subsequent subsections.

2.1.3 Trader Agent Specification

Each simulation consists of N trader agents, enumerated by the index i.

During an arbitrary simulation step, $t \leq T$, the i-th trader agent sets their desired demand for the asset, defined as the quantity they intend to buy or sell, according to

$$\omega_t^i = x_t^i - x_{t-1}^i, \tag{2.1}$$

where x_t^i is the position of the i-th trader agent at time t, which is set according to their associated strategy, chartist or fundamentalist.

In the case of a chartist trader agent, this position is based upon past trends, with the trader agent taking a long (positive) position if the asset has recently been increasing in value, or a short (negative) position if the asset has recently been decreasing in value. We set this position according to

$$x_{t+1}^i = c(p_t - p_{t-d}), \tag{2.2}$$

where c is a positive constant, d is the time lag, and $p_t = \log P_t$ is the logarithm of the market price (in units of currency) of the asset at time t.

Fundamentalists on the other hand, believe that the true asset value is not necessarily reflected in the current market price and thus base their decisions on price deviations from a perceived fundamental asset value. When a fundamentalist trader agent believes that the asset is undervalued, it takes a long position in the asset. When a fundamentalist trader agent believes that the asset is overvalued, it takes a short position in the asset. We set this position according to

$$x_{t+1}^i = c(\nu_t - p_t), \tag{2.3}$$

where ν_t is the logarithm of the fundamental value of the asset (in units of currency) at time t. We assume the logarithm of the fundamental value follows a random walk given by

$$\nu_{t+1} = \nu_t + \eta_{t+1}, \tag{2.4}$$

where η_t is an IID noise process with mean μ_η and standard deviation σ_η.

It is important to note that not all fundamentalist trader agents are active in all simulation steps, unlike chartist trader agents, but rather activate (enter or exit positions) only when a certain threshold of mispricing ($m_t = p_t - \nu_t$) is realized. If positions were changed every simulation step, this would equate to excessive transaction costs in a real-world setting, making the model incredibly unrealistic.

We thus assume that each fundamentalist trader agent has an associated position entry threshold, T^i, drawn from $\mathcal{U}(T_{min}, T_{max})$, and an associated position exit threshold, τ^i, drawn from $\mathcal{U}(\tau_{min}, \tau_{max})$.

A short position, $-\omega$, is only entered when $m_t > T^i$ and only exited when $m_t < \tau^i$. A long position, ω, is only entered when $m_t < -T^i$ and only exited when $m_t > -\tau^i$.

2.1.4 Trader Demand Aggregation

Once all active trader agents have indicated their demands in an arbitrary simulation step, $t \leq T$, the market maker determines the price at which to fill the associated orders by aggregating these trader demands.

This is achieved by first determining the net order according to

$$\omega_t = \sum_{i=1}^{N} \omega_t^i \tag{2.5}$$

We now consider the market impact function given by

$$P_{t+1} = P_t e^{\frac{\omega_t}{\lambda}}, \tag{2.6}$$

where we refer to λ as the liquidity parameter.

Finally, we let $p_t = \log P_t$ and add the noise term ξ_{t+1}, yielding

$$p_{t+1} = p_t + \frac{1}{\lambda}\omega_t + \xi_{t+1}, \tag{2.7}$$

which is a closed-form solution for the logarithm of the market price at time $t+1$, a key characteristic of this model. It is worth noting, however, that earlier models also often possess tractable closed-form solutions for returns, such as the Kirman (1991) model, which is not the case for this model.

2.1.5 Model Discussion

The Farmer and Joshi (2002) model is important to consider as it presents a simplified framework through which to develop an understanding of the basic structure of financial ABMs. The underlying chartist and fundamentalist rule constructions used within it are still employed by a number of contemporary models, most notably the Jacob Leal et al. (2015) model. New models, however, have largely replaced the demand aggregation mechanisms and central market maker employed in the Farmer and Joshi (2002) model with a central LOB and realistic order matching processes to generate trades and market prices (Preis et al. (2006); Jacob Leal et al. (2015); Mandes (2015)).

The use of a closed-form demand aggregation equation offers a significant advantage over realistic matching processes in terms of speed, but such assumptions are only realistic at a daily time scale, where prices are determined through closing auctions, similar to the aforementioned demand aggregation equations. Intraday time scales, on the other hand, involve prices which are determined through a continuous double auction mechanism and central LOB, meaning that such mechanisms would need to be more accurately replicated to construct theoretically sound intraday ABMs.

Since the calibration of the Farmer and Joshi (2002) model has been thoroughly documented by Fabretti (2013), we do not present a replication of such results.

2.2 The Jacob Leal et al. Model

The first intraday ABM that we consider is the Jacob Leal et al. (2015) model, which is both implemented and subjected to calibration experiments in our investigation.

The subsequent subsections detailing the model are largely reproduced from Platt and Gebbie (2016a), with minor alterations.

2.2.1 Model Overview

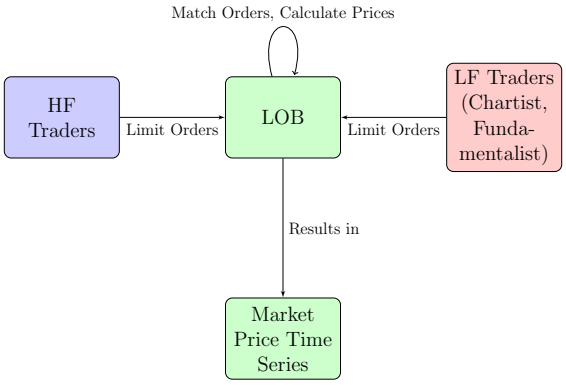


Figure 2.3: An illustration of the agent types and interactions within the Jacob Leal et al. (2015) model

Referring to Figure 2.3, the model consists of two agent types, HF and LF traders, where each LF trader may be following either a chartist or fundamentalist strategy. These agents interact through the posting of limit orders to a central LOB, with time progressing through a series of T one-minute trading sessions. We thus see

that the Jacob Leal et al. (2015) model is largely an extension of the framework of Farmer and Joshi (2002), with realistic matching processes and HF trader activity being added.

During each session, each active trader agent places a single limit order, with price and size determined by the trader's current strategy. Thereafter, limit orders in the order book are matched according to price and then time priority until all possible trades have been executed, with the final trade price defining the market price for the session.

It should be noted that at any point in the simulation, all traders have access to the history of past market prices and fundamental values as well as the current state of the LOB.

2.2.2 Simulation Event Flowchart

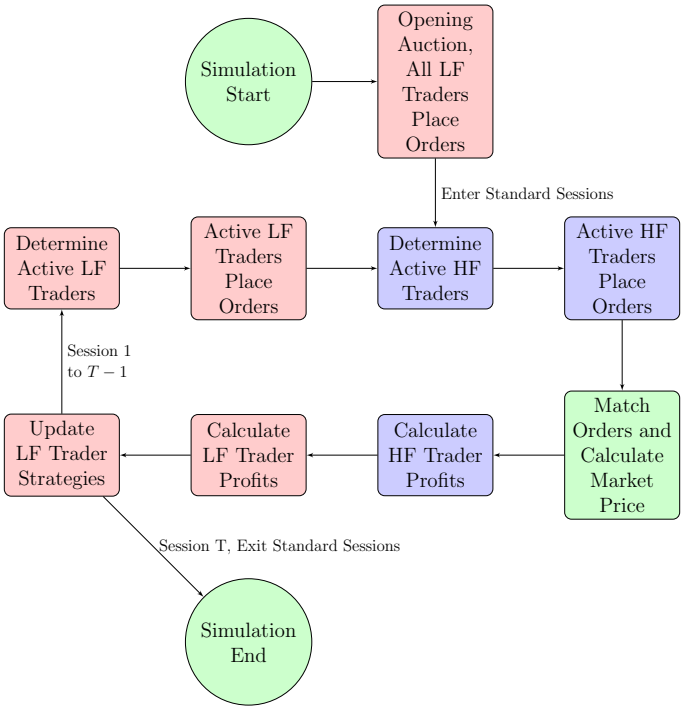


Figure 2.4: Flowchart describing a typical simulation of T one-minute trading sessions within the Jacob Leal et al. (2015) model

In Figure 2.4, we summarize the key events associated with Jacob Leal et al. (2015) model trading sessions. The opening auction is our own added initialization process, discussed in detail in Section 2.2.6.

2.2.3 LF Trader Agent Specification

Each simulation contains N_L LF traders, where different LF traders are represented by the index $i = 1, ..., N_L$. Each LF trader has its own trading frequency, θ_i, drawn from a truncated exponential distribution with mean θ and upper and lower bounds θ_{max} and θ_{min} respectively. This is described by Jacob Leal et al. (2015) as representing the chronological time frame associated with LF traders. This time frame was discussed at length by Easley et al. (2012). At each activation, active LF traders randomly select either the chartist or fundamentalist strategy for the session, with the probability of being chartist given by $\Phi^c_{i,t}$.

We now elaborate on the simulation event overview presented in Figure 2.4 with appropriate equations for an arbitrary session $t \leq T$.

After activation, the i-th LF trader places a limit order, which requires the specification of both an order size and an order price. Should the i-th LF trader agent be following the chartist strategy, we calculate the order size according to

$$D^c_{i,t} = \alpha^c(\bar{P}_{t-1} - \bar{P}_{t-2}) + \epsilon^c_t \tag{2.8}$$

where $\bar{P}_t$ is the market price (in units of currency) for session t, $0 < \alpha^c < 1$ and $\epsilon^c_t \sim \mathcal{N}(0, (\sigma^c)^2)$.

Conversely, in the case of the fundamentalist strategy, we calculate the order size according to

$$D^f_{i,t} = \alpha^f(F_t - \bar{P}_{t-1}) + \epsilon^f_t \tag{2.9}$$

where $F_t = F_{t-1}(1+\delta)(1+y_t)$ is the fundamental price (in units of currency) for session t, $\delta > 0$, $0 < \alpha^f < 1$, $\epsilon^f_t \sim \mathcal{N}\left(0, (\sigma^f)^2\right)$ and $y_t \sim \mathcal{N}(0, (\sigma^y)^2)$.

The order price is identical under both strategies and is determined according to

$$P_{i,t} = \bar{P}_{t-1}(1+\delta)(1+z_{i,t}) \tag{2.10}$$

where $z_{i,t} \sim \mathcal{N}(0, (\sigma^z)^2)$.

Once the order is placed, it remains in the LOB until it is matched or until γ^L sessions have passed, resulting in its removal from the order book. At the end of each session, the i-th LF trader calculates their profits according to

$$\pi^s_{i,t} = (\bar{P}_t - P_{i,t})D^s_{i,t} \tag{2.11}$$

where $s = c$ for the chartist strategy and $s = f$ for the fundamentalist strategy.

Finally, the i-th LF trader determines their probability of being chartist in the next session according to

$$\Phi^c_{i,t} = \frac{e^{\pi^c_{i,t}/\zeta}}{e^{\pi^c_{i,t}/\zeta} + e^{\pi^f_{i,t}/\zeta}} \tag{2.12}$$

where $\zeta > 0$ is the intensity of switching parameter.

2.2.4 HF Trader Agent Specification

Each simulation contains N_H HF traders, where different HF traders are represented by the index $j = 1, ..., N_H$. In contrast to LF traders, HF traders in the model follow an event-based activation procedure as opposed to a chronological trading frequency. This is again consistent with the work of Easley et al. (2012), who suggested that HF traders follow an event time paradigm.

In an arbitrary session, $t \leq T$, the j-th HF trader will activate should the following activation condition be satisfied

$$\left| \frac{\bar{P}_{t-1} - \bar{P}_{t-2}}{\bar{P}_{t-2}} \right| > \Delta x_j \tag{2.13}$$

where Δx_j is drawn from a truncated uniform distribution, with support between η_{min} and η_{max}.

Following activation, active HF traders either buy or sell, where this choice is made randomly with equal probability. In a construction again consistent with the empirical findings of Easley et al. (2012), HF trader agents in the model exploit order book dynamics created by LF trader agents earlier in each trading session by setting their order sizes according to order volumes present in the order book, in an attempt to absorb the orders of LF traders. This is represented by each active HF trader in the simulation setting their order size randomly according to an exponential distribution [1], with the mean being the average order volume in the opposite side of the order book. For an HF trader who is selling units of the asset, the opposite side of the order book would consist of buy orders and vice-versa.

After the setting of the order size as previously discussed, the j-th HF trader sets their order price (provided they are active) according to

$$P_{j,t} = P_t^{bid}(1 - \kappa_J) \tag{2.14}$$

if selling, or according to

$$P_{j,t} = P_t^{ask}(1 + \kappa_j) \tag{2.15}$$

if buying, where P_t^{bid} and P_t^{ask} are the best bid and ask in the order book respectively for session t and κ_j is a uniform random variable with support between κ_{min} and κ_{max}.

Once the order is placed, it remains in the LOB until it is matched or until $\gamma^H < \gamma^L$ sessions have passed, resulting in its removal from the order book. Finally, at the end of each session, the j-th HF trader calculates their profit according to:

$$\pi_{j,t} = (\bar{P}_t - P_{j,t})D_{j,t} \tag{2.16}$$

where $D_{j,t}$ is the j-th HF trader's order size for the session.

[1] In the original paper by Jacob Leal et al. (2015), a Poisson distribution was used, where the mean volume was weighted by $0 < \lambda < 1$. We retain this weighting by λ, but discussions on why the distribution was changed can be found in Section 2.2.6.

2.2.5 Order Matching and Price Determination

Jacob Leal et al. (2015) indicated that orders are to be matched by price and then time priority. In our implementation, it is noted that after all trader agents have placed orders in a particular session, a crossing of the bid-ask spread is typically observed. In other words, $P_t^{ask} - P_t^{bid} < 0$ after all orders have been placed. This implies that the best bid and best ask can immediately be matched, resulting in a trade.

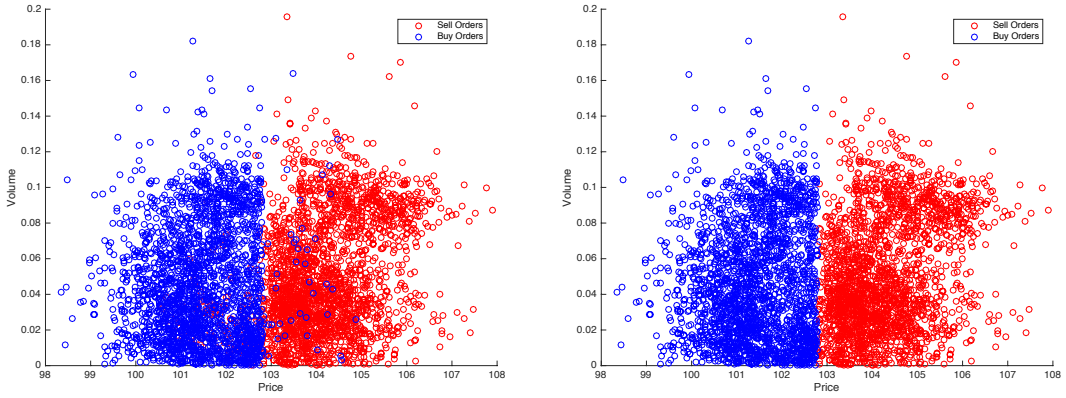


Figure 2.5: Illustration of a typical LOB configuration before (left) and after (right) order matching for an arbitrary session in the Jacob Leal et al. (2015) model, showcasing an initial crossing of the bid-ask spread

We set the trade price equal to the average of the prices of the matched buy and sell orders and the trade size equal to the size of the smallest order in the matched pair. We can thus execute all executable limit orders at the end of each session by continually matching the best bid to the best ask until the bid-ask spread is no longer crossed, with the best bid and best ask changing as orders are depleted. Should there be a tie for the best bid or best ask, the order that was placed earliest is always executed first. The market price for each session is simply the final trade price for that session.

The above mechanism represents a significant departure from the framework of Farmer and Joshi (2002).

2.2.6 Required Implementation Modifications

As indicated by Hamill and Gilbert (2016), replicating ABM results is often challenging due to the opacity of and lack of detail in much of the ABM literature, and not surprisingly, we encountered similar difficulties when implementing the Jacob Leal et al. (2015) model.

While we remained as faithful as possible to the original model, we required two minor changes in order to obtain a functioning implementation.

Firstly, we implement our own method of initialization. This stems from the fact that model initialization procedures were not discussed in detail by Jacob Leal et al. (2015). In order to be consistent with the opening auctions found at the start of each trading day in actual double auction markets, we begin with an analogous event in the first simulated trading session. In this session, all LF traders in the simulation place limit orders to buy or sell according to a random initial strategy, where the probability of each LF trader being either chartist or fundamentalist is initially equal. This mimics the large spike in activity that can be observed at the start of a trading day, an aspect of intraday seasonality (Lehalle (2013); Cartea et al. (2015)).

Secondly, for the parameters originally suggested by Jacob Leal et al. (2015), $\alpha^c = 0.04$, $\sigma^c = 0.05$, $\alpha^f = 0.04$ and $\sigma^f = 0.01$, as well as an initial market price of $\bar{P}_0 = 100$ and an initial fundamental value of $F_0 = 100$, it was found that the model produced LF trader order sizes typically less than one under both the chartist and fundamentalist strategies. Referring to Eqn. (2.8), it is evident that even for relatively large price changes over a one-minute period ($\sim$ 10 units of currency), the model will produce order sizes typically less than one for the given parameters. Even if α^c and σ^c were to be made larger, the fact that α^c is bounded above by one means that typical price changes ($\sim$ 1 unit of currency) will still result in the same problematic order size dynamics. Analogous issues apply to Eqn. (2.9). This in itself is somewhat problematic, as real markets only allow for whole number order sizes.

Furthermore, the fact that HF trader order sizes were originally set according to a Poisson distribution, with mean dependent on the average order size on the opposite side of the order book, resulted in the majority of active HF trader order sizes being zero when combined with these unusual LF trader order sizes. For this reason, an exponential distribution was used instead, allowing for the unusually small mean order sizes found in the model to remain as is without resulting in zero values for HF trader order sizes. Exponential distributions are used to determine order sizes in many models throughout existing literature, a notable example being given by Mandes (2015). This provides suitable justification for this choice of distribution.

It should be noted that when matching orders, the only consideration with regard to order sizes is that they be sensible in relation to one another, with actual magnitudes being of more importance for studies involving trade volume effects. Thus, for our purposes, this simple correction is sufficient and these unusual order sizes will not affect the dynamics of the obtained market price time series. A more comprehensive correction of this order size issue should be considered in studies

in which trade volumes and order sizes are important characteristics.

Despite the required changes, we demonstrate that we are still able to replicate the results of Jacob Leal et al. (2015) in Chapter 5.

2.2.7 Model Discussion

The Jacob Leal et al. (2015) model is an effective starting point for our investigation into the calibration of intraday ABMs, since the model bears some similarity to the Farmer and Joshi (2002) model, which has already been successfully calibrated by our selected calibration framework (Fabretti (2013)). Furthermore, the model is also fairly recent and attempts to model HF trader activity, making the calibration of such a model incredibly topical.

In addition to these key advantages, the model's mechanism of market clearing at the end of each trading session is also relatively fast in comparison to continuous order matching. It is worth noting that this process resembles the second type of price determination mechanism described by LeBaron (2005), which we discussed in Section 1.2.

Despite these very important advantages, the Jacob Leal et al. (2015) model also suffers from a number of flaws, which required us to eventually extend our calibration experiments to more theoretically sound models.

Firstly, the model's greatest flaw is its exclusion of market orders, since it uses limit orders exclusively. This is problematic, since market orders are an essential consideration in modern market microstructure and are as likely to be encountered in continuous double auction markets as limit orders. (Lehalle (2013); Cartea et al. (2015)).

Secondly, the model's mechanism of batch order placement by all active traders before matching orders at the end of each session is not realistic. In a true, continuous double auction market, attempts to match orders would be made continuously, as orders are placed, rather than at the end of a specified period. Therefore, the increased speed of the model is achieved at the expense of its theoretical soundness.

Finally, it was discovered through our calibration experiments that the random walk determining LF trader order prices in Eqn. (2.10) tended to dominate the model dynamics. The use of a random walk is somewhat arbitrary, and for that reason, we originally surmised that more realistic order placement mechanisms based on the current state of the order book would be advantageous.

The above concerns were addressed in both the Mandes (2015) and Preis et al. (2006) models, though in different ways, which we discuss in more detail in subsequent subsections. It was originally intended that the Mandes (2015) model be implemented and calibrated once the Jacob Leal et al. (2015) had been considered, since it presented many promising ideas and a high degree of sophistication. Despite this, the Mandes (2015) model presented its own significant challenges,

which we discuss in Section 2.4.6. For this reason, we eventually considered the Preis et al. (2006) model for our second round of calibration experiments.

2.3 The Preis et al. Model

The second intraday ABM we consider is that presented by Preis et al. (2006) and later elaborated upon by Preis et al. (2007). We both implemented the model and subjected it to calibration experiments in our investigation.

The subsequent subsections detailing the model are largely reproduced from Platt and Gebbie (2016b), with minor alterations.

2.3.1 Model Overview

Referring to Figure 2.6, the model consists of two agent types, liquidity providers, who submit limit orders to buy or sell an asset to an LOB, and liquidity takers, who submit market orders to buy or sell an asset to the same LOB, through a series of T Monte Carlo steps.

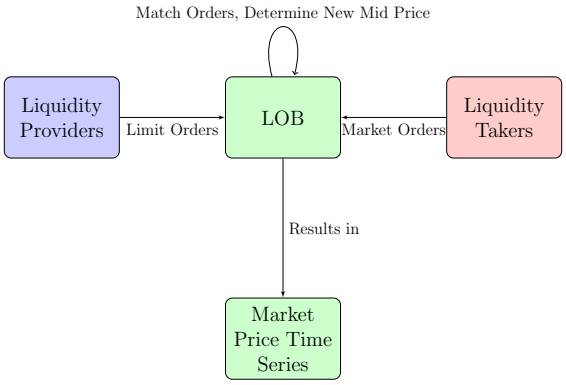


Figure 2.6: An illustration of the agent types and interactions within the Preis et al. (2006) model

During each of these Monte Carlo steps, active liquidity providers first submit limit orders, referencing a decision based on the current state of the LOB, after which liquidity takers submit market orders, resulting in a number of trades. Based on the mid price of limit orders in the LOB following the execution of all trades in a Monte Carlo step, a new market price for the asset can be determined, ultimately resulting a series of market prices, one for each Monte Carlo step.

2.3.2 Simulation Event Flowchart

The flowchart in Figure 2.7 presents the events associated with each Monte Carlo step. The presented steps are elaborated upon in subsequent subsections.

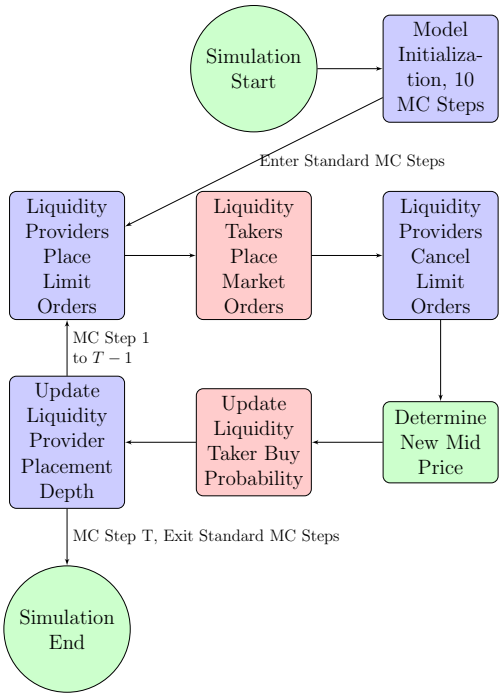


Figure 2.7: Flowchart describing a typical simulation of T Monte Carlo steps within the Preis et al. (2006) model

2.3.3 Liquidity Provider Agent Specification

Each simulation contains N_A liquidity providers, who submit limit orders to the LOB at a given frequency, α. This means that for any given Monte Carlo step, approximately $\lfloor \alpha N_A \rfloor$ limit orders are placed by liquidity providers. Each of the aforementioned orders is of size one and is a buy order with probability $q_{provider} = \frac{1}{2}$.

The price of each of these orders is determined by the current placement depth parameter, $\lambda(t)$, and the current state of the order book, namely the current best bid, p_b, and the current best ask, p_a, where the price of a limit buy order is given by

$$p = p_a - 1 - \eta \tag{2.17}$$

and the price of a limit sell order is given by

$$p = p_b + 1 + \eta, \tag{2.18}$$

where η is an exponentially distributed random number, generated according to

$$\eta = \lfloor -\lambda(t) \ln(u) \rfloor \tag{2.19}$$

and $u \sim U(0, 1)$.

The placement depth parameter is time varying, and is directly related to the buy probability of liquidity takers, $q_{taker}(t)$, discussed in more detail in subsequent subsections, and the initial placement depth parameter, λ_0. It is given by

$$\lambda(t) = \lambda_0 \left(1 + \frac{\left| q_{taker}(t) - \frac{1}{2} \right|}{\sqrt{\left\langle \left[q_{taker}(t) - \frac{1}{2} \right]^2 \right\rangle}} C_\lambda \right), \tag{2.20}$$

where C_λ is an integer parameter and $\langle [q_{taker}(t) - \frac{1}{2}]^2 \rangle$ is determined by iterating $q_{taker}(t)$ for 10^5 Monte Carlo steps separately before the main simulation and obtaining the mean value of $[q_{taker}(t) - \frac{1}{2}]^2$.

It should be noted that the above order placement mechanism is entirely driven by the current characteristics of the LOB. This is in contrast to the placement mechanism used by Jacob Leal et al. (2015), which is based on the previous history of market prices. A more detailed LOB-based order placement mechanism was presented by Mandes (2014).

During each Monte Carlo step, liquidity providers may also cancel previously placed orders. This is implemented by randomly cancelling each limit order in the LOB with probability δ.

Finally, the model requires some form of initialization to produce a viable order book before actual trading can begin. This is done in a series of 10 initial Monte Carlo steps, in which liquidity providers place limit orders around an initial price, $p_a = p_b = p_0$, with no liquidity taker activity present. This corresponds to the placement of approximately $\lfloor 10\alpha N_A \rfloor$ limit orders.

While never explicitly presented as such, the behavior of liquidity providers in the Preis et al. (2006) model is not unlike that of HF traders, in particular ELP traders, as discussed by Mandes (2015). Considering that the model emerged at a time when HF trading was not as highlighted in the literature as it is now, it is not surprising that this explicit link to HF trading was not presented.

2.3.4 Liquidity Taker Agent Specification

Each simulation contains N_A liquidity takers, who submit market orders to the LOB at a given frequency, μ. This means that for any given Monte Carlo step,

approximately $\lfloor \mu N_A \rfloor$ market orders are placed by liquidity takers. Each of the aforementioned orders is of size one and is a buy order with probability $q_{taker}(t)$, with $q_{taker}(0) = \frac{1}{2}$.

A sell market order simply executes and results in the removal of the best bid from the LOB and a buy market order simply executes and results in the removal of the best ask from the LOB, since all orders are of size one.

Unlike $q_{provider}$, $q_{taker}(t)$ is not fixed, but rather evolves over time. It is implemented as a mean-reverting random walk, with mean $q_{taker}(0)$, increment Δ_S, and a mean reversion probability of $\frac{1}{2} + |q_{taker}(t) - \frac{1}{2}|$.

2.3.5 Order Matching and Price Determination

Since the order placement depth process, $\lambda(t)$, is strictly positive, newly placed sell (buy) limit orders always have a price greater (less) than the best bid (ask). This ensures that limit orders never degenerate to market orders and that limit orders themselves are only matched to incoming market orders.

Therefore, since both limit and market orders have size one by default, an incoming buy (sell) market order simply results in the execution of the current best ask (bid), leading to its removal from the LOB. After all market orders have been placed in a particular MC step, the new market price is set to be the new mid price of limit orders in the LOB.

Since market orders are essentially matched as they arrive, this mechanism is far closer to the continuous order matching found in real double auction markets when compared to the unrealistic market clearing of the Jacob Leal et al. (2015) model, even though it still makes use of a session structure. This mechanism is therefore similar to the third price determination mechanism described by LeBaron (2005), discussed in Section 1.2.

2.3.6 Required Implementation Modifications

It is worth noting that the model was implemented without needing to make any modifications or assumptions and that the replication of the results of Preis et al. (2006) was a relatively straightforward process.

2.3.7 Model Discussion

This model was primarily selected as the candidate for our second round of calibration experiments because it addressed our key criticisms of the Jacob Leal et al. (2015) model.

Firstly, as discussed extensively in the preceding subsections, the model employs both market and limit orders, and makes use of an order matching process that more closely resembles continuous trading.

Secondly, the model does not make use of an unrealistic random walk order price determination mechanism, but rather makes use of an order placement depth process designed to replicate empirically-measured order depth distributions (Preis et al. (2007)).

Finally, the model is generally more parsimonious, requiring fewer parameters to replicate the same number of well-known stylized facts of log return time series as in the Jacob Leal et al. (2015) model, while also considering additional features, such as the Hurst exponent.

2.4 The Mandes Model

The final intraday ABM we consider is the Mandes (2015) model. While we had originally intended to implement this model for consideration in the second round of calibration experiments, difficulties in replicating the original paper's results led us to select the Preis et al. (2006) model instead. Detailed discussions on the reason for this decision are presented in Section 2.4.6.

2.4.1 Model Overview

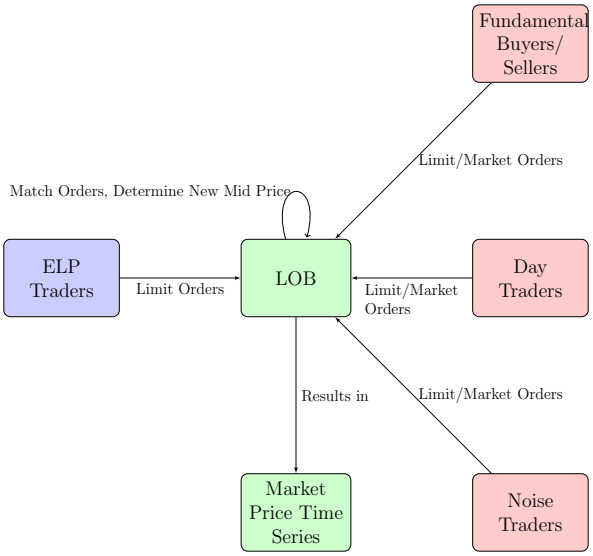


Figure 2.8: An illustration of the agent types and interactions within the Mandes (2015) model

Despite the fact that we ultimately did not implement the model, it is still worth presenting as it exemplifies a number of the pitfalls currently facing financial agent-based modeling, including result replication difficulties and a lack of clarity when describing certain details.

Referring to Figure 2.8, the model consists of three LF trader agent types, including fundamental buyers (or sellers), day traders and noise traders. In addition to the LF trader agents present within the model, a single type of HF trader is also represented, namely an ELP trader. These traders interact through the posting of limit and market orders to a central LOB. Unlike the previously discussed models, the Mandes (2015) model dispenses with session or simulation step structures entirely, instead focusing on a continuous stream of asynchronous agent and LOB events.

2.4.2 Simulation Event Description

One of the model's most promising features is the fact that it dispenses with the session mechanics commonplace in a number of other models and presents a fairly good approximation to continuous trading. Because of this design change, it is no longer possible to provide a flowchart depicting sequential model events.

Rather than activating at a specific stage in a predefined sequence of events, each agent has activation times that are governed by a Poisson process. The Poisson process for each agent draws activation interval lengths from an exponential distribution with rate parameter $\lambda = \frac{1}{\beta_{tf}}$, where β_{tf} is the characteristic frequency of the agent type. These activation times reference a simulation clock or centrally-controlled, continuously incremented timer, measured in microseconds.

When agent activations occur, existing orders are cancelled or new orders placed, with a certain probability associated with each event. If an order is placed and arrives at the LOB, it is either queued, in cases of busy periods with many orders being placed, or added to the order book (and matched immediately if possible).

Because of the continuous nature of the simulation, we are no longer limited to a session-based timeline for market price changes, with the market price of the asset changing as events occur. This allows us to record three time series at tick-by-tick, minute-by-minute and day-by-day frequencies. We set the market price to be the current mid price in the LOB whenever it is to be calculated, as determined by the simulation's central timer.

2.4.3 LF Trader Agent Specification

The Mandes (2015) model makes use of a fairly unorthodox LF trader implementation. LF traders are represented as 'clusters', which are essentially single agents

representing entire populations of particular LF trader types in the simulation.

At each activation, a cluster has probability λ_c of cancelling its oldest standing limit order and probability $1 - \lambda_c$ of placing a new order.

When placing a new order at time t, an LF trader cluster calculates the expected future value of the market price at time $t+1$ using a weighted sum of price expectations associated with different strategies, s, drawn from a pool of available strategies, S. This weighted sum is similar to the approach taken by Chiarella and Iori (2002).

The time index presented in the equations governing these price expectations has meaning relative to whether a strategy is implemented as an intraday (ID) strategy or end of day (EOD) strategy.

If applying a strategy as an EOD strategy, p_{t-1} represents the market price at the end of the previous day (in units of currency), whereas if applying a strategy as an ID strategy, p_{t-1} represents the market price at the end of the previous minute (in units of currency).

We further define f_t to be the fundamental value of the asset at time t, $r_t = (p_t - p_{t-1})/p_t$ to be the one-period linear return at time t, and m_t to be the mid price of limit orders in the LOB at time t, with all of these quantities being measured in units of currency and the distinction between EOD and ID strategies being the same as for p_t. We now consider the associated equations in detail.

Under the fundamentalist strategy, which may be implemented only at an EOD frequency, the price expectation is given by

$$\mathbb{E}_t[p^{fund}] = f_t \epsilon^{fund}, \tag{2.21}$$

where $\epsilon^{fund} \sim U(0.99, 1.01)$.

Under the trend following strategy, which may be implemented at both EOD and ID frequencies, the price expectation is given by

$$\mathbb{E}_t[p^{trend}] = p_t \left(1 + \left(0.95 \frac{1}{\tau} \sum_{i=1}^{\tau} r_{t-i} + 0.05 r_t\right) \tau\right), \tag{2.22}$$

where $\tau \sim U(1, 9)$ for EOD time scales and $\tau \sim U(10, 60)$ for ID time scales.

Under the reversal strategy, which may be implemented at both EOD and ID frequencies, the price expectation is given by

$$\mathbb{E}_t[p^{rev}] = \frac{1}{\tau} \sum_{i=1}^{\tau} p_{t-i}, \tag{2.23}$$

with τ as in the trend following strategy.

Finally, under the noise strategy, which may be implemented at both EOD and ID frequencies, the price expectation is given by

$$\mathbb{E}_t[p^{noise}] = m_t \epsilon^{noise}, \tag{2.24}$$

where $\epsilon^{noise} \sim U(0.96, 1.04)$.

While the above equations present expectations for each individual strategy, each LF trader cluster determines its future market price expectation according to

$$\mathbb{E}_t[p] = \frac{\sum_{s \in S} w_s \mathbb{E}_t[p^s]}{\sum_{s \in S} w_s}, \tag{2.25}$$

where fundamental buyers and sellers only employ the EOD fundamentalist, trend following and reversal strategies, noise traders only employ the EOD and ID noise strategies and day traders employ all EOD and ID strategies.

The weights associated with each strategy are themselves parameters that can be calibrated, but reasonable estimates for each trader type are presented in Table 1 of the work by Mandes (2015).

Following the determination of the expected future market price of the asset, a decision of whether to post a buy or sell order must be made. Fundamental buyers (sellers) always submit buy (sell) orders, noise traders randomly submit buy or sell orders with equally probability, and day traders submit buy orders if $\mathbb{E}_t[p] - p_t > 0$ and submit sell orders if $\mathbb{E}_t[p] - p_t < 0$.

After an LF trader cluster has made a decision to place a buy or sell order, the size of the order is drawn from an exponential distribution with mean β_{size}, truncated above by b_{size}. Both b_{size} and β_{size} are constants based on empirical findings for each trader type by Kirilenko et al. (2011). In the case of fundamental buyers or sellers, the mean is slightly modified according to the calculated price expectation, with $\tilde{\beta}_{size} = (\mathbb{E}_t[p]/p_t)\beta_{size}$. All order sizes are rounded, with the ceiling function being used if the order size is smaller than one.

Once the size of the order has been determined, the order must be placed. Order placement is based on prior work by Mandes (2014) and involves the solving of an optimization problem, determining whether the order will be placed as a limit or market order and what the posting depth of the associated limit order will be. This problem is based upon minimizing price impact and execution costs. This sophisticated mechanism is very much in line with modern market microstructure approaches and is thus very desirable (Cartea et al. (2015)).

We aim to solve

$$\min_{M,\Delta} f(M, \Delta), \tag{2.26}$$

an optimal acquisition/liquidation problem.

The objective function is given by

$$f(M, \Delta) = cost(M, \Delta) + \lambda_u risk(M, \Delta), \tag{2.27}$$

where M is the fraction of the total order to be placed as a market order, Δ is the relative limit distance of the candidate limit order and λ_u is the urgency.

The cost function is given by

$$cost(M,\Delta) = \begin{cases} imp.sh(M,\Delta) & imp.sh \leq \sigma_{is} \\ \beta\sigma_{is}(imp.sh(M,\Delta)/\sigma_{is})^{\theta} & imp.sh > \sigma_{is} \end{cases} \tag{2.28}$$

and the implementation shortfall is defined as

$$imp.sh(M,\Delta) = BM_{is} + Mmk.imp(MV) - (1-M)\Delta, \tag{2.29}$$

where BM_{is} is the benchmark price, V is the order volume, $mk.imp(.)$ is the market impact function, σ_{is} is a multiple of the short term volatility and $\beta > 1$ and θ are constants.

Now, we define σ_{dyn} to be a multiple of the short-term volatility, BM_{dyn} to be the benchmark price, N to be the desired depth level, BQ_{Δ} to be the volume of orders situated in front of the client limit order and μ, ω, η, α_0, α_1 and α_2 to be constants.

The risk associated with the uncertain execution of the limit order is therefore given by

$$risk(M,\Delta) = (1-M)flow(OBI)size((1-M)V)(\alpha_0 + \alpha_1 dyn(\Delta) + \alpha_2 queue(\Delta)), \tag{2.30}$$

where the market dynamics effect is given by

$$dyn(\Delta) = \sigma_{dyn}\left(\frac{\Delta - BM_{dyn}}{\sigma_{dyn}}\right)^{\omega}, \tag{2.31}$$

the order flow is given by

$$flow(OBI) = \mu^{OBI}, \tag{2.32}$$

the order book imbalance is given by,

$$OBI = (-1)^{\mathbb{I}_{sell}}\frac{\sum_{i=1}^{N} bid_i - \sum_{i=1}^{N} ask_i}{\max(\sum_{i=1}^{N} bid_i, \sum_{i=1}^{N} ask_i)}, \tag{2.33}$$

the order queue effect is given by

$$queue(\Delta) = BQ_{\Delta}, \tag{2.34}$$

and the opportunity component is given by

$$size(\nu) = \exp(\nu^{\eta}). \tag{2.35}$$

It is apparent that the above optimization problem is fairly difficult to solve. Considering that it needs to be solved every time an order is placed, Mandes (2014)

made a necessary simplification, restricting the possible values of M to only 0 or 1, greatly simplifying the problem.

We are now required to find the optimal relative limit distance, denoted Δ^*, numerically with fixed M. Once Δ^* is calculated, we need only consider three possible cases and take the minimum to solve the problem.

Letting $A = \lambda_u flow(OBI) size(V)$, we have the following cases:

1.

$$f(M=1) = \begin{cases} BM_{is} + mk.imp(V) & mk.imp(V) \leq \sigma_{is} - BM_{is} \\ \beta(BM_{is} + mk.imp(V))^2/\sigma_{is} & otherwise \end{cases} \tag{2.36}$$

2.

$$f(\Delta|M=0, \Delta \geq BM_{is} - \sigma_{is}) = BM_{is} - \Delta + A(\alpha_0 + \alpha_1 dyn(\Delta) + \alpha_2 queue(\Delta)) \tag{2.37}$$

3.

$$f(\Delta|M=0, \Delta < BM_{is} - \sigma_{is}) = \beta\frac{(BM_{is} - \Delta)^2}{\sigma_{is}} + A(\alpha_0 + \alpha_1 dyn(\Delta) + \alpha_2 queue(\Delta)) \tag{2.38}$$

For the sake of brevity, certain details, including the methodology used to solve the above optimization problem, have been omitted in the above discussions. The interested reader should consider a more thorough reading of Mandes (2014) and Mandes (2015).

2.4.4 HF Trader Agent Specification

The implementation of HF trader agents in the Mandes (2015) model is limited to ELP traders, similar to the HF trader agents specified in the work of Vuorenmaa and Wang (2013), though it is suggested that predatory strategies could also be included in future work. Unlike LF traders in the model, HF traders are implemented as individual agents and not as clusters.

ELP trader agents essentially calculate a reservation price and an associated spread, after which limit orders to buy at the lower end of the spread and sell at the higher end of the spread are submitted. In this way, they play a quasi market making role.

It should be noted that ELP trader agents do not place orders at every activation, but update their quotes by placing orders with a certain probability. This probability is given by

$$Prob_t(I_t) = \max(1, (I_t/I_{\max}^{ELP})^2), \tag{2.39}$$

where I_t is the current inventory imbalance and $I_{\max}^{ELP}$ is the maximum allowed inventory threshold.

Whenever an ELP trader agent activates and is required to update their quotes, they first calculate their reservation price according to

$$r_t(I_t) = m_t - \gamma^{ELP} I_t \sigma_t^2, \tag{2.40}$$

where m_t is the current mid price, σ_t^2 is the instant volatility and γ^{ELP} is the risk aversion coefficient.

Thereafter, the agent sets their spread according to

$$s_t + 2e^{|\xi|}, \tag{2.41}$$

where s_t is the current bid-ask spread in the LOB and $\xi \sim \mathcal{N}(0, 0.0001)$.

Unlike LF traders, ELP traders have a fixed order size, β_{size}^{ELP}, and follow the tick-by-tick time series of market prices as opposed to the minute-by-minute and day-by-day time series followed by LF traders.

2.4.5 Order Matching and Price Determination

The matching engine employed in the model could be seen as a hybrid of the approaches employed in the Preis et al. (2006) and Jacob Leal et al. (2015) models. A similar methodology to that used in implementing the Preis et al. (2006) model is employed when a market order arrives at the LOB and a similar methodology to that used in implementing the Jacob Leal et al. (2015) model is employed when a limit order arrives at the LOB.

As previously stated, the market price of the asset at any given time is the current mid price of limit orders in the LOB, sampled at tick-by-tick, minute-by-minute and day-by-day frequencies.

2.4.6 Model Discussion

The Mandes (2015) model stands as one of the most ambitious attempts at modeling a continuous double auction market using an agent-based approach, with the preceding subsections aiming to primarily illustrate the key ideas underpinning the model and provide a sense of its expansiveness in comparison to the other models

considered. Given this ambitious scope, it is not surprising that the model offers a number of key advantages over other attempts at implementing realistic, intraday simulations of financial markets.

Firstly, the model offers a more realistic implementation of time, especially with regard to HF trader implementation. As previously discussed, three separate time series are recorded, at one-minute, daily and tick-by-tick frequencies, with different agents operating on different time scales. For example, fundamental buyers and sellers tend to follow daily trends whereas HF traders tend to follow the tick-by-tick series. Ultimately, this is far more realistic than the session structures present in many other financial ABMs, which can be particularly problematic when HF traders reference the same session timeline as LF traders, as this limits the differences in granularity and order placement speed that can exist between HF and LF traders.

Secondly, the model makes attempts at a far more sophisticated order placement mechanic for LF traders in comparison to traditional models, using a simplified optimal liquidation/acquisition framework (Cartea et al. (2015)) in order to determine whether a market or limit order should be placed and what the optimal placement depth should be in the case of a limit order. This is far more realistic than the random walk employed by Jacob Leal et al. (2015) and more sophisticated than the methodology of Preis et al. (2006).

Finally, the rules defining HF traders in the Mandes (2015) model are far more realistic than other existing approaches, such as that of Jacob Leal et al. (2015). The latter entirely neglected considerations of inventory imbalance (Easley et al. (2012)) and hence did not accurately portray the market making features of ELP traders, who tend to post bids and asks based on their current inventory in order to profit off the spread.

The above features of the Mandes (2015) model motivated its consideration as a candidate for calibration in this project. Despite this, actual implementation of the model presented a number of difficulties that required careful consideration.

The most significant problem encountered during the implementation of the model was the ambiguity related to many of its details, particularly in relation to initialization. Very little detail is given with regard to how the LOB is initialized, particularly problematic since the order placement mechanism for LF traders requires a fairly well-populated order book to function meaningfully. Equally problematic is the case of either side of the order book becoming empty due to frequent cancelations or large market orders, since no clear instructions are given with regard to dealing with an empty LOB.

This is not a problem exclusive to the Mandes (2015) model, however, and highlights a fundamental problem with agent-based modeling literature in general, previously identified by Hamill and Gilbert (2016). While ABMs are indeed math-

ematical models with parameters and equations, unlike traditional models, they also have an overarching algorithmic structure that links these components into a cohesive simulation. Very often, however, because there is a tendency, possibly incorrectly, to view ABMs as mathematical models in the traditional sense, most of the literature simply presents equation and parameter descriptions, with very little provided with regard to the overall algorithm defining the sequence of events. This leads to the aforementioned ambiguities.

In most cases, the models in question are simple enough for one to infer these missing details, but the sophistication of the Mandes (2015) model makes it very difficult to implement in a faithful, or even practical way, limiting its use in this investigation.

Apart from the above major concern, the model also presents a number of other smaller concerns.

Firstly, the decision to represent LF traders as four clusters as opposed to a set of interacting, heterogeneous agents did not seem troubling at first, but after more careful consideration, this seems to be a fairly controversial point of the model. In general, ABMs are generally known for their ability to replicate complex behavior through emergent behaviors as a result of agent interaction. The replacement of a population of heterogeneous agents with order generating clusters is a heavy departure from the fundamentals of agent-based modeling and insufficient justification is provided for this decision.

Secondly, the LF trader order placement mechanism, though promising in theory, makes use of an unusual percentage return scaling of the parameters. We do not include these in our discussion of the model for the sake of brevity and clarity of explanation, but the interested reader is directed to Mandes (2014). This scaling is not well motivated and in many cases seems counter intuitive. A Java implementation[2] by the original author was investigated in which the same methodology was applied to an earlier model, but the scaling choices seemed to result in many ad hoc special cases needing to be added to the implementation to compensate for many possible problems resulting from the choice of scaling. In other words, additional software overheads were required to manage the increased rules-based complexity.

Therefore, it was decided, based on the fact that the model results could not be recovered, even when attempting to make appropriate assumptions, that the model no longer be considered in favor of the Preis et al. (2006) model, which could be recovered fairly easily.

[2] https://github.com/mandes/MandesCDA

Chapter 3

Calibration Experiment Design

The following discussions are largely reproduced (with minor alterations) from Platt and Gebbie (2016a) and provide a comprehensive description of the approach taken to calibrate both the Jacob Leal et al. (2015) and Preis et al. (2006) models.

3.1 Data

3.1.1 Sampling Methodology

The dataset used in all calibration experiments is acquired in the TRTH (Thomson Reuters (2016)) format, presenting a tick-by-tick series of trades, quotes and auction quotes. We consider a single stock, Anglo American PLC (listed on the JSE), over the period beginning at 9:10 on 1 November 2013 and ending at 16:50 on 5 November 2013.

We convert the dataset to a series of one-minute price bars, with each price corresponding to the final quote mid price for each minute, where the mid price of a quote is given as the average of the level 1 bid price and level 1 ask price associated with that quote. From this series of prices, we may obtain a series of log prices, which is the series we attempt to calibrate the model to. The rationale behind this sampling strategy is as follows:

Trades appear more infrequently in the transaction data than in both models, where trades occur every minute. For this reason, the sampling of trade prices becomes problematic, with the vast majority of the one-minute periods considered in the data containing no trades. Quotes were therefore chosen instead, as we typically have at least one quote available during each minute of trading in the transaction data.

This better reflects the state of the LOB as the fundamental component of the system being observed instead of the emerging trade price dynamics. In other

words, the mid price best represents the state of the order book at the end of each one minute bar, analogous to the state of the simulated markets generated by the ABMs at the end of each session or MC step.

The consideration of quote mid prices is also an attempt to be consistent with the fact that trade prices in the Jacob Leal et al. (2015) model are essentially also mid prices and the final quote mid price was considered as the market price for each minute since the final trade price was selected as the market price in the Jacob Leal et al. (2015) model.

This is also consistent with the Preis et al. (2006) model, since market prices within it are set to be the mid price of limit orders in the LOB at the end of each MC step, where the limit orders considered in determining the mid price are analogous to the level 1 bid and ask prices.

The transaction dataset often presents events occurring outside of standard trading hours, 9:00 to 17:00, but we consider only quotes with a timestamp occurring in the period from 9:10 to 16:50 on any particular trading day. This is a result of the opening auction that occurs between 8:30 and 9:00 that produces erratic mid prices until the market clears at 9:00, after which it take a few minutes for the LOB to stabilize due to poor opening market liquidity. For this reason, we ignore the first 10 minutes of continuous trading. The period from 16:50 until 17:00 is the closing call auction period, with the market clearing after a randomization period occurring just after 17:00, again causing us to ignore this period.

Therefore, the one-week period in our dataset corresponds to a total of 2300 one-minute price bars, representing 460 minutes of trading each day from Monday to Friday.

3.1.2 Sampling Challenges

It is evident that reconciling an event-based, intraday transaction dataset and models that produce series of chronologically sampled prices is by no means intuitive and for this reason the above assumptions were necessary.

This highlights the fundamental problem that intraday datasets are best approached from an event time perspective while most intraday ABMs still retain the calendar time perspective common within daily ABMs, sometimes incorporating basic event time elements. While obtaining a series of daily prices is relatively straightforward, the construction of one-minute price bars often presents one with a number of decisions that need to be approached pragmatically. This usually requires a form of smoothing, sometimes on the sides of both the model and data, in order to introduce compatibility between the two, which may lead to potential errors.

Similarly, it is also worth mentioning the regime change that occurs from one day to the next in an actual double auction market. The opening and closing auc-

tions occurring at the beginning and end of each trading day demarcate separate periods of continuous trading. For our purposes, we consider the price occurring at 16:50 on one day and the price occurring at 9:10 the next day to be consecutive prices, though this is not the case, strictly speaking. This so-called overnight gap can be dealt with in a variety of ways in practice.

When considering returns, there may be a significant shift in the mid price between trading days, which might result in an erroneously large return which may skew the data and affect its moments. These values are typically ignored and removed in the dataset in investigations considering measured returns. It was mentioned earlier, however, that we consider log prices and not log returns.

This stems not from data-related considerations, but from the fact that the calibration method of Fabretti (2013), who also considered log prices, is typically very unstable when considering log returns. This is because log returns are typically very small for the one-minute time periods considered and the weight matrix employed by the method of Fabretti (2013) involves the inversion of matrices that become increasingly closer to singular as the magnitudes of the data items in the measured series decrease.

Therefore, while Fabretti (2013) chose log prices for convenience, log returns result in a weight matrix that is simply too unstable to be used for our purposes, necessitating our use of log prices. While one might consider detrending the measured log price time series to compensate for the overnight gap, this again results in the magnitude of the data items being considered in the measured series resulting in a problematic weight matrix.

Therefore, detrending is not practical for our purposes and we retain the log prices as is. While we acknowledge the existence of the overnight gap and associated regime change, it must still be noted that the associated data points represent less than 0.15% of the total dataset and we therefore consider this a relatively minor issue in this case. It is also worth noting that the moments associated with the measured log prices, presented in Table 6.4, are sensible and in line with those obtained by Fabretti (2013).

While we have endeavored to be transparent and pragmatic in our approach, the fundamental difference in paradigm between the models considered and real intraday data must be noted and is worth considering in the development of future models to improve model realism.

3.2 Calibration Framework

We consider the approach presented by Fabretti (2013). Considering the framework's success in existing calibration experiments involving the Farmer and Joshi (2002) model, we reproduce it without alteration where possible in an attempt

to ascertain if the framework produces equally successful results when applied to intraday models.

The calibration problem is formally defined as follows:

$$\min_{\theta \in \Theta} f(\theta) \tag{3.1}$$

where θ is a vector of parameters, Θ is the space of feasible parameters and f is the objective function.

3.2.1 Objective Function Construction

We construct the objective function for the dataset considered using the method proposed by Winker et al. (2007).

As previously stated, the transaction dataset is obtained in the TRTH (Thomson Reuters (2016)) format, presenting a series of trades, quotes and auction quotes. We convert the dataset to a series of one-minute price bars, with each price corresponding to the final quote mid price for each minute, where the mid price of a quote is given as the average of the level 1 bid price and level 1 ask price associated with that quote. From this series of prices, we may obtain a series of log prices [1], which is the series we attempt to calibrate the model to.

We make use of a combination of $k = 5$ moments and test statistics to represent the statistical properties of the log price time series previously mentioned, remaining consistent with Fabretti (2013). We use:

1. The mean, representing the central tendency of the measured series.
2. The standard deviation, giving an indication of the spread of the measured series.
3. The kurtosis, acting as a measure of the mass contained in the tails of the probability distribution of the measured series.
4. The Kolmogorov-Smirnov (KS) test, comparing the empirical CDFs of the log price time series obtained from the transaction data and another simulated using the model.
5. The generalized Hurst exponent (Di Matteo (2007)), with $q = 1$, representing the scaling properties of the measured series.

We define the:

[1] While Fabretti (2013) chose log prices for convenience, it was found that the weight matrix required by the method was extremely ill-conditioned when considering log returns, motivating our choice to retain this convention.

1. *realized log prices*, $\mathbf{R} = \{r_t\}, t = 1, ..., T$, to be a realization of the log price process, drawn from transaction data.

2. *exact moments*, $\mathbf{m} = (m_1, ..., m_k)$, to be the true values of the k selected moments of the log price process. We cannot precisely know these true values; we can only estimate their values according to a particular realization of the log price process.

3. *estimated moments*, $\mathbf{m}^e = (m_1^e, ..., m_k^e)$, to be the estimate of $\mathbf{m}$ based on $\mathbf{R}$.

4. *simulated moments*, $\mathbf{m}_i^s(\theta)$, to be the estimate of the k selected moments for the simulated series, generated using parameter set θ, where the index i corresponds to the estimate associated with a particular simulation run.

Using the method of simulated moments, we require:

$$\mathbb{E}[(\mathbf{m}^s|\theta)] = \mathbf{m} \tag{3.2}$$

which yields:

$$\mathbb{E}[(\mathbf{m}^s|\theta) - \mathbf{m}] = 0. \tag{3.3}$$

Assuming $\mathbf{m}$ is given and calculating the required expectation as the arithmetic average:

$$\frac{1}{I}\sum_{i=1}^{I}[(\mathbf{m}_i^s|\theta) - \mathbf{m}] = 0. \tag{3.4}$$

Using our estimate, $\mathbf{m}^e$, we set:

$$\hat{G}(\theta) = \frac{1}{I}\sum_{i=1}^{I}[(\mathbf{m}_i^s|\theta) - \mathbf{m}^e]. \tag{3.5}$$

For most of our experiments, we consider $I = 5$, with any deviations from this convention stated. This limited number of Monte Carlo replications is selected primarily due to computational constraints, since a single simulation is capable of taking a fairly significant amount of time to complete.

Roughly 7 hours of computational time is required by each Nelder and Mead (1965) simplex experiment and roughly 10 hours of computational time is required by each of the genetic algorithm experiments, despite being parallelized and run on a relatively powerful desktop computer [2].

While this is not ideal, computational difficulties remain a major obstacle in ABM calibration (Grazzini et al. (2015)) and are difficult to solve in practice,

[2]HP Z840 workstation with 2 Intel Xeon E5-2683 CPUs, providing $2 \times 16 = 32$ workers.

especially when simulations are complex and involve realistic matching processes. Nevertheless, it is important to note this relevant caveat.

Our objective function is finally given by:

$$f(\theta) = \hat{G}(\theta)^T \mathbf{W} \hat{G}(\theta) \tag{3.6}$$

where $\mathbf{W}$ is a $k \times k$ matrix of weights.

$\mathbf{W}$ is set to be the inverse of the variance-covariance matrix of $\mathbf{m}^e$, denoted $Cov[\mathbf{m}^e]$. This selected weight matrix takes the uncertainty of estimation associated with $\mathbf{m}^e$ into account and assigns larger weights to moments associated with greater uncertainty and vice-versa.

We can estimate $Cov[\mathbf{m}^e]$ by applying a moving block bootstrap (Kunsch (1989)) to $\mathbf{R}$ with a window length of b to create n bootstrapped samples and then use these bootstrapped samples to calculate approximate distributions for each estimated moment or test statistic. Thereafter, we can use these distributions to calculate a covariance matrix [3] approximating $Cov[\mathbf{m}^e]$.

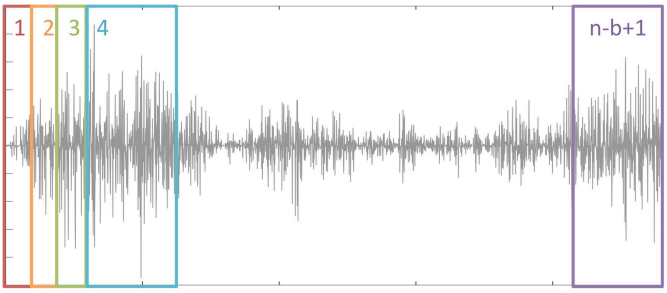


Figure 3.1: An illustration of the moving block bootstrap (Kunsch (1989)) principle, where block 1 runs from data item 1 to b, block 2 runs from data item 2 to $b+1$ and block $n-b+1$ runs from data item $n-b+1$ to n, where n is the number of data items

We set $b = 100$ and $n = 10000$, again consistent with Fabretti (2013).

In its simplest form, bootstrapping involves the resampling of the items of a particular series, with replacement, until a new series of the same length is obtained. In our case, we would obtain a new realization of the log price process by bootstrapping $\mathbf{R}$, which we could use to obtain new values for each of the moments in $\mathbf{m}^e$. Repeating this process a sufficient number of times would allow us to obtain a distribution for each moment in $\mathbf{m}^e$.

We cannot assume our selected moments are independently distributed, however. We thus use what is known as a moving block bootstrap in which we consider

[3] Specifically, we set $Cov[\mathbf{m}^e]_{i,j} = Cov[m_i^e, m_j^e]$, where $Cov[a, b]$ corresponds to the empirical covariance of a and b.

windows of a particular length containing b data items rather than individual values and resample entire windows with replacement until we obtain a series of the same length as the original. The process is illustrated in Figure 3.1.

Relating the preceding discussions to our calibration experiments, we present the weight matrix for our considered dataset below. The diagonal elements correspond to the mean, standard deviation, kurtosis, KS test statistic and Hurst exponent respectively.

$$\mathbf{W} = \begin{pmatrix} 5.0346 \times 10^4 & -1.2885 \times 10^4 & -736.6343 & 3.0220 \times 10^3 & 391.2534 \\ -1.2885 \times 10^4 & 7.8957 \times 10^5 & 2.5435 \times 10^3 & -341.4378 & -6.1999 \times 10^3 \\ -736.6343 & 2.5435 \times 10^3 & 28.7473 & -88.4746 & 17.2640 \\ 3.0220 \times 10^3 & -341.4378 & -88.4746 & 723.4611 & 56.7301 \\ 391.2534 & -6.1999 \times 10^3 & 17.2640 & 56.7301 & 2.3549 \times 10^3 \end{pmatrix}$$

It should be noted that the above weight matrix is similar to that obtained by Fabretti (2013). Our weight matrix is obtained by inverting $Cov[\mathbf{m}^e]$, which had a condition number of 1.2772×10^5. While this may seem problematic, it must be noted that Fabretti (2013) inverts $Cov[\mathbf{m}^e]$ with a condition number of 2.6964×10^5 to obtain the weight matrix, indicating that our weight matrix is at least as stable.

3.2.2 The Nelder-Mead Simplex Algorithm

The first optimization method we employ in attempting to solve the optimization problem presented in Eqn. (3.1) is the Nelder and Mead (1965) simplex algorithm.

This algorithm is based upon the fundamental idea that the current solution consists of the points of a simplex with $n+1$ vertices in the parameter space, Θ, where n is the number of parameters being calibrated in our context.

Table 3.1: Nelder-Mead Simplex Vertex Transformations

Operation	Equation
Mean	$\bar{\theta} = \frac{1}{n}\sum_{i=1}^{n} \theta^{(i)}$
Reflection	$\theta^R = (1+\rho)\bar{\theta} - \rho\theta^{(n+1)}$
Expansion	$\theta^E = (1+\rho\xi)\bar{\theta} - \rho\xi\theta^{(n+1)}$
Out-contraction	$\theta^O = (1+\psi\rho)\bar{\theta} - \psi\rho\theta^{(n+1)}$
In-contraction	$\theta^I = (1-\psi\rho)\bar{\theta} + \psi\rho\theta^{(n+1)}$
Shrink simplex toward $\theta^{(1)}$	$\theta^{(i)} = \theta^{(1)} - \sigma(\theta^{(i)} - \theta^{(1)}), i = 2, ..., n+1$

Computation of transformed points as required by the Nelder-Mead simplex algorithm, where we set $\rho = 1$, $\xi = 2$, $\psi = \frac{1}{2}$ and $\sigma = \frac{1}{2}$, as is the standard for the algorithm (Gao and Han (2010))

Algorithm 1 Nelder-Mead Simplex Algorithm

while stopping criteria not met **do**
  Rename vertices such that $f(\theta^{(1)}) \leq ... \leq f(\theta^{(n+1)})$
  if $f(\theta^R) < f(\theta^{(1)})$ **then**
    if $f(\theta^E) < f(\theta^R)$ **then**
      $\theta^* = \theta^E$
    else
      $\theta^* = \theta^R$
    end if
  else
    if $f(\theta^R) < f(\theta^{(n)})$ **then**
      $\theta^* = \theta^R$
    else
      if $f(\theta^R) < f(\theta^{(n+1)})$ **then**
        if $f(\theta^O) < f(\theta^{(n+1)})$ **then**
          $\theta^* = \theta^O$
        else
          shrink
        end if
      else
        if $f(\theta^I) < f(\theta^{(n+1)})$ **then**
          $\theta^* = \theta^I$
        else
          shrink
        end if
      end if
    end if
  end if
  if not shrink **then**
    $\theta^{(n+1)} = \theta^*$
  end if
end while

During each iteration, we calculate the value of f, the objective function, at the vertices of the simplex and consider the vertex with the worst objective function value. We then consider the reflection, expansion, in-contraction and out-contraction points of this vertex. If this leads to a point where the value of the objective function is improved, we update the vertex with the worst objective function value, but if none of these points leads to an improvement, the simplex shrinks. We consider the version of the algorithm presented by Gilli and Winker

(2003) and perform 5 Monte Carlo replications when applying the method.

While the Nelder-Mead simplex algorithm is relatively robust in the case of a smooth objective function, it tends to converge to local minima when the objective function being considered is not smooth. Gilli and Winker (2003) highlighted this problem in the context of agent-based modeling, as typical objective functions associated with agent-based modeling problems are usually not smooth, which we illustrate in Figure 3.2. For this reason, the Nelder-Mead simplex algorithm is combined with the threshold accepting heuristic to aid convergence to a global minimum.

In threshold accepting, we randomly perturb a set of parameters and accept new parameter sets which are not worse than our current best set, as measured by the objective function, by more than a certain threshold, τ. We typically have a set of n_R thresholds, $\tau_r, r = 1, ..., n_R$. When combining this heuristic with the Nelder-Mead simplex algorithm, we begin by shifting the current simplex in a random direction, with the magnitude of this shift also being random. We then accept this new simplex if the vertex with the best objective function value in the new simplex has an objective function value less than the best in the previous simplex, after the threshold has been added.

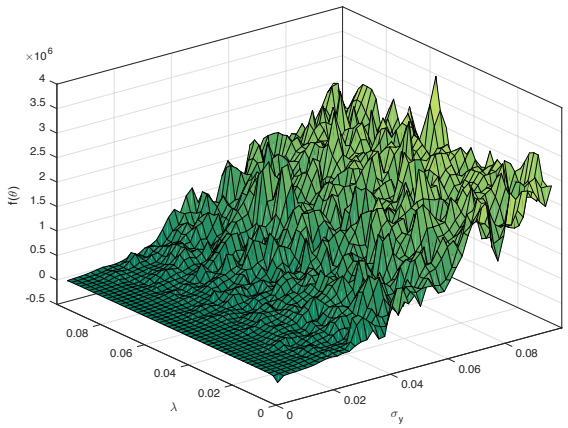


Figure 3.2: An illustration of a typical objective function found in agent-based modeling calibration problems, taken from the Jacob Leal et al. (2015) model calibration experiments in this investigation and clearly illustrating a lack of smoothness

We implement the required random shift as follows:

1. We begin by *randomly selecting* one of the free parameters considered in the optimization problem.

2. We then *multiply the mean value* of this parameter across the simplex vertices

by a random number drawn from $\mathcal{U}(-0.5, 0.5)$ and add this quantity to the current value of the selected parameter at each vertex.

Algorithm 2 Combined Algorithm (NM + TA)

Generate u from $\mathcal{U}(0, 1)$
if $u < 0.15$ **then**
  for $r = 1$ to n_R **do**
    for $s = 1$ to n_s **do**
      Generate $\theta_{new} \in \mathcal{N}_{\theta_{old}}$ using a random shift
      if $f(\theta_{new}) < f(\theta_{old}) + \tau_r$ **then**
        Accept θ_{new}
      end if
    end for
  end for
else
  Generate $\theta_{new} \in \mathcal{N}_{\theta_{old}}$ with a simplex search and check for acceptance
end if
if θ_{new} is accepted **then**
  $\theta_{old} = \theta_{new}$
end if

We perform $n_R = 3$ rounds in each application of the threshold accepting algorithm, each consisting of $n_s = 5$ steps, with $\tau_1 = \frac{1}{5}$, $\tau_2 = \frac{1}{10}$ and $\tau_3 = 0$. We perform 3 Monte Carlo replications in round 1, 4 in round 2 and 5 in Round 3.

A single step in the combined algorithm (Gilli and Winker (2003)), involving both the Nelder-Mead simplex algorithm and threshold accepting heuristic, is described in Algorithm 2.

3.2.3 Genetic Algorithm

We refer to the class of algorithms proposed by Holland (1975). In essence, genetic algorithms are parameter selection and perturbation strategies, inspired by evolutionary processes in nature, used to solve optimization problems. Like organisms in the natural world, superior solutions survive and pass on characteristics, whereas inferior solutions tend to become extinct.

The basic principles of genetic algorithms, as summarized by Whitley (1994), are given as follows:

1. We begin with an *initial population*, usually randomly generated, consisting of possible solutions to the optimization problem, which are called chromosomes in the context of the literature.

2. Chromosomes that represent better solutions to the target problem (as measured by a fitness function), are given *better opportunities* to reproduce (crossover) as compared to inferior chromosomes.

3. *Reproduction* (crossover) ultimately results in a new population that hopefully contains improved chromosomes, in other words, those that result in lower (higher) objective function values in the context of a minimization (maximization) problem.

Genetic algorithms have become increasingly prevalent in contemporary literature, a result of the fact that they are both intuitive and relatively robust. They do not require gradient information and are thus well suited to functions with many local minima (or maxima) and non-differentiable functions.

For this reason, genetic algorithms are typically widely applied, unless an applicable gradient or problem-specific method exists (Whitley (1994)). Considering that the constructed objective function in our framework does indeed have many local minima and considering the lack of any specialized algorithms for ABM calibration, a genetic algorithm seems to be a viable candidate.

When applying genetic algorithms, we usually discretize the domain of a particular parameter using some form of encoding. Earlier literature focuses on representing each parameter value as a sequence of binary digits in a process we call binary digit encoding. The binary representations of all parameters to be calibrated are of the same length, L binary digits, where L is an integer, allowing for 2^L possible values for each parameter.

It therefore follows that a possible solution to an optimization problem with $L = 2$ and three parameters will consist of a series of six binary digits, with the first two digits representing the first parameter and so on.

Figure 3.3 shows an example of how we might apply binary digit encoding to the Jacob Leal et al. (2015) model. In this example, we showcase the fact that each encoded string represents an entire parameter set, with every sequence of L bits in the string representing a different parameter.

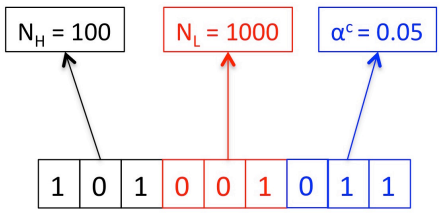


Figure 3.3: An example of binary digit encoding for a candidate solution in the context of the Jacob Leal et al. (2015) model, where each parameter will have 8 possible values, since $L = 3$

One does not necessarily need to use binary digit encoding methods. There exist many other alternatives, an example being value encoding, where we simply use the parameter values themselves.

While many genetic algorithm variants exist, we consider the Simple Genetic Algorithm (SGA), described by Goldberg (1989) as follows:

Algorithm 3 Simple Genetic Algorithm

1: Randomly initialize the population
2: Determine the fitness values of the current population
3: **while** best individual in the population does not meet the required tolerance **do**
4: **Select** parents at random to create an intermediate population
5: Perform a **crossover** operation on the parents to generate offspring
6: Apply the **mutation** operation to the population of offspring
7: Calculate the fitness values for the new population
8: **end while**

The genetic operators highlighted in the above algorithm are described in detail as follows:

Selection During a selection operation, chromosomes are randomly selected with replacement and probability proportional to their current fitness values to create an intermediate population.

Crossover During a crossover operation, chromosomes are first randomly paired for recombination. Since the selection process sufficiently shuffles the population, the pairs are simply set to be the first and second chromosomes in the intermediate population, the third and fourth chromosomes in the intermediate population and so on. Thereafter, the strings recombine, with probability p_c, to form new chromosomes that constitute the new population. The specific recombination process is dependent on the implementation. A simple illustration of this process is presented in Figure 3.4.

Mutation During the mutation operation, a slight perturbation is applied to the population, with a very small probability, p_m. This typically involves the slight shifting of a single parameter value associated with a single chromosome in the population. A simple illustration of this process is presented in Figure 3.5.

Fitness Function It should be noted that fitness and objective functions need not be the same. Fitness functions typically measure the performance of a chromo-

some relative to the population whereas objective functions tend to measure the performance of the chromosome in the context of the optimization problem itself. In our case, however, we set the fitness function to be identical to the objective function defined for the calibration framework.

Implementation of Operators We make use of MATLAB's *ga* function, an implementation of the preceding algorithm, packaged as part of the *Global Optimization Toolbox*. The problem encoding as well as the crossover and mutation operators in our context are defined as per the defaults for this function. Parallel objective function evaluations are performed using a master-slave paradigm, similar to that implemented by Hendricks et al. (2016b). We set the population size to be 100 and consider 5 Monte Carlo replications per individual.

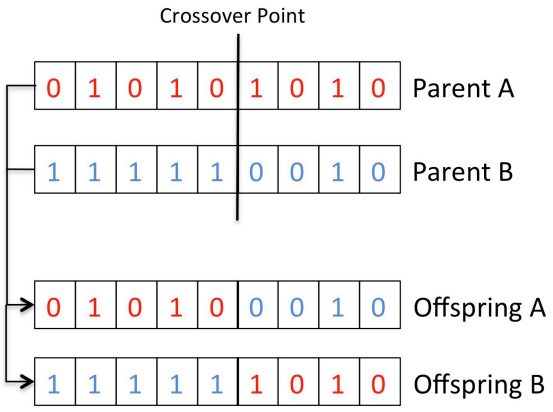


Figure 3.4: An illustration of the crossover process for a genetic algorithm employing a binary digit encoding scheme

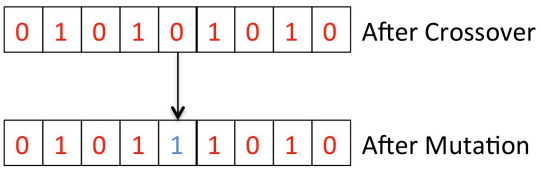


Figure 3.5: An illustration of the mutation process for a genetic algorithm employing a binary digit encoding scheme

3.2.4 Alternative Optimization Methods

While beyond the scope of this investigation, it is worth noting alternatives to the optimization methods employed.

The first of these alternatives we describe is particle swarm optimization, which Hassan et al. (2005) summarized as being inspired by animal collectives, such as flocks of birds and schools of fish, and their behaviors in finding food and avoiding predators. An initial random set of solutions (particles) propagates within the parameter space through a number of iterations and information is assimilated and shared by the particles in the swarm. As in the case of a genetic algorithm, the overall swarm (population) is improved with each iteration, but evidence suggests that particle swarm optimization has faster convergence when compared to genetic algorithms.

The second alternative we describe is the Markov chain Monte Carlo approach, which is strongly related to the approach known as simulated annealing (Kirkpatrick et al. (1983)). Like genetic algorithms, simulated annealing is a method of searching a large parameter space without resorting to a random search. These approaches share some similarities, but have different search mechanisms. In essence, simulated annealing allows for the probabilistic acceptance of inferior solutions in relation to the best known solution to an optimization problem in order to avoid convergence to local minima.

We had originally intended to employ this method as a third approach in the implemented calibration framework, but found that it demonstrated significantly poorer performance than the Nelder-Mead simplex and genetic algorithms during the early stages of the investigation and we thus excluded it.

Chapter 4

Implementation Overview

This chapter aims to provide a brief overview of the approach taken to implement the ABMs and calibration methodologies employed in this investigation. A more detailed discussion is presented in Appendix A.

4.1 Programming Language Selection

It was ultimately decided that all implementation be done in MATLAB, for a number of reasons.

Firstly, the ABMs considered center around the storage of and operations on a large amount of numerical data representing orders stored in a central LOB. This makes a matrix in MATLAB a logical choice for the storage of this data and allows for fast and efficient processing, especially relevant to order matching procedures, since MATLAB is extremely efficient when handling such data structures.

Secondly, MATLAB is well-suited to rapid prototyping. Unlike more traditional programming languages, including C++ and Java, MATLAB generally requires less emphasis on memory management and other system-related procedures and includes an extensive library of built-in functions that significantly expedited the process of implementing the selected calibration methodologies and ABMs.

Considering that this particular investigation is situated within the domain of applied mathematics rather than software engineering, this rapid prototyping is essential, ensuring that greater time can be allocated to the analysis of results and the development of scientific theories.

This does not imply that MATLAB is without its own disadvantages, however. With this increased ease of implementation comes a certain decrease in control and compatibility.

While MATAB performs sufficiently well in the context of our applications, the increased control over various aspects of execution within more traditional

programming languages implies the possibility of the construction of more efficient implementations than may be possible with MATLAB. In addition to this, MATLAB tends to have a more narrow base of compatibility with existing parallelization technologies, which are also essential to this investigation.

Despite this, MATLAB's advantages strongly outweigh its disadvantages in this particular investigation.

4.2 Object-Oriented Programming

The majority of our implementation structures make extensive use of Object-Oriented Programming (OOP), primarily to facilitate ease of parallelization and to encapsulate the many procedures and data structures required by the various aspects of the investigation in a small number of well-defined classes.

Within OOP, a class is a file or structure that specifies attributes, which are properties that might include integer values, string values or matrices, and methods, which are functions that can be programmed to perform a variety of tasks. A specific instantiation of a class is called an object, where different objects of the same class have their own independent attribute values. This is similar to how an integer number may be seen an instantiation of the integer data type and how a series of characters may be seen as an instantiation of the string data type.

Almost all of the methods and attributes required to implement full simulations using either the Jacob Leal et al. (2015) or Preis et al. (2006) models are stored within the *JacobLealModel* and *PreisModel* classes respectively. This implies that in order to begin a new simulation with different parameters, one would simply need to create a new object of either class by initializing its constructor with a different set of parameters and thereafter starting the simulation by calling the appropriate (*simulate*) method. Once the simulation is complete, all simulation results are then stored within the object.

Therefore, when evaluating the objective function for new parameter sets in both the Nelder-Mead simplex and genetic algorithms, we simply create a new *JacobLealModel* or *PreisModel* object (depending on which model we are calibrating) for each parameter set to be evaluated and call the *simulate* methods of these objects in parallel. Thereafter, the simulated log price time series stored in each object can be used to evaluate the objective function. In this, the ease of parallelization offered by OOP is evident.

In addition to the above, the objective function itself is also an object, containing all the required methods to construct a new objective function from transaction data and evaluate the objective function given a particular simulated log price time series. Without OOP, the implementation of the objective function would become significantly less elegant, requiring a large number of separate function files to be

developed as opposed to a single class file.

Similarly, calibration involving the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic is also implemented using an object, where all results from a particular calibration experiment, along with the required methods and data structures, are stored in a single object.

4.3 Implementation Hardware and Parallelization

It is worth noting that parallelization and efficient implementation are vital to the success of ABM calibration experiments since a large number of independent simulations need to be conducted. This necessitates the use of relatively powerful hardware platforms that allow for large-scale parallelization.

For this reason, all numerical experiments were conducted using an HP Z840 workstation with 2 Intel Xeon E5-2683 CPUs, providing $2 \times 16 = 32$ workers. This hardware offered full compatibility with MATLAB Distributed Computing Server (MDCS) and a MATLAB Job Scheduler (MJS) was used to submit all required scripts for processing.

Therefore, MATLAB's relatively limited scope of hardware compatibility when compared to programming languages such as C++ or Java did not impede our investigation.

Chapter 5

Stylized Fact Replication Experiments

Before attempting calibration experiments of any kind, it is vital that we verify the validity of our implementations of both the Jacob Leal et al. (2015) and Preis et al. (2006) models. We do this by demonstrating that our implementations of both models are capable of reproducing a number of well-known stylized facts of log return time series which are currently used to validate models of this class.

5.1 Description of Well-Known Return Time Series Stylized Facts

There exists fairly detailed literature describing a number of well-known stylized facts observed in financial markets. A comprehensive general survey was presented by Cont (2001) and a study of stylized facts in the context of the JSE was presented by Wilcox and Gebbie (2008).

Among the most well-known of these stylized facts is the observation that empirically-observed log return distributions are not normal as they are often assumed to be in more traditional branches of financial mathematics, but are in fact leptokurtic and fat-tailed, with a greater probability of extreme negative events than extreme positive events (Cont (2001)).

Furthermore, it has also been observed that empirically measured log returns exhibit no significant autocorrelation at most lags (Cont (2001)). The autocorrelation function of the absolute values of empirically measured log returns, however, does exhibit significant autocorrelation for shorter lags, which slowly decays as the length of the lag increases. This is evidence of volatility clustering in financial markets (Cont et al. (1997)).

While the above are perhaps among the most widely-known stylized facts and

are usually those considered in stylized-fact centric validation, they are by no means exhaustive. The LOB itself also presents many important stylized facts that are largely ignored in such validation procedures. This largely stems from the fact that stylized fact-centric methods of validation find their origins in earlier closed-form, daily ABMs that did not employ realistic order matching procedures or make use of a central LOB.

It would therefore seem apparent that more attention should be paid to LOB stylized facts in intraday ABMs of the class we are investigating if stylized fact-centric validation is to be employed. Despite this, we will limit our investigation to replicating the validation procedures originally applied to our candidate models. The interested reader is directed to the comprehensive survey of LOB stylized facts provided by Bouchaud et al. (2002).

5.2 Jacob Leal et al. Model

The results, figures and discussions presented in this section have been reproduced from Platt and Gebbie (2016a), with minor alterations.

We demonstrate the ability of our implementation of the Jacob Leal et al. (2015) model to reproduce a number of well-known log return times series stylized facts by simulating series of prices using the implemented model and using these prices to calculate series of log returns. We generate a total of 50 price paths (Monte Carlo replications) using the default parameters presented in Table 5.1.

Table 5.1: Default Parameters for the Jacob Leal et al. (2015) Model

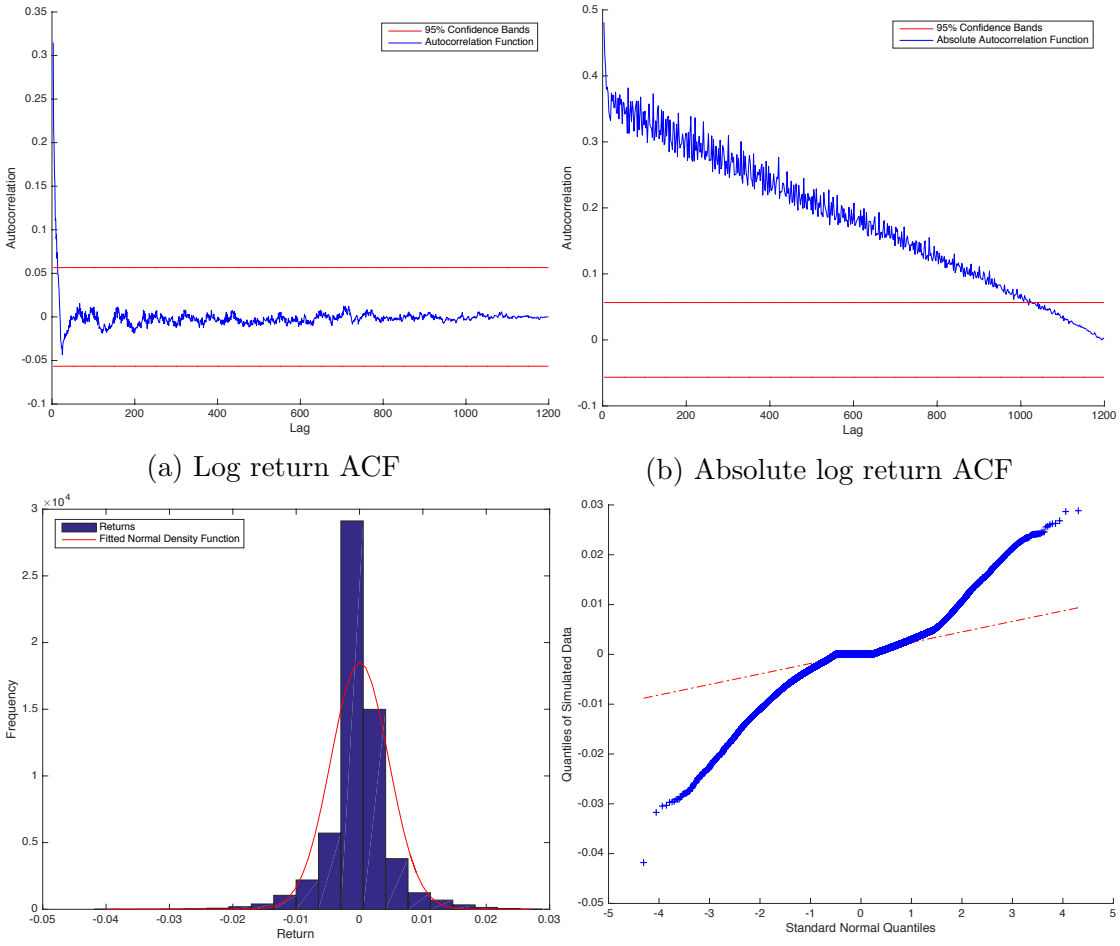
Parameter	Value	Parameter	Value
T	1200	σ^y	0.01
N_L	10000	δ	0.0001
N_H	100	σ^z	0.01
θ	20	ζ	1
$[\theta_{min}, \theta_{max}]$	$[10, 40]$	γ^L	20
α^c	0.04	γ^H	1
σ^c	0.05	$[\eta_{min}, \eta_{max}]$	$[0, 0.2]$
α^f	0.04	λ	0.625
σ^f	0.01	$[\kappa_{min}, \kappa_{max}]$	$[0, 0.01]$

Parameters for stylized fact replication using the Jacob Leal et al. (2015) model, set according to those specified in Table 4 in the work conducted by Jacob Leal et al. (2015)

We then proceed to construct a histogram approximating the distribution of the obtained log returns pooled over all the conducted Monte Carlo replications,

along with a fitted normal density function, as well as a Q-Q plot for the same simulated data.

Thereafter, we consider the autocorrelation functions of the returns and absolute values of the returns and provide appropriate 95% confidence bands.



(a) Log return ACF

(b) Absolute log return ACF

(c) Return distribution histogram

(d) Q-Q plot

Figure 5.1: Autocorrelation functions with one-minute session lags for the series of log returns (top left) and the series of the absolute values of the log returns (top right), along with a histogram approximating the distribution of the series of log returns (bottom left) and a Q-Q plot comparing the aforementioned distribution to a normal distribution (bottom right), all generated using the Jacob Leal et al. (2015) model

Referring to Figure 5.1a, we see that the autocorrelation function of the series of simulated log returns demonstrates no significant pattern or deviation from zero

at any lag. When considering Figure 5.1b, however, we observe initially significant autocorrelation for shorter lags with a steady decline in autocorrelation as the lag increases in the case of the series of the absolute values of the simulated log returns.

As is evident in Figures 5.1c and 5.1d, the distribution of the simulated log returns is leptokurtic and fat-tailed when compared to a normal distribution.

It is therefore apparent that our implementation of the Jacob Leal et al. (2015) model is able to reproduce well-known log return time series stylized facts and generates behaviors consistent with those demonstrated by Jacob Leal et al. (2015).

5.3 Preis et al. Model

We now demonstrate the ability of our implementation of the Preis et al. (2006) model to reproduce a number of well-known log return times series stylized facts through the use of the same procedure we applied to the Jacob Leal et al. (2015) model, with the default model parameters used in all Preis et al. (2006) model simulations presented in Table 5.2.

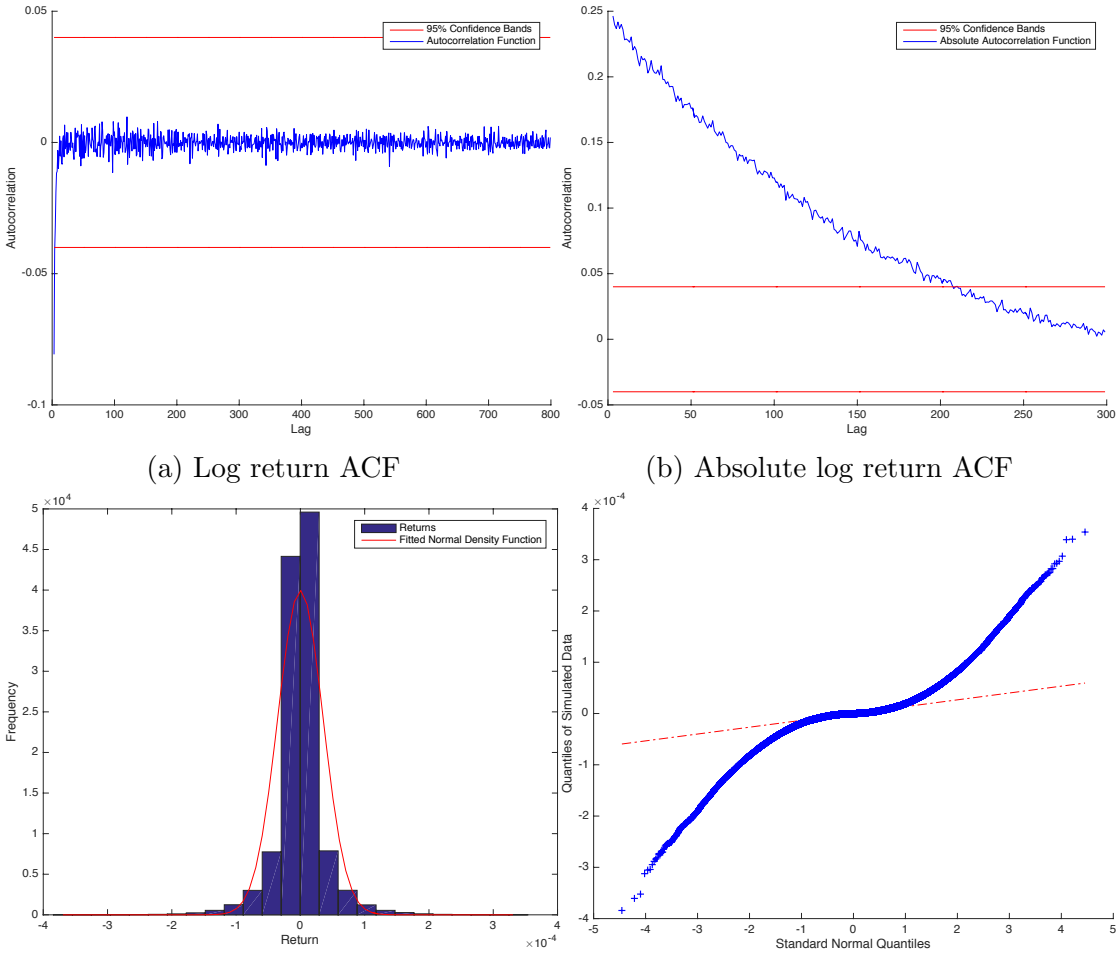
Table 5.2: Default Parameters for the Preis et al. (2006) Model

Parameter	Value
T	2400
δ	0.025
λ_0	100
C_λ	10
Δ_S	0.001
α	0.15
μ	0.025

Parameters for stylized fact replication using the Preis et al. (2006) model, set according to those specified in Figure 2 in the work conducted by Preis et al. (2006)

Referring to Figure 5.2a, we see that the autocorrelation function of the series of simulated log returns demonstrates no significant pattern or deviation from zero at any lag. When considering Figure 5.2b, however, we observe initially significant autocorrelation for shorter lags with a steady decline in autocorrelation as the lag increases in the case of the series of the absolute values of the simulated log returns.

As is evident in Figures 5.2c and 5.2d, the distribution of the simulated log returns is leptokurtic and fat-tailed when compared to a normal distribution.



(a) Log return ACF

(b) Absolute log return ACF

(c) Return distribution histogram

(d) Q-Q plot

Figure 5.2: Autocorrelation functions with one-minute session lags for the series of log returns (top left) and the series of the absolute values of the log returns (top right), along with a histogram approximating the distribution of the series of log returns (bottom left) and a Q-Q plot comparing the aforementioned distribution to a normal distribution (bottom right), all generated using the Preis et al. (2006) model

It is therefore apparent that our implementation of the Preis et al. (2006) model is able to reproduce well-known log return time series stylized facts and generates behaviors consistent with those demonstrated by Preis et al. (2006).

Chapter 6

Calibration Results

In this chapter, we present the results associated with the application of the calibration framework of Fabretti (2013) to the Jacob Leal et al. (2015) and Preis et al. (2006) models.

6.1 Jacob Leal et al. Model

The results, figures and discussions presented in this section have been reproduced from Platt and Gebbie (2016a), with minor alterations.

6.1.1 Free and Fixed Parameters

In all calibration experiments, we consider 10 free parameters, with the remainder of the parameters set to be fixed. More specifically, we set α^c, α^f, σ^c, σ^f, σ^y, δ, σ^z, λ, N_L and N_H, defined in Section 2.2, to be free parameters and randomly generate initial values for these parameters during each calibration experiment. We initialize N_H and N_L between 100 and 10000 for each population member or simplex vertex and initialize the remaining free parameters with values between 0 and 0.1. This particular set of free parameters was selected due to the fact that it represents a broad range of model characteristics, including order size, order price and trader population dynamics, while still remaining computationally tractable.

All other parameters are constant and their values are set to be identical to those presented in Table 5.1, with $\bar{P}_0 = F_0$ and $\bar{P}_1$ set to be the first two prices of the measured price time series from which we calculate the series of measured log prices.

It is important to ensure that the initial value to which the simulated price time series is set does not happen to be an outlier of the empirically-observed price time series. In order to demonstrate this, we apply the well-known boxplot

method proposed by Tukey (1977), in which we are required to determine the interval

$$[Q_1 - 1.5IQR, Q_3 + 1.5IQR],$$

where Q_1 is the lower quartile, Q_3 is the upper quartile and IQR is the interquartile range of the measured price time series.

Applying this method to our processed dataset, we obtain the interval

$$[217.2638, 278.3337],$$

with any points outside of this interval predicted to be potential outliers. Therefore, our chosen value of $\bar{P}_0 = F_0 = 238.75$ is well within the interval and we can therefore say with relative confidence that our chosen initial price is not an outlier.

6.1.2 Nelder-Mead Simplex Algorithm

Due to the fact that we consider $n = 10$ free parameters in our calibration experiments, we begin with $n + 1 = 11$ simplex vertices, each consisting of randomly generated initial values for the 10 free parameters.

Table 6.1: Nelder-Mead Simplex Calibration Results (Jacob Leal et al. (2015) Model)

Parameter	95% Conf Int	$\frac{s}{\sqrt{n}}$
α^c	$[0.0249, 0.111]$	0.0205
σ^c	$[0.0322, 0.0939]$	0.0147
α^f	$[0.0299, 0.0597]$	0.00711
σ^f	$[0.0389, 0.0747]$	0.00853
σ^y	$[0.00689, 0.0547]$	0.0114
δ	$[0.000391, 0.00212]$	0.000413
σ^z	$[0.0159, 0.0429]$	0.00643
λ	$[0.0250, 0.0696]$	0.0107
N_L	$[2420, 5693]$	782.0283
N_H	$[3611, 7746]$	987.594

95% confidence intervals for the set of free parameters, obtained from 20 independent calibration experiments involving the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic applied to the Jacob Leal et al. (2015) model

In our experiments, it was noted that 250 iterations of the Nelder-Mead simplex algorithm with threshold accepting almost always produced convergent behavior

and we thus conducted 20 calibration experiments with this fixed number of iterations. Despite the presence of convergent behavior, the obtained parameter sets were surprisingly different for various calibration experiments, with a significant dependence on the set of initial vertices. This is strongly indicative of convergence to local minima, even with the inclusion of methods aiming to overcome this problem, namely the threshold accepting heuristic.

More optimistically, however, a fairly significant number of these experiments (more than half) seemed to converge to parameter sets with objective function values in a relatively small range, a similar result to that obtained by Fabretti (2013), indicating that our implementation has, at the very least, produced similar behaviors when attempting to calibrate the model. Furthermore, the obtained parameter sets in this region of similar objective function values produced model behaviors that were fairly consistent with those demonstrated by the empirical data, with the error between the simulated and measured moments, notably the mean, standard deviation and Hurst exponent, being within similar ranges to those obtained by Fabretti (2013). An assessment of the quality of the model fit is presented in Section 6.1.4.

Referring to Table 6.1, it is apparent that the calibration process, while able to reduce the search region from $[0, 0.1]$ and $[100, 10000]$ to the regions roughly specified by the confidence intervals[1] shown, still produces results where the vast majority of these intervals are still far too large to identify any true convergence to a unique set of optimal parameters. Despite this general trend, however, we see an interesting irregularity. The parameter δ does indeed seem to demonstrate convergent behavior and the algorithm has identified a very small confidence interval for δ. It is clear that this requires more thorough investigation and is discussed in detail in Section 6.1.5.

Therefore, the obtained results are more or less indicative of an ability to find feasible parameter sets, but we find it problematic that we do not obtain stable or unique parameters and that there is evidence of a strong dependence on the initial simplex. It is evident that the method converges to reasonable solutions, but we cannot be sure that the obtained solutions truly minimize the objective function, which can be perceived as fairly problematic.

6.1.3 Genetic Algorithm

We found that convergence was observed in almost all of the genetic algorithm experiments after approximately 100 generations. For this reason, we conducted 10 calibration experiments, with random initial populations using the same ini-

[1] The intervals are calculated as: $\bar{x} \pm t^* \frac{s}{\sqrt{n}}$, where $\bar{x}$ is the sample mean, s is the sample standard deviation, n is the sample size, and t^* is the appropriate critical value for the t distribution.

tial parameter range as in the Nelder-Mead simplex calibration experiments, and iterated the populations for 100 generations. This smaller number of calibration experiments was selected due to the increased computational cost of the genetic algorithm.

Referring to Table 6.2, it is clear that the obtained calibration results point to similar conclusions as were obtained using the Nelder-Mead simplex algorithm. We obtain similar parameter confidence intervals for each calibrated parameter and again find that all parameters, with the exception of δ, seem unable to be uniquely determined.

Table 6.2: Genetic Algorithm Calibration Results (Jacob Leal et al. (2015) Model)

Parameter	95% Conf Int	$\frac{s}{\sqrt{n}}$
α^c	$[0.0367, 0.0773]$	0.00899
σ^c	$[0.0459, 0.0840]$	0.00843
α^f	$[0.0234, 0.0706]$	0.0104
σ^f	$[0.0187, 0.0708]$	0.0115
σ^y	$[0.0332, 0.0932]$	0.0133
δ	$[0.000114, 0.00108]$	0.000213
σ^z	$[-0.00222, 0.0144]$	0.00367
λ	$[0.0468, 0.0924]$	0.0101
N_L	$[469, 5337]$	1075.947
N_H	$[3335, 7881]$	1004.866

95% confidence intervals for the set of free parameters, obtained from 10 independent calibration experiments involving the genetic algorithm applied to the Jacob Leal et al. (2015) model

6.1.4 Comparison of Calibrated Model and Empirical Data

It was mentioned in the preceding sections that while no unique parameter set could be established, a number of the experiments converged to different parameter sets with similar objective function values producing reasonable behavior when compared to the empirical data. In this section, we consider the best parameter sets, presented in Table 6.3, obtained in both the Nelder-Mead simplex and genetic algorithm experiments, associated with Tables 6.1 and 6.2 respectively.

For each of these parameter sets, we simulate 50 price paths and obtain the 95% confidence interval for the estimates of the mean, standard deviation, kurtosis, and Hurst exponent of the simulated log prices and compare these to the empirical moments obtained from the transaction data.

Table 6.3: Best Parameter Sets Obtained Using Each Calibration Method (Jacob Leal et al. (2015) Model)

Calibrated Parameter	θ_{NM+TA}	θ_{GA}
α^c	0.0433	0.0025
σ^c	0.0233	0.0728
α^f	0.0068	0.0222
σ^f	0.0805	0.0228
σ^y	0.0017	0.0021
δ	0.00004	0.00002
σ^z	0.0309	0.0238
λ	0.0657	0.0149
N_L	9783	9668
N_H	5150	5949

Best parameter sets obtained through the application of the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic and the genetic algorithm to the Jacob Leal et al. (2015) model

Referring to Table 6.4, we see that the calibration strategy has resulted in log price paths with comparable means and standard deviations to that observed in the empirical data, along with a reasonable, but slightly underestimated generalized Hurst exponent estimate for the Nelder-Mead simplex parameters and a fairly accurate generalized Hurst exponent estimate for the genetic algorithm parameters. This indicates that the calibration experiments performed, though not conclusive in determining a unique parameter set, were able to produce reasonable fits for some of the moments. We also see, as was the case in the investigation of Fabretti (2013), that the genetic algorithm tends to perform slightly better than the Nelder-Mead simplex algorithm with threshold accepting.

Table 6.4: Comparison of Empirical and Calibrated Moments (Jacob Leal et al. (2015) Model)

	θ_{NM+TA}	θ_{GA}	Data
Mean	$[5.4931, 5.5436]$	$[5.4450, 5.5273]$	5.5143
Std Dev	$[0.0292, 0.0449]$	$[0.0263, 0.0532]$	0.0300
Kurtosis	$[2.1092, 2.6760]$	$[1.9029, 2.6317]$	1.2402
Hurst Exponent	$[0.5110, 0.5239]$	$[0.5551, 0.5826]$	0.5658

95% confidence intervals for the simulated moments generated by the calibrated Jacob Leal et al. (2015) model compared to the empirically-measured moments

In contrast to this, the kurtosis is more significantly overestimated. This seems to stem from flaws in the calibration methodology. In both our own investigation and that of Fabretti (2013), the kurtosis tended to be the most poorly fit moment after model calibration. This is likely explained by the fact that both our own weight matrix and that of Fabretti (2013) tend to assign a very small weight to errors on the kurtosis. Considering that the method employed was reproduced without alteration, it may be necessary for the choice of moments or the construction of the weight matrix to be revisited in future, since both our weight matrix and that of Fabretti (2013) tend to be significantly biased towards finding good fits for the mean and standard deviation at the expense of the other considered moments. A possible alternative to the currently constructed objective function is the information theoretic criterion proposed by Lamperti (2015) and applied in Lamperti (2016).

This has, however, demonstrated that the method in question has indeed found reasonable fits for the most significantly weighted moments in the objective function.

6.1.5 Parameter Analysis

In an attempt to illustrate the behavior of the objective function with respect to changes in parameter values, we generate surfaces representing the objective function values corresponding to various parameter value pairs, as has been undertaken by Gilli and Winker (2003) and Fabretti (2013). The process of generating these surfaces is as follows:

1. We choose any pair of parameters drawn from the 10 free parameters selected in the calibration experiments.

2. We then define the space of possible values for each parameter in the pair. In general, we set the space to be $[0, 0.1]$ on $\mathbb{R}$ for all parameters, with the exception of N_L and N_H, where we set the space to be $[100, 10000]$ on $\mathbb{R}$.

3. This results in a two-dimensional space whose x-axis is defined by the possible values of one of these selected parameters and whose y-axis is similarly defined by the possible values of the other parameter.

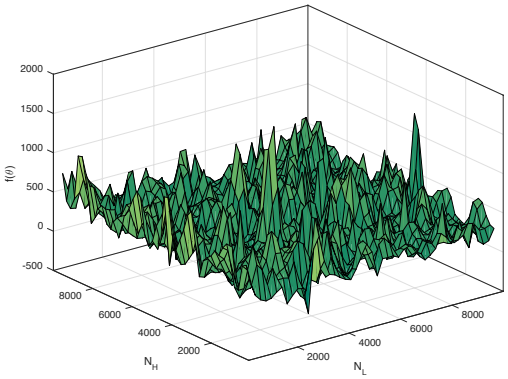
4. We then populate this two-dimensional space by generating 1000 points using an appropriate two-dimensional Sobol sequence. In the case of N_L and N_H, we round the values of the obtained points.

5. These randomly generated parameter values are then used to simulate 5 price and hence log price time series (Monte Carlo replications) using the Jacob

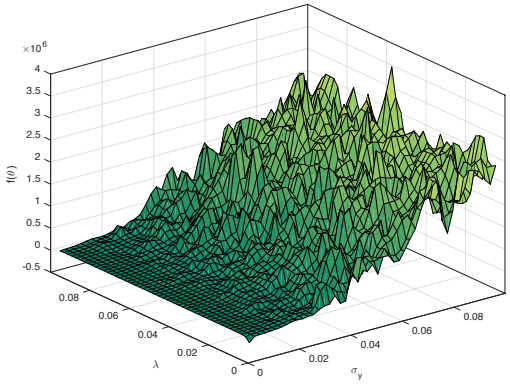
Leal et al. (2015) model, with all other free parameter values set according to θ_{NM+TA} in Table 6.3.

6. We can thus evaluate the objective function at each of the generated points and obtain an approximate surface of objective function values corresponding to all possible values of the parameters in the pair using cubic interpolation.

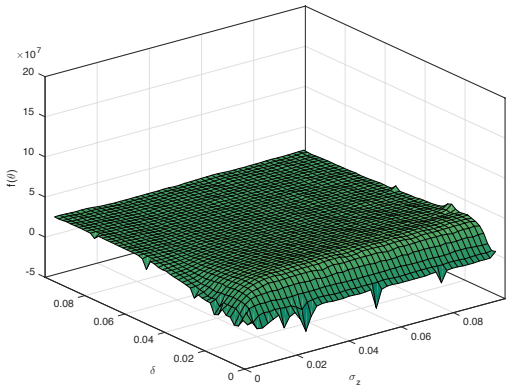
After the consideration of surfaces for a number of parameter pairs, we found that there are 3 main types of surface behaviors that emerge, with only one of these behaviors resulting in satisfactory calibration results.



(a) Type 1 behavior: N_H and N_L



(b) Type 2 behavior: λ and σ^y



(c) Type 3 behavior: δ and σ^z

Figure 6.1: Objective function surfaces for the Jacob Leal et al. (2015) model, illustrating 3 types of behavior, namely type 1, where the model parameters have a haphazard effect on the value of the objective function, type 2, where there is an underlying objective function trend that is obscured by many local minima and maxima, and type 3, where we find a well-behaved objective function that is amenable to calibration

Referring to Figure 6.1a, illustrating what we will call type 1 behavior, we see that changes in N_L and N_H have a haphazard effect on the objective function, with very little evidence of structure. It is evident that even slight changes in these parameters have the potential to result in fairly large relative differences in objective function values, leading to a surface dominated by a series of peaks and valleys, with no clear path to a global minimum or discernable trend emerging. This is the likely reason that both the Nelder-Mead simplex algorithm with threshold accepting and genetic algorithm had a tendency to converge to local minima, even with attempts made in both search methods to avoid this. It seems very unlikely that the model can be calibrated in a meaningful way with respect to parameters demonstrating this type of behavior, as the observed degeneracies are fairly severe, with such parameters generating unpredictable, near-random effects on the considered objective function. Other parameters generating similar behaviors include α^c, α^f, σ^c, σ^f and λ.

Referring to Figure 6.1b, illustrating what we call type 2 behavior, we see that there is indeed a trend presented by σ^y, albeit with many local minima and maxima obscuring this trend, resulting in a fairly noisy surface with some form of overall path to a global minimum emerging. While this type of behavior is indeed more promising than the first, both optimization methods failed to demonstrate any real convergence with respect to σ^y. This is most likely due to the presence of a very large number of local minima and a lack of smoothness hiding the observed trend from the optimization methods considered.

Finally, referring to Figure 6.1c, illustrating what we will call type 3 behavior, we see that δ demonstrates a very well-behaved effect on the objective function, resulting in a very smooth surface, far more amenable to calibration. Being the parameter that best demonstrates this kind of behavior, it is not surprising that it was the only parameter that demonstrated true convergence in the calibration experiments conducted.

It should be noted that the effect of σ^z is far less significant than that of δ, but still appears to result in well-defined effects on the objective function, observable when δ is close to zero. In the calibration results previously presented, σ^z typically performed much better than the other parameters, with the exception of δ, having a much smaller confidence interval, but likely did not show true convergence since the effect of δ on the objective function is extremely dominant, with σ^z only producing a noticeable effect when δ is small, but still larger than values to which δ tends to converge in the calibration experiments.

Calibration experiments involving a fixed δ may result in convincing convergence for σ^z, but such investigations are beyond the scope of this investigation.

A key realization is the fact that the effects of the other parameters on the objective function are small in comparison to the effect of δ. Parameters demon-

strating type 1 behavior, as shown in Figure 6.1a, tend to result in objective function values of no more than 5000 for their worst possible combinations, absolutely insignificant in comparison to δ, resulting in values in excess of 2×10^7 if chosen poorly. Though σ^y produces a significant effect on the objective function, the worst values of δ still produce objective function values more than 10 times that of the worst values of σ^y.

This is somewhat disconcerting, as it points to one of two possible conclusions.

The first, which we tend to lean toward, is that model itself may be flawed in some respects. To substantiate these claims, 7 of the 10 free parameters we considered demonstrated what we referred to as type 1 behavior, producing haphazard and predominantly small effects on the objective function values. In contrast to this, δ and to a lesser extent σ^y and σ^z, demonstrate far more significant effects on the objective function, which seem to have resulted in δ dominating the calibration process, leading it to be the only successfully calibrated parameter. This seems to indicate that 7 of the parameters considered demonstrate some form of parameter degeneracy, resulting in a model that is primarily driven by a single parameter, δ. This is further illustrated by the fact that similar objective function values were obtained for many of the calibration experiments conducted and that these in general resulted in a reasonable fit to the empirical data. This suggests that choosing δ well may be enough to obtain reasonable calibration, at least with regard to the moments considered. We suggest possible reasons for this in Chapter 7.

Alternatively, it may simply be a case that the method of calibration, more specifically the weight matrix, is flawed in some respects, resulting in these observed behaviors. While we do indeed acknowledge that there are flaws in method of Fabretti (2013) and hence our own, the successful use of the methodology to calibrate the Farmer and Joshi (2002) model and the fact that we were able to obtain reasonable fits to empirical data and obtain convergence in the parameter δ seems to negate these concerns. We address these issues in more detail in Chapter 7.

6.2 Preis et al. Model

The results, figures and discussions presented in this section have been reproduced from Platt and Gebbie (2016b), with minor alterations.

6.2.1 Free and Fixed Parameters

In all calibration experiments, we consider 6 free parameters, with only N_A set to be fixed. This particular simplification was made for the purposes of maintaining

computational tractability, since increasing the value of N_A can have a detrimental impact on the speed of simulation. We have chosen $N_A = 250$, as was selected by Preis et al. (2006).

The remaining model parameters, namely δ, λ_0, C_λ, Δ_S, α and μ, defined in Section 2.3, are set to be free parameters and we randomly generate initial values for these parameters during each calibration experiment.

We initialize δ, Δ_S and μ between 0 and 0.1, α between 0.1 and 0.5, λ_0 between 0 and 200, and C_λ between 0 and 20. The initial price, p_0, is determined as in the case of the Jacob Leal et al. (2015) model.

6.2.2 Nelder-Mead Simplex Algorithm

Since we consider $n = 6$ free parameters in our calibration experiments, we begin with $n + 1 = 7$ simplex vertices, each consisting of randomly generated initial values for the 6 free parameters.

In our experiments, it was noted that 100 iterations of the Nelder-Mead simplex algorithm with threshold accepting almost always produced convergent behavior and we thus conducted 20 calibration experiments with this fixed number of iterations.

Table 6.5: Nelder-Mead Simplex Calibration Results (Preis et al. (2006) Model)

Parameter	95% Conf Int	$\frac{s}{\sqrt{n}}$
δ	$[0.0554, 0.0782]$	0.0085
λ_0	$[137.3818, 190.9182]$	11.33934
C_λ	$[11.0808, 21.3192]$	2.6959
Δ_S	$[0.0272, 0.0572]$	0.0086
α	$[0.1545, 0.2517]$	0.0186
μ	$[0.0692, 0.1014]$	0.0102

95% confidence intervals for the set of free parameters, obtained from 20 independent calibration experiments involving the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic applied to the Preis et al. (2006) model

Despite the presence of convergent behavior, we find that the obtained parameter sets are somewhat different for various calibration experiments, with dependence on the set of initial vertices. We also observed this phenomenon in the case of the Jacob Leal et al. (2015) model, but find a greater reduction in the initial search space when considering the parameter confidence intervals obtained for the Preis et al. (2006) model. These confidence intervals are shown in Table 6.5.

This is indicative of convergence to local minima, even with the inclusion of methods aiming to overcome this problem, namely the threshold accepting heuristic. It would therefore appear that there exists significant difficulty in identifying a unique, optimal parameter set.

As was discussed by Fabretti (2013), we also find that these different parameter sets form a region of feasible parameters, all producing similar objective function values, despite differences in the parameter values themselves. In the case of the Jacob Leal et al. (2015) model, we found that such a trend existed, but not as prominently as in this case, where we now find that all the calibration experiments conform to this behavior. Furthermore, it is also worth noting that, as was found in the case of the Jacob Leal et al. (2015) model, that the majority of the obtained parameter sets again produce fits of similar quality to that of Fabretti (2013), which we demonstrate in Section 6.2.4.

6.2.3 Genetic Algorithm

Considering that the results obtained using the genetic algorithm were qualitatively similar to those obtained using the Nelder-Mead simplex algorithm in the case of the Jacob Leal et al. (2015) model, we do not repeat the genetic algorithm experiments for the Preis et al. (2006) model, in an attempt to avoid redundancy.

6.2.4 Comparison of Calibrated Model and Empirical Data

Table 6.6: Best Parameter Set Obtained Through Calibration (Preis et al. (2006) Model)

Calibrated Parameter	Parameter Value
δ	0.0733
λ_0	180
C_λ	33
Δ_S	0.0328
α	0.2129
μ	0.0653

Best parameter set obtained through the application of the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic to the Preis et al. (2006) model

It was mentioned in the preceding sections that while no unique parameter set could be established, all of the experiments converged to different parameter sets

with similar objective function values producing reasonable behavior when compared to the simulated data. In this section, we consider the best parameter set, as measured by the objective function, obtained in the Nelder-Mead simplex experiments.

For this parameter set, we simulate 25 price paths using the Preis et al. (2006) model and obtain the 95% confidence intervals for the estimates of the mean, standard deviation, kurtosis, and Hurst exponent of the simulated log prices and compare this to the empirical moments obtained from the transaction data.

As seen in Table 6.7 and also demonstrated in the case of the Jacob Leal et al. (2015) model, we observe that the parameter sets obtained through the calibration procedure employed are indeed able to produce fits of similar quality to that of Fabretti (2013) for the mean and standard deviation, while slightly overestimating the Hurst exponent and severely overestimating the kurtosis. This is predominantly due to the fact that the weight matrix obtained through the method of Fabretti (2013) tends to assign very large weights to errors on the mean and standard deviation, while assigning very small weights to errors on kurtosis, leading calibration to either underestimate or overestimate kurtosis in our own investigation and that Fabretti (2013).

Table 6.7: Comparison of Empirical and Calibrated Moments (Preis et al. (2006) Model)

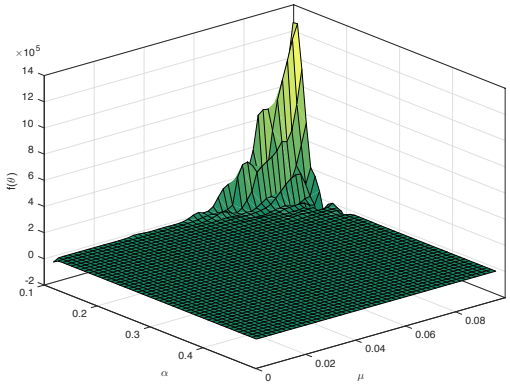
Moment	Simulated Value	Empirical Value
Mean	$[5.4764, 5.5120]$	5.5143
Std Dev	$[0.0296, 0.0394]$	0.0300
Kurtosis	$[1.9829, 2.7979]$	1.2402
Hurst Exponent	$[0.5837, 0.6032]$	0.5658

95% confidence intervals for the simulated moments generated by the calibrated Preis et al. (2006) model compared to the empirically-measured moments

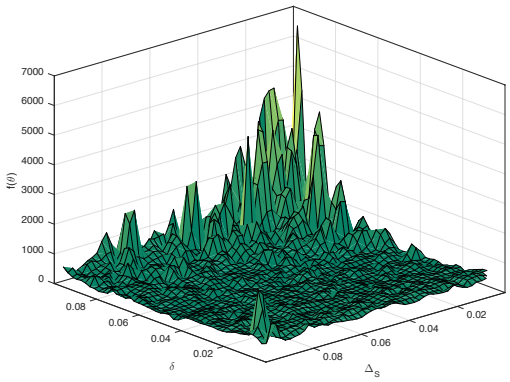
Despite this flaw, the above seems to suggest that members of this class of models, namely ABMs of continuous double auction markets using realistic order matching processes, are indeed capable of reproducing a number of characteristics of empirically-sampled data to some degree and that the calibration methodology of Fabretti (2013) can indeed determine appropriate parameter sets, but that the parameters producing such fits are simply not unique. In the following subsections, we detail possible reasons for this behavior through an investigation of the behavior of the objective function.

6.2.5 Parameter Analysis

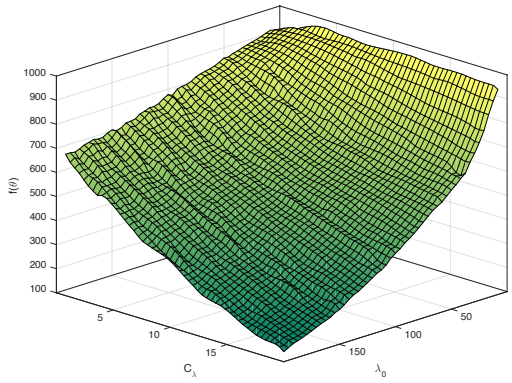
For this section, we employ the process discussed is Section 6.1.5 to construct objective function surfaces for the Preis et al. (2006) model. The base parameter set used for these experiments is presented in Table 6.6.



(a) Type 1 parameter surface illustrating the effect of α and μ on the objective function



(b) Type 1 parameter surface illustrating the effect of δ and Δ_S on the objective function



(c) Type 3 parameter surface illustrating the effect of λ_0 and C_λ on the objective function

Figure 6.2: Objective function surfaces for the Preis et al. (2006) model

As is clearly visible in Figures 6.2a and 6.2b, we see that δ, Δ_S, α and μ all exhibit a similar effect on the objective function. While there are indeed certain regions that produce particularly bad fits to the data, the vast majority of the surfaces are relatively flat, indicating a very wide range of feasible parameters producing a reasonable fit to the data, with no clear way to distinguish an optimal global minimum. This is a similar insight to that expressed by Fabretti (2013).

This is the likely reason why parameter convergence was not observed, despite the fact that the objective function values were similar for each experiment, as a very large area of the parameter space produces very similar dynamics. The effect of these parameters on the dynamics of the obtained log price time series is thus not clearly defined, again indicative of parameter degeneracies in a similar vein to those found for the Jacob Leal et al. (2015) model.

In contrast to this, however, Figure 6.2c indicates that both C_λ and λ_0 produce a very well-defined effect on the objective function. This is an interesting observation. We observed a similar trend in the case of the Jacob Leal et al. (2015) model, with all parameter surfaces behaving in a haphazard manner, except for those dealing with parameters that drove the random walk which defined order prices. It is therefore not surprising that C_λ and λ_0 happen to drive the order placement depth process, $\lambda(t)$, and hence order prices in this model.

It is important to note that in the case of the Jacob Leal et al. (2015) model, a single parameter associated with one of the smooth objective function surfaces previously mentioned did show convergence, though neither λ_0 nor C_λ have demonstrated this behavior in this investigation. This is most likely due to the fact that other parameters in the Jacob Leal et al. (2015) model never produced significant effects on the objective function in comparison to the single successfully calibrated parameter.

In the Preis et al. (2006) model, however, there are regions where the objective function value can become very large for certain values of δ, Δ_S, α and μ, whereas they typically have a very small effect on the objective function. This is because these regions typically represent unstable model dynamics, for example α close to μ and a high order cancelation rate, δ, leading to an illiquid order book that is frequently depleted of orders, which in turn drives unrealistic and unstable simulated price dynamics.

These very high objective function values in these regions may have affected convergence in λ_0 and C_λ, since their effect would not be as dominant as it would be for typical parameter values, this dominance being the reason for the observed convergence in the case of the Jacob Leal et al. (2015) model. Rather than blindly accepting such reasoning, however, we rigorously demonstrate that such convergence is indeed possible in Section 6.2.6.

6.2.6 Verification of λ_0 and C_λ Convergence

In order to verify that λ_0 and C_λ can be uniquely determined when the dynamics of the other parameters do not introduce noise into the calibration procedures, we consider a separate calibration experiment involving only λ_0 and C_λ.

Table 6.8: Genetic Algorithm Calibration Results (Preis et al. (2006) Model)

Parameter	95% Conf Int	$\frac{s}{\sqrt{n}}$
λ_0	$[193.2596, 196.5567]$	0.6972
C_λ	$[18.7165, 19.4306]$	0.1510

95% confidence intervals for λ_0 and C_λ, obtained from 8 independent calibration experiments involving a genetic algorithm

Rather than using the Nelder-Mead simplex algorithm as before, we now consider a genetic algorithm, since 3 initial simplex vertices would be too small a parameter space to randomly initialize.

We begin with a randomly initialized initial population of 100 individuals, using the same parameter bounds as in the Nelder-Mead simplex calibration experiments, and iterate this population for 50 generations, determined to be sufficient for convergence. We then repeat this process a total of 8 times in order to obtain confidence intervals as in Section 6.2.2. All other parameters are set according to the values in Table 6.6.

Referring to Table 6.8, we see that when only λ_0 and C_λ are considered in a calibration experiment, we observe meaningful convergence in both parameters. The identified region of convergence is also consistent with the visible minimum in Figure 6.2c.

Chapter 7

Analysis of Results and Supplementary Experiments

In this chapter, we analyze the behaviors observed when attempting to calibrate the Jacob Leal et al. (2015) and Preis et al. (2006) models. This chapter is based on discussions presented in both Platt and Gebbie (2016a) and Platt and Gebbie (2016b).

7.1 Discussion of Calibration Results

As was shown in Section 6.1, the parameters α^c, σ^c, α^f, σ^f, λ, N_L and N_H of the Jacob Leal et al. (2015) model demonstrate evidence of parameter degeneracies and have a haphazard effect on the objective function, while σ^y and σ^z present evidence of an observable objective function trend and δ demonstrates a clear relationship between its values and the objective function value. We originally conjectured that this dominance of δ on the model dynamics was likely a direct result of the model's construction and possibly an isolated phenomenon:

Firstly, referring to Eqn. (2.10), we see that LF trader order prices for a particular session are set according to a single iteration of a geometric random walk referencing the market price of the previous session. Not surprisingly, δ is an important parameter in determining the nature of the aforementioned geometric random walk and hence δ plays an important role in the setting of LF trader order prices in the model. Furthermore, HF trader orders expire at the end of each session ($\gamma^H = 1$) and HF trader order prices are set to be slight perturbations of existing LF trader orders in the order book, hence we see that δ would have a significant impact on HF trader order prices in each session as well.

Secondly, referring to Eqn. (2.9), we see that both δ and σ^y, another parameter shown to demonstrate structure in the calibration experiments, drive the

equation which determines the fundamental value. Because order sizes under both the chartist and fundamentalist strategies are predominantly determined by the market price and fundamental value time series, both of which are driven by or heavily influenced by geometric random walks including δ as a parameter, we see that δ has a significant effect on the Jacob Leal et al. (2015) model's order size and order price dynamics. This is in contrast to the remaining parameters, which typically only influence one aspect of the model. This leads to a fairly significant coupling of dynamics, making δ far more important than the remaining model parameters, which we thought might lead it to dominate the calibration experiments as it has. Chakraborti et al. (2011) stated that it is often difficult to identify the role played by and dependencies between the various parameters present in ABMs, despite their ability to reproduce well-known stylized facts, which we have observed in the case of the Jacob Leal et al. (2015) model.

σ^y, σ^z and δ, the only parameters demonstrating any structured effect on the objective function in the calibration experiments, appear in two geometric random walks included in the model's equations. It was originally thought that it may be possible that these random walk dynamics dominate the model, with the parameters not considered in these random walks degenerating to noise, since it was previously shown in Section 6.1.5 that the effects of σ^y, σ^z and δ on the objective function are far more significant than those of the remaining parameters.

This ultimately led to the application of similar calibration experiments to the Preis et al. (2006) model to determine the validity of this conjecture, since the Preis et al. (2006) model replaced the aforementioned random walks with a more realistic order price determination mechanism rooted in the current state of the order book. In general, this type of order placement mechanism is more desirable, since the use of a random walk by Jacob Leal et al. (2015) is unintuitive and results in orders being placed without any consideration of the dynamics of the order book. Considering the importance of price impact in contemporary literature (Lehalle (2013); Cartea et al. (2015)), order placement behaviors more directly rooted in the dynamics of the order book are more theoretically sound in the context of a double auction market.

Referring to Section 6.2, we see that despite differing design philosophes and order placement mechanisms, most of the parameters of the Preis et al. (2006) and Jacob Leal et al. (2015) models demonstrate degeneracies, with the exception of the parameters that drive the order price determination mechanisms in both models. It is therefore far more likely that such degeneracies find their origin in the more realistic matching processes used to determine market prices in intraday ABMs of double auction markets. These mechanisms are in contrast to the approximating equations used in daily models, which have been more successfully calibrated by Fabretti (2013) and Gilli and Winker (2003). Perhaps the indirect calculation of

prices through order matching does not lead to the same well-defined dynamics as an approximating equation that considers the effects of all parameters with simple assumptions regarding their relationship?

In both the Jacob Leal et al. (2015) and Preis et al. (2006) models, it would therefore appear that the prices produced through realistic order matching procedures are only affected in a direct and structured way by the processes used to determine the prices of limit orders stored in the order book, with other processes tending to produce poorly-behaved or insignificant effects. We present possible reasons for this behavior in Section 7.2, where we analyze the dynamics of realistic matching procedures in more detail.

7.2 Realistic Order Matching Procedures and Price Dynamics

In the case of both the Jacob Leal et al. (2015) and Preis et al. (2006) models, it is apparent that certain model features rooted in market microstructure, such as order book depth and the associated dynamics of the prices of limit orders, can be reliably determined by the calibration of intraday ABMs to transaction data. In contrast to this, however, it seems that the parameters of these models associated with the frequency at which traders submit orders, the specific number of trader agents participating in a particular market and various other phenomena that are rooted in agent behavior tend to be difficult to infer with significant confidence.

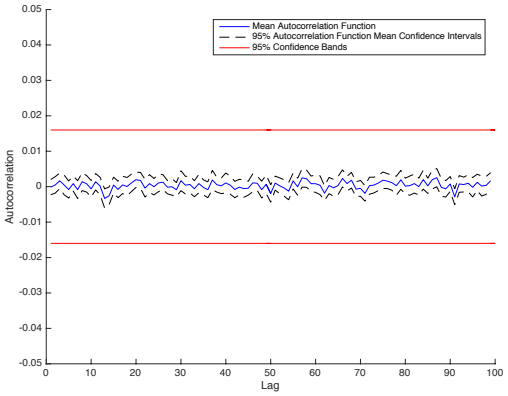
This naturally leads to a discussion on the effect of various processes within an intraday ABM on the simulated price time series when this time series is generated using realistic order matching processes. Bouchaud et al. (2008) argued that prices in real markets are driven by two main phenomena, the response of prices to individual orders and the flow of orders arriving in the market. We therefore investigate how this may apply to a market simulated using an intraday ABM through supplementary experiments involving the Preis et al. (2006) model.

It is evident that the processes which determine order prices within an intraday ABM directly influence the first of these phenomena in a well-defined way. These processes directly determine the prices of limit orders within the LOB and these order prices define the mid prices or trade prices (depending on the model in question) that will eventually be set as market prices. This likely explains the fairly well-defined effect of the parameters influencing order prices on both the simulated price time series and objective function in the case of the Jacob Leal et al. (2015) and Preis et al. (2006) models.

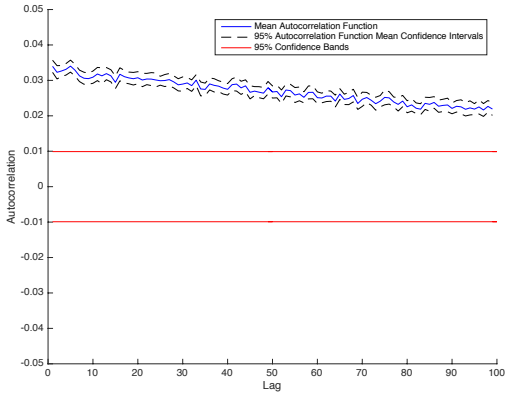
The second mechanism, namely the order flow, requires more careful consideration. A key question to consider is whether the Preis et al. (2006) model does in

fact produce a realistic, nontrivial order flow.

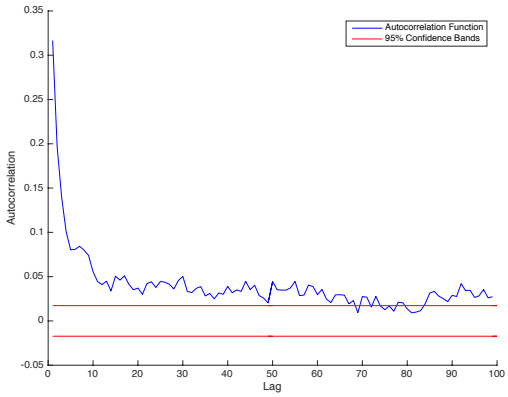
A well-known property of order flow in real financial markets is long memory (Bouchaud et al. (2008)). This phenomenon can be observed by plotting the autocorrelation function of the time series of trade signs for a particular market over some period (+1 for buyer initiated trades and -1 for seller initiated trades), which typically demonstrates a slow decay in autocorrelation as the number of lags is increased in most empirical investigations (Kuroda et al. (2011)).



(a) Preis et al. (2006) model trade signs for the set of default parameters



(b) Preis et al. (2006) model trade signs for the set of calibrated parameters



(c) Measured trade signs

Figure 7.1: Autocorrelation functions for the empirical and simulated trade signs, with simulated trade sign autocorrelation functions averaged over 50 Monte Carlo replications

Using the measured data and following the method of Lee and Ready (1991), which was also applied to similar datasets in Harvey et al. (2017), we classified trades as either buyer or seller initiated and computed the associated trade sign autocorrelation function. This autocorrelation function is presented in Figure 7.1c

and confirms that long memory dynamics are captured in the order flow associated with the data to which the model was calibrated.

Figure 7.1a shows the autocorrelation function for trade signs generated by the Preis et al. (2006) model, initialized with the default parameters described by Preis et al. (2006), which we present in Table 5.2. We see that the characteristic slow decay in autocorrelation associated with long memory is absent, indicative that such dynamics are not captured by the model when initialized with the default parameters of Preis et al. (2006).

Referring to Figure 7.1b, where we repeat this exercise using the calibrated parameters presented in Table 6.6, we see that the slow decay in autocorrelation associated with long memory processes is now present, indicating that the calibration process has determined a parameter combination that results in a more realistic order flow than the model's default parameters.

On closer inspection, it was found that shifting the values of individual parameters in Table 6.6, with the exception of Δ_S, and repeating this exercise did not cause these more realistic long memory dynamics to disappear. When shifting Δ_S, however, it was found that smaller values of this parameter resulted in long memory dynamics no longer being present in the order flow.

This is in fact consistent with the surface presented in Figure 6.2b:

For the δ value presented in Table 6.6, approximately 0.07, we see that smaller Δ_S values tend to result in poorer objective function values, whereas values of Δ_S roughly larger than 0.03 tend to result in fairly similar and relatively low objective function values. Not surprisingly, this region of lower objective function values in fact corresponds almost exactly to those in which long memory was observed in the simulated order flow.

Therefore, Δ_S does indeed exert some effect on the order flow within the model, but a very wide range of possible values exists for Δ_S that produces long memory dynamics. This is reflected in the effect of Δ_S on the obtained price time series, which is itself reflected in the effect of Δ_S on the objective function, leading to a large region where no unique minimum can be determined. While the simulated order flow obtained through calibration is by no means identical to that observed in the data, likely due to model limitations, calibration has resulted in an attempt to capture these dynamics.

In contrast to C_λ, λ_0 and Δ_S, parameters α, μ and δ relate to agent behaviors and thus do not directly affect the individual orders or order flow within the model. The fact that altering these parameters did not affect the presence of long memory in the simulated order flow provides evidence of the second aspect of the preceding statement. This implies that these parameters do not directly relate to the two processes conjectured by Bouchaud et al. (2008) to be the driving forces behind prices in real markets and this seems to be mirrored in the realistic matching

processes employed in intraday ABMs, where we observed that the effect of these parameters on the simulated price time series was not well-defined.

We therefore conjecture that the encountered parameter degeneracies emerge from the application of realistic matching processes within intraday ABMs. Parameters which exert well-defined effects on either order flow or individual orders themselves produce well-defined effects on the obtained simulated price time series and are therefore amenable to calibration. These parameters tend to be rooted in the microstructure of the market and the nature of the LOB. In contrast to this, parameters which relate more to agent behavior and do not tend to have a well-defined or direct effect on individual order dynamics or order flow tend to produce poorly-defined effects on the obtained price time series, resulting in calibration difficulties.

This may explain why the calibration of closed-form, daily ABMs proved more successful in past experiments, such as those of Fabretti (2013). In using a closed-form solution to determine market prices, one directly enforces a well-defined, albeit simplified, relationship between parameter values and the obtained market price time series. In contrast to this, the realistic matching processes in intraday models seem to be instead driven by order flow and the nature of individual orders, leading parameters which do not directly determine these dynamics in a well-defined way to produce haphazard dynamics.

The recovery of trade sign autocorrelation through model calibration leads us to a discussion of the latent order book.

As suggested by Bonart (2015), the LOB itself only represents a fraction of the actual liquidity in a particular market. In real financial markets, liquidity providers attempt to hide their intentions and thus tend to avoid the placement of very large limit orders. This implies that liquidity takers attempting to acquire or liquidate large numbers of shares cannot do so through the placement of a single order and thus break a single parent order into a number of child orders, resulting in the observed trade sign autocorrelation. The true intentions of liquidity providers and the true nature of supply and demand are therefore not captured in the order book itself, but rather in a fictitious order book known as the latent order book. The purpose of trading that drives the latent liquidity can operate on very different, and possibly much longer, time scales than those inherent in the short term dynamics of the order book (Bouchaud et al. (2008)).

It is clear that the Preis et al. (2006) model, though not explicitly considering the latent order book, has been able to reproduce some of the features of financial markers thought to originate from it, suggesting that there may be some hidden explanatory potential embedded within ABMs of continuous double auction markets with regard to the underlying microstructure of the market. The ability to uniquely determine the parameters associated with limit order price genera-

tion processes in both the Jacob Leal et al. (2015) and Preis et al. (2006) models provides further evidence of this possibility.

While similar experiments could be applied to the Jacob Leal et al. (2015) model, the model is not as strongly grounded in market microstructure and was therefore not investigated in this way.

7.3 Consequences of this Investigation

It is evident through the calibration experiments conducted that there is at least some evidence that intraday ABMs of continuous double auction markets involving realistic order matching suffer from a degree of parameter degeneracy and present a number of obstacles that render them difficult to calibrate definitively. This, in essence, presents a number of challenging questions. How can we be sure that the parameter N_H truly influences the nature of HF trading in a simulated market rather than introducing noise into a model that is predominantly driven by order price dynamics? Furthermore, how can we trust the conclusions drawn by Jacob Leal et al. (2015) regarding the parameter N_H and the prevalence of flash crashes, when N_H seems to be a degenerate parameter?

Being unable to obtain unique calibrations for a particular model rigorously is in fact a more serious issue than it is presented as in much of the literature. Without a unique set of parameters allowing us to reproduce the properties of financial time series drawn from a particular market, how can we argue that introducing and observing changes in the model truly reflects the same changes in the market?

Regardless of one's stance on whether the behavior observed in this investigation is isolated, or indicative of general trends in rules-based ABMs involving realistic order-matching processes at an intraday time scale, we have demonstrated a fairly significant concern. While Chapter 5 demonstrates the ability of our implementations of both models to replicate a number of established log return time series stylized facts, this did not account for the fact that they demonstrated evidence of parameter degeneracies and appeared to be driven predominantly by the parameters determining order prices, with much of the remaining processes tending towards noise. This is testament to a problematic oversight that permeates the breadth of much of the contemporary literature on the subject: the sufficiency of stylized facts as a form of validation. The use of stylized fact-centric validation is so pervasive that Panayi et al. (2013) suggested it is more or less standard practice in financial agent-based modeling. While a model must replicate stylized facts to be validated, the replication of stylized facts can only be taken as a first step and successful calibration to transaction data must be achieved to truly validate a model.

This therefore suggests that the current paradigm widely employed in the de-

velopment of intraday ABMs approximating continuous double auction markets, in which the mechanistic complexity of existing models or model components is increased such that well-known stylized facts are replicated, may lead to flawed insights due to parameter degeneracies not detected by stylized-fact centric validation.

In this context, one should consider the work of Bookstaber et al. (2016), which provided a number of significant mechanistic extensions to the Preis et al. (2006) model, most notably the inclusion of heterogeneous decision cycles and new agent types, such as large firms. As with much of the existing ABM literature, stylized fact-centric validation procedures are employed exclusively, with the authors explicitly stating that such validation procedures are fairly rigorous. Our results indicate that this may in fact not be the case and that there is at least cause for some skepticism as to whether certain of the extensions to the model result in parameter degeneracies not detected by stylized fact-centric validation.

When considering our preceding analysis, however, the following must be taken into account:

Firstly, it is important to remember that we have attempted to calibrate the models to a single week of data for a particular stock traded on a particular market. Therefore, the above are not definitive conclusions, but rather discussions related to potential calibration challenges identified by this particular investigation that require further exploration. In future, it would certainly be worth applying similar methodologies to other intraday datasets to verify whether this behavior is common or isolated. It may very well be that other datasets yield more well-defined behaviors with respect to important parameters, such as N_H in the Jacob Leal et al. (2015) model.

Secondly, it is entirely possible, but in our opinion unlikely, that the aforementioned concerns result from the calibration methodology and not the models themselves. Firstly, it must be noted that the weight matrix, shown in Section 3.2.1, was obtained by inverting a matrix that is non-singular, but only just. Fabretti (2013) also performed the aforementioned inversion of a near-singular matrix, though we inverted a less ill-conditioned matrix. This could of course be used to explain the haphazard behaviors demonstrated by some of the parameters, but if and only if this was a general trend. As we have shown, this same weight matrix results in fairly well-defined behaviors with regard to the parameter δ in the Jacob Leal et al. (2015) model and the parameters λ_0 and C_λ in the Preis et al. (2006) model, making this argument more difficult to justify. Furthermore, it should be noted that the calibration experiments did not fail as such. While most of the parameters could not be uniquely identified, the three aforementioned parameters showed evidence of convergence and the obtained fits of both models to the empirical data, shown in Sections 6.1.4 and 6.2.4, were in fact relatively good, more

or less in line with that of Fabretti (2013), who was considered to have been successful. It is also worth noting that we have not deviated from the methodology of Fabretti (2013), regarded as a valid calibration approach.

Of course, it would be worthwhile to consider applying alterative calibration methods to the same selection of models and datasets in future work to observe if similar behaviors emerge.

Finally, we must also address the potential criticism that the number of parameters considered in the calibration experiments exceeds the number of moments used to construct the objective function. If, in the continuous time limit and under the assumptions of ergodicity and stationarity, the number of moments in the objective function is at least the same as the number of parameters in model to be calibrated, then there can be perfect model parameter identification and the parameter values found are the solution that minimizes the objective function (Canova and Sala (2009)). This is not in general possible, in part due to the large number of parameters characteristic of the class of models considered. We hence demonstrate the robustness of our results by fixing a number of the parameters to reasonable values and repeating the Jacob Leal et al. (2015) model experiments with a free parameter set consisting of fewer free parameters than objective function moments, which we present in Appendix B.

Despite a number of shortcomings, ABMs still have many important contributions to offer. In particular, the effect of HF trader activity on the stability and dynamics of financial markets is still being debated (Easley et al. (2012)) and ABMs may hold the answers to many unanswered questions. Other modeling approaches, such as those employed by Gerig (2012), have been used to show the price synchronizing features of HF trading, as well as features leading to improvements in market efficiency during normal periods and negative consequences during times of stress. ABMs are well placed to contribute similarly to this debate.

In future work, it may very well arise that rules-based, intraday ABMs involving realistic matching processes may all demonstrate similar flaws when subjected to rigorous calibration. For this reason, it would be necessary to mention possible alternatives that may represent a step up from overly-simplified daily closed-form models.

One such alternative is to revisit the mathematics of trader rules and consider the replacement of the disjointed order price, order size and trader activation rules with a single structure, namely a Hawkes process, to generate trade events. In recent years, much progress has been made in using Hawkes processes to generate realistic occurrences of order book events (Bacry et al. (2015); Martins and Hendricks (2016)). Hawkes processes may also allow the model to function more appropriately in an event-based paradigm (Hendricks et al. (2016a)), possibly negating a need to down-sample event-based tick data into a chronological series, which

may have resulted in the problems previously observed. An example of the incorporation of Hawkes process into agent-based order book simulation was presented by Toke (2011).

Alternatively, it may advantageous to dispense with rules entirely, focusing instead on large scale simulations involving many purposeful, reinforcement-learning agents with no initially assumed rules (Hendricks and Wilcox (2014); Hendricks (2016); Wilcox and Gebbie (2014)) and focus on competing, adaptive agents in a game theoretic framework (Galla and Farmer (2013); Lachapelle et al. (2015)). This may require a more effective adoption of a large-scale multi-agent system (MAS) approach for the modeling of ABM systems for market simulation applications, which would require appropriately reactive software platforms, such as Akka (Wampler (2015)). This in turn would create new challenges for model validation.

Chapter 8

Conclusion

The preceding investigation has revealed a number of important considerations that relate to current approaches to financial agent-based modeling and the validation of the associated models.

We have demonstrated that both of the models considered, namely the Jacob Leal et al. (2015) and Preis et al. (2006) models, are able to replicate a number of well-known stylized facts of return time series and as shown in Sections 6.1.4 and 6.2.4, the calibration procedures employed are indeed capable of determining parameter sets that allow both models to produce log price time series with statistical properties that are fairly similar to those associated with a log price time series implied from transaction data.

This is not unexpected, since stylized fact-centric validation, at its core, demonstrates that the overall properties of the simulated price time series are at least qualitatively similar to those of a real price time series, meaning that one would expect that there exists at least some number of parameter sets that are able to reproduce the properties of a desired price time series in the case of both models.

This is not, however, a completely satisfactory form of validation, as we have demonstrated in our investigation. This is because stylized fact-centric validation does not guarantee that the parameters associated with a particular model are relevant or behave as expected and may fail to identify situations in which a number of the model parameters produce noisy or insignificant effects on the obtained price time series. In our investigation, this phenomenon manifests itself as the dominance of model dynamics by parameters determining order prices, with the remaining parameters seemingly becoming degenerate in both models.

This is a fairly significant concern, since investigations involving financial ABMs often involve the manual tuning of parameter values and the observation of resultant changes in price and other model behaviors. This implies that certain parameters in models validated using stylized fact-centric procedures exclusively may suffer from hidden parameter degeneracies and that the varying of the parameters

in question may result in little more than introducing noise into a model that is instead driven by another process. This can ultimately lead to flawed insights.

Despite this, stylized fact-centric validation procedures retain a degree of relevance. The replication of stylized facts remains a necessary condition that reveals whether the overall dynamics of the model are realistic, but cannot verify that each parameter behaves in a well-defined and meaningful fashion. For this reason, we argue that the augmentation of existing stylized fact-centric validation with attempts at calibration is essential in future work and thus propose that calibration should become more central to the validation process. In addition, we argue that the existing paradigm of continually increasing the mechanistic complexity of intraday ABMs while employing stylized fact-centric validation exclusively may ultimately result in compromised conclusions.

Accidental agreements between models and stylized facts need to be guarded against. The pragmatic calibration approach of Fabretti (2013) provides an additional probe of model structure in the context of continuous double-auction markets, one that may further help balance the tenants of verisimilitude and model corroboration with the conceptual benefits of ABMs.

Our investigation has further revealed that intraday ABMs of double auction markets employing realistic matching procedures are primarily driven by the parameters and processes determining the prices of individual limit orders within them, with the remaining parameters relating to behavioral processes tending to become degenerate. We conjecture that this is likely due to the fact that like in real financial markets, simulated market prices in intraday ABMs employing realistic matching processes are driven by responses to individual orders and the nature of order flow and therefore only parameters associated with processes which affect these phenomena in well-defined ways produce well-defined and significant effects on the market price time series.

This, along with the fact that the calibration of the Preis et al. (2006) model resulted in order flow correlation not recovered by the model's default parameters, suggests that increased emphasis may have to be placed on market microstructure, in particular order book structure and latent liquidity, since these have seemed to emerge as dominant features in our investigation.

Bibliography

Alfarano S, Lux T, Wagner F (2005) Estimation of agent-based models: the case of an asymmetric herding model. Comput Econ 26(1):19-49

Alfarano S, Lux T, Wagner F (2006) Estimation of a simple agent-based model of financial markets: an application to Australian stock and foreign exchange data. Physica A 370(1):38-42

Alfarano S, Lux T, Wagner F (2007) Empirical validation of stochastic models of interacting agents. Eur Phys J B 55(2):183-187

Amilon H (2008) Estimation of an adaptive stock market model with heterogeneous agents. J Empir Financ 15(2):342-362

Bacry E, Mastromatteo I, Muzy JF (2015) Hawkes processes in finance. arXiv:1502.04592

Barde S (2016) Direct calibration and comparison of agent-based herding models of financial markets. J Econ Dyn Control 73:329-353

Beltratti A, Margarita S (1992) Evolution of trading strategies among heterogeneous artificial economic agents. In: Meyer JA, Roitblat HL, Wilson SW (eds) From Animals to Animats 2. MIT Press, Boston

Bonart J (2015) Mathematical aspects of delayed market clearing in order driven markets and its applications to non-Markovian price impact and optimal execution. SSRN:2659092

Bookstaber R, Foley MD, Tivnan BF (2016) Toward an understanding of market resilience: market liquidity and heterogeneity in the investor decision cycle. J Econ Interact Coord 11:205-227

Bouchaud JP, Farmer JD, Lillo F (2008) How markets slowly digest changes in supply and demand. arXiv:0809.0822.

Bouchaud JP, Mezard M, Potters M (2002) Statistical properties of stock order books: empirical results and models. Quant Financ 2(4):251-256

Brock WA, Hommes CH (1998) Heterogeneous beliefs and routes to chaos in a simple asset pricing model. J Econ Dyn Control 22(8-9):1235-1274

Canova F, Sala L (2009) Back to square one: Identification issues in DSGE models. J Monetary Econ 56:431-449

Cartea A, Jaimungal S, Penalva J (2015) Algorithmic and High-Frequency Trading. Cambridge University Press, Cambridge

Chakraborti A, Toke IM, Patriarca M, Abergel F (2011) Econophysics review: II. agent-based models. Quant Financ 11(7):1013-1041

Chiarella C, Iori G (2002) A simulation analysis of the microstructure of double auction markets. Quant Finan 2:346-353

Chiarella C, Iori G, Perello J (2009) The impact of heterogeneous trading rules on the limit order book and order flows. J Econ Dyn Control 33:525-537

Cont R (2001) Empirical properties of asset returns: stylized facts and statistical issues. Quant Financ 1:223-236

Cont R, Potters M, Bouchaud JP (1997) Scaling in stock market data: stable laws and beyond. arXiv:9705087

Cooley T (1997) Calibrated models. Oxf Rev Econ Policy 13(3):55-69

Di Matteo T (2007) Multi-scaling in finance. Quant Fin 7:21-36

Easley D, Lopez de Prado M, O'Hara M (2012) The Volume Clock: Insights into the High-Frequency Paradigm. J Portfolio Manage 39:19-29

Fabretti A (2013) On the problem of calibrating an agent-based model for financial markets. J Econ Interact Coord 8:277-293

Farmer JD, Joshi S (2002) The price dynamics of common trading strategies. J Econ Behav Organ 49:149-171

Galla T, Farmer JD (2013) Complex dynamics in learning complicated games. Proc Natl Acad Sci USA 110:1232-1236

Gao F, Han L (2010) Implementing the Nelder-Mead simplex algorithm with adaptive parameters. Comput Optim Appl 51:259-277

Gerig A (2012) High-Frequency Trading Synchronizes Prices in Financial Markets. arXiv:1211.1919

Gilli M, Winker P (2003) A global optimization heuristic for estimating agent based models. Comput Stat Data Anal 42:299-312

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman, Boston

Gould MD, Porter MA, Williams S, McDonald M, Fenn DJ, Howison SD (2013) Limit order books. Quant Financ 13(11):1709-1742

Grazzini J (2012) Analysis of the Emergent Properties: Stationarity and Ergodicity. J Artif Soc Soc Simulat 15(2):7

Grazzini J, Richiardi M (2015) Estimation of ergodic agent-based models by simulated minimum distance. J Econ Dyn Control 51:148-165

Grazzini J, Richiardi M, Tsionas M (2015) Bayesian estimation of agent-based models. Nuffield College, Economic Working Papers Series, 2015-W12

Guerini M, Moneta A (2016) A Method for Agent-Based Models Validation. Working Paper Series, Institute for New Economic Thinking, 42

Hamill L, Gilbert N (2016) Agent-Based Modelling in Economics. John Wiley & Sons, Chichester

Harvey M, Hendricks D, Gebbie T, Wilcox D (2017) Deviations in expected price impact for small transaction volumes under fee restructuring. Physica A 471:416-426

Hassan R, Cohanim B, de Weck O (2005) A comparison of particle swarm optimization and the genetic algorithm. Proceedings of the AIAA/ASME/ASCE/AHS/ASC 46th Structures, Structural Dynamics and Materials Conference, Austin, Texas, April 2005

Hendricks D (2016) Using real-time cluster configurations of streaming asynchronous features as online state descriptors in financial markets. arXiv:1603.06805

Hendricks D, Gebbie T, Wilcox D (2016a) Detecting intraday financial market states using temporal clustering. Quant Financ 16(11):1657-1678

Hendricks D, Gebbie T, Wilcox D (2016b) High-speed detection of emergent market clustering via an unsupervised parallel genetic algorithm. S Afr J Sci 122(1/2)

Hendricks D, Wilcox D (2014) A reinforcement learning extension to the Almgren-Chriss model for optimal trade execution. Computational Intelligence for Financial Engineering & Economics, 2014 IEEE Conference, pp 457-464

Heylighen F (2008) Complexity and Self-organization. In: Bates MJ, Maack MN (eds) Encyclopedia of Library and Information Sciences. CRC Press, Boca Raton

Holland JH (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor

Jacob Leal S, Napoletano M, Roventini A, Fagiolo G (2015) Rock Around the Clock: An Agent-Based Model of Low- and High-frequency Trading. J Evol Econ 25:1-25

Kirilenko A, Kyle A, Samadi M and Tuzun T (2011) The flash crash: The impact of high frequency trading on an electronic market. SSRN:1686004

Kirkpatrick S, Gelett CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:621-630

Kirman A (1991) Epidemics of opinion and speculative bubbles in financial markets. In: Taylor M (ed) Money and Financial Markets. Blackwell, Oxford, pp 354-368

Kukacka J, Barunik J (2016) Estimation of Financial Agent-Based Models with Simulated Maximum Likelihood. SSRN:2783663

Kunsch HR (1989) The jackknife and the bootstrap for general stationary observations. Ann Stat 17:1217-1241

Kuroda K, Maskawa J, Murai J (2011) Stock price process and long memory in trade signs. Adv Math Econ 14:69-92

Lachapelle A, Lasry JM, Lehalle CA, Lions PL (2015) Efficiency of the Price Formation Process in Presence of High Frequency Participants: a Mean Field Game Analysis. arXiv:1305.6323

Lamperti F (2015) An Information Theoretic Criterion for Empirical Validation of Time Series Models. LEM Papers Series, Laboratory of Economics and Management, Sant'Anna School of Advanced Studies, 2-2015

Lamperti F (2016) Empirical Validation of Simulated Models through the GSL-div: an Illustrative Application. LEM Papers Series, Laboratory of Economics and Management, Sant'Anna School of Advanced Studies, 18-2016

LeBaron B (2005) Agent-based Computational Finance. In: Judd KL, Tesfatsion L (eds) The Handbook of Computational Economics, Vol. 2. Elsevier, Amsterdam, pp 1187-1233

Lee CMC, Ready MJ (1991) Inferring Trade Direction from Intraday Data. J Finance 46(2):733-746

Lehalle CA (2013) Market Microstructure Knowledge Needed for Controlling an Intra-Day Trading Process. arXiv:1302.4592

Macal CM, North MJ (2010) Tutorial on agent-based modelling and simulation. J Simulat 4:151-162

Mandes A (2014) Order placement in a continuous double auction agent based model. Joint Discussion Paper Series in Economics, Department of Business Administration and Economics, University of Marburg, 43-2014

Mandes A (2015) Impact of inventory-based electronic liquidity providers with a high frequency event and agent-based modeling framework. Joint Discussion Paper Series in Economics, Department of Business Administration and Economics, University of Marburg, 15-2015

Martins R, Hendricks D (2016) The statistical significance of multivariate Hawkes processes fitted to limit order book data. arXiv:1604.01824

Nair P (2014) Agent based modelling of a single-stock market on the JSE. Dissertation, University of the Witwatersrand

Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7:308-313

O'Hara M (1995) Market microstructure theory. Blackwell Publishing, New Jersey

Panayi E, Harman M, Wetherilt A (2013) Agent-based modelling of stock markets using existing order book data. In: Giardini F, Amblard F (eds) Multi-Agent-Based Simulation XIII. Springer-Verlag, Berlin, pp 101-114

Platt DF, Gebbie TJ (2016a) The Problem of Calibrating a Simple Agent-Based Model of High-Frequency Trading. arXiv:1606.01495

Platt DF, Gebbie TJ (2016b) Can Agent-Based Models Probe Market Microstructure? arXiv:1611.08510

Preis T, Golke S, Paul W, Schneider (2006) Multi-agent-based Order Book Model of financial markets. Europhys Lett 75(3):510-516

Preis T, Golke S, Paul W, Schneider (2007) Statistical analysis of financial returns for a multiagent order book model of asset trading. Phys Rev E 76:016108

Recchioni MC, Tedeschi G, Gallegati M (2015) A calibration procedure for analyzing stock price dynamics in an agent-based framework. J Econ Dyn Control 60:1-25

Thomson Reuters (2016) Thomson Reuters Tick History. https://tickhistory.thomsonreuters.com. Accessed 23 May 2016

Toke IM (2011) Market Making in an Order Book Model and Its Impact on the Spread. In: Abergel F, Chakrabarti BK, Chakraborti A, Mitra M (eds) Econophysics of Order-driven Markets. Springer, Milan, pp 49-64

Topping CJ, Dalkvist T, Forbes VE, Grimm V, Sibly RM (2009) The Potential for the Use of Agent-Based Models in Ecotoxicology. In: Devillers J (ed) Ecotoxicology Modeling. Springer, Berlin, pp 205-235

Tukey JW (1977) Exploratory Data Analysis. Addison-Wesley, Boston, pp 39-49

Vuorenmaa TA, Wang L (2013) An agent-based model of the flash crash of may 6, 2010, with policy implications. SSRN:2336772

Wampler D (2015) Fast Data: Big Data Evolved. https://info.lightbend.com/rs/558-NCX-702/images/COLL-white-paper-fast-data-big-data-evolved.pdf. Accessed 4 June 2016

Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65-85

Wilcox D, Gebbie T (2008) Serial correlation, periodicity and scaling of eigenmodes in an emerging market. Int J Theoretical Appl Finance 11(7):739-760

Wilcox D, Gebbie T (2014) Hierarchical causality in financial economics. SSRN:2544327

Winker P, Gilli M, Jeleskovic V (2007) An Objective Function for Simulation Based Inference on Exchange Rate Data. SSRN:964131

Appendix A

Detailed Implementation Description

As is to be expected with a project of this nature, a fairly large amount of code must be written in order to implement the various ABMs considered and calibration methodologies employed. This chapter therefore aims to present examples of how certain aspects of the project were implemented by describing the overall structure of the project's associated code and providing code extracts where appropriate.

Considering the fairly large scale of the project, it would be impractical to describe all aspects in detail and provide all code written for it. Therefore, this chapter aims to be illustrative rather than exhaustive.

A.1 Project Directory Structure and Contents

In Figure A.1, we present an overview of the directory structure containing the various classes, functions and scripts which comprise the project's associated code. We have omitted folders related to data and result storage for the sake of brevity and clarity of explanation.

The contents of each directory is elaborated upon as follows:

- *Classes*: Classes for ABM implementation, objective function implementation, and Nelder-Mead simplex algorithm implementation
- *Functions*: Functions for GA evaluation, data preprocessing, and Lee-Ready trade sign classification
- *Scripts*: Scripts to perform calibration experiments, generate objective function surfaces, summarize calibration results, and test for return time series stylized facts

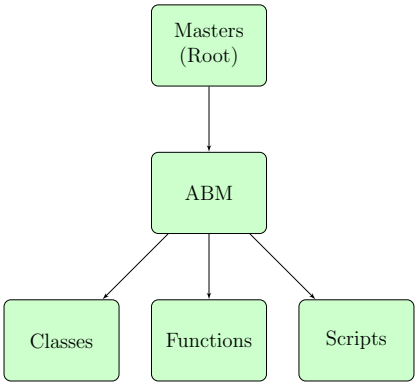


Figure A.1: Code Directory Structure Diagram

A.2 Discussion of Implemented Classes

In this section, we provide a brief overview of the purpose of each class and provide illustrative code extracts where appropriate.

A.2.1 JLParams

The Jacob Leal et al. (2015) model has over 20 parameters. It was therefore decided to store the values of a number of these parameters in a single object that could be passed to the class constructor of a *JacobLealModel* object as opposed to a long string of parameters to allow for a more elegant implementation. We therefore created the *JLParams* class for this purpose.

A.2.2 JacobLealModel

The *JacobLealModel* class encapsulates all the methods and attributes required to implement a complete Jacob Leal et al. (2015) model simulation, with a single *JacobLealModel* object representing a single simulation initialized with a particular parameter set. This includes processes such as the implementation of order placement behaviors by agents, agent strategy adjustments and order matching.

While the majority of the class's methods evaluate the model's equations and store data as needed, the class's *matchEngine* method, which matches all executable limit orders at the end of each session, is more nuanced. We present the full MATLAB code for this method below:

```
function simRun = matchEngine(simRun, t)
```

```
% matchEngine: Match orders and execute all possible trades
% this session, with matching done according to price and time
% priority.
%
% simRun = matchEngine(simRun, t) takes in a JacobLealModel object,
% simRun, and session number, t, and proceeds to match all
% limit orders in the order book which can be executed this
% session. Orders are matched first by price and then by time
% and all executed orders are then removed from the limit
% order book. Following this, the closingPrice attribute is
% modified, where the closing price for the current session (t)
% is set to be the final trade price for the current session.
% If, however, there are no orders of a certain type (buy/sell)
% and no matching can take place, the method exits. The modified
% JacobLealModel object is then returned.
%
% Basic Algorithm:
%
% The starting point is to sort all orders in the orderBook matrix
% according to column 3, which is the order type. This results
% in sell orders (1) appearing together, followed by buy orders
% (2). We then sort buy and sell orders in descending order by price,
% separately. This places the order book in a format where the best bid
% and ask are adjacent and the item before the best ask is the second
% best ask and the item after the best bid is the second best bid and so on.
% Now that the order book is in the correct format, we have the following
% steps:
%
% 1. Find the index of the best bid, call it bidIndex.
%
% 2. Set askIndex to bidIndex — 1, the index of the best ask.
%
% 3. Set the current bestBidPrice and bestAskPrice to the prices at
% indices bidIndex and askIndex respectively.
%
% 4. Repeating the following steps until bestBidPrice <
% bestAskPrice or all buy or sell orders are executed:
%
%   i. Set bestBidSize and bestAskSize to the sizes at indices
%   bidIndex and askIndex respectively.
%
%   ii. If bestAskSize < bestBidSize, mark the order at
%   askIndex as executed, set the size of the order at
%   bidIndex to bestBidSize — bestAskSize and decrement askIndex
%   by 1.
%
%   iii. If bestAskSize > bestBidSize, mark the order at
%   bidIndex as executed, set the size of the order at askIndex
%   to bestAskSize — bestBidSize and increment bidIndex by 1.
```

```
%
%   iv. If bestAskSize = bestBidSize, mark the orders at
%   bidIndex and askIndex as executed, decrement askIndex by 1
%   and increment bidIndex by 1.
%
%   v. Set tradePrice to be the midpoint of bestBidPrice and
%   bestAskPrice.
%
%   vi. Set the current bestBidPrice and bestAskPrice to the
%   prices at indices bidIndex and askIndex respectively.
%
% 6. Remove all orders marked as executed from the order book.

    % We begin by formatting the order book.

    % Order Book Order Type (Buy/Sell) Column
    orderType = simRun.orderBook(:, 3);

    % Logical Indexing for Sell Orders in the Limit Order Book
    sellOrder = (orderType == 1);

    % Logical Indexing for Buy Orders in the Limit Order Book
    buyOrder = (sellOrder == 0);

    % If there are no orders of a certain type, buy or sell, then
    % there are no orders to match and the method should do
    % nothing.

    % Catch Error (No Buy Orders)
    if sum(buyOrder) == 0
        simRun.closingPrice(t + 2) = simRun.closingPrice(t + 1);
        return;
    end

    % Catch Error (No Sell Orders)
    if sum(sellOrder) == 0
        simRun.closingPrice(t + 2) = simRun.closingPrice(t + 1);
        return;
    end

    % Sort Buy Orders
    [~, buySort] = sort(simRun.orderBook(buyOrder, 2), 'descend');
    sortedBuyOrders = simRun.orderBook(buyOrder, :);
    sortedBuyOrders = sortedBuyOrders(buySort, :);

    % Sort Sell Orders
    [~, sellSort] = sort(simRun.orderBook(sellOrder, 2), 'descend');
    sortedSellOrders = simRun.orderBook(sellOrder, :);
    sortedSellOrders = sortedSellOrders(sellSort, :);
```

```
% Combine Sorted Buy and Sell Orders
simRun.orderBook = [sortedSellOrders; sortedBuyOrders;];

% Determine Order Book Length
orderBookLength = length(simRun.orderBook(:, 1));

% Matching Algorithm
% With the order book now in the correct format, we are able to
% execute trades. Trades execute as long as the best ask is less
% than the best bid.

% Initialize Trade Price
tradePrice = 0;

% Find Index of Best Bid
bidIndex = sum(orderType == 1) + 1;

% Index for Best Ask
askIndex = bidIndex - 1;

% Best Bid and Ask Price
bestBidPrice = simRun.orderBook(bidIndex, 2);
bestAskPrice = simRun.orderBook(askIndex, 2);

% Loop Until All Applicable Orders are Executed
while bestAskPrice <= bestBidPrice

    % Order Sizes
    bestBidSize = simRun.orderBook(bidIndex, 1);
    bestAskSize = simRun.orderBook(askIndex, 1);

    % Order Resolution Cases
    if bestBidSize > bestAskSize
        simRun.orderBook(askIndex, 5) = 1; %Mark as executed
        simRun.orderBook(bidIndex, 1) = bestBidSize - bestAskSize;
        askIndex = askIndex - 1;

    elseif bestBidSize < bestAskSize
        simRun.orderBook(bidIndex, 5) = 1; %Mark as executed
        simRun.orderBook(askIndex, 1) = bestAskSize - bestBidSize;
        bidIndex = bidIndex + 1;

    else
        simRun.orderBook(bidIndex, 5) = 1; %Mark as executed
        simRun.orderBook(askIndex, 5) = 1; %Mark as executed
        bidIndex = bidIndex + 1;
        askIndex = askIndex - 1;
```

```
            end

            % Calculate Trade Price
            tradePrice = (bestBidPrice + bestAskPrice) / 2;

            % Check if All Buy or Sell Orders are Executed
            if bidIndex > orderBookLength || askIndex < 1
                break;
            end

            % Calculate New Best Bid and Ask Price
            bestBidPrice = simRun.orderBook(bidIndex, 2);
            bestAskPrice = simRun.orderBook(askIndex, 2);

        end

        % Obtain Indices for Executed Orders
        executedIndices = simRun.orderBook(:, 5) == 1;

        % Remove All Executed Orders from the Order Book
        simRun.orderBook = simRun.orderBook(executedIndices == 0, :);

        % Set Next Closing Price
        if tradePrice > 0 %At least one trade occurs
            simRun.closingPrice(t + 2) = tradePrice;
        else
            simRun.closingPrice(t + 2) = simRun.closingPrice(t + 1);
        end

    end
```

The *orderBook* attribute, considered frequently above, consists of 6 columns, namely order size, order price, order type (buy/sell), time placed, trade flag and remaining sessions. Columns 1 and 2 are self explanatory, column 3 holds a 1 for sell orders and a 2 for buy orders, column 4 indicates the number of minutes from the start of the simulation until the order was placed, column 5 is used to mark orders for removal in the matching algorithm and column 6 indicates how many sessions the order has until removal (if this number reaches 0, the order is removed).

A.2.3 PreisModel

The *PreisModel* class, like the *JacobLealModel* class, encapsulates all the methods and attributes required to implement a complete Preis et al. (2006) model simulation, with a single *PreisModel* object representing a single simulation initialized

with a particular parameter set. This includes processes such as the implementation of agent order placement behaviors, liquidity provider placement depth process increments and liquidity taker buy probability adjustments, with the majority of the class's methods evaluating the model's equations and storing data as needed.

Unlike in the case of the *JacobLealModel* class, order matching procedures are encapsulated in the separate *PreisOrderBook* class, with a *PreisOrderBook* object being stored as an attribute in each *PreisModel* object due to the increased sophistication of the order matching process.

Also worth noting is that we do not create a parameter container class for the Preis et al. (2006) model, as it has significantly fewer parameters than the Jacob Leal et al. (2015) model.

A.2.4 PreisOrderBook

The *PreisOrderBook* class encapsulates all attributes required to store order book data and all methods required to facilitate the placement of limit and market orders in the Preis et al. (2006) model.

New limit orders are placed using the *limitOrder* method. We present the full MATLAB code for this method below:

```
function limitOrderBook = limitOrder(limitOrderBook, buyFlag, ...
        orderSize, limitPrice)
% limitOrder: Insert a specified limit order into the correct
% location in the limit order book.
%
% limitOrderBook = limitOrder(limitOrderBook, buyFlag, orderSize,
% limitPrice) takes in a PreisOrderBook object, limitOrderBook, a
% binary value, buyFlag, with 1 corresponding to an agent who is
% buying, the size of the order, orderSize, and the limit price,
% limitPrice, and proceeds to insert the corresponding order into
% the orderBook attribute stored in the provided PreisOrderBook
% object. Thereafter, the bestBid, bestAsk, numBuy and numSell
% attributes are modified appropriately and the modified
% PreisOrderBook object is then returned.

    % Differentiate Between Buy and Sell Order Cases
    if buyFlag == 1

        % Create Order
        order = [orderSize, limitPrice, 2, 0];

        % Check if Orders Exist in Book
        if limitOrderBook.numBuy == 0
```

```
            % Insert Order
            limitOrderBook.orderBook = [order; limitOrderBook.orderBook];

        else

            % Determine Number of Buy Orders with a Lower Price
            lowerOrders = limitOrderBook.orderBook(:, 2) < limitPrice;
            buyOrders = limitOrderBook.orderBook(:, 3) == 2;
            lowerBuyOrders = sum(lowerOrders .* buyOrders);

            % Place Order in Correct Location
            if lowerBuyOrders ~= 0

                % Insert Order
                limitOrderBook.orderBook = [limitOrderBook.orderBook(1 : ...
                    lowerBuyOrders, :); order; limitOrderBook.orderBook( ...
                    lowerBuyOrders + 1 :  end, :)];

            else

                % Insert Order
                limitOrderBook.orderBook = [order; limitOrderBook.orderBook];

            end

        end

        % Update Number of Buy Orders
        limitOrderBook.numBuy = limitOrderBook.numBuy + 1;

        % Check for New Best Bid
        if (limitPrice > limitOrderBook.bestBid) || (limitOrderBook.bestBid == 0)

            % Update Best Bid
            limitOrderBook.bestBid = limitPrice;

        end

    else

        % Create Order
        order = [orderSize, limitPrice, 1, 0];

        % Check if Sell Orders Exist in Book
        if limitOrderBook.numSell == 0

            % Insert Order
            limitOrderBook.orderBook = [limitOrderBook.orderBook; order];
```

```
        else

            % Determine Number of Sell Orders with a Higher Price
            higherOrders = limitOrderBook.orderBook(:, 2) > limitPrice;
            sellOrders = limitOrderBook.orderBook(:, 3) == 1;
            higherSellOrders = sum(higherOrders .* sellOrders);

            % Place Order in Correct Location
            if higherSellOrders ~= 0

                % Insert Order
                limitOrderBook.orderBook = [limitOrderBook.orderBook(1 : ...
                    end - higherSellOrders, :); order; limitOrderBook.orderBook( ...
                    end - higherSellOrders + 1 :  end, :)];
            else

                % Insert Order
                limitOrderBook.orderBook = [limitOrderBook.orderBook; order];

            end

        end

        % Update Number of Sell Orders
        limitOrderBook.numSell = limitOrderBook.numSell + 1;

        % Check for new Best Ask
        if (limitPrice < limitOrderBook.bestAsk) || (limitOrderBook.bestAsk == 0)

            % Update Best Ask
            limitOrderBook.bestAsk = limitPrice;

        end

    end

end
```

New market orders are placed using the *marketOrder* method. We present the full MATLAB code for this method below:

```
function limitOrderBook = marketOrder(limitOrderBook, buyFlag, orderSize)
% marketOrder: Execute a market order of a certain size.
%
% limitOrderBook = marketOrder(limitOrderBook, buyFlag, orderSize)
% takes in a PreisOrderBook object, limitOrderBook, a binary value,
% buyFlag, with 1 corresponding to an agent who is buying, and a
```

```
% desired order size, orderSize, and proceeds to execute a market
% order of the specified size, modifying the bestBid, bestAsk,
% numBuy and numSell attributes of limitOrderBook accordingly. The
% modified PreisOrderBook object is then returned.

    % Differentiate Between Buy and Sell Order Cases
    if buyFlag == 0

        % Determine Index of Best Bid
        bidIndex = limitOrderBook.numBuy;

        % Exit if One or Fewer Buy Orders
        if limitOrderBook.numBuy <= 1
            return;
        end

        % Iterate Until the Market Order is Fully Executed
        while orderSize > 0

            % Size of Best Bid
            bidSize = limitOrderBook.orderBook(bidIndex, 1);

            % Check if Best Bid is Large Enough to Satisfy Market Order
            if bidSize > orderSize

                % Update Order Sizes
                orderSize = orderSize - bidSize;
                limitOrderBook.orderBook(bidIndex, 1) = bidSize - orderSize;

            else

                % Update Trade Flag
                limitOrderBook.orderBook(bidIndex, 4) = 1;

                % Update Order Size
                orderSize = orderSize - bidSize;

                % Update Index of Best Bid
                bidIndex = bidIndex - 1;

                % Update Best Bid
                limitOrderBook.bestBid = limitOrderBook.orderBook(bidIndex, 2);

                % Update Number of Buy Orders
                limitOrderBook.numBuy = limitOrderBook.numBuy - 1;

            end

        end
```

```
else

    % Determine Index of Best Ask
    askIndex = limitOrderBook.numBuy + 1;

    % Exit if One or Fewer Sell Orders
    if limitOrderBook.numSell <= 1
        return;
    end

    % Iterate Until the Market Order is Fully Executed
    while orderSize > 0

        % Size of Best Ask
        askSize = limitOrderBook.orderBook(askIndex, 1);

        % Check if Best Ask is Large Enough to Satisfy Market Order
        if askSize > orderSize

            % Update Order Sizes
            orderSize = orderSize - askSize;
            limitOrderBook.orderBook(askIndex, 1) = askSize - orderSize;

        else

            % Update Trade Flag
            limitOrderBook.orderBook(askIndex, 4) = 1;

            % Update Order Size
            orderSize = orderSize - askSize;

            % Update Index of Best Ask
            askIndex = askIndex + 1;

            % Update Best Ask
            limitOrderBook.bestAsk = limitOrderBook.orderBook(askIndex, 2);

            % Update Number of Sell Orders
            limitOrderBook.numSell = limitOrderBook.numSell - 1;

        end

    end

end

% Remove Executed Orders
limitOrderBook.orderBook = limitOrderBook.orderBook(...
```

```
        limitOrderBook.orderBook(:, 4) ~= 1, :);

end
```

The *orderBook* attribute, considered frequently above, consists of 4 columns, namely order size, order price, order type (buy/sell) and trade flag. Columns 1 and 2 are self explanatory, column 3 holds a 1 for sell orders and a 2 for buy orders, and column 4 is used to mark orders for removal in the matching algorithm.

A.2.5 ObjectiveFunction

The *ObjectiveFunction* class encapsulates all attributes and methods required to construct and store a weight matrix for a particular set of TRTH (Thomson Reuters (2016)) data and evaluate the objective function for the associated dataset, given some simulated series.

An *ObjectiveFunction* object is typically created for a particular set of TRTH (Thomson Reuters (2016)) data and saved in a *.mat* file. It is then loaded as required whenever the objective function needs to be evaluated.

The key method of this class is the *evaluate* method, which returns the corresponding objective function value for a single series or multiple series (multiple Monte Carlo replications). We present the full MATLAB code for this method below:

```
function functionValue = evaluate(f, simR)
% evaluate: Evaluate the objective function associated with a
% particular dataset, given a measured time series simulated by the
% ABM to be calibrated.
%
% functionValue = evaluate(f, simR) takes in an ObjectiveFunction
% object, f, and a measured time series, simR, and proceeds to
% evaluate the objective function associated with f using the method
% of simulated moments. The calculated objective function value is
% then returned.

    % Create Data Structure to Store G
    G = zeros(size(simR, 2), 5);

    % Repeat for Number of Replications
    for i = 1 : size(simR, 2)

        % Sample Statistics for Simulated Series
        meanSim = mean(simR(:, i));
        stdSim = std(simR(:, i));
        kurtSim = kurtosis(simR(:, i));
```

```
            hurstSim = f.hurstExponent(simR(:, i));

            % Kolmogorov–Smirnov Test Statistic for Simulated and Measured
            % Series
            [~, ~, ksStat] = kstest2(f.R, simR(:, i));

            % Calculate G for Method of Simulated Moments
            G(i, :) = ([meanSim; stdSim; kurtSim; ksStat; hurstSim] - [f.meanR; ...
                f.stdR; f.kurtR; 0; f.hurstR])';

        end

        % Reformat G
        if size(G, 1) > 1
            G = mean(G)';
        else
            G = G';
        end

        % Use Calculated Weight Matrix and Method of Simulated Moments
        % to Obtain an Objective Function Value
        functionValue = G' * f.W * G;

    end
```

The function argument *simR*, appearing frequently above, is a $T \times R$ matrix, where T is the length of a simulated series and R is the number of Monte Carlo replications. In other words, *simR* is a matrix of R simulated series, each of length T.

A.2.6 NelderMeadSimplexJL/NelderMeadSimplexPreis

The *NelderMeadSimplexJL* and *NelderMeadSimplexPreis* classes encapsulate all methods and attributes required to run a single Nelder-Mead simplex calibration experiment for the Jacob Leal et al. (2015) and Preis et al. (2006) models respectively.

They were originally implemented as separate classes in order to maximize computational efficiency and avoid unnecessary comparison operators to select the correct methods, but can essentially be viewed as two subclasses of the *NelderMeadSimplex* superclass.

We detail a number of the key methods common to both classes, representing the essential components of the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic. We consider the versions of these methods presented in the *NelderMeadSimplexJL* class.

The first method we consider, namely *simplex*, performs a single iteration of the standard Nelder-Mead simplex algorithm. We present the full MATLAB code for this method below:

```
function calRun = simplex(calRun)
% simplex: Apply a single iteration of the standard Nelder–Mead
% simplex algorithm.
%
% calRun = simplex(calRun) takes in a NelderMeadSimplexJL object,
% calRun, and proceeds to calculate all transformations to be
% considered using the transform method, after which the simplex
% is modified as per Algorithm 5 in Gilli and Winker (2003),
% modifying theta and thetaf (in calRun) as required.
% Thereafter, the modified NelderMeadSimplexJL object is
% returned.

    % Sort Vertices, Best to Worst
    [calRun.thetaf, index] = sort(calRun.thetaf);
    calRun.theta = calRun.theta(:, index);

    % Transform Points
    calRun = calRun.transform;

    % Set Shrink Flag
    shrinkFlag = false;

    % Reflection < Best Vertex
    if calRun.transPointsf(1) < calRun.thetaf(1)

        % Expansion < Reflection
        if calRun.transPointsf(2) < calRun.transPointsf(1)

            % Set Expansion as Vertex of Interest
            thetaStar  = calRun.transPoints(:, 2);
            thetaStarf = calRun.transPointsf(2);

        else

            % Set Reflection as Vertex of Interest
            thetaStar  = calRun.transPoints(:, 1);
            thetaStarf = calRun.transPointsf(1);

        end

    else
        % Reflection < Second Worst Vertex
        if calRun.transPointsf(1) < calRun.thetaf(calRun.n)
```

```
        % Set Reflection as Vertex of Interest
        thetaStar  = calRun.transPoints(:, 1);
        thetaStarf = calRun.transPointsf(1);

        else

            % Reflection < Worst Vertex
            if calRun.transPointsf(1) < calRun.thetaf(calRun.n + 1)

                % Out-Contraction < Worst Vertex
                if calRun.transPointsf(3) < calRun.thetaf(calRun.n + 1)

                    % Set Out-Contraction as Vertex of Interest
                    thetaStar  = calRun.transPoints(:, 3);
                    thetaStarf = calRun.transPointsf(3);

                else

                    % Shrink
                    calRun = calRun.shrink;
                    shrinkFlag = true;

                end

            else

                 % In-Contraction < Worst Vertex
                if calRun.transPointsf(4) < calRun.thetaf(calRun.n + 1)

                    % Set In-Contraction as Vertex of Interest
                    thetaStar  = calRun.transPoints(:, 4);
                    thetaStarf = calRun.transPointsf(4);

                else

                    % Shrink
                    calRun = calRun.shrink;
                    shrinkFlag = true;

                end

            end

        end

    end

    % Check Shrink Flag
    if shrinkFlag == false
```

```
        calRun.theta(:, calRun.n + 1) = thetaStar;
        calRun.thetaf(calRun.n + 1) = thetaStarf;

    end

end
```

The above makes use of the *shrink* helper method, which shrinks the simplex towards the current best vertex. We present the full MATLAB code for this method below:

```
function calRun = shrink(calRun)
% shrink: Perform a shrink operation required by the Nelder–Mead
% simplex algorithm.
%
% calRun = shrink(calRun) takes in a NelderMeadSimplexJL object,
% calRun, and proceeds to shrink all current simplex vertices,
% stored in the attribute theta, towards the current best
% vertex, in the first column of theta. After the operation is
% performed and theta modified appropriately, the objective
% function values for theta, stored in thetaf, are updated
% using the evaluateJL method. The modified NelderMeadSimplexJL
% object is then returned.
%
% See also evaluateJL.

    % Variable to Store Transformed Points
    shrinkPointsTemp = zeros(calRun.n, calRun.n + 1);
    shrinkPointsTemp(:, 1) = calRun.theta(calRun.freePos, 1);

    % Shrink Other Vertices Towards Best Current Vertex in Parallel
    parfor i = 2 : calRun.n + 1
        shrinkPointsTemp(:, i) = calRun.theta(calRun.freePos, 1) ...
            - calRun.sigma * (calRun.theta(calRun.freePos, i) -  ...
            calRun.theta(calRun.freePos, 1));
    end

    % Set Value of Transformed Points in NelderMeadSimplexJL Object
    calRun.theta(calRun.freePos, :) = shrinkPointsTemp;

    % Create Data Structure to Store Objective Function Values for
    % Simplex Vertices After Shrinking
    objFunctionShrink = zeros(calRun.n + 1, 1);
    objFunctionShrink(1) = calRun.thetaf(1);

    % Calculate Objective Function Values for Each Vertex After
    % Shrinking, 5 Replications
```

```
    parfor i = 2 : calRun.n + 1
        objFunctionShrink(i) = calRun.evaluateJL(calRun.theta(:, i), 5);
    end

    % Set Objective Function Values in NelderMeadSimplexJL Object
    calRun.thetaf = objFunctionShrink;

end
```

The next method we consider, namely *thresholdAccepting*, performs a single iteration of the threshold accepting heuristic. We present the full MATLAB code for this method below:

```
function calRun = thresholdAccepting(calRun)
% thresholdAccepting: Apply a single iteration of threshold
% accepting to the current simplex.
%
% calRun = thresholdAccepting(calRun, iteration) takes in a
% NelderMeadSimplexJL object, calRun, and proceeds to apply the
% threshold accepting algorithm as discussed by Gilli and Winker
% (2003).The modified NelderMeadSimplexJL object is then returned.

    % Randomize Seed
    rng('shuffle');

    % Generate Thresholds
    tau = linspace(1/5, 0, 3);

    % Repeat for 3 Rounds
    for r = 1 : 3

        % Repeat for 5 Steps
        for s = 1 : 5

            % Randomly Determine Direction of Shift (Select Parameter to
            % Shift)
            direction = randsample(calRun.freePos, 1);

            % Randomly Determine Magnitude of Shift
            magnitude = (-0.5 + rand) * mean(calRun.theta(direction, :));

            % Shift Simplex
            thetaTemp = calRun.theta;
            thetaTemp(direction, :) = thetaTemp(direction, :) + magnitude;

            % Structure to Store Objective Function Values for Shifted
            % Simplex
            thetaTempf = zeros(calRun.n + 1, 1);
```

```
            % Calculate Objective Function Values for Each Shifted
            % Vertex, r + 2 Replications
            parfor i = 1 : calRun.n + 1
                thetaTempf(i) = calRun.evaluateJL(thetaTemp(:, i), r + 2);
            end

            % Accept if no Worse than Current Threshold
            if min(thetaTempf) < min(calRun.thetaf) + tau(r)
                calRun.thetaf = thetaTempf;
                calRun.theta = thetaTemp;
            end

        end

    end

end
```

The final method we consider, namely *calibrate*, combines the Nelder-Mead simplex algorithm and threshold accepting heuristic and performs the desired number of iterations of the combined algorithm. We present the full MATLAB code for this method below:

```
function calRun = calibrate(calRun)
% calibrate: Perform the desired number of calibration iterations,
% randomly choosing between the threshold accepting heuristic and
% Nelder–Mead Simplex algorithm at each iteration.
%
% calRun = calibrate(calRun) takes in a NelderMeadSimplexJL object,
% calRun, and proceeds to implement Algorithm 7 presented by
% Gilli and Winker (2003) for the desired number of iterations,
% stored in the nCal attribute of calRun. After the desired number
% of iterations has been performed, the modified NelderMeadSimplexJL
% object is returned.
%
% See also simplex, thresholdAccepting.

    % Set–Up Waitbar
    calibrationProgress = waitbar(0, 'Initializing calibration...');

    % Iterate for the Desired Number of Calibration Iterations
    for i = 1 : calRun.nCal

        % Update Waitbar Progress
        progressString = sprintf('Calibration Run %d of %d', i, calRun.nCal);
        waitbar(i/calRun.nCal, calibrationProgress, progressString);
```

```
        % Shuffle Random Number Generators
        rng('shuffle');

        % Determine Strategy
        if rand <= 0.15

            % Threshold Accepting
            calRun = calRun.thresholdAccepting;

        else

            % Nelder–Mead Simplex
            calRun = calRun.simplex;

        end

        % Round Trader Population Parameters (Must be Whole Numbers)
        calRun.theta(19, :) = round(calRun.theta(19, :));
        calRun.theta(20, :) = round(calRun.theta(20, :));

        % Determine Current Best Vertex
        [currentBestf, index] = min(calRun.thetaf);

        % Check if Better than Previous Best
        if currentBestf < calRun.bestThetaf

            % Update Stored Best
            calRun.bestTheta = calRun.theta(:, index);
            calRun.bestThetaf = currentBestf;

        end

        % Display Current Simplex
        clc;
        display(calRun.theta, 'Current Vertices');
        display(calRun.thetaf, 'Current Objective Function Values');
        display(calRun.bestThetaf, 'Best Objective Function Value');

    end

    % Close Waitbar
    close(calibrationProgress);

end
```

The following attributes appear frequently in the above code excerpts and are described in the context of the Jacob Leal et al. (2015) model as follows. Note

that analogous constructions apply to the Preis et al. (2006) model.

theta

Each column in *theta* represents a single vertex, with each row representing a parameter value. In the case of the Jacob Leal et al. (2015) model, each column has the format:

$$[\theta; \theta_{max}; \theta_{min}; \eta_{max}; \eta_{min}; \kappa_{max}; \kappa_{min}; \alpha^c; \sigma^c; \alpha^f; \sigma^f; \sigma^y; \delta; \sigma^z; \zeta; \gamma_L; \gamma_H; \lambda; N_L; N_H; P_0; P_1]$$

freePos

A vector that contains the indices of free parameters in each column of *theta*. As an example, if N_H and N_L were free parameters in a calibration experiment involving the Jacob Leal et al. (2015) model, *freePos* would be:

$$[19, 20]$$

transPoints

A matrix that has a similar structure to *theta*, where the first column represents the reflection, the second column represents the expansion, the third column represents the out-contraction and the fourth column represents the in-contraction of the current worst vertex.

thetaf

A vector that stores the corresponding objective function value for each simplex vertex in *theta*. The first item in this vector corresponds to the first column in theta and so on.

transPointsf

A vector that is analogous to *thetaf*, but representing objective function values associated with *transPoints*.

A.3 Discussion of Implemented Functions

In this section, we provide a brief overview of the purpose of each implemented function and provide illustrative code extracts where appropriate.

A.3.1 Data Preprocessing and Trade Classification Functions

The project includes a number of functions designed to preprocess sets of TRTH (Thomson Reuters (2016)) data and classify the signs of all occurring trades, ultimately used to construct the trade sign ACF in Figure 7.1c. These include:

1. *dataClean*, which removes any erroneously recorded data items and ensures that only data items occurring between 9:10 and 16:50 on any particular trading day are considered, ensuring that auction data items are not included in the preprocessed dataset. This method also performs a variety of other data cleaning tasks.

2. *dataCompact*, which aggregates all trades and quotes with duplicate timestamps into single entries. The final quote in a sequence is kept, whereas a sequence of trades is aggregated into a single trade with volume being the total volume of the considered trades and price being the volume weighted average price of the same set of trades.

3. *dataClassify*, which classifies all trades according to the Lee and Ready (1991) algorithm.

The above make use of a number of helper functions, which we omit for the sake of brevity and due to the fact that the construction of the aforementioned ACF is a relatively minor component of this investigation.

A.3.2 fitnessJLGA/fitnessPreisGA

The functions *fitnessJLGA* and *fitnessPreisGA* are designed to act as fitness functions to be provided as input to MATLAB's *ga* function for genetic algorithm calibration experiments involving the Jacob Leal et al. (2015) and Preis et al. (2006) models respectively.

The full MATLAB code for *fitnessJLGA* is presented below, with *fitnessPreisGA* being similar in nature.

```
function functionVal = fitnessJLGA(param)
% fitnessJLGA: Determine the objective function value associated with a
% particular set of Jacob Leal et al. (2015) model parameters. Formatted for
% use as a fitness function in the ga method provided by MATLAB.
%
% functionVal = fitnessJLGA(param) takes in a set of Jacob Leal et al.
% (2015) model parameters, param, used as free parameters in calibration
% experiments, and proceeds to construct JacobLealModel and JLParams objects
```

```
% in order to generate simulated time series. Thereafter, the desired
% objective function is evaluated for the simulated series, resulting in an
% objective function value.
%
% Params is a vector of 10 elements, corresponding in order to: alpha_c,
% sigma_c, alpha_f, sigma_f, sigma_y, delta, sigma_z, lambda, N_L and N_H.
%
% The objective function is manually specified in the function source code
% and the specified filename should be changed as required for different
% ObjectiveFunction objects.
%
% See also JacobLealModel, JLParams, ga.
%
% References
% 1. Jacob Leal S, Napoletano M, Roventini A, Fagiolo G (2015) Rock Around
% the Clock: An Agent-Based Model of Low- and High-frequency Trading.
% J Evol Econ 25:1-25
%
% Donovan Platt
% School of Computer Science and Applied Mathematics
% University of the Witwatersrand, Johannesburg, South Africa.

    %% Parameter Values

    % Calibration Data Set
    fileName = 'AGLJ_01Nov2013_05Nov2013_Bars_2300';

    % Fixed Parameters
    theta = 20;               %LF traders' trading frequency mean
    thetaUpper = 40;          %LF traders' max trading frequency
    thetaLower = 10;          %LF traders' min trading frequency
    zeta = 1;                 %LF traders' intensity of switching
    etaLower = 0;             %HF traders' activation threshold distribution support
    etaUpper = 0.2;           %HF traders' activation threshold distribution support
    kappaLower = 0;           %HF traders' order price distribution support
    kappaUpper = 0.01;        %HF traders' order price distribution support
    gamma_L = 20;             %LF traders' resting order periods
    gamma_H = 1;              %HF traders' resting order periods
    initialPriceA = 238.745; %First closing price
    initialPriceB = 238.720; %Second closing price

    % Free Parameters
    alpha_c = param(1);      %Chartists' order size parameter
    sigma_c = param(2);      %Chartists' shock standard deviation
    alpha_f = param(3);      %Fundamentalists' order size parameter
    sigma_f = param(4);      %Fundamentalists' shock standard deviation
    sigma_y = param(5);      %Fundamental value shock standard deviation
    delta = param(6);        %Price drift parameter
    sigma_z = param(7);      %LF traders' price tick standard deviation
```

```
    lambda =  param(8);      %Market volumes weight in HF traders order size distribut
    N_L = round(param(9));  %Number of LF traders
    N_H = round(param(10)); %Number of HF traders

    %% Evaluate Objective Function

    % Load Objective Function
    f = load(fileName, 'f');
    f = f.f;

    % Create Object to Hold Model Parameters
    paramSet = JLParams(theta, thetaUpper, thetaLower, etaUpper, etaLower, ...
        kappaUpper, kappaLower, alpha_c, sigma_c, alpha_f, sigma_f, sigma_y,...
        delta, sigma_z, zeta, gamma_L, gamma_H, lambda);

    % Create Data Structure to Store Simulated Log Prices
    prices = zeros(length(f.R), 5);

    % Repeat for 5 Replications
    for i = 1 : 5

        % Create JacobLealModel Object
        simRun = JacobLealModel(N_L, N_H, N_L + N_H, length(f.R) - 2, ...
            initialPriceA, initialPriceB, paramSet);

        % Start Simulation
        simRun = simRun.simulate(0, 19 + i - 1);

        % Calculate Log Prices
        prices(:, i) = log(simRun.closingPrice);

    end

    % Calculate Objective Function Value
    functionVal = f.evaluate(prices);

    % Shuffle Random Number Generators
    rng('shuffle');

end
```

A.4 Discussion of Implemented Scripts

In this section, we provide a brief overview of the purpose of each implemented script. The majority of these scripts simply initialize objects and call methods as required to perform certain experiments, but brief descriptions are included for

the sake of completeness. They are as follows:

1. *Masters_ABM_Scripts_CalibrationResultSummaryJL* and *Masters_ABM_Scripts_CalibrationResultSummaryPreis* summarize the results of multiple Jacob Leal et al. (2015) and Preis et al. (2006) model calibration experiments respectively by determining appropriate parameter estimate confidence intervals and generating sample price paths using the best calibrated parameter set, ultimately used to calculate simulated moment confidence intervals.

2. *Masters_ABM_Scripts_GACalibrationJL* and *Masters_ABM_Scripts_GACalibrationPreis* initialize genetic algorithm calibration experiments involving the Jacob Leal et al. (2015) and Preis et al. (2006) models respectively.

3. *Masters_ABM_Scripts_NelderMeadCalibrationJL* and *Masters_ABM_Scripts_NelderMeadCalibrationPreis* initialize Nelder-Mead simplex calibration experiments involving the Jacob Leal et al. (2015) and Preis et al. (2006) models respectively.

4. *Masters_ABM_Scripts_StylizedFactTestingJL* and *Masters_ABM_Scripts_StylizedFactTestingPreis* test for empirically-observed log return time series stylized facts in time series generated using the Jacob Leal et al. (2015) and Preis et al. (2006) models respectively.

5. *Masters_ABM_Scripts_ParameterSurfacesJL_A* generates a Jacob Leal et al. (2015) model objective function surface for any two model parameters, excluding N_H and N_L, and *Masters_ABM_Scripts_ParameterSurfacesJL_B* generates a Jacob Leal et al. (2015) model objective function surface for N_H and N_L.

6. *Masters_ABM_Scripts_ParameterSurfacesPreis* generates a Preis et al. (2006) model objective function surface for any two model parameters.

7. *Masters_ABM_Scripts_MeasuredOrderFlow* plots the ACF for trade signs associated with a particular set of TRTH (Thomson Reuters (2016)) data.

8. *Masters_ABM_Scripts_OutlierTesting* determines if the first data point in a time series is an outlier using the boxplot method proposed by Tukey (1977).

Providing the code for the above would significantly increase the length of this dissertation and is thus impractical.

A.5 Parallel Simulation Examples

This section provides examples of the basic procedures employed to perform ABM simulations in parallel and illustrates the elegance of OOP, where an array of objects can be used to store and run each simulation.

A.5.1 Jacob Leal et al. Model

A parallel simulation involving 50 Jacob Leal et al. (2015) model Monte Carlo replications can be conducted in MATLAB as follows:

```
%% Set Model Parameters

% Number of Monte Carlo Replications
R = 50;

% Fixed Model Parameters
T = 1200;           %Number of trading sessions
N_L = 10000;        %Number of low-frequency traders
N_H = 100;          %Number of high-frequency traders
theta = 20;         %LF traders' trading frequency mean
thetaUpper = 40;    %LF traders' max trading frequency
thetaLower = 10;    %LF traders' min trading frequency
alpha_c = 0.04;     %Chartists' order size parameter
sigma_c = 0.05;     %Chartists' shock standard deviation
alpha_f = 0.04;     %Fundamentalists' order size parameter
sigma_f = 0.01;     %Fundamentalists' shock standard deviation
sigma_y = 0.01;     %Fundamental value shock standard deviation
delta = 0.0001;     %Price drift parameter
sigma_z = 0.01;     %LF traders' price tick standard deviation
zeta = 1;           %LF traders' intensity of switching
gamma_L = 20;       %LF traders' resting order periods
gamma_H = 1;        %HF traders' resting order periods
etaLower = 0;       %HF traders' activation threshold distribution support
etaUpper = 0.2;     %HF traders' activation threshold distribution support
lambda = 0.625;     %Market volumes weight in HF traders order size distribution
kappaLower = 0;     %HF traders' order price distribution support
kappaUpper = 0.01;  %HF traders' order price distribution support

% Initial Price Settings
initialPriceA = 100;
initialPriceB = 100 * (1 + delta) * (1 + normrnd(0, sigma_z));

% Create JLParams Object to Hold Parameter Values
paramSet = JLParams(theta, thetaUpper, thetaLower, etaUpper, ...
    etaLower, kappaUpper, kappaLower, alpha_c, sigma_c,      ...
```

```
    alpha_f, sigma_f, sigma_y, delta, sigma_z, zeta,          ...
    gamma_L, gamma_H, lambda);

%% Model Simulation

% Create Data Structure to Store Simulation Objects
simRun = cell(R, 1);

% Create JacobLealModel Objects
for i = 1 : R
    simRun{i} = JacobLealModel(N_L, N_H, N_L + N_H, T, initialPriceA, ...
        initialPriceB, paramSet);
end

% Start Simulation
parfor i = 1 : R
    simRun{i} = simRun{i}.simulate(0, 100 + i);
end
```

A.5.2 Preis et al. Model

A parallel simulation involving 50 Preis et al. (2006) model Monte Carlo replications can be conducted in MATLAB as follows:

```
%% Set Model Parameters

% Number of Monte Carlo Replications
R = 50;

% Fixed Parameters
N_A = 250;          %Number of traders of each type
delta = 0.025;      %Order cancellation probability
lambda_0 = 100;     %Initial limit order placement depth parameter
C_lambda = 10;      %Placement depth coefficient
delta_S = 0.001;    %Liquidity taker buy probability random walk increment
alpha = 0.15;       %Activation frequency for liquidity providers
mu = 0.025;         %Activation frequency for liquidity takers

%% Model Simulation

% Create Data Structure to Store Simulation Objects
simRun = cell(R, 1);

% Create PreisModel Objects
for i = 1 : R
    simRun{i} = PreisModel(N_A, delta, lambda_0, C_lambda, delta_S, alpha, mu, ...
```

```
        10 ^ 6, 2400);
end

% Start Simulation Experiments
parfor i = 1 : R

    % Initialize Simulation
    simRun{i} = simRun{i}.initialize;

    % Start Simulation
    simRun{i} = simRun{i}.simulate;

end
```

In the *PreisModel* class constructor above, 2400 is the number of MC steps and 10^6 is the initial price, in tick units.

A.6 Online Source Code Availability

It is intended that the complete source code for this project be made publicly available at a later stage, once all of the associated publications have been published. This source code repository can be found at: https://github.com/DFPlatt/MSc-ABM

Appendix B

Robustness of Conclusions: Number of Objective Function Moments Versus Number of Free Parameters

It is important to address potential criticism relating to the approach taken to calibrate the models considered, such that we are able to ensure that our conclusions are robust to some extent. One such criticism may be the fact that we attempt to calibrate 10 free parameters in the Jacob Leal et al. (2015) model experiments but employ only 5 moments in our objective function.

In this context, it is important to realize that the work of Fabretti (2013) and the body of work on which it is based, namely the work of Gilli and Winker (2003), similarly employs fewer moments than free parameters in the conducted estimation procedures.

Nevertheless, not addressing such criticism would place significant doubt on whether the conclusions reached in this investigation are robust or simply the result of experimental design. We therefore repeat the Jacob Leal et al. (2015) model calibration experiments, as before, but this time consider a smaller free parameter set and apply only the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic.

In these experiments, we set N_H, N_L, δ and σ^z to be the only free parameters and verify that δ once again emerges as a more easily identifiable parameter than the remaining free parameters and that the associated objective function surface once again demonstrates type 3 behavior. In addition to this, we verify that N_H and N_L are once again difficult to uniquely identify and that the associated parameter surface again demonstrates type 1 behavior, indicative of parameter degeneracies.

Table B.1: Nelder-Mead Simplex Calibration Results (Reduced Jacob Leal et al. (2015) Model Free Parameter Set)

Parameter	95% Conf Int	$\frac{s}{\sqrt{n}}$
δ	$[0.0016, 0.0063]$	0.0011
σ^z	$[0.0238, 0.0452]$	0.0051
N_L	$[835, 3696]$	683.5941
N_H	$[2082, 8935]$	1637.0516

95% confidence intervals for the reduced set of Jacob Leal et al. (2015) model free parameters, obtained from 20 independent calibration experiments involving the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic

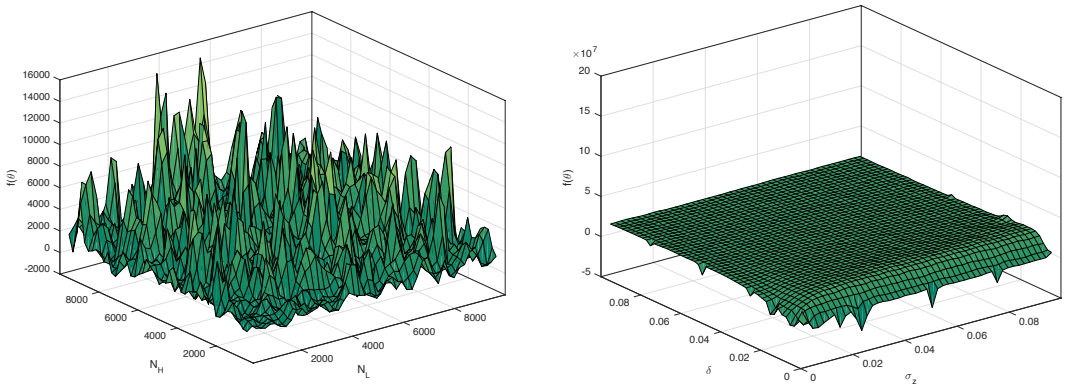
Table B.2: Best Parameter Set Obtained Through Calibration (Reduced Jacob Leal et al. (2015) Model Free Parameter Set)

Calibrated Parameter	Value
δ	0.000008
σ^z	0.0001
N_L	1005
N_H	7299

Best parameter set obtained through the implementation of the Nelder-Mead simplex algorithm combined with the threshold accepting heuristic in experiments involving the reduced Jacob Leal et al. (2015) model free parameter set

As is clearly evident in Table B.1, we observe that δ once again emerges as showing some evidence of convergence, with a much smaller confidence interval and standard error than the remaining free parameters. Furthermore, referring to Figures B.1a and B.1b, we observe that N_H and N_L once again result in a type 1 objective function surface, though slightly different to that observed in Figure 6.1a, while δ and σ^z once again result in a type 3 objective function surface that is nearly identical to that presented in Figure 6.1c. This is in spite of the fact that the calibration experiments were repeated with fewer free parameters than objective function moments.

It is therefore evident that our results cannot be explained as a mere consequence of the relative number of moments and parameters employed in the calibration procedures, with this discussion also applying to the Preis et al. (2006) model experiments.



(a) Type 1 behavior: N_H and N_L

(b) Type 3 behavior: δ and σ^z

Figure B.1: Objective function surfaces associated with the reduced Jacob Leal et al. (2015) model free parameter set, generated using the parameter values in Table B.2 as the base values for the free parameters, with all other parameters set according to the values specified in Table 5.1