An agent-based simulation of double-auction markets

by

Ted Xiao Guo

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

An agent-based simulation of double-auction markets

Ted Xiao Guo

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

Agent-based simulation has emerged as a promising research alternative to the traditional analytical approach in studying financial markets. The idea behind agent-based simulation is to constructively model a market in a bottom-up fashion using artificial agents. This thesis aims to accomplish four objectives. The first objective is to present a high-quality software platform that is capable of carrying out agent-based financial market simulations. The second objective is to simulate a double-auction market with this software platform, using homogeneous zero-intelligence agents. The simulation is based on the artificial market model proposed by Smith, Farmer, Gillermot and Krishnamurthy (2003). The third objective is to study the microstructure properties of the simulated double-auction market. In particular, we investigate certain statistical properties for the mid-price, bid-ask spread and limit order book. The fourth and the last objective is to explore and evaluate trading strategies associated with the time-constrained asset liquidation problem through computational agents.

Dedication

To my dear Mom and Dad, whom I owe so much to.

Acknowledgements

I cannot express in words how grateful and fortunate I feel for having been surpervised by two terrific professors: Robert Almgren and Ken Jackson. They have provided me with by far the most valuable and enjoyable learning experience I have ever had. Many things that I have learned from them will benefit me for a life time. Rob suggested the research topic and walked me through every single stage of this thesis. He is always very insightful and able to quickly pinpoint the source of many problems that frustrated me at times. But above all, he is a patient teacher who is always enthusiastic about conveying knowledge to his students. Ken also spent a tremendous amount of time contributing to this thesis. His detailed comments and corrections are always very valuable. Moreover, Ken is one of the most patient and considerate persons I have ever met. His consistent support in many aspects of my student life has made my experience as a graduate student a very smooth and pleasant one.

I would also like to thank the University of Toronto for providing generous funding support, which made it possible for me to focus on learning and conducting research as an international student.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advance of computing power and modeling tools, computer simulations have become increasingly important in many financial applications, such as derivative pricing and risk management. Now the already-booming family of financial simulations is witnessing the rise of a new member: agent-based market simulation. Since the beginning of the 1990's, there has been a surge of interest within the finance community (both in industry and academia) in employing agent-based methods to obtain insights about market microstructure and to experiment with trading strategies. Research in agent-based financial markets naturally provides an excellent opportunity for interdisciplinary collaborations, because it often involves joint projects involving financial engineers, economists, computer scientists, mathematicians, statisticians, physicists, etc. This thesis aims to contribute to the field of agent-based computational finance from the perspective of a computer scientist, by introducing a flexible simulation and modeling environment for double-auction markets.

1.1 Background

1.1.1 Agent-based financial markets

Agent-based financial markets are market simulations populated with artificial agents that fill the roles of participants of a real market, such as traders and market makers. These agents can vary from simple random zero-intelligence (ZI) agents as in Gode and Sunder (1993) to sophisticated inductive learning agents as in the Santa Fe artificial stock market (LeBaron, Arthur, and Palmer 1999). Regardless of this variation in the levels of agent intelligence, most research in agent-based artificial markets centers on modeling financial markets as dynamic structures that emerge from the interactions of individual agents.

The general agent-based modeling method is not new and it has already been applied in many different economic and social contexts.[1] However, there are certain reasons which make financial markets particularly appropriate and appealing for agent-based modelers. As LeBaron (2001) points out, financial markets are one of the most important applications for agent-based modeling because issues of price and information aggregation and dissemination tend to be sharper in financial settings, where objectives of agents are usually clearer. Further, the availability of massive amounts of real financial data allows for comparison with the results of agent-based simulations. As of now, agent-based financial modeling has emerged as a third major research approach to the analysis of market microstructure, rivaling theoretical and empirical approaches.

The fast-growing interest and popularity of agent-based financial modeling is motivated by the desire for an alternative research methodology (in studying financial markets) to the traditional theoretical approach, which tends to focus on constructing analytic frameworks based on the assumption of rational behaviors and the simplification of

[1]Readers interested in the more general field of Agent-based Computational Economics (ACE) should see the comprehensive and extensive guide maintained by Leigh Tesfatsion at http://www.econ.iastate.edu/tesfatsi/ace.htm.

market complexities. A common criticism of the traditional theoretical approach is that actual markets often exhibit evidence that contradicts the perfect rationality assumption. Also the simplification of market complexities often makes the analytical models less applicable in the real world. Agent-based modeling allows one to overcome these shortcomings of the theoretical approach and to gain insights into market phenomena through computational experiments. As Das (2003) points out: "The artificial market approach allows a fine-grained level of experimental control that is not available in real markets. Thus, data obtained from artificial market experiments can be compared to the predictions of theoretical models and to data from real-world markets, and the level of control allows one to examine precisely which settings and conditions lead to the deviations from theoretical predictions usually seen in the behavior of real markets."

Because of the surge of activities in developing artificial financial markets, today we are faced with so many different styles of agent-based simulations that it is almost impossible to classify them. However, most of the research in this field seems to fall into the following two categories.

1. The first category focuses on the studies of market microstructure. Typically, the goal of research in this category is to replicate observed properties of financial time series observed in real markets and to try to explain the cause of these properties. For example, Chiarella and Iori (2002) construct an agent-based simulation of a double-auction market, from which they are able to gain some insights into the determinants of order flow dynamics, such as tick size, liquidity and average order life time. The working mechanism of their artificial market is reviewed in section 2.1.

2. The second category focuses more on the design and evaluation of trading strategies. It is often either infeasible or too costly to carry out trading strategy experiments in real markets. Researchers in this category use agent-based artificial markets as underlying testbeds, on which they can test many trading strategies by deploying

special agents (agents that are programmed to follow specific trading strategies) to trade in these simulated markets. For example Das (2003) describes and evaluates a market-making algorithm for setting prices in financial markets with asymmetric information. His experimental results are obtained from a simulated double-auction market populated by artificial agents that represent either informed or uninformed traders. We review Das' model in section 2.2.

LeBaron (2000) provides a summary of some of the early and influential research work on agent-based computational finance.

1.1.2 Double-auction markets

This thesis focuses on the agent-based simulation of a continuous double-auction market. In what follows, we provide a brief introduction to the basic microstructure and trading mechanism of a standard double-auction market, on which most modern financial markets are based.

Impatient traders submit *market orders*, which are requests to buy or sell a given number of shares immediately at the best available price. More patient traders submit *limit orders*, which also state a *limit price* corresponding to the worst allowable price for the potential transcation. Market orders guarantee execution, but not price. On the other hand, limit orders normally guarantee price, but not execution. Since limit orders often fail to result in immediate transactions, they frequently accumulate as they arrive at the market and form a *limit order book*, inside which limit buy orders are stored in decreasing order of limit price and limit sell orders are stored in increasing order of limit price. Limit buy orders are often called *bids* and limit sell orders are often called *asks*. The limit order book is the key to understanding the double-auction market. A snapshot of a typical limit order book is illustrated in Table 1.1.

Limit Buy Orders		Limit Sell Orders	
Size	*Price*	*Size*	*Price*
20	$1.10	25	$1.17
20	$1.08	30	$1.18
25	$1.03	20	$1.21
35	$1.02	30	$1.29
20	$1.00	30	$1.33
40	$0.97	35	$1.36
......		......	

Table 1.1: A snapshot example of a limit order book

Limit Buy Orders		Limit Sell Orders	
Size	*Price*	*Size*	*Price*
15	$1.03	25	$1.17
35	$1.02	30	$1.18
20	$1.00	20	$1.21
40	$0.97	30	$1.29
		30	$1.33
		35	$1.36
......		......	

Table 1.2: The limit order book, after the execution of a market sell order of 50 shares

Limit orders (either buy or sell) are the source of *liquidity* in the market as they provide the necessary pools of supply and demand. For a normal double-auction market, the best (highest) bid price is always less than the best (lowest) ask price. The difference between the two is called the *spread* of the market. For example, the limit order book shown in Table 1.1 has a market spread of \$0.07. Market orders, on the other hand, consume the liquidity, because, when they arrive, they get executed against limit orders on the opposite side of the book. For example, when a market sell order for 50 shares arrives at the market (assuming that the market at this moment has the limit order book shown in Table 1), it would first get executed against the current best bid (i.e., the limit buy order that has the highest limit price) which is located at the top of the bids list (25 shares at \$1.17 per share). Since the size of the market sell (50 shares) is greater than that of the best bid (25 shares), the remainder of the market sell order (25 shares) will then get executed against the next best bid (20 shares at \$1.08 per share) and so on, until it has been completely satisfied. After this trade, the limit order book will change to Table 1.2. And the market spread now widens to \$0.14. The trader's revenue from the sale is

$$\$1.10 \times 20 + \$1.08 \times 20 + \$1.03 \times 10 = \$53.90$$

Note limit orders at prices within the spread cause the spread to narrow as they arrive at the market, whereas market orders may widen the spread as they get executed.

1.2 Contributions

The research described in this thesis can be viewed as an attempt to bridge the gap between the relevant disciplines (such as finance, computer science, mathematics, statistics, physics, etc.) involved in the field of agent-based computational finance. In particular, we hope that our work can bring the following benefits to researchers from those disciplines who wish to enter the field of agent-based financial modeling.

1. This thesis introduces the “big picture” of agent-based computational finance by identifying and explaining the key building blocks and applications.

2. This thesis contributes a high-quality software platform that is specifically designed to carry out agent-based simulations of financial markets. The platform is flexible enough to accommodate many different kinds of agent-based artificial market models.

3. This thesis presents experimental results for the properties of the artificial double-auction market proposed by Smith, Farmer, Gillemot, and Krishnamurthy (2003) (referred to as SFGK henceforth) , such as the mid-price, bid-ask spread and limit order book.

4. This thesis explores and evaluates trading strategies for the practical time-constrained asset liquidation problem through extensive simulation experiments.

1.3 Overview

This thesis is structured as follows. Chapter 2 surveys three research papers on artificial double-auction markets which we think are representative. Chapter 3 presents the software simulation platform, explains its properties and describes its design and structure. It also shares some insights about building simulation software of this kind. Chapter 4 formally introduces the SFGK artificial market model. It justifies why we choose to use this particular model in this thesis, explains how the model generates market dynamics and how it is implemented in an agent-based setting on top of our simulation platform. Chapter 5 investigates the market microstructure of the SFGK artificial market. In particular, it shows certain statistical results for the mid-price, bid-ask spread and limit order book. Chapter 6 focuses on trading strategy experiments. It describes the time-contrained asset liquidation problem and introduces two different trading strategies

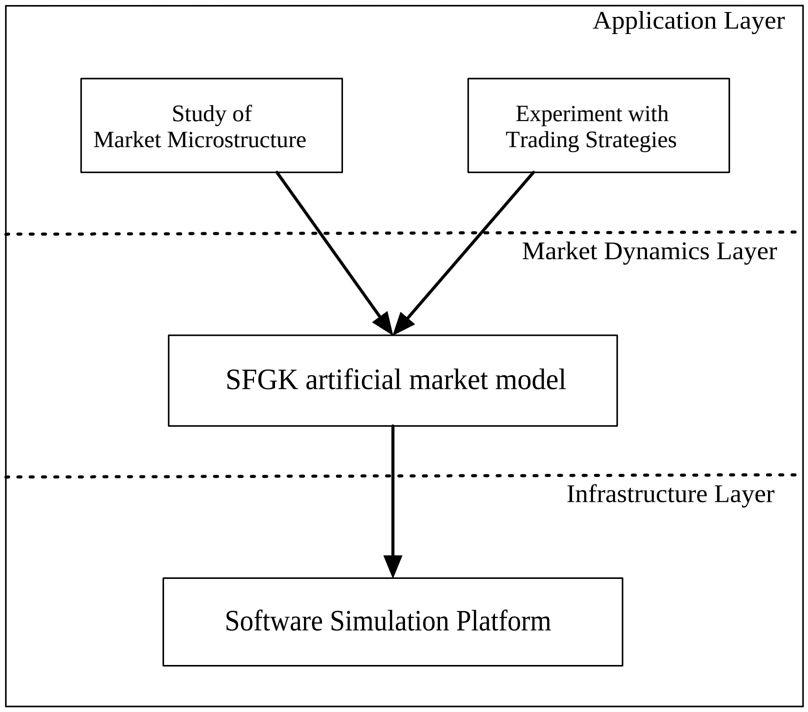

Application Layer
Study of Market Microstructure
Experiment with Trading Strategies
Market Dynamics Layer
SFGK artificial market model
Infrastructure Layer
Software Simulation Platform

Figure 1.1: Illustraction of thesis layout

associated with it. Through simulation experiments, it evaluates the performance of the two strategies and shows that on average one strategy outperforms the other. Chapter 7 summarizes the work of this thesis and suggests avenues for future research.

Figure 1.1 gives an illustration of the structure and key components of this thesis.

Chapter 2

Survey of artificial double-auction markets

Agent-based financial markets have been undergoing significant growth in recent years. The ever-increasing number of artificial market models and the huge variations among them make it a daunting task for anyone who wishes to survey the entire literature. In this chapter, we survey a few research paper which we think are representative in terms of modeling double-auction markets through computational agents. To give readers a good taste of the agent-based modeling methodology, the survey focuses on the mechanisms that are used in these papers to generate the market dynamics of a double-auction market, not on the experimental results or statistical conclusions that are obtained from the simulations of these artificial models.

2.1 The Chiarella-Iori model

Chiarella and Iori (2002) introduce an artificial double-auction market model with heterogeneous agents which set bids and asks and post market and limit orders according to certain pre-defined rules. The model makes the following assumptions.

- A single asset is traded in the market. Orders arrive at the market sequentially and at random times and have a finite life time τ. Each agent can trade one stock only at a time.

- Agents know the fundamental value p^f of the asset, and p^f is taken to be constant in their model.

- The demands of agents (traders) are assumed to consist of three components, a fundamentalist component, a chartist component and a noise-induced component.

- All agents have access to the past history of prices. At any time t, the price p_t is given by the price at which a transaction occurs, if a transaction occurred at time t. If no transaction occurred at time t, the price p_t is set to the mid-point price (i.e., the average price of the highest bid and the lowest ask) at time t.

- Agents can submit limit orders at any limit price on a pre-specified grid, defined by the tick size Δ.

The way the model generates market dynamics is as follows. At any time t, a trader i is chosen, with a given probability λ, to enter the market. The chosen agent, using both the fundamental value and chartist rules, predicts a spot return $\hat{r}^i_{t,t+\tau}$of the asset that he thinks will prevail in the time period $(t, t+\tau)$. The precise formula that the agent uses to compute his expectation of the spot return $\hat{r}^i_{t,t+\tau}$ is

$$\hat{r}^i_{t,t+\tau} = g^i_1 \frac{(p^f - p_t)}{p_t} + g^i_2 \bar{r}_{L_i} + n_i \epsilon_t$$

The quantities $g^i_1 > 0$ and g^i_2 represent the weights given to the fundamentalist and chartist components respectively. The sign of g^i_2 indicates a trend chasing (> 0) or contrarian (< 0) chartist strategy. Since the degree of fundamentalism and chartism will vary across agents, Chiarella and Iori model these parameters as random variables independently chosen for each agent with $g^i_1 \sim |N(0, \sigma_1)|$, $g^i_2 \sim N(0, \sigma_2)$, $n_i \sim N(0, n_0)$

and $\epsilon_t \sim N(0,1)$, where ϵ_t represents noise. Note that a trader for whom $g_1^i = g_2^i = 0$ is a noise trader.

The quantity $\bar{r}_{L_i}$ is the spot return averaged over the time interval L_i. The reason that L has a subscript i is because L is uniformly and independently distributed across agents over the interval $(1, L_{max})$ and L_i indicates the particular value randomly assigned to agent i. The computational formula for $\bar{r}_{L_i}$ is

$$\bar{r}_{L_i} = \frac{1}{L_i} \sum_{j=1}^{L_i} \frac{p_{t-j} - p_{t-j-1}}{p_{t-j-1}}$$

Hence the future asset price expected at time $t+\tau$ by agent i is given by

$$\hat{p}^i_{t+\tau} = p_t \cdot e^{\hat{r}^i_{t,t+\tau}\tau}$$

If the agent expects a price increase (decrease), he decides to buy (sell) one unit of the stock at a price b_t^i (a_t^i) that is lower (higher) than his expected future price $\hat{p}^i_{t+\tau}$. More precisely, b_t^i and a_t^i are given as

$$\begin{aligned} b_t^i &= \hat{p}^i_{t+\tau}(1-k^i) \\ a_t^i &= \hat{p}^i_{t+\tau}(1+k^i) \end{aligned}$$

where k^i is uniformly drawn from the interval $(0, k_{max})$ with $0 < k_{max} \leq 1$.

Let b_t^q and a_t^q denote the highest bid price and the lowest ask price, respectively, quoted in the limit order book at time t. Now the trading action taken by the agent is determined as follows. If b_t^i (a_t^i) is smaller (larger) than a_t^q (b_t^q), the agent submits a limit buy (sell) order with a limit price of b_t^i (a_t^i). Otherwise, if b_t^i (a_t^i) is larger (smaller) than a_t^q (b_t^q), the trader submits a market order and the trade is executed at the price of a_t^q (b_t^q). At the end of the period τ unmatched limit orders are removed from the limit order book (recall that every limit order has a finite time τ).

Given the above model that generates the dynamics of a double-auction market, Chiarella and Iori focuses on studying, through numerical simulations, how the dynamics of price and spread are affected by liquidity (λ), tick size (Δ) and the average life of an order (τ). The results of their simulations are not reviewed here. Interested readers can refer to the original paper (Chiarella and Iori 2002) for the details of their results and conclusions.

One attractive feature of the Chiarella-Iori model is that it has a very creative, clean and simple way to model the phenomena that fundamentalist, chartist and noise components often co-exist in real markets. The model also introduces a rich set of parameters, which gives the flexibility needed to allow modelers to gain insights into market microstructure issues in a variety of settings. One major drawback of the Chiarella-Iori model comes from the assumption that there exists a constant fundamental value of the asset that all agents know. This assumption is troublesome because (1) it is not realistic and (2) it eliminates the possibility for studying the influence of informational issues on order flow dynamics.

2.2 The Das model

Das (2003) proposes and evaluates a market-making algorithm in the context of an artificial double-auction market that has an asymmetric informational structure. Das' model is unique because, unlike other agent-based markets, which generally produce the underlying market dynamics through the interactions of traders, his model centers around the role of a market maker and generates the market dynamics through the interaction between the traders and the market maker. Also the agent that represents the market marker is highly sophisticated and equipped with machine learning techniques that allow it to induce price information from the activities of traders. The setting of the model is as follows.

The market modeled in Das' paper is a discrete-time dealer-type double-auction market (such as the NYSE) with only one stock. The dealer or market maker sets bid and ask prices (P_b^i and P_a^i respectively) at which he is willing to buy or sell one unit of the stock at time period i. All transactions occur with the market-maker taking one side of the trade and a trader taking the other side of the trade.

The stock has a true underlying value (or fundamental value) V^i at time period i. The market marker is informed of V^0 at the beginning of a simulation, but does not receive any direct information about V thereafter. That is, the only signals the market marker receives about the true value of the stock are through the buy and sell orders placed by the traders. At time period i, a single trader is selected and allowed to place either a (market) buy or (market) sell order for one unit of the stock. There are two types of traders in the market, uninformed traders and informed traders. An uniformed trader will place a buy or sell order for one unit at random if selected to trade. An informed trader knows V^i and will place a buy order if $V^i > P_a^i$, a sell order if $V^i < P_b^i$ and no order if $P_b^i \leq V^i \leq P_a^i$.

The true underlying value of the stock evolves according to a jump process. At time $i+1$, with probability p (p is typically small, of the order of 1 in 1000 in most of Das' simulations), a jump in the true value occurs: $V^{i+1} = V^i + \tilde{\omega}(0, \sigma)$ where $\tilde{\omega}(0, \sigma)$ represents a sample from a normal distribution with mean 0 and variance σ^2. Modeling the true value as a jump process reflects the belief that the true value evolves as a result of occasional news arrivals. When a jump occurs, the informed traders are in an advantageous position. The periods immediately following the jumps are the periods in which informed traders can trade most profitably, because the information they have on the true value has not been disseminated to the market yet, and the market marker is not informed of changes in the true value. Thus it is important for the market maker to track the true value persistently through the orders placed by the traders. The price dynamics in Das' model is entirely caused by the market maker's updates of his posted bid and ask prices. The following paragraph briefly describes the market-making algorithm.

The market maker sets his bid and ask prices based on the following formulas when he receives a market sell order (a signal for him to update his bid price) or market buy order (a signal for him to update his ask price).

$$\begin{aligned} P_b &= E[V|Sell] \\ &= \int_o^\infty x Pr(V = x|Sell) dx \\ P_a &= E[V|Buy] \\ &= \int_o^\infty x Pr(V = x|Buy) dx \end{aligned}$$

The above equations are solved approximately by discretizing the price into intervals, with each interval representing one cent. For example, the formula for P_b becomes

$$P_b = \sum_{V_i=V_{min}}^{V_{max}} V_i Pr(V = V_i|Sell)$$

Applying Bayes' rule and simplifying, we get

$$P_b = \sum_{V_i=V_{min}}^{V_{max}} \frac{V_i Pr(Sell|V = V_i) Pr(V = V_i)}{Pr(Sell)}$$

Note that, since the prior probability $Pr(Sell) = 1/2$ and $V_{min} < P_b < V_{max}$, we have

$$\begin{aligned} P_b &= 2 \sum_{V_i=V_{min}}^{V_{max}} V_i Pr(Sell|V = V_i) Pr(V = V_i) \\ &= 2 \sum_{V_i=V_{min}}^{P_b} V_i Pr(Sell|V = V_i) Pr(V = V_i) + 2 \sum_{V_i=P_b+1}^{V_{max}} V_i Pr(Sell|V = V_i) Pr(V = V_i) \end{aligned}$$

By splitting P_b into the sum of two terms as above, we can calculate the appropriate value for $Pr(Sell|V = V_i)$ in each of the terms. An uninformed trader is equally likely to sell whatever the market maker's bid price. On the other hand, an informed trader will sell if $V \leq P_b$ and not if $V > P_b$. Suppose the proportion of informed traders in the trading crowd is α, then

$$\begin{aligned} Pr(Sell|V \leq P_b) &= \frac{1}{2} \cdot (1 - \alpha) + 1 \cdot \alpha \\ &= (1 + \alpha)/2 \\ Pr(Sell|V > P_b) &= \frac{1}{2} \cdot (1 - \alpha) + 0 \cdot \alpha \\ &= (1 - \alpha)/2 \end{aligned}$$

Hence the computational formula for P_b becomes

$$P_b \;=\; (1+\alpha)\sum_{V_i=V_{min}}^{P_b} V_i Pr(V=V_i) + (1-\alpha)\sum_{V_i=P_b+1}^{V_{max}} V_i Pr(V=V_i)$$

In order to use the above equation to set the value of P_b, the market marker needs to estimate the quantity $Pr(V=V_i)$ for various V_is. To do so, he maintains an online estimate of the probability density function (PDF) of the true value as follows. At the beginning of the simulation, he sets the PDF to be a normal distribution[1] centered at V_0 (the initial true value of the stock). During the simulation, each time that the market maker receives a signal about the true value by receiving a market buy (sell) order, he updates the PDF by setting $P(V=V_i)$ to the posterior probability $P(V=V_i|Buy)$ ($P(V=V_i|Sell)$) for each V_i. The posterior probability can be easily computed using Bayes' rule. For example, if a market sell order is received,

$$Pr(V=V_i|Sell) = \frac{Pr(Sell|V=V_i)Pr(V=V_i)}{Pr(Sell)}$$

where the prior probability $Pr(V=V_i)$ is known from the current density estimate, the prior probability of a sell order $Pr(Sell)=1/2$ and $Pr(Sell|V=V_i)$ is given by $Pr(Sell|V=V_i, V_i \leq P_b) = (1+\alpha)/2$ and $Pr(Sell|V=V_i, V_i > P_b) = (1-\alpha)/2$ as described earlier.

This concludes the basic algorithm used by the market-making agent in Das' artificial market model. Note that the computational formula for updating P_a can be derived using a similar argument. Please refer to Das (2003) for many extensions to the basic strategy described above, such as the inclusion of noisy informed traders, inventory control constraints, etc.

[1]The distribution is actually a discrete approximation of the normal distribution with probability mass from V_{min} to V_{max}. The distribution is also maintained in a normalized state at all times whenever it is updated. See Das (2003) for details about constructing the initial distribution estimate.

2.3 The Smith-Farmer-Gillemot-Krishnamurthy (SFGK) model

Strictly speaking, the SFGK artificial double-auction market is more of a statistical random model than an agent-based one. There is no explicit use of artificial agents in their original simulations. However the research initiative and the underlying order-generating mechanism both fit very well in the context of agent-based modeling. The most amazing feature of the SFGK model is that, although it is very simple (completely random and zero-intelligence), it has the power to simulate the evolution of the seemingly complex structure of double-auction markets. The SFGK model shares the spirit of the zero-intelligence model proposed by Gode and Sunder (1993) who made the famous claim that the "Allocative efficiency of a double-auction market derives largely from its structure, independent of traders' motivation, intelligence, or learning. Adam Smith's invisible hand may be more powerful than some may have thought; it can generate aggregate rationality not only from individual rationality but also from individual irrationality."

The SFGK artificial market is central to much of the research work carried out in this thesis (see Figure 1.1). We choose to implement the SFGK model in an agent-based setting on top of our simulation platform so that we are able to examine the quality of the simulation software. We also study the microstructure properties of the model through extensive simulation experiments and use it as an underlying testbed for evaluating trading strategies. Section 4.2 describes the SFGK model in detail.

Chapter 3

Building the agent-based simulation platform

Agent-based modeling in the financial context often requires researchers to construct a robust and reliable software system, on which they can implement particular types of market models and artificial agents, run the simulation and record statistical results for future analysis. Therefore a software simulation platform is a fundamental and important component to agent-based modeling that cannot be ignored. However building a high-quality software platform with desirable properties such as scalability, high-performance and accuracy is often a difficult and challenging task for many researchers in finance who have no or little computer science background. In this chapter, I will present our software platform that is specifically designed for agent-based modeling and simulation of double-auction markets (CDAM). In creating such a software system, we hope to be able to bridge or narrow the gap between computer science and finance by providing researchers interested in agent-based financial modeling with a ready-to-use simulation environment so that they can focus more on other issues such as designing market and agent models, analyzing simulation results, etc. I will also share some insights and experience that I learned through developing such a software system, with those who may be interested in building their own agent-based simulation environment.

3.1 Design and structure overview

Our simulation platform is a distributed (client/server architecture) and multi-threaded computing environment that is capable of simulating a variety of financial markets for different research purposes. Central to the simulation platform is an artificial market (implemented as the server) which takes requests from market participants (implemented as clients) and processes them according to rules that are consistent with the type of the artificial market. The artificial market currently implemented on the simulation platform is a continuous double-auction market. Market participants can either be real people or artificial agents with varying levels of intelligence.

3.1.1 General properties of the software platform

Throughout the development process, we consistently strived to develop a software platform with the following properties.

1. *Scalability* is a common term used in the software engineering literature and is defined as "the ease with which a system or component can be modified to fit the problem area" (Software Engineering Institute, Carnegie Mellon University 2003). Scalability is a desirable property because it allows a researcher to change his or her underlying market or agent model without having to recode much of the current simulation platform. For example, it is often the case that for a given market model, a researcher may want to experiment with different types of artificial agents. With a scalable simulation platform, the researcher can achieve that by simply replacing the current agent class (in which agent logic is coded) with a new one, leaving the rest intact.

2. The server must achieve *high performance.* In other words, we want the server (market) to be able to process incoming requests and generate responses at a very fast pace. High-performance simulation software should be able to map the *real world times* to significantly smaller ones in the *simulated world* so that a researcher who wants to simulate a 50-year-long time series of the price of some stock doesn't have to actually wait for 50 years. In the context of agent-based simulation, software performance can be significantly improved by choosing efficient data structures and reducing unnecessary communications between threads.

And of course, most importantly, the system must also be *correct.* We want the software platform to produce results that accurately reflect the underlying model. Accuracy is especially critical for large-scale and fast-paced simulations because even a tiny error may cause an accumulated and compounded impact which over time may entirely distort the property of the underlying model. A systematic testing strategy will help to ensure correctness.

In the following two subsections, I will provide an overview of the structure and working mechanism of our simulation platform. The overview provided below is from an high-level perspective and it deliberately omits some lower-level details in order not to distract a reader's attendion from the big picture.

3.1.2 Server side structure and working mechanism

Figure 3.1 illustrates the high-level structure of the server, from which we can identify several key components.

After the server is started, the main process will listen and monitor the main socket port for incoming agent connections. Everytime an agent sends a connection request, the server spawns a pair of threads (read thread and write thread) that will be dedicated to serve the agent throughtout its connection lifetime. The read thread is mainly responsible for fetching the agent's requests from the peer socket port, pre-processing the request

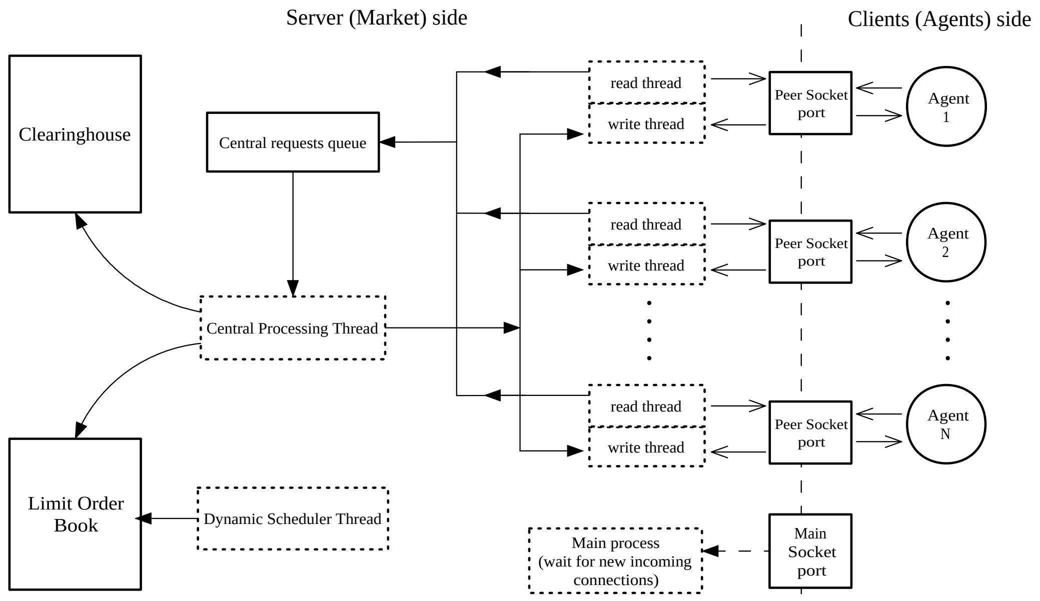


Figure 3.1: High-level server structure diagram. Shapes with solid borders on the server side represents essential data structures residing in computer memory and shapes with dashed borders represents major processes/threads running on the server. A link between a thread and a data structure (represented by a thick arrow) suggests that the thread can access and modify the content of the data structure.

and pushing the pre-processed request into the central queue for execution. At the very beginning, it also authenticates the agent and synchronizes the agent's trading account with the server's clearinghouse. The write thread is responsible for writing responses (from the server) to the agent. Responses are typically generated after an agent's requests get executed. Both the read and write threads will terminate when an agent leaves the market and disconnects.

The central processing thread is the *brain* of the server. It determines how agents' requests are to be treated and it has to make sure that the executions of those requests are 100% consistent with the underlying market model. In our case, since we have chosen the market model to be of double-auction type, the central processing thread is responsible for maintaining a limit order book and properly handling order matchings.

The last key thread on the server side is the *dynamic scheduler thread*. It, as its name suggests, performs a job-scheduling task and the job in our case is to maintain a priority queue which stores the expiration times associated with limit orders and removes limit orders from the limit order book as they expire. The prefix *dynamic* alludes to this thread's need to constantly re-balance the priority queue in real time as random orders with random expiration times arrive at the market.

The two major data structures on the server side are the *Limit order book* and the *Clearinghouse*. The limit order book is central to a double-auction market: it stores both limit buy and limit sell orders. An augmented *heap data structure* was used to implement the limit order book (actually there are two such heaps, one for limit sell orders and one for limit buy orders). A heap is the most efficient choice of data structure for implementing a limit order book as most trading operations can be completed with either $\mathcal{O}(1)$ time (such as querying the current best bid and ask price) or $\mathcal{O}(\log n)$ time (such as executing market orders, adding new limit orders and deleting expired limit orders). The term clearinghouse in the investment literature commonly refers to a financial organization, typically associated with one or more exchanges, that matches the buy and sell orders that are submitted during the day and keeps track of the obligations and payments required of the members of the clearinghouse. In the context of our simulation software, it is a dictionary type data structure that stores up-to-date account information (such as cash and asset balance) for each registered client. The central processing thread will modify the clearinghouse (update the accounts of relevant parties) each time a transaction occurs.

Notice in Figure 3.1 that the central processing thread and the dynamic scheduler thread can both access and modify the limit order book. This creates a problem that occurs frequently in multithreaded (or multiprocessed) programing and is referred as the concurrency control problem. Concurrency issues must be handled carefully because they could not only introduce significant errors to the simulation results but also create tremendous difficulties for programmers to debug the system.

3.1.3 Client side structure and working mechanism

Figure 3.2 identifies the key threads and data structures that are associated with an artificial agent. An important feature of the agent structure is that a functional agent can be viewed as two separate and interacting parts: the “physical body part” and the “head (or brain) part”. Such a high-level design is consistent with our goal of achieving software scalability, since it provides great ease and flexibility for building artificial agents with completely different logic and levels of intelligence. The working mechanism of the artificial agent is analogous to the two-way interaction between a real human being’s brain and body: the brain part is responsible for making logical decisions and instructing the body to act based on its decisions. The human body also provides feedback to the brain about its dealing with the outside world. In our simulation framework, an artificial agent is very much like a plug-and-play device: to experiment with a new agent, one simply needs to code the agent’s logic into the brain part, plug the brain part into the body and start the simulation.

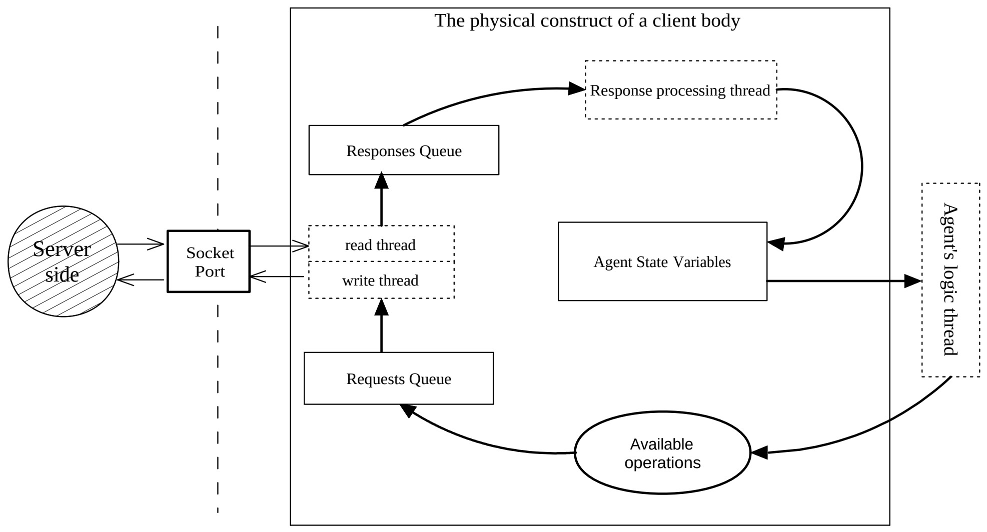


Figure 3.2: High-level structure diagram for an artificial client. Rectangles with dashed borders represent threads and those with solid borders represents data structures. Black arrows linking a thread to a data structure indicate that the thread can access or modify the content of the data structure. An oval represents a set of available operations for the agent to interact with the market.

An agent can also be a human being. The client side structure for a human agent is almost identical to that of an artificial one as they both share the same "physical body". The only difference between them lies in the "head" part: a human agent can actively take trading actions through a graphical user interface; whereas an artificial agent is controlled by some artificial thread into which the agent's logic is pre-programmed. Figure 3.3 illustrates the software structure of a human trader and Figure 3.4 gives a snapshot of the GUI through which a human trader interacts with the market.

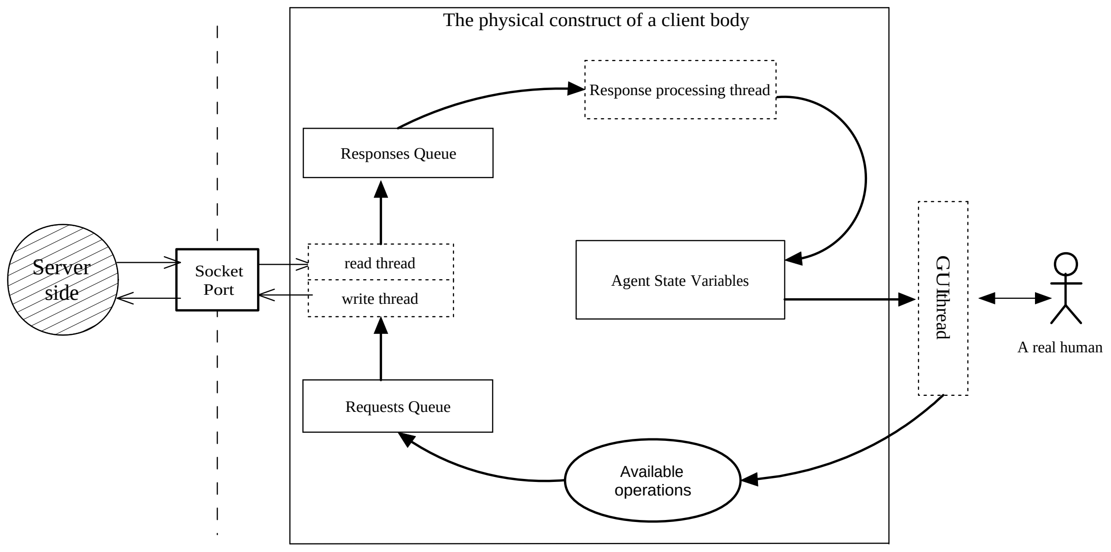
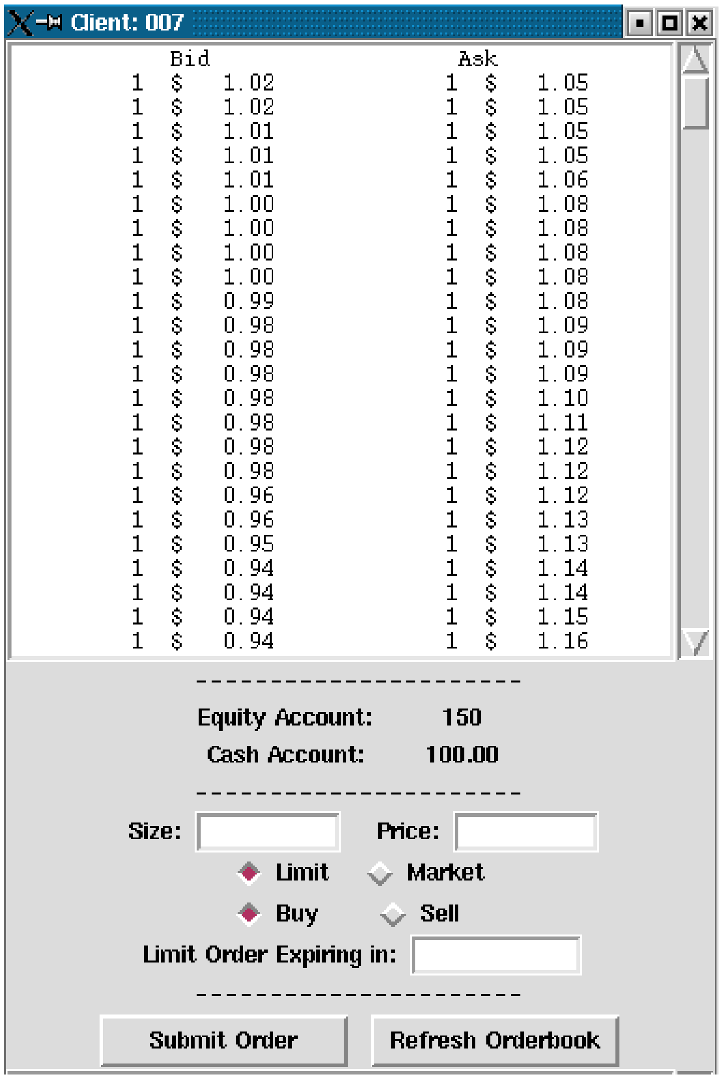


Figure 3.3: High-level structure diagram for a human client. Rectangles with dashed borders represent threads and those with solid borders represents data structures. Black arrows linking a thread to a data structure indicate that the thread can access or modify the content of the data structure. An oval represents a set of available operations for the agent to interact with the market. Note the similarity between this diagram and the one for the artificial client: both clients share the same “physical body”.



```
Client: 007

          Bid                        Ask
    1  $   1.02               1  $   1.05
    1  $   1.02               1  $   1.05
    1  $   1.01               1  $   1.05
    1  $   1.01               1  $   1.05
    1  $   1.01               1  $   1.06
    1  $   1.00               1  $   1.08
    1  $   1.00               1  $   1.08
    1  $   1.00               1  $   1.08
    1  $   1.00               1  $   1.08
    1  $   0.99               1  $   1.08
    1  $   0.98               1  $   1.09
    1  $   0.98               1  $   1.09
    1  $   0.98               1  $   1.09
    1  $   0.98               1  $   1.10
    1  $   0.98               1  $   1.11
    1  $   0.98               1  $   1.12
    1  $   0.98               1  $   1.12
    1  $   0.96               1  $   1.12
    1  $   0.96               1  $   1.13
    1  $   0.95               1  $   1.13
    1  $   0.94               1  $   1.14
    1  $   0.94               1  $   1.14
    1  $   0.94               1  $   1.15
    1  $   0.94               1  $   1.16

      ----------------------
      Equity Account:     150
       Cash Account:    100.00
      ----------------------
```

Figure 3.4: The graphical user interface (GUI) through which a human trader interacts with the market.

3.2 Issues and our advice

This section addresses some important issues associated with building an agent-based simulation platform in the hope that the insights provided in this section will be helpful to those who are interested in building such software systems of their own.

3.2.1 Choice of programming language

There is no easy answer to the question of what programming language is best for implementing an agent-based simulation system. However, there are certain important considerations that a person has to take into account before making his selection of a particular programming language.

System complexity is one consideration. If the desired simulation system involves complex structure and multiple processes, an object-oriented language (such as C++, Java and Python) is likely to be the only feasible choice. A high-complexity simulation system also suggests that one choose a language that has a relatively rich library providing supports for data structures, operating system services, etc, so that the developer does not have the burden of programming everything from scratch. In terms of library support, Java and Python are more desirable than the standard C++ language.

Performance is another consideration. Depending on the nature and purpose of the artificial market model, certain agent-based simulations can be very computationally intensive. For example, of the three artificial market models surveyed in Chapter 2, the Das model has a much higher demand for high-performance computing than the Chiarella-Iori and SFGK models. In general, assuming everything else is equal, a fully compiled language (such as C++) can result in a faster and more efficient simulation platform than a "half-way compiled" language (such as Java) and interpreted languages (such as Python).

The last main consideration is *portability*. If one wants to allow the simulation software to be able to run on different operating system platforms (such as Linux, Unix and Windows), one should choose a platform-independent language such as Java and Python. Based on our experience, agent-based simulation software often needs to use low-level system calls (such as those that deal with socket communication and threads) that are platform-dependent. Thus, if one implements a simulation software with C++ on Linux and later decides to migrate it to Windows, one will very likely have to undergo a considerable revision of the code.

Our agent-based simulation platform is implemented using the Python[1] programming language (version 2.3.4). We are very pleased with what Python has offered us throughout the development process. In particular, Python's extremely elegant and clean syntax (comparing to C++ or even Java) allows us to write code faster and with less bugs. For this particular research project, Python is a good language choice that satisfies all the considerations mentioned above.

3.2.2 Distributed computing

In general, an agent-based simulation platform should provide a distributed computing environment with a client/server type of architecture that mimics the exchange/trader setting of real financial markets. However, programming a distributed system can be a daunting task, especially for those with little computer science background. In what follows, we will provide a few insights into some common issues associated with building a distributed simulation platform.

The first issue concerns socket programming. In a client/server architecture, clients and the server communicate through sockets. Most high-level programming languages have library support for socket programming and they provide programmers with a pair of seemingly straightforward functions: "send()" and "recv()". An inexperienced programmer may think that he could simply place a call to the two functions whenever

[1]Details about the language can be found at http://www.python.org

he wants to send or receive a message and those functions would take care of the rest. Unfortunately, it is not that simple. The “send()” function returns the number of bytes actually sent out, which might be less than the length of the outgoing message. Hence a programmer must *always* check whether the value returned by “send” matches the length of the string he wants to send out. If it doesn’t, the programmer is responsible for repeatedly calling “send()” until the remaining bytes are also sent out. Similarly, the “recv()” function returns the number of bytes actually received from the socket stream, which could be less than the number in a message that a programmer is expecting. What’s worse is that the bytes returned by “recv()” might contain fractions of multiple messages. Based on our experience, a program should do the following two things to ensure the integrity of the incoming messages. First, all messages should have a consistent and well-defined format.[2] They can either be of fixed-length or delimited. Second, the program must write its *own buffer function* to accumulate and process data returned from “recv()”. It is very important for programmers to follow the above procedures when they use “send()” and “recv()”. Being careless about the subtle deficiencies of the two functions can cause the simulation program to crash in a random and bizarre fashion. Also it can be extremely frustrating and difficult to debug errors resulting from message disintegrity.

The second issue concerns multithreaded programming. As soon as one starts to program the server part of a distributed computing system, one is likely to face a tough design question: what mechanism the will serve adopt to handle multiple clients? One has the following two choices.

1. The event-driven multiplexing approach: the server has a single process only which serves all clients. The basic idea behind this approach is that the single process

[2]The message format we used in our simulation platform is rather simple. Every message contains a fixed number of fields which are delimited by the special character “|” and is terminated by “\r\n”. For more serious applications, we recommend the FIX (Financial Information eXchange) protocol, which is a messaging standard developed specifically for the real-time electronic exchange of securities transactions. Details of the FIX protocol can be found at http://www.fixprotocol.org.

continuously monitors all connected clients until some clients trigger an event (such as an incoming message), at which point the process starts to serve those clients one by one before going back to the monitoring stage.

2. The multithreaded approach: the server has multiple processes (threads), each of which is dedicated to a particular client. The basic idea behind this approach is that the server has a main process which continuously listens for new connections and once a client makes a connection, the main process spawns a new thread to deal with that client for as long as the client stays connected.

Both approaches have pros and cons. The multiplexing approach is relatively efficient and simple, since it involves only one process and a programmer doesn't need to be concerned about the overhead associated with multithreaded programming. However, we find that it has limited ability to handle more complex distributed system, especially those that require state-dependent and persistent two way communications. On the other hand, the multithreaded approach is more powerful and more flexible. The downside of it is that multithreaded programming can be very complicated and a programmer must handle the often tricky tasks of thread synchronization and concurrency control. From our experience, if the agent-based simulation is sophisticated or the simulation platform being built needs to be scalable (to allow further extensions), the multithreaded approach is very likely the only way to go.

Chapter 4

Experiments with the SFGK model

There are two main components in agent-based financial modeling: a computational environment for simulation and an artificial model that drives the simulation. Having successfully constructed a software simulation platform (or the first component), the next step is to choose a particular artificial market model with which we can experiment on our platform. In this chapter, we justify our choice of adopting the SFGK artificial market and present the model and its implementation in detail.

4.1 Why do we choose the SFGK model?

As we mentioned earlier, there are many agent-based market models available that differ from one another in how "smart" their artificial agents are. The SFGK model is designed to be as analytically tractable as possible while capturing key features of a double-auction market. In terms of the intelligence level of the participating agents, the model only uses the so-called *zero intelligence* (ZI) agents, which are agents that do not possess any inductive learning abilities. That is, there are no machine learning algorithms embedded in the agents that would allow them to improve their performance with time by learning from experience. We have decided to implement the SFGK model on top of our simulation platform for the following four reasons.

First, the model presents a very simple and elegant way to mimic the microdynamics of a continuous double-auction market. Simulating a double-auction market is more complicated than just building price movement dynamics (which could be easily done by simple Monte Carlo simulations) because our interest in studying the microstructure of double-auction markets is really centered around the evolution of the *limit order book* (not just a one dimensional price). However SFGK achieved this seemingly complicated task of modeling the dynamics of a double-auction market by essentially using and combining only a series of *Poisson* processes. The simplicity of the model provides several benefits. First since it doesn't involve any learning procedures for agents, it considerably reduces the computational complexity required to generate the market dynamics. If agents were too *intelligent* (i.e., they were made to participate in market activities according to sophisticated machine learning algorithms), it would often result in very large-scale and time-consuming real-time computations. Second, because of the simplicity of the model, many time-series properties for this artificial market are analytically tractable, which allows us to make certain predictions about quantities such as the stationary mean and the volatility of the mid price.

Second, as we mentioned earlier in Chapter 3, large scale and multithreaded simulation software requires extensive and rigorous testings to ensure its accuracy. By implementing and running the SFGK model on our simulation platform, we are able to debug and stress-test (i.e., test the server when it's being stretched to its processing capacity) the platform in a comprehensive and sustainable fashion. The SFGK model is better in the testing context than some other more complex models because its core logical operations directly interact with the software platform (whereas, for some other more complex models, there may be several additional layers of code sitting between the platform and the models' high-level logical operations) and thus it is relatively easier to identify the source of an error should it occurred.

Third, we are interested in studying statistical properties associated with the SFGK artificial double-auction market. In particular, we are interested in the evolution of the market mid-price, bid-ask spread and order book structure. The detailed statistical analysis and results are presented in Chapter 5.

Fourth and last, we are also interested in exploring some simple liquidation strategies in the context of double-auction markets by designing special agents (each coded with a different trading strategy) and observing their liquidation results. However, before we can experiment with those special agents, we first must have active market dynamics that mimics a double-auction market. Thus, when we take the agent-based approach to study optimal trading strategies, the SFGK model serves the purpose of proving a underlying market environment. Trading strategy experiments (for the time-constrained asset liquidation problem) are carried out in Chapter 6.

4.2 Market mechanism of the SFGK model

4.2.1 The original version of SFGK model

The original SFGK model proposed by Smith, Farmer, Gillemot, and Krishnamurthy (2003) is described in detail below.

All order flows to the market (including limit orders and market orders) are modeled as Poisson processes. Market orders arrive at the market in chunks of σ shares where σ is a fixed integer, at an average rate of μ shares per unit time. A market order may either be a buy market order or a sell market order with equal probability, which means that the Poisson process followed by market buy orders or sell orders individually has an average arrival rate of $\mu/2$.

Limit orders arrive at the market in chunks of σ shares, at an average rate of α shares *per unit price per unit time*. A limit order may either be a limit buy order or a limit sell order with equal probability, which implies that limit buy orders or sell orders individually

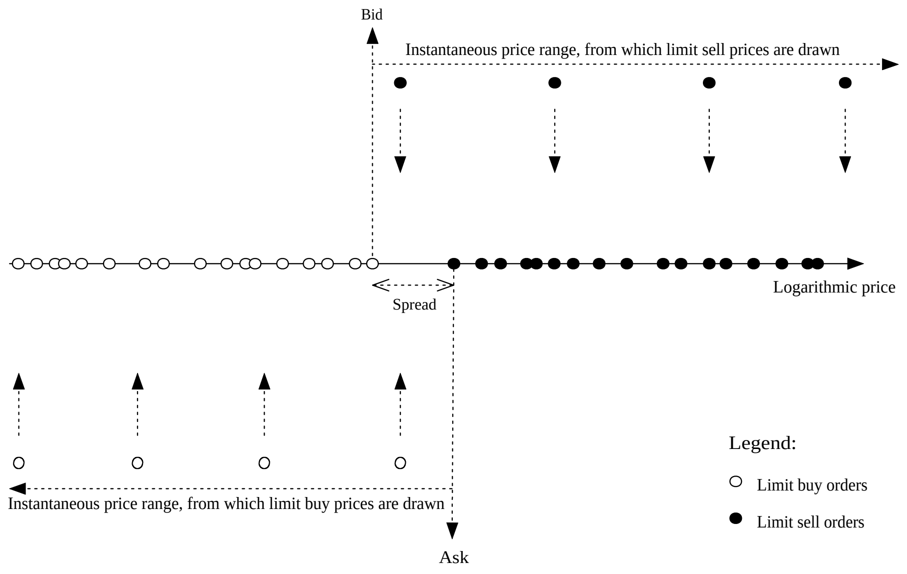

Bid
Instantaneous price range, from which limit sell prices are drawn
Spread
Logarithmic price
Legend:
Limit buy orders
Limit sell orders
Instantaneous price range, from which limit buy prices are drawn
Ask

Figure 4.1: Limit order generating mechanism

have an average Poisson arrival rate of $\alpha/2$ shares *per unit price per unit time.*[1]. Recall that in addition to order size, a limit order must also specify a limit price. The *logarithm* of limit prices are chosen randomly from uniform distributions. Specifically, the *log* of limit buy prices are drawn from a uniform distribution on the price interval of $(-\infty, a(t))$, where a(t) denotes the best (lowest) ask price in the double-auction market at time t. Similarly the log of limit sell prices are drawn from a uniform distribution on the price interval of $(b(t), \infty)$, where b(t) denotes the best (highest) bid price in the market at time t. Note that in their original model, prices are not continuous, but rather have discrete quanta called *ticks* (represented by *dp*). The limit order generating mechanism is illustrated in Figure 4.1.

Note that since all limit prices are generated in the logarithmic space in the SFGK model, before limit orders are submitted to the market, the limit prices associated with them must be first transformed to their corresponding physical prices. The use of loga-

[1]In their original paper, SFKG used α as the arrival rate for both limit buy orders and limit sell orders. Thus the total limit order arrival rate in their paper is actually $2 \cdot \alpha$

Parameter	Description	Dimension
α	avg. limit order rate	share/price·time
μ	avg. market order rate	share/time
δ	avg. limit order decay rate	1/time
σ	order size	share
dp	tick size	price

Table 4.1: Summary of parameters: SFGK model

rithmic prices has the advantage that it ensures the prices seen by the market are always positive.

When a market order (either a buy or sell) arrives at the market, it results in an immediate transaction: a buy market order removes the current best (lowest) ask from the limit order book (note that the trade is executed at price $a(t)$) and moves the best ask price up from $a(t)$ to the next occupied price tick (which is higher than $a(t)$). Similarly, a sell market order removes the current best (highest) bid from the limit order book (note that the trade is executed at price $b(t)$) and moves down the current best bid price down to the next occupied price tick (which is lower than $b(t)$). Please note that the above market order effect is only true when we make the assumption that all orders are of the same size (in this case σ) otherwise a single market order may not even have any effect on the market's current bid or ask price.

SFGK also allow limit orders to expire (or be cancelled) after being placed in the market so that limit orders can be removed spontaneously from the limit order book without a transaction having taken place. Limit order expiration (or cancellation) is also modeled by a Poisson process, with an average decay rate of δ per unit time.

Table 4.1 summarizes all the relevant parameters of the SFGK model.

The relatively simple design of the SFGK model allows us to make certain analytical predictions about some properties of the simulated double-auction market. For example, we can claim that the projected mean spread (denoted by $\bar{s}$) should be

$$\bar{s} = \frac{\mu + 2\delta}{\alpha}$$

The reasoning is as follows. Inside the spread, limit order removals are either caused by arriving market orders or expirations. Hence the total removal rate within the spread is equal to $\mu + 2\delta$ shares per second. 2δ is the combined expiration rate for limit orders currently at the best bid or ask (i.e., the two end points of the spread). On the other hand, limit order arrivals fill the spread at a rate of $\bar{s} \cdot \alpha$ shares per second. Therefore in order for $\bar{s}$ to be the stationary mean spread, the two *opposing forces* must balance, which implies

$$\mu + 2\delta = \bar{s} \cdot \alpha \iff \bar{s} = \frac{\mu + 2\delta}{\alpha}$$

Note that when δ is close to zero (i.e., when limit order removal by expiration is dominated by market order execution), $\bar{s}$ is approximately μ/α.

4.2.2 The practical version of the SFGK model

The actual artificial market model implemented on our simulation platform is the same as the original SFGK theoretical model except for the following.

- The tick size or the parameter *dp* is almost 0. In other words, the price in our simulation is nearly continuous. The near-zero tick size is a result of using high-precision floating point numbers to represent price.
- The order size (or σ) is set to 1 in our simulation. In other words, all limit and market orders arrive at the market in chunks of 1 share, at all times.

Also the price intervals, within which limit prices are drawn, are capped. Recall that SFGK's theoretical model requires that limit prices are generated from a uniform distribution on price interval $(-\infty, a(t))$ for limit buy orders and on price interval $(b(t), \infty)$ for limit sell orders. Clearly we have to deal with the notion of *infinity* in practical simulations as the computing environment is finite.[2] In order to avoid the infinity problem, we introduced a cap L (where L is some sufficiently large number) on the price intervals. Thus for our simulation, limit buy prices are uniformly drawn from the price interval $(a(t) - L, a(t))$ and limit sell prices are uniformly drawn from the price interval $(b(t), b(t) + L)$

4.3 Implementing the SFGK model

Technically speaking, one doesn't have to use a distributed environment to experiment with the SFGK model. In fact, the model is not even intrinsically related to agent-based simulations. A researcher with sufficient programming background probably could implement the model as a single-process program without introducing any agents. However as mentioned earlier, our goal isn't just to see what the model can produce. We are also interested in exploring optimal trading strategies and testing the quality of our simulation software, both of which would naturally require the model to be implemented in a distributed and agent-based environment. The role of agents in implementing the SFGK model is to generate order flows to the market and hence the central part of the implementation is to correctly program the *logic* of the participating agents so that, when they act together, the aggregate effect they create is consistent with what the underlying model requires. There are a few possible ways to implement the SFGK model using multiple agents that differ from one another only in terms of how trading tasks are assigned across all agents. For our implementation, we employ N (N is chosen to be 50 in our simulations) independent and homogeneous artificial agents, each of which can generate all four types of order flows (i.e., limit buys, limit sells, market buys and market sells) and submit them to the market. An alternative way to using homogeneous agents would

[2]More precisely, *infinity* poses two problems: first, we can not represent a infinitely large price on computers. Second, if the price interval is of infinite length, the limit order arrival rate (shares/second) will also be infinite, which will certainly crash the market server.

be, for example to separate agents into two groups and dedicate one group for generating limit orders and the other for generating market orders.

The implementation process consists of two main steps: parameter determination and logic codification. The following two subsections explain each of the two steps in more detail.

4.3.1 Parameter determination

There are two categories of parameters involved in the implementation process: the model-level parameters and the agent-level parameters. Model-level parameters are simply those parameters that are introduced in the SFGK model. As we mentioned earlier, for the practical implementation purpose, the relevant model-level parameters are:

$$\begin{aligned}\alpha &= \text{aggregate limit order rate}\\ \mu &= \text{aggregate market order rate}\\ \delta &= \text{limit order decay rate}\end{aligned}$$

Agent-level parameters are those that are responsible for controlling an agent's behavior during the simulation, such as how frequent the agent should generate orders, how much it should favor limit orders over market orders and so on. The values of agent-level parameters often depend on the desired model-level parameters and they can be derived after the model-level parameters have been set.

We first discuss the choices in selecting model-level parameters. Technically speaking, one is free to assign any arbitrary values to model-level parameters. However, in practice, we do need to impose certain constraints on these model-level parameters so that the simulation software can run stably and the simulation results are *reasonably realistic*.[3] For

[3] *Reasonably realistic* is a rather weak condition, by which we mean the simulated market only needs to resemble some general features of a double-auction market reasonably well. Calibrating model parameters to replicate real financial markets is a more difficult task, which we do not address in this thesis.

Model-level parameter	Parameter value
$\bar{s}$	0.015 (log price)
μ	1 share/second
δ	0.2 per second
$\alpha = (\mu + 2\delta)/\bar{s}$	93.33 shares/(second $\cdot$ log price)
$L = 75 \cdot \bar{s}$	1.125 (log price)

Table 4.2: The setting of model-level parameters

example, assigning arbitrarily large values for model parameters may cause the server's processing capacity limit to be exceeded and result in a crash of the simulation. Similarly if the limit order rate is too low (relative to the market order rate), we may frequently exhaust the liquidity in the simulated market[4], which is not desirable in most cases, in part because it can cause discontinuities in the time series produced by the simulation.

Besides the above rather obvious concerns, we may also decide to impose a constraint on the projected stationary spread (by simply fixing it at a certain level) so that we can have some control over the simulated market. As described in the earlier sections, fixing the stationary spread effectively reduces the degrees of freedom of the model from three to two. The setting of model-level parameters in our simulation is summarized in Table 4.2.

After having decided the values for model-level parameteres, we can now consider the agent-level parameters and their values. The set of agent-level parameters are summarized below.

- τ: mean time interval between successive order submissions.

[4]In the theoretical SFGK model, liquidity will never be exhausted because the price interval from which limit prices are drawn is infinite, which yields an infinite number of limit orders filling the order book at all times. However, the liquidity can be exhausted in the practical implementation of the SFGK model because the price interval is capped by a finite number L.

- p: probability that a limit order is generated for submission.
- N: number of participating artificial agents.

We first give the formulas for τ and p respectively and then verify that the setting of τ and p conforms to the required model-level parameters.
The formula for τ is

$$\tau = \frac{N}{\mu + \alpha \cdot L} \tag{4.1}$$

and the formula for p is

$$p = \frac{\alpha \cdot L}{\mu + \alpha \cdot L}$$

Verification:
For each individual agent, events (submitting an order) are generated by a Poisson process with average rate $1/\tau$. A generated event can either be a limit order submission with probability p or a market order submission with probability $1 - p$. Therefore the limit order submissions by a single agent follow a Poisson process with an average rate of p/τ and similarly the market order submissions by a single agent follow a Poisson process with an average rate of $(1 - p)/\tau$. Since all agents operate independently, the overall limit order arrivals seen by the market will be a Poisson process with an average arrival rate of $N \cdot p/\tau$ orders per second (i.e., it is the sum of the individual agents' limit order submission rates) and the overall market order arrivals seen by the market will also be a Poisson process with an average arrival rate of $N \cdot (1 - p)/\tau$ orders per second. Now, if we substitute the formulas of τ and p into the above quantities, we obtain the following:

- Aggregate limit order rate: $N \cdot p/\tau = N \cdot \frac{\alpha \cdot L}{\mu+\alpha \cdot L} \cdot \frac{\mu+\alpha \cdot L}{N} = \alpha \cdot L$
- Aggregate market order rate: $N \cdot (1 - p)/\tau = N \cdot \frac{\mu}{\mu+\alpha \cdot L} \cdot \frac{\mu+\alpha \cdot L}{N} = \mu$

The above result verifies that the formulas for τ and p are consistent with the required model-level parameters.

For the given model-level parameters in Table 4.2, the values of τ and p are

- $\tau = \frac{50}{1+93.33\times1.125} \approx 0.4717$ (seconds)

- $p = \frac{93.33\times1.125}{1+93.33\times1.125} \approx 0.99$

The agent-level parameter N (the number of participating agents) has been chosen to be 50 in our simulation. Although it seems that N is merely a scaling parameter that can not affect the intrinsic properties of the theoretical model, the value of N does need to be chosen with some care. In the actual implementation of the SFGK model (on a client-server type of architecture), before an agent can submit a limit buy (sell) order to the market, it must first query the market to get the current ask (bid) price (recall that limit price is drawn from a price interval that depends on the current bid or ask price). Although at first glance, the average quering time (the length of time that an agent has to wait before the query result arrives from the market) may seem pretty small (the average is about 0.004 second), it could still become a relatively significant delay when it is close to $1/\tau$, the average time interval between successive events generated by a particular agent. This is because, for a given set of model parameters and chosen N, an agent needs to be at a certain *hardworking* level, measured by how often it generates orders, which is in turn measured by $1/\tau$. When the quering time is close to $1/\tau$, it can no longer be neglected, because the average time interval between successive order submissions effectively becomes $1/\tau$ + the quering time. Hence each agent is working less hard than it should. Consequently, the simulation results coming from the above scenario will not be consistent with what the true model would produce. To overcome this problem, one must choose N to be sufficiently large so that $1/\tau$ is significantly larger than the quering time. Note from equation (4.1) that $1/\tau$ is proportional to N. Putting it simply, the more agents we have, the less hard each agent has to work. In our simulation, the choice of $N = 50$ yields $1/\tau = 0.4717$ second, which is significantly larger than the average quering time of 0.004 second.

4.3.2 Logic codification

After setting the agent-level parameters, we can proceed to code the *brain* or the *logic* of the agents. Since we use homogeneous agents, all agents share the same code. The pseudo code for the agent's logic is presented below.

```
while simulation not terminated:
   generate t from EXPONENTIAL(tau)
   sleep for t seconds
   generate independent U1, U2 from UNIFORM(0,1]
   if U1 <= p:  /* a limit order to be submitted */
      if U2 <= 0.5:  /* the limit order will be a buy order */
         query server for the current best ask a
         wait until query result is received
         generate limit price c from UNIFORM(a - L, a]
         generate expiration time w from EXPONENTIAL(1/delta)
         submit limit buy order [exp(c), w]
      else:  /* the limit order will be a sell order */
         query server for the current best bid b
         wait until query result is received
         generate limit price c from UNIFORM(b, b + L]
         generate expiration time w from EXPONENTIAL(1/delta)
         submit limit sell order [exp(c), w]
   else:  /* a market order to be submitted */
      if U2 <= 0.5:  /* the market order will be a buy order */
         submit market buy order
      else:  /* the market order will be a sell order */
         submit market sell order
end while
```

Figure 4.2: The pseudo code for an artificial agent

Chapter 5

Statistical results and analysis

Having properly implemented the SFKG double-auction market model on our distributed and agent-based platform, we are able to carry out thorough and large-scale simulations, from which we can obtain various relevant data in sufficient amounts. By performing certain statistical analyses on these data, we can investigate and explore many interesting properties of the artificial market. In particular, we want to understand the evolution of the market's mid-price, spread and order book structure.

5.1 The mid-price

Statistical analysis on mid-price is done in the logarithmic space. To avoid endless repetition, throughout this section, the word *mid-price* will actually mean the logarithm of the mid-price. Table 5.1 and Table 5.2 present two independent sample time series of the market's mid-price, each of which was produced by running the simulation for about 24 hours and each contains approximately $200,000$ data points. These tables allow readers to take a close look at the time series on different time scales.

We are interested in comparing the sample paths of the simulated market's mid-price to those of a Brownian motion process. A variable X, which follows a general Brownian

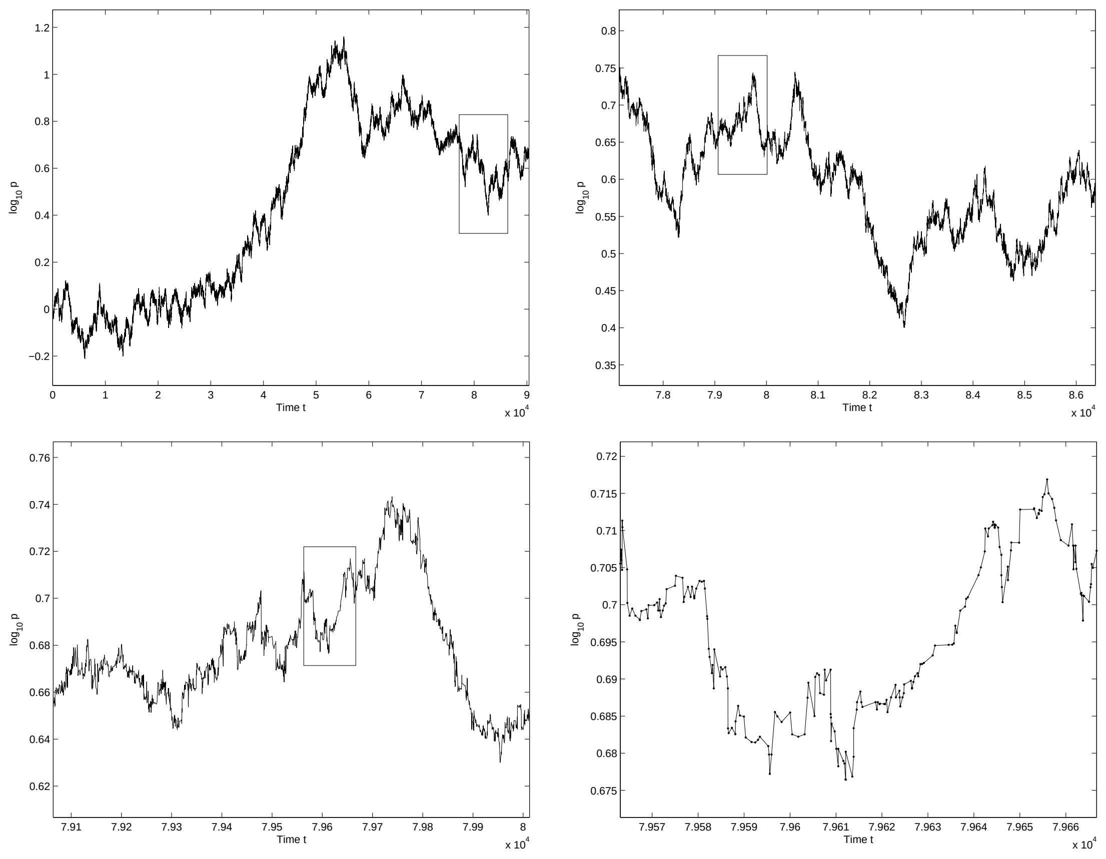


Figure 5.1: Sample time series #1 of mid-price with multi-level zooms. The north-west graph plots the complete time series and the box inside the graph indicates the portion to be zoomed in and shown on the next graph (i.e., the north-east one). Similarly the box inside the north-east graph indicates the portion to be zoomed in and shown on the next graph (i.e., the south-west one) and so on.

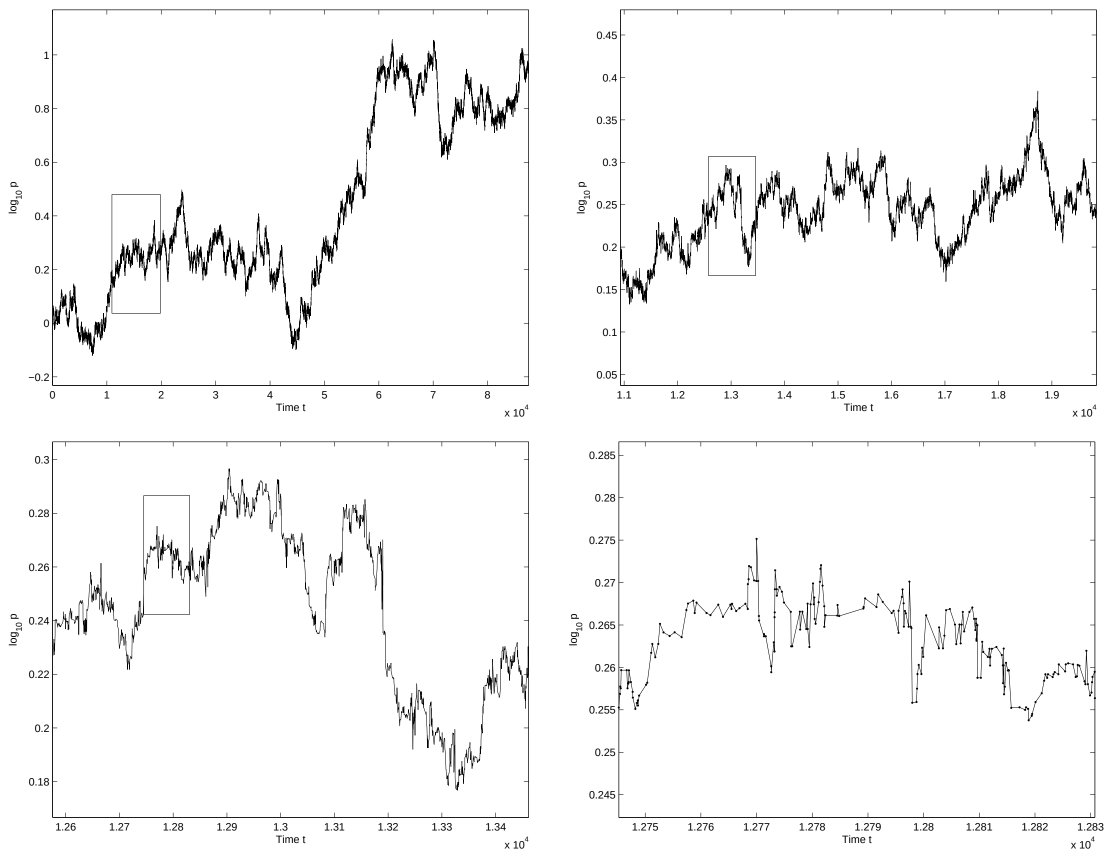


Figure 5.2: Sample time series #2 of mid-price with multi-level zooms. The north-west graph plots the complete time series and the box inside the graph indicates the portion to be zoomed in and shown on the next graph (i.e., the north-east one). Similarly the box inside the north-east graph indicates the portion to be zoomed in and shown on the next graph (i.e., the south-west one) and so on.

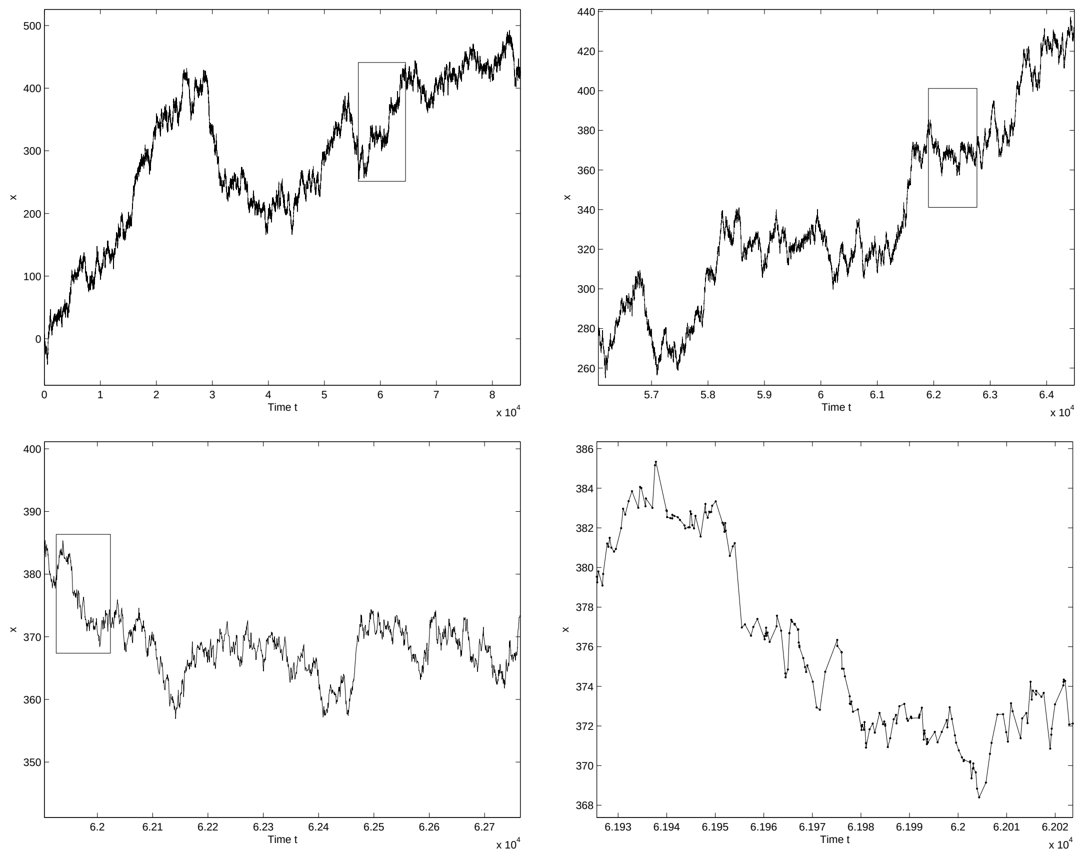


Figure 5.3: Sample time series of a standard Brownian motion with multi-level zooms. The north-west graph plots the complete time series and the box inside the graph indicates the portion to be zoomed in and shown on the next graph (i.e., the north-east one). Similarly the box inside the north-east graph indicates the portion to be zoomed in and shown on the next graph (i.e., the south-west one) and so on.

motion, can be described by the equation

$$X(t) = \mu \cdot t + \sigma \cdot W(t) \tag{5.1}$$

where W(t) denotes a standard one-dimensional Brownian motion. Table 5.3 shows a sample path (or a sample time series) of a pure Brownian motion, for which the drift coefficient μ is 0 and the diffusion coefficient σ is 1. This Brownian motion sample path is constructed by running a Monte Carlo simulation (Glasserman 2004) on random and discrete times ($t_0 < t_1 < ... < t_N$). The set up of Table 5.3 is entirely analogous to that of the Tables 5.1 and 5.2. Also the values of N and t_N are set to match their counterparts in the mid-price time series so that the graphs in all three tables are comparable.

By comparing the graphs in Tables 5.1, 5.2 and 5.3, we can see that the evolution of the mid-price of the simulated market does have certain similarities to a Brownian motion process. In particular, in Tables 5.1 and 5.2, we are able to roughly observe the well-known *statistically self-similar* property of a Brownian motion process , which intuitively means that all zoomed graphs of a Brownian motion process should look similar.

But how similar is the mid-price evolution to a pure Brownian motion process? If there are differences between them, what are they? Staring at the graphs of the sample time series won't help much to answer these questions. Instead, one effective way to find the difference or similarity between time-series for the mid-price (let's call it P) and for a variable X that follows a Brownian motion is to investigate the variance of the time series for P and X with varying time lags. More precisely, from the mid-price time series obtained from the simulation, we would like to measure the following quantity for different values of τ (which denotes the time lag between sampled data points)

$$\frac{1}{\tau} \cdot Var(P_{(j+1)\tau} - P_{j\tau})$$

where $P_{j\tau}$ denotes the value of P at time $j\tau$.

Suppose a variable X follows a pure Brownian motion process $X(t) = \sigma W(t)$. It is well known that

$$X_{(j+1)\tau} - X_{\tau} = \sigma \cdot \sqrt{\tau} \cdot \xi_j$$

where ξ_js are independent and identically distributed (IID) standard normal random variables. Let $E[\cdot]$ denote the mean value of $[\cdot]$ and N denote the total number of data points in the complete Brownian motion time series (hence N/τ is the sample size). Then we have

$$\begin{aligned} E[(X_{(j+1)\tau} - X_{j\tau})^2] &= \sigma^2 \cdot \tau \cdot E[{\xi_j}^2] \\ &= \sigma^2 \cdot \tau \cdot (1 \pm \mathcal{O}((N/\tau)^{-\frac{1}{2}})) \\ &= \sigma^2 \cdot \tau \cdot (1 \pm \mathcal{O}(\sqrt{\tau/N})) \end{aligned} \tag{5.2}$$

Since X has 0 drift (i.e. $E[(X_{(j+1)\tau} - X_{j\tau})] = 0$), we can rewrite equation 5.2 as

$$\begin{aligned} E[(X_{(j+1)\tau} - X_{j\tau})^2] &= Var(X_{(j+1)\tau} - X_{j\tau}) \\ &= \sigma^2 \cdot \tau \cdot (1 \pm \mathcal{O}(\sqrt{\tau/N})) \end{aligned} \tag{5.3}$$

or equivalently

$$\frac{1}{\tau} \cdot Var(X_{(j+1)\tau} - X_{j\tau}) = \sigma^2 \cdot (1 \pm \mathcal{O}(\sqrt{\tau/N})) \tag{5.4}$$

Equation (5.4) suggests an important property of a pure Brownian motion process: the time-lag-scaled variances of change corresponding to varying time lags should on average remain flat at the level of σ^2 (where σ is the diffusion coefficient of the underlying Brownian motion) but deviate more and more from the average as the time lag τ increases. This property also allows us to estimate the diffusion coefficient of a sample path of an unknown Brownian motion process. Figure 5.4 shows the time-lag-scaled variances versus time lags plot for a pure Brownian motion process with drift 0 and diffusion coefficient 1.

Now we construct the time-lag-scaled variances (of change) plot for the mid-price time series of the simulated market. The plot creation procedure is described in the following steps:

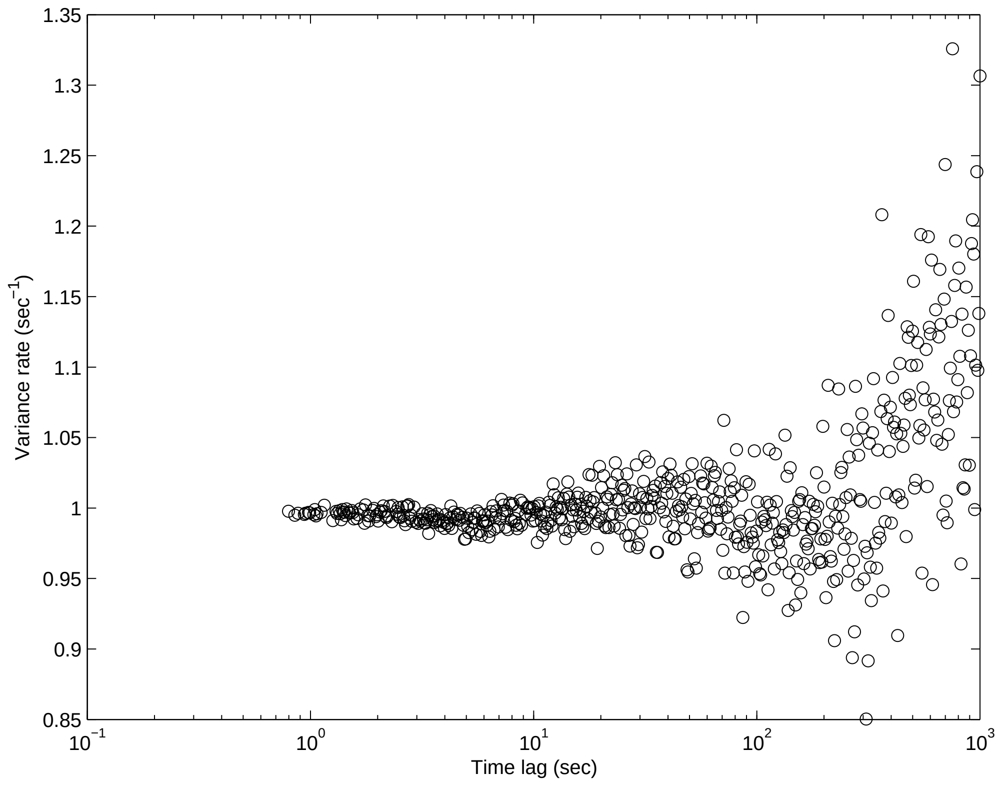


Figure 5.4: Time-lag-scaled variance versus time lag plot for a Brownian motion process X(t) = W(t). One can observe from the graph that the variances remain flat around 1 and deviate more and more from 1 as the time lag τ increases.

1. We first need to interpolate the data points from the original time series, because the times ($t_0 < t_1 < ... < t_N - 1$) in the time series produced by the simulation are not uniformly spread out along the time axis. In fact, $t_{i+1} - t_i$ is exponentially distributed due to the fact that all market events in the SFKG artificial market are modeled by Poisson processes. The interpolation is done using the *nearest neighbour* method, which assigns the interpolated value of an intermediate point to the value of its closest neighbour.

2. For a given time lag τ, we extract data points at 0, τ, 2τ, 3τ, ..., etc from the interpolated time series, form these sample points into a sub-time-series. We measure the variances of change (scaled by $1/\tau$) for this sub-time-series. We repeat this procedure for different values of the time lag τ and graph the results, which is a plot of time-lag-scaled variances (of change) vs. their corresponding time lags. Such a plot is shown in Figure 5.5.

Figure 5.5 suggests a significant difference between the mid-price evolution and a Brownian motion process. More specifically, the change of mid-price exhibits anti-correlations on small time scales. That is, the successive changes of mid-price are not independent (as in the case for a pure Brownian motion) but instead, they are negatively correlated. The anti-correlation effect decreases gradually as the time lag increases.

5.2 The bid-ask spread

Spread is an another important characteristic of double auction markets. Being able to understand and to model the evolution of the spread is central to many optimization problems in finance. In this section, we develop a simple model for the spread in our artificial double-auction market using a Brownian motion process with deterministic, but spread-dependent, drift and diffusion coefficients. In other words, suppose the spread

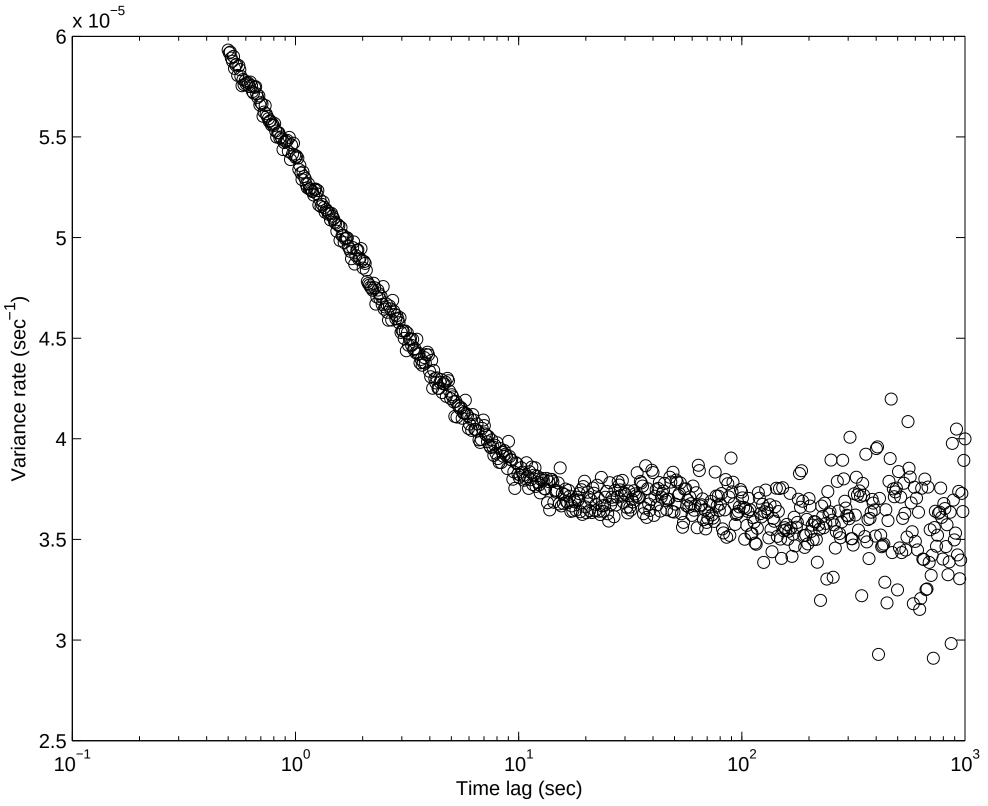


Figure 5.5: Time-lag-scaled variance versus time lag plot for the mid-price. One can observe from the graph that the variances first decrease linearly and then become quite flat as the time lag increases. The flat region is similar to its Brownian motion counterpart (shown in Figure 5.4). Also note that the anti-correlation decays to zero at about 10 sec.

(denoted by S) can be modeled by the stochastic differential equation

$$dS(t) = \mu(S) \cdot dt + \sigma(S) \cdot dW(t) \tag{5.5}$$

where $W(t)$ is a standard Brownian motion. We want to estimate the functional form of $\mu(t)$ and $\sigma(t)$ from the spread time series generated by the simulation. Again, as for the mid-price, the statistical analysis for the spread is done in the logarithmic space. That is, the word "spread" and the symbol S used throughout this section actually refer to the logarithm of the spread.

To estimate the functional forms of the drift and diffusion, we first discretize equation (5.5)

$$\Delta S_j \approx \mu(S_j) \cdot \Delta t_j + \sigma(S_j) \cdot \sqrt{\Delta t_j} \cdot \xi_j \tag{5.6}$$

where the notation S_j means the value of the spread at time t_j, $\Delta S_j = S_{j+1} - S_j$, $\Delta t_j = t_{j+1} - t_j$ and the ξ_js are IID standard normal random variables.

From equation (5.6) the drift function can be written as

$$\mu(S_j) \approx \frac{\Delta S_j}{\Delta t_j} - \frac{\sigma(S_j)}{\sqrt{\Delta t_j}} \cdot \xi_j \tag{5.7}$$

Let's define a set $\mathcal{T}$ containing times, at which the spread S is close to a given value $\hat{S}$

$$\mathcal{T}(\hat{S}) = \{k \mid |S_k - \hat{S}| < \epsilon\}$$

where ϵ is a small positive number. Then equation (5.7) implies that the value of the drift function at a particular value $\hat{S}$ can be estimated as

$$\begin{aligned}
\mu(\hat{S}) &= E(\frac{\Delta S_k}{\Delta t_k} - \frac{\sigma(S_k)}{\sqrt{\Delta t_k}} \cdot \xi_k \mid k \in \mathcal{T}(\hat{S})) \\
&= E(\frac{\Delta S_k}{\Delta t_k} \mid k \in \mathcal{T}) - E(\frac{\sigma(S_k)}{\sqrt{\Delta t_k}} \cdot \xi_k \mid k \in \mathcal{T}(\hat{S})) \\
&= E(\frac{\Delta S_k}{\Delta t_k} \mid k \in \mathcal{T}(\hat{S})) - 0 \\
&= \frac{1}{|\mathcal{T}(\hat{\mathcal{S}})|} \sum_{k \in T(\hat{S})} \frac{\Delta S_k}{\Delta t_k}
\end{aligned} \tag{5.8}$$

Equation (5.8) suggests the following procedure for estimating the drift function from a given time series of the spread.

1. We first sample the spread data points at uniformly spaced time points $t_0 < t_1 < t_2 ... < t_n$, where $t_{j+1} - tj = \Delta t$ for all j. Interpolation may be required in this step.

2. For each S_j (S_j is short for the value of S at time t_j), j is from 1 to n-1, compute $\Delta S_j / \Delta t = (S_{j+1} - S_j)/\Delta t$ and form the $n-1$ by 2 vector $[S_j \ \Delta S_j/\Delta t]$. Sort the rows of the vector by its first column (i.e., by values of S_j) in an ascending order.

3. Separate the rows of the sorted vector into N bins. For each bin, compute the average of the second column (i.e., the average of $\Delta S/\Delta t$). Also record the center value for each bin (i.e., the median value of S belonging to the bin).

4. Now we have N data pairs, each of which contains the median center of the bin and the corresponding average value of $\Delta S/\Delta t$ for the bin. Plot the N data pairs. This plot gives an estimated functional form of the drift coefficient.

Figure 5.6 is a plot resulting from the above drift function estimation procedure. From the plot, one can observe some clear patterns of the drift function: it is a smooth, decreasing, concave-down function that crosses zero at almost exactly 0.015. This implies that the stochastic process followed by the spread is *mean-reverting* and the mean or stationary level of the spread is about 0.015. In other words, when the spread is greater than 0.015, the drift of the spread becomes negative, which creates a tendency to pull back the spread to 0.015; and when the spread is less than 0.015, the drift becomes positive, which creates a tendency to push the spread back up to 0.015. Note that the mean spread 0.015 seen on Figure 5.6 is consistent with the given model-level parameters (see Table 4.2). This is strong evidence of the correctness and reliability of our software simulation platform.

To estimate an algebraic form of the drift function, we can fit or regress (in the least square sense) the data points (shown as circles) from the resulting plot on a second-degree polynomial. The best-fit quadratic approximation of the drift function is

$$u(S) = -30.7064 \cdot S^2 + 0.1309 \cdot S + 0.0046 \tag{5.9}$$

and it is shown in Figure 5.6 as the solid curve going through the circles.

Now we estimate the functional form of the diffusion coefficient for the spread. From equation (5.6) we have

$$\sigma(S_j) \cdot \xi_j = \frac{\Delta S_j}{\sqrt{\Delta t_j}} - u(S_j) \cdot \sqrt{\Delta t_j} \tag{5.10}$$

As we did for estimating the drift function, let's define a set $\mathcal{T}$ containing times at which the spread S is close to a given value $\hat{S}$:

$$\mathcal{T}(\hat{S}) = \{k \mid |S_k - \hat{S}| < \epsilon\}$$

where ϵ is a small positive number. Then equation (5.10) suggests that $\sigma(\hat{S})$ is approximately equal to the standard deviation of the following random sample (denoted by $D(\hat{S})$):

$$D(\hat{S}) = \{\frac{\Delta S_k}{\sqrt{\Delta t_k}} - u(\hat{S}) \cdot \sqrt{\Delta t_k} \mid k \in \mathcal{T}(\hat{S})\}$$

In other words, the value of the diffusion function at some given value $\hat{S}$ can be estimated as

$$\sigma(\hat{S}) = Std[D(\hat{S})] \tag{5.11}$$

where *Std* is short for standard deviation. Note that since $u(\hat{S}) \cdot \sqrt{\Delta t_k}$ is a constant term for a given $\hat{S}$ (assuming time points are equally spaced, i.e., Δt is constant). Therefore, the estimation of $\sigma(\hat{S})$ can be further reduced to

$$\sigma(\hat{S}) = Std[\bar{D}(\hat{S})] \tag{5.12}$$

where $\bar{D}(\hat{S})$ is a random sample defined as

$$\bar{D}(\hat{S}) = \{\frac{\Delta S_k}{\sqrt{\Delta t_k}} \mid k \in \mathcal{T}(\hat{S})\}$$

Based on equation (5.12) we have the following procedure for estimating the spread's diffusion function.

1. We first sample spread data points at uniformly spaced time points $t_0 < t_1 < t_2 ... < t_n$, where $t_{j+1} - tj = \Delta t$ for all j. Interpolation may be required in this step.

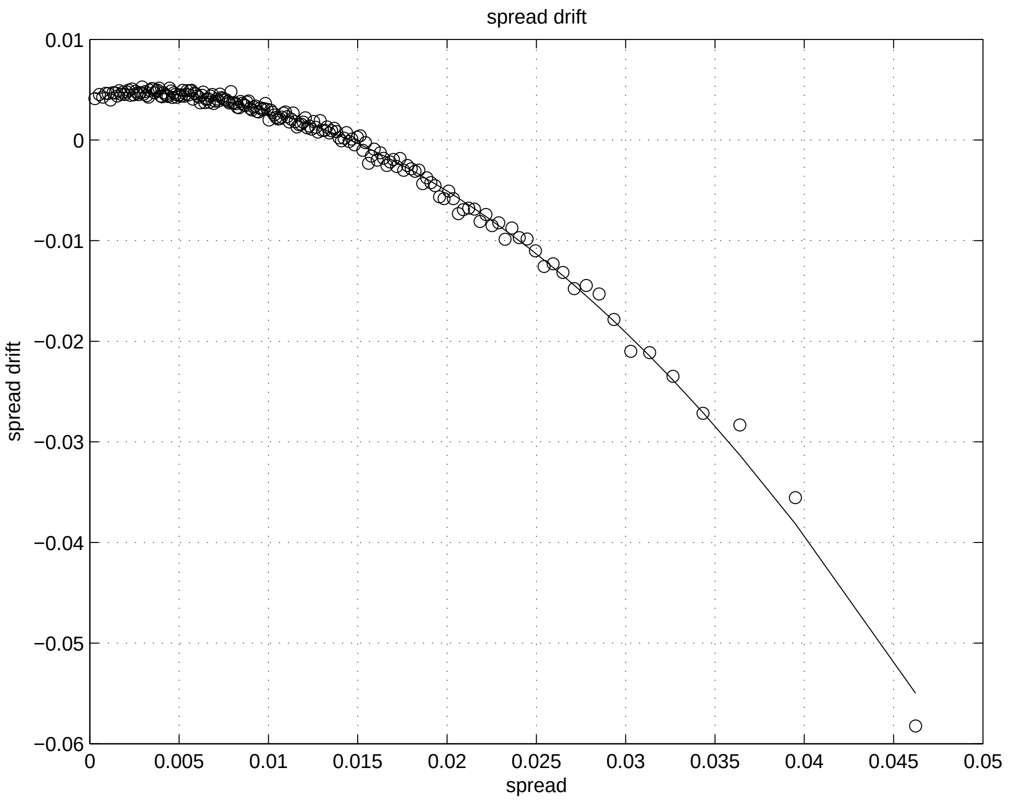


Figure 5.6: The functional form of the spread's drift coefficient estimated from the spread time series. The circles are the original plot produced by the estimation procedure and the solid curve is a second-degree polynomial that best fits (in the least square sense) the original plot. The estimation becomes more noisy at the far right end as there are fewer spread data points that fall into those high value regions. This figure suggests that the stochastic process followed by the spread is mean-reverting and the long-term stationary mean spread (where the drift function crosses 0) is about 0.015.

2. For each S_j (S_j is short for the value of S at time t_j), j is from 1 to n-1, compute $\Delta S_j/\sqrt{\Delta t} = (S_{j+1} - S_j)/\sqrt{\Delta t}$ and form the $n-1$ by 2 vector $[S_j \ \Delta S_j/\sqrt{\Delta t}]$. Sort the rows of the vector by its first column (i.e., by values of S_j) in an ascending order.

3. Separate the rows of the sorted vector into N bins. For each of the N bins, compute the standard deviation of the second column (i.e., the standard deviation of $\Delta S/\sqrt{\Delta t}$). Also record the center value for each bin (i.e., the median value of S belonging to the bin).

4. Now we have N data pairs, each of which contains the median center of the bin and the corresponding standard deviation of $\Delta S/\sqrt{\Delta t}$ for the bin. Plot these N data pairs. This plot gives an estimated functional form of the spread's diffusion coefficient.

Figure 5.7 is the plot resulting from the above diffusion function estimation procedure. From the plot, we see that the diffusion function is smooth, increasing and concave up. To estimate an algebraic form of the diffusion function, we fit or regress (in the least square sense) the data points (shown as circles) from the resulting plot on a second-degree polynomial. The best-fit quadratic function is computed to be

$$\sigma(S) = 7.752 \cdot S^2 + 0.2399 \cdot S + 0.0052 \tag{5.13}$$

It is shown in Figure 5.7 as the solid curve going through the circles. Note that the quadratic function for $\sigma(S)$ does not go to zero as S becomes close to zero. As a result, theoretically speaking, there is a possibility (however slim) that the solutions to the SDE $dS(t) = \mu(S) \cdot dt + \sigma(S) \cdot dW(t)$ may cross zero. This is a limit of the SDE model developed above.

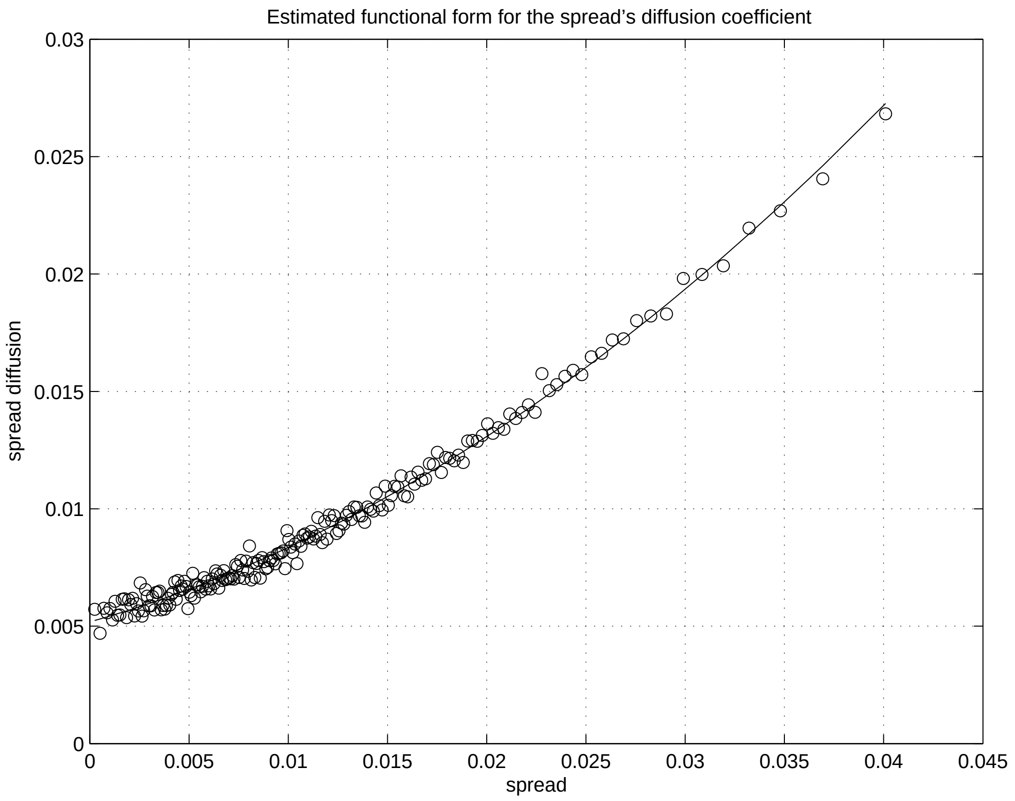


Figure 5.7: The functional form of the spread's diffusion coefficient estimated from the spread time series. The circles are the original plot produced by the estimation procedure and the solid curve is a second-degree polynomial that best fits (in the least square sense) the original plot. The estimation becomes more noisy at the far right end as there are fewer spread data points that fall into those high value regions.

5.3 The limit order book

The limit order book is a central component of double-auction markets and it is relevant to many optimization problems in finance. However, a challenge for researchers to study the structural properties of the limit order book is that the empirical data regarding limit order books are much less accessible than other real world financial data, such as stock prices, spreads, etc. Information about the limit order book is termed *Level 2 Data* in the financial world and the scarcity of level 2 data may either be caused by the natural difficulty to compose and record them or by the fact that many exchanges (such as NYSE and NASDAQ) charge fees for accessing them. The use of agent-based artificial markets, on the other hand, can provide us sufficient simulated level 2 data and, to a certain extent, allow us to gain insights about the structure of the limit order book.

This section focuses on limit order densities in the limit order book. In particular, we present several "snapshots" of the cumulative limit order density plot, taken at random times during a simulation run of the agent-based SFGK artificial market. For limit buy orders, the cumulative density plot is a step plot, for which the X axis records the limit buy prices (denoted by P_b) currently existing in the book and the Y axis gives the corresponding number of buy orders whose limit prices are greater than or equal to P_b. Similarly, for limit sell orders, its cumulative order density plot is a step plot where the X axis records the limit sell prices (denoted by P_s) and the Y axis gives the corresponding number of sell orders whose limit prices are smaller than or equal to P_s. The cumulative order density plot interests many people because it is closely related to the *price impact* function for selling or buying a large block of assets. For example, if a financial institution submits a market sell order of size 1000 to a double-auction market, the price impact (of the average selling price) from this sale can be estimated from the cumulative limit buy order density a follows. Let $D(p)$ be a continuous function that approximates the cumulative buy order density. And let b denote the current bid price and k denote the limit price such that $D(k) = 1000$. Then the price impact $\bar{P}$ can be estimated by calculating the integral

$$\bar{P} = \int_b^k D(p)dp$$

Hence with a good understanding of the cumulative order density in a double-auction market, it is possible for a financial institution to estimate price impact for large transactions.

This section does not intend to rigorously model the cumulative order density from the level 2 data obtained from simulations. Rather, it simply produces some sample cumulative order density plots by processing the "snapshots" of the limit order book, taken at random times during a simulation run. Figure 5.8 provides two such sample plots for limit orders that reside reasonably close to the spread region. These plots can give a decent indication of the cumulative order density associated with the SFGK model. Serious researchers who want an accurate picture of the cumulative order density should develop a reasonable and systematic technique to average these sample plots.

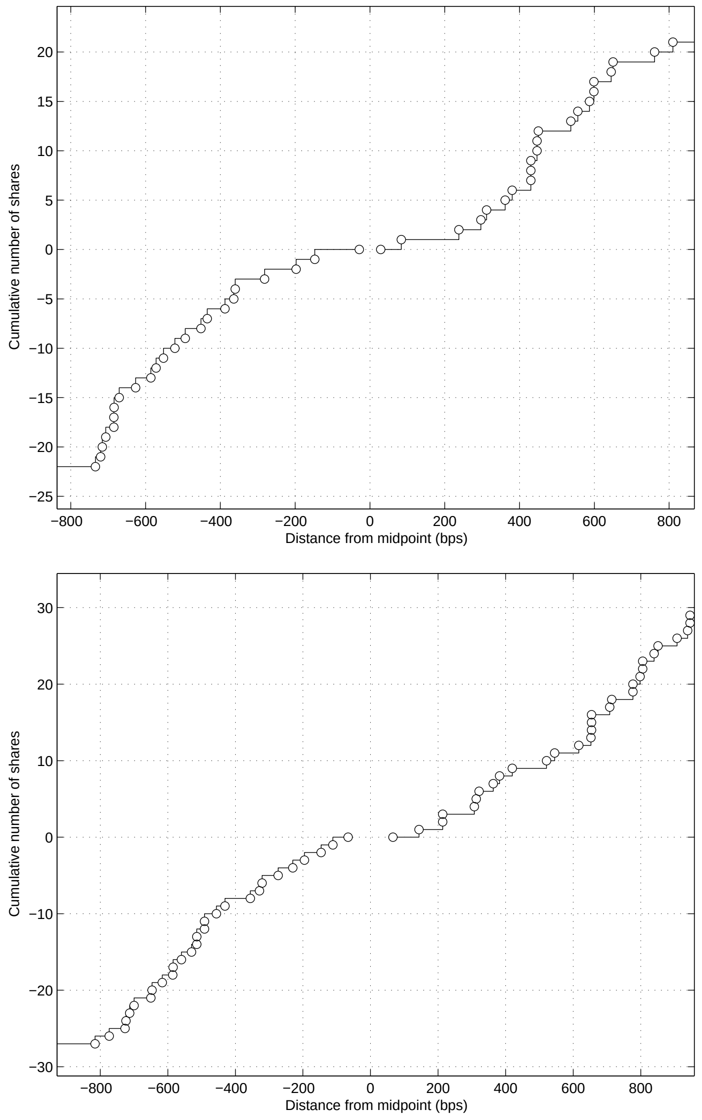


Figure 5.8: Sample cumulative order density plots for limit orders that are close to the spread region. The empty circles represent limit orders stored in the limit order book. The X axis gives the logarithmic distance of limit prices away from the mid-point. The left (right) half of each plot gives the cumulative order density for limit buy (sell) orders. The Y axis gives the cumulative number of shares for a given limit price. Note that on the grid line $Y = 0$, the gap between the two disconnected circles indicates the current spread.

Chapter 6

Trading strategy experiments

The software platform together with the SFGK model provide us with an agent-based simulation laboratory for a double-auction market. The role of the software platform is to lay out the necessary infrastructure for the market and the role of the SFGK model is to generate certain market dynamics which allows the artificial market to evolve automatically itself and to mimic reasonably the behaviors of a real financial market. Such a simulation laboratory can give researchers and financial institutions an edge in solving many optimization problems in finance because it allows them to carry out experiments and test strategies on a simulated market instead of on an actual market (which could be very costly). In this chapter, we will show how one can take advantage of our simulation laboratory to explore and experiment with trading strategies associated with time-constrained asset-liquidation.

6.1 The time constrained asset liquidation problem

The problem that we explore here is how to find good trading strategies for liquidating a large block of assets with a given time constraint. Such problems arise frequently for large institutional investors (such as pension funds, hedge funds and insurance companies). For example, a financial institution may need to liquidate part of its portfolio to fund short-term cash obligations coming due or it may be the case that it is required to sell portions of a portfolio (as part of the portfolio rebalancing effort) to meet risk management criteria. In either case, the financial institution will need to complete the liquidation process within a given period of time.

What trading strategies should the financial institution adopt in order to maximize its revenue from liquidating the asset? The answer to this question is not trivial. Markets have finite liquidity and the asset under liquidation is often significant in size compared to the available liquidity in the market. Hence it is either impossible or infeasible to liquidate instantly the entire block of assets: the market either may simply not have sufficient liquidity to satisfy the order or, even if it does, selling the entire asset at once may yield an unacceptably low average selling price. Recall that in the context of a double-auction market (the most common form of financial market today), such as the one whose limit order book is shown in Table 1.1, market liquidity is provided by limit orders residing in the limit order book and the larger the size of a market sell order being placed on the market, the deeper it will go below the current best bid price to get satisfied and the poorer the average selling price will be. Given the above argument, it is clear that the liquidation must be done in a "chunk-wise" fashion. In other words, the financial institution must divide the whole block of asset into chunks and sell them one chunk at a time.

However, what remains unclear is at what rhythm those chunks should be sold to the market. In the next section, we present two simple and feasible strategies for the asset liquidation problem. In the following section, we perform simulation experiments to evaluate the performance of the two strategies. It should be noted that this paper does not intend to provide a rigorous and theoretical treatment to the general optimization problem of asset liquidation. Rather, we are primarily interested in taking an experimental approach to explore a subset of the time-constrained asset liquidation strategies, for which we assume that the liquidation is done though market sell orders only. One purpose of such an exploration is to demonstrate how our simulation laboratory can benefit institutional investors by allowing them to experiment with and vet various trading strategies. For a comprehensive and mathematical treatment to the general asset liquidation problem (especially about the optimal trajectory for the rate of trading), please see the paper written by Almgren and Chriss (2000) who pioneered much of the early research in this field.

6.2 Two different trading strategies

Perhaps the most obvious choice of the liquidation rhythm is to sell the chunks of the asset at fixed and uniformly-spaced time points: given that the size of the asset is X, the deadline for completing the liquidation is T and the asset is divided into N chunks, the "uniform rhythm" strategy instructs a trader to sell X/N shares every T/N seconds, regardless of the market conditions. Now the question is: Can we find a better trading strategy than the "uniform rhythm" one just described?

The answer is yes. With the help of our simulation laboratory, we are able to take an experimental approach to search for other possible trading strategies that can yield a better average selling price for the asset. In what follows, we will propose a "non-uniform rhythm" trading strategy and in the next section we will show through simulation experiments that this strategy can outperform the "uniform rhythm" strategy in the average sense.

Unlike the "uniform rhythm" strategy, which sells chunks of the asset at fixed time-points (uniformly spaced) regardless of market conditions, the "non-uniform rhythm" strategy is more sophisticated in that it dynamically decides when to initiate selling based on the current spread in the market. More precisely, given that X is the total amount of asset to be liquidated, T is the length of time within which the liquidation process must be completed and N is the number of chunks that the asset is divided into, the "non-uniform rhythm" trading strategy does the following.

1. Before starting to sell, the strategy first picks a threshold for the spread. The spread threshold should be small. For example, it can be set to be a small fraction of the mean spread observed in the market. The liquidation timeline (from $t = 0$ to $t = T$) is divided into N segments, each of which has a length of T/N seconds and, within each time segment, X/N amount of shares must be sold.

2. So far, the "non-uniform rhythm" strategy is almost identical to its uniform counterpart. What distinguishes them is that, instead of selling the X/N shares of the asset at a fixed time point within each time segment (which is the case for the "uniform rhythm" strategy), the "non-uniform rhythm" strategy requires a trader to continuously observe the market spread and initiates the selling of the X/N shares as soon as the current market spread, for the first time within the time segment, falls below the pre-determined spread threshold. If the market spread never falls below the spread threshold for a time segment, the strategy will sell the X/N shares at the end of the time segment.

Figure 6.1 illustrates the difference between the two trading strategies.

6.3 Performance evaluation experiment

In the previous section, we introduced two trading strategies associated with the asset liquidation problem. To evaluate the performance of the two strategies, we implement two special artificial agents, each of which is programmed to follow one of the two strategies to liquidate a certain amount of asset within a fixed length of time. Each agent (or strategy) is tested individually on the simulated double-auction market for 200 runs. The final evaluation result will be based on the agents averaged performance on these tests. The detailed layout of our experiment is summarized below.

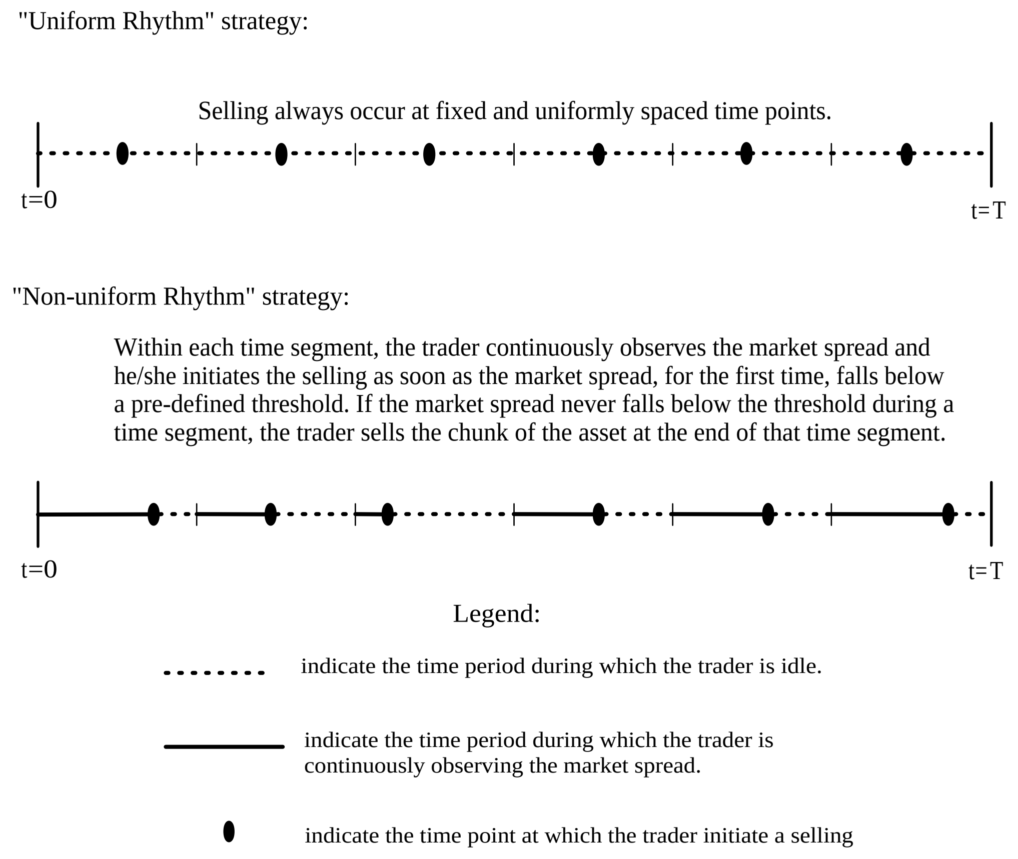

"Uniform Rhythm" strategy:
Selling always occur at fixed and uniformly spaced time points.
t=0
t=T
"Non-uniform Rhythm" strategy:
Within each time segment, the trader continuously observes the market spread and he/she initiates the selling as soon as the market spread, for the first time, falls below a pre-defined threshold. If the market spread never falls below the threshold during a time segment, the trader sells the chunk of the asset at the end of that time segment.
t=0
t=T
Legend:
indicate the time period during which the trader is idle.
indicate the time period during which the trader is continuously observing the market spread.
indicate the time point at which the trader initiate a selling

Figure 6.1: An illustraction of the “uniform rhythm” and “non-uniform rhythm” trading strategies.

1. Both special agents are given the same objective, which is to liquidate 20 shares within 5 minutes of time. Note the number of shares and the time duration of the liquidation are both scaled down to be consistent with the scales in the "simulated world". In particular, the size of the asset under liquidation (20 shares) is actually quite significant compared to the available liquidity in the simulated market. The total asset is divided into 10 chunks, each of which contains 2 shares that will be sold within a time segment of 30 seconds.

2. Implement two special agents A (with "uniformand rhythm" strategy) and B (with "non-uniform rhythm" strategy) by coding new artificial threads in the simulation framework (see Figure 3.2). For agent B, the required spread threshold is chosen to be one-eighth of the stationary mean spread of the simulated double-auction market, (i.e., the spread threshold is equal to $0.125 \times 0.015 = 0.001875$, in logarithmic space).

3. Each special agent will be tested individually on the simulated market for 200 runs. At the end of each test, we calculate and record the following quantity

$$G_i = \frac{\bar{P}_i}{\text{VWAP}_i}$$

   where $\bar{P}_i$ denotes the average selling price per share realized for the ith run and $VWAP_i$ denotes the corresponding VWAP[1] associated with the time period of the ith test (300 seconds in length). G_i is a performance measure benchmarked to VWAP, which tells us how much better or worse an agent performs during the ith test comparing to the market. It is important to use the benchmarked measure G_i, rather than P_i, as the evaluation criterion because market fluctuations can make P_i a very noisy and biased measure. In other words, a bigger P_i alone doesn't

[1]VWAP is the abbreviation of "Volume Weighted Average Price" and it is an objective and effective measure of the average execution price taking place in the market for a given period of time. In the real financial world, VWAP is commonly used as a benchmark to measure the efficiency of institutional trading or the performance of traders themselves.

necessarily mean a better performance. For example, when the market is bullish, a trader will generally get higher values of P_i than when the market is bearish, but in order to fairly judge their performance, we benchmark it to the market.

4. The last stage of the experiment is to evaluate the test results (from the previous step) for both agents and to draw conclusions about their performance. Let G^A (G^B) denote the random variable representing the benchmarked performance for agent A (agent B), for which we have collected a sample that contains 200 realizations: G_i^A (G_i^B), $i = 1, 2, ...200$ in the previous step. We compute the sample mean and sample standard deviation for both G^A and G^B, from which a conclusion is drawn about whether there is a significant difference between the two strategies.

6.4 Performance evaluation results

Now we report the results from the evaluation experiment:

- For Agent A (based on the "uniform rhythm" strategy): G^A has a sample mean of 0.99436 and a sample standard deviation of 0.0042, which implies that agent A on average underperforms the market by 56.4 ± 2.97 basis points.[2]
- For agent B (based on the "non-uniform rhythm" strategy): G^B has a sample mean of 1.00034 and a sample standard deviation of 0.0052, which implies that agent B on average outperforms the market by 3.4 ± 3.68 basis points.

Figure 6.2 plots the empirical cumulative distribution functions (CDFs) for both G^A and G^B. Observe that there is a clear gap between the two CDFs and that the CDF for G^B is to the right of that for G^A, which suggests that on average agent B (based on the "non-uniform rhythm" strategy) outperforms agent A (based on the "uniform

[2]The number 2.97 gives the approximated error range (or standard deviation) of the sample mean. It is calculated by dividing the sample standard deviation (0.0042 or 42 basis points) by the square root of the sample size (200).

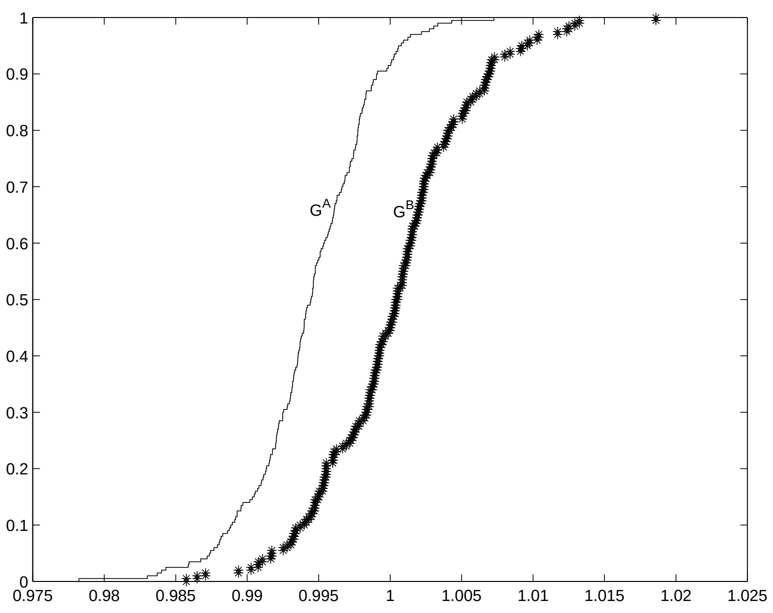


Figure 6.2: Empirical cumulative distribution functions (CDFs) for G^A and G^B. We can observe a clear gap between the two CDFs. Moreover the CDF for G^B is to the right of the one for G^A, which is statistical evidence suggesting that on average agent B (based on the "non-uniform rhythm" strategy) outperforms agent A (based on the "uniform rhythm " strategy).

rhythm" strategy). The "non-uniform rhythm" trading strategy introduced here was actually inspired by traders in the real financial world who use similar strategies to liquidate large blocks of assets. Our experiment and the statistical results from it suggest that such a strategy is indeed quite effective. A trader using the "non-uniform rhythm" strategy will have almost a 60-basis-point advantage over another trader using the simpler "uniform rhythm" trading strategy, which in a large transaction could mean a difference of thousands of dollars.

Chapter 7

Conclusion and future work

7.1 Conclusion

The main contributions of this thesis include the presentation of an agent-based simulation environment, the investigation of market microstructure properties of an artificial double-auction market and the evaluation of trading strategies for the time-constrained asset-liquidation problem. In particular, we have achieved the following objectives.

1. We successfully built a high-quality software platform that is specifically designed to carry out agent-based simulations of financial markets. The software platform is fully multi-threaded, employs a client/server architecture and offers a distributed computing framework on which researchers can experiment with their own agent-based artificial markets. The high scalability of the software makes the integration of new artificial agents (or the replacement of existing ones) as easy as dealing with plug-and-play devices.

2. We implement the SFGK artificial model in an agent-based setting on top of our software platform. The SFGK model provides a simple and robust mechanism for simulating the evolution of a double-auction market. It is central to much of the research work carried out in this thesis because it not only provides a natural

and comprehensive test case for our software infrastructure but, more importantly, generates the necessary and reasonably realistic market dynamics that allows us to study its market microstructure properties and to use it as a testbed to evaluate trading strategies.

3. We perform an in-depth investigation on the microstructure properties of the SFGK artificial market through extensive simulation experiments. We first study the mid-price dynamics produced by the model and compare it to a pure Brownian motion process. We find that the mid-price in the SFGK model has "short-term memory" (or anticorrelation) which gradually diminishes over time. We then study the evolution of the bid-ask spread. In particular, we model the spread using a stochastic differential equation of the form $dS(t) = \mu(S) \cdot dt + \sigma(S) \cdot dW(t)$ and we show that the drift coefficient $\mu(S)$ and the diffusion coefficient $\sigma(S)$ can both be approximated well by simple quadratic functions. We also look into the limit order book structure by providing sample cumulative order density plots.

4. We evaluate trading strategies for the time-constrained asset liquidation problem, based on simulation experiments conducted on the SFGK artificial double-auction market. We propose and compare two feasible trading strategies, one of which employs a fixed and uniform selling rhythm, and the other of which employs a dynamic and non-uniform selling rhythm. We implement two special agents, each of which is programmed to follow one of the strategies. And we individually and repeatedly evaluate each special agent's trading performance (benchmarked to the market) on the simulated market. The experimental results show that on average the dynamic non-uniform strategy outperforms the uniform one by almost 60 basis points.

7.2 Future work

The agent-based simulation environment (consisting of the software platform and the SFGK artificial market model) presented in this thesis has the advantage of being flexible enough to allow for many extensions and applications. In what follows, we suggest two avenues of future research work that can directly proceed from here.

The first avenue involves possible enhancements to the SFGK model. Recall that in the current model, the log of limit prices are chosen randomly from uniform distributions. For example, the log limit buy prices are drawn uniformly over the price interval $(-\infty, a(t))$, where $a(t)$ denotes the lowest ask price at time t. The setting that assigns equal weight or probability (because of the uniform distribution) to all limit prices is wasteful because it results in many "not-so-useful" limit orders whose limit prices are unreasonably far way from the spread. One possible enhancement is to replace the uniform distribution with an exponential distribution, which assigns more weight or probability to limit prices that are close to the spread than those that are further away from it. Smith, Farmer, Gillemot, and Krishnamurthy (2003) suggest several other enhancements in their original paper that interested readers may refer to.

The second avenue involves possible applications of our simulation environment. As we have demonstrated in this thesis, the simulation environment can be used as a testbed for exploring and evaluating trading strategies. On a broader front, it may be interesting to see how it can be applied to the area of risk management. For example, the SFGK model has the potential to simulate various disastrous scenarios such as a market crash or a liquidity crisis, which could be very useful in risk management. To simulate a market crash using the SFGK model, for example, one can simply adjust the relevant rate parameters of the model to cause an imbalance of buy orders and sell orders arriving at the market. The excess of sell orders (supply) over buy orders (demand) will result in a price decrease. The duration, severity and frequency of such market crashes can be completely manipulated by setting the agent-level parameters appropriately. Similarly, to simulate a liquidity crisis, one can simply adjust the ratio of the arrival rate of limit orders versus that of the market orders to cause an imbalance of limit orders and market orders arriving at the market. The excess of market orders over limit orders will decrease the market liquidity and widen the spread. Again, the duration, severity and frequency of a liquidity crisis can be completely controlled.

Bibliography

Almgren, R. and N. Chriss (2000). Optimal execution of portfolio transactions. *J. Risk 3*(2), 5–39. 64

Chiarella, C. and G. Iori (2002). A simulation analysis of the microstructure of double auction markets. *Quant. Finance 2*, 346–353. 3, 9, 12

Das, S. (2003). Intelligent market-making in artificial financial markets. Master's thesis, Massachusetts Institute of Technology. 3, 4, 12, 15

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*, Chapter 3.1, pp. 79–81. Springer. 48

Gode, D. K. and S. Sunder (1993). Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *J. Political Econ. 101*(1), 119–137. 2, 16

LeBaron, B. (2000). Agent-based computational finance: Suggested readings and early research. *J. Econom. Dynam. Control 24*, 679–702. 4

LeBaron, B. (2001). A builder's guide to agent-based financial markets. *Quant. Finance 1*, 254–261. 2

LeBaron, B., W. B. Arthur, and R. Palmer (1999). Time series properties of an artificial stock market. *J. Econom. Dynam. Control 23*, 1487–1516. 2

Smith, E., J. D. Farmer, L. Gillemot, and S. Krishnamurthy (2003). Statistical theory of the continuous double auction. *Quant. Finance 3*, 481–514. 7, 33, 72

Software Engineering Institute, Carnegie Mellon University (2003). *Software Technology Roadmap*. Software Engineering Institute, Carnegie Mellon University. 18