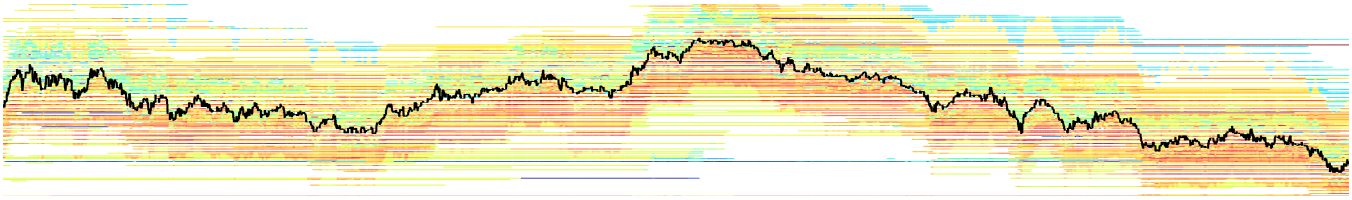
ABIDES: Towards High-Fidelity Multi-Agent Market Simulation

David Byrd
db@gatech.edu
School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia

Maria Hybinette
maria@cs.uga.edu
Department of Computer Science
University of Georgia
Athens, Georgia

Tucker Hybinette Balch*
tucker.balch@jpmchase.com
J.P. Morgan AI Research
New York, New York



ABSTRACT

We introduce ABIDES, an open source Agent-Based Interactive Discrete Event Simulation environment. ABIDES is designed from the ground up to support agent-based research in market applications. While proprietary simulations are available within trading firms, there are no broadly available high-fidelity market simulation environments. ABIDES enables the simulation of tens of thousands of trading agents interacting with an exchange agent to facilitate transactions. It supports configurable pairwise noisy network latency between each individual agent as well as the exchange. Our simulator's message-based design is modeled after NASDAQ's published equity trading protocols ITCH and OUCH. We introduce the design of the simulator and illustrate its use and configuration with sample code, validating the environment with example trading scenarios. The utility of ABIDES for financial research is illustrated through experiments to develop a market impact model. The core of ABIDES is a general-purpose discrete event simulation, and we demonstrate its breadth of application with a non-finance work-in-progress simulating secure multiparty federated learning. We close with discussion of additional experimental problems it can be, or is being, used to explore, such as the development of machine learning trading algorithms. We hope that the availability of such a platform will facilitate research in this important area.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; *Agent / discrete models*; • **Applied computing** → *Economics.*

KEYWORDS

simulation, finance, market, discrete, multiagent

ACM Reference Format:
David Byrd, Maria Hybinette, and Tucker Hybinette Balch. 2020. ABIDES: Towards High-Fidelity Multi-Agent Market Simulation. In *Proceedings of the SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '20), June 15–17, 2020, Miami, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3384441.3395986

1 INTRODUCTION

We present ABIDES (Agent-Based Interactive Discrete Event Simulation), to facilitate the creation, deployment, and study of strategic agents in a highly configurable market environment. We were inspired by Daniel Freidman's view that simulation of markets provides a powerful tool to analyze individual participant behavior as well as overall market outcomes that emerge from the interaction of the individual agents [7]. In Freidman's review of empirical approaches to the analysis of continuous double auction (CDA) markets such as NASDAQ and the New York Stock Exchange, he outlines the strengths and weaknesses of three major approaches:

(1) Field studies of actual operating markets,
(2) Laboratory studies of small controlled markets,
(3) Computer simulation of markets.

Freidman concludes that field studies are clearly relevant, but do not provide experimental access to all relevant information; laboratory studies improve control and observation, but are of necessity small and expensive; while computer simulations feature perfect control and observation. However, he notes a significant shortcoming, that a "trader's strategies are not endogenously chosen, but rather must be specified exogenously" [7].

Accordingly, simulation provides an attractive platform for research in equity trading questions. This has led to the development of a number of simulation platforms such as those on which X. Wang and Wellman and J. Wang et al. have reported their results [27, 28]. ABIDES is a fresh implementation incorporating lessons learned from prior platforms.

With ABIDES, we aim to address Freidman's primary concern regarding computerized market simulations – that strategies must be exogenously specified – by enabling powerful *learning* agents

*Also with School of Interactive Computing
Georgia Institute of Technology.

SIGSIM-PADS '20, June 15–17, 2020, Miami, FL, USA

ACM ISBN 978-1-4503-7592-4/20/06…$15.00
https://doi.org/10.1145/3384441.3395986

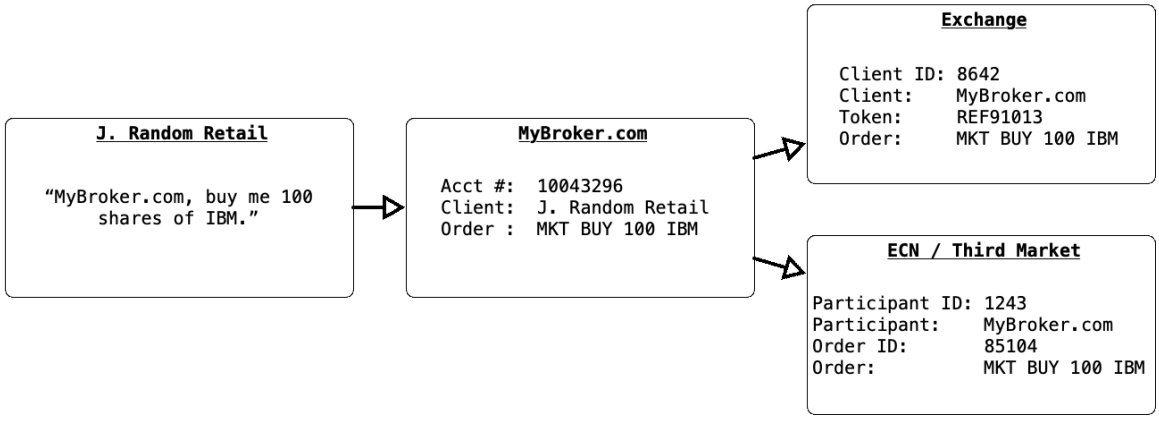


Figure 1: Simulation allows agent-identifiable data which is lost in the flow of real-world orders.

to participate in a realistically structured market via a common framework. We believe this is necessary to properly investigate the behavior and impact of intelligent agents interacting in a complex market environment.

ABIDES is a curated, collaborative open-source project that provides researchers with *tools* that support the rapid prototyping and evaluation of complex market agents. With it, we hope to further empower researchers of financial markets to undertake studies which would be difficult or impossible in the field, due to the absence of fine-grained data identifiable to individual traders (see Figure 1), a lack of knowledge concerning participant motivation, and an inability to run controlled "what if" studies against particular historical dates.

X. Wang and Wellman have examined the behavior of markets populated by multiple agents as well. In a recent study they used their simulation platform to study spoofing agents in a market environment populated by zero intelligence (ZI) and heuristic belief learning (HBL) traders [28]. Their approach analyzes the results from an empirical game-theoretic view [29]. ABIDES makes a complementary contribution by enabling experimental focus on the "market physics" of the real world including:

- Support for continuous double-auction trading at the same nanosecond time resolution as real markets such as NASDAQ;
- Ability to simulate specific dates in market history with gated access to historical data;
- Variable electronic network latency, a realistic cubic network jitter model, and agent computation delays;
- Requirement that all agents intercommunicate solely by means of standardized message protocols;
- Easy implementation of complex agents through a full-featured hierarchy of base agent classes.

These features enable an expanded range of experimental studies. We believe ABIDES is also the first full-featured, modern market simulator to be shared with the community as an open source project.

2 IMPORTANT QUESTIONS ABOUT MARKETS THAT SIMULATION CAN HELP US ADDRESS

ABIDES can support a number of different kinds of investigations into market behavior that are not easily conducted using historical data or live experiments.

- **The benefits of co-location**: In the past 20 years hedge funds and other market participants have invested in the deployment of computing resources at major exchanges [30]. This so-called "co-location" enables quicker access to market information than if the trading server were located further away. It is not feasible to investigate the value of the advantage co-location provides with available historical data, because it does not include information about the geographic location, network latency, or network reliability of each actor. With a platform that does not require formal arms-length messaging using a realistic network model, we cannot simulate the effects of these factors even if they are known. ABIDES provides a network model and mandatory messaging protocol that enables detailed experiments in this area: Creating a population of agents with a realistic and known distribution of network latency, jitter, and reliability; conducting trials in which one agent, pursuing a low-latency order book imbalance strategy, is incrementally shifted from a co-location facility out to a great distance; and evaluating the impact of this shift on all agents' profitability while otherwise pursuing the same strategies.
- **The impact of large orders on price**: The very act of placing orders in a market may affect the price. For instance, if there is significant selling pressure evidenced by a large volume of sell orders, it is generally expected that the price will go down. The extent to which the price moves because of an

order is referred to as *market impact*. Market participants of course want to minimize such impact, because the market usually moves contrary to their profit incentives. In a market field study, it is not feasible to perform controlled A/B tests. One cannot place a market buy at the NYSE for one million shares of IBM at 10 AM on Oct 22, 2018, and then also *not* place that order, and compare the difference. Without the "control", any observed result from the large order could be attributable to some other factor. A key feature of ABIDES is the ability to re-simulate the same historical market day with known, limited changes while holding all other factors constant, thus enabling the desired experimental control population.

- **Cost-benefit analysis of AI**: In a simulation without a model for computational time delays that directly impact time-to-market for the resulting orders, we cannot readily study the trade-off between simpler, faster predictors and slower, more powerful predictors. ABIDES introduces a flexible, integrated model for computation delay that permits the "speed" of each agent's thought process to be represented, and to have that representation affect the timing of all of outbound messages as well as the next time at which the agent can be roused for participation. These computation delays can be specified, or can be measured and applied in real time during simulation, such that an agent is delayed according to the actual runtime of each computation. Thus heavier thinkers will take longer to deliver a resulting order to the exchange and will be unable to act as frequently.
- **Explanation of behavior**: When analyzing historical market data, we cannot know the motivation behind individual trader actions, but *intent* is a key component of many market regulations. For example, "spoofing" (placing limit orders one does not intend to fulfill) is not permitted in U.S. markets. It is also extremely difficult to detect or study, because an identical pattern of placing and canceling orders may be lawful or unlawful depending on the trader's intentions. Similarly, with the shift away from knowledge-based AI toward "black box" ML models, explaining the actions of intelligent agents has become more challenging. ABIDES provides a platform that features high-resolution time-synced event logging and visualization for: trading agent state, portfolio, strategy, and orders; exchange agent order books, order handling, and order execution; and any extrinsic price-time series used to guide value-conscious strategies. Combining the precision logs with a quality simulation architecture which requires all inquiries and impulses to pass as messages through a central Kernel for scheduling and tracking, such that each agent's decisions, intentions, communications, and results for every action are fully visible, we produce the full scope of information needed for explanatory reconstructions. We hope to use this ability to dive deeply into the *why* of trading policies learned by agents or observed in real markets.

3 ABIDES OVERVIEW

Financial markets operate over continuous time, such that transactions may be concluded at an exchange anytime the market is open. Simulations of these systems rely on representations of entities within the simulated environment to estimate the state of the system at times after initialization. Two conventional approaches include continuous systems and discrete event systems. Continuous simulation typically relies on differential equation models to estimate the state of a system at any arbitrary time t. Because t is continuous, the potential values for t are unaccountably infinite. As an example from Finance, the Black Scholes model of options pricing uses a continuous differential equation approach [5, 19]. Their model assumes price follows a geometric Brownian motion.

In contrast to continuous simulation, discrete event simulation (DES) methods can be used to estimate the state of a system at specific discrete points in time. The system changes state only at the edges of those discrete time slices. Discrete event simulations often use random variables as models. For example, a Poisson distribution might be used to determine the periods between arrival times (inter-arrival time) of phone calls at a call center. A discrete event simulation system is fast and efficient because the time between state changes can be ignored and skipped over.

Discrete-event models are better suited than continuous-time models when underlying parameters change. Further, DES are amenable to computational parallelization that further speeds up execution. Approaches in parallel discrete event simulation (PDES) can follow Optimistic or Conservative protocols [6, 13].

A parallel simulation can be viewed as an extension to a sequential discrete event simulation where each sequential simulation is modeled by a logical process (LP). A change in state is defined by processing an event at some scheduled time t at a logical process. While ABIDES is currently not parallelized nor distributed at the LP level, its software design follows principles of distributed simulation. For example "agents" in ABIDES are mapped to logical processes and ABIDES progresses in time by scheduling events at these agents or logical process.

Several emerging simulation applications, like the modeling of the stock market, call for an agent-based view. An agent-based model (ABM) is a model that is formed by a set of autonomous agents that interact with their environment (including other agents) through a set of internal rules to achieve their objectives. Agent-based modeling and simulation (ABMS) is useful, usable, and already used in a variety of application domains [17]. ABMS helps research and investigation in social sciences [2], computational economics [25], and marketing [23]. Many agent-based simulators have been developed (e.g., Swarm [20] and Mason [16]). ABIDES entities, mapped to logical processes, are indeed agents. Consequently, ABIDES provides both performance and efficiency leveraging from the design of PDES, and flexibility and familiarity of an of an agent based interface leveraging the growing literature in ABM.

Agent-based financial market simulation has been shown to be an effective approach when agents can learn and adapt to different investment strategies [14]. In the financial literature there are simulators that use learning behaviors with differing perspectives of past data [11]. Financial market approaches are either synchronous or asynchronous. Levy et al [15] propose a synchronous approach, but we believe that asynchronous approaches are more flexible and scalable. This view is shared by Jacobs et al [11, 12], who proposed a framework called JLMSim. JLMSim is a discrete event simulator

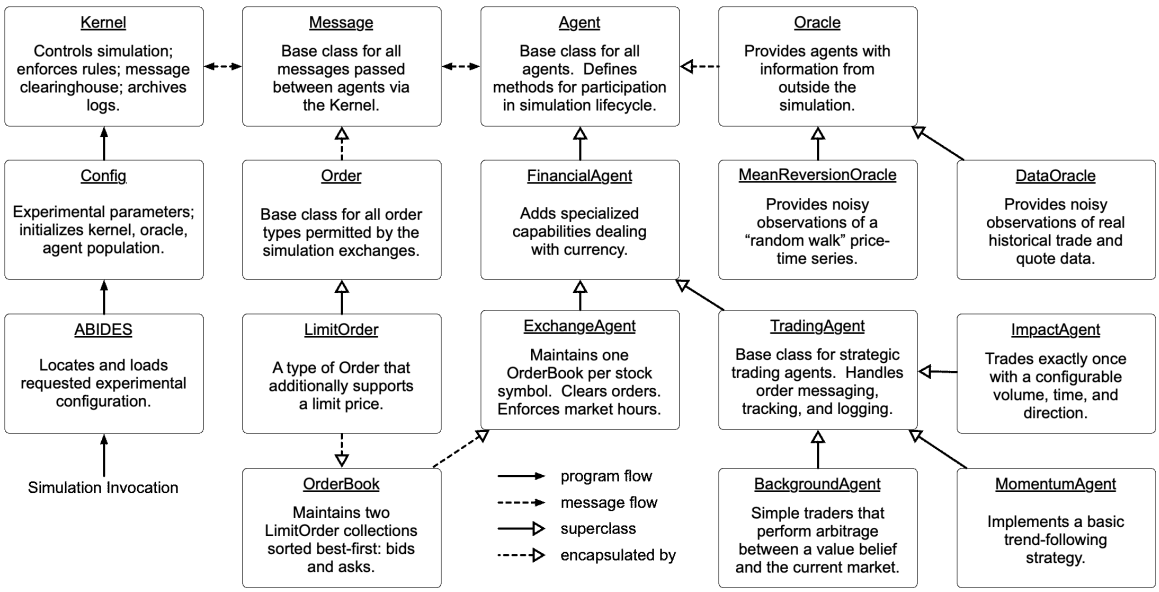


Figure 2: Class relations within the ABIDES simulation framework.

that incorporates trading rules (albeit simple strategies) and reproduces the changes in the market by executing buy and sell orders from the order book. In ABIDES we represent individual investors as agents, which is similar to the approaches of Levy et al [15] and Jacobs et al [11]. JLMSim is implemented in C++ and runs at a few thousands events per second on a 2.4 GHz PC. ABIDES is implemented in Python (which offers MATLAB-like functionality) and can run over 10K events per second on a similar 2.4GHz processor. Unlike ABIDES, JLMSim does not provide interfaces to implement complex trading strategies or learning agents, and is currently not available as open source software.

4 ABIDES ARCHITECTURE

The ABIDES framework includes a customizable configuration system, a simulation kernel, and a rich hierarchy of agent functionality, partially illustrated in Figure 2.

4.1 Functions and Features of the ABIDES Kernel

ABIDES is built around a discrete event-based kernel [4] which is required in all simulations. All agent messages must pass through the kernel's event queue. The kernel supports simulation of geography, computation time, and network latency. It also acts as enforcer of simulation physics, maintaining the current simulation time, tracking a separate "current time" for each agent, and ensuring there is no inappropriate time travel. Some key features of the ABIDES kernel include:

- **Historical date simulation** All simulation occurs on a configurable historical date. This permits "real" historical information to be seamlessly injected into the simulation at appropriate times when required for a particular experiment. ABIDES can currently be configured to run a market replay with a liquidity injection agent placing orders from historical data, a historical agent-based simulation in which background agents can receive noisy observations of historic transactions, or an ahistoric agent-based simulation in which fundamental stock values follow a mean-reverting or other mathematical process.
- **Nanosecond resolution**: Because we seek to emulate real markets, we simulate time at the same resolution as an example exchange: the NASDAQ. All simulation times are represented as timestamp objects with nanosecond resolution. This allows a mixture of agents to participate in the simulation on very different time scales with minimal developer overhead. In the unlikely case that multiple events occur in the same nanosecond, they are handled in order of event object creation.
- **Global Virtual Time (GVT)**: GVT is the latest simulated time for which all messages are guaranteed to have been processed. The kernel tracks GVT as the simulation progresses. Since the simulation is single-threaded (although agents act in *simulated* parallel), it is not possible for any agent to affect the past. GVT may thus simply advance to the delivery timestamp of each dequeued message in chronological order and remain monotonically non-decreasing. It is usually the case that GVT advances much more quickly than wall clock time, but for very complex scenarios, it may not. The GVT value is not available to the agents.
- **Current time per agent**: The kernel tracks a "current time" per individual participating agent which is incremented upon return from any activation of that agent. In situations where the current time for the agent is "in the future" (i.e., larger than GVT), the kernel will delay delivery of messages or wakeup calls to this agent until GVT catches up.

- **Computation delay**: The kernel stores a computation delay per agent which is added to the agent's "current time" after each activity. The delay is also added to the sent time and delivery time of any outbound message from an agent to account for the agent's computation effort. Agents may alter this computation delay to account for different sorts of computation events, or the simulation can be configured to measure and use the *real* computation time of each agent action.
- **Configurable network latency**: The kernel maintains a pairwise agent latency matrix and a realistic cubic network jitter model which are applied to all messages between agents. This permits simulation of network conditions and agent location, including co-location.
- **Deterministic but random execution**: The kernel accepts a single, global pseudo-random number generator (PRNG) seed at initialization. This PRNG is then used to generate seeds for an individual PRNG object per agent, which must rely solely on that object for stochastic methods. Since our system is currently single-threaded, this allows the entire simulation to be guaranteed identical when the same seed is initialized within the same experimental configuration. This would not ordinarily permit the desired A/B testing, because the "agent of change" might consume an additional pseudo-random number from the sequence and thus change the stochastic source for all subsequent agents. Because of our careful use of the primary PRNG only to generate subsidiary PRNGs per agent, the "agent of change" in an ABIDES A/B experiment will not alter the set of pseudo-random numbers given to any other agent throughout the simulation, even if it uses more or fewer such inputs for its changed activity. In this way, changes in the behavior of other agents will be caused by a changed simulation environment (e.g. stock prices) and not simple stochastic perturbation.

During a simulation, the kernel follows a typical series of life cycle phases: kernel initialization, kernel start, event queue processing, kernel stop, and kernel termination. All except the event queue processing phase consist entirely of sending a corresponding event notification to all agents.

While processing the event queue, the kernel extracts the next scheduled event in chronological order and advances the global virtual time (GVT) to match it. Recall that each agent has an individual "current" time representing the conclusion of its most recent activity. If the target agent is still in the future with respect to GVT, the event is rescheduled for the target agent's current time, placed back into the priority queue, and the kernel moves to the next chronological event. Otherwise, the target agent's current time is advanced to the GVT and the event is dispatched to the agent. When the agent's event handling method returns, the agent's current time is advanced by its computation delay.

Agents may request several critical functions from the kernel: To send a message to another agent; To schedule a wakeup call for some future time; And to learn the simulation identifier of another agent of a specific type (for example, a stock exchange). Messages will be sent as of the sender's current time, plus its computation delay, plus an optional additional delay upon request. Message receipt will be scheduled based on the send time plus network latency and jitter. Agents may learn the numeric identifier of other agents, but may never receive a *reference* to another agent (as this could allow bypassing the kernel in the future).

4.2 The Agent Class

All participants in a simulation must inherit from a base agent class, which implements a number of required methods that allow participation in the full life cycle of the simulation.

The simulation lifecycle methods for kernel initialization, kernel start, kernel stop, and kernel termination must be supported by all simulation agents and will be called exactly one time per agent by the kernel. The order in which agents are activated in each life cycle phase is arbitrary. The basic agent class provides sensible default behavior for each phase.

Two simulation activation methods, for receipt of messages and wakeup calls, must also be supported by all simulation agents. These are called repeatedly by the kernel in order of increasing delivery timestamp of queued messages and wakeup calls. The basic agent class handles these methods by simply updating its internal current time and displaying an informative message.

While not required by the simulation kernel, the basic agent class also provides functionality for fine resolution timestamped activity logging and serialization to disk.

4.3 The Exchange Agent Class

The provided exchange agent inherits from the basic agent class and represents a stock exchange such as NASDAQ. The message protocols supported by this agent are based on NASDAQ's published ITCH and OUCH protocols. [21, 22] The exchange is initialized with market opening and closing times, which it will enforce. These are not required to match the simulation start and stop times. The exchange agent is not privileged in any way; it must participate in the simulation just as any other agent.

The exchange agent understands how to respond to these types of messages that are specific to the operation of a financial market:

- **Market Open Time**: Returns the timestamp at which the exchange will begin processing order-related messages.
- **Market Close Time**: Returns the timestamp at which the exchange will stop processing order-related messages.
- **Query Last Trade**: Returns the last trade price for a requested symbol. Until the first trade of the day, the exchange reports the oracle open price (historical or generated data) as the "last trade price". The exchange does not yet implement the opening cross auction.
- **Query Spread / Depth**: Returns a list of the N best bid and best ask prices for a requested symbol and the aggregate volume available at each price point. With a requested depth of one, this is equivalent to querying "the spread".
- **Limit Order**: Forwards the attached limit order to the requested symbol's order book for matching or acceptance. Agents currently simulate market orders using a limit order with an arbitrarily high or low limit price.
- **Cancel Order**: Forwards the attached order to the requested symbol's order book to attempt cancellation.

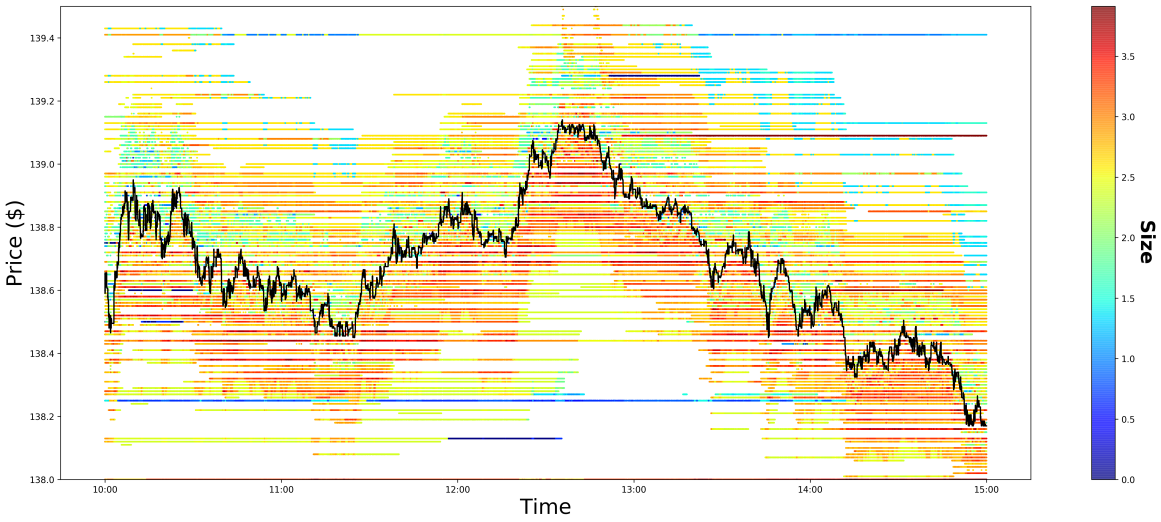


Figure 3: Example simulation of IBM stock for 2019-06-28.

Outside of market hours, the exchange will only honor messages relating to market hour inquiries and final trade prices (after the close). The exchange sends a "market closed" message to any agent which contacts it with disallowed messages outside of market hours.

The exchange agent demonstrates one use of the inbuilt Kernel logging facility, recording either the full order stream or snapshots of its order books at a requested frequency, enabling extremely detailed visualization and analysis of the order book at any time during simulation. For example, Figure 3 shows a "market replay" style simulation of IBM stock on June 28, 2019, in which autonomous trading agents can also participate and affect the market.

4.4 The Order Book

Within an Exchange Agent, an order book tracks all open orders, plus the last trade price, for a single stock symbol. All order book activity is logged through the exchange agent. The order book implements the following functionality:

- **Order Matching** Attempts to match the incoming order against the appropriate side of the order book. The best price match is selected. In the case of multiple orders at the same price, the oldest order is selected.
- **Partial Execution** Either the incoming order or the matched limit order may be partially executed. When the matched limit order is partially executed, the order is left in the book with its quantity reduced. When the incoming order is partially executed, its quantity is reduced and a new round of matching begins. Participants receive one "order executed" message, sent via the exchange, per partial execution noting the fill price of each. When the incoming order is executed in multiple parts, the average price per share is recorded as the last trade price for the symbol.
- **Order Acceptance** When the incoming limit order has remaining quantity after all possible matches have been executed, it will be added to the order book for later fulfillment, and an "order accepted" message will be sent via the exchange.
- **Order Cancellation** The order book locates the requested order by unique order id, removes any remaining unfilled quantity from the order book, and sends an "order cancelled" message via the exchange.

One might reasonably expect the order book in a market simulation to include a model for slippage. We assert that our platform produces realistic slippage naturally, without the need for such a model. Orders directed to the exchange suffer dynamic computation and network delays, during which time other orders are being executed.

4.5 The Trading Agent Class

The provided trading agent inherits from the basic agent class and represents the base class for a financial trading agent. It implements a number of additional features upon which subclassed strategy agents may rely:

- **Portfolio** The base trading agent maintains an equity portfolio including a cash position. It automatically updates this portfolio in response to "order executed" messages.
- **Open Orders** The trading agent keeps a list of unfilled orders that is automatically updated upon receipt of "order executed" and "order cancelled" messages, and when new orders are originated.
- **Last Known Symbol Info** The trading agent tracks known information about all symbols in its awareness, including the most recent trade prices, daily close prices (after the close), and order book spread or depth. These are automatically updated when receiving related messages.
- **Market Status** Upon initially waking at simulation start, the trading agent automatically locates an exchange agent, requests market open and close times, and schedules a second wakeup call for the time of market open. It also maintains

and provides a simple "market closed" flag for the benefit of subclassing agents.
- **Mark to Market** The trading agent understands how to mark its portfolio to market at any time, using its most current knowledge of equity pricing. It automatically marks to market at the end of the day.
- **Messages** The trading agent knows how to originate all of the messages the exchange understands, and to usefully interpret and store all of the possible responses from the exchange.
- **Logging** The trading agent logs all significant activity: when it places orders; receives notification of order acceptance, execution, or cancellation; when its holdings change for any reason; or when it marks to market at the end of the day.

5 ABIDES IMPLEMENTATION

The ABIDES simulator is implemented using Python, currently 3.7, and the data analytical libraries NumPy [24], and Pandas [18]. It makes use of a virtual environment to provide platform independence and provides a straightforward deployment. It is seamlessly built to facilitate quick reconfiguration of varying agent populations, market conditions, exchange rules, and agent hyperparameters.

Basic execution of the simulation can be as simple as: `python abides.py -c config`, where `config` is the name of an experimental configuration file. Additional command line parameters are forwarded to the configuration code for processing, so each experimental configuration can add its own required parameters to a standard interface. Complex experimental configuration can be performed directly within the config file since it is simply Python code, however the inclusion of command line arguments is beneficial for coarse grain parallelization of multiple experiments of the same type, but with varied simulation parameters.

A typical configuration file will specify a historical date to simulate and a simulation start and stop time as nanosecond-precision timestamps. It will then initialize a population of agents for the experiment, configuring each as desired. For example, an experiment could involve 1,000 background agents (perhaps Zero Intelligence agents or Heuristic Belief Learning agents), 100 high-frequency trading agents, and one agent designed to create a market price impact by placing a very large order, with various initialization parameters to control their behavior. Each agent will at least be given a unique identifier and name. The configuration file will also construct a latency matrix (pairwise between all agents at nanosecond precision) and cubic network jitter model which will be applied to all inter-agent communications. If a "data oracle", a utility with access to a data source outside the simulation, is required for the experiment, the configuration file will initialize one. Finally a simulation kernel will be initialized and run, passing it the agent population, oracle, and other simulation parameters.

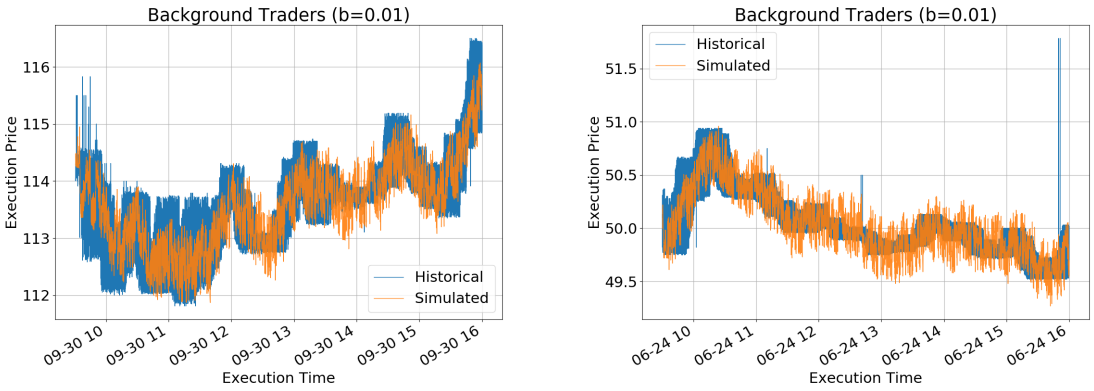
In its current form, ABIDES completes simulation of 1,000 typical "ping pong" agents that each send a single message to all other agents, and then respond to all incoming messages (for a total of approximately two million messages) in 3 minutes 18 seconds including all setup, overhead, and teardown, at a kernel processing rate of 10,230 events per second. Because the simulation is single-threaded, as many trials can be run simultaneously as available memory and processing cores permit with relatively little performance degradation. For example, running two of the above ping pong experiments simultaneously on the same computer increases the total runtime by only four seconds. Similarly, a simulation of 1,000 Zero Intelligence (ZI) agents participating in a full day of trading at a NASDAQ-like exchange, with a mean market inter-arrival time of approximately one second, is completed in an average of 36 seconds total runtime. All simulation runtime data was collected on a notebook computer with a 2.4 GHz Intel Core i5 processor and 16 GB RAM.

Note that there is nothing finance-specific about the bootstrapper, configuration template, simulation kernel, or the basic agent class. All are appropriate for use in any discrete event simulation.

5.1 Example: A Momentum Trading Agent

To highlight the simplicity of creating a functional trading agent in our simulated environment, we present the code for a basic momentum trader. It wakes each minute during the day, queries the last trade price, projects a future price using linear regression over a configurable last N data points, and places a market order based on this projection. Following is the complete source, excluding import statements:

```
class MomentumAgent(TradingAgent):

  def __init__(self, id, name, symbol, startingCash,
               lookback):
    super().__init__(id, name, startingCash)

    self.symbol = symbol
    self.lookback = lookback
    self.state = "AWAITING_WAKEUP"

    self.trades = []


  def wakeup (self, currentTime):
    can_trade = super().wakeup(currentTime)

    if not can_trade: return

    self.getLastTrade(self.symbol)
    self.state = "AWAITING_LAST_TRADE"


  def receiveMessage (self, currentTime, msg):
    super().receiveMessage(currentTime, msg)

    if self.state == "AWAITING_LAST_TRADE" and \
         msg.type == "QUERY_LAST_TRADE":

      last = self.last_trade[self.symbol]
      self.trades = (self.trades + [last])[:self.lookback]

      if len(self.trades) >= self.lookback:
        m, b = np.polyfit(range(len(self.trades)),
                          self.trades, 1)
        pred = self.lookback * m + b

        holdings = self.getHoldings(self.symbol)

        if pred > last:
```



(a) IBM: September 30, 2008

(b) MSFT: June 24, 2016

Figure 4: Simulated trades versus historical trades on two days.

```
            self.placeLimitOrder(self.symbol, 100-holdings,
                                 True, self.MKT_BUY)
        else:
            self.placeLimitOrder(self.symbol, 100+holdings,
                                 False, self.MKT_SELL)

    self.setWakeup(currentTime + pd.Timedelta("1m"))
    self.state = "AWAITING_WAKEUP"
```

6 CASE STUDY: BACKGROUND AGENTS

One long-term goal is to produce realistic but possibly noisy re-simulations of particular days in history to play out various "what if" scenarios. The idea is to populate the simulation with a large number of trading agents that provide a realistic environment into which experimental agents can be injected.

Our initial effort towards this goal involves the introduction of a data oracle with access to fine-resolution historical trade information, and the creation of a set of "background" agents which are able to request a noisy observation of the most recent historical trade as of the agent's current simulated time. The approach is meant to reproduce the behavior of a trader whose beliefs regarding the fundamental value of a stock are informed by interpretations of news and other incoming information. It was inspired by the concept of a stock's "fundamental value" as used in the work of Wang and Wellman. [28] Our approach is similar, but it uses historical data as a baseline rather than a mean-reverting stochastic process.

As background agents, we have implemented two common baseline agents from the continuous double auction literature. The Zero Intelligence (ZI) trader [10] submits random bids and offers to the market, usually drawn from some stochastic distribution around a central value belief for the underlying instrument. The Heuristic Belief Learning (HBL) agent [8] maintains a Bayesian belief distribution for likelihood of successful order transaction by offered price, and uses this to place orders which maximize expected surplus. HBL is based on the earlier GD agent [9], named for its authors Gjerstad and Dickhaut. We implement HBL as described by Wang and Wellman. [28] Each background agent trades only a single symbol on a single exchange.

Figure 4 compares the behavior of 100 background agents interacting in ABIDES with the actual intra-day price on two separate days in history. Ideally, we will see a price history that closely resembles the day in history, with similar statistical properties.

7 CASE STUDY: MARKET IMPACT

One area in which we believe simulation can add significant value to the current state of knowledge in finance is more accurate models of the market impact of large trades. Each order placed at the exchange potentially "moves the market" due to the nature of the market microstructure within the order book: arriving orders can add liquidity at a better price, altering the spread; or can match existing orders and remove liquidity from the market. See Figure 5 for an example of mechanical market impact.

Models that rely on historical data encounter limitations stemming from the inability to repeat history while introducing an experimental change and allowing subsequent events to be *altered* by that change. Models can attempt to compare "similar" days in history, but no two market days are ever the same.

If one could instead create a multi-agent simulation of a particular date in history such that a near approximation of historical trades emerged in the absence of any significant change, but the trading agents would realistically react to any such changes, a more accurate understanding of large trade impact could be attained. Here we present a preliminary investigation of this idea.

We begin each simulation with a population of background agents and at least one exchange agent. For this experiment, we add a single experimental impact agent, which simply places a single large market order at a predetermined time of day. The experimental parameter for the agent is its "greed"; that is, the proportion of available order book liquidity near the spread it consumes at the time of trade. For example, a long impact agent with $greed = 0.1$ will place a market buy order for 10% of the shares on offer.

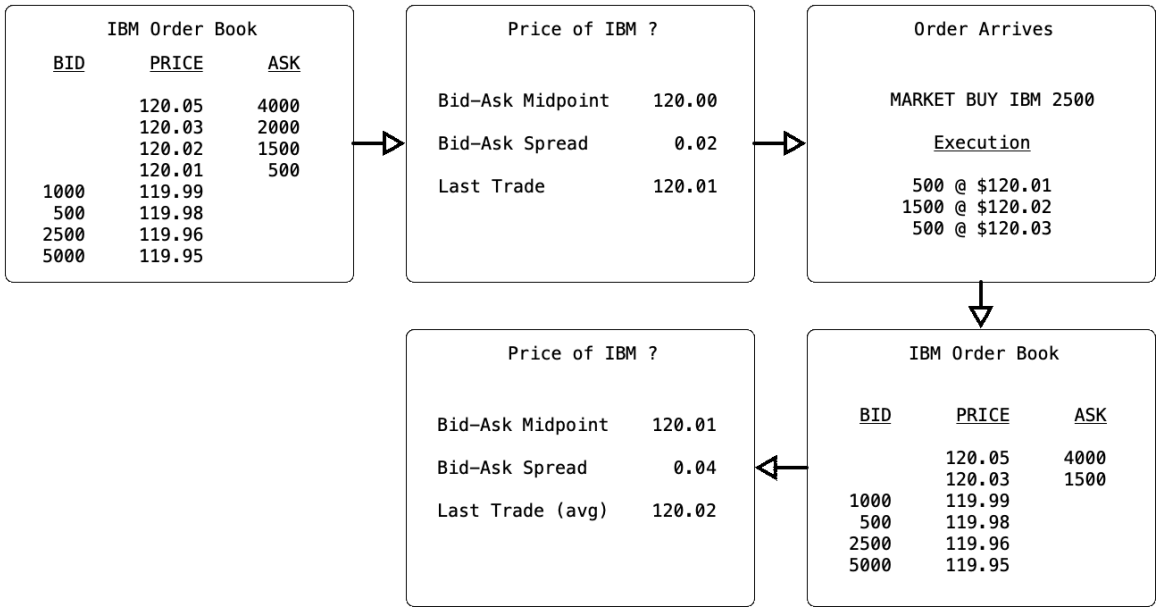


Figure 5: Example of mechanical market impact.

Our experiment includes 100 background agents and one exchange agent handling an order book for a set of symbols including IBM. In Figure 6, the blue line represents each trade made by our population of background agents in the absence of an impact trader. The orange line shows each trade made by the simulated trading agents given the introduction of a single impact agent with varying "greed", acting one time with one trade at 10:00 AM on September 30, 2008. Both series are smoothed to improve visibility of the differences.

The impact trader has a clear effect on the market, despite the background agents' central tendency to arbitrage the price toward historical levels, and the impact grows larger proportionally with its market bid size. The change is particularly noticeable in the cyclical peaks of the auction. Due to the price elevation it caused, the impact trader's total profit increased with the size of its bid from an average of \$2,633 with $greed = 0.3$ to \$12,502 with $greed = 1.9$. However its profit per share declined from \$2.14 to \$1.60. We found a correlation between profit per share and trade size of $r = -0.31$ across sixty experimental trials.

It is useful to consider these market impacts in aggregate across multiple experimental examples. ABIDES makes it easy to produce study plots from logged simulation data. Figure 7 shows a time-aligned event study of many impact trades at different times, on different days, to illustrate the range of likely price effects after the time of impact.

8 OTHER ONGOING WORK

The ABIDES simulation platform is already supporting promising research efforts both inside and outside the financial domain. Ongoing research using ABIDES has been presented at the ICML and NeurIPS AI in Finance Workshops, including: Investigating whether exchange-traded funds (ETFs) worsen the spread of volatility events in a market; Whether market replay or interactive multiagent systems are better to evaluate trading strategies [3]; And what stylized facts about markets are most appropriate to judge the realism of a simulated market [26].

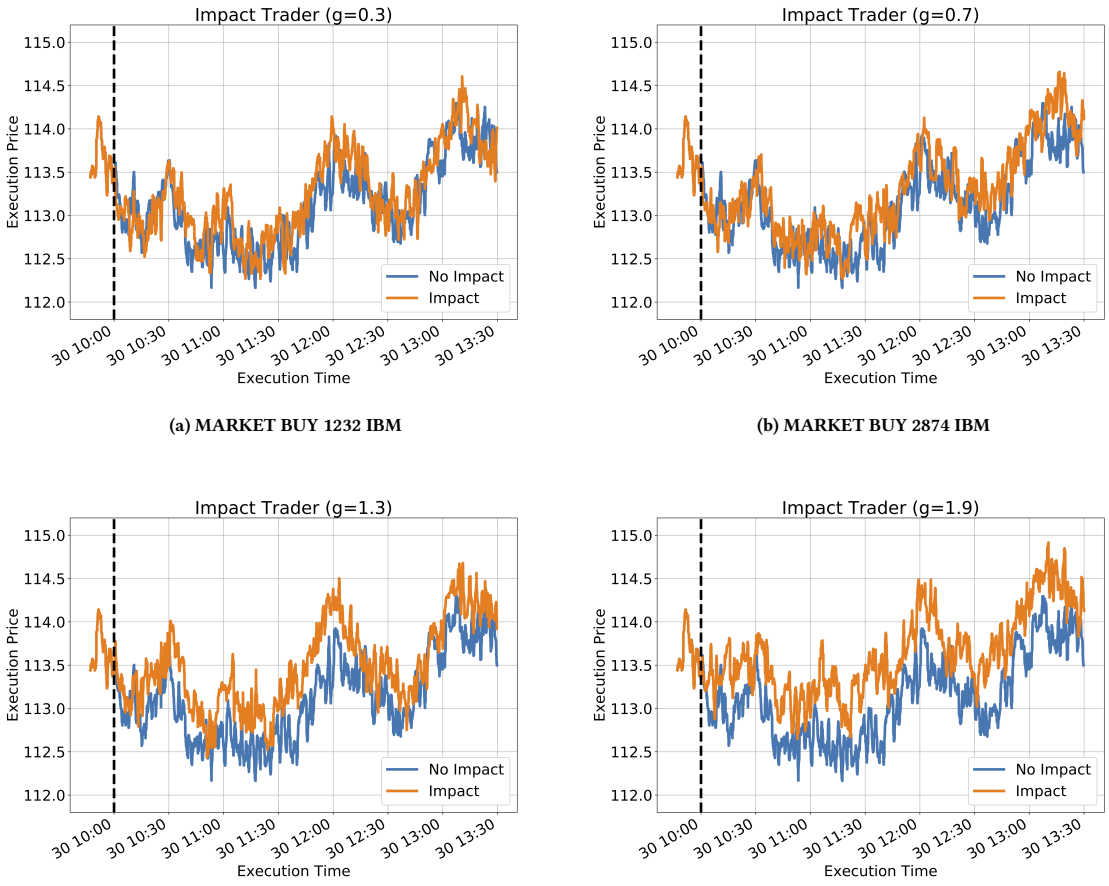
ABIDES is also supporting research in non-financial application fields, including as the development platform of a new protocol for secure multiparty artificial intelligence. This complex technique combines secure multiparty communication (SMC), differential privacy, and federated learning to permit a population of clients to collaboratively leverage their private data to learn a superior shared model without revealing training data or requiring any trust among the parties.

As an example, Figure 8 shows the empirical effect the client population size and differential privacy factor (epsilon) have on the ability to learn an accurate shared solution to the census income data set [1]. ABIDES has been used to rapidly iterate and validate the proposed protocol, and to evaluate its computation and communication overhead. Obtaining such results in simulation on a single local processor, for a distributed protocol representing a thousand remote clients (e.g. cell phones), represents a significant time and cost savings.

9 CONCLUSION

We presented the design and implementation of ABIDES, a high-fidelity equity market simulator. ABIDES provides an environment within which complex research questions regarding trading agents and market behavior can be investigated. We discussed additional ongoing research being supported by the ABIDES platform both within and outside the financial domain.

The simulation is demonstrated in two case studies. The first case study shows how previous intra-day transaction histories are closely reproduced by a population of interacting background



(a) MARKET BUY 1232 IBM

(b) MARKET BUY 2874 IBM

(c) MARKET BUY 5338 IBM

(d) MARKET BUY 7801 IBM

Figure 6: Market impact of trades on the same date at 10:00 AM.

trading agents communicating with an exchange agent. These background agents are designed to provide a realistic market environment into which experimental agents can be injected. The second case study illustrates how large market orders impact simulated prices not just immediately, but for a significant period after the order arrives at the exchange. It is also intended to demonstrate the experimental potential of the ABIDES platform.

We now have a robust simulation environment in which to develop and experiment with more complex trading agents, including those based on approaches in machine learning and artificial intelligence.

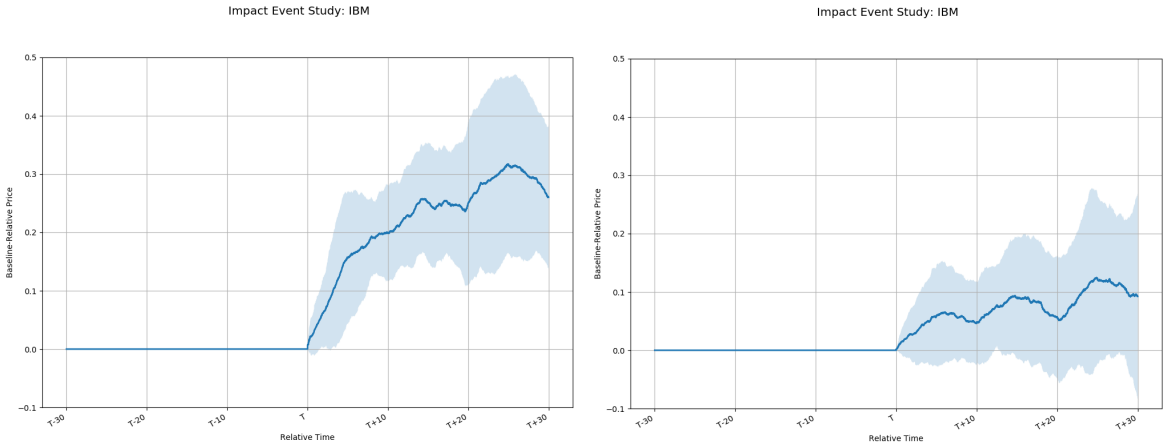
10 OPEN SOURCE ACCESS AND LICENSE

ABIDES is available under the BSD 3-clause license at GitHub: `https://github.com/abides-sim/abides`.

ACKNOWLEDGMENTS

This material is based upon research supported by the National Science Foundation under Grant No. 1741026 and by a JP Morgan PhD Fellowship.

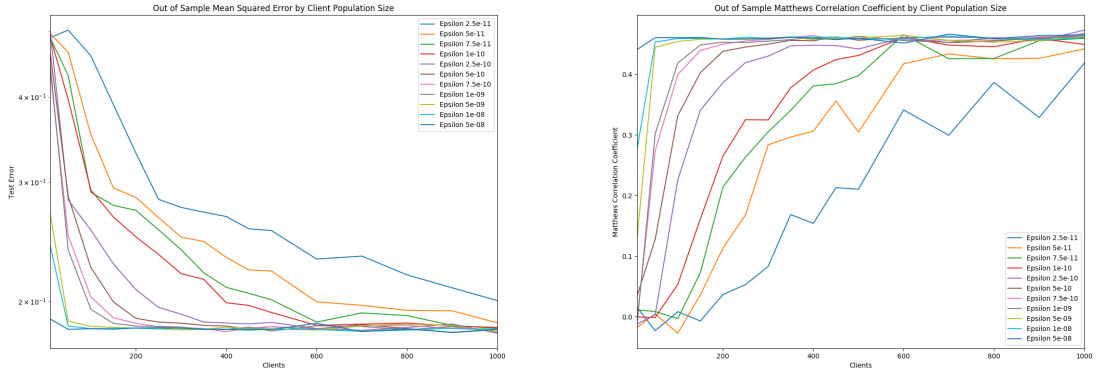
We gratefully thank Danial Dervovic, Joshua Lockhart, Mahmoud Mahfouz, and Svitlana Vyetrenko for their technical contributions to ABIDES and for the improved limit order book visualization code.



(a) Impact agent with greed 0.5

(b) Impact agent with greed 0.1

Figure 7: Market impact event studies.



(a) Out of Sample error of shared model.

(b) Out of sample accuracy of shared model.

Figure 8: Evaluating a new secure multiparty federated learning protocol in simulation.

under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

REFERENCES

[1] A. Asuncion and D.J. Newman. 2007. UCI Machine Learning Repository. http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html

[2] Robert Axelrod. 1997. Advancing the art of simulation in the social sciences. In *Simulating social phenomena*. Springer, 21–40.

[3] Tucker Hybinette Balch, Mahmoud Mahfouz, Joshua Lockhart, Maria Hybinette, and David Byrd. 2019. How to Evaluate Trading Strategies: Single Agent Market Replay or Multiple Agent Interactive Simulation? *arXiv preprint arXiv:1906.12010* (2019).

[4] Jerry Banks. 1998. *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

[5] Fischer Black and Myron Scholes. 1973. The pricing of options and corporate liabilities. *Journal of political economy* 81, 3 (1973), 637–654.

[6] K. M. Chandy and J. Misra. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Commun. ACM* 24, 4 (April 1981), 198–205.

[7] D Freidman. 1993. The Double Auction Market Institution: A Survey. *The Double Auction Market Institutions, Theories and Evidence, Addison Wesley* (1993).

[8] Steven Gjerstad. 2007. The competitive market paradox. *Journal of Economic Dynamics and Control* 31, 5 (2007), 1753–1780.

[9] Steven Gjerstad and John Dickhaut. 1998. Price formation in double auctions. *Games and economic behavior* 22, 1 (1998), 1–29.

[10] Dhananjay K Gode and Shyam Sunder. 1993. Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *Journal of political economy* 101, 1 (1993), 119–137.

[11] Bruce I. Jacobs, Kenneth N. Levy, and Harry M. Markowitz. 2004. Financial Market Simulation. *The Journal of Portfolio Management* 30, 5 (2004), 142–152. https://doi.org/10.3905/jpm.2004.442640 arXiv:https://jpm.pm-research.com/content/30/5/142.full.pdf

[12] Bruce I. Jacobs, Kenneth N. Levy, and Harry M. Markowitz. 2010. Simulating Security Markets in Dynamic and Equilibrium Modes. *Financial Analysts Journal* 66, 5 (2010), 42–53. https://doi.org/10.2469/faj.v66.n5.7 arXiv:https://doi.org/10.2469/faj.v66.n5.7

[13] David R. Jefferson and Henry Sowizral. 1985. Fast concurrent simulation using the time warp mechanism. In *Distributed Simulation 1985*. Simulation Council Proceedings, Vol. 15. Society for Computer Simulation (SCS), 63–69.

[14] B. LeBaron. 2001. A builder's guide to agent-based financial markets. *Quantitative Finance* 1, 2 (2001), 254–261. https://doi.org/10.1088/1469-7688/1/2/307 arXiv:https://doi.org/10.1088/1469-7688/1/2/307

[15] Moshe Levy, Haim Levy, and Sorin Solomon. 1994. A microscopic model of the stock market: Cycles, booms, and crashes. *Economics Letters* 45, 1 (1994), 103–111. https://EconPapers.repec.org/RePEc:eee:ecolet:v:45:y:1994:i:1:p:103-111

[16] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. "MASON": A Multiagent Simulation Environment. *SIMULATION* 81 (2005), 517–527.

[17] C. M. Macal and M. J. North. 2009. Agent-based modeling and simulation. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*. 86–98. https://doi.org/10.1109/WSC.2009.5429318

[18] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[19] Robert C Merton. 1973. Theory of rational option pricing. *The Bell Journal of economics and management science* (1973), 141–183.

[20] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. 1996. The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations. citeseer.ist.psu.edu/minar96swarm.html

[21] NASDAQ OMX Group. [n.d.]. NASDAQ TotalView - ITCH 5.0. http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTVITCHSpecification.pdf. Accessed: 2018-10-25.

[22] NASDAQ OMX Group. [n.d.]. O*U*C*H Version 4.2. http://www.nasdaqtrader.com/content/technicalsupport/specifications/TradingProducts/OUCH4.2.pdf. Accessed: 2018-10-25.

[23] Ashkan Negahban and Levent Yilmaz. 2014. Agent-based simulation applications in marketing research: an integrated review. *Journal of Simulation* 8, 2 (2014), 129–142.

[24] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[25] Leigh Tesfatsion. 2002. Agent-based computational economics: Growing economies from the bottom up. *Artificial life* 8, 1 (2002), 55–82.

[26] Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Hybinette Balch. 2019. Get Real: Realism Metrics for Robust Limit Order Book Market Simulations. *arXiv preprint arXiv:1912.04941* (2019).

[27] Jianling Wang, Vivek George, Tucker Balch, and Maria Hybinette. 2017. Stockyard: A discrete event-based stock market exchange simulator. In *Simulation Conference (WSC), 2017 Winter*. IEEE, 1193–1203.

[28] Xintong Wang and Michael P Wellman. 2017. Spoofing the limit order book: An agent-based model. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 651–659.

[29] Michael P Wellman. 2006. Methods for empirical game-theoretic analysis. In *AAAI*. 1552–1556.

[30] Matthew Zook and Michael H Grote. 2017. The microgeographies of global finance: High-frequency trading and the construction of information inequality. *Environment and Planning A: Economy and Space* 49, 1 (2017), 121–140. https://doi.org/10.1177/0308518X16667298