University College London
Department of Computing

# BUCLSE

Henry Ashton

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University College London , June 2019

# Contents

**1 Platform** **1**

1.1 Description . . . . . 1

1.2 Introduction to BUCLSE . . . . . 2

1.3 Timer and Messenger . . . . . 3

1.4 Exchange . . . . . 5

1.5 Traders . . . . . 6

1.5.1 Trader Zoo - Supply Demand type . . . . . 7

1.5.2 Trader Types - Price Series type . . . . . 8

1.5.3 Noise Trader . . . . . 9

1.5.4 Cont Imbalance . . . . . 9

1.6 Supply and Demand . . . . . 9

1.6.1 Supply Demand setup 1: BSE . . . . . 10

1.6.2 Supply Demand setup 2: Wang Wellman . . . . . 11

1.7 Market Session Objects . . . . . 11

1.7.1 Market session setup 1: BSE . . . . . 11

1.7.2 Market session setup 2: Wang Wellman . . . . . 12

1.8 Reinforcement Learning adaptation . . . . . 12

1.9 Future development . . . . . 13

**Bibliography** **13**

# List of Tables

# List of Figures

1.1 A message route map for the BUCLSE system. In a single core setup, the timer object is common to all objects, obviating its need to send messages. As the system is developed for multi core setups and asynchronous operation, time updates via message will become necessary. . . . . 4

1.2 Equilibrium price defined over time, as defined by the intersection of demand and supply curves at any moment in time. In this figure, the demand and supply curves are taken as of t=50, hence their intersection is in line with green curve at t=50. . . . . 10

# Chapter 1

# Platform

## 1.1 Description

The study of emergent trading behaviour necessitates an environment which provides feedback to the trading agents within it. That is to say, the actions of the every trading agent, have the potential to affect the environment and thereby the actions of the other agents. As mentioned before, this precludes the use of historic pricing data as a way to explore the problem. In machine learning terminology, learning must be done predominantly online (as apposed to a batch setting).

This is a two edged sword. On one hand data can be obtained for free; it is generated by the trading environment itself. Any enquiries about the cost of acquring multi-level orderbook trading data from external vendors will convince the reader of the attraction of this. It is kept under lock and key by a small number of guardians and the cost of the key is prohibitive for most independent researchers. On the other hand there are two problems with a market simulation approach, one theoretical and one mechanical. Firstly the theoretical problem is that there is no guarantee that any data produced in a market simulation is realistic and of any practical use. Thankfully, since the objective of this research is not to find magical money making machines but to investigate the emergence of illegal behaviour and how to arrest it, we

don't need to worry too much about this problem [1].That said, the trading environment will work as realistically as possible and is designed to have the flexibility to produce a realistic simulation. The parameter search to achieve this is left for another research project.

The mechanical problem relates to the efficiency of the trading environment and is one which we discuss at greater length in this chapter. If data generation and learning are online, can we make the environment sufficiently efficient enough to generate data at the speed and volume which modern machine learning methods require? Benchmark Reinforcement Learning problems enjoy fast, lightweight toy environments as a given. Before we even think about doing any machine learning our approach requires a significant amount of effort to make the environment robust, efficient and fast [2].

## 1.2 Introduction to BUCLSE

BUCLSE was built on the back of the BSE ([Cliff, 2018]) to provide a minimal trading environment to investigate the emergence of illegal trading behaviour. The BSE was built as a teaching aid for a university algorithmic trading course. As such it was a convenient starting point for BUCLSE. It is programmed in Python 3 and suitable for versions 3.7 and beyond.

Within a multi agent trading environment there are four main elements to consider:

1. The **Traders** - Variable in quantity and type, these are the agents that make trading decisions within the simulation.
2. The **Exchange** - This provides the price matching mechanism through which traders transact.
3. **Supply and Demand** dynamics for the asset - This can be described through supply and demand curves, or an underlying fundamental price for an asset, but some of the trader's need external information in order to trade and drive the simulation.

[1] A robust finding of emergence should translate across all market places
[2] aka 'engineering', a vulgar term for many in academia

4. The experiment **Controller** - The most varied of the four elements, it is the mechanism that coordinates the previous three elements into a simulation.

Since Python is an object oriented language, these elements have natural implementations as objects and appear as such in BUCLSE.

A point of differentiation in BUCLSE is the method through which the elements interact. From inception BUCLSE has been designed with one eye towards multi-core operability with the introduction of timer and messenger objects. As such we add two more necessary elements to the preceding list:

1. A **Message** system - The mechanism through which the various simulation objects communicate with each other.
2. A unified **Timer** - Time needs to be agreed upon between all parties since this is an online, sequential simulator. The passage of time is controlled by the Controller.

## 1.3 Timer and Messenger

The messenger object is the conduit through which all elements of the simulator communicate with each other. Enforcing inter object communication through a messenger system is desirable because it removes the limiting factor of a single computer's memory or CPU on the simulation. Thus BUCLSE has the potential to be scaled to an arbitrary number of independent cores - a multi agent simulator using multiple computing units. A route map of the system's messages is shown in figure 1.1 on page 4.

The messenger object is in effect a message server which maintains a list of subscribers. Traders subscribe and submit messages to the messenger which in turn sends them on to the exchange and vice versa. Likewise the Controller can coordinate its traders through the emission of messages. Thus, every element of the BUCLSE becomes independent of every other element by having a method of receiving and sending messages. Whilst in the single core environment this appears to be an over elaborate way of getting two objects within the same memory

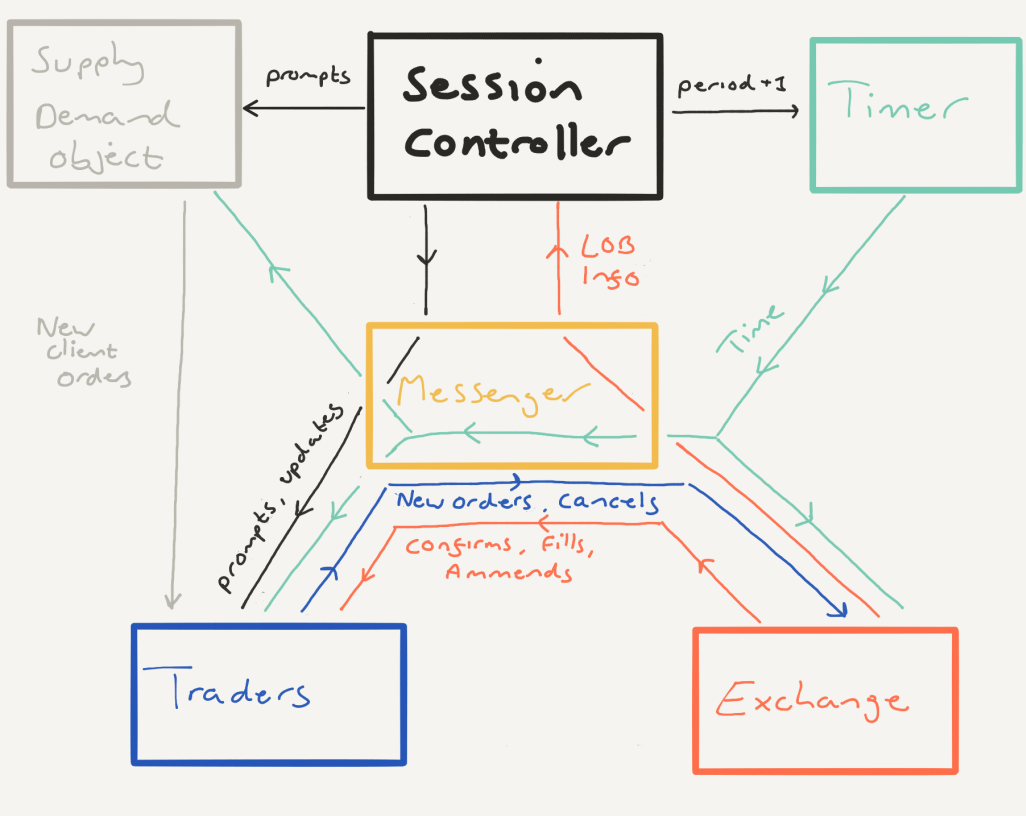


Figure 1.1: A message route map for the BUCLSE system. In a single core setup, the timer object is common to all objects, obviating its need to send messages. As the system is developed for multi core setups and asynchronous operation, time updates via message will become necessary.

to communicate with each other, the log of passed messages becomes in itself a very useful debugging tool. [3]

In initial tests for a multi-core setup, we used the MQTT message protocol because it is lightweight, easy to understand and is freely supported in Python. A further attraction to using a messaging system in the environment is that traders and exchanges communicate with each other using a standardised messaging protocol called FIX (Financial Information eXchange). Further iterations of BUCLSE might adopt this protocol, thereby facilitating the testing of actual trading algorithms within the simulation environment.

The timer object is a simple object which is common across all elements within BUCLSE [4] It allows all objects in the environment to agree about what 'time' it is within the simulation. Unlike using a simply clock however, we are able to control the passage of time within the environment. This means differing computation time is not an issue once the environment is distributed across multiple cores.

## 1.4 Exchange

The Exchange object maintains a limit order book (henceforth LOB) for a single asset. Individual orders at a price are given a price based on FIFO ordering. Orders can be of any integer quantity.

For ease of communication, an Order object has been created which contains the vital information of an order and is used extensively in the communication between trader and exchange. The exchange can receive two types of message, "New Order" and "Cancel". It can send "Fill", "Ammend"[5] and "Confirm" messages.

In the event of a New Order message, the exchange replies with a quote ID and then checks

[3] And as the author found out, multi agent simulations are finickity to develop

[4] Time changes can be communicated and coordinated through messaging in the true decentralised implementation

[5] Orders with heterogenous quantity necessitate the ability to ammend orders on the orderbook since orders can be 'partially filled.'

for execution if the order price crosses best bid or ask. On execution, all counterparties are contacted with Fill messages (and Ammends when applicable).

The exchange also functions as an information repository for the traders. It can respond to the following types of request:

1. **LOB requests**: Publish a version of the LOB which is anonymised. Best Ask and Best Bid are also given.
2. **Tape requests**: Publish the history of new orders, cancels, fills and trade ammendments up to a user defined duration.
3. **Personalised position**: The positions (levels and ordering) of a trader within the current LOB.

## 1.5 Traders

The role of the basic trader object is to receive information and submit orders to the exchange. A bookkeeping function exists to keep track of profitability once trades have been executed. Various data structures exist within the trader object to keep track of submitted orders and the current state of the LOB. Where a trader has the capability of holding inventory, this is calculated on a FIFO basis.

The trader object can receive Order 'Confirm', 'Fill' and 'Ammend' messages from the exchange. On receipt of these messages, internal records are updated accordingly. From the Controller it can receive order 'Prompt' messages which invite the trader to submit or refresh orders to the exchange through invocation of the "getOrder" method. Typically, the market variables of a trader are updated at the end of each period on receipt of a "Respond" message sent by the controller.

Alternatively in a setup where there is an underlying fundamental price sequence driving dynamics, the trader can receive information through prompt messages or otherwise.

The trader can send New order and Cancel orders to the exchange.

In practice, specific traders are subclasses of the the default trader object. They are typically differentiated by the mechanism through which they decide on orders to submit to the exchange. Depending on what information they use to reach trading decisions, their internal data maintenance methods will also differ.

In practice, to save on memory and calculation time, the statistics based on order book information which traders maintain are calculated at a class level. This means that only one representative trader of that class need update market statistics each period for all other traders to have the same information. Whilst this decouples the overhead of having more traders of any particular class by eliminating duplication of market statistic calculation, it does come at the expense of restricting decentralisation (Traders of any particular class would have to be housed in the same core). Parallelisation of experiments is also made more slightly more complicated.

### 1.5.1 Trader Zoo - Supply Demand type

These traders are taken from the BSE experiment setup where there is a Supply and Demand object which provides traders with 'client' orders randomly throughout the experiment. These client orders are either buys or sells with a limit price, and the traders are tasked with executing at these limit prices or better. Traders do not hold inventory and execute only in a single direction at any time.

#### GiveAway

When prompted, trader submits order for price equal to limit price of client order.

#### ZIC

Trader submits orders for prices randomly selected between client order limit price and a some percentage of the limit price. This type of zero intelligence trader appeared in Gode and Sundar 1993.

#### Shaver

Trader improves best bid (ask) by a price increment if their client limit order price is higher (lower) than best bid (ask).

#### ZIP

First appearing in Cliff 1997,

### 1.5.2 Trader Types - Price Series type

In this experiment setup, traders are given noisy signals about an underlying price sequence by the Controller. Traders are free to take inventory within bounds and can submit both bids and asks simultaneously. Typically they are given a randomly generated preference over their inventory holdings which affects their price submissions.

#### WW ZIP

The trader receives a noisy signal from the controller when prompted to submit orders. With this, prior beliefs about the fundamental price and knowledge of the fundamental price process and its preference function over inventory, the trader submits bid and ask orders for unit orders. This trader is taken directly from [Wang and Wellman, 2017].

#### WW Heuristic Belief (HBL)

In addition to the information that the WW ZIP trader receives, the HBL trader also estimates the probability of execution as a function of order price. This is maintained through a separate memory object, where memory and rejection period are parameters which dictate how much history is used to calculate the probability, and how old orders have to be to be considered rejected. This is taken from [Wang and Wellman, 2017].

### 1.5.3 Noise Trader

The Noise trader receives no information about the fundamental price but instead calculates the average price of order submissions over some period of time determined by a memory coefficient. Cancelled orders are adjusted for.

If there was a bid improvement last (best bid increased), the trader will further improve the best bid. Otherwise the trader submits an order based on the average price that they calculated.

### 1.5.4 Cont Imbalance

The Cont Imbalance trader calculates the order book imbalance as defined in [Cont et al., 2013] calculated over a trader specific memory. If greater than zero it will place an order equal to best bid plus the imbalance multiplied by a constant (a randomly chosen parameter between 0 and 1 when the trader is created). The price is further modified by the trader's inventory preference as before.

## 1.6 Supply and Demand

In some ways this is the hardest of the main four elements to grasp, but all multi agent market simulations do have some kind of mechanism which drives the moves of the LOB. Sometimes this has a strong economic interpretation (demand and supply curves) but more often than not, there is just an underlying 'fundamental' price sequence which we assume implicitly reflects supply and demand conditions.

We will describe the Supply and Demand mechanisms that BUCLSE has been tested on.

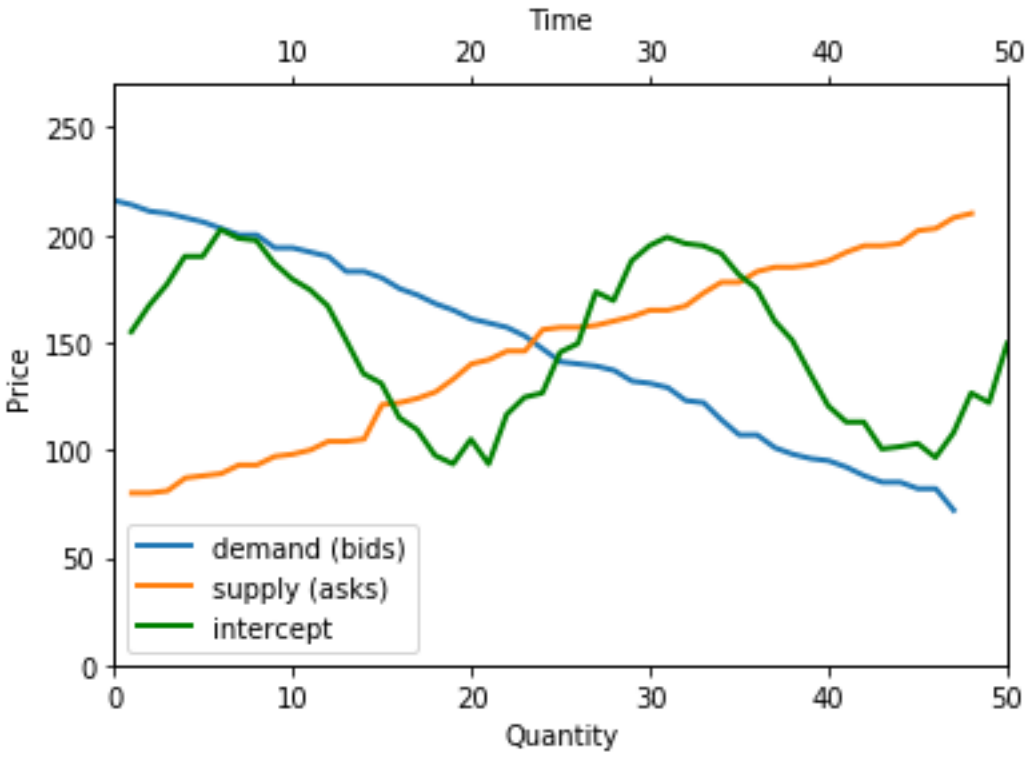


Figure 1.2: Equilibrium price defined over time, as defined by the intersection of demand and supply curves at any moment in time. In this figure, the demand and supply curves are taken as of t=50, hence their intersection is in line with green curve at t=50.

### 1.6.1 Supply Demand setup 1: BSE

In the BSE setup, there is a list of buy and sell 'client' orders defined over time which can be in turn used to define supply and demand curves. Order prices are distributed over some user defined, time varying range. From this order list a curve can be calculated by plotting cumulative quantity of orders below any price for demand and cumulative over any price for supply.In other words, the curves define the quantity theoretically in the market at any period of time that is willing to sell above a certain price or buy below a certain price. Demand declines as quantity increases whilst supply increases as price increases. Classic microeconomics tells us that equilibrium is found where the two lines meet; where demand equals supply. Typically the demand and supply curves are linear at any moment in time, but they can be translated according to a user defined function, thereby giving a shifting equilibrium over time, see figure 1.2.

The BSE setup gives us freedom to define how the curves are formed, how they change over time

and at what rate the orders are distributed to traders. One experimental improvement from the original BSE formulation is that BUCLSE pre-specifies the orders and destination before the experiment begins (originally this is done on demand). We are therefore able to rerun the same experiment (albeit with potentially different stochastic behaviour from the traders).

### 1.6.2 Supply Demand setup 2: Wang Wellman

This formulation specifies a fundamental price sequence from which traders receive noisy signals. The price process is a mean reverting random walk and noise is Gaussian. As before, a sequence is set which determines when traders receive information and are subsequently prompted for orders. At most one trader is prompted for an order per period. The price, noise and trader prompt sequences are set before the experiment begins, allowing the repeat of any experiment.

## 1.7 Market Session Objects

The final element organises the experiment - it defines traders within the market, and the supply and demand related schedules and records any information where appropriate. One of the principle methods of control is through the timer object - the Market Session is able to control the passage of time. Experiments are structured as a set sequence of events that happen within a period. Periods are repeated by incrementing the timer object until the timer has reached its limit.

During a period, a number of events occur:

It is likely that this object will change the most as different experiments are attempted. The modular nature of the

### 1.7.1 Market session setup 1: BSE

The sequence for this type of experiment is as follows:

Customer Orders and trader submission sequence is predefined.

1. New customer orders are dispatched to traders
2. A trader is picked to submit an order to the exchange
3. Any resulting trades are processed
4. Traders update their records according to changes in the LOB.

### 1.7.2 Market session setup 2: Wang Wellman

The sequence for this type of experiment is as follows:

Fundamental price sequence, noisy signals thereof and trader submission sequence is predefined.

1. A trader is picked to submit an order to the exchange. It is given a signal about the value of the fundamental with which to base its order submission. It can submit both bids and asks concurrently as long as it is below its inventory limits.
2. Trader submits order directly to exchage and any resulting trades are processed.
3. Traders update their records according to changes on the LOB.

## 1.8 Reinforcement Learning adaptation

BUCLSE can be easily adapted for the purposes of reinforcement learning with the addition of a RL Trader object and an RL Environment object which sits above the experiment controller.

The RL trader inherits much of the machinery developed for other traders (principally order submission and bookkeeping). It has an interface to submit any type of order to exchange.

The RL Environment object is a subclass of the environment class found in OpenAI Gym. This was chosen because of its simplicity and familiarity to anyone who has undertaken RL research. It also means standardised RL toolkits can be brought to BUCLSE at low cost.

The bulk of the effort in adapting the platform for RL is spent formatting state space information (and to a lesser extent action space and writing reward functions). Since this is experiment specific, the RL Environment object will always be quite variable.

Custom Step, Render and Reset methods.

## 1.9 Future development

# Bibliography

[Cliff, 2018] Cliff, D. (2018). BSE: A Minimal Simulation of a Limit-Order-Book Stock Exchange. *European Modelling and Simulation Symposium (EMSS-2018)*.

[Cont et al., 2013] Cont, R., Kukanov, A., and Stoikov, S. (2013). The price impact of order book events. *Journal of Financial Econometrics*, 12(1):47–88.

[Wang and Wellman, 2017] Wang, X. and Wellman, M. P. (2017). Spoofing the Limit Order Book: An Agent-Based Model. *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems*, pages 651–659.