ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Σχολή Τεχνολογιών Πληροφορικής και Επικοινωνιών
Τμήμα Ψηφιακών Συστημάτων

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
Πληροφοριακά Συστήματα & Υπηρεσίες
Κατεύθυνση: Μεγάλα Δεδομένα και Αναλυτική

Experimental Study of Online Learning Algorithms for Market Making

Διπλωματική Εργασία

Κωνσταντίνος Γεράσιμος Λασκαράτος
Α.Μ. ΜΕ1814

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ.Ορέστη Τελέλη για την καθοδήγηση και τις συμβουλές του κατά τη διάρκεια της συγγραφής της εργασίας μου.

Περίληψη

Στο χρηματιστήριο, κεντρικό ρόλο έχει ο Market Maker ο οποίος παρέχει ρευστότητα στην αγορά αγοράζοντας και πουλώντας μετοχές. Ένας απο τους βασικούς σκοπούς του είναι να επωφεληθεί από τις συνεχόμενες αλλαγές της τιμής της μετοχής. Στις μέρες μας, το μεγαλύτερο μέρος της αγοραπωλησίας των μετοχών γίνεται με την βοήθεια των υπολογιστών, με αποτέλεσμα οι ταχύτητες και η πολυποκότητα των αγοραπωλησιών να εκτοξευθεί. Έτσι ο Market Maker θα πρέπει να ανταποκριθεί στην αύξηση της ταχύτητας και της πολυπλοκότητας με την εφαρμογή ευφυών αλγορίθμων και σύγχρονων τεχνολογιών. Στην παρούσα εργασία υλοποιήσαμε σε Python μια αναλυτική προσομοίωση ενος limit order book με τις βασικές λειτουργίες του. Επίσης υλοποιήσαμε μια γεννήτρια τυχαίων αγοραπωλησιών που τροφοδοτεί το limit order book με τυχαίες εντολές αγορών και πωλήσεων. Επιπλέον ο σχεδιασμός της γεννήτριας βασίστηκε σε πραγματικά δεδομένα του Kaggle, και πιο συγκεκριμένα στο Algorithmic Trading Challenge dataset. Τέλος υλοποιήσαμε εναν Agent ο οποίος χρησιμοποιεί Online Αλγορίθμους, τον ε_n-Greedy,τον UCB και τον EXP3, οι οποίοι χρησιμοποιούν στρατηγικές παραθύρου και τις οποίες εμπνευστήκαμε απο την βιβλιογραφία. Πειραματικά, συγκρίναμε την απόδοση των Αλγορίθμων όσον αφορά το κέρδος του Market Maker, αλλά και όσον αφορά το Regret σε σχέση με την καλύτερη στρατηγική.

Abstract

The key role of a Market Maker in stock exchange is provision of liquidity via issuing of buy and sell orders for a security; at the same time he is willing to profit from the changes of the stock price. Nowadays, in modern stock exchanges, most of the trading occurs in computers and thus the environment has become more demanding in terms of the speed and complexity of transactions. For that reason, the Market Maker must use high tech software and efficient Algorithms to adapt to these changes. In this thesis, we developed a limit order book simulation with its basic functions using Python. We further developed a "random order generator" which feeds the Limit Order Book with pseudorandom buy and sell orders. In order to create the generator, we analysed the Algorithmic Trading Challenge dataset from Kaggle which consists of real stock market data. Finally, we designed an Agent that uses a class of Online learning algorithms, the ε_nGreedy, the UCB1 and the EXP3. These Algorithms are using a class of window based strategies, inspired from the literature. In our experiments we compared these algorithms with each other, focusing on the Agent's profit and their regret in terms of the overall best window strategy.

Contents

Chapter 1

Introduction

In this thesis, we consider the problem of designing an automated Agent for stock trading exchanges. An automated Agent is a bot (automated program) which observes a certain environment, takes actions on that environment and learns how that environment operates so as to take better actions. Our Agent's environment is the Limit Order Book where the Agent constantly decides whether to sell or to buy a stock.

Limit Order Book is a continuous record which holds all the incoming buy and sell orders from the people who are willing to trade, for a certain stock. It also matches the buy orders with the sell orders in the appropriate and legal order. The Agent observes some of Limit Order Book's variables, such as the price of the stock, in order to learn how it operates at a certain period so as to make better decisions on buying or on selling.

The use of automated Agents that continuously sell and buy stocks in a duration of milliseconds is called High Frequency Trading (HFT). HFT is a type of trading where the trader sells or buys stocks in fraction of seconds, with the help of computer system which can provide large computational power. HFT also uses Algorithms from the field of Machine Learning and AI. A common class of Algorithms used in HFT are Online Learning Algorithms.

An Online Algorithm is one that receives input sequentially in small well defined chunks. Upon reception of every chunk of input data, the Algorithm must process it and make an irrevocable decision. The whole sequence of decisions makes up a solution for the whole set of data. In constrast, an offline Algorithm receives the whole input upfront and outputs a global decision [12, 5].

In Machine Learning, an Online Algorithm at timestep t, receives input x_t, processes x_t and makes a prediction $\hat{y}_t$. In the offline setting of Machine Learning, the Algorithm processes the whole dataset at once and then it can make predictions.

We apply a class of online algorithms for High Frequency Automated trading. We also create a custom trading environment where we conducted our experinments.

High Frequency trading (HFT), from its appearance, has made some fundamental changes, in the way that people are trading stocks (and other goods) in modern financial markets. Nowadays, most of the trading occurs in computer systems, whose computing power outperform compared to humans. That is because computers can react faster and more frequently to changes that occur in stock markets [16].

For that reason, a large field has been developing in the past 20 years, that combines computer science with financial science. More specificaly for HFT, scientists have developed (and they are still developing) algorithms which are applicable to stock market. Aldridge metnions in [2] that the profitability of high-frequency enterprises is further reinforced by the exponential growth of the trading industry and that HFT now accounts for over 60 percent of trading volume coming through in stock exchanges.

We study a class of Online Learning Algorithms and we measure their performance in a HFT trading setting. We also have created from scratch a Limit Order Book simulation and its basic functionality based on the rules of the largest stock exchanges. We have also analyse the data of the dataset "Algorithmic Trading Challenge" in order to create a generator which simulates the transaction flow that happens in a stock market.

After we have set up our custom stock market environment, we create an Agent that trades in that environment. The Agent also trades based on these Online Learning Algorithms that instruct him when to buy and sell a stock.

Finally we evaluate and we compare these Online learning Algorithms based on the profit of the Agent, the amount of stocks the Agent holds overtime and with the notion of Regret, which is how good the Agent's is action from the optimal action that Agent could have taken.

In more detail in these thesis:

- In chapter one, we give a brief presentation of every key component of our study. We present some basic terms about stock market. We also provide the basic theory of Online Learning and we present the fundamental online algorithms that we will use in our setting.
- In chapter two, we put existing work which we used to build these study. These chapter includes prior work about Online Learning, High Frequency trading, Limit order books and Market Making. We also present some basic features of the LOB.
- In chapter three, we present our developments which we created in order to conduct

our experiments.

- In chapter four we present the results of our experiments and we analyze the performance of our HFT Automated Agend that use three Online Learning Algorithms.

1.1 Stock market and exchanges

The stock market is an exchange market where the companies' shares are bought and sold. Some of the largest stock exchanges are the New York Stock Exchange, Nasdaq, London Stock Exchange, Japan Exchange Group where thousands of trades occur every hour.

All exchanges and markets provide a secure and regulated environment where all participants can trade stocks and other financial goods. For convenience, in this work, we discuss only stock trading. A stock is a security that represents the ownership of a fraction of a corporation.

The basic function of a stock exchange is as follows:

A number of participants who want to buy or sell stocks enter the exchange. A regulator determines the correct market price p_t of a particular stock based on supply and demand. The participants start placing their orders. In general, there are two types of orders: market orders M_t and limit orders L_t.

- M_t **Market orders** are the most common orders in an exchange and the most direct ones. When a participant wants to buy or sell a certain amount of stocks, he places a buy market order M_{B_t} (or respectively a sell market order M_{S_t}) for $\mathcal{X}$, and the transaction is executed at the current market price.
- L_t **Limit orders** are slightly more complicated than market orders. A participant places a buy L_{B_t} or a sell L_{S_t} for a certain amount of $\mathcal{X}$ shares at a specific price p. Then he is waiting until there is a counterparty that agrees to transact at that p. This means there is a possibility that the order will never be executed. A participant can also cancel his order before it is executed.

The execution of an order happens after two participants (a seller and a buyer) agree on a price p and on a quantity of stocks $\mathcal{X}$. When the transaction is executed we call it a match, because a sell order matches with a buy order. Since there are thousands of participants which are continuously placing orders, the exchange has a priority on how the orders are executed and in which order. Firstly the orders that are closer to market price p_t are executed, i.e. the buy orders with highest price and the sell orders with the lowest prices. Time priority is also observed.

A stock exchange operates like every other market with its own rules and regulations. The exhange ensures fair transactions, efficient price discovery and maintenance of liquidity.

The participants in stock exchange can have a variety of roles and functions. An investor, for example, buys stocks and holds them for a long time (months or years). A trader buys and sells stocks continuously in seconds or minutes seeking a profit from small price changes. A market maker's role is provision of liquidity via issuing of buy and sell orders for a security; at the same time he is willing to profit from the changes of the stock price. In our work we view the stock exchange from the perspective of a market maker.

A Market Maker (MM) is a firm or individual who actively quotes both a buy or a sell price in a stock exchange, hoping to make profit by exploiting the difference between bid price b_t and the ask price a_t, known as the spread s_t. MMs serve an important role in a stock exchange, as they help to reduce liquidity risk and generally they provide a higher level of service compared to electronic trading. They also avoid the risk of holding a large number of stocks because they may see a decline in the price of a security after it has been purchased from a seller and before it is sold to a buyer.

Bid price b_t: is the highest price that a buyer is willing to pay

Ask price a_t: is the lowest price that a seller is willing to accept for selling a security

The MM uses a Limit Order Book (LOB). The LOB rocords limit orders for a specific stock. Each limit order entering the market, is recorded by the LOB. The LOB executes buy orders at a price at most equal to the specified price, and sell orders at a price at least equal to the specified price.

High frequency trading with online learning

In the recent years, trading occurs mostly in electronic servers in data centers [14], where the trade orders are transferred via the internet. Most of the trading is not performed by humans but by computers. That is because nowadays it is more efficient for a human to create a trading algorithm and let the computer do all the "dirty" job than to trade oneself alone. As commented in [14] a computer reacts faster than a human (operating at the submillisecond time scale), more frequently, and can expose itself significantly less in the market.

An efficient HFT algorithm receives a large amount of data every microsecond and it must be able to process the received data very quickly and make the right decisions. For example, if it detects a change on the price for a stock, it should react by placing a buy or a sell order.

Online learning algorithms are suitable for high frequency trading. At an online setting new input is revealed sequentially. The algorithm should react to that input by making a decision, e.g whether or not to submit a trade. An HFT trading algorithm should react as fast as possible to the incoming inputs. The larger the amount of data the algorithm has to process, the slower it will be in decision making. For that reason a trading algorithm should only keep a minimal amount of data and parameters which should reflect the current state of the market and must be updated in real time when new changes are observed [14].

Market Making with HFT

HFT Algorithms are commonly used for Market Making. As mentioned above, Online learning is very applicable to trading algorithms. Another class of algorithms that can be very useful in HFT and especially in market making are reinforcement learning algorithms. In Reinforcement learning (RL), an Agent takes actions in an environment, in order to maximize its cumulative reward. In our case the Agent is the MM, the environment is the limit order book, the actions are the trade orders and the cumulative reward can be MM's profit.

1.2 Online Learning

Online Learning is a subcategory of Machine Learning where data appear in a sequential order at each step and a prediction is made based on those data. As a result, in Online learning the training and the prediction occurs simultaneously.

At each time step t, an Online Algorithm receives the data $x_t \in \mathcal{X}$ and makes prediction $\hat{y_t} \in \mathcal{Y}$. After that it receives the true value of y_t and calculates how far the prediction $\hat{y_t}$ was from y_t. The latter is calculated by a Loss Function $\mathcal{L}(y_t, \hat{y_t})$ and the main goal of the Online Algorithm is to reduce the cummulative loss $\sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y_t})$ over horizon T [15].

$$\text{Loss} : \mathcal{L}(y_t, \hat{y_t}) = |\hat{y_t} - y_t)| \tag{1.1}$$

$$\text{Regret} : \mathcal{R}_{\mathcal{T}} = \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y_t}) - \min \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y_t}) \tag{1.2}$$

1.2.1 Expert advice algorithms

A basic online setting is the Expert Advice [7],[15]. In this setting there are N experts. The Online Algorithm now after it receives $x_t \in \mathcal{X}$, it then receives a prediction from every expert $\widehat{y_{t,i}} \in \mathcal{Y}$ where $i = 1, .., N$. Every $\widehat{y_{t,i}}$ is compared with the true y_t and it calculates cumulative $\mathcal{L}$. The objective is to minimize the Regret $\mathcal{R}$. Regret defines the difference between Algorithms cumulative $\mathcal{L}$ and the cumulative Loss of the best expert.

A basic Online Algorithm on the Expert setting is the Weighted Majority Algorithm (WM) [13]. WM assigns a weight to an expert. All Experts start with the same weight. At every timestep WM generates predictions using a weighted majority vote. When the true output is revealed, the weights of the Experts that predicted wrong are reduced. Thus, every time an expert gives a wrong advice we reduce our trust in him so we have to reduce his vote on which action we should take.

Algorithm 1 Weighted Majority [15]

```
Ensure: N experts: $i = 1, 2, ..., N$ and K actions $a_j(t)$ $j = 1, .., K$
Ensure: Initialize: $w_i(1) = 1$ for $i = 1, 2, ..., N$ and $\beta \in (0, 1]$
  for t = 1,2,... do
    Play $a_j$ with $max \sum_{i:a_i(t)} w_{t,i}$
    Action $\hat{a}_j$ with highest $r_i$ revieled
    for i = 1,...,N do
      if $a_i! = \hat{a}_j$ then
        $w_i = w_i \beta$
      end if
    end for
  end for
```

An evolution of WM Algorithm is a randomized version of it, the Randomized Weighted Majority RWM. That Algorithm uses a probability distribution to select an expert rather than a weighted majority vote. In the beginning, a weight $w_i(1) = 1$ and a probability $p_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$ are assigned to every expert. At every timestep we select a random expert based on p_i and we receive a reward. If the a_i is not the $\hat{a}_j$ then w_i is reduced and then the probabilities p_i are recalculated.

Algorithm 2 Randomized Weighted Majority [15]

```
Ensure: N experts: $i = 1, 2, ..., N$ and K actions $a_j(t)$ $j = 1, .., K$
Ensure: Initialize: $w_i(1) = 1$ for $i = 1, 2, ..., N$, $\beta \in (0, 1]$ and $p_i = \frac{1}{N}$
  for t = 1,2,... do
    Every expert advices for an action $a_i$
    Select a random expert based on $p_i$
    Action $\hat{a}_j$ with highest $r_i$ revieled
    for i = 1,...,N do
      if $a_i! = \hat{a}_j$ then
        $w_{t+1,i} = w_{t,i}\beta$
      end if
      $W_{t+1} = \sum_{i=1}^{N} w_{t+1,i}$
    end for
    for i = 1,...,N do
      $p_i = \frac{w_{t+1,i}}{\sum_{i=1}^{N} W_{t+1}}$
    end for
  end for
```

Another extension of WM algorithm is the exponential Weighted Average Algorithm. That algorithm selects the action that maximizes $\frac{\sum_{i=1}^{N} w_t a_i}{\sum_{i=1}^{N} w_t}$. For the experts that did not select the best action, their w_i is reduced according the following rule: $w_{t+1,i} = w_{t,i}e^{-\eta L(\hat{a}_t, a_t)}$.

Algorithm 3 Exponential Weighted Majority [15]

```
Ensure: N experts: $i = 1, 2, ..., N$ and K actions $a_j(t)$ $j = 1, .., K$
Ensure: Initialize: $w_i(1) = 1$ for $i = 1, 2, ..., N$ and $\beta \in (0, 1]$
  for t = 1,2,... do
    Play $a_j$ with $\frac{\sum_{i=1}^{N} w_t a_i}{\sum_{i=1}^{N} w_t}$
    Action $\hat{a}_j$ with highest $r_i$ revieled
    for i = 1,...,N do
      if $a_i! = \hat{a}_j$ then
        $w_{t+1,i} = w_{t,i}e^{-\eta L(\hat{a}_t, a_t)}$
      end if
    end for
  end for
```

1.2.2 Multi-armed Bandit Algorithms

Another class of Algorithms which can do Online Learning are Multi-armed bandit algorithms. Those Algorithms are for solving the exploration vs exploitation dilemma, i.e. the tradeoff between collecting information from a set of actions, in relation to their outcome, in order to learn which one is the best, and to use that information to avoid choosing the actions with the worst outcomes.

Multi-armed Bandit problem and exploration vs exploitation dilemma

Consider we have a slot machine with N arms. Every arm gives a reward $r_i \in (0, 1)$ where $i = 1, .., N$, which is generated from a probability distribution with unknown expected reward $\mathbb{E}[r_i]$. The goal is to select the arm, always yielding the highest reward in order to maximize our total profit.

Since the expected rewards $\mathbb{E}[r_i]$ of the arms are unknown, we have to constantly evaluate every arm we choose in order to know how good the arm is in the long term. For example, keeping track of the average reward $\overline{r_i}$ which every arm produces will help us determine which is the best arm until then.

$$\text{Average reward } \overline{r_i} = \frac{\sum_{j=1}^{n_i} r_i}{n_i}, \text{where } n_i \text{ is the number of times arm i has been played}$$

It is a fact that, at any timestep there is at least one arm which has the highest $\overline{r_i}$. In that case,we can use $\overline{r_i}$ as an estimate that will instruct us which arm to select next. A method that selects the best arm based on $\overline{r_i}$ is the Greedy Algorithm. The Greedy Algorithm forces us to always select the arm with the best $\overline{r_i}$. More specifically, the Greedy method 1) finds the arm with the highest $\overline{r_i}$, 2) plays that arm and then 3) receives the reward and 4) updates the $\overline{r_i}$ of that arm.

Algorithm 4 Greedy Algorithm [18]

```
Ensure: N arms: $arm_i$ with $i = 1, 2, ..., N$
  for t = 1,2,... do
    $j = argmax(\overline{r_i}[arm_i])$
    play arm j and get reward
    update $\overline{r_i}_{arm_j}$
  end for
```

By always playing the arm with the highest $\overline{r_i}$, we might miss a better reward by an arm that did not have the highest $\overline{r_i}$. For this reason sometimes it might be better to select another arm in case we discover a better reward. In other words, we have to explore to find a better arm. That can be achieved with ε-Greedy Algorithm. The difference between ε-Greedy Algorithm and Greedy, is that ε-Greedy not only selects the arm with the best $\overline{r_i}$, but it also explores a random arm once in a while with a probability ε, in case it receives a better reward. More specifically, ε-Greedy Algorithm, with probability ε plays the arm with the best $\overline{r_i}$ while with probability 1-ε it plays a random arm. Then it receives the reward of that arm and updates the $\overline{r_i}$.

When probability $\varepsilon = 1$ ε-Greedy Algorithm selects only the arm with the best SR, like Greedy algorithm, it always exploits the arm with the highest $\overline{r_i}$. The smaller the ε, the more ε-Greedy explores other arms in case it finds an arm with higher reward at that time.

The above setting illustrates the exploration vs exploitation dilemma. Depending on the expected reward of the arms, we have to balance exploration and exploitation by choosing the right ε. If we are sure that an arm is far better than other arms, it will be better to exploit, but if the expected rewards of the arms are close to each other it will be better to explore more.

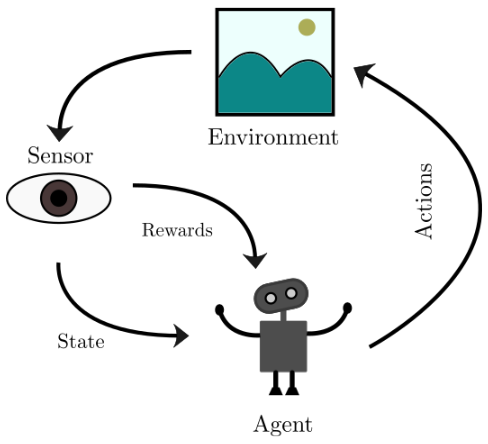
Reinforcement Learning

The multiarmed bandits problem, the expoitations vs exlporation dilemma and the algorithms that we discuss above are some basic fundamentals of Reinforcement Learning.

Reinforcement learning(RL) is the area of Machine Learning where an Agent learns by interacting with an environment in order to receive a reward. More specifically, the Agent takes some actions $a(t)$ inside that environment and depending on that action it receives a reward. The main goal of Agent is to always take the actions which yield the highest reward in order to maximize our total profit. For example, at the multiarmed bandit setting we discussed above,the Agent interacts with the environment, the environment is the slot machine, and Agent's goal is to find the arm that produces the best outcome over time. As the Agent interacts with the slot machine, he tries to discover which is the arm that will produce the highest reward at every time step.

$$Reward\ r(a_i) = \{\text{ outcome from action } a_i\}$$
$$a_t \text{ defines the action taken at time step t}$$

As we have previously mentioned in Online learning we make a prediction y_t at timestep t, after x_t is received. In this thesis we use Reinforcement Learning along with Online learning for the training process. For example the Agent receives at every t the state of the environment (x_t) i.e. how the actions have performed so far, and then it takes $a(t)$ (y_t) and receives the reward ($\hat{y}_t$).



Source: `http://technopremium.com/`

$$\mathit{Strategy}\ A = \{\ a_t \text{ where t=1,2,.. } \}$$

To solve Reinforcement learning problems we use algorithms which instruct us what $a(t)$ we should take every time. We call those algorithms Strategies A. Every A has a value function $Q_A(a_i)$ for evaluating every action $a(t)$ it takes. For example a common $Q_A(a_i)$ calculates the average of the reward $\overline{r_i}$ of each action taken.

> *Value function* $Q_A(a_i)$ is the value of action a_i under strategy A:
> $Q_A(a_i) = \frac{\sum_{t=1}^{\infty} r(a_i)}{N(a_i)}$ where $N(a_i)$ defines the times a_i has been selected

Averaging Strategies A are appropriate for problems where the rewards do not change. Two fundamental averaging strategies we mentioned before are Greedy and ε-Greedy Algorithm. The Greedy Algorithm exploits the action with the highest $\overline{r_i}$. The ε-Greedy Algorithm exploits also the action with the highest $\overline{r_i}$, but sometimes with some probability ε it explores in case it receives a higher r_i. Sutton and Barto at their book [18] compare these two algorithms and have come to the conlusion after experiments that ε-Greedy Algorithm not only give average overall reward $\overline{r_i}$ higher than Greedy Algorithm, but selects the optimal action more often.

Algorithm 5 E-Greedy [18]

```
Ensure: K actions: $a_i$ with $i = 1, 2, ..., K$
  for n = 1,2,... do
    with propability $\epsilon$:
        select $a_i$ with highest $\overline{r_i}$
    with propability $1 - \epsilon$:
        select random $a_i$
  end for
```

Most of times exploring will be better especially at the begining of the executions when the r_i of all actions are uknown. At further steps, as the probability distribution starts to reveal, we can focus on the best actions. An evolution of ε-Greedy algorithms is presented by Auer et al [4] and it is called ε_n-Greedy. ε_n-Greedy operates like ε-Greedy, except for the fact that probability ε decreaces with a certain rate as long as the uknown $\mathbb{E}[r_i]$ start to converge to $\overline{r_i}$.

Algorithm 6 E_n-Greedy [4]

```
Ensure: K actions: $a_i$ with $i = 1, 2, ..., K$, $c > 0$ and $0 < d < 1$
  for n = 1,2,... do
    $\varepsilon = \min\{1, \frac{cK}{d^2 n}\}$
    with propability $\varepsilon$:
        select $a_i$ with highest $\overline{r_i}$
    with propability $1 - \varepsilon$:
        select random $a_i$
  end for
```

Another way to encourage exploration at the first steps of the execution is by adding optimistic initial values to the $Q_A(a_i)$. For example initialize the $Q_A(a_i) = x$ where $x > 0$. The bigger the x is, the more the a_i will be used. Sutton and Barto [18] show that the Greedy Algorithm with optimistic initial value performs slightly better than ε-Greedy with no initial values.

The Greedy Algorithm always selects the a_i with the highest $Q_A(a_i)$ and the ε-Greedy Algorithm selects once in a while a random action indiscriminately. Another type of algorithms that balance expoitation and exploration in a more efficient way are the Upper-Confidence Bound (UCB) strategies. These strategies choose the next action based not only on $Q_A(a_i)$, but on a UCB as well. The UCB is a quantity that is added to the actual value of the $Q_A(a_i)$, like an optimistic initial value. The only difference is that UCB decreases as the action is selected, so after a number of selections the real value of the $Q_A(a_i)$ remains. In that way, a UCB forces us to explore an a_i that might produce high results, until the real value of $Q_A(a_i)$ is revealed.

Auer et al present in [4] the UCB1 Algorithm that uses UCB. At every step UCB1 selects the a_i with the maximum $(Q_A(a_i) + c\sqrt{\frac{2\ln t}{N(a_i)}})$. Every time a_i is selected, $N(a_i)$ increases while the quantity $c\sqrt{\frac{2\ln t}{N(a_i)}}$ decreases. That means after a number of selections, UCB converges to 0 and UCB1 will select the action based on the true value of $Q_A(a_i)$.

Algorithm 7 UCB1 [4]

```
Ensure: K actions: $a_i$ with $i = 1, 2, ..., K$
  for i = 1,2,... do
    take the action $a_i(t) = \max_{i=1,..,K}(Q_A(a_i) + c\sqrt{\frac{2\ln t}{N(a_i)}})$
    update $N(a_i) = N(a_i) + 1$
  end for
```

All the above algorithms: Greedy, E-Greedy, E_n-Greedy and UCB1 are the best to use when the probability distribution of the r_i does not change over time. When the $\mathbb{E}[r_i]$ changes from time to time, we can not make any statistical assumptions about the outcomes of a. In that case, these algorithms will underperform.

Auer et al [3] presented an Algorithm, called EXP3. EXP3 in a non stationary setting, where $\mathbb{E}[r_i]$ is not stable and selects the best arm at the rate $O(T^{-\frac{1}{2}})$. EXP3 assigns weights to each a_i and in the beginning it initializes them as w_i=1. At each step it uses the weights to create a uniform distribution from which it will pick an a_t randomly. It also uses a factor $\gamma \in (0, 1]$ to control exploration. The closest γ is to 1, the less EXP3 takes into account the w_i and the chances of selecting an action a_i become equal. After the selection of an a_i, EXP3 updates its w_i based on the r_i of the chosen a_i and then it normalizes the list of all w to be added to 1.

Algorithm 8 EXP3 [3]

```
Ensure: K actions: $a_i$ with $i = 1, 2, ..., K$
Ensure: Initialize: $w_i(1) = 1$ for $i = 1, 2, ..., K$ and $\gamma \in (0, 1]$
  for t = 1,2,... do
    set Probability distribution P [$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}$], i=1,...,K
    draw $a_i$ randomly from P and receive reward $r_i(t)$
    set $w_i(t+1) = w_i(t) exp(\frac{\gamma \overline{r_i}(t)}{K})$ where $\overline{r_i}(t) = \frac{r_i(t)}{p_i(t)}$
    for j = 1,...,K do
      $w_j(t+1) = \frac{w_j(t+1)}{\sum_{j=1}^{K} w_j(t+1)}$
    end for
  end for
```

1.2.3 Online Reinforcement Learning with Market Making

Online Reinforcement Learning and specificaly the Algorithms we have discussed so far, can be used for Market Making in HFT. There is a little work on the subject. For example Abernethy and Kale do market making using spread strategies along with Online Algorithms [1] and Spooner et al. created a custom reward function and designed a market maker Agent using reinforcement learning [17].

Some of the basic issues online Reinforcement Learning can solve on market making:

- Minimizing MMs inventory risk. That is to not hold a large amount of stocks for a large period of time because it might not be able to liquify them or it might has to sell them cheaper that it bought them.
- The MM can provide efficiently liquidity by transacting when other traders do not, at situations like when spread is large.
- The MM can earn profit by exploiting price movements earlier that other traders.

Chapter 2

Market making with Limit Order Books

In this chapter, we discuss previous work related to the subject of this thesis. That work helped us not only understand the theoritical consepts of the field we study but also helped us in our developments. First we discuss features of the Limit Order Book, as presented in the work about Limit Order Books [9, 11, 10]. Most of the concepts we learned there, were used in manufacturing a custom LOB with its basic functionalities. Then we discuss previous work related to Market Making.

2.1 Limit Order Book work

A very detailed explanation about what is the LOB and how it works is written by Gould et al. [9]. They analyse all the basic components of LOB ,like spread,mid price, ask price, bid price etc, from theoritical and mathematical perspective. They examine how each component behaves inside the environment of the LOB, and how that behavour affects the other components. For some components like price and volatility they investigate in greater detail the impact they have upon the function of the LOB. They also explain the impact these components have on people's behavour. For example how a change on the price will affect a buyer's or a seller's decision on placing an order.

A simulation of an order book is presented in [11] by Kane et al. They created a development package which can simulate the function of the LOB, given order data. They analyse the functionality of their simulation and how someone can use these methods to reproduce their own LOB.

Below we describe some basic features of the Limit Order Book (LOB) and introduce associated terminology based on the above related works.

2.1.1 LOB Basics

LOB is a data structure that keeps track of all orders, buy or sell, for a specific security. It also matches buy orders with the corresponding sell orders (and the opposite). An order that is submitted at some time t $\mathcal{Z}_t = \{type, \mathcal{W}_t, p_t\}$, consists of three components:

- **Order type** : Order type can be either Market Orders M_t or Limit Orders L_t. The initiator of the order also has to specify either if he wants to sell or buy stocks. We explain M_t and L_t bellow.
- **Order size** $\mathcal{W}_t$: Order size is the amount of stocks, the initiator of the order desires to buy or sell. The $\mathcal{W}_t$ must be a multiple of the lot size σ, which is the smallest amount of a stock (asset) that can be bought or sold in a market.

  Order Size: $\mathcal{W}_t = k \cdot \sigma$ where k=1,2,...,N $N \in \mathbb{N}$

- **Order price** p_t: Order price is the amount of money the initiator is willing to pay or take in order to buy or sell. p_t must be a multiple of the tick size π. Tick size π which is the smallest interval that two consecutive prices can have.

  Order Price: $p_z = k \cdot \pi$ where k=1,2,...,N $N \in \mathbb{N}$

For better visualization, the LOB order list can be divided in two queues:

- The **ask queue** $\mathcal{A}_q$ which is the queue of the current sell orders which are in ascending order depending on the price.
- The **ask queue** $\mathcal{B}_q$ which is the queue of the current buy orders which are in descending order depending on the price.

The depth $\mathcal{D}$ of the LOB is defined as the sum of the quantities of all orders of the book. Also the depth of $\mathcal{A}_q$, $\mathcal{D}_{ask}$ is the sum of all quantities of all ask orders and the depth of $\mathcal{B}_q$, $\mathcal{D}_{bid}$ is the sum of all quantities of all bid orders, thus $\mathcal{D}$=$\mathcal{D}_{ask}$+$\mathcal{D}_{bid}$. As we see in figure 2.1 $\mathcal{D}_{ask}$=600 and $\mathcal{D}_{bid}$=1130.

As we have already mentioned there are two types of orders, limit orders L_t and market orders M_t:

- A **Limit order** L_t is a type of order where you specify the quantity $\mathcal{W}$ of the stocks and at what p you are willing to buy or sell. For example if someone wants to sell 10 stocks for 49.6$, he places $z_t = (L_{sell}, 10, 49.6\$)$. When the order is placed the depth of the ask price in figure 2.1 will be updated to 110. After someone places a L_t, then he waits for the order to be executed. The order is executed when someone wants to buy these 10 stocks for 49.6$. More specifically, this will happen when someone places a buy order (L_t or M_t) that will be matched with z_t. We explain about matching below.
- A **Market order** M_t is a type of order where you specify only the quantity $\mathcal{W}$ of the stocks you are willing to buy or sell. If someone wants to buy 50 stocks immediately, he places $z_t = (M_{buy}, 50)$, and the order will be executed at the lowest available sell price in the LOB. In the figure 2.1 that order will be executed at $p = 49.5\$$ and the depth at that price will be reduced to 300.

	Price	*Ask queue*
	49.7	150
	49.6	100
	49.5	350
700	49.1	
330	49.0	
100	48.9	
Bid queue		

Figure 2.1: Limit order book representation

Each queue, $\mathcal{A}_q$ and $\mathcal{B}_q$, has price levels. Each price level has a depth, e.g. the depth at $p = 48.9\$$ is $n_{48.9} = 100$ $\mathcal{A}_q$□ and also has a list with all active orders which are placed at that price (Figure 2.2).

Time	Initiator	Quantity
02:24:13	user13	20
02:30:14	user4	45
02:31:00	user50	25
04:15:34	user1	10

Figure 2.2: Active orders at price level $p = 48.9\$$

Order matching The LOB must have a matching algorithm that arranges in which order the active orders will get executed when an opposite order arrives, e.g. which buy order with price p$ will get executed after the arrival of a sell order at that price. The most common matching algorithm is to execute first the orders that have been placed earlier. For example, based on figures 2.1 and 2.2 let us consider that someone places a sell limit order at price $p = 48.9\$$ with $\mathcal{W} = 50$. As we can see in figure 2.2 there are 4 active orders at that price level. The earlier order is the one placed by user13, so it is the first to be executed. After the first matching, there is a remaining quantity of $\mathcal{W} = 30$. That will be matched with the second earlier order that was placed by user4. The new depth of the price level $p = 48.9\$$ appears in figure 2.3.

Time	Initiator	Quantity
02:30:14	user4	15
02:31:00	user50	25
04:15:34	user1	10

Figure 2.3: Active orders at price level $p = 48.9\$$ after L_{sell} with $\mathcal{W} = 50$

Three basic price components which define the state of the LOB at some time t are:

- **Ask price** a_t is the lowest price level at which an active sell order is placed. Considering the figure 1.1 $a_t = 49.5\$$.
- **Bid Price** b_t is the highest price level at which an active buy order is placed. Considering the figure 1.1 $b_t = 49.1\$$.
- **Mid Price** m_t is the middle price between a_t and b_t. Considering the figure 1.1 $m_t = 49.3\$$.
- **Spread** S_t is the difference between a_t and b_t. Considering the figure 1.1 $S_t = 0.4\$$

Price changes When orders enter the LOB, the above components constantly change, and so the state of the LOB changes. Let's take for example figure 1.1. At some time t someone places a $\mathcal{M}_{sell}(w = 1050)$ or a $\mathcal{L}_{sell}(p = 49.1\$, w = 1050)$. Then the depth of price levels 49.1\$, 49.0\$ will become 0 and the depth of the price level 48.9\$ will be reduced by 20 $(1050 - (700 + 330))$. The outcome of this execution is presented in figure 1.4. After the execution, b_t will change to 48.9\$, m_t to 49.2\$ and S_t to 0.6\$

	Price	Ask queue
	49.7	150
	49.6	100
	49.5	350
80	48.9	
Bid queue		

Figure 2.4: Limit order book representation

2.2 Market Making work

Abernethy and Kale [1] designed an online low-regret algorithm that uses spread based strategies to do market making, that achieves almost as much payoff as that of the best strategy.

The Spread based strategies that are used are parameterized by a window size $b \in \{\delta, 2\delta, ..., B\}$ where B is multiple of δ. At timestap t, when the market price is p_t, stategy $S(b)$ selects a size b, $[p_t, p_t + b]$. When p_t drops to $p_{t+1} = p_t - k\delta$, a buy limit order is placed at each price $p_t - \delta$,..., $p_t - k\delta$ for a fixed number of X shares. If the buy limit order is executed, the Agent will hold kX. When p_t rises to $p_{t+1} = p_t + k\delta$, a sell limit order is placed at each price $p_t + b - k\delta$,..., $p_t + b - \delta$ for a fixed number of X shares. If the sell limit order is executed, the Agent will profit from the sell by kbp_t. So the Agent is trying to benefit from the price fluctuations by buying low and selling high inside the window.

Every strategy is valued based on portfolio $V_t = C_t + p_t H_t$ at time step t, where C_t defines the cash and H_t defines the shares that Agent is holding. So V_t is the amount of cash the strategy would have if it liquidated all holdings at the current market price. They assume that there is a bounded price volatility $\Delta \geqslant |p_{t+1} - p_t|$. This means that each strategy trades at most Δ shares at each trading period.

Using these spread based strategies, the authors created a low regret meta algorithm that applies online algorithms $\mathcal{A}$ for learning with expert advice with one expert corresponding to each strategy $S(b)$. In the beginning the distribution of the weights of the strategies are generated by $\mathcal{A}$ at t be w_t. At each time step t, when some orders are executed, the holdings are updated to $H_{t+1} w_t$ and cash to $(C_{t+1} - C_t) w_t$. Then for each strategy $S(b)$ the payoff is set to be $V_{t+a} - V_t$ and it is sent to $\mathcal{A}$ in order to update the distribution w_{t+1}. The regret of the meta algorithm is $\mathrm{Regret}(\mathcal{A}) = \frac{G}{2} \sum_{t=1}^{T} ||w_t - w_{t+1}||$, where $G = 2\Delta B + \Delta^2$.

Abernethy and Kale applied two online algorithms in Experts model to the meta algorithm.

They applied the classic Multiplicative Weights (MW) algorithm and the Follow-The-Perturbed-Leader (FPL). MW updates the weights using the rule: $w_{t+1} = w_t \frac{\exp(\eta_t(V_{t+a} - V_t))}{Z_t}$ where Z_t is the normalization parameter. With $\eta_t = \frac{1}{2G}\min\{\sqrt{\frac{\log N}{t}}, 1\}$ they proved that the regret is bounded by $13G\sqrt{\log(N)T}$. The distribution w_t for FPL is set to be $w_t = P_r[V_t(b) + p(b) \geqslant V_t(b') + p(b')]$, where $p(b)$ is a sample from exponential distribution with mean $\frac{1}{\eta}$. With $\eta = \frac{1}{2G}\sqrt{\frac{\log N}{T}}$ they prove that regret bounded by $7G\sqrt{\log(N)T}$.

For their experinments they used stock price data for the following stocks: MSFT (Microsoft Corporation stock), HPQ (Hewlett-Packard stock) and WMT (Walmart stock). They set N=10 window sizes quoted in cents with max window size B = 100. The set of the Window sizes, quoted in cents is the following: $\mathcal{B} = \{1, 2, 3, 4, 5, 10, 20, 40, 80, 100\}$ They impimented MW, FPL, simple FTL and a simple uniform averaging over all strategies and they compare their performance to the best strategy in hindsight. In their results the MW performed nearly as well as the best strategy and FPL did not perform that well. They also concluded that the best strategy is different every day, so they raised a need for creating an adaptive learning algorithm.

Spooner et al [17] developed a high-fidelity simulation using high-frequency historical data and they designed a temporal difference (TD) reinforcement learning Agent to perform market making. To create their simulation, they used historical data for 10 securities from 4 different stocks and they reconstructed a Limit Order Book.

Their Agent uses spread based strategies similar to the Abernethy and Kale [1]. Each strategy has two basic variables θ_a and θ_b, which define the distance between the midprice μ_t and the price the Agent will place his orders, $\mu_t + \theta_a$ for sell orders and $\mu_t - \theta_b$ for buy limit orders. There is also an option for a market order when the Agent wants to clear his inventory, due to constraints in how much shares it can hold. The reward function of the Agent is defined as PnL: $r_i = \Psi(t) = \psi(t)_a + \psi(t)_b + Inv(t)\Delta\mu$, where $\psi(t)_a$,$\psi(t)_b$ compute the money lost or gained from the executed orders of the Agent and the quantity $Inv(t)\Delta\mu$ represents the cash if it liquifies his holdings. The authors also use two alternative dampened definitions of reward: the Symmetrically dampened PnL $r_i = \Psi(t) - \eta Inv(t)\Delta\mu$ and the Asymmetrically dampened PnL $r_i = \Psi(t) - max[0, Inv(t)\Delta\mu]$.

In their setting, they also provide an evaluation of three different state space constructions and propose a linear combination of tile codings as their final representation. The tile codings use combination of three states: the Agent-state, the market-state and the full-state which is the combination of the previous two, [17] at 4.3. The learning algorithms they use are: Double Q-learning, Expected Sarsa, R-learning, Double R-learning and On-policy R-learning.

They compared their basic Agent, that uses fixed distances θ_a and θ_b i.e $\theta_a = \theta_b = 5$, with a similar to one of Abernethy and Kale [1]. The basic Agent used a state representation consisting only of the Agent-state and the non-dampenedPnL reward function in order to be more adaptive in choosing the right θ and it was trained using one-step Q-learning and SARSA. The basic Agent performed better, it had better average reward and it did not maintain a large amount of holdings.

Finally, they consider a consolidated Agent, which holds the best variants of the Agent. It uses the Asymmetrically dampened PnLreward function with a LCTC (linear combination of tile codings) state-space, trained used SARSA. The consolidate Agent outperforms the basic Agent and the Agent of Abernethy and Kale [1] and it holds smaller inventory.

T.Chakraborty and M.Kearns presented profitable Market Making algorithms with Mean Reversion. Mean reversion is a financial term for the assumption that a stock price will tend to move to the average price over time. Their algorithms are based on a online setting where their Agent observes the price and its holdings at every timestep and it places buy and sell limit orders.

The profit obtained from mean reverting models is $(\kappa - z^2)/2$ where z is the difference between opening and closing prices and κ is the absolute value of all local price movements. That means the algorithm of Chakraborty and Kearns is profitable when there is a large amount of local price movement, but only a small net changes in the price.

Their basic Market Making algorithm works as follows:

- There are t=1,...,T steps and the observed price P_t. Thus $P_0, ..., P_T$ is the asset price time series.
- At time t the algorithm cancels all unexecuted orders and places buy orders at prices $Y_t, Y_t - 1, .., Y_t - C_t$ and sell orders at $X_t, X_t - 1, .., X_t + C_t$ where C_t is the depth of the price ladder at time t

Based on the above basic model they prove that for any random walk $P_0, ..., P_T$ there is mean reverting towards $\mu = P_0$. The expected profit of the market making that sets $X_t = P_{t+1}$ and $Y_t = P_{t-1}$ is positive.

Apart from the basic model they also present two other mean reveting MM models, one based on Ornstein-Unlbeck Process, which is a canonical stohastic mean reverting proccess, and a stohastic mean reverting model that has been studied in finance literature, the Schwartz model.

Chapter 3

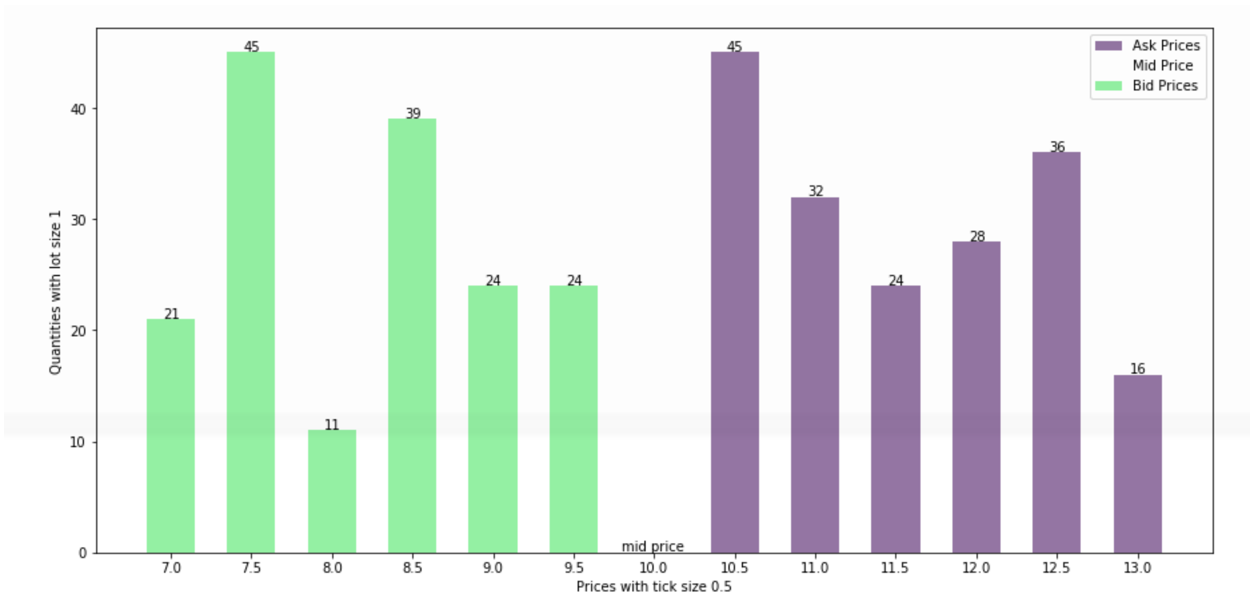
Developments

In this chapter, we present the algorithms that have been developed in order to conduct our experiments. At first we present the basic functionality of our custom LOB. Most of the functionality is taken by the theoritical works [9, 11, 10]. We also present the basic functionalities of our Automated Agent along with the strategies that it uses. Finally we show how we created a random generator that feeds the LOB with random orders.

For our developments we used Python. The LOB and the Automated Agent were created from scratch with the use of only some of basic libraries like Pandas,NumPy and Scikit-learn.

3.1 LOB representation

LOB can be considered as a data structure which consists of two queues: the ask order queue and the bid order queue. The ask order queue holds all sell orders that have not been matched yet and the bid order queue the buy orders that have not been matched yet.



Bid Queue

price	7.0	7.5	8.0	8.5	9.0	9.5
quantity	21	45	11	39	24	24

Ask Queue

price	10.5	11.0	11.5	12.0	12.5	13.0
quantity	45	32	24	28	36	16

Figure 3.1: Example of a LOB snapshot

In figure 3.1 a LOB snapshot is represented. The lot size is 1 and the tick size is 0.5. This snapshot only shows the LOB state from price 7.0 to price 13. The bid price is 9.5 and the ask price is 10.5. Therefore the spread is 1 and the mid price is 10.

Price changes

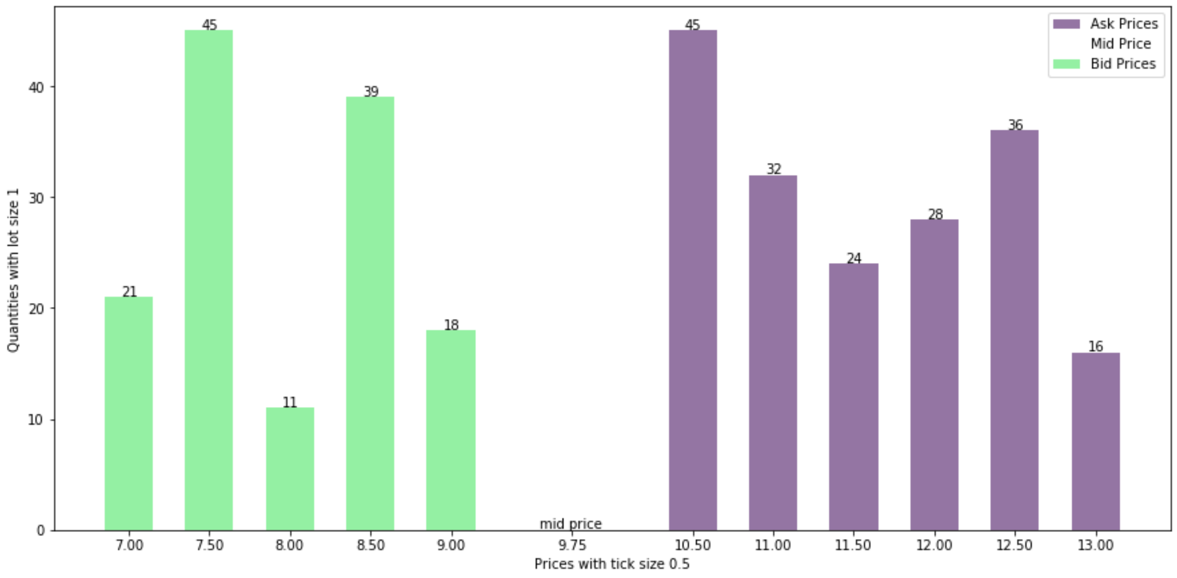
As the order flow progresses, the bid and ask price, the mid price and the spread change. The size of the price change depends on various things, such the trading volume which is associated to the number of the participants which are in the market and to the size of orders.

When a sell (or a buy) market order comes to the market with a specified quantity Q_i, it starts consuming the quantities of the bid price (or ask price) and continues until it consumes all the quantities of the order. For example at LOB snapshot at Figure 3.1, if someone places a sell market order with quantity Q_i 30, all quantities(24) at price 9.5 will be consumed and the remaining 6 will be consumed from price 9. So the new bid price will be 9.0 with quantity

18. Because of the bid price change, the bid-ask spread will change to 1.5 and the mid price to 9.75. The changes are presented at figure 3.2.

When a buy (or a sell) limit order is placed with price P_t and with quantity Q_t there will be two possible situations. Either the price P_t is less or equal to the bid price and as a result the order will be placed in queue waiting to match, or the price P_t is greater than or equal to ask price and thus there is a change to be matched. In the second case, the order will start to consume the quantities at that price in the ask queue and then will continue consuming the quantities at lower prices until all are consumed.

For example, in the LOB snapshot of Figure 3.2 if someone places a buy limit order at price 11.5 with quantity Q_i, there will be a match at that price. Then all quantities (24) at price 11.5 will be consumed and the remaining 6 will be matched and consumed by those at price 11. The changes are presented in the figure 3.3.



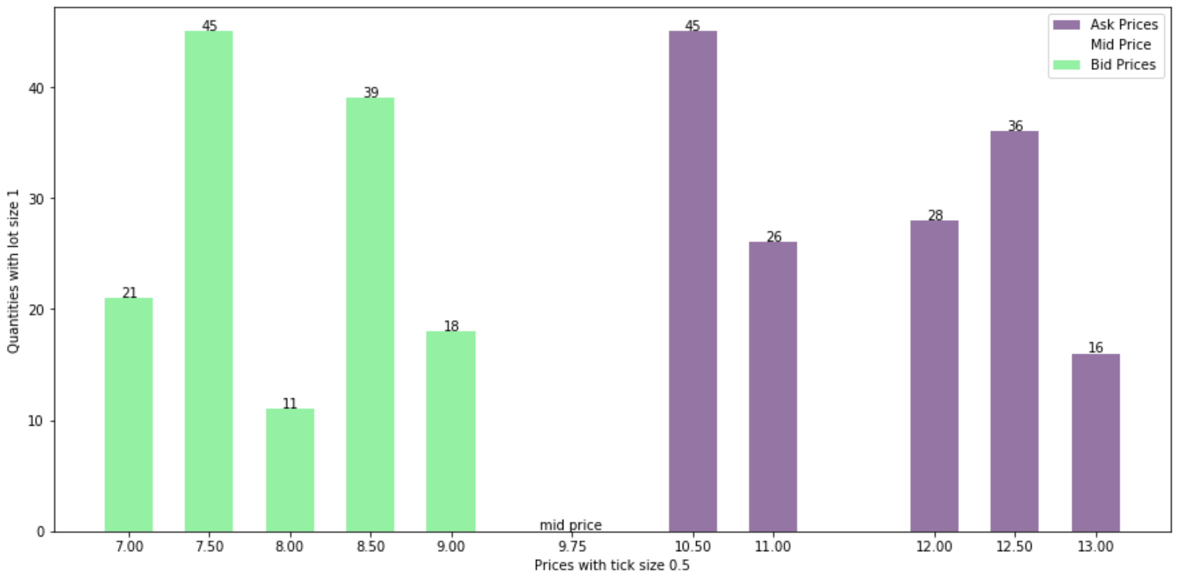
Bid Queue

price	7.0	7.5	8.0	8.5	9.0
quantity	21	45	11	39	18

Ask Queue

price	10.5	11.0	11.5	12.0	12.5	13.0
quantity	45	32	24	28	36	16

Figure 3.2: LOB snapshot after a sell market order



Bid Queue

price	7.0	7.5	8.0	8.5	9.0
quantity	21	45	11	39	18

Ask Queue

price	10.5	11.0	12.0	12.5	13.0
quantity	45	26	28	36	16

Figure 3.3: LOB snapshot after a sell market order

3.2 LOB Simulation

For the purpose of this work, a Limit Order Book simulation is created with the basic functionality. For the simulation we followed the work of Gould et al. [9] and from Kane et al. [11]. We also draw some ideas from the work of Huang et al. [10].

The simulation consists of two queues. The **ask queue** A_q holds the quantities of the sell orders that are placed at the corresponding level of the LOB and the **bid queue** B_q holds the quantities of the buy orders that are placed at the corresponding level in the LOB. Figures 3.1,3.2,3.3 give a definite representation of these queues. As shown on figure 3.2 for prices 8.0 and 11.0, we have $A_q[11.0] = 32$ and $B_q[8.0] = 11$.

We initialize the simulation with $m(t)$ and π. Based on these parameters, we calculate the starting values of $a(t)$, $b(t)$ and $s(t)$. Provided that the starting parameters are set, the LOB Simulation is ready to receive orders. There are two basic methods for placing orders: the Limit Order Method and the Market Order Method.

3.2.1 Limit Order Method

By placing a Limit Order, one has to specify the action, which can be sell or buy, the price, at which he wants to buy or sell, and the quantity of stocks.

For every action there are three possible situations:

- Placing a sell limit order at $p \geqslant a(t)$ (respectively placing a buy order at $p \leqslant a(t)$) for quantity q. In this case the given quantity is being placed in $A_q[p]$ (or in $B_q[p]$ if buy order).
- Placing a sell order (or a buy order) inside the spread (at $p \geqslant b(t)$ and $p \leqslant a(t)$). In this case p is placed in the queue $A_q[p]$ (or $B_q[p]$) and p becomes the new $a(t)$ (or the new $b(t)$ if buy order). There is also a change in $s(t)$.
- Placing a sell order (or a buy order) at $p \leqslant b(t)$ (or $p \geqslant a(t)$ if buy order) for quantity q, a transaction is executed at $B_q[p]$ (respectively a $A_q[p]$). If the quantity at the price level $B_q[p]$ (q_p) is less than q, the remaining quantity $q' = q - q_p$ will be consumed from the next price levels $B_q[p-\pi]$, $B_q[p-2\pi]$ until $q' = 0$. If all the available quantities $B_q[b(t)]$ are consumed, $b(t)$ will change to the next price level which has available quantities waiting to get consumed. The spread $s(t)$ will change also.

3.2.2 Market Order Method

By placing a Market Order one has to specify the action, which can be a sell or buy, and the quantity of stocks to be bought. When a sell order is submited (respectively buy order) with quantity q, q will be consumed from B_q (respectively A_q) starting from the price level of $B_q[b(t)]$ (respectively $A_q[a(t)]$) and it will continue to the lower price levels until q is consumed. If $B_q[b(t)] = 0$ (respectively $A_q[a(t)] = 0$), b(t) (respectively a(t)) and s(t) change.

3.3 Pseudo-random data creation

The data that we are using for this study have been created based on the "Algorithmic trading" Kaggle Dataset [6]. We analyzed this dataset in order to discover some insights about the distribution of the orders (buy and sell), the transaction volume and the price fluctuations. Based on the insights we have discovered, we created an algorithm that feeds the LOB simulation with pseudo random orders.

3.3.1 Data Analysis

The preprocessed Algorithmic trading dataset comprises observations at market prices before and after a liquidity shock (a trade that results in widening of the bid-ask spread). At each row the change of market prices for a certain security is represented before and after a liquidity shock. The data schema provided for each liquidity shock is the following:

- **Row id**: Is the unique identifier for each row (or the unique identifier for each liquidity shock.
- **Security id**: Is the unique identifier of the stock in which it happens the liquidity shock.
- **P_tcount**: The count of the previous day on market trades in current security.
- **P_value**: The sum of the previous day on market trades in current security.
- **Trade_vwap**: Volume-weighted average price of trade causing the liquidity shock.
- **Trade_volume**: Size of the trade causing the liquidity shock (number of stocks).
- **Initiator**: Defines whether the trade is initiated by a buyer or a seller.
- **Transtype<t>**:Defines whether the time-series event is a trade or a quote.
- **Time<t>**: Defines the event time.
- **Bid<t>**: The bid price b_t at event time t.
- **Ask<t>**: The ask price a_t at event time t.

Before we analyse the dataset we did some processing for our convenience.

1. We sorted the dataset per security id so we can work with only one stock at a time.
2. Then we sorted the events per event time t and we cleaned events that appeared twice.
3. The attributes of the order we focused on are: the **transtype**, the **bid price** and the **ask price**.

security_id	p_tcount	p_value	trade_vwap	trade_volume	initiator	transtype1
1	9154	6831386312	2395.0	887	B	Q
1	9154	6831386312	2398.0	748	B	Q
1	9154	6831386312	2398.0	151	S	T
1	9154	6831386312	2398.5	110	S	Q
1	9154	6831386312	2398.0	108	B	T

time1	bid1	...	bid96	ask96	bid97	ask97
08:00:20.799	2225.0	...	2397.0	2399.0	2397.0	2399.0
08:00:21.996	2393.0	...	2398.0	2399.0	2398.0	2399.0
08:00:29.200	2393.0	...	2397.0	2398.0	2397.0	2399.0
08:00:31.073	2393.0	...	2397.0	2398.0	2397.0	2399.0
08:00:31.867	2393.0	...	2398.0	2399.0	2397.0	2399.0

Figure 3.4: Table Algorithmic trading Dataset for stock with security id 1

time	transtype	bid	ask
08:00:20.799	Q	2225.0	2314.5
08:00:20.799	Q	2225.0	2314.5
08:00:20.799	Q	2225.0	2314.5
08:00:20.799	Q	2225.0	2314.5
08:00:20.799	Q	2225.0	2314.5
...	...	...	...
16:29:54.882	Q	2298.0	2299.5
16:29:54.882	Q	2298.0	2299.5
16:29:54.978	Q	2298.0	2299.5
16:29:54.997	T	2298.0	2299.5
16:29:54.997	Q	2298.0	2300.0

Figure 3.5: Dataset after cleaning the data

After the data cleaning we investigated the data in order to find some insights. We analyse the data from figure 3.5 for a single day and we found the following results 3.6.

p_tcount	p_value	trade volume average	Buyers	Sellers
1239	543554971	243.01	1333	1862

Number of Trades	Number of Quotes	Change of Bid Price	Change of Ask Price
7797	77420	7917	6814

Figure 3.6: After analysing the data for a certain day

3.3.2 Insights and probability distribution

After we had analysed the data, we used our findings (3.6) to create a propability distribution with which we feed to our algorithm. The insights we discovered are the following:

- **Percentage of Trades and Quotes** As shown on figure 3.6, on a certain day, the number of quotes is 7797 and the number of trades is 7797. This means that the 90% of the transactions are quotes and 10% are trades. So, in our random order generator a random order has 0.1 probability to be a trade and 0.9 propability to be a quote.
- **Percentage of Sellers and Buyers** The second insight is about the initiator of the order which causes the liquidity shock. As we see in figure 3.6 the number of buyers in a certain day were 1333 and the number of sellers were 1862. This means that ~60% of initiators were sellers and ~40% were buyers.
- **Change of Prices** The third insight is about how many times the bid and the ask prices have changed during the day. According to figure 3.6, the bid price b_t has changed 7917 times and the ask price a_t has changed 6814 times. Diving deeper to this insight, we counted how much they changed after a transaction, according to the tick size π. For example, according to figure 3.8 (top) a_t has changed to $-\pi$, 2826 times and to π, 2658 times. Also according to figure 3.8 (bottom) b_t has changed to -2π, 611 times and to 2π, 531 times.
- **Changes in Spread** Another insight is about spread. We calculated the spread prices depending on π. According to figure 3.7 spread was π 11451 times, 2π 26073 times, 3π 244665 times etc.

Spread/π	1	2	3	4	5	6	7	8	9	10	...
times	11451	26073	24465	13839	5469	2263	711	236	149	102	...

Figure 3.7: Frequency of spread values

Ask prices	...	-4	-3	-2	-1	1	2	3	4	...
times	...	63	99	331	2826	2658	447	137	70	...

Bid prices	...	-4	-3	-2	-1	1	2	3	4	...
times	...	118	224	611	2695	3358	531	175	85	...

Figure 3.8: Frequency of Ask and Bid Prices

We use all the above insights to feed our algorithm, in order to create random order data. Our goal is the data to be closer to reality.

3.3.3 Algorithm that creates random data

The algorithm uses a pre defined function from the python package numpy which is called choice(). This method takes as parameters the items that we want to choose and their probabilities. For example, to choose randomly whether an order is limit or market we use $choice(\{limit, market\}, \{0.9, 0.1\})$. The limit choice has 90% probability to be chosen and the market choice has 10% probability to be chosen.

If we combine the probability distributions from all the insights we have a random order. The probability of having a sell limit order where bid price changes and also the spread changes can be calculated as the product of the multiplication of the propabilities.

$$\text{Random Order} = P(limit) \times P(seller) \times P(\text{price changes}) \times P(\text{spread changes})$$

An easy way to represent the above calculation is with a decision tree. The first random choice is if the order is either a limit order or a market order. On the next node, the second random choice is if the initiator is a seller or a buyer. Above the initiator node there is the third random choice for the change of the price. Finally there is a fourth node for the spread change. The tree is represented in figure 1.4.

Apart from the main calculation of the random order creation, we have created other random choice distributions for more specific tasks. If the random order is a limit order, then it must be specified if it is either a sell limit order or a buy limit order. The quantity of the stocks also depend on the price of the order. For example the closer the order price is to the bid price or the ask price, the bigger the volume will be. If the order price is closer to the lower price levels (or to the higher price levels) the quantity will be very small.

For this reason, we divide the price of ask queue and the prices of bid queue into price sets. The price range may differ from one price set to another. For the A_q: the first price set can be $\{a(t), \ldots, p_r\}$ where $p_r > a(t)$; the second price set can be $\{p_r \ldots, p_{r+1}\}$ where $p_{r+1} > p_r$ etc. For the B_q: the first price set can be $\{p_r, \ldots, b(t)\}$ where $p_r < b(t)$, the second price set can be $\{p_{r+1}, \ldots, p_r\}$ where $p_{r+1} < p_r$ etc. For each price set we have a quantity distribution from which the algorithm will choose a random quantity number. For the first set the quantity range will be (Q_1, Q_2) where $Q_1 > Q_2$. For the second set the quantity range will be (Q_2, Q_3) where $Q_2 > Q_3$ etc. The further away the price set is from the $a(t)$ or the $b(t)$, the smaller the quantities in the quantity range will be.

With the above separation in price sets, we can control more efficiently how random orders will be distributed over the price range. For example, if we assign a large probability to the price set that contains the bid price (or the ask price), most of the generated random orders will be very close to the bid price (or the ask price).

Price and Spread change. Our algorithm uses the price change distributions to choose randomly if the price changes and how much it changes. Based on our distribution, a small price change (e.g. $p \pm \pi$) has higher chance to occur rather than a large price change (e.g. $p \pm 5\pi$). The change of spread depends on price change. If there is a sell order with price change of $+3\pi$, that means the a(t) will be $a(t) = a(t) + 3\pi$. This will have an impact on the spread price, so $s(t)$ will grow to $s(t) = s(t) + 3\pi$. If there is a sell order with price change of -3π and $s(t) > 3\pi$, $a(t)$ will become $a(t) = a(t) - 3\pi$ and $s(t)$ will change also to $s(t) = s(t) - 3\pi$.

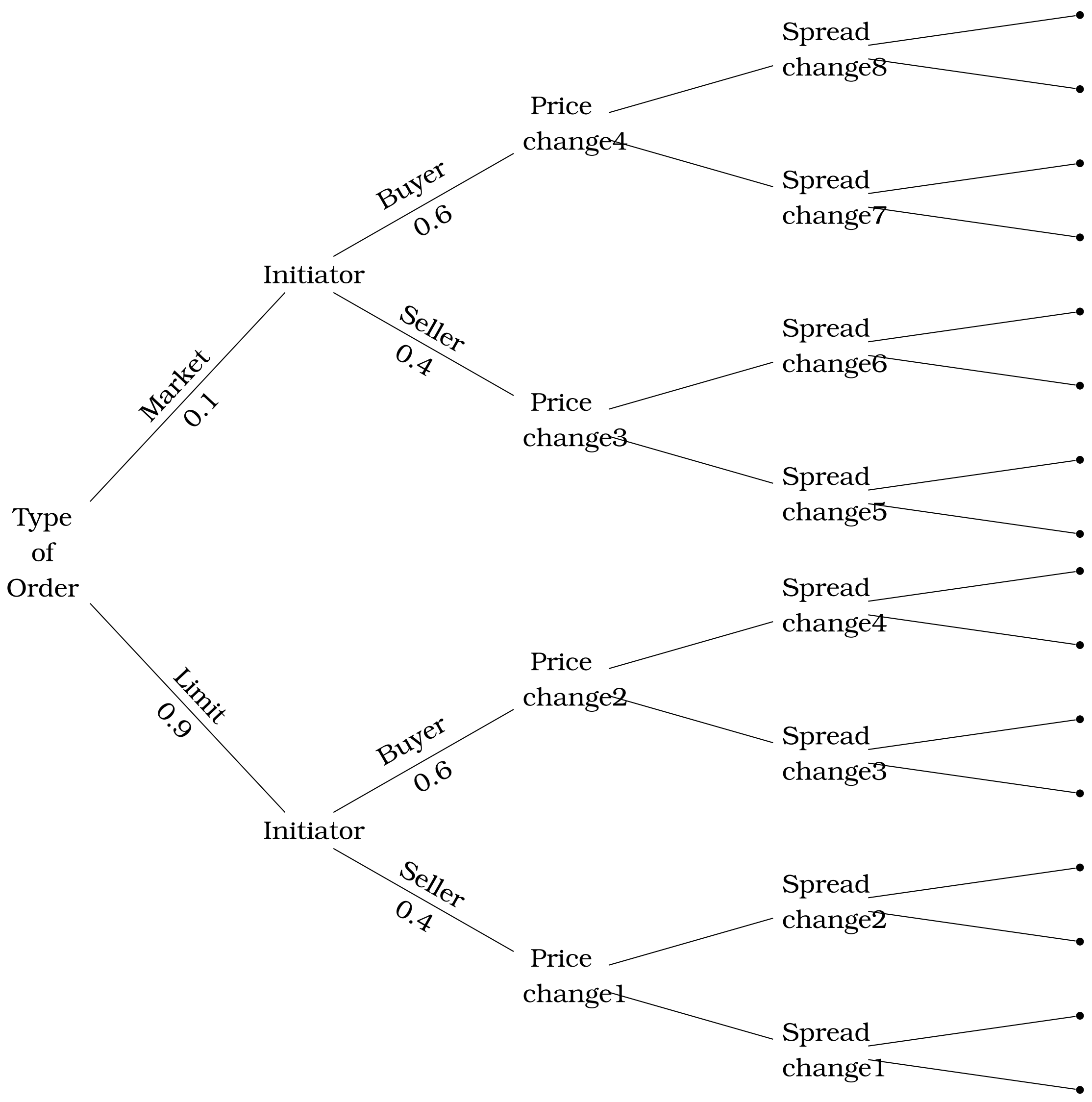


Figure 3.9: Probability Tree for random orders

Random orders balancers. If we run the algorithm and let it produce random orders for a large amount of steps, we will face some issues. In the first steps of the algorithm, there are not many orders that wait to be consumed in A_q and B_q. If our algorithm produces an order that is going to consume a large amount of stocks from a queue, then it will empty the queue and the price is going to change dramatically. For example, if we have $B_q = \{(vol = 15, price = 48.5), (vol = 30, price = 50), (vol = 50, price = 50.5)\}$ and our algorithm produces a sell market order $(sell, volume = 100)$, then B_q will be empty and $b(t)$ will be undefined. For this reason, we have created a *"balancer"* which, for the first N steps of our algorithm creates the same amount of sell and buy limit orders close to starting price, with no consumption of any queue taking place.

Similarly to the above issue, since our algorithm produces random data, at some period of time it might produce more quantities for one queue than for the other, e.g. $S_B = \sum_{n=1}^{\mathbb{N}_B} B_q[p] >> S_A = \sum_{n=1}^{\mathbb{N}_A} A_q[p]$. For this reason, we have created another balancer in our Algorithm that monitors when the difference between S_B and S_A exceeds a certain limit. If that happens, the algorithm produces only limit orders for the queue that has the fewest quantities.

Another issue concerns spread. As we have already mentioned, the spread changes when $a(t)$ or $b(t)$ changes. If for example $a(t)$ changes n times continuously by the amount $\kappa\pi$ times ($a(t) = a(t) + \kappa\pi$ where $\kappa = 1, 2, ...$) and $b(t)$ changes n times continuously by the amount $\lambda\pi$ times ($b(t) = b(t) + \lambda\pi$ where $\lambda = 1, 2, ...$)) that means $s(t)$ will grow like: $s(t) = s(t) + \lambda\pi + \kappa\pi$. In the real world spread rarely grows more than 10π. So if $\lambda + \kappa >> 10$, $s(t)$ becomes very large and that does not respond to reality at all. For this purpose, we have created a balancer in our Algorithm that monitors $s(t)$ and, if it becomes very large, we increase the possibility that the algorithm will produce an order inside spread and at the same time we decrease the possibility that the algorithm will produce an order that increases $a(t)$ (or decreases $b(t)$). This is achieved by assigning greater probability to the price set that contains prices inside spread.

3.4 Agent Representation

A core component in our study is the Market Maker which we call Agent. The Agent is interacting with our LOB simulation by placing orders. Its goal is to maximize its profit (cash $\mathcal{C}$) and also to minimize its exposure, i.e not to hold a large amount of stocks (holdings $\mathcal{H}$) at the end of a trading period.

At the beginning of a trading period, the Agent has a cash budget $\mathcal{C}$, which it can use to place buy orders, and a number of stocks $\mathcal{H}$ which he can sell in order to increase $\mathcal{C}$. When we initialize our Agent for the first time we supply it with a certain amount of cash $\mathcal{C}_0 > 0$ and it does not hold any stocks $\mathcal{H}_0 = 0$.

At every time step t, the Agent observes the market price p_t, and depending on his $\mathcal{C}$ and $\mathcal{H}$ it chooses what order to place. It can place either a sell (respectively a buy) limit order L_t or a sell (respectively a buy) market order M_t. With a L_t it has to specify the price and the quantity $\mathcal{X}$ it wants to place the order. Then it has to wait for the L_t to execute. A M_t is a more direct order, and the Agent has to specify only the quantity $\mathcal{X}$ it wants to sell or to buy.

In order to place the right order, the Agent should keep track of its $\mathcal{H}$ and its $\mathcal{C}$. For our convenience we divide them in two features each:

- **The available** $\mathcal{C}_{avl}$ is the cash that the Agent can spend in order to buy stocks.
- **The available** $\mathcal{H}_{avl}$ are the stocks which have not been placed for sale yet.
- **The overall** $\mathcal{C}_o$ is the $\mathcal{C}_{avl}$ plus the cash he has placed with a limit buy order which has not been executed yet.
- **The overall** $\mathcal{H}_o$ are the $\mathcal{H}_{avl}$ plus the stocks that have been placed for sale with a sell limit order and have not been executed yet.

Every time the Agent places a buy limit order at price p for X stocks inside the order book, the $\mathcal{C}_{avl}$ is reduced by pX, $\mathcal{C}'_{avl} = \mathcal{C}_{avl} - pX$. When this order is finally executed, the $\mathcal{C}_o$ is reduced by pX too, $\mathcal{C}'_o = \mathcal{C}_o - pX$. At the same time the $\mathcal{H}_o$ and $\mathcal{H}_{avl}$ are increased by X, $\mathcal{H}'_o = \mathcal{H}_o + X$, $\mathcal{H}'_{avl} = \mathcal{H}_{avl} + X$. On the other hand, when the Agent places a limit order at price p for X stocks , the $\mathcal{H}_{avl}$ is reduced, $\mathcal{H}'_{avl} = \mathcal{H}_{avl} - X$. When that order is finally executed, the $\mathcal{H}_o$ is reduced by X, $\mathcal{H}'_o = \mathcal{H}_o + X$, and available and $\mathcal{C}_o$ increased by pX, $\mathcal{C}'_o = \mathcal{C}_o + pX$. When there is no $\mathcal{H}_{avl}$ Agent can not place a sell order and if there is no $\mathcal{C}_{avl}$ he can not place a buy order.

Except for $\mathcal{C}_o$,$\mathcal{C}_{avl}$, $\mathcal{H}_o$, $\mathcal{H}_{avl}$, another important quantity that defines the overall state of the Agent or its position in the LOB, is the portfolio $\mathcal{V}$.

$$\mathcal{V} = \mathcal{C}_{avl} + a_t\mathcal{H}_o \text{ where } a_t \text{ is the ask price.}$$

$\mathcal{V}$ is the the sum of the $\mathcal{C}_{avl}$ the Agent has plus the cash it would get if it sells all its $\mathcal{H}_o$ at the current sell price.

3.5 Window Based Strategies

Our Agent uses window-based Strategies S for market making, inspired from the spread-based strategies of Abernethy and Kale [1]; however, in defining the window, we use as a point of reference the bid price instead of the ask price. We define a window size $w_s = i\pi$, where $i = 1, 2, ...$ and π is the tick size, and $\mathcal{B}$ which is the maximum w_s. The left bound of the window w_L is defined by current market price p_t and the right bound of the window w_R is defined by $p_t + w_s$.

$$Window\ W : [w_L, w_R] = [p_t, p_t + w_s]$$

In our implementation we use bid price b_t as the p_t, so the starting window W_0 is $[b_0, b_0{+}w_s]$. At time step $t+1$ $S(w_s)$ observes b_{t+1}. If $b_{t+1} < b_t$, the Agent places buy limit orders at all prices $[b_{t+1}, b_t)$ for a fixed number of shares $\mathcal{X}$. If $b_{t+1} > b_t + w_s$, the Agent places sell limit orders at all prices $[b_t + w_s, b_{t+1} + w_s)$ for a fixed number of shares $\mathcal{X}$.

Algorithm 9 Window Strategy

```
Initialize parameters $w_s$, $\mathcal{X}$
Initialize window $w_L = b_0$ and $w_R = b_0 + w_s$
for t = 1,...,T do
    Observe $b_t$
    if $b_t < w_L$ then
        $w_L = b_t$, $w_R = b_t + w_s$
        $d_B = (w_L - b_t)/\pi$
        Place $L_B(\mathcal{X})$ at prices $b_t - \kappa\pi$ where $\kappa = 1, .., d_B$
    end if
    if $b_t > w_R$ then
        $w_R = b_t$, $w_L = b_t - w_s$
        $d_S = (b_t - w_R)/\pi$
        Place $L_S(\mathcal{X})$ at prices $b_t + w_s + \kappa\pi$ where $\kappa = 1, .., d_S$
    end if
end for
```

At time step t, if b_t drops to $b_{t-1} - \kappa\pi$, the Agent places buy limit orders L_{B_t} at prices $b_{t-1} - \pi$, $b_{t-1} - 2\pi$,..., $b_{t-1} - \kappa\pi$ for $\mathcal{X}$ shares. If all orders L_{B_t} are executed then the Agent will hold $\mathcal{H}_t = \mathcal{H}_{t-1} + \mathcal{X}\kappa$ shares and his cash will be $\mathcal{C}_t = \mathcal{C}_{t-1} - \mathcal{X} \cdot \{(b_{t-1} - \pi) + (b_{t-1} - 2\pi) + ... + (b_{t-1} - \kappa\pi)\}$. At time step t+1, if b_{t+1} rises to $b_t + w_s + \kappa\pi$ the Agent places sell market orders L_{S_t} at prices $b_t + w_s - \pi$, $b_t + w_s - 2\pi$,..., $b_t + w_s - \kappa\pi$ for $\mathcal{X}$ shares. If all orders L_{S_t} are executed then the Agent will hold $\mathcal{H}_{t+1} = \mathcal{H}_t - \mathcal{X}\kappa$ shares and his cash will be $\mathcal{C}_{t+1} = \mathcal{C}_t + \mathcal{X} \cdot \{(b_t + w_s - \pi) + (b_t + w_s - 2\pi) + ... + (b_t + w_s - \kappa\pi)\}$. So the Agent bought $\mathcal{X}$ shares at time step t and sold them at higher price receiving a profit of $\kappa w_s \mathcal{X}$.

The intuition behind our Window Strategies is that the Agent waits for the price to drop below window so he can buy stocks at lower price. Then it waits for the price to rise above window in order to sell the stocks he holds at higher price so he profits from the window size w_s.

In order to understand these strategies better, let's consider a situation where: tick size $\pi = 0.1$, bid price $b_t = 47.1$, window $w_s = 0.5$, Agents overall Cash $\mathcal{C}_o = 200\$$ and $\mathcal{X}$=1. At first, window is [47.1,47.6] and $\mathcal{C}_o = 200\$$. If b_t drops by 3π, $b_t' = b_t - 3 * 0.1$, window becomes [46.8,47.3]. Then the Agent places buy limit order for 1 stock at prices 47.1, 47.0, 46.9. When these orders get executed Agent will hold 3 stocks and will have $\mathcal{C}_o' = 200 - 47.1 - 47.0 - 46.9 = 59\$$. If then b_t rises by 3π, $b_t' = b_t + 3 * 0.1$, window becomes [47.1,47.6] again. Then the Agent will place sell limit order for 1 stock at prices 47.3, 47.4, 47.5. When these orders get executed Agent will have $\mathcal{C}_0' = 59 + 47.5 + 47.4 + 46.3 = 201.2\$$. So at the end the Agent will benefit for that price change by $1.2\$$

A Strategy's performance can be measured based on three things.

- **Cash** $\mathcal{C}$: The amount of cash the strategy has lost or gained within a trading period.
- **Exposure or Holdings** $\mathcal{H}$: The amount of stocks remained on Agents hands following the Strategy. The more stocks Agents have at the end of the period, the more exposed the Agent is in the Market. The problem with exposure is that the Agent might have to sell the stocks at a lower price in order to have cash immediately.
- **Portfolio** $\mathcal{V}$: The amount of cash the strategy would have if it liquidated all holding at the current ask price a_t. Measure performance with $\mathcal{C}$ or $\mathcal{H}$ alone is essential but can be misleading. For example if S at the end of the trading period has: $\mathcal{C}_T$<$\mathcal{C}_{T-1}$, $\mathcal{H}_T$>$\mathcal{H}_{T-1}$ and a_T>a_{T-1} can be better than $\mathcal{C}_T$>$\mathcal{C}_{T-1}$, $\mathcal{H}_T$=$\mathcal{H}_{T-1}$ and a_T<a_{T-1}.

Chapter 4

Experimental Evaluation

In this chapter we present our experimentation framework with which we conducted our experinments. This framework consists of: the Agent, which interacts with the LOB by placing sell and buy order and a set of Strategies which the Agent follows in order to gain profit. In order to follow the most profitable Strategy, the Agent uses an Online Reinforcement Learning Algorithm which applies every strategy as a bandit. Then we present the results of our experinments. We run our framework with three bandit Online Algorithms: the E_n-greedy, the UCB1 and the EXP3.

4.1 Experimentation Framework

We have developed a framework based on reinforcement and online algorithms $\mathcal{A}$ which use the above window strategies. Our goal is to find which $\mathcal{A}$ will bring in the most profits. In contrast with Abernethy and Kale [1], who applied Expert Advice Algoritms to their framework, we apply Bandit Algorithms and more spefically we use the E_n-greedy, the UCB1 and the EXP3.

Our framework works as follows:

- The duration of each experiment is finite and is divided in trading periods. Each time period is divided in time steps.
- We have a set of N Strategies $S_n = \{S_1(w_{s_1}), S_2(w_{s_2}), ..., S_n(w_{s_n})\}$ with different window sizes each. The performance of each Strategy is calculated with the average $\overline{R_i}$ with $i = 1, .., n$. R_i is the difference between Agents portfolio $\mathcal{V}_T$ at the start of a trading

period and its portfolio at the end of that trading period $\mathcal{V}_{T+1}$, $R_i = \mathcal{V}_{T+1} - \mathcal{V}_T$. So $\overline{R_i} = \sum_{j=1}^{m} R_j$ where m defines the number of times that a strategy has been chosen.

- There is an Agent which interacts with the LOB and its main goal is to use S_n in the most profitable way. The Agent chooses which strategy to follow from S_n at the beginning of each time period and it places orders based on that strategy. It also places orders based on his $\mathcal{C}_{av}$ and $\mathcal{H}_{av}$, in the sense that if $\mathcal{C}_{av}$=0 it cannot place buy orders and if $\mathcal{H}_{av}$=0 it cannot place sell orders.
- We assign an Algorithm $\mathcal{A}$={$E_n greedy, UCB1, EXP3$} to our Agent, which instructs him which S_n should play at each trading period.
- At every trading period T the Agent receives an $S_i(w_i)$ where $i = 1, ..., n$ from $\mathcal{A}$.
- At every time step t, where $t = 1, ..T$, the Agents trades inside the LOB based on the strategy $S_n(w_n)$. At every buy or sell order $\mathcal{C}_o$, $\mathcal{C}_{av}$, $\mathcal{H}_o$ and $\mathcal{H}_{av}$ are change respectively.
- At the end of T the portfolio $\mathcal{V}_T$ is sent to $\mathcal{A}$ which updates the $\overline{R_i}$ of $S_i(w_i)$ where $i = 1, ..., n$.

Algorithm 10 Reinforcement Framework

```
Initialize Agents parameters $C_T$,$H_T$,$V_T$
Choose Strategies $S = \{S_1(w_{s_1}), S_2(w_{s_2}), ..., S_n(w_{s_n})\}$ with windows $w_n$
Set $\hat{R_i} = 0$ for each $S_n$.
for n = 1,...,T do
    Algorithm $\mathcal{A}$ selects $S_n$
    for t = 1,... do
        Agent is trading following the $S_n$ Strategy
    end for
end for
```

4.2 Experinments

We conducted experiments with the same dataset with order data which we created with the random order algorithm. The whole dataset consists of 3200000 random orders which are divided in smaller datasets of 2000 random orders each. We called these smaller datasets trading periods. 400 trading periods form a full trading day. So we run our experinment for 4 trading days.

At each trading period T the Agent uses only one window based strategy $\mathcal{S}$. We have N = 10 strategies with window sizes $W_s = \{0.1, 0.7, 1.3, 1.7, 2.2, 2.6, 3.1, 3.7, 4.4, 5\}$. At the end of T we calculate the reward $\mathcal{R}_i$ of $\mathcal{S}$ as the difference between the porfolio $\mathcal{V}$ of the current T and the $\mathcal{V}$ of the previous period.

$$\mathcal{R}_i = \mathcal{V}_T - \mathcal{V}_{T-1}$$

We divide each trading day into 400 periods which consist of 2000 orders each. That means, each Strategy will be played for 2000 orders. That number of orders will be enough in order to understand how the performance of the strategy is.

We also calculated the remaining cash $\mathcal{C}_o$ of that day and also the stocks $\mathcal{H}_o$ that the Agent kept in its hands and did not manage to sell at a price better than the price in which it bought them. We initialize the Agent $\mathcal{C}_0 = 10000\$$ and $\mathcal{H}_0 = 0$. Consequently $\mathcal{V}_0 = 10000$.

We used our Reinforcement framework along with three different basic algorithms $\mathcal{A}$: ϵ-Greedy, UCB1 and EXP3. Then we compared the performance of these three algorithms inside our framework. The performance was measured in the basis of:

- **Average reward** $\overline{R_i}$ The Average reward that the algorithm $\mathcal{A}$ produced at the end of the experiment
- **Agents assets** How the Agent managed $\mathcal{C}$ $\mathcal{H}$ over time under Algorithm $\mathcal{A}$ based on how prices of the stock performed.
- **The Regret** We define our notion of Regret as how our framework will perform with $\mathcal{A}$, versus how it would have been performed if it only chooses the best strategy i.e. the strategy with the highest $\overline{R_i}$.

Price fluctuation

As we observe in Figure 4.1, the price gradualy decreases through execution.

- On **day one** (steps 0.0-0.8) price starts at 50$ and at the end of the day it is down to 42.2$. During the day, price shows some mild fluctuations with price reaching max at 51+$ two times and went 3 times under 45$.
- On **day two** (steps 0.8-1.6) price starts at 42.2$ and at the end of the day it is down to 36.0$. In the beginning, price rises slighly to near 49$, then it has a huge drop to 32 $ and in the end, it slightly rises to 36.0$.

- On **day three** (steps 1.6-2.4) price starts at 36.0$ and at the end of the day it is down to 32.5$. Price gradually decreaces till it reaches 27$ in the middle of the day. Then it rises up to 32.5$.
- On **day four** (steps 2.4-3.2) price starts at 32.5$ and at the end of the day it is down to 27.05$. Price gradually decreaces till it reaches the lowest point of all time 24 $, then it has a slight rise to 29$ and ends up at 27.05$.

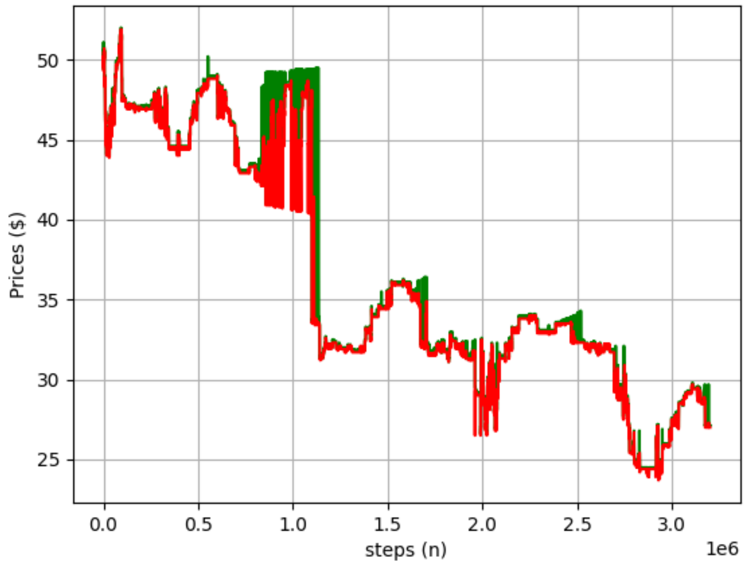
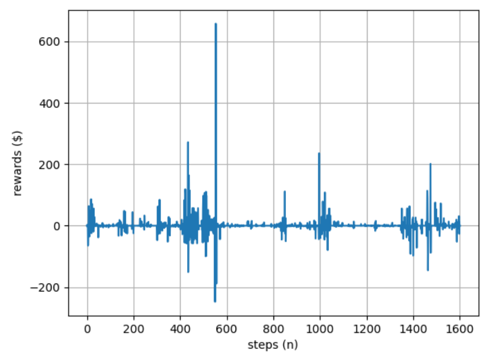


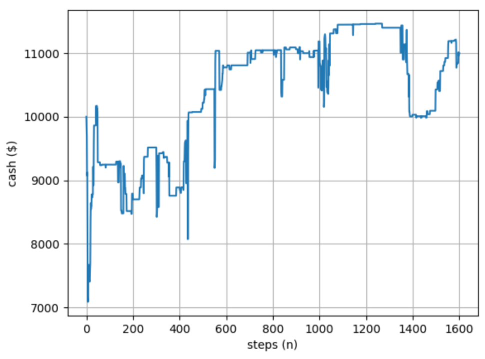
Figure 4.1: How price change during the execution. Steps(n) in millions.

Best Strategy

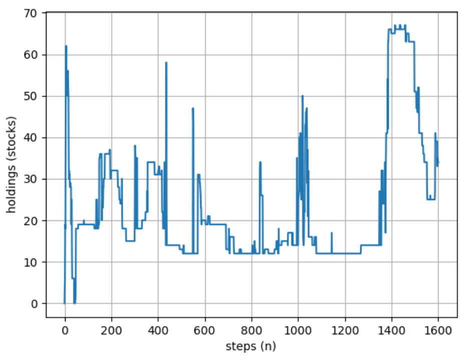
To calculate the Regret, we compare the performance of our framework with $\mathcal{A}$ opposed to the Strategy that performs better during the execution of our experiments. The strategy which performs better is the one with window size $w_s = 0.7$. With this Strategy the Agent perfoms with an Average Reward of 1.20. Also at the end of execution the Agent holds $\mathcal{C}_{\text{final}} = 11000\$$ and $\mathcal{H}_{\text{final}} = 34$ stocks. We can see the exact performance of the Agent at the following figures 4.2.



(a) Rewards



(b) Cash



(c) Holdings

Figure 4.2: Rewards, Cash and Holding of the Strategy with $w_s = 0.7$ during the execution)

4.3 Window based Strategies with E_n-greedy

We used $\mathcal{A} = \epsilon_n$-Greedy algorithm with $c = 3$ and $d = 0.6$. At each T, ϵ_n-Greedy chooses a Strategy and at the end of T, it receives the reward and it updates the reward of that action. The algorithm performs with an Average Reward of 1.39. Also at the end of the experiment the Agent holds $\mathcal{C}_{final} = 11030.7\$$ and $\mathcal{H}_{final} = 53$ stocks.

Rewards $\mathcal{R}$ As we observe in Figure 4.4a, Agent's Reward with E-greedy fluctuates much at some periods and at other periods it is close to 0. The most common range where Reward lays at is [-100,100]. On the first day (steps 0-400), the reward lays at between [-100,100] and Agent has not any extreme outcomes. On the second day (steps 400-800), on the first half (steps 400-600) Agent receives some extreme values. That happens because the price has some mild ups and downs so the Agent wants to make more orders. On the second half (steps 600-800) the Reward is very small, due to large price drop. When price drops that much, the environment is very unstable for the Agent so it can place many orders. On the third day (steps 800-1200), the Reward fluctuations rise gradually as the Agent adapts to the previous days price drop and the environment is more stable. On the fourth day (steps 1200-1600), the reward outcomes are low at start, due to another price drop. Then price becomes more stable and rises so the Agent places some orders.

Cash $\mathcal{C}$ Agent's cash on the first day (figure 4.4b, steps 0-400) is most of the time slighly below starting cash $\mathcal{C}_0$, around 9000\$ and only in the middle of the day $\mathcal{C}$ drops at 6000 \$. At the end of the first day, the Agent ends up with $\mathcal{C}_{400}$ 8100\$. On the second day (steps 400-800), $\mathcal{C}$ rises drammatically and stays most of the day over $\mathcal{C}_0$ and reaches 11000 at its peak. That happened because of the huge drop of price at the start of that day. In the end, the price rises so the Agent might benefit from that by selling stocks which has been bought at a lower price. On day three (steps 800-1200), there is a huge drop on the $\mathcal{C}$ (it reaches 7500\$) and that might happen because price had a mild drop so he bought some stocks. On day four (steps 1200-1600), $\mathcal{C}$ is stable over 11000 most of the day. In the middle of that day ,when $\mathcal{C}$ dropped to 8000\$,because the price started drop so Agent started buying stocks . That day finished with the Agent having 10650\$.

Holdings $\mathcal{H}$ with ϵ_n-Greedy (figure 4.4c), during the period of four days , have many fluctuations. The largest $\mathcal{H}$ which is observed is 150 stocks and the most common range of its $\mathcal{H}$ is between [18,60]. On the first day (steps 0-400), the Agent buys and sells stocks constantly. The highest number of stocks it has held that day is 90 and it has not managed to have less than 20 stocks most of the time. On the second day (steps 400-800), $\mathcal{H}$ does not dropped under 18 stocks and it exceeds 45 very few times. That happens because at first price drops, so the Agent can not sell last day's stocks but can only buy, and when price increases a little, it sells very few stocks. On the third day (steps 800-1200), $\mathcal{H}$ increased very much and sometimes it exceeds 100, due to a mild price drop and then the stabilization of the environment. On the fourth day (steps 1200-1600), $\mathcal{H}$ reaches maximum (152 stocks) after the middle of the day. The Agent's holdings at the end of the day are 53.

Regret As we can see at Figures 4.4d, 4.4e, on the first day (steps 0-400), the cumulative Regret has an almost linear increase and finishes at 1850. On the first half of the second day (steps 400-600), the cumulative Regret increases very fast (Regret=4050) due to the large price fall. On the second half (steps 600-800), on the other hand, when the environment is more stable, the Agent performs much better and at the end of that day cumulative Regret finishes at 4100. Then on third day (steps 800-1200) the cumulative Regret increases until step 1050 to 6600. Then from step 1050 to step 1350 on the fourth day, it is stable. Finally on the second half of the fourth day and until the end, the cumulative Regret is again on increase and it finishes at 11000. In figure 4.4e mean Regret starts with a huge increase and then it drops very quickly to 5 and stays as it is till the end of the execution.

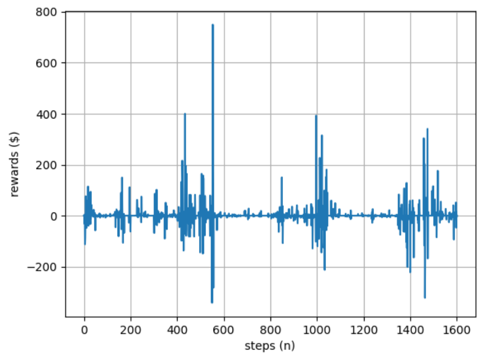
	Avg Reward	Max Reward	Min Reward	Max Cash	Min Cash	Max Holdings	Min Holdings
Day 1 (0-400)	0.65	149.9	-112.5	10000	5950	90	0
Day 2 (400-800)	3.45	749.7	-342.1	11000	7900	52	18
Day 3 (800-1200)	1.49	392.5	-212.7	11000	7500	142	20
Day4 (1200-1600)	-0.03	340.8	-322.5	11500+	7900	150	20

Figure 4.3: Reward,Cash and Holdings information at each day with ϵ_nGreedy

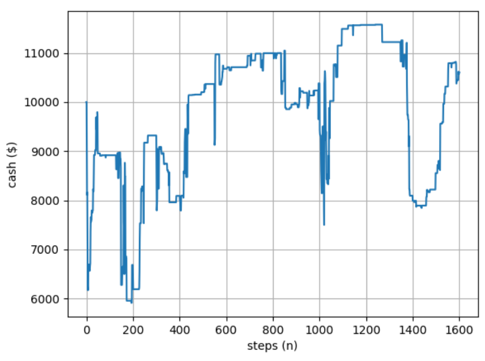
4.4 Window based Strategies with UCB1

We used $\mathcal{A}$ = UCB1 algorithm. At each T, UCB1 selects a Strategy and at the end of T, it receives the reward and it updates the bounds of the actions. The algorithm performs with an Average Reward of 0.92. Also, at the end of the execution, the Agent holds $\mathcal{C}_{final} = 10635.7\$$ and $\mathcal{H}_{final} = 30$ stocks.

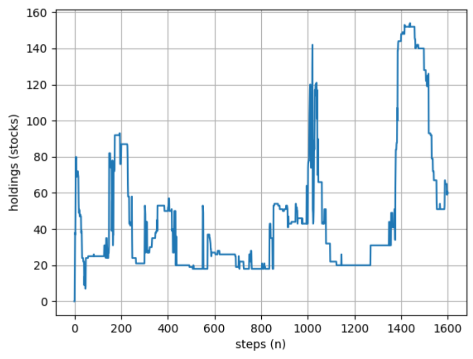
Rewards $\mathcal{R}$ As we observe in Figure 4.6a, Agents Reward with UCB1, The Reward fluctuates in the same way as with Egreedy but gradually the range of fluctuations decreases. Also maximum and minimum Reward outcomes are smaller than the outcomes observed with Egreedy. On the first day (steps 0-400), the reward lays between [-100,100] and there are not any extreme reward outcomes observed. On the second day (steps 400-800), on the first half (steps 400-600), the Agent receives some extreme values. That happens because the price has some mild ups and downs so the Agent wants to make more orders. On the second half (steps 600-800), the Reward is very small, due to large price drop. When price drops that much ,the environment is very unstable for the Agent so it can not place many sell orders. On the third day (steps 800-1200), the Reward fluctuations rise gradually as the Agents adapts to the previous days price drop and the environment is more stable. On the fourth day (steps 1200-1600) the environment is stable for the Agent so we can observe some mild fluctuations with no extreme outcomes.



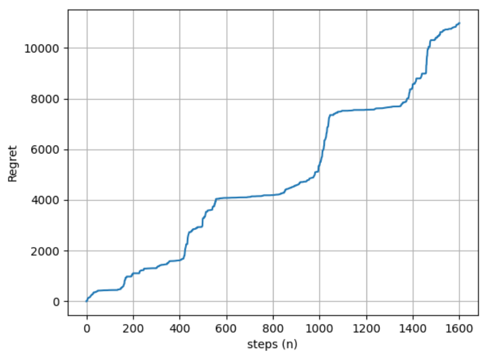
(a) Rewards



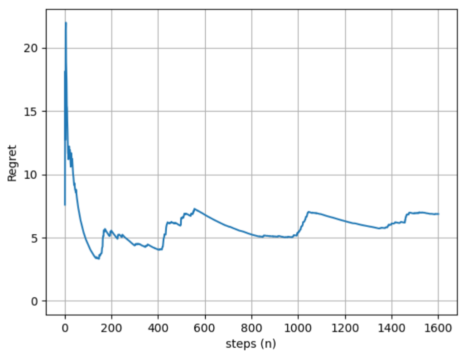
(b) Cash



(c) Holdings



(d) Cummulative Regret



(e) Average Regret

Figure 4.4: ϵ_n Rewards, Cash, Holdings, Cummulative and Mean Regret during the execution

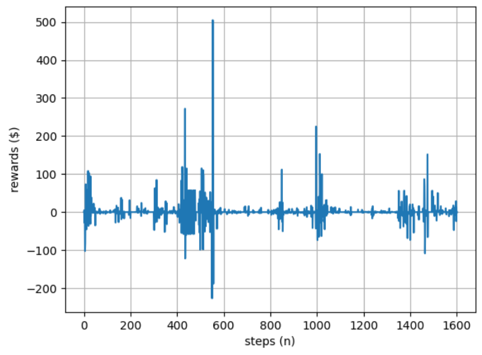
Cash $\mathcal{C}$ Agent's cash with UCB1, on the first day (figure 4.6b, steps 0-400),starts with a huge drop (that means Agent bought many stocks) $\mathcal{C}$<4000$. Then $\mathcal{C}$ rises and for the rest of the day it fluctuates between [8500$,9500$]. On the second day (steps 400-800), $\mathcal{C}$ rises over 10000$ and the rest of the day it fluctuates around 10500$. That happens because price drops dramatically so the Agent does not buy, due to the price drop. During the third day (steps 800-1200) $\mathcal{C}$ drops below 8000$ and at the end of the day it rises up to 11000$. Then on the fourth day $\mathcal{C}$ drops to 8000$. That happened because price drops in the middle of the day so Agent buys some stocks. The day finishes with Agent having 10635.7$.

Holdings $\mathcal{H}$ Agent's Holdings with UCB1 (figure 4.6c) during the four days period, have many fluctuations. The largest $\mathcal{H}$ observed is 82 stocks and the most common range of his $\mathcal{H}$ is between [18,25]. At the start of the first day (steps 0-400), Agent holds a large amount of stocks $\mathcal{H}$>60 and the rest of the day its holdings do not exceed 40. At the end of the day, it remains with 32 stocks. On the second day (steps 400-800), Agent's holdings fluctuate between [17,33]. It bought many stocks because a large price drop happens. On the third day (steps 800-1200), and after the price stabilizes a bit, $\mathcal{H}$ reaches the maximum number(84 stocks). On the fourth day (steps 1200-1600), $\mathcal{H}$ reaches another pick (50 stocks) in the middle of the day. The Agent's holdings at the end of the day are 30.

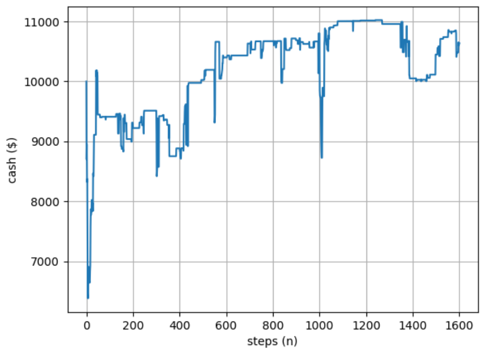
Regret As we observe in 4.6d, the cumulative Regret with UCB1 starts on the first day (steps 0-400) with an increase and in the middle of the day it stabilizes at 500. Similarly on the second day (steps 400-800) till step 570 it increases abruptly to 1400 but then it becomes stable. On the third day (steps 800-1200), cumulative regret is stable at start but at step 1000 it increases abruptly to 2000. From step 1100 to 1200 it remains stable at 2000. On the fourth day (steps 1200-1400), the first half of the day cumulative regret is stable and then it has a linear increase to 2500. As we observe in figure 4.6e mean Regret starts with a huge rise and then it drops very quickly to around 2.5 and stays as it is.

	Avg Reward	Max Reward	Min Reward	Max Cash	Min Cash	Max Holdings	Min Holdings
Day 1 (0-400)	0.57	108.2	-102.5	10100	3000	74	0
Day 2 (400-800)	2.18	504.9	-226.9	10500	8800	37	16
Day 3 (800-1200)	0.79	225.1	-73.8	11000	8800	83+	16
Day4 (1200-1600)	0.14	151.9	-108.6	11100	10000	50	16

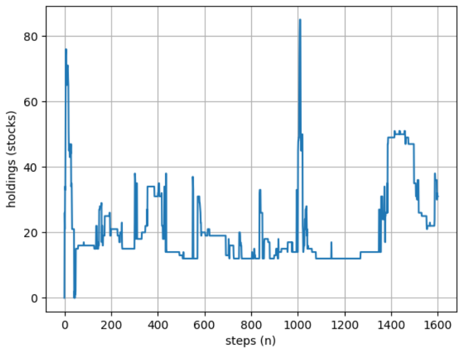
Figure 4.5: Reward,Cash and Holdings information at each day with UCB1



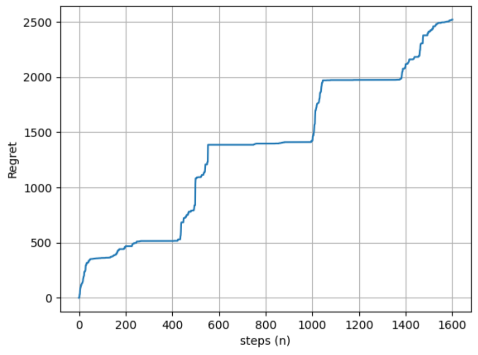
(a) Rewards



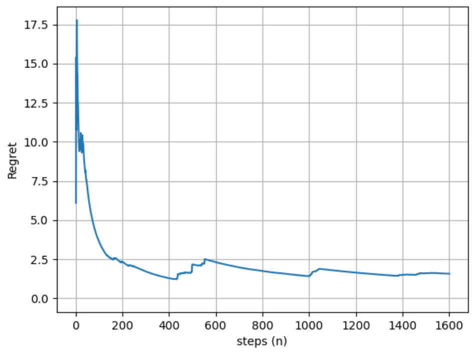
(b) Cash



(c) Holdings



(d) Cummulative Regret



(e) Mean Regret

Figure 4.6: UCB1 Rewards, Cash, Holdings, Cummulative and Mean Regret during the execution

4.5 Window based Strategies with EXP3

We use $\mathcal{A}$ = EXP3 algorithm. At each T, EXP3 selects a Strategy and at the end of T, it receives the reward and it updates the average reward of those actions. The algorithm performes with an Average Reward of 0.82. Also at the end of the execution the Agent holds $\mathcal{C}_{final} = 10302.69\$$ and $\mathcal{H}_{final} = 32$ stocks.

Rewards $\mathcal{R}$ As we can see in figure 4.8a, Reward fluctuates pretty similar with EXP3 as for Egreey and UCB. The only difference is that on the first two days the reward has many fluctuations, and some large Reward outcomes are also observed, and on the last two days the fluctuations are significally small. On the first day (steps 0-400), the reward lays between [-60,100] and there are not any extreme reward outcomes observed. On the second day (steps 400-800), on the first half (steps 400-600), the Agent receives some extreme values. That happens because the price has some mild ups and downs so the Agent wants to make more orders. The second half (steps 600-800) the Reward is very small, due to large price drop. When price drops that much ,the environment is very unstable for the Agent so he can place many orders. On the third day (steps 800-1200), the Reward fluctuations rise gradually as the Agent adapts to the previous days price drop and the environment is more stable. On the fourth day (steps 1200-1600) , the reward lays between [-50,120] and there are not any extreme reward outcomes observed.

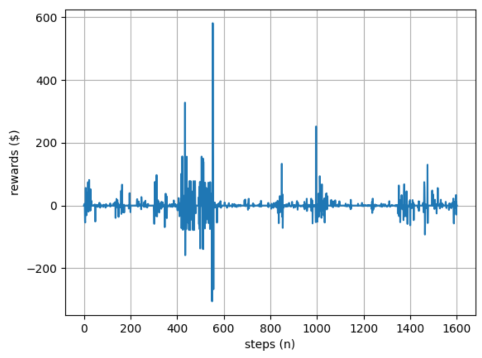
Cash $\mathcal{C}$ The Agent's Cash with EXP3 on the first day (figure 4.8b, steps 0-400) starts with a huge drop $\mathcal{C}$<7000$ and for the rest of the day it ranges around 9000$ with some mild droppings. At the end of the day it is $\mathcal{C}$=8550$. On the second day (steps 400-800), $\mathcal{C}$ rises up to 10000$ and stays there. That is because of a large price drop which occurs in the middle of the second day. The Agent does not want to do any buys because of the instability of the environment. On the third day (steps 800-1200), $\mathcal{C}$ has only some small fluctuations, only drops twice below 10000$. That means that Agent does not make many buys, except for the two situations that are mentioned above. On the fourth day (steps 1200-1600), $\mathcal{C}$ drops a little bit in the middle of that day due to some mild price dropping that occurs then.

Holdings $\mathcal{H}$ The Agent's Holdings with EXP3 during the four days period have many fluctuations (figure 4.8c). The largest $\mathcal{H}$ observed is 56 stocks and the most common range of his $\mathcal{H}$ is between [18,40]. On the first day (steps 0-400), the Agent's holdings have many fluctuations from 50 to 0. On the second day (steps 400-800), $\mathcal{H}$ again has many fluctuations but less extreme than the previous day and it ranges between [18,40]. On the third day (steps 800-1200), a large spike occurs (55 stocks) and that happens because of the stabilization of the price after the drop of the previous day. On the fourth day (steps 1200-1600), the Agent's holding are most of the day $\mathcal{H}$>30. The Agent's holdings at the end of the day are 32.

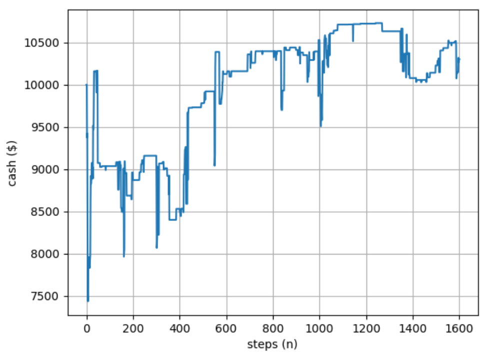
Regret As we observe in figure 4.8d, the cummulative Regret with EXP3 increases linearly on the first day (steps 0-400) and it finishes at 500. Then a huge increase occurs to 2500 from the start of the first day until the middle of that day. Then from the start of the third day (steps 800-1200) till the end (step 1600), the cummulative regret is gradually increasing linearly till it finishes at 3900. Similarly, in figure 4.8e mean Regret starts with a huge rise and then it drops very quickly to 1 in the middle of the first day. At the end of the second day, another small rise to 4 observed and it drops again quickly to around 3.

	Avg Reward	Max Reward	Min Reward	Max Cash	Min Cash	Max Holdings	Min Holdings
Day 1 (0-400)	0.45	97.2	-69.1	10100	7000	55	0
Day 2 (400-800)	2.07	581.4	-305.4	10400	8500	42	18
Day 3 (800-1200)	0.71	252.1	-71.4	10700	9500	55	18
Day4 (1200-1600)	0.03	130.2	-92.5	10700	10050	44	18

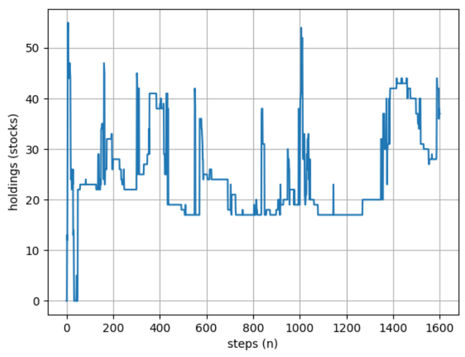
Figure 4.7: Reward,Cash and Holdings information at each day with EXP3



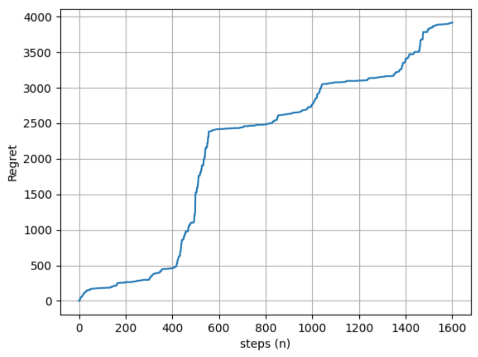
(a) Rewards



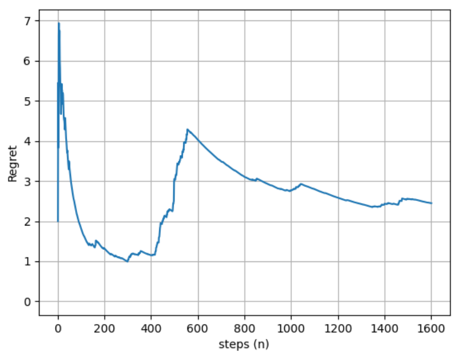
(b) Cash



(c) Holdings



(d) Cummulative Regret



(e) Mean Regret

Figure 4.8: EXP3 Rewards, Cash, Holdings, Cummulative and Mean Regret during the execution

4.6 Summary

As for the Rewards ϵ_n-greedy gives the best Average Reward (1.39) compared to UCB1(0.92) and EXP3(0.82), throughout the experiments. As we can observe from the respective figures (4.4a,4.6a,4.8a) that show the Reward at every step during the execution, ϵ_n-greedy gives some extreme outcomes at some steps, so it has an impact on the Average reward to be slighly higher than the other two algorithms. ϵ_n-greedy has also a better average Reward than the Best Strategy(1.20).

Again with ϵ_n-greedy the Agent remains with the best Porfolio $\mathcal{V}$ 12464.35$ with $\mathcal{C}$=11030.7$ and $\mathcal{H}$=53 stocks. It also performs better than the Best Strategy which has $\mathcal{V}$ 11919.7$ with $\mathcal{C}$=11000$ and $\mathcal{H}$=34 stocks. Next best porfolio is UCB1 with $\mathcal{V}$=11447.2$,$\mathcal{C}$=10635.7$,$\mathcal{H}$=30 and final portfolio is the EXP3 with $\mathcal{V}$=11168.29$,$\mathcal{C}$=10302.69$,$\mathcal{H}$=32.

As we observe from Figures (4.4c,4.6c,4.8c) about how much stocks the Agent holds at every step, with ϵ_n-greedy, the Agent holds large positions most of the time, which is dangerous, in the sense that it might not be able to sell the stocks or he migh sell them in a lower price. The Holdings of UCB1 and EXP3 are pretty much the same during the execution and no extreme values are observed.

	Average Reward	Final Cash ($)	Remaining stocks	Portfolio ($)
Best Strategy	1.20	11000	34	11919.7
E_n-Greedy	**1.39**	**11030.7**	**53**	**12464.35**
UCB1	0.92	10302.69	30	11447.2
EXP3	0.82	10302.69	32	11168.29

Figure 4.9: Comparing Algorithms on Reward, Cash, Holdings and Porfolio

The UCB1 Algorithm gives the best cumulative Regret=2500. It also finds the best strategy very quickly at each time period as we can see at mean Regret 4.6e. As we observe in figure 4.6d, after some increases of the cumulative Regret, there is always a large period where the cumulative Regret remains stable, which means that it follows the best strategy at that period. EXP3 performs a little bit worse than UCB1 with final cumulative Regret=3900. It also has less periods where cummulative Regret remains stable while it does not have extreme increases. E_n-greedy ends up with the worst performance of cummulative Regret=11000. It has some small periods where the cummulative Regret is stable, but most of the time it has large increases and that means it cannot easily find the best Strategy. Also the mean Regret of E_n-greedy.

	Average Regret E_nGreedy	Average Regret UCB	Average Regret EXP3
Day 1 (0-400)	4.04	1.29	1.14
Day 2 (400-800)	6.44	2.20	5.06
Day 3 (800-1200)	8.42	1.44	1.54
Day4 (1200-1600)	8.55	1.37	2.03

Figure 4.10: Comparing Algorithms on Average Regret at each day

	Cummulative Regret
E_n-Greedy	11000
UCB1	**2500**
EXP3	3900

Figure 4.11: Comparing Algorithms on Cummulative Regret

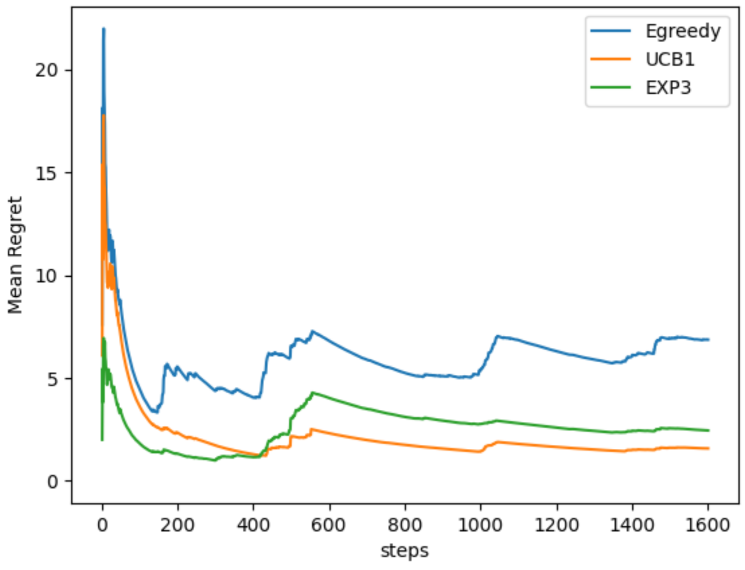


Figure 4.12: Comparison of mean Regret for Egreedy, UCB1 and EXP3

Chapter 5

Conclusions

In this thesis, an automated Agent has been developed in order to trade stocks. Furthermore, we proposed a class of window based strategies, inspired from Abernethy and Kale [1], based on which, the Agent tries to profit from the price changes.

The basic goal of the Agent is to explore which is the most profitable strategy at any given time period, and use it, as long as it remains profitable. For that reason, we applied a class of Online learning Algorithms ($\epsilon_n$greedy, UCB1 and EXP3) drawn from the bandit problem, in contrast to Abernethy and Kale [1] which used Online Algorithms based on expert advice.

In the conducted experiments, we compared these Algorithms with each other, based on Agent's profitability i.e. Agent's cash, holdings and the reward it receives after following a certain strategy. A further comparison of these Algorithms has been made, based on the notion of regret. More precisely, the performance of the algorithm is compared to the way it would have performed if the best strategy was known and followed in advance.

In order to simulate the whole trading process, we created a custom Limit Order Book from scratch, in which the Agent buys and sells stocks, as well as an order generator which generates fake orders. We also analysed the data of Algorithmic trading dataset [6] in order to create a more realistic flow of orders.

Our results proved that in terms of the Regret, all the algorithms converge quickly to the performance of the best strategy 4.12. The mean regret of $\epsilon_n$greedy is slightly worse than the UCB and the EXP3. The same parameter, drops really quickly near to 5 and then it remains stable with some mild ups and downs. Overall, the UCB1 and the EXP3 perform better than $\epsilon_n$greedy. Now, as far as EXP3 and UCB1 are concerned, the EXP3 does not start with such extreme values, as UCB1 does, while during the remaining execution period they are both near 3, with UCB1 performing slightly better.

Even though all three algorithms along with the strategies have been profitable for the Agent, it would have been more beneficial if the experiments had been performed with real data, in an effort to investigate whether those algorithms would be suitable for realistic problems. In future work, it would be interesting to compare how the bandit Algorithms we studied will perform against online expert advice algorithms in terms of regret.

Bibliography

[1] J. Abernethy and S. Kale. Adaptive Market Making via Online Learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, volume 2, pages 2058–2066, 2013.

[2] I. Aldridge. *A Practical guide to Algorithmic Strategies and Trading Systems*. Wiley, 2010.

[3] P. Auer, N. Cesa-Bianchi, and Y. Freund amd R. E. Schapire. The Nonstochastic Multi-armed Bandit Problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 1603(47):235–256, 2002.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Campridge University press, 2005.

[6] Capital Markets Cooperative Research Centre. Kaggle algorithmic trading challenge. https://www.kaggle.com/c/AlgorithmicTradingChallenge, 11 2011.

[7] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. How to use expert advice. *Journal of the ACM*, pages 427–485, 5 1997.

[8] T. Chakraborty and M. Kearns. Market Making and Mean Reversion. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 307–314, 2011.

[9] M. D. Gould, M. A. Porter, S. Williams, M. Mcdonald, D. J. Fenn, and S. D. Howison. Limit Order Books. *Quantitative Finance*, 13(11):1709–1742, 12 2010.

[10] W. Huang, C.-A. Lehalle, and M. Rosenbaum. Simulating and Analyzing Order Book Data: The Queue-Reactive Model. *Journal of the American Statistical Association*, 110(509):107–122, 12 2013.

[11] D. Kane, A. Liu, and K. Nguyen. Analyzing an Electronic Limit Order Book. *The R Journal*, 3(1):64–68, 2011.

[12] R. M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? *INTERNATIONAL COMPUTER SCIENCE INSTITUTE*, 1603:235–256, 1992.

[13] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, pages 212–261, 2 1994.

[14] J. Loveless, S. Stoikov, and R. Waeber. Online algorithms in high-frequency trading-the challenges faced by competing hft algorithms. *ACM Queue*, 11(8):30–41, 2013.

[15] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2018.

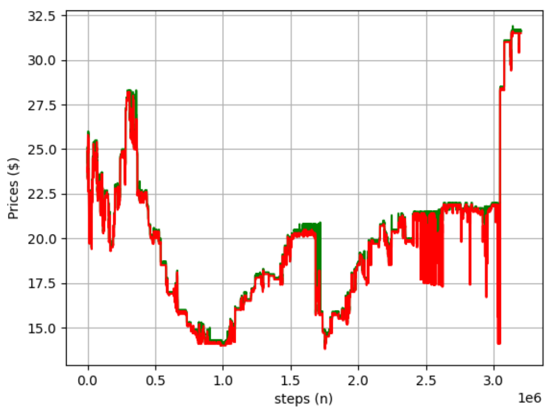
[16] R. Savani. High frequency trading: The faster, the better? *IEEE Intelligent Systems*, 27:70–73, 2012.

[17] T. Spooner, R. Savani, J. Fearnley, and A. Koukorinis. Market Making via Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 434–442, 2018.

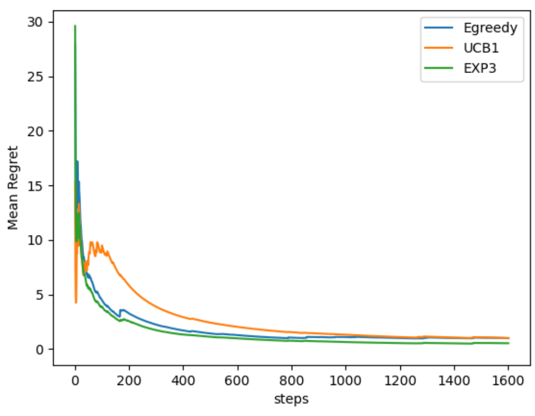
[18] R.S. Sutton and A.G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

Appendix A

Additional results

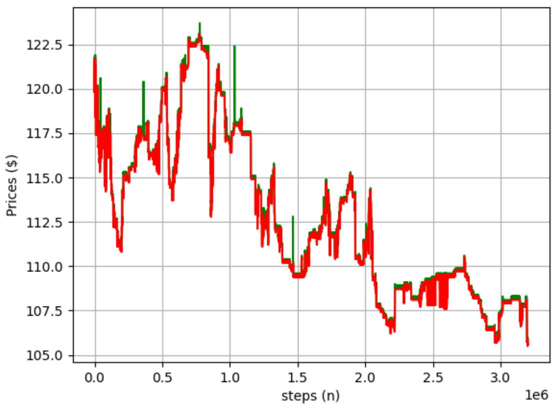


(a) Price

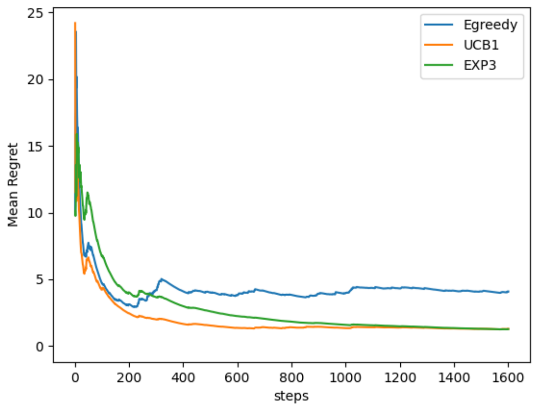


(b) Mean Regret

Figure A.1: 2nd Comparison of mean Regret for Egreedy, UCB1 and EXP3

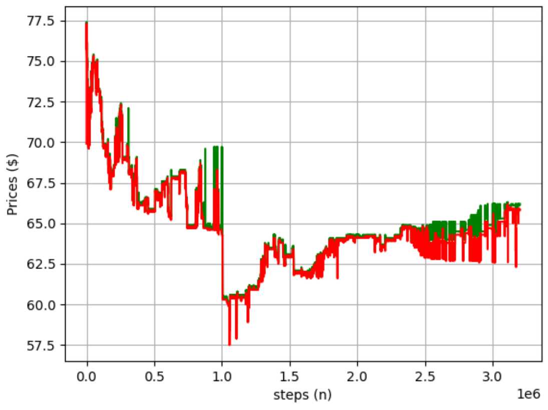


(a) Price

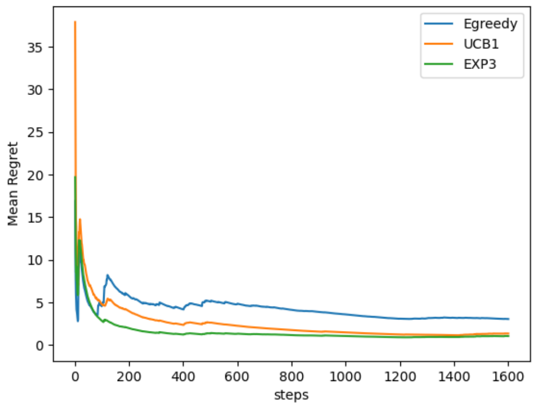


(b) Mean Regret

Figure A.2: 3rd Comparison of mean Regret for Egreedy, UCB1 and EXP3



(a) Price



(b) Mean Regret

Figure A.3: 4th Comparison of mean Regret for Egreedy, UCB1 and EXP3