Simulation of Time-Series Market with Trading Agents

Cao Khanh Nguyen[1]

BSc Computer Science

Dr. Paolo Barucca

Submission date: 19 May 2020

[1]**Disclaimer:** This report is submitted as part requirement for the MY DEGREE at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

Algorithmic trading has been extensively studied in the past and applied to real-time trading. Trading algorithms aim to create automated trading strategies and generate profits from the market. The majority of researches are focusing on optimizing the trading algorithms to improve the efficiency of strategies for more significant profit return; however, a problem raised with the trend of algorithmic trading usage is how theses trading strategies can disrupt the market on a large scale. Furthermore, with the growing popularity of artificial intelligence, we also want to explore how we can apply the study of algorithmic trading into developing higher intelligent algorithms.

This project aim is to simulate a time-series agent-based market model, which served to investigate the impacts of different combinations of traders has on driving the market price. This time-series market model is further extended to a simple environment to train higher intelligent trader that can stably make profits from the market environment. The goal of the project is to utilize algorithmic trading and study its impact and efficiency, at the same time research simple artificial intelligent technique on its application in trading. Trading agents algorithm and market mechanism design is the main factor in this project, which are essential in modelling an expected behaviour of a time-series stock price.

Contents

Chapter 1

Introduction

This section will describe the aims and goals of the project, followed by a personal approach on how to achieve them. Finally, there would be a brief outline of the project report.

1.1 Problem Outline

Agent-based simulation of a financial market has been useful in conducting market and stock price analysis. While there are undoubtedly many simulations which are either too abstract or straightforward to resemble the actual financial markets, they can be a solid fundamental and manageable foundation to study algorithmic trading and its impact on a market. The market simulation of this project is based on time-series modelling, which is useful in analyzing correlated price data given the current price context and agents interaction.

Market simulation and agent-based modelling have been an extensive study. However, a time-series simulation of the financial market remains relatively low as being too simple, yet it provides a fundamental in understanding price impact and algorithmic trading influences. This project seeks to tackle these problems to conduct meaningful study materials, findings and experiments for these subjects.

1.2 Project Aims and Goals

1.2.1 Project Goals

The first goal of the project is to create a simulation of a financial market that can act as a simple framework for further experiments on algorithmic trading impacts and artificial intelligence playground. The various components of the simulated market are outlined below:

- A time-series price moving model with concrete time steps

- Algorithmic trading agents that interact with the simulated price market environment on each time step
- Customize test case environment for various experiments on trading agents composition
- A simple playground environment that can be used to train reinforcement learning agents

Based on this simulated environment, another goal of the project is to analyze the impact trading agents has on driving the price, their wealth and other agents behaviour.

1.2.2 Project Aims

This project aims to gain a substantial learning curve in many subcategories of technology and finance industry, including but not limited to:

- Financial Computing
- Algorithmic Trading and Market Impacts
- Agent-based Environment and Modeling
- Practical application of Reinforcement Learning

Academia research and exposure are also the core aims of this project. With a decent amount of academic discussions with the research group, the project is primarily a research project rather than an industrial one. The chosen subfield, Financial Computing, also needs a measurable amount of research and development initiatives to grasps the underlying principle.

1.3 Project Approach and Timeline

The main development and experiment approach we adopted was an iterative feedback loop following a thorough discussion of the progress and room for improvement. The project timeline has several phases, which are described below.

1.3.1 Planning and Researching

As the subject of financial computing was a new subject for vastly new at the beginning, during the first months of the project duration, we constantly had discussions to seek for materials and explanation to concepts and scope of the project. During this phase, we had meetings with the research team, formally describe our initial plannings and received feedback on the expected behaviour of the simulated market.

1.3.2 Market Simulation Implementation

The main phase of the project is to implement the simulation. Actual development was done based on prior research. The pipeline was mostly research and development, in which we iteratively programmed the algorithm trading and market functions, made a few testing experiments, presented them, received feedback, then fixed the bugs and improved the systems. We held weekly meetings and discussions to maximise the feedback loop. Robust tests are also implemented on this stage to determine that the simulation would behave as expected.

Occasionally there would be a research group meeting where we present our ideas and seek help from others on topics which can be an add-on to our project. This project received decent informative help from research peers in the subject of reinforcement learning.

1.3.3 Experiments and Reinforcement Learning Environment

Experimental research is the last phase of the project, excluding writing the report. During this phase, we maintained the feedback loop approach. We discussed the experiments with others to find the insights that would be helpful, then suggested different approaches for further research.

We also implemented a custom environment to train a reinforcement learning trader, following the suggestion from a peer in the research group. For most of the programming tasks, we followed the agile approach.

1.4 Overview of this report

This report document the work on the market simulation, showing the design and algorithmic decisions on implementing the system. It also presents some interesting findings we discovered during the experimental phase with trading agents and market price impacts. The following is the breakdown of each chapter content:

- Chapter 2: The context and background overview of algorithmic trading and time-series analysis regarding these subjects' current study and research.
- Chapter 3: Detailed outline of the project requirement and the desired outcome.
- Chapter 4: Design and implementation of the project. This chapter will go into details the market structure and functions, algorithm of the implemented trading agents, and the design of the customised training environment.
- Chapter 5: Testing and experiments. This chapter will provide some robust tests on the system. It will also include the experiments with the simulated market environment and

conclude its economic significance.

- Chapter 6: Conclusion, evaluation and acknowledgement. This final chapter is about self-criticism of the project and comments on room for improvements. This section also includes a subsection to acknowledge the challenges and support we had during the project.

Chapter 2

Background

This chapter will provide a better insight into the current state of study and research of subject topics that are relevant to this project. The topics are consists of these sub-categories: a time-series analysis in the stock market, algorithmic trading and reinforcement learning in trading.

2.1 Algorithmic Trading

This section will summarise the context and technique used to simulate market in this project. The simulation of the market is a time-series model filled with non-zero intelligent agents. These agents are automated trading agents following algorithmic trading.

2.1.1 Trading Algorithm

Algorithmic trading is the study of applying algorithms to generate automated-trading strategies. It largely depends on complicated real-time systems and programs that enter the trading markets and place a trade without human intervention. The automated trades are often predefined by a set of instructions that are based on various factors, including price, timing and any quantitative implications. In theory, algorithm trading commits activities and generate profits at a frequency that is impossible for a human trader. It also renders the market more liquid and trading more systematic by limiting human emotions on trading activities [1]. For a more extensive overview of Algorithmic Trading, see [2].

Often than not, studies on algorithmic trading aim to devise algorithm that can maximize profits from automated trading. The main challenges were to take account of random external factors and generalize them in devising the algorithm. This process requires a great knowledge in data analytic and mathematical modeling. Besides, a foundation in computer programming is also needed, as it is also challenging to transform idealized trading strategies into automated computerized process [1]. The following is a simple example of trading algorithm:

- Buy n shares of stocks when the price moving average in the last x days goes above that of the last y days, with $x > y$.

This strategies can be formulated by the following mathematical notation,

$$X_{t+1} = \begin{cases} X_t + n, & \text{if } \frac{\sum_{n=t-x}^{t} S_n}{x} \geq \frac{\sum_{n=t-y}^{t} S_n}{y} \\ X_t, & \text{otherwise} \end{cases}$$

and can then be transformed to this computer programming instructions

Algorithm 1 Simple trading algorithm

Require: array of known prices $[S_0, ..., S_t]$ at time t
  X, n
  if $\frac{S_{t-x}+S_{t-x+1}+...+S_t}{x} > \frac{S_{t-y}+S_{t-y+1}+...+S_t}{y}$ **then**
    $X \leftarrow X + n$
  end if

As the market becomes increasingly complicated and the popularity of artificial intelligence in data analysis, several automated trading algorithms have been devised and applied into real-life trading. These automated trading would provide a confidential basis that trading is accurate, timely with low risk of manual errors. These strategies are also beneficial for high-frequency trading, when orders are placed in large rational numbers with the rapid speed of transactions, in order to maximise small interval profits. Several popular strategies are Trend Following, Trading Range (Mean Reversion), Volume-weighted Average Price (VWAP), and Percentage of Volume (POV) [1].

2.1.2 Impact of Automated Trading on Market

A goal of this project is to investigate the potential impacts of algorithmic trading on the market price, volatility and return. Researches on this matter showed that securities with higher algorithmic trading strategies often have lower liquidity costs, order imbalance, and order volatility [3]. Besides, contrary to common belief, algorithmic trading often did not lead to an extreme anomaly in price changes, often referred to as *flash crash*. However, these assertions are mostly dependents on the algorithm rule and the way they interact with other automated trading strategies. For extensive research on algorithmic trading impact, see [3].

2.2 Artificial Stock Market Simulation

2.2.1 Time-series Analysis

Time-series is a set of observations or data points collected at a predefined period and a specific time. Time-series presentation often lends a unique character to statistical analysis of the model. In real life, a time-series analysis has several aims. However, the most important is to depict the behaviour of observations, model the characteristics of the series, and even control or forecast its behaviour [4]. Studies often depicted the observed data points as an array of discrete observation with equal time intervals like day, week or year [5].

Time-series analysis often adds additional perspectives to pure observations of concrete data points. Researchers commonly examined these separate components of the array of statistic data:

1. The insight trends or movement than can be analysed and observed through the model.
2. Oscillation patterns within the trends. These patterns do not have to be unanimous. However, often if they happen, they are in the form of repeated and predictable patterns that resemble each other, and often related to some internal and external factors.
3. Seasonal elements, which are the movement that correlates to a fixed period that would appear periodically.
4. Anomalous data point - a downward or upward lead in the trend that is either random or resulting from an unexpected internal or external factor.

Paul Whiteley's paper provided a detailed examination of time series analysis and its use cases [4].

Application of time-series analysis in analysing the stock market has been a common approach both in academia research and practical financial analysis. One of the widest usages of time-series analysis is to forecast the stock market prices and future patterns in order to formulate trading strategies that can generate profits. As the stock market volatility is gradually becoming erratic by external impacts, such as economic policy, interest rates, inflation and deflation, complex time-series modelling can also be combined with a modern approach to improve forecasting accuracy. For a further study on a sample hybrid approach of neural network and ARIMA time-series models, see [6].

The use of time-series modelling in this project aims to observe and understand the trend of stock market price and detect any anomalous impact that algorithmic trading can have on market volatility.

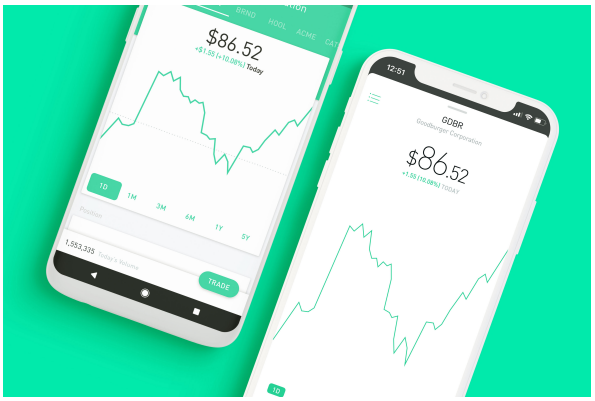


Figure 2.1: Trading apps display stock prices as time-series models. *Image: Robinhood*

2.2.2 Discrete time-series market

It is best to use agent-based modelling when the interactions between agents are discrete and discontinuous [7]. Historically, most of the attempts for agent-based modelling of the financial market are discrete-time in nature. A high-level overview of this nature can be best described as a turn-based game, of which during each turn agents will take their turn to place order, buying or selling their securities. These trading actions can be heterogeneous and asynchronous, meaning their tradings can be similar and happen at the same turn. The following simple rules can formulate the discrete-time-series simulation can:

- There will be continuous-time step $[0, 1, ..., t]$
- At each time step t, the agents will place their tradings to the market
- At the end of the time step t, the market price will be updated based on agents activities, which will be the price at time step $t + 1$
- The agents' corresponding portfolio value at time step t is determined at the market price t, and their position at t

Arguments are saying that this discrete-time representation is failing to depict the continuous trading sessions of a real market. These arguments lead to the need to extends research on discrete-time models to continuous-time and asynchronous ones. Boer-Sorban demonstrated a prominent study on this topic. He later proposed ABSTRACTE, *Agent-Based Simulation Trading Roles in an Asynchronous Continuous Trading Environment.* Further study on continuous-time modelling can be found in [8] and [9].

However, for the simplicity of the project, only discrete-time modelling will be implemented. This simulation will be filled with non-zero intelligent traders and then extended to be a training ground for a higher intelligent trader that applied reinforcement learning techniques.

2.2.3 Agent-based Market Simulation

Agent-based modelling is a powerful modelling technique that has applications span extensively in many real-world business problems. In agent-based modelling, a system is modelled as a collection of autonomous decision-making entities called agents [7]. These agents are often predefined with a set of rules in interacting with others and the environment. Repetitive interactions of agents are a feature of the modelling, and it often resembles the real-life interaction between individual human in a given environment. Agents can improve their intelligence, provided that the technical development environment provides sufficient standards for training the agents. Thus, agent-based modelling is recently the common practice in researching artificial intelligence of a multi-agent environment that resembles the complex human interacting decision system within societies. An exemplary interesting research on application of agent-based simulation can be *hide-and-seek* game from *OpenAI* in researching reinforcement learning [10].

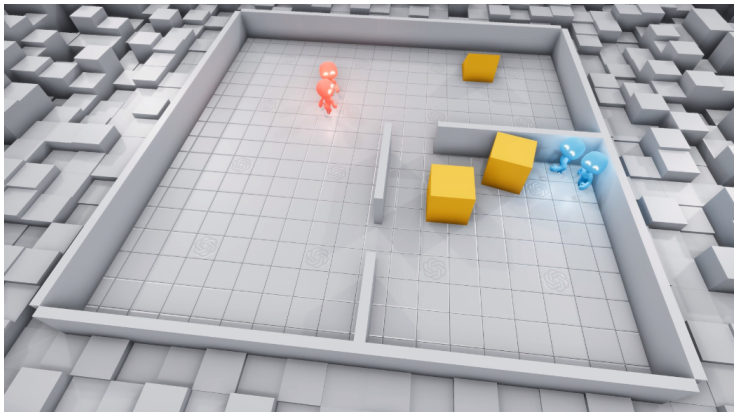


Figure 2.2: OpenAI *hide-and-seek* environment with red chasing agents and blue hiding agents

The stock market, software agents and strategic trading simulation have been one of the innovative areas of applications for agent-based modelling. One great benefit of this technique is an accurate assessment of the impact of individual models on the system as a whole. Besides, this method also resembles the basic nature of the stock market, as the status of a market at each discrete time is largely the result of complex interactions and actions between individuals within the market, namely investors. As a result, there has been an upsurge interest in agent-based models of the market in recent years [7].

2.2.4 Order Execution

The interaction between market and agents (i.e. How price dynamics are affected by orders placed by agents) is modelled following the trading model proposed by Almgren-Chriss [11].

The paper from Almgren-Chriss focuses on defining a framework that maximises the utility of trading revenue. They introduced the concept of trading strategy and a dynamic price model that served to develop this framework, and this trading strategy for the execution of placing orders is perfectly suitable for this project: it contains market impacts and orders executed in a time-series manner. *Section 4.2* detail more information on the modelling of the market.

2.3 Reinforcement Learning Agent

An initial goal of this project is to extend the market environment into a playground utilised to train reinforcement learning, to study the application of reinforcement learning within the trading market. While this goal is not yet fully achieved with a high standard of functionality and complexity, we partially developed and fulfilled the fundamental requirement of extending the simulation to a simple custom training gym. This section will summarise the current context of reinforcement learning application in the trading market and the state of the custom market gym.

2.3.1 Reinforcement Learning in Trading

Reinforcement Learning is a branch of Machine Learning that concerns with an agent learning to act within a given environment in order to maximise its reward. The reward it received often depends on each action it took, and a lot of time also dependent on the environment context it is in. The simple concept of reinforcement learning is iterative action by *trials-and-errors*, which enable an agent to learn from its previous actions and experiences. Different from other subsets of Machine Learning, Reinforcement Learning does not have a predefined output, meaning that it does not solve classification problems and does not know whether its output is correct or incorrect. It gears its artificial decision-making system towards maximising the cumulative rewards it achieves.

The stock market trading algorithm can perfectly apply this principle of Reinforcement Learning. While an individual agent within a stock market does not know whether its place of trading is correct or not, its simple goal is to maximise profit at the end of the day. Therefore, there has been an increase in interest of Reinforcement Learning application within the financial market. In particular, firms have widely applied reinforcement learning to develop investment and trading strategies [12]. Examples of research on FX Trading and High Frequency can be found at [12] and [13].

This subset of Machine Learning is particularly interesting, as it the one that most resembles the humankind process of learning and decision making – through **trials**, **errors** and

experiences. Therefore, as part of research and development-oriented project, there is a motivation to extend the simulation to be a training playground for Reinforcement Learning, serving to understand how algorithmic trading can have an impact on potential higher intelligent automated traders.

2.3.2 Training Playground

In order to train any Machine Learning algorithm, we have to provide it with substantial data as input and define an expected form of output from its processing. Reinforcement Learning is no exception. The input to Reinforcement Learning algorithm is any observations from its surrounding environment, the process is its actions, and the expected output is the rewards it received from doing the actions.

Therefore, in order to train any Reinforcement Learning algorithm, an informative environment is required, in which agents can collect factors that might affect its decision-making process as 'observations'. A great example is the set of open-source Reinforcement Learning environments from OpenAI, often referred to OpenAI gyms, which can be found at `https://gym.openai.com/`. These gym environments are the training ground for Reinforcement Learning algorithms, providing easy-to-implement calls to gather essential information, including observations, actions and rewards.

However, these predefined gym environments do not contain one that is related to market trading. Therefore, the goal is to make a simple environment that is similar to these gyms in functional calls to information by extending the market simulation.

2.4 Tools and Frameworks

2.4.1 Python

Python is growing in popularity as it is easy for us, powerful and versatile. A stand-out characteristic of Python is that it is readable and straightforward to program, allows us to work on several designing and functional aspects without focusing too much on the syntax. As for data science, this advantage of Python also makes it easy to captures ideas, and concisely display the data and discovery comprehensively. At the same time, it is reasonably fast and powerful enough to handle heavy computational analytic. Besides, Python comes with a great interactive shell, easing the process of debugging and researching.

2.4.2 Jupyter Notebook

As the application of Python in data analytics raise, a demand for a versatile interactive Python shell also grows. Jupyter Notebook was developed in 2014 out of IPython, and quickly becomes a default environment for research. The reasons we chose to use Jupyter Notebook are its interactivity, ease of usage with data visualisation, familiarity with the supervisor, and it is almost a perfect environment for experiments and presentations of work. Jupyter Notebook also supports Markdown cells so that we can properly explain the thought process within the code context.

2.4.3 numpy and matplotlib

As stated above, the simulation in this project is developed from scratch, only referencing to the mathematical models and algorithmic trading formulas. Therefore, the only frameworks we used in this project are standard and fundamental package for scientific computing and graphing, namely NumPy and matplotlib. While many open packages support market simulation, both open source and offered by peers, we chose to create one from scratch to fully understand the models that we came across during our research.

2.4.4 Alpha Vantage

For the need of real market data and stock price, we need a Stock Market API to search original stock price. Alpha Vantage is the ideal candidate, as it is free and is by many researchers and business professionals. There is also an open-source Python wrapper package for Alpha Vantage, which made it easily integrated with the system. The package can be found at `https://github.com/RomelTorres/alpha_vantage`.

Chapter 3

Requirements and Analysis

This chapter is an extension from the Introduction chapter, delve into the formulation of the problem statements, its requirements, use cases, and expected deliverable.

3.1 Problem Statements

Studies on trading algorithms have attained successes on the market in recent years, as applications of automated algorithmic trading in the financial markets have attracted a great deal of attention. With the rise of machine learning techniques in forecasting and formulating trading strategies, these computerized activities are becoming an integral part of the financial market.

While there are many pieces of research on optimizing trading strategies, there were not many studies on the collaborative impact these algorithms has on driving stock prices and its wealth. This project aims to study the collaborative nature of automated trading agents by placing them in a simple simulated version of the stock market and observes their impacts. Another problem this project aims to tackle is to research on how reinforcement learning algorithm would act on the simplified market, observe its activities and potential impacts other trading agents can have on a higher intelligent algorithm.

3.2 Requirements

3.2.1 Overall Requirements

We formalized the following requirements below based on our academic discussions and the research-oriented nature of the project.

The high level requirements of the project, as agreed upon discussion, are:

- Simulate a simple time-series stock market
- Generate trading agents based on popular trading algorithms
- Make various experimental phases, either for testing robustness of trading agents and market or to observe and analyze the impact of agents
- Extend the market to include higher intelligent traders using artificial intelligence techniques, namely Reinforcement Learning.
- Observe how higher intelligent agent act in a market filled with other zero-intelligent traders
- Readable code, so as to be extendable in the future
- Produce debatable output to be discussed on research group meeting

Besides, several other requirements on implementation are also specified on the above components. These requirements are defined on my own and presented to my supervisor, so as to easily follow the feedback loop approach.

3.2.2 Specified Requirements for Market Components

The table below will list the sub-components of the project and its requirements according to the MoSCoW approach.
(M - Must have, S - Should have, C - Could have)

Requirements for expected Market Simulation		
ID	Requirement	MoSCoW
M01	The market is time-series model with discrete time data points	M
M02	The market is built with the model from Almgren-Chriss trading model	M
M03	Market price is updated sequentially	M
M04	Number of market agents can be customized	M
M04	Number and type of market agents can be customized	M
M05	Test environment included, customizable and extensible	M
M06	Market price is impacted by agents' orders	M
M07	Customizable underlying price trend	C
M08	Use real market price data	C

Requirements for Trading Agents		
ID	Requirement	MoSCoW
A01	Trading Agents are based on sample popular trading algorithm	M
A02	Agents place orders based on current market price	M
A03	Agents follow a standardized interface	M
A04	Agents keep track of its position and can display its stock portfolio in a time-series manner	M
A04	Agents keep track of its buying and selling activities	S
A05	Agents keep track of the number of securities it traded	S
A06	The number of securities agents trade its depends on both its current position and market price	C
A07	A diversified combination of agents	C

Requirements for Reinforcement Learning Playground		
ID	Requirement	MoSCoW
R01	An extension of the market simulation	M
R02	Provide information necessary for RL training, which are observations, actions and rewards	M
R03	Simple Q-Learning Agent	C
R04	Higher Level Deep Reinforcement Learning Agent	C

3.3 Use cases

Since the project is research-oriented, it does not have an obvious guideline on use cases. However, There are possible extensions for the usage of the project both in academic and research. Examples of further usage include:

- A simple market for algorithmic trading modules
- A simple playground for reinforcement learning approach in financial analysis
- A model for time-series financial analysis

3.4 Analysis

After defining the above requirements, the analysis approach to the project can be better visualized and formulated.

As mentioned above, the expected market simulation would be following the trading model defined by Almgren-Chriss paper [11]. Therefore, the first expected modelling of the market should be a time-series model with continuous data generation. The requirements on the most significant components are agents the following sample popular trading algorithms. The chosen algorithms are Trend Following and Trading Range. Besides, Random Trading also served to generate noise. On the higher intelligent trader part, in order to implement a Reinforcement Learning Trader, a proper trading environment must be created. Therefore, the sequential data in the market must be updated step-by-step in order to feed this step-wise information to the RL Trader.

The requirements itself did not come up immediately after the first few discussions, as the nature of the project is research-oriented. However, we had a gradual formulation of this requirements list during discussions and feedbacks with the supervisor and research group. It is important to note that many of these requirements were formed during the implementation phase to place restriction and standardize the simulation, including every requirement from the reinforcement learning list.

Chapter 4

Design and Implementation

This chapter will discuss the in-depth implementation of the project and the motivation behind the design. As many of these other sections have hinted at, there are three main functional sub-components in the overall codebase for the market simulation: Market, Agents and Reinforcement Learning Environment.

4.1 Overall design and approach

The initial thought of the project was to design a simulation of the market that include different agents within its functions. Therefore, after the initial discussions, we made it crystal clear that the required components are the Agents and Market. Furthermore, each of these sub-components must be interacting with each other in a time-series manner. This principle means that there would be time intervals, of which at each time points, there is an interaction between agents and market price that will affect the subsequent price and agents' portfolio. Thus when a price is updated, the agents will base its decision for lacing orders in the next time step on that updated price.

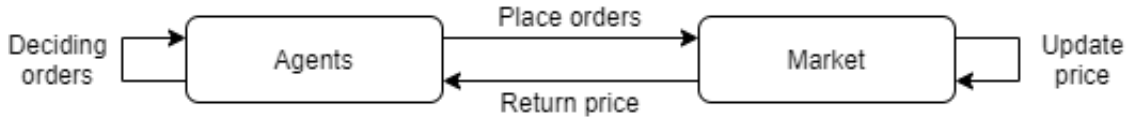


Figure 4.1: The initial overall thoughts of the market mechanism

Besides the Agents and Market components, a higher-intelligence agent is a desired requirement of the project, in order to learn about reinforcement application in financial practice and see how we can apply those algorithms in a simulated environment. The reinforcement learning agent can follow the practice and interface of a standard agent but

would be extended with complex variables that represent state, reward and action. Therefore, it would not necessarily be included in the Agents part in the figure above but instead acts on its own and deem the whole figure as the learning environment.

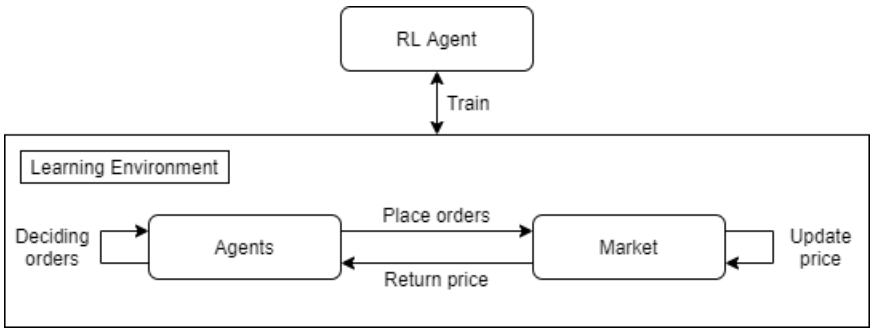


Figure 4.2: Market and Agents act as Training ground for Reinforcement Learning

These are only the initial thoughts on the overall thoughts and system designs for the simulation. These designs might look simple for such a complex system. It was indeed agreed and advised by other research members that the simulation should be fairly simple so that it is doable given the time frame of the project.

However, as an important part of this project is to create experimental simulations and observe its impact, the project included many experiments within the implementation phase. With constant modifications to the market and agents functions, we realized the needs to re-structure the design and formalize the testing process, in order to make simulations easier and more systematic, instead of manually create new a market every time a new test is needed.

Figure 4.3 is the resulting final design of the project. Essentially, we created a new Test-Case Class. This Class serves as the Test Generating Engine and as the foundation for Robust Testings as well as Simulation Testings. This approach allowed us to make mass amendments to the test cases, given that there is an essential change in the Market and Agents components. The other components of this design are quite self-explained: The Real Data is for feeding an original market price into the simulation, Agents and Market formulates the Reinforcement Learning Environment as in *Figure 4.2*, which in turn served to train the Reinforcement Learning Agent.

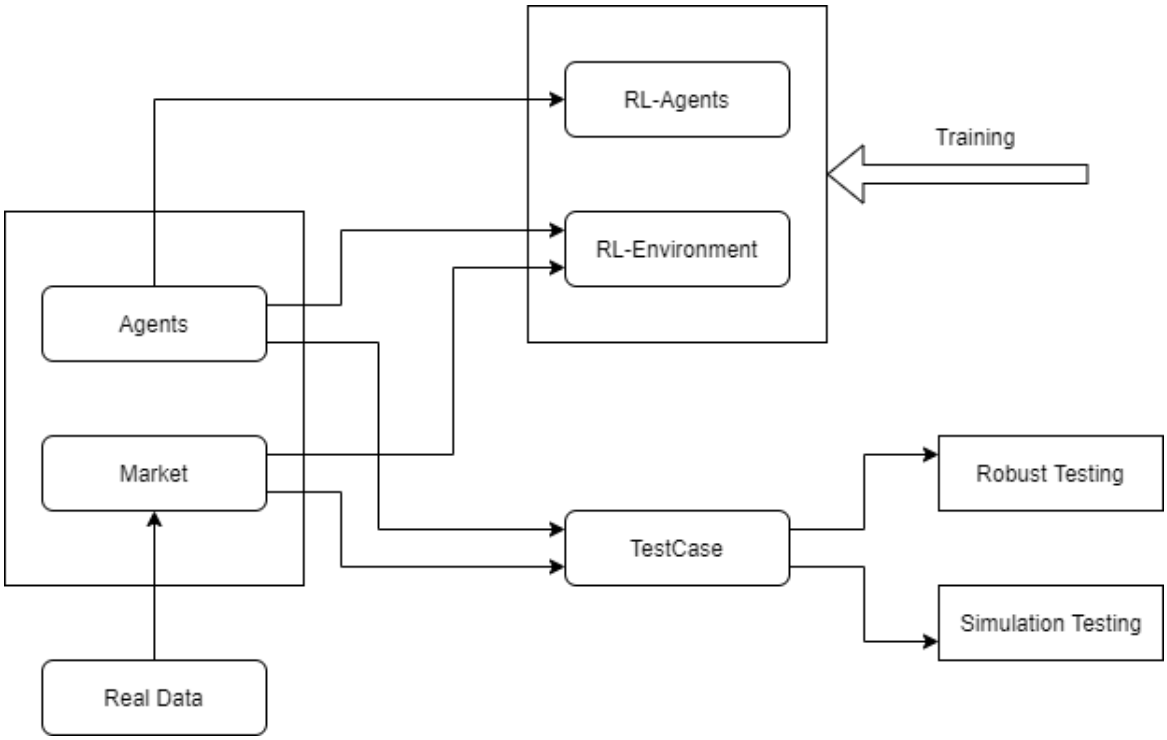


Figure 4.3: Final Design

4.2 Market Trading Model

There are currently many exemplary modes to generate market simulation. One of the very prominent models is limit book order, which depicts limiting the order to buying or selling securities at a specific price or better [14]. We also extensively studied the simulation of this market model and generally agreed that creating such a simulation will likely take too much time that might be left little to no time left to invest on market impacts and reinforcement learning.

As a result, a different approach to simulating a market is to use a time-series model. The simulated market in this project highly reflects the trading model from Almgren-Chriss paper [11]. The simulation in the project used this paper as the foundation to build the time-series model and agents' impacts.

4.2.1 Almgren-Chriss market

The underlying principle of any market is the price. In the "Optimal Execution of Portfolio Transactions" paper, Almgren-Chriss has derived a concrete market trading function that follows a time-series manner.

Let assume that the price of securities is S_t, where t is referring to the current time step. In an ideal market environment, this price will be affected by two factors: exogenous factors and endogenous factors. Exogenous factors are commonly volatility and drift, e.g. the result of market forces that occurred randomly and independently of the trading activities. Endogenous factors are the opposite: the direct impact that trading activities had on securities price.

The price dynamics can be derived using the formula

$$S_{t+1} = S_t + \sigma\tau^{\frac{1}{2}}\xi_t - \tau g(\frac{n_t}{\tau})$$

For simplicity of further explanation, let formulate $ex(t)$ and $en(t)$ to be exogenous and endogenous factors, respectively. Thus, we have

$$S_{t+1} = S_t + ex() + en()$$

where

$$ex(t) = \sigma\tau^{\frac{1}{2}}\xi_t$$
$$en(t) = -\tau g(\frac{n_t}{\tau})$$

4.2.2 Volatility

The first components of the above function can be majorly represented by volatility. This volatility is not the standard deviation of price, that is often referred as the measurement of how price changes over a period of time, which can be see in [15]. Instead, in Almgren-Chriss paper, this volatility factor is described as "independent random variables each with zero mean and unit variance". Essentially, this volatility factor is a Gaussian noise with $\mu = 0$ and $\sigma^2 = 1$. More information on Gaussian noise can be found at [16].

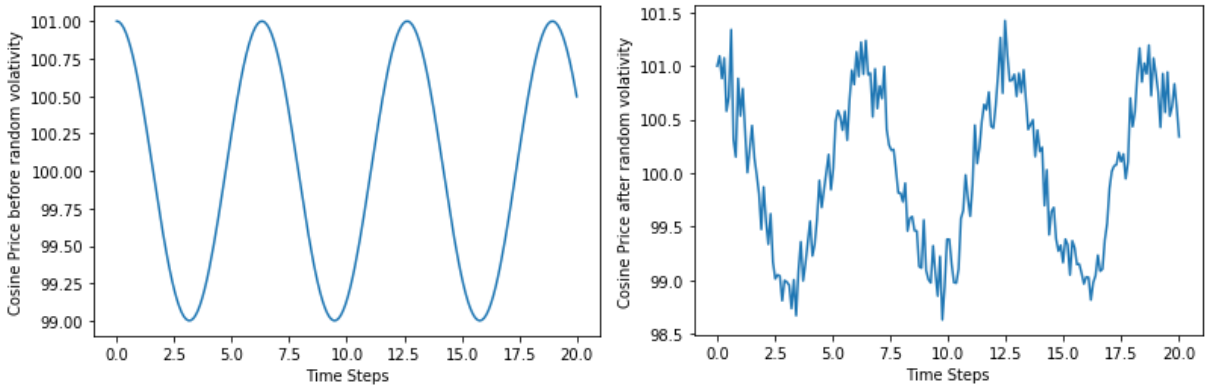
Therefore, the the affect of exogenous factors on price is

$$ex() = \sigma\tau^{\frac{1}{2}}N(0,1)$$

and since the time intervals are simply defined to be unit variance, $\tau = 1$, the resulting volatility is

$$ex() = \sigma N(0,1)$$

with σ being the volatility of the assets.



(a) Cosine without noise

(b) Cosine with noise

Figure 4.4: Affect of volatility on a cosine model market

4.2.3 Market Impact

In order to fully understand the market impact, we need to introduce a new variable: market orders. Let assume that in time step t, a trader decided to place an order, either buying or selling, that made it current holdings of securities change from $X_{t-1} \rightarrow X_t$, then the orders it placed on the market at t is $n_t = X_{t-1} - X_t$. This order, combined with orders from other traders, will be the main factor that affects the market price.

There are two sub-components of the market impact: Permanent Impact and Temporary Impact. To put in simple terms, Temporary Impact refers to the subtle price changes in each time steps, but in overall will not be resulting in a permanent impact on price. Assuming that the time interval is unit variance, $\tau = 1$, we can derive the following formula for temporary impact

$$h(t) = \epsilon sgn(n_t) + \eta n_t$$

This function considers the approach of computing impact functions using linear correlation with the rate of trading. Similarly, Permanent Impact, which makes lasting changes to the continuous price, is computed using

$$g(t) = \gamma n_t$$

However, the impact of endogenous factors on price, taking into consideration n_t, which is the total volume of orders agents place onto the market, only considers permanent impact. Therefore, the formula for the endogenous factor is

$$en(t) = g(t) = \gamma n_t$$

The temporary impact will only be considered at each timestep, and would not permanently result in the time-series model in the simulation. Thus at each timestep t, the temporary price is

$$\tilde{S}_t = S_t + h(t) = S_t + \epsilon sgn(n_t) + \eta n_t$$

Note that, within the simulation itself, the chosen value for ϵ, η, and γ is completely ad-hoc, but still not too significant to interfere too much with the price dynamics itself.

4.2.4 Slight Modifications

The simulation in this project is not entirely similar to the trading model of Almgren-Chriss paper. Instead, as the trading model Almgren-Chriss proposed is to reflect the real market with the confounding complexity of traders interaction and strategies, thus resulting in an underlying function that might explain the price dynamics observed in the real market.

However, the purpose of this project is not to scrutinize the drive in a real market, but instead, simulate one and observe artificial traders interaction. Therefore, if a similar manner of generating time-series data where subsequent data in only dependent on previous price points data, the result would always be a monotonous market if the combination of agents is always similar. Since the project also used a set of price data with predefined trend or period, the formula should be modified to take into accounts the underlying original trend and period. Thus, the new formula is

$$S_t = S0_t + \sigma N(0,1) - \gamma n_t$$

in which $S0_t$ is the original price. This formula is fundamentally adequate since it also takes into account the impact on the market based on previous price data points if there is any correlation in the original price.

4.3 Agents

Agents are at the heart of the market. In a real-life marketplace, agents are the investors who make decisions based on the information they have, being it inside information or observable information, and make profits from it. The rise of algorithmic trading and artificial intelligence interrupts these market standard by introducing automated trading, which are machine-investors making decisions based on a predefined set of instructions.

Any changes in the stock market price are partially affected by agents activities. It is almost impossible to simulate a realistic market with agents that accurately reflect humanity's activities. This projects introduced three different types of agents: Random Traders, Trend

Following Agents and Mean Reversion Agents.

4.3.1 Agents Function

Each agent in the simulation will have variables representing its portfolios. It will participate in the market system, where it can have observations of the price, and place orders accordingly. Each agent type would have its trading algorithm, which takes the price of the securities as an input, processes the input and its current portfolio with its algorithm and devises an appropriate output, which are the orders it is going to place on the market. The agents also need to keep track of their stock and cash holdings, their previous buying and selling decisions, and orders transactions.

The agents are implemented in an object-oriented manner, in order to formalize the generation of agents into the market. Besides, they also follow an inheritance hierarchy, in order to formulate organization and future extension, if there are further needs in adding new types of agents.

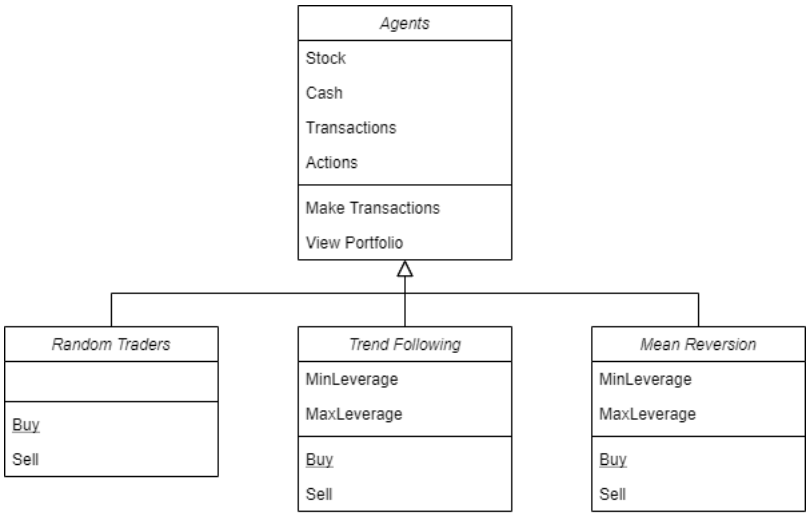


Figure 4.5: Class Diagram of Agents

4.3.2 Random Agents

As the name suggests, these Random Agents will place a random order every time it participates in the market. A formal definition of Random Agents is that this is the type of agent with no specific information. Therefore, at each time step, it will randomly buy or sell, with a random amount of securities, no matter the current market price of the securities.

While Random Agents might seem to be unrealistic in their behaviour, they are an important factor of any simulated environment. Random Agents' responsibility is to create a realistic sense of the market when any particular trader does not possess information about other traders' strategies. They also generate a decent amount of noise to reflect irregular trading patterns found in real markets.

The pseudocode for Random Agents is

Algorithm 2 Random Agents

Require: None
  X, n,
  $r \leftarrow random(0, 1)$
  if $r = 0$ **then**
    $n \leftarrow random(-2, -1)$
  else
    $n \leftarrow random(1, 2)$
  end if
  $X \leftarrow X + n$

4.3.3 Trend Following Agents

Trend Following algorithm is one of the most famous trading algorithms in automated algorithmic trading. The intuition behind Trend Following might resemble many of our conceptions and strategies in trading: We would likely to buy a stock if we see an increase in the value of the stock in the last period; and if we started to see the value decrease, we are likely to be afraid that it will continue the trend and our investment in the stock will be losing in profit; thus we would sell it.

The actual trading algorithm is the systematic approach of this idea. We formalize this idea into automated reasoning and decision making. The concrete algorithm for Trend Following Traders is

1. Buy securities when the last price goes above the moving average price (of 3 intervals)
2. Sell securities when the last price goes below the moving average price (of 3 intervals)

Based on this principle, the algorithm of trend Following is of following

Algorithm 3 trend Following

Require: History of Prices
  X, n
  $mean \leftarrow \frac{S_{t-4}+S_{t-3}+S_{t-2}}{3}$
  if $S_{t-1} > mean$ **then**
    $n \leftarrow Buy(t)$
  else
    $n \leftarrow Sell(t)$
  end if
  $X \leftarrow X + n$

4.3.4 Mean Reversion Agents

The Mean Reversion in finance suggests that asset prices and historical returns eventually revert to their long-term mean or average levels [17]. In other terms, people following this strategy would have a belief that every change which diverts away from the initial price will lead to the price returning to its original point in the future. That is, if the price has an upward trend, it is likely that soon this trend will diminish and a downward trend will appear, drive the price down. Therefore, those people will devise a strategy that is almost a direct opposite of Trend Following

1. Sell securities when the last price goes above the trend
2. Buy securities when the last price goes below the trend

As a result, the algorithm for a Mean Reversion is a direct opposite of the algorithm for Trend Following

Algorithm 4 Mean Reversion

Require: History of Prices
  X, n
  $mean \leftarrow \frac{S_{t-4}+S_{t-3}+S_{t-2}}{3}$
  if $S_{t-1} < mean$ **then**
    $n \leftarrow Buy(t)$
  else
    $n \leftarrow Sell(t)$
  end if
  $X \leftarrow X + n$

Note that, this algorithm takes many assumptions into account. One of the most obvious is that Mean Reversion Agents often look at the market as a whole picture, so would often have an interval of price moving average much greater than that of Trend Following Agents. However, a similar price moving average interval was taken into account further to investigate the interactions between two seemingly opposite agents types.

4.3.5 Leverage

So how much securities will Trend Following Agents and Mean Reversion Agents will buy/sell as soon as they decided the action? Using leverage was an idea brought up during the discussion on this topic. In real life, many investors and companies used leverage. The concept of leverage in this project is somewhat similar to Leverage in Forex Trading, which requires a trader to have at least $x\%$ value of trade available as cash [18]. In this simulation, the leverage α will denote the value of securities over cash. α will be set as the target point for selling and buying.

Each agent will have its own minimum and maximum leverage α_{min} and α_{max}. When the agent decide to buy, with its current securities X_t and cash C_t, leverage would be α_{max}, so that it can maximize the portfolio with value from securities. Therefore, the target securities value would be

$$TargetValue = X_{target} \times S_t = \frac{\alpha_{max}}{\alpha_{max} + 1} \times (X_t S_t + C_t)$$

From this equation, we can derive X_{target} to find the needed orders amount. Vice versa, when it decides to sell, the target leverage would be α_{min}.

However, a further experiment has shown that, if the amount of selling and buying depends on an agents' current portfolio value, it will always widen the gap between orders sold and bought, especially for Mean Reversion, whose activities is frequent. If there is a stack of a large number of Mean Reversion Agents, this gap is likely to skew the data. Therefore, we made the design decision to fix the X_t and C_t for the above equation, and only depends the amount of the order on the current price.

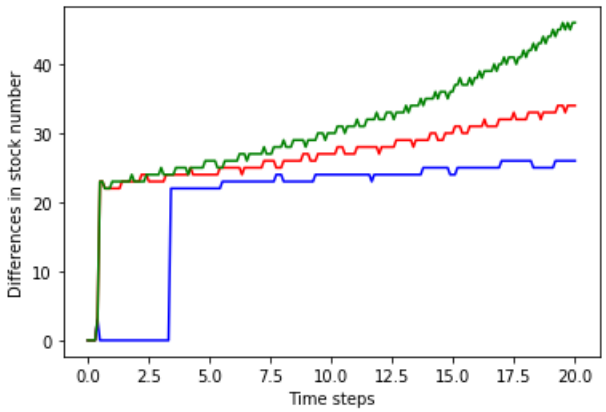


Figure 4.6: Securities gap between selling and buying episodes of a single (blue), 2 (red), and 3 (green) Mean Reversion Agents. This gap is expending exponentially.

4.4 Reinforcement Learning Formulation

4.4.1 Reinforcement Learning Environment

So far, this market simulation had create a foundation for the interaction between market price and agents. It is in fact can be served as an artificial trading environment to run many simulations on algorithmic trading.

A proper Reinforcement Learning (RL) Formulation needs to take into account other key elements, which are **State**, **Reward**, and **Action**.

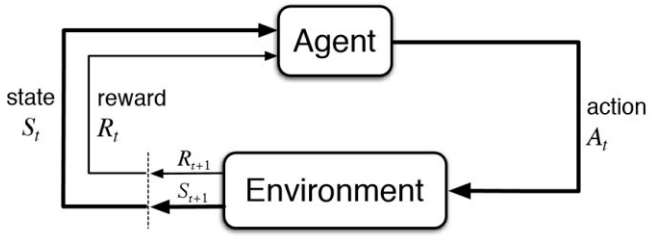


Figure 4.7: Reinforcement Learning Model

To simplify this approach in the project, a very simple approach was made in choosing the elements corresponding to these elements:

- **State:** A simple discretised value of the price in each time steps.
- **Reward:** The profit return after each transaction, which is

$$R_{t+1} = (X_{t+1} \times S_{t+1} + C_{t+1}) - (X_t \times S_t + C_t)$$

- **Action:** Transactions of the agents. These transactions are modified so that it fit the requirements of concrete state-action space for the learning algorithm being used, Q-learning. To be specific
  - **Action = 0:** The agent stay without placing any transactions
  - **Action = 1:** The agent sells a random number of securities (1 or 2)
  - **Action = 2:** The agent buys a random number of securities (1 or 2)

4.4.2 Q-learning Agent

The algorithm used to train the higher intelligent agent is Q-learning. In general, the agent is instantiated with its own Q-learning table, which would be continuously updated as each training

episode is implemented.

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma maxQ'(s',a') - Q(s,a)]$$

with

- α is the learning rate
- γ is the discount rate
- $Q(s,a)$ is the current Q value
- $R(s,a)$ is the reward for action a in state s
- $maxQ'(s',a')$ is the maximum expected future Q value given a new state s' and all of the subsequent action from s'

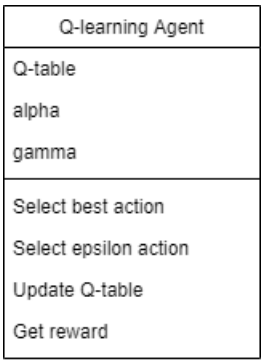


Figure 4.8: Class structure of RL Agent as an inheritance of Agent from Figure 4.5

The Q-learning formula above is implemented using the update Q-table function of the agent. The agent also has an action choosing function, which would select an appropriate action every step, based on the epsilon greedy algorithm [19], to balance the trade-off between exploration and exploitation. However, during the test episodes to test the policy generated from the Q-learning algorithm, since exploration is not necessary, a total greedy approach is implemented to maximize the reward.

Chapter 5

Testings, Experiments and Simulations

This chapter discusses the robust testings on functional requirements of the market and agents. As a research-oriented project, the simulations. The chapter will also feature results and findings of agents' impact on the market simulation.

5.1 Robust Testings

Robust Testings serve to testify and solidify the creation of agents and market simulation implemented in the project. Apart from the Random Agent, we tested both a Trend Following Agent and a Mean Reversion Agent and found the results complying with expected functional behaviour. In most of the simulations, the stock price model is a cosine function. This decision is to address the understandably expected behaviour of a trading strategy in such a market. The figure below is the price observation of a single Trend Following Agent entering the market.

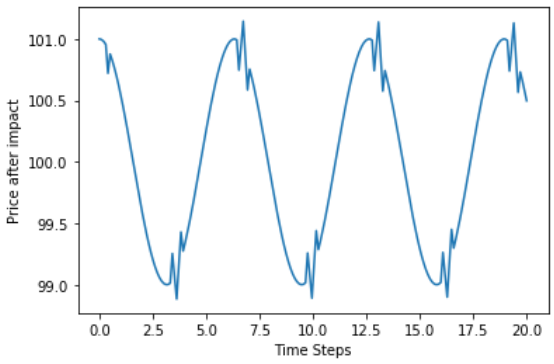


Figure 5.1: Cosine stock price when a single Trend Following Agent enters the market

In an ideal cosine price market, the algorithm expects trend following agent to sell its securities after the highest points in price (the peaks of the cosine graph) and buy more securities after the lowest points in price (the valleys of the cosine graph). The resulting time-series price showed that at each critical point, there were changes in price as a result of the activities of the Trend Following Agent. The explanation is if there is an upward or downward trend in the price, the comparison between price moving average and the current price is likely to be unchanged (higher or lower). In contrast, at these critical points, this comparison will change, since there is an opposite change in the trend. Note that although the concrete output for the comparison during the upward and downward slope imposes a decision of buying or selling securities on the agent, thanks to leverage principle in *Section 4.3.5*, the agent number of orders the agent would transact is 0. This principle effectively prevented scenarios when agents are constantly buying or selling without limitation and skewing the data points.

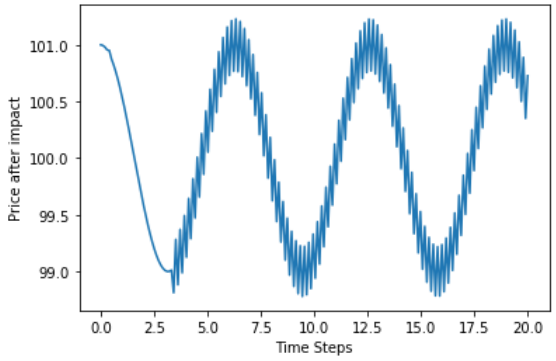
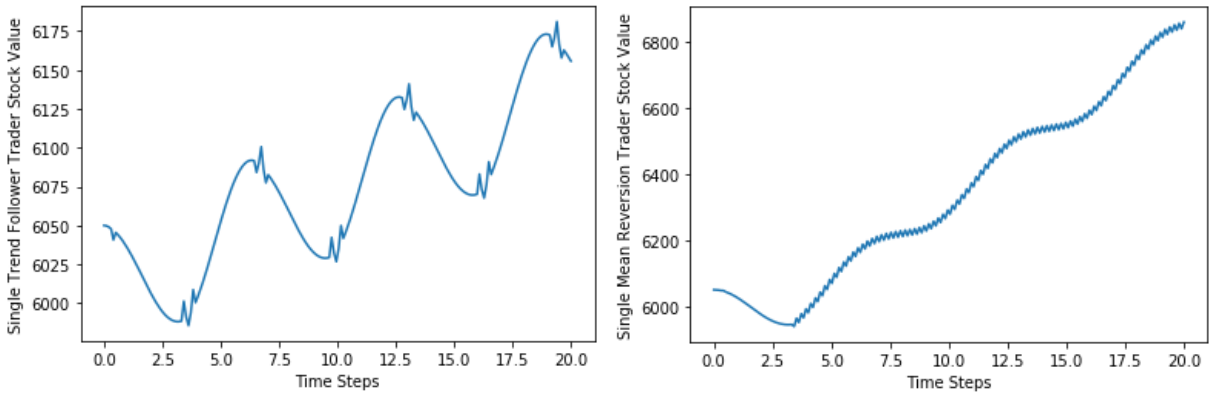


Figure 5.2: Cosine stock price when a single mean reversion agent enters the market

Unlike the smooth and predictable activities of Trend Following Agent, the test on Mean Reversion Agent might be a little difficult to understand. As Trend Following Agent, Mean Reversion Agent also starts its trading session at one of those critical peaks or valleys. However, once it enters the market, Mean Reversion Agent will constantly make a transaction in each time step, resulting in the zic-zac movements of the price. This movement is because Mean Reversion Agent acts oppositely to the market trend. E.g. At the first valley of the price model, Mean Reversion will be selling its securities, which effectively drives down the price. This decrease in price combined with the upward trend would meet Mean Reversion's condition to make a buying decision, and continue to cycle through buy/sell decisions within the trading session.

And finally, as the main purpose of algorithmic trading is to make automated profit, tests on both scenarios showed that these agents are making profits in the long-term, with Mean Reversion Agent has its profit more aggressively developed due to condensed activities within the trading session.



(a) Trend Following

(b) Mean Reversion

Figure 5.3: Portfolio Value of a single agent enter the Cosine Stock Model market

5.2 Experiments

5.2.1 Analysis criteria

There are several simulations have been executed in order to observe the impact of agents on this simulation of market. The analysis is often based on observing the following variables: price return, average price and volatility. Price return is often used to investigate the price different in each time step. It is the measurement of percentage change in the price, and its formula is

$$R_t = \frac{S_t - S_{t-1}}{S_{t-1}}$$

The price volatility is different from the noise volatility described in *Section 4.2.2*, and it actually refers to the stability of price over a period of time [15], and can be describe using this formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(R_i - m)^2}{N-1}}$$

with

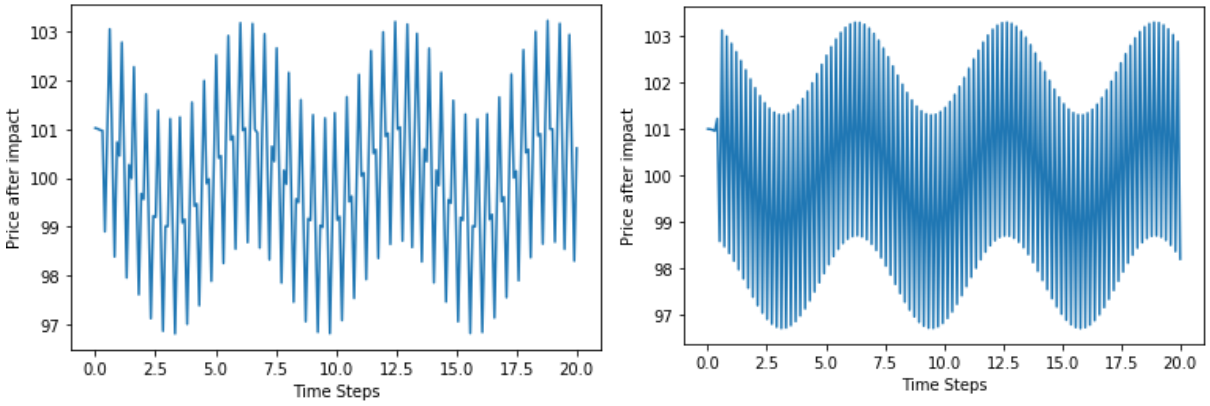
- R_i is the price return
- m is the mean of all returns
- N is the total number of time steps

For the Cosine Stock Model, this volatility value is close to 0.0006. The following table explained the test cases name being used, with each of the cases includes an appropriate number of Random Trader for noises.

Name	Trend Following	Mean Reversion
TF	100%	0%
FifthTF	20%	80%
QuarterTF	25%	75%
Half	50%	50%
QuarterMR	25%	75%
FifthMR	20%	80%
MR	0%	100%

5.2.2 Simulations

The above approach in trader compositions was carried out to observe agents impact on the price market. The number of Random Trader is 2. It is quite expected that the in the **TF** and **MR** tests, a large number of Trend Follower and Mean Reversion alone will be very disruptive since all these agents are acting towards a single price change, thus leading to unanimous actions. This price dynamic is also similar to the robust test, in a sense that Mean Reversion tends to be more active and more disruptive to the market price.

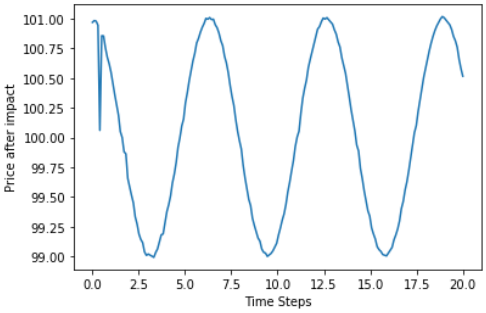


(a) Trend Following

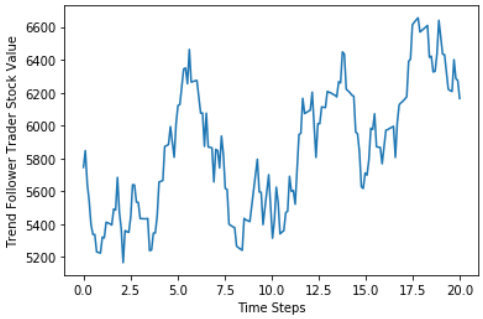
(b) Mean Reversion

Figure 5.4: TF with 10 Agents and MR with 10 Agents

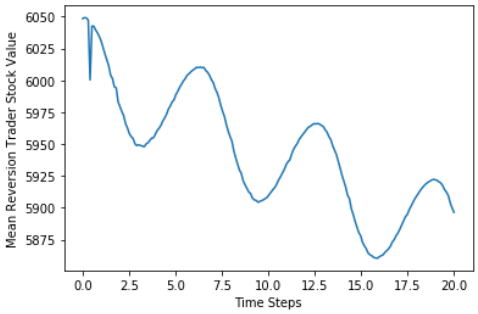
While the impacts on price observed in **TF** and **MR** tests is largely different, in the **Half**, the resulting price is actually a stable dynamics that closely resembles that of the original Cosine Model. The portfolio value of a Trend Following Agent and a Mean Reversion Agent suggests that if there is a dominant number of Trend Following Agents (more than half), it is likely that their algorithms will perform better than that of Mean Reversion. These values also suggests that the Trend Following are more likely to be affected by randomized trading activities from the Random Agents, which suggested that it is more susceptible to external activities, especially in this Cosine Model market. Other experiments showed that in scenarios where Mean Reversion Agents are dominating in number, they will generate profit from Trend Following Agents' lost.



(a) Half Test



(b) Portfolio Value of a TF Agent



(c) Portfolio Value of a MR Agent

Figure 5.5: Half Test of 10 Agents and their Portfolio Value

The price return tests on these different scarios also suggest that the **Half** composition is the most stable and most closely matches the original Cosine Model.

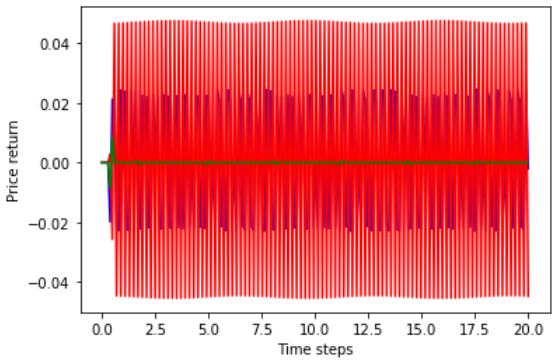
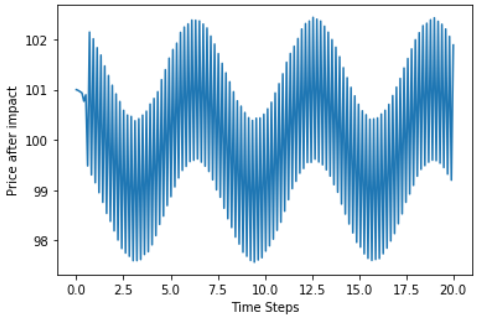
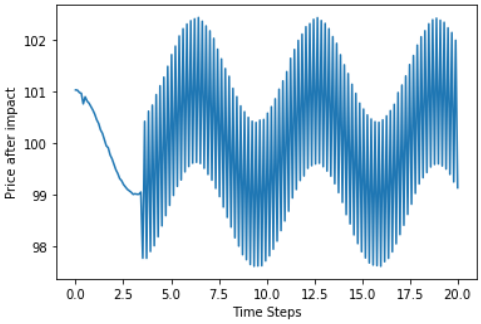


Figure 5.6: Difference in Price Return for TF (blue), MR (red), Half (green) compared to Cosine

However, there is an interesting finding on the **FifthTF** experiments, suggesting that the imbalance number of Trend Following and Mean Reversion Agents can sometimes have no impact on some stages in the trading session.



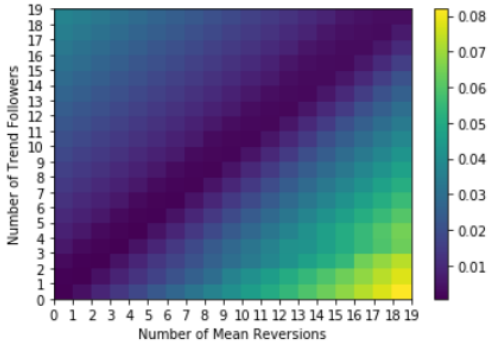
(a) QuarterTF Test



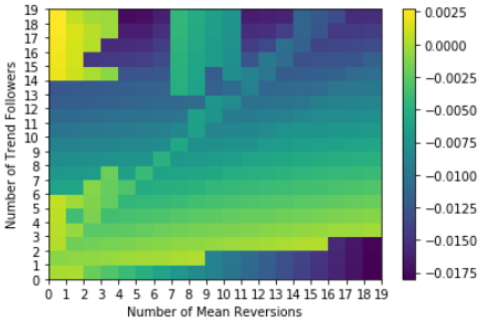
(b) FifthTF Test

The graphs showed that during the time from the beginning of trading session until the first critical valley, the composition of Agents in the **FifthTF** Test does not have an impact on the price at all, while it is not the case in the **QuarterTF** Test. The likely explanation is that the amount of orders both Trend Following and Mean Reversion Agents placed results in the price impact that is correlated and similar to the original Cosine Model, thus leading to inactivity or similarly cancel-out activity in the subsequent timesteps.

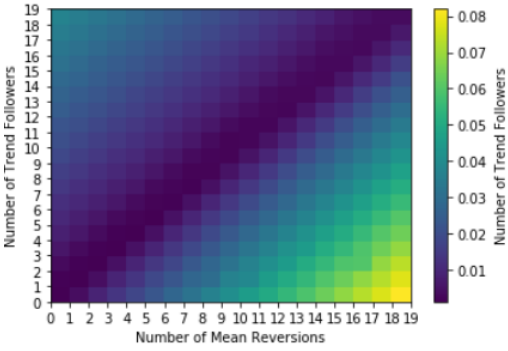
5.2.3 2D Simulations



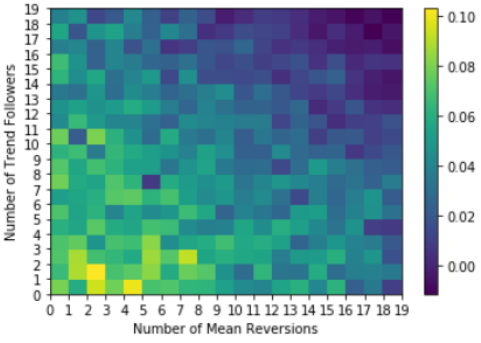
(a) Free Test Price Volatility



(b) Free Test Average Price



(c) Fixed Test Price Volatility



(d) Fixed Test Average Price

In comprehending the impact of different automated agents, the above test cases might not provide a generalized overview of those impacts. These simulations provide an additional approach to be more comprehensive in the composition of agents. These simulations showed the volatility and the average price (compared to the original average price) of a Cosine Model market, when there are 0 to 19 Trend Following Mean Reversion Agents. The first experiment is on a **Free** Market, where there is only Trend Following and Mean Reversion Agents. The second one is a **Fixed** Market, where there would be a fixed total of 38 agents, and any agents which are neither Trend Following or Mean Reversion would be Random Trader.

The 2D graphs affirmed that the drawn conclusions still hold, as in both cases, a market containing an equal number of Trend Following and Mean Reversion tend to be more stable, with relatively low price volatility. The average price graphs also confirm the observation in **FifthTF** test (note the unusual pattern in the lower right corner of Free Test Average Price), and suggests that a fraction of Trend Following to Mean Reversion Agents often results in smaller price changes. The Average Price in Fixed Test also showed the irregularities resulting from the randomized activities of Random Traders.

5.2.4 Real Market Data

So far, the Cosine Model is ideal and far from resembling a realistic market. Therefore, we attempted to use a real time-series stock price and placed it within the market model, concerning orders execution and agents activities, to observe the impact from a more irregular and realistic price pattern. The stock chosen is the **AAPL** stock index, and the data were the last 200 daily opening prices.

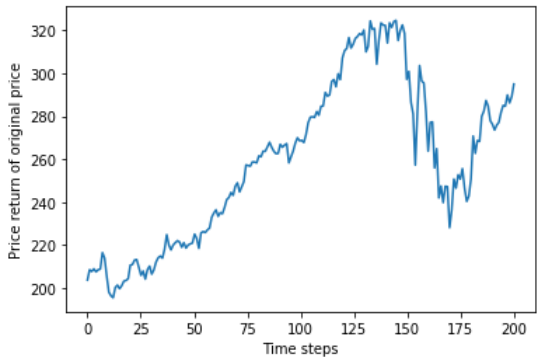
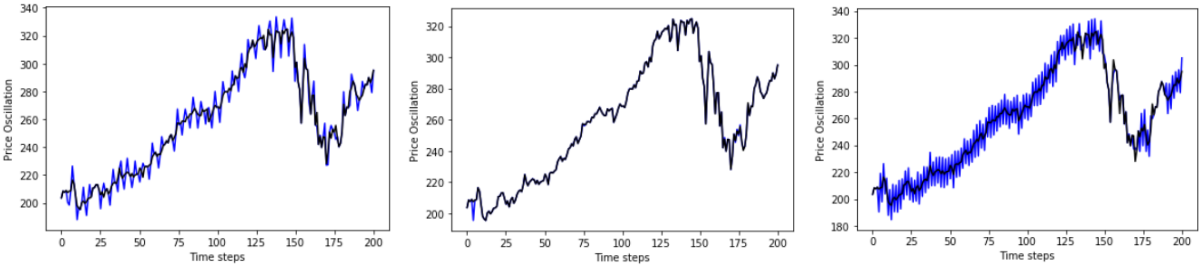


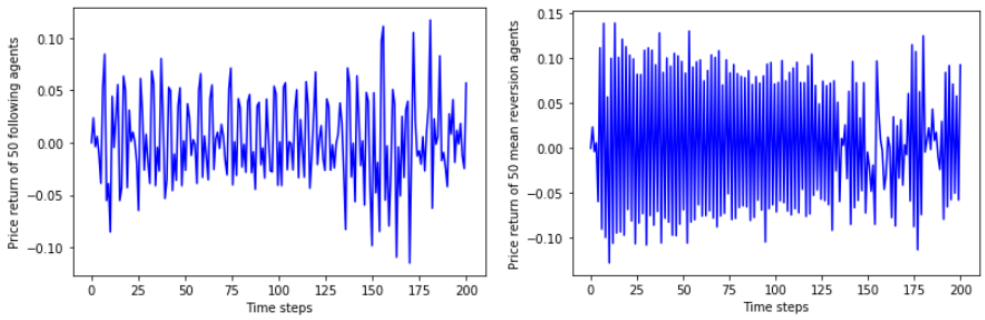
Figure 5.9: Apple opening stock price as a time-series model

The test cases are **QuarterTF**, **Half** and **QuarterMR**. The following graphs are the stock prices compared to the original **AAPL** price:



(a) QuarterTF Test (b) Half Test (c) Quarter MR

The graphs showed that, as expected, a balanced composition of Trend Following and Mean Reversion Agents will likely have no impact on the stock prices. Observations also point out that in this realistic setting, Trend Following Agents will likely have impact when there is an opposite turn in the price, driving the prices in that critical direction. Meanwhile, the Mean Reversion Agents would likely be active for most of the trading session. However, they showed significant inactivity when there is a significant trend in the price, either upward or downward. Observations of price return in extreme **TF** and **MR** tests with no Random Traders confirmed these findings. The extreme **MR** test showed that on the duration when there is a clear decline in **AAPL** stock value, the drop in price remains mostly unaffected.



(a) Extreme TF Test (b) Extreme MR Test

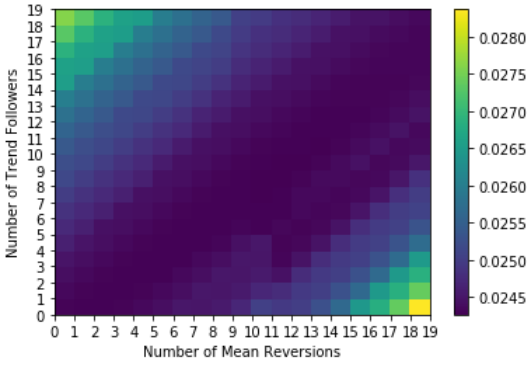


Figure 5.12: Free Test for AAPL Market

This real stock market also contained The 2D **Free** Test, which showed the same stability for an equal number of Trend Following and Mean Reversion Agents.

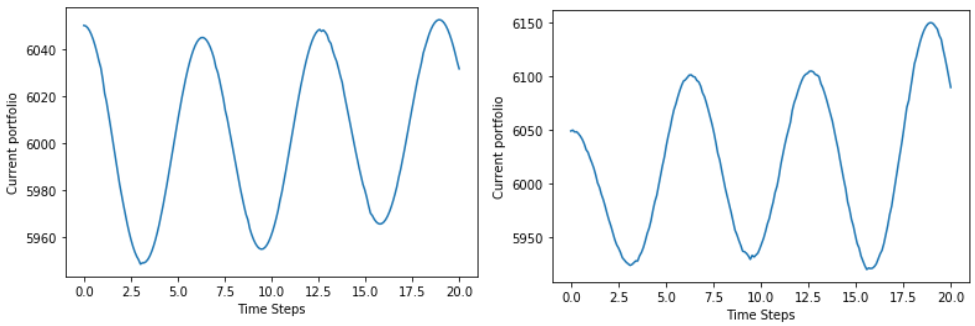
5.2.5 Conclusions

Previous studies on a simulated artificial market have suggested that the market impact of trading algorithm strategies may not only depend on its rule but also on its collaboration with other strategies [20]. The study in this project points out a further amendment to the above argument:

- Impact of automated strategies not only relies on its algorithm and combination with other strategies but also on the nature of the market (as seen on the Cosine Model and AAPL real market).
- Opposite strategies rule, given an ideal market environment with the same amount of participants in the market, might have little to no impact on the market themselves.
- A dominant number of agents from a strategy might skew the market towards their favour, and likely to generate profits from it.

5.3 Reinforcement Learning Agent

The implementation and algorithm in *Section 4.4* are carried out in training the Reinforcement Learning Agent. In training the agent, there are a total of 15000 training episodes. There are 100 policy tests carried out periodically every 1000 training episodes. The market environment is the Cosine Model. There were no other traders apart from the Reinforcement Learning Agent itself. Here are its portfolio value after 1000 and 15000 training episodes



(a) 1000 episodes (b) 15000 episodes

Figure 5.13: Portfolio of Reinforcement Learning Agent

The cosine-alike shape of the portfolio value is due to the small number of orders the agent placed onto the market. Therefore, its value largely depends on the price. Even in such cases, the agent

tends to generate more sequential rewards over time, leading to a higher value in its peak after more training episodes. The graph below showed the mean accumulative rewards during each 100 policy tests period, over the 15000 testing episodes. The actual observation showed that there would be periodically testing episodes when the agent would gain substantial profit, followed by a short period of lower rewards. The likely explanation is that the choice of state space is too simple, only relying on the discretized current value of the stock price. Therefore, it failed to address the current price in the market context, whether it is on an upward or downward trend. Nevertheless, the agent proves some higher intelligence by still making an incremental profit from the environment.

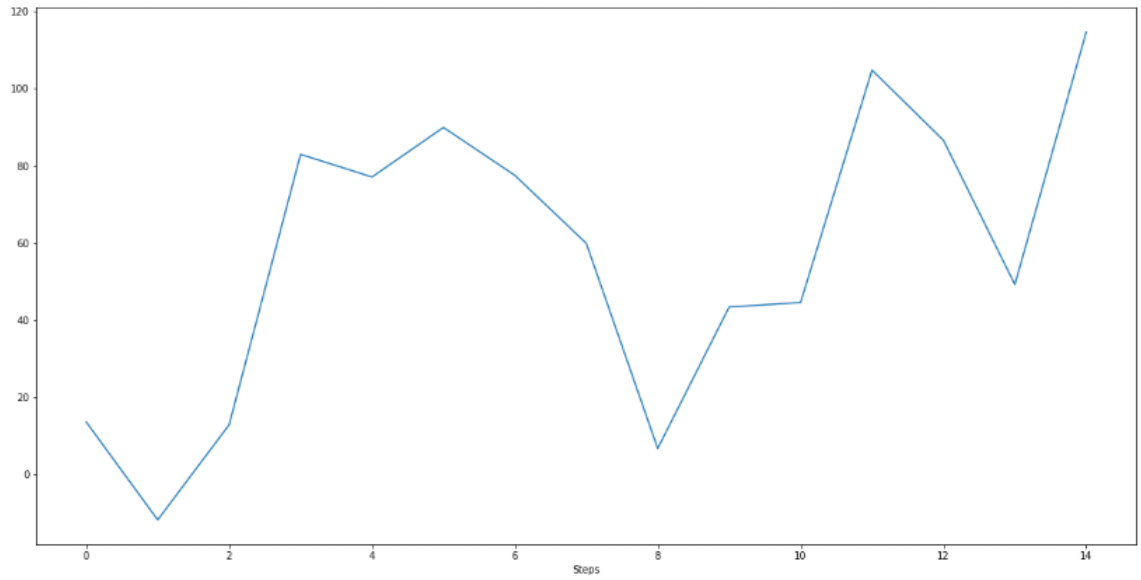


Figure 5.14: Accumulated Reward of Reinforcement Learning Agent

Note that this result would subject to be changed with each training sessions, as a portfolio optimized higher intelligent agent would require a more sophisticated state-action space definition and training algorithms. A further study for extended Q-learning or Neural Network application on Deep Reinforcement Learning can be found in [21].

Chapter 6

Conclusions

6.1 Achievements

The project has successfully achieved the fundamental defined goals:

- It has generated a time-series market model (Cosine Model) which can easily modify the model following any mathematically modelled market. The market also has extensional functions with real stock data.
- Three Algorithmic Trading Algorithms included, which are Random Trader, Trend Following, and Mean Reversion. The codebase also provided an abstract Agent Template for potential future extensions on the classes of Trading Algorithms.
- A general Test Cases Class to formalize experiments to form most of the simulations.
- A Reinforcement Learning gym was created from the market simulation. The project included a trained Reinforcement Learning Agent, although the expected portfolio is not guaranteed to be optimized.
- Many simulations on Cosine Market Model and Real Market Data Model to experiment and research market impact. The report presented some exciting findings.
- An appropriate structure code in Jupyter Notebook with Markdown comments.

6.2 Evaluation

In general, the project satisfied most of its goals and requirements of this project theoretically, and some many components also had its functions extended. As a research-based project, the self-developed project structure and exploration process have been of rational quality.

However, there would be many drawbacks and potential extensions on the project. For example, the main problem of the project is that the mathematical models are far too simple and far from resembling a real market structure and micro-behaviour. The trading strategies, while being suitable for the research purpose of this project, are not a very complicated one that would reflect real automated trading strategies. The state-action space of the reinforcement learning is limited, and the Q-learning algorithm is not among the current state-of-the-art approaches of Reinforcement Learning in Stock Trading.

There are also many assumptions within the implementation of a stabilized, naive, and ideal stock market with little to no external factor. The modifications of Almgren-Chriss trading model is an example, where we assumed that there has a previous trend encrypted in the underlying market model. Also, the market is lacking in its model in concurrent trading on each time step, where the temporary impact of trading activities plays a huge role.

These limitations are not necessarily hindering of the project, but instead acted as a motivation for further self-research and implementation ideas. These limitations within the simulation generated an impulse to formulate the problem and design to alleviate future extensions. At the same time, these limitations are also a great topic of discussion to explore feedback on incremental steps in improving the market simulations.

6.3 Impact of COVID-19

Since the project has research and experimental nature, the impact of this pandemic does not reflect on the overall achievements in simulating the stock market. However, COVID-19 had a clear negative impact on my working process on the project. As academic discussions with my supervisor and his research team inspire many ideas, including the important mathematical models' whiteboarding, virtual meetings limit the ideas flow of creating simulations and implementations. Besides, I suffered from severe symptoms of COVID-19 (explained in the EC form). This illness and the uncertainties of relocation during the pandemic lead to a month of disconnection from the project.

Aspects of the project that would be improved without COVID-19 include, but not limited to, generating new types of agents and different simulations, a more sophisticated market mathematical model, and a better state-action space for the reinforcement learning agent.

6.4 Future Work

This project has the potential to be extended. There is a myriad of market models and trading algorithms can be implemented into the structure of this market simulation, such as VWAP and POV agents [1]. There are plenty of opportunities to fit a better reinforcement learning algorithm, especially Deep Reinforcement Learning with Neural Network, which kept a history of prices as the stat spaces and immediately addresses the problem of pricing trend I addressed above. Finally, with the extended scope of trading algorithms, there would be more simulations and derived findings.

This project was an inspiration to engage in academic research and discussion. The process of bringing ideas into implementation and experiments was enjoyable, as it provided a sense of freedom in deciding the target findings that we would want to explore. For future work, we hope to extend the project on the above criterion in *Section 6.2*, so that it would be an user-friendly yet diversified simulation. If possible, we also hope to put the development into a concrete Python package and formalize its usage within the industry (for Algorithmic Trading teaching materials, for example).

Bibliography

[1] S. Shobhit. Basic of Algorithm Trading: Concepts and Examples. *Investopedia*, March 2020.

[2] D. MacKenzie. How to Make Money in Microseconds, 22: 16–18, 2011.

[3] N. Aggarwal and S. Thomas. The Causal Impact of Algorithmic Trading on Market Quality. *Indira Grandhi Institute of Development Research*, July 2014.

[4] P. Whiteley. Time-series Analysis. *Qual Quant*, 14: 225–247, 1980.

[5] A. Joseph. Time Series Analysis on Stock Market Forecasting (ARIMA & Prophet). *Medium*, December 2019.

[6] R.M. Kapila Tharanga Rathnayaka, D.M.K.N. Seneviratna, W. Jianguo, and H.I. Arumawadu A hybrid statistical approach for stock market forecasting based on Artificial Neural Network and ARIMA time series models. *2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC)*, 54–60, 2015.

[7] E. Bonabeau. Agent-based Modeling: Methods and Techniques for Simulating Human Systems. *PNAS*, 99: 7280–7287, 2002.

[8] K. Boer-Sorban, U. Kaymak, and J. Spiering. From Discrete-Time Models to Continuous-Time, Asynchronous Models of Financial Markets. *ERIM Report Series Reference No. ERS-2006-009-LIS*, March 2006.

[9] K. Boer-Sorban, U. Kaymak, A.D. Bruin, and M. Polman. An Agent-Based Framework for Artificial Stock Markets. *16th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'04)*, 83–90, March 2006.

[10] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew and I. Mordatch. Emergent Tool Use From Multi-Agent Autocurricula. September 2019.

[11] R. Almgrem and N. Chriss. Optimal Execution of Portfolio Transactions. *Journal of Risk*, 5–39, 1999.

[12] Y. Dai, C. Wang, I. Wang, Y. Xu. Reinforcement Learning for FX trading. *Stanford University*.

[13] Y.S. Lim and D. Gorse. Reinforcement Learning for High-Frequency Market Making. *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 25–27, 2018.

[14] W. Kenton. Limit Order Book. *Investopedia*, January 2020.

[15] Volatility. *AssetMacro*. `https://www.assetmacro.com/financial-terms/volatility/`

[16] K. Kafadar. Gaussian white-noise generation for digital signal synthesis. *IEEE Transactions on Instrumentation and Measurement*, 492–495, 1986.

[17] J. Chen. Mean Reversion Definition. *Investopedia*, May 2019.

[18] O. Momoh. How Leverage Works in the Forex Market. *Investopedia*, February 2019.

[19] A. Parkinson. The Epsilon-Greedy Algorithm for Reinforcement Learning. *Medium*, December 2019.

[20] K. Izumi, F. Toriumi, and H. Matsui. Evaluation of Automated-trading Strategies using an Artificial Market. *Neurocomputing*, 72: 16–18, October 2009.

[21] S. Duerson, F.S. Khan, V. Kovalev, and A.H. Malik. Reinforcement Learning in Online Stock Trading Systems. *Georgia Tech*, 2005.

Appendix A

Market and Agents variables

Market Variables

T	Session Time
dt	Time Interval
N	Time steps
X	Number of stock held
$cash$	Amount of cash held
S	Stock prices
S_{flat}	Flat market model
S_{cos}	Cosine market model
γ, ϵ, **and** η	Market Impact Variables

Agents Variables

$max_leverage$	Maximum Leverage
$min_leverage$	Minimum Leverage
$action$	Buy/Sell Decision
$transaction$	Orders Placed

Q-Agent Variables

num_action	Action States Size
ϵ	Decay Rate for learning
$learning_rate$	Learning Rate
$gamma$	Discount Rate
$table$	Q-table

Appendix B

Classes, Functions and Parameters

Parameters

`TestClass(X, S)`

- `X:` original securities of each traders
- `S:` price model

`Env(qTrader, S_original)`

- `qTrader:` Q-Agent within the market environment
- `S_original:` original price model

Functions

Only important functions are mentioned

`TestClass(X, S)`

- `add_trader(num, trader_type):` Add traders of specific quantity and type
- `update_price():` Update the price mode with the current state of traders
- `get_volatility(); get_price_return():` Get price volatility and price return
- `get_price(); plot_price():` Get and plot price

`Env(qTrader, S original)`

- `reset():` Reset the gym and return the observations
- `step(action):` Execute the action and return the next observation, reward and game state

`Real Market Data`

- `real_stock_price(symbol, outputsize=200, price_at='1. open')`: Return the stock price of the selected symbol, output size, and defined price state point (open, close, high, low)

Appendix C

Code Listings

Only the important source code will be listed.

Agent Class

```
class TradingAgent:
    def __init__(self, stock, cash):
        self.X = stock.copy()
        self.cash = cash.copy()
        self.min_leverage = 1
        self.max_leverage = 8
        self.action = np.zeros(np.shape(X))
        self.transaction = np.zeros(np.shape(X))

    def make_transaction(self, S, current_step):
        return np.random.randint(-2, 3)

    def trade(self, S, current_step):
        self.X[current_step] = self.X[current_step - 1] + self.make_transaction(
   S, current_step)
        self.cash[current_step] = self.cash[current_step - 1] - self.
   make_transaction(S, current_step) * S[current_step - 1]

    def get_stock(self):
        return self.X

    def get_cash(self):
        return self.cash

    def get_value(self, S):
        return [x * s + c for x, s, c in zip(self.X, S, self.cash)]
```

```
    def get_action(self):
        return self.action

    def get_transaction(self):
        return self.transaction
```

Random Trader

```
class RandomTrader(TradingAgent):
    def make_transaction(self, S, current_step):
        buyOrSell = np.random.randint(0, 2)
        if buyOrSell == 0:
            self.transaction[current_step] = self.buy(S, current_step)
            return self.buy(S, current_step)
        elif buyOrSell == 1:
            self.transaction[current_step] = self.sell(S, current_step)
            return self.sell(S, current_step)

    def buy(self, S, current_step):
        self.action[current_step] = 1
        if self.cash[current_step - 1] >= S[current_step - 1] * 2:
            return np.random.randint(-2, 1)
        else:
            return 0

    def sell(self, S, current_step):
        self.action[current_step] = -1
        if self.X[current_step - 1] >= 2:
            return np.random.randint(0, 3)
        else:
            return 0
```

Trend Following

```
class TrendFollower(TradingAgent):
    def make_transaction(self, S, current_step):
        if current_step <= 3:
            return 0
        mean = sum(S[current_step - 4:current_step - 1])/3
        if S[current_step - 1] < mean:
            self.transaction[current_step] = self.sell(S, current_step)
            return self.sell(S, current_step)
        elif S[current_step - 1] > mean:
```

```
            self.transaction[current_step] = self.buy(S, current_step)
            return self.buy(S, current_step)
        else:
            return 0

    def buy(self, S, current_step):
        self.action[current_step] = 1
        totalValue = X[current_step - 1] * S[current_step - 1] + cash[
    current_step - 1]
        return np.floor((self.max_leverage / (self.max_leverage + 1)) *
    totalValue / S[current_step - 1]) - self.X[current_step - 1]

    def sell(self, S, current_step):
        self.action[current_step] = -1
        totalValue = X[current_step - 1] * S[current_step - 1] + cash[
    current_step - 1]
        return np.ceil((self.min_leverage / (self.min_leverage + 1)) *
    totalValue / S[current_step - 1]) - self.X[current_step - 1]
```

Mean Reversion

```
class MeanReversion(TradingAgent):
    def make_transaction(self, S, current_step):
        if current_step <= 3:
            return 0
        mean = sum(S[current_step - 4:current_step - 1])/3
        if S[current_step - 1] > mean:
            self.transaction[current_step] = self.sell(S, current_step)
            return self.sell(S, current_step)
        elif S[current_step - 1] < mean:
            self.transaction[current_step] = self.buy(S, current_step)
            return self.buy(S, current_step)
        else:
            return 0

    def buy(self, S, current_step):
        self.action[current_step] = 1
        totalValue = X[current_step - 1] * S[current_step - 1] + cash[
    current_step - 1]
        return np.floor((self.max_leverage / (self.max_leverage + 1)) *
    totalValue / S[current_step - 1]) - self.X[current_step - 1]

    def sell(self, S, current_step):
        self.action[current_step] = -1
        totalValue = X[current_step - 1] * S[current_step - 1] + cash[
```

```
    current_step - 1]
        return np.ceil((self.min_leverage / (self.min_leverage + 1)) *
    totalValue / S[current_step - 1]) - self.X[current_step - 1]
```

Volatility factor

```
## independent random variable in volativity
xi = np.insert(np.random.normal(0, 1.0, N - 1), 0, 0.0)

# volatility
volatility = 0.2

#volatility factor
def volatility_factor(volatility, xi):
    return volatility * xi
```

Market Impact

```
def permanent_impact(n, gamma):
    return gamma * n

def temporary_impact(n, epsilon, eta):
    return epsilon * np.sign(n) + eta * n
```

Real Market Fetching Function

```
def real_stock_price(symbol, outputsize=200, price_at='1. open'):
    ts = TimeSeries(key=api_key, output_format='pandas')
    data, _ = ts.get_daily(symbol=symbol, outputsize='full')
    return data[price_at][-outputsize:], np.linspace(0, outputsize, outputsize)
```

Test Case Class

```
class TestCase:
    def __init__(self, X, S_original):
        self.S = S_original.copy()
        self.n = np.zeros(np.shape(X))
        self.traders = []

    def add_trader(self, num, trader_type):
```

```
        if num == 0:
            return
        for i in range(num):
            self.traders.append(trader_type(X, cash))

    def get_trader(self, index):
        return self.traders[index]

    def update_price(self):
        step = 0
        for i in range(N):
            step = step + 1
            for trader in self.traders:
                trader.trade(self.S, i)
                self.n[i] += trader.get_stock()[i - 1] - trader.get_stock()[i]
            self.S[i] = self.S[i] - permanent_impact(self.n[i], gamma)

    def get_price(self):
        return self.S

    def get_price_return(self):
        price_return = [(self.S[t + 1] - self.S[t]) / self.S[t] for t in range(N
    - 1)]
        price_return.insert(0, 0)
        return price_return

    def get_stock_difference(self):
        return self.n

    def get_volatility(self):
        price_return = self.get_price_return()[1:]
        variance = np.square([x - np.average(price_return) for x in price_return
    ])
        return math.sqrt(np.sum(variance) / (N - 1))

    def get_standard_deviation(self):
        variance = np.square([x - np.average(self.S) for x in self.S])
        return math.sqrt(np.sum(variance) / N)

    def get_variance(self):
        return [self.S[i] - S[i] for i in range(N)]

    def plot_price(self):
        plt.plot(t, self.S)
        plt.xlabel('Time Steps')
        plt.ylabel('Price after impact')
```

```
        plt.show()

    def plot_price_return(self):
        price_return = self.get_price_return()
        plt.plot(t, price_return)
        plt.xlabel('Time Steps')
        plt.ylabel('Price return')
        plt.show()
```

Q-learning Agent

```
class QTrader(TradingAgent):
    def __init__(self, stock, cash):
        super().__init__(stock, cash)
        self.num_actions = 3
        self.epsilon = 1
        self.gamma = 0.95
        self.learning_rate = 0.8
        self.table = collections.defaultdict(float)

    def reset_table(self):
        self.table = collections.defaultdict(float)
        self.epsilon = 1

    def reset_trader(self, stock, cash):
        self.X = stock.copy()
        self.cash = cash.copy()

    def select_eps_greedy_action(self, obs):
        _, action = self.best_action_value(obs)
        if random.random() < epsilon:
            return random.randint(0, self.num_actions - 1)
        else:
            return action

    def select_greedy_action(self, obs):
        _, action = self.best_action_value(obs)
        return action

    def best_action_value(self, state):
        best_action = 0
        max_value = 0
        for action in range(self.num_actions):
            if self.table[(state, action)] > max_value:
                best_action = action
```

```
                max_value = self.table[(state, action)]
        return max_value, best_action

    def q_learning(self, prev_obs, next_obs, reward, action):
        best_value, _ = self.best_action_value(next_obs)
        q_target = reward + self.gamma * best_value
        q_error = q_target - self.table[(prev_obs, action)]
        self.table[(prev_obs, action)] += self.learning_rate * q_error

    def update_epsilon(self, decay_rate):
        self.epsilon *= decay_rate

    def make_transaction(self, S, current_step, action):
        if action == 0:
            self.action[current_step] = 0
            self.transaction[current_step] = 0
            return 0
        elif action == 1:
            self.transaction[current_step] = np.random.randint(-2, 0)
            return self.transaction[current_step]
        elif action == 2:
            self.transaction[current_step] = np.random.randint(1, 3)
            return self.transaction[current_step]

    def trade(self, S, current_step, action):
        trade = self.make_transaction(S, current_step, action)
        self.X[current_step] = self.X[current_step - 1] + trade
        self.cash[current_step] = self.cash[current_step - 1] - trade * S[
    current_step - 1]

    def get_reward(self, S, current_step):
        return (self.X[current_step] * S[current_step] + self.cash[current_step
    ]) - (self.X[current_step - 1] * S[current_step - 1] + self.cash[current_step
     - 1])
```

Reinforcement Learning Environment

```
class Environment:
    def __init__(self, qTrader, S_original):
        self.S_original = S_original.copy()
        self.S = self.S_original.copy()
        self.n = np.zeros(np.shape(X))
        self.current_step = 0
        self.traders = []
        self.qTrader = qTrader
```

```
def reset(self):
    self.qTrader.reset_trader(X, cash)
    self.S = self.S_original.copy()
    self.n = np.zeros(np.shape(X))
    self.current_step = 0
    return discretise(self.S[0])

def step(self, action):
    for trader in self.traders:
        trader.trade(self.S, self.current_step)
        self.n[self.current_step] += trader.get_stock()[self.current_step -
1] - trader.get_stock()[self.current_step]
    self.qTrader.trade(self.S, self.current_step, action)
    self.n[self.current_step] += self.qTrader.get_stock()[self.current_step
- 1] - self.qTrader.get_stock()[self.current_step]
    self.S[self.current_step] = self.S[self.current_step] - permanent_impact
(self.n[self.current_step], gamma)
    self.current_step += 1
    return discretise(self.S[self.current_step - 1]), self.qTrader.
get_reward(self.S, self.current_step - 1), self.current_step >= N

def plot_price(self):
    plt.plot(t, self.S)
    plt.xlabel('Time Steps')
    plt.ylabel('Price after impact')
    plt.show()

def plot_value(self):
    value = self.qTrader.get_value(self.S)
    plt.plot(t, value)
    plt.xlabel('Time Steps')
    plt.ylabel('Current portfolio')
    plt.show()
```

Appendix D

Project Plan

D.1 Project abstract and objectives

D.1.1 Abstract

The innovations in artificial intelligence have opened new approaches to market analysis in financial computing. In this project, a very simplified version of a trading market with the transactions of a single stock will be implemented. The market will then be filled with trading bots with different intelligent parameters, ranging from randomized traders to potentially simple reinforcement learning algorithm, to be involved in those transactions. The impact of these agents on market volatility and financial stability will be observed and analyzed.

D.1.2 Objectives

The objectives of the project are:

- To gain a better understanding of the finance market and price impacts in trading.
- To program, specifically Python, with applied scientific data analysis and quantitative modelling.
- To learn about algorithmic trading and simple reinforcement learning approaches that can be applied.
- To explore about research academic as a career.

D.2 Expected Deliverables

- Summary of literature review that contains work from referenced resources that has been extracted with information relevant to the research topics (trading strategy, market impact and reinforcement algorithm).

- Code base for the market simulation and intelligent traders, implemented with Python in Jupyter Notebook.
- Detail analysis on price dynamics and explanation on intelligent parameter.
- Final Report.

D.3 Work Plan

D.3.1 Phase 1: Meetings with supervisor, project goals and knowledge exchange

Time: Early October – Early November
Goal:

- Get to know more about supervisor.
- Get a clear understand about the project, including requirements, context and implementation.
- Review literature relating to the topic matters and start documenting them.
- Get in touch with other people who can potentially assist with the project.

D.3.2 Phase 2: Simulation of market

Time: Early Mid November – Mid December
Goal:

- Implement the simple market environment.
- Add simple non-intelligent traders (e.g. randomized traders).
- Fill the market with zero intelligent traders, if possible (e.g. trend followers).
- Learn about applied reinforcement learning.

D.3.3 Phase 3: Adding traders

Time: Mid Late December – Mid February Goal:

- Add traders to the environment, modifying market price parameters if necessary.
- The target is to have a reinforcement learning trading algorithm applied.

D.3.4 Final Phase: Observation, improvement and analysis

Time: Mid February – End of Project
Goal:

- Observing the different combination of traders' intelligence impact on price to analyze.
- Make changes to market environment and traders' intelligence in necessary.
- Write the final report.
- Finalize the codebase.

Appendix E

Interim Report

E.1 Project Expected Plan

The expected step by step progress of the project are:

- Create a static market with underlying price functions.
- Add traders to the market.
- Research on different trader structures and its overall impact on driving the market price.
- Add advanced intelligent traders (potentially Reinforced Traders).

E.2 Progress Up-to-date

- Simulated the price dynamics on a market.
- Added volatile noise to the price/fixed the initial price point for better analysis on traders' impact.
- Efficiently optimized market price updates.
- Added different traders – including Randomized Traders, Trend Follower and Mean Reversion.
- Inspected the effect of those traders with different composition.

E.3 Future Plan

- Diversify the space of traders.
- Implement higher/advanced intelligent traders, if possible.

- Complexify the trading algorithm to the market (e.g. Buy/sell decision).
- Inspect further market price impacts.
- Project Report.