Imperial College London

Imperial College London

Department of Mathematics

Discrete Hedging and Pricing of European Options using Reinforcement Learning

Author: Umor Sami (CID: 00934743)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2019-2020

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to express my sincere gratitude and appreciation to Gordon Lee for his advice, guidance and for the opportunity to work on this topic.

I would also like to thank Dr. Thomas Cass, for his support and important feedback whenever needed. I would also like to thank the Maths and Finance department for providing a pleasant learning experience at Imperial College London.

Lastly, I would like to thank my family and friends. This would not have been possible without their support in these testing times.

Abstract

Presented in this paper will be a discrete option pricing and hedging model using Reinforcement Learning, specifically, the Q-learning algorithm implemented by Chris Watkins in 1989 [1]. This is achieved by first establishing a Markov Decision Process for a discrete-time Black-Scholes setting where the price of the option is the optimal action-value, while the optimal hedge is represented as the optimal action. Hence, both pricing and hedging are contained within the same formula. This is achieved by an agent, which optimally re-balances a portfolio of bonds and the underlying asset as to maximize the risk-adjusted returns. We will also evaluate the use of neural networks and kernel estimation to help us estimate optimal hedges in a reinforcement learning setting. Reinforcement learning is particularly attractive to finance as it also allows for model-free methods and can learn to solve the pricing/hedging problem without the explicit dynamics of the world; learning directly from market data.

Contents

List of Figures

List of Tables

Introduction

Reinforcement Learning (RL) is one of the fastest growing research sectors within the field of machine learning. The application of reinforcement learning in the real world is endless, from robotics to optimising chemical reactions [2] to superhuman performance in Atari games [3]. The fundamental aim of reinforcement learning is to train an agent to undertake actions in an environment in which the dynamics are unknown, and to do so optimally in the sense of maximising rewards received by the environment. This idea of choosing optimal actions seems very well suited to many problems within the realm of finance such as trading or investment decisions. In this paper, we will be following a variety of ideas formulated by Igor Halperin in his paper on Q-learning in the Black-Scholes World (QLBS) [4]. In particular, we will see how we can reformulate the discrete-hedging problem in an RL setting.

Before we delve into the world of finance, one of the most popular platforms for testing RL algorithms is the OpenAI Gym [5]. This library offers a variety of continuous space and discrete space environments such as the classical Cartpole and Gridworld environments. As such, we can create similar environments to evaluate RL algorithms for financial purposes. These environments allow one to test both discrete and continuous action space RL algorithms. In the problems we will encounter, our action-space is the ability to trade an asset for hedging purposes.

The benchmark model used in this paper is the famous and classical Black-Scholes model (BS), we will be using the discrete version of the Black-Scholes model [6]. One of the major underlying ideas of the BS model is that derivatives can be priced using a replicating portfolio consisting of tradable assets (generally underlying stock and cash) called the hedge portfolio. The wealth in this portfolio is re-balanced between the stock and cash in a self-financing manner such that there are no cash injections/withdrawals apart from at the inception of the portfolio. The purpose of the replicating portfolio and dynamic nature of re-balancing is to emulate the pay-off of the derivative.

In the original continuous time-setting of Black-Scholes, the hedge portfolio can be re-balanced continuously to dynamically replicate the option. In this setting, the hedge portfolio *perfectly* replicates the option, thus eliminating mis-hedging risk. As such, the combined portfolio of the option and hedge portfolio earns the *risk-free* rate. As the mis-hedging risk is eradicated, the price of the option does not depend on the risk preferences of the investor and is equal to the unique price of the perfectly replicated hedge portfolio of stock and cash.

In this setting, an option can be perfectly replicated by a simple, straightforward portfolio of stock and cash. If the real world followed this model, it would render options superfluous as they can be easily replicated and no one would trade them. However, the derivatives market is vast, the notional amount of OTC derivatives rose to $640 trillion at end-June 2019 [7]. They are of paramount importance in the financial sector.

This is because the assumptions of the original Black-Scholes model break down in the real world which causes the option to no longer be risk-free. Many factors contribute to options being risky, such as transaction costs, volatility risk, cost of funding, etc [8]. We will mainly focus on mi-shedging risk, which arises when we relax the unrealistic assumption of continuous re-balancing of the hedge portfolio.

In most option markets, the replicating portfolio is re-balanced in discrete time steps, e.g. daily. Moreover, with the introduction of transaction costs, re-balancing frequently can lead to accruing large costs. In this setting, a continuous-time limit analytical solution may not even exist as the transaction costs may cause the prices to blow out of proportion due to the infinite number of trades taking place in the continuous-time limit.

As we are now working in a discrete-version of the BS model, creating a perfectly replicating portfolio of the option is no longer possible. The value of the hedging portfolio will generally be different from the option value, this will depend on the stock price evolution between consecutive hedges of the portfolio. The option buyer/seller will now charge a risk premium to offset the potential losses due and uncertainty introduced by mis-hedging risk. The price of the option is adjusted accordingly to encapsulate the risk taken on by the investor and depends on the investors' view on risk.

The aim of an option seller in a discrete-time setting is to minimize the costs caused by mis-hedging risk over the lifetime of the option by dynamic option replication. The option seller has to make decisions (rebalancing hedge position) in a sequential manner with the current observations of the market (e.g. stock price) with the purpose of minimizing (or maximizing rewards defined as negative costs) mis-hedging risk. The sequential decision-making nature of dynamic option replication with a goal of minimizing risk makes it highly compatible with **Reinforcement Learning** (RL). We can use RL methods in various ways to gain insight into the hedging problem. For example, one could amend the reward function to include transaction costs, this is a straightforward task yet the analytical solution to incorporating transaction costs can be challenging and taxing. Extensions using RL is relatively simple and straightforward. In this paper we will mainly focus on hedging the option rather than pricing.

One of the major advantages of using RL methods is the flexibility of either using a model or going model-free. One of the most famous model-free methods is the Q-learning algorithm which allows one to obtain the optimal policy (e.g. best actions) without the need to build a model of the dynamics of the environment. The original Q-learning algorithm was built to work in a discrete-state setting, which is suited for simple environments (i.e. Gridworld). This algorithm breaks down for continuous state settings (i.e. Cartpole). One solution to this is to incorporate neural networks to approximate action-value functions or even policies. Neural Networks paved the way for a new sector of RL, namely Deep Q-Networks (DQN) [9].

The method we use in this paper is called the QLBS model as it combines both Q-learning and the discrete-version of the BS model [4]. In this model, the option price and delta are contained in the same function, the optimal action-value function Q. This method could be used for model-based RL or model-free when the dynamics are unknown. In a model-free setting, the QLBS model can work with real data to train an agent to learn the optimal price and hedge for a dynamic replicating portfolio. Without an explicit model, one can use value-based RL methods to try and solve the Bellman Optimality Equation using only sample data. [10, Chapter 3.6, page 64] In this paper, we will only work with simple vanilla European Options but it is possible to extend the model to include more complex derivatives.

In the first section of the paper, we will look at the foundations of Reinforcement Learning and analyze common RL methods as well as explaining some key concepts. In the second section, we will formulate a discrete-time, continuous observation space setting of the Black-Scholes model and introduce the QLBS model to help us obtain a Dynamic Programming based solution for hedging and pricing. We will look at a variety of methods of solving from discrete-state RL algorithms to kernel estimation and utilizing neural networks. In the last chapter of this paper, we will move away from a Black-Scholes world, and see how well our Q-learning framework works in a different setting to Black-Scholes.

Chapter 1

Foundations of RL

In this section, we will discuss the basics of reinforcement learning and define key ideas within RL, which entails Markov Decision Processes (MDP) and Dynamic Programming. We will also touch upon Q-learning and other RL methods.

1.1 Basic Definitions

- **State** - Information which depicts the internal representation of the environment. This is the information used by the agent to choose the next action.
- **Agent** - This is the entity that interacts with the environment by undertaking actions based on the information available.
- **Environment** - The environment is the entity that analyzes the action taken by the agent and emits the subsequent state/observation and reward.
- **Reward** - This is a special numerical value the agent receives from the environment as a consequence of undertaking a certain action.

1.1.1 Finite Markov Decision Process (MDP)

Definition 1.1.1 (Markov Decision Process)**.** This is a 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$ where:

- $\mathcal{S}$ is a set of finite states called the state space
- $\mathcal{A}$ is a set of finite actions called the action space
- $\mathcal{P}$ is a probability state transition matrix such that $P_a(s, s', r) = \mathbb{P}(S_{t+1} = s, R_{t+1} = r | S_t = s, A_t = a)$ is the probability that action a will cause state s to transition to state s' at time t whilst receiving reward r.
- $\mathcal{R}$ is a Reward matrix such that $R_a(s, s')$ is the immediate reward received when state s transitions to state s' under the action a.

The agent and the environment interact with each other at discrete time steps. At time t, the agent receives some information about the environment's state $S_t \in \mathcal{S}$, in our setting, this information could be stock price. The agent then analyzes this information to select an action $A_t \in \mathcal{A}$, e.g. hedge position. The environment then processes this action and a time step later, emits a reward R_{t+1} and new state S_{t+1} to the agent. The agent then processes this new state and repeats this entire process either indefinitely or until it reaches a terminal state, producing a trajectory or episode as follows:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots$$

Definition 1.1.2 (**Policy**)**.** The way in which an agent selects actions is called a policy. A policy is a map π defined as follows:

$\pi : \mathcal{A}$ x $\mathcal{S}$
$\pi(a, s) = \mathbb{P}(a_t = a | s_t = s)$

This is the probability of the agent selecting action a when in state s at time t.

1.1.2 Returns and Value Functions

The goal of an agent is to find a policy that will maximize the *total* reward received. This is the cumulative reward received in the long run and not the short-sighted immediate reward. An action may lead to great immediate reward but in the future, it may be inferior to other possible actions.

Definition 1.1.3 (Returns)**.** The total discounted accumulated rewards from time t up to a final time step T with discount factor $\gamma \in [0, 1]$ is called the Returns G_t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{T-1} R_T$$

The terminal time step could be thought of as the expiration of an option in our problem or it could be the end of a game as another example. The episodes end in a terminal state and episodes start independently from one another. An episode can be seen as one simulation of the lifetime of an option. The purpose of the discount factor γ is to make sure that the total reward is finite as some problems do not have a terminal state and so rewards are received continually. Moreover, the discount factor allows us to adjust our view on short-term or long-term rewards. A discount factor of 1 means that we value rewards far into the future equally to immediate rewards (far-sighted), whereas a discount factor of 0 means we ignore future rewards and only care about immediate rewards (myopic). The way in which the agent should maximize the total reward is by maximizing the *expected* returns $\mathbb{E}_t[G_t]$.

Definition 1.1.4 (Value Function $V_\pi(s)$)**.** The Value function $V_\pi(s)$ is the expected return when the agent follows a policy π and starts in state s:

$$V_\pi(s) = \mathbb{E}_t^\pi[G_t | S_t = s]$$

The value function can be seen as a measure of how beneficial it is for the agent to be in a given state under a certain policy. The policy that maximizes the expected returns is the optimal policy and is not necessarily unique.

Definition 1.1.5 (Action-Value Function $Q_\pi(s, a)$)**.** The Action-Value function $Q_\pi(s, a)$ is the expected return when the agent and starts in state s and takes action a and thereafter follows policy π:

$$Q_\pi(s, a) = \mathbb{E}_t^\pi[G_t | S_t = s, A_t = a]$$

1.2 Bellman Equations

We will now explore the famous Bellman equations by Richard Bellman [11]. These equations are essentially the pillars of Reinforcement Learning. One of the core ideas used throughout RL is the recursive relationship that is satisfied by the value functions which we will formulate below.

1.2.1 Bellman Expectation Equation

The value function can be rewritten into a recursive relationship in the following way using the law of iterated expectation:

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_t^\pi[G_t | S_t = s] \\
&= \mathbb{E}_t^\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \gamma^{T-1} R_T | S_t = s] \\
&= \mathbb{E}_t^\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}_t^\pi[R_{t+1} + \gamma \mathbb{E}_{t+1}^\pi[G_{t+1} | S_{t+1} = s'] | S_t = s] \\
&= \mathbb{E}_t^\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s', r} \mathbb{P}(s', r | s, a)[r + \gamma V_\pi(s')] \qquad (1.2.1)
\end{aligned}$$

Analogously, we can rewrite the action-value function in the same way:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_t^\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')] \end{aligned} \tag{1.2.2}$$

If the MDP dynamics are known (i.e. transition probabilities and reward matrix), then the Bellman expectation equation is a linear equation, it can be solved explicitly with some basic linear algebra. However, if the state space is large, this may cause matrix manipulation to be computationally inefficient. We can use iterative algorithms which we will explore later on to approximately solve for the value-functions. Solving the Bellman Expectation equations informs one of the actual value functions under a chosen policy π. Solving these equations do not tell us the optimal policy, it just allows us to find the values of states under a certain policy. We can use policy iterative methods to try and find the optimal policy. Another method is using value iterative methods based on the Bellman Optimality equations which we will discuss below.

1.2.2 Bellman Optimality Equation

The purpose of solving the RL problem is to find a policy that returns a large amount of rewards in the long run. A policy π is said to be better than another policy π' if it's expected return is greater for all states $s \in \mathcal{S}$, or in other words $V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in \mathcal{S}$. There always exists at least one policy that is better or equal to all other policies. This is called the *optimal policy* and may not be unique and we denote it by π^*. The optimal value-function V_* is defined as the value-function which follows the optimal policy, the same goes for the optimal action-value function:

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s) && \forall s \in \mathcal{S} \\ Q_*(s,a) &= \max_\pi Q_\pi(s,a) && \forall s \in \mathcal{S},\ \forall a \in \mathcal{A} \end{aligned}$$

The optimal action-value function $Q_*(s,a)$ is the expected returns from taking action a in state s and thereafter following the optimal policy, thus we can rewrite Q_* in terms of V_*.

$$Q_*(s,a) = \mathbb{E}_t[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a] \tag{1.2.3}$$

$$= \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*(s')] \tag{1.2.4}$$

Moreover, the optimal value function can be rewritten in terms of Q_*. This is because the optimal value-function is equal to the optimal action-value function under the constraint of undertaking the optimal action always:

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s,a) \tag{1.2.5}$$

We can now substitute equations (1.2.4) and (1.2.5) into one another to obtain both versions of the Bellman Optimality equation.

$$V_*(s) = \max_{a \in \mathcal{A}} \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*(s')] \tag{1.2.6}$$

$$Q_*(s,a) = \mathbb{P}(s', r|s, a)[r + \gamma \max_{a' \in \mathcal{A}} Q_*(s', a')] \tag{1.2.7}$$

Unlike the Bellman Expectation equations, the optimality equations are not linear and cannot be solved explicitly. As such iterative algrorithms have to be used. Once the algorithms are applied and the value functions are knows, obtaining the optimal policy is very simple as we just choose the action that maximizes the optimal action-value function, as follows:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if} \quad a = \text{argmax}_{a'}\, Q_*(s, a') \\ 0 & \text{otherwise} \end{cases}$$

1.3 Model-Free Reinforcement Learning

One can use dynamic programming techniques to learn an optimal strategy where the model dynamics are *known*. We will not include dynamic programming techniques in the main body of this paper and discussion of these methods can be found in the appendix. We will mainly focus on Reinforcement Learning. The whole basis of Reinforcement Learning is for an agent to learn an optimal strategy in an environment where the model dynamics are *unknown* (transition probabilities and reward function). In this section, we will be discussing different methods that do not assume a model for the environment and are able to solve for the optimal strategy, in both discrete and continuous state spaces.

1.3.1 Exploration vs Exploitation

Before we look at any model-free RL methods, we will first introduce the idea of exploration vs exploitation. Exploration is the idea of trying different actions and analyzing the outcomes. Exploitation is acting greedily and undertaking actions that are known to give out high rewards. The agent has to be careful with acting greedily as this could lead to a sub-optimal policy. An agent has to compromise *exploiting* the action values and *exploring* the environment to discover new states and actions which may in fact prove better. One analogy to explain this further is for an agent to pick the best restaurant to dine in London. The agent can pick the best restaurant to their own knowledge (*exploitation*) or the agent can pick a new randomly chosen restaurant (*exploration*) which may prove to be a better choice.

One method of choosing actions, which balances exploration and exploitation is the $\epsilon - greedy$ algorithm. With probability ϵ, the agent chooses the 'greedy' action and with probability $1-\epsilon$, the agent chooses a random action. This algorithm does improve the policy as well. The $\epsilon - greedy$ policy can be expressed as:

$$\pi'(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{if} \quad a = \text{argmax}_{a'} \, Q_*(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

Theorem 1.3.1 ($\epsilon - greedy$ Policy Improvement)**.** *For any policy π, the $\epsilon - greedy$ policy with respect to Q_π is an improvement such that $V_{\pi'}(s) \geq V_\pi(s)$ for all $s \in \mathcal{S}$*

Proof.

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a) \\ &= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a) \\ &\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/|\mathcal{A}(s)|}{1 - \epsilon} \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a) \\ &= V_\pi(s) \end{aligned}$$

As we have shown, if $Q(s, \pi'(s)) \geq V_\pi(s)$, using theorem A.2.1 in the appendix, it follows that $V_{\pi'}(s) \geq V_\pi(s)$ for all $s \in \mathcal{S}$ □

1.3.2 Q-Learning

Q-learning is probably the most famous method within the realm of reinforcement learning, developed by Christ Watkins in 1989 [1]. Q-learning is an off-policy method within the sect of temporal difference learning. Generally, the policy that is used to generate the episodes is an $\epsilon - greedy$ policy whilst a pure greedy method is used to improve the policy. Q-learning is an effective method to obtain the optimal Q-values and optimal policy in a discrete state setting with unknown model dynamics. The reason why this method is effective is because the learned action-value function Q that approximates the optimal action-value function is independent to the policy that generates

the episodes and data. The target for the update of Q-learning is essentially based on the Bellman Optimality Equation (1.2.4). Let us recall the form of the Bellman Optimality equation:

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}_t[R_{t+1} + \gamma V_*(S_{t+1})|S_t = s, A_t = a] \\ &= \mathbb{E}_t[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_*(S_{t+1}, a')|S_t = s, A_t = a] \end{aligned}$$

Q-learning essentially uses $R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_*(S_{t+1}, a')$ as a target in the update to estimate $Q_*(S_t, a)$. One major advantage here is that updates can happen on-line. The Q-learning algorithm is below:

Algorithm 1: Q-Learning Algorithm

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Initialize step size $\alpha \in (0,1]$
Initialize n as number of episodes

for $i \leftarrow 0$ **to** n **do**
  Initialize current state S
  while *S is not terminal* **do**
    Sample action A from $\epsilon - greedy$ policy based off Q
    Take action A and observe reward R and next state S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_{a \in \mathcal{A}(S')} Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  end
end
Return $Q(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Return $\pi^*(a|s) = \text{argmax}_{a' \in \mathcal{A}(s)} Q(s,a')$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

The Q-learning algorithm is relatively simple yet very effective for discrete-state RL problems. This algorithm can be used in a wide variety of environments with little amendments made. The simplest version of Q-learning stores data in a tabular form, this can be efficient for discrete state settings. However, once we work in a continuous-state setting, this method is no longer efficient as the number of states is infinite. Even in a large state setting, this algorithm is no longer feasible. One workaround in a continuous-state setting is to quantize the possible continuous values describing the environment into bins/buckets. Another method is to use function approximation to approximate the value functions.

1.4 Model-Based Reinforcement Learning

In this section, we will now discuss how the agent can slowly have some understanding of the model dynamics using real experiences. So far we have been discussing Model-Free RL, where we do not assume any model dynamics and the agent only learns the value functions and optimal policy. Just like Model-Free RL, the agent starts with unknown initial knowledge of the model dynamics. We will discuss how the agent can gradually learn and utilize the model dynamics to solve problems in both discrete and continuous state settings.

1.4.1 Tabular Model-Based RL

This particular method is used for discrete state settings. The first phase of this method is to estimate the model dynamics using real experience; we want to estimate the reward function and transition probabilities of an MDP. The second phase is to either solve for this *estimated* MDP using value iteration methods, or to use sample-based planning; producing sample episodes using the estimated MDP dynamics and then applying model-free methods such as Q-learning on both the sampled and real experience for the agent to learn. Sample-based planning can also be implemented by not estimating the model, rather storing transitions in a block of memory just like the Deep Q-learning algorithm and sampling transitions from this memory. We can count the visits to each state-action pair, and produce sample probabilities of a transition to a next state

due to this state-action pair. We could also use a sample mean to estimate the reward function for a given state-action pair. The mathematical formulation is below:

$$\hat{\mathcal{P}}\big(S_{t+1} = s' \mid S_t = s, A_t = a\big) = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}\big(S_{t+1} = s' \mid S_t = s, A_t = a\big)$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}\big(S_t = s, A_t = a\big) R_t$$

Where $N(s,a)$ is the total number of visits to the state-action pair $(S_t = s, A_t = a)$ and $\hat{\mathcal{P}}\big(s' \mid s, a\big)$ is the estimated probability transition from state s to state s' under action a whilst $\hat{\mathcal{R}}_s^a$ is the estimated immediate reward of executing action a while in state s.

As we now have an estimate for the dynamics of the model; we can implement a version of the value iteration algorithm A.2.1 to find the optimal policy. This algorithm is a dynamic programming method which is originally used when the agent has knowledge of the model dynamics. Value iteration improves the policy after each update of the value function. The update of the value iteration algorithm is based on the Bellman Optimality equation (1.2.6) and can be found in the appendix A.2.1.

Frozen Gridworld

We will implement the tabular model-based RL method on the Frozen Gridworld environment. We will be performing value iteration while learning the model dynamics. The Frozen Gridworld environment is a 4x4 grid of tiles where the agent always starts on the upper left corner and can move up, down, left, and right. Some of the tiles have "holes" which causes the agent to fall through and terminates the episode. All other tiles are "slippery" and so undertaking a certain action does not guarantee a transition to a certain state; this environment is thus *stochastic*. The goal of the agent is to reach the bottom left corner at which point the episode terminates and the agent receives a reward of +1; the agent receives no reward elsewhere.

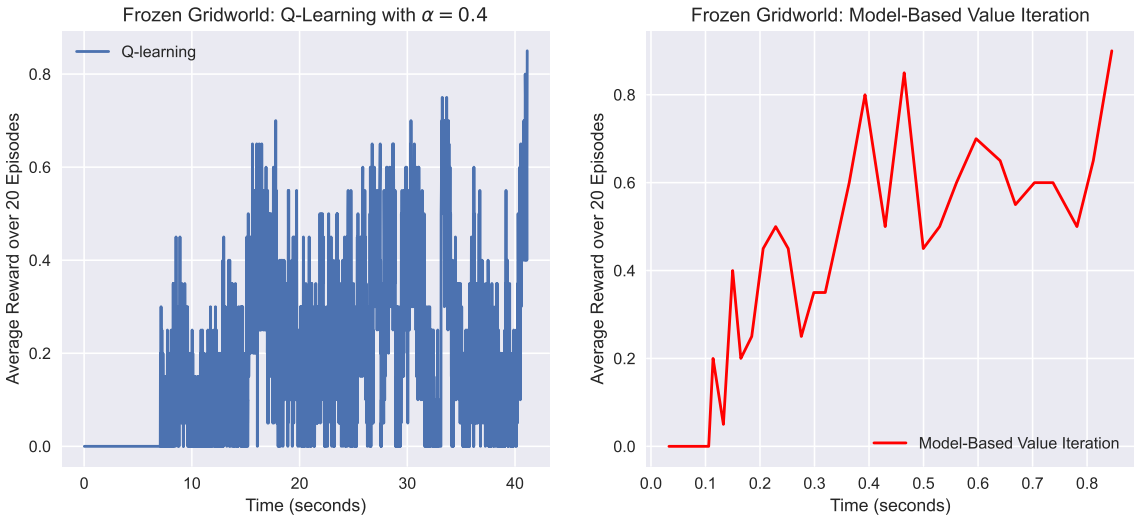


Figure 1.1: Average Total Reward over 20 episodes for Frozen Gridworld

The left plot of Figure 1.1 shows the training of the agent using Q-learning. At each update, we played 20 episodes and averaged the total reward received by each episode. One can see that it took more than 5 seconds for at least one episode to terminate at the goal state. As the policy is stochastic and we cannot guarantee a reward of +1 every episode, we set the threshold to stop training at 0.8 for Q-learning. This took roughly 40 seconds to train the agent with 5655 iterations which is relatively slow. This is because rewards are given out rarely and so learning is slow with Q-learning. On the other hand, learning is very quick with Model-Based

Value iteration. We implemented this by interchanging the value function iteration and model learning; at each iteration, we played 100 steps and used these steps to update the probability matrix and reward function. A threshold of 0.85 was achieved within a second and required only 74 iterations. Using real experience to build a model can be advantageous and speed up training time. In environments where rewards are emitted infrequently, it is best to first try and learn the dynamics of the environment before trying to learn the optimal policy.

1.4.2 Kernel-Based Reinforcement Learning

The tabular model-based RL methods do not work in a continuous state setting. We require a different method to facilitate continuous state settings. One way is to use kernel-based methods to approximate a finite state MDP using historical outcomes. The way this works essentially is that the kernel-based methods approximate the consequences of an action a from a given state s as the weighted average of previous results of that action. The weightings are computed using a continuous kernel function of the distance metrics between state s and previous states. Let us assume we have access to historical transitions (s_i, a_i, r_i, s_i') for $i = 1 \dots N$. Where s_i is the current state, a_i is the executed action, r_i is the immediate reward and s_i' is the next state. The Bellman Optimality equation can be approximated using kernel-based methods in the following way:

$$\hat{Q}(s,a) = \frac{1}{Z_{s,a}} \sum_{i|a_i=a} \phi\left(\frac{d(s_i,s)}{b}\right)\left[r_i + \gamma \max_{a'} \hat{Q}(s_i', a')\right] \tag{1.4.1}$$

[12]

Where $Z_{s,a} = \sum_{i|a_i=a} \phi\left(\frac{d(s_i,s)}{b}\right)$ is used to normalize the weights and ϕ is a non-negative kernel function used to compute the weights depending on the relative distance of the states. Typically, one uses Gaussian kernels ($\phi(x) \propto e^{-x^2}$) to compute the weights. The parameter b is a smoothing parameter. Equation (1.4.1) can be solved using value iteration.

Chapter 2

Discrete-Time hedging and QLBS

In this chapter, we will now formulate the classical Black-Scholes model in a discrete-time setting. In particular, we will be looking at the point of view of a seller of a vanilla European option (e.g. Call Option) with maturity T and terminal payoff $G_T(S_T)$ which is dependent on the underlying stock price at maturity. The seller of the option will setup a replicating portfolio Π_t consisting of some position u_t of the stock S_t and risk-free bond B_t at time t. As discussed previously in the introduction, the goal of hedging and pricing in this setting is to reduce mis-hedging risk in a sequential manner.

2.1 Hedge Portfolio

The purpose of the replicating portfolio is to try and match the value of the option at all times by re-balancing the wealth between the bonds and stocks. Exact replication is achievable in the classical BS model where continuous re-balancing is hypothetically possible. In the setting of re-balancing the portfolio at discrete-time steps; exact replication is no longer possible at all times. At maturity T, the option is exercised and the position of the hedge u_T is also closed, i.e. the stocks are bought/sold and converted to cash. This is constrained by setting $u_T = 0$. Thus, at time T, the hedge portfolio only consists of risk-free bonds. The portfolio value at maturity is:

$$\Pi_T = B_T = G_T(S_T) \tag{2.1.1}$$

The portfolio value at times $t < T$ is given by:

$$\Pi_t = u_t S_t + B_t \tag{2.1.2}$$

Moreover, using the self-financing constraint, we can find out the amount of bonds required at times $t < T$. The self-financing constraint means that all changes in the hedge position should be funded by an initial bond. Wealth can only be balanced between the stock and bond; there are no external cash flows in or out of the portfolio during the lifetime of the option. One can then formulate the following recursive equation by equating the value of the portfolio at time $t+1$ just before and after re-balancing:

$$\Pi_{t+1} = u_t S_{t+1} + e^{r\Delta t} B_t = u_{t+1} S_{t+1} + B_{t+1} \tag{2.1.3}$$

Where Δt is the difference between time $t+1$ and time t. The middle section of equation (2.1.3) is the portfolio value Π_{t+1} expressed in terms of the previous hedge position and bond with accrued interest. Due to no cash flows in or out of the portfolio, this must be equal to the value of the portfolio after re-balancing which is expressed as the right-hand side of equation (2.1.3). We can rearrange this equation for B_t to produce a recursive equation for the bond:

$$B_t = e^{-r\Delta t}[B_{t+1} + (u_{t+1} - u_t)S_{t+1}], \qquad t = T-1, \ldots, 0 \tag{2.1.4}$$

We can plug equation (2.1.4) into equation (2.1.2) to produce a recursive equation for the portfolio value Π_t. The substitution is as follows:

$$\begin{aligned}\Pi_t &= u_t S_t + B_t \\ &= u_t S_t + e^{-r\Delta t}[B_{t+1} + (u_{t+1} - u_t)S_{t+1}] \\ &= u_t S_t + e^{-r\Delta t}[\Pi_{t+1} - u_t S_{t+1}] \\ &= e^{-r\Delta t}[\Pi_{t+1} - u_t \Delta S_t] \end{aligned} \tag{2.1.5}$$

Where $\Delta S_t = S_{t+1} - e^{-r\Delta t} S_t$

From the recursive equations for Π_t and B_t, one can see that these variables are not $\mathcal{F}_t$ measurable where $\mathcal{F}_t$ is the information up to time t. This is because both B_t and Π_t depend on future values of bond and portfolio value, respectively. As such, the initial values B_0 and Π_0 are random variables with some distributions. We can use Monte Carlo simulations to estimate these distributions for any given hedging strategy $(u_t)_{t=0}^T$. This is achieved by simulating N stock price paths $(S_t^i)_{t=0}^T$ under the Black-Scholes model, where the super-script i denotes the ith path. Under the Black-Scholes model, one can produce a stock price path using the following recursive formulation:

$$S_{t+1} = S_t \exp\left((r - \frac{\sigma^2}{2})\Delta t + \sigma \Delta t Z_t\right)$$

Where r is the risk-free interest rate, σ is the volatility of the underlying stock and Z_t is an independent standard normal random variable. Once all the stock price paths have been simulated, one can compute the portfolio value $\Pi_T(S_T)$ at maturity T for all N paths. We can then evaluate Π_t for $t < T$ using the recursive equation (2.1.5) and our trading strategy $(u_t)_{t=0}^T$. Under the Black-Scholes model, we assume that the sellers trading of the stock has no impact on the stock price and so the stock price evolution $(S_t^i)_{t=0}^T$ is independent of the trading strategy. So, we only need to produce the sample paths once and we can reuse these paths for a variety of trading strategies $(u_t)_{t=0}^T$. In this paper, we use synthetic simulated data, however, it is possible to use real historical data for the underlying stock prices.

One of the objectives of the seller is to price the option today ($t = 0$). The fair price of the option can be the mean across all price paths of Π_0, the dealer can also add a premium to be compensated for taking on risk. Pricing can only happen once the seller has a trading strategy. In the next section, we will discuss how the seller can choose an optimal trading strategy.

2.2 Optimal Trading Strategy

To compute the optimal hedging strategy $u_t^*(S_t)$, we need to utilise all MC paths at time t, this is essentially a cross-sectional analysis of the simulated paths. We used a pathwise backward recursion to compute the portfolio value. We still use a backward recursive formula to compute the optimal hedge, but we use *every* MC path available as well, starting from $t = T$ to $t = 0$. Though, like most trading strategies, our strategy must be adapted and not predictive; it can only depend on information $\mathcal{F}_t$ up to time t. We cannot use future information to decide how we hedge today. Hence, any computation for the optimal hedge must be conditioned with the information $\mathcal{F}_t$ at the current time t. In this setting, we can compute the optimal hedge at time t by minimising the variance of the portfolio value Π_t across all MC paths conditioned on the current cross-sectional information $\mathcal{F}_t$:

$$\begin{aligned} u_t^*(S_t) &= \underset{u}{\operatorname{argmin}}\, Var[\Pi_t \mid \mathcal{F}_t] \\ &= \underset{u}{\operatorname{argmin}}\, Var[\Pi_{t+1} - u_t \Delta S_t \mid \mathcal{F}_t] \\ &= \underset{u}{\operatorname{argmin}} \left[Var_t[\Pi_{t+1}] - 2u_t Cov_t[\Pi_{t+1}, \Delta S_t] + u_t^2 Var_t[\Delta S_t] \right] \end{aligned} \tag{2.2.1}$$

We can minimize equation (2.2.1) by setting the derivative with respect to u to zero and by doing so, we have an analytical solution to the optimal hedge:

$$0 = -2Cov_t[\Pi_{t+1}, \Delta S_t] + 2u_t^* Var_t[\Delta S_t]$$

Rearranging this equation for u_t^* gives us the optimal hedge:

$$u_t^*(S_t) = \frac{Cov[\Pi_{t+1}, \Delta S_t \mid \mathcal{F}_t]}{Var[\Delta S_t \mid \mathcal{F}_t]} \tag{2.2.2}$$

This expression can be computed in a variety of ways depending if the state space is discrete or continuous. In a discrete state setting, one can use the transitions probabilities and finite sums to compute the expectations of variables at time $t+1$ conditioned on information at time t. If we are in a continuous state setting, we can use Kernel-Regression, in particular the Nadraya-Watson estimator to compute the expectations conditioned on the stock price S_t. The expectations can also be computed using basis functions. (Note from now on we will denote an expectation conditioned on information $\mathcal{F}_t$) at time t, as $\mathbb{E}_t[\cdot]$ rather then $\mathbb{E}[\cdot \mid \mathcal{F}_t]$ for brevity).

2.3 Option Pricing in a Discrete-Time Setting

The fair price of an option $\hat{C}_t$ at time t is defined as the expected value of the hedge portfolio Π_t:

$$\hat{C}_t = \mathbb{E}_t[\Pi_t] \tag{2.3.1}$$

Using the tower property of expectation and equation (2.1.5), we can express the fair option price $\hat{C}_t$ in a recursive form:

$$\begin{aligned}
\hat{C}_t &= \mathbb{E}_t[\Pi_t] = \mathbb{E}_t[e^{-r\Delta t}\Pi_{t+1} - u_t \Delta S_t] \\
&= \mathbb{E}_t\big[\mathbb{E}_{t+1}[e^{-r\Delta t}\Pi_{t+1} - u_t \Delta S_t]\big] && \text{(using Tower Property)} \\
&= \mathbb{E}_t\big[e^{-r\Delta t}\mathbb{E}_{t+1}[\Pi_{t+1}]\big] - u_t \mathbb{E}_t[\Delta S_t] && (u_t \text{ is } \mathcal{F}_t \text{ measurable}) \\
&= \mathbb{E}_t[e^{-r\Delta t}\hat{C}_{t+1}] - u_t \mathbb{E}_t[\Delta S_t]
\end{aligned} \tag{2.3.2}$$

We can use the tower property to express the optimal hedge $u_t^*(S_t)$ in terms of $\hat{C}_{t+1}$ rather than Π_{t+1}:

$$u_t^*(S_t) = \frac{Cov[\hat{C}_{t+1}, \Delta S_t \mid \mathcal{F}_t]}{Var[\Delta S_t \mid \mathcal{F}_t]} \tag{2.3.3}$$

Let us now substitute equation (2.3.3) into equation (2.3.2) and use a change of measure to obtain a recursive formula for the fair price $\hat{C}_t$:

$$\begin{aligned}
\hat{C}_t &= \mathbb{E}_t[e^{-r\Delta t}\hat{C}_{t+1}] - u_t \mathbb{E}_t[\Delta S_t] \\
&= \mathbb{E}_t[e^{-r\Delta t}\hat{C}_{t+1}] - \left[\frac{\mathbb{E}_t[\hat{C}_{t+1}\Delta S_t] - \mathbb{E}_t[\hat{C}_{t+1}]\mathbb{E}_t[\Delta S_t]}{Var[\Delta S_t \mid \mathcal{F}_t]}\right]\mathbb{E}_t[\Delta S_t] \\
&= \mathbb{E}_t[e^{-r\Delta t}\hat{C}_{t+1}] - \left[\frac{\mathbb{E}_t\big[\hat{C}_{t+1}\Delta S_t \mathbb{E}_t[\Delta S_t]\big] - \mathbb{E}_t\big[\hat{C}_{t+1}(\mathbb{E}_t[\Delta S)^2\big]}{Var[\Delta S_t \mid \mathcal{F}_t]}\right] \\
&= \mathbb{E}_t\left[e^{-r\Delta t}\hat{C}_{t+1} - \frac{\hat{C}_{t+1}\Delta S_t \mathbb{E}_t[\Delta S_t] - \hat{C}_{t+1}\big(\mathbb{E}_t[\Delta S_t]\big)^2}{Var[\Delta S_t \mid \mathcal{F}_t]}\right] \\
&= \mathbb{E}_t\left[e^{-r\Delta t}\hat{C}_{t+1}\left(1 - e^{r\Delta t}\frac{(\Delta S_t - \mathbb{E}_t[\Delta S_t])\mathbb{E}_t[\Delta S_t]}{Var[\Delta S_t \mid \mathcal{F}_t]}\right)\right]
\end{aligned}$$

$$= \mathbb{E}_t\left[e^{-r\Delta t}\hat{C}_{t+1}\frac{d\hat{\mathbb{Q}}}{d\mathbb{P}}\right]$$
$$= \mathbb{E}^{\hat{\mathbb{Q}}}\Big[e^{-r\Delta t}\hat{C}_{t+1} \mid \mathcal{F}_t\Big]$$

Where $\mathbb{P}$ is the physical probability measure and $\hat{\mathbb{Q}}$ is a signed measure. We have used a change of measure with the transition probabilities as [4, pg 8 eqn, 12]:

$$\hat{q}(S_{t+1} \mid S_t) = p(S_{t+1} \mid S_t)\left(1 - e^{r\Delta t}\frac{\big(\Delta S_t - \mathbb{E}_t[\Delta S_t]\big)\mathbb{E}_t[\Delta S_t]}{Var[\Delta S_t \mid \mathcal{F}_t]}\right)$$

Where $p(S_{t+1} \mid S_t)$ is the transition probability under the physical measure $\mathbb{P}$. $\hat{\mathbb{Q}}$ is *not* a legitimate probability measure as for large movements of ΔS_t, the value for $\hat{q}(S_{t+1} \mid S_t)$ can become negative. The fair option price $\hat{C}_t$ can potentially be negative but this is *not* the price the seller charges. In actual fact, the seller charges a risk-adjusted price which depends on the sellers risk preferences by a risk aversion term λ. So, by choosing an appropriate risk aversion term, the price of the option can be forced to be non-negative.

The option seller has to charge a risk premium to compensate for the risk of depleting the bank account at some time in the future B_t. If the bank account is depleted, then the seller would require external cash injections into the portfolio hence nullifying the self-financing constraint. One possible method to introduce a risk premium is to have an additional term of the cumulative expected discounted variance of the portfolio value at each discrete time step $t = 0, \ldots, T$ with a risk-aversion parameter λ. This was first suggested by Potters and Bouchaud.[13]

$$C_0(s,u) = \mathbb{E}_0\left[\Pi_0 + \lambda\sum_{t=0}^{T} e^{-rt}Var[\Pi_t \mid \mathcal{F}_t] \;\middle|\; S_0 = s, u_0 = u\right]$$

The objective of the seller is to minimize this expression by choosing an appropriate adapted trading strategy $u_t^*(S_t)$. This is equivalent to maximizing the following expression which is defined as the state-value function:

$$V(S_t) = \mathbb{E}_t\left[-\Pi_t - \lambda\sum_{t'=t}^{T} e^{-r(t'-t)}Var[\Pi_{t'} \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t\right] \tag{2.3.4}$$

2.4 Hedging and Pricing in the Continuous BS limit

In our framework, one can show that we obtain the classical continuous Black-Scholes solutions for the hedge and price when we take the limit $\Delta t \to 0$ in our model. Let us recall that in the continuous BS-model, the underlying stock price follows a Geometric Brownian motion with drift μ and volatility σ:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

Where W_t is a standard Brownian Motion.

Let us first show that our optimal hedge $u_t^*(S_t)$ tends to the the Black-Scholes Delta $\frac{\partial C_t}{\partial S_t}$ under the limit $\Delta t \to 0$. Let us express the option price C_{t+1} using a first-order Taylor expansion:

$$C_{t+1} = C_t + \frac{\partial C_t}{\partial S_t}\Delta S_t + \mathcal{O}(\Delta t) \tag{2.4.1}$$

Let us substitute equation (2.4.1) into equation (2.3.3) and take the limit $\Delta t \to 0$:

$$\begin{aligned}
\lim_{\Delta t\to 0} u_t^*(S_t) &= \lim_{\Delta t\to 0} \frac{Cov[C_{t+1}, \Delta S_t \mid \mathcal{F}_t]}{Var[\Delta S_t \mid \mathcal{F}_t]} \\
&= \lim_{\Delta t\to 0} \frac{Cov[C_t + \frac{\partial C_t}{\partial S_t}\Delta S_t + \mathcal{O}(\Delta t), \Delta S_t \mid \mathcal{F}_t]}{Var[\Delta S_t \mid \mathcal{F}_t]} \\
&= \frac{\partial C_t}{\partial S_t} \frac{Cov[\Delta S_t, \Delta S_t \mid \mathcal{F}_t]}{Var[\Delta S_t \mid \mathcal{F}_t]} \\
&= \frac{\partial C_t}{\partial S_t}
\end{aligned}$$

We will utilise equation (2.3.2) to find the continuous-time limit of the option price. Let us first compute the limit of $u_t \mathbb{E}_t[\Delta S_t]$.

$$\begin{aligned}
\lim_{\Delta t\to 0} u_t \mathbb{E}_t[\Delta S_t] &= \lim_{\Delta t\to 0} u_t(\mathbb{E}_t[S_{t+1}] - e^{r\Delta t} S_t) = \lim_{\Delta t\to 0} u_t(e^{\mu\Delta t} S_t - e^{r\Delta t} S_t) \\
&= \lim_{dt\to 0} (\mu - r) S_t \frac{\partial C_t}{\partial S_t} dt
\end{aligned} \tag{2.4.2}$$

Let us use a second order Taylor expansion to evaluate C_{t+1}:

$$\begin{aligned}
C_{t+1} &= C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t} dS_t + \frac{1}{2}\frac{\partial^2 C_t}{\partial S_t^2}(dS_t)^2 + \dots \\
&= C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t}(\mu S_t dt + \sigma S_t dW_t) + \frac{1}{2}\frac{\partial^2 C_t}{\partial S_t^2}(\sigma^2 S_t^2 dt) + \mathcal{O}(dt^2)
\end{aligned} \tag{2.4.3}$$

Let us now substitute equation (2.4.3) and (2.4.2) into (2.3.2) and recall that $\mathbb{E}_t[dW_t] = 0$ and take the limit $\Delta t \to 0$:

$$\begin{aligned}
\lim_{\Delta t\to 0} C_t &= \lim_{\Delta t\to 0} \mathbb{E}_t[e^{-r\Delta t} C_{t+1}] - u_t \mathbb{E}_t[\Delta S_t] \\
&= \lim_{dt\to 0} (1 - r dt)\left[C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t}(\mu S_t dt) + \frac{1}{2}\frac{\partial^2 C_t}{\partial S_t^2}(\sigma^2 S_t^2 dt)\right] - (\mu - r)\frac{\partial C_t}{\partial S_t} S_t dt \\
&= \lim_{dt\to 0}\left[C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t}(r S_t dt) + \frac{1}{2}\frac{\partial^2 C_t}{\partial S_t^2}(\sigma^2 S_t^2 dt) - r C_t\right]
\end{aligned}$$

If we rearrange this expression and ignore terms in order of dt^2, we obtain the famous Black-Scholes PDE in the limit $dt \to 0$:

$$\frac{\partial C_t}{\partial t} + r S_t \frac{\partial C_t}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 C_t}{\partial S_t^2} - r C_t = 0$$

Thus, we have shown that in our discrete-time framework, we can obtain the classical formulae of the continuous BS-model in the limit of $\Delta t \to 0$. Note, the formulae for the price and delta can be found in the appendix B.1.

2.5 Q-Learning: Black-Scholes (QLBS)

We will now reformulate the hedging and pricing methods presented in the previous section as a Markov Decision Process (MDP) and explore methods in both a discrete and continuous state setting. We are still in a framework of dynamically rebalancing a hedge portfolio at discrete time steps to replicate a vanilla European option. We will view this problem as a sequential decision process, where we try to maximize the "rewards" received which in this case is the negative of the variances of the portfolio value Π_t conditioned at time t multiplied by a risk aversion term λ. This is achieved by finding the optimal hedge at each re-balancing step dependent on the current stock price S_t.

If the transition probabilities and reward function are *known*, then one can use value iteration to solve the Bellman Optimality equation. In fact, in our case of QLBS, this equation can be solved *analytically*, as it is a simple quadratic optimization problem. However, if we do not have information pertaining to the transition probabilities or reward function; we could instead solve the optimality equation using sampled data using RL algorithms.

We will first solve the problem in a discrete state setting by aggregating the stock price into fixed-length bins and use efficient RL algorithms such as Q-learning or Model-Based Value iteration and compare the efficiency of different algorithms. We will then solve this in a continuous-state setting initially using basis functions and later on a feed-forward neural network. The last method we will evaluate is Kernel Estimation. For all methods, we will try and show convergence to the classical BSM model for hedging and pricing in the limit $\Delta t \to 0$.

2.6 QLBS: Setup

2.6.1 Change of Variable

We will also be using a variable X_t which is a transformation of the stock price such that this is a stochastic variable with zero drift. X_t can be expressed in terms of S_t and vice versa in the following way:

$$X_t = -\left(\mu - \frac{\sigma^2}{2}\right)t + \log(S_t)$$

$$S_t = \exp\left(X_t + \left(\mu - \frac{\sigma^2}{2}\right)t\right)$$

Thus the differential equation for X_t is the following:

$$dX_t = -\left(\mu - \frac{\sigma^2}{2}\right)dt + \frac{1}{S_t}dS_t = \sigma dW_t$$

Where W_t is a standard Brownian motion. This implies that X_t is a Brownian motion with volatility σ. As such, X_t has no drift and is homogeneous with time and is stationary, unlike the stock price which has some non-zero drift. Furthermore, as X_t is a martingale (under a Black-Scholes setting), throughout the lifetime of the option, the variable should not differ on average far from the initial value X_0. Throughout this section, we will use the stock price S_t and X_t interchangeably for different methods.

2.6.2 Bellman Equations

Let us now formulate the Bellman Equations in our discrete-time Black-Scholes setting using sequential risk minimization as stated in the previous chapter. We will now express the stock price as the time-uniform variable X_t and $a_t = a_t(X_t)$ is the hedge as a function of X_t whilst $u_t = u_t(S_t)$ is the hedge position as a function of S_t. We will also introduce the notion of a deterministic policy $\pi(t, X_t)$:

$$\pi : \{0, \dots, T-1\} \times \mathcal{X} \to \mathcal{A} \tag{2.6.1}$$

Where π is a policy which maps the time t and state $X_t \in \mathcal{X}$ to action $a_t \in \mathcal{A}$

As for our value function, we will utilise equation (2.3.4) in terms of our new variable X_t, following policy π:

$$\begin{aligned} V_t^\pi(X_t) &= \mathbb{E}_t\left[-\Pi_t(X_t) - \lambda \sum_{t'=t}^{T} e^{-r(t'-t)} Var[\Pi_{t'}(X_t) \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t \right] \\ &= \mathbb{E}_t\left[-\Pi_t(X_t) - \lambda Var\big[\Pi_t(X_t)\big] - \lambda \sum_{t'=t+1}^{T} e^{-r(t'-t)} Var[\Pi_{t'}(X_t) \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t \right] \end{aligned} \tag{2.6.2}$$

The summation in equation (2.6.2) can be expressed in terms of V_{t+1} and Π_{t+1} using the expression for the value function at time $t+1$ in the following way:

$$\begin{aligned} &\mathbb{E}_{t+1}\left[-\Pi_t(X_{t+1}) - \lambda \sum_{t'=t}^{T} e^{-r(t'-(t+1))} Var[\Pi_{t'}(X_{t'}) \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t \right] = V_{t+1}^\pi(X_{t+1}) \\ &\mathbb{E}_{t+1}\left[-\lambda \sum_{t'=t}^{T} e^{-r(t'-(t+1))} Var[\Pi_{t'}(X_{t'}) \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t \right] = V_{t+1}^\pi(X_{t+1}) + \mathbb{E}_{t+1}[\Pi_{t+1}(X_{t+1})] \\ &\mathbb{E}_{t+1}\left[-\lambda \sum_{t'=t}^{T} e^{-r(t'-t)} Var[\Pi_{t'}(X_{t'}) \mid \mathcal{F}_{t'}] \;\middle|\; \mathcal{F}_t \right] = \gamma\Big(V_{t+1}^\pi(X_{t+1}) + \mathbb{E}_{t+1}[\Pi_{t+1}(X_{t+1})] \Big) \end{aligned} \tag{2.6.3}$$

Let us now substitute equation (2.6.3) into equation (2.6.2) so we can obtain the Bellman equation in a Black-Scholes setting:

$$\begin{aligned} V_t^\pi(X_t) &= \mathbb{E}_t\left[-\Pi_t(X_t) - \lambda Var\big[\Pi_t(X_t)\big] + \gamma\Big(V_{t+1}^\pi(X_{t+1}) + \Pi_{t+1}(X_{t+1}) \Big) \;\middle|\; \mathcal{F}_t \right] \\ &= \mathbb{E}_t\left[a_t(X_t)\Delta S_t(X_t, X_{t+1}) - \lambda Var\big[\Pi_t(X_t)\big] + \gamma V_{t+1}^\pi(X_{t+1}) \;\middle|\; \mathcal{F}_t \right] \\ &= \mathbb{E}_t\left[R(X_t, a_t, X_{t+1}) + \gamma V_{t+1}^\pi(X_{t+1}) \;\middle|\; \mathcal{F}_t \right] \end{aligned}$$

Where the random reward can be fully expressed as:

$$\begin{aligned} R(X_t, a_t, X_{t+1}) &= a_t \Delta S_t - \lambda Var\big[\Pi_t(X_t)\big] \\ &= a_t \Delta S_t - \lambda\gamma^2 \mathbb{E}_t\Big[\hat{\Pi}_{t+1}^2 - 2a_t \Delta\hat{S}_t \hat{\Pi}_{t+1} + (a_t \Delta\hat{S}_t)^2 \Big] \end{aligned} \tag{2.6.4}$$

From equation (2.6.4), one can infer that the *expected* reward is a quadratic function of the variable a_t. This is a crucial result as it would allow for an analytical solution for the optimal hedge. One should also note that in the limit of $\lambda \to 0$, the expected reward is linear in a_t and so a maximum expected reward does not exist which can cause problems in terms of finding an optimal hedge.

The action-value function can be expressed in a similar way to the state value function, the only difference is that the action-value function is conditioned on both the current state $X_t = x$ and action taken $a_t = a$ just as usual.

$$Q_t^{\pi}(x,a) = \mathbb{E}_t\left[-\Pi_t(X_t) - \lambda \sum_{t'=t}^{T} e^{-r(t'-t)} Var[\Pi_{t'}(X_{t'}) \;\middle|\; X_t = x, a_t = a \right] \qquad t = 0, \dots, T-1$$

As usual, the optimal policy π^*, is the policy that maximises the value function for all states $X_t \in \mathcal{X}$. Hence, it can be expressed as an argmax:

$$\pi_t^*(X_t) = \underset{a_t \in \mathcal{A}}{\operatorname{argmax}}\, Q_t^*(X_t, a_t)$$

Let us recall the Bellman Optimality equation, and express it in terms of state variable X_t:

$$Q_t^*(x,a) = \mathbb{E}_t\left[R_t(X_t, a_t, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) \;\middle|\; X_t = x, a_t = a \right] \tag{2.6.5}$$

The terminal condition for the action-value at time $t = T$ with constraint $a_T = 0$ is:

$$Q_T^*(X_T, a_T = 0) = -\Pi_T(X_T) - \lambda Var[\Pi_T(X_T)]$$

Where $\Pi_T(X_T)$ is the terminal payoff of the option and the variance is with respect to all MC paths.

2.6.3 Optimal Trading Strategy

We mentioned earlier that the expected reward is a quadratic function in a_t, which leads to a simple optimization problem that has an analytical solution under certain assumptions. Let us substitute the equation for the reward (2.6.4) into the Bellman Optimality equation (2.6.5) to obtain:

$$Q_t^*(X_t, a_t) = \gamma \mathbb{E}_t\left[Q_{t+1}^*(X_{t+1}, a_{t+1}^*) + a_t \Delta S_t - \lambda\gamma\big(\hat{\Pi}_{t+1}^2 - 2a_t \Delta\hat{S}_t \hat{\Pi}_{t+1} + (a_t \Delta\hat{S}_t)^2\big) \right] \tag{2.6.6}$$

The above expression can be quadratic in a_t under certain assumptions. This may not be quadratic as the first term $Q_{t+1}^*(X_{t+1}, a_{t+1}^*)$ could depend on the action a_t. However, this term can only depend on the action a_t through the transition probability $\mathbb{P}(X_{t+1} \mid X_t, a_t)$ but we know from our assumptions in the Black-Scholes model that the seller's actions do not affect future states of the underlying asset of the option, i.e. no market impact. Under this assumption, the expression is indeed a quadratic which we can maximize and so the optimal hedge at time t is :

$$\underset{a_t \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(X_t, a_t) = a_t^*(X_t) = \frac{\mathbb{E}_t\left[\Delta\hat{S}_t \hat{\Pi}_{t+1} + \frac{1}{2\gamma\lambda}\Delta S_t\right]}{\mathbb{E}_t\left[\Delta\big(\hat{S}_t\big)^2\right]} \tag{2.6.7}$$

We can show using a Taylor expansion and under the right limits; that this term tends to the continuous Black-Scholes Delta. Let us Taylor expand under the limit $\Delta t \to 0$:

$$\lim_{\Delta t \to 0} a_t^* = \frac{\partial C_t}{\partial S_t} + \frac{\mu - r}{2\lambda\sigma^2 S_t} \tag{2.6.8}$$

Once we have taken the limit $\Delta t \to 0$, we can cause equation (2.6.8) to tend to the Black-Scholes Delta in two different ways. The first way is to simply set $\mu = r$, so we are working in the risk-neutral measure rather than the physical measure and the drift of the stock price is the same as the risk-free rate. The second method is to take the limit $\lambda \to \infty$. Once we execute either of these two, we obtain the Black-Scholes Delta and the expression becomes identical to equation (2.2.2). These two methods can be interpreted in a similar way.

In the previous chapter, we obtained the optimal hedge (2.2.2) by only considering the risk of the hedge portfolio whilst in this chapter we now include a drift term $\mathbb{E}_t[\Pi_t]$ in the value function (2.6.2) in a fashion similar to that of Markowitz risk-reward analysis [14]. The inclusion of this drift term leads to the linear term $\gamma a_t \Delta S_t$ within the reward function. By taking the limit $\lambda \to \infty$ suggests a purely risk-centered hedging strategy that has the same effect as setting the drift of the stock μ to the risk-free rate r.

Both optimal hedge and fair price of the previous chapter can be obtained from equations (2.6.7) and (2.6.6) respectively, if we first set the drift of the underlying asset to the risk-free rate ($\mu = r$) in (2.6.7), substitute this into (2.6.6) and take the limit $\lambda \to 0$. Once this is achieved, we can then take the limit $\Delta t \to 0$ to obtain the continuous Black-Scholes Hedge and Price for the option. The order of setting $\mu = r$ and subsequently taking the limit $\lambda \to 0$ is crucial to obtain the correct expression and is consistent with the concept of hedging the option before pricing.

Let us now plug equation (2.6.7) into equation (2.6.6) which results in a compact and recursive version of the Bellman Optimality equation:

$$Q_t^*(X_t, a_t^*) = \gamma \mathbb{E}_t \left[Q_{t+1}^*(X_{t+1}, a_{t+1}^*) + \left(\lambda\gamma \hat{\Pi}_{t+1}^2 - \left(a_t^*(X_t) \Delta \hat{S}_t \right)^2 \right) \right] \qquad t = 0, \ldots, T-1$$

Where a_t^* is the optimal hedge at time t as defined by (2.6.7). Taking the limit $\lambda \to 0$ will *not* lead to a risk-free limit as the order of first setting $\mu = r$ is not obeyed.

To summarize, we can hedge and price in a backward recursive manner from $t = T-1$ to $t = 0$ by maximizing the value function at each time step, which in the QLBS is simply a quadratic optimization problem with the analytical solution (2.6.7). Note, however, if the problem is more complex, then the optimization may no longer be simple enough to obtain an analytical solution and numerical implementations such as a neural network may be required to achieve an optimal hedge. Examples of introducing complexity are by relaxing assumptions such as introducing transaction costs, initial margin cost, liquidity costs, etc. The final result of the backward recursion is the action-value function at time 0, which by definition is the negative of the option price. Thus, the risk-adjusted option price is:

$$C_0(S_t) = -Q_0^*(X_t, a_t^*) \tag{2.6.9}$$

In the next few sections, we will discuss how we can use Monte Carlo sample paths of the stock to estimate the optimal hedge and price under our Discrete Black-Scholes model. We will achieve this in a variety of ways ranging from Kernel-Estimation to Q-learning in a discrete state setting.

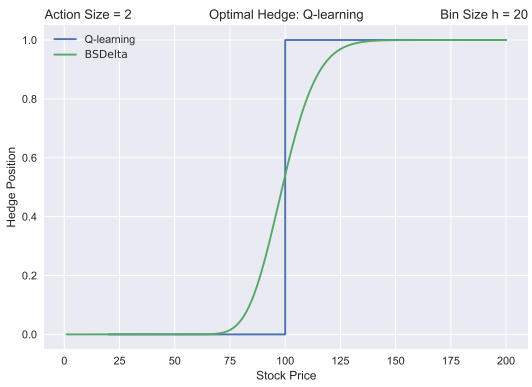
2.7 Discrete State Aggregation with Discrete Actions

We will now implement some of our discrete setting RL algorithms from chapter 1 in a Black-Scholes framework to compute the optimal hedge from a finite discrete set of actions. We will assume the agent is a seller of a European Call Option and has to hedge this option with a replicating portfolio. The agent has to learn how to hedge using sampled data in a situation where the agent has no knowledge of the transition probabilities and reward function. We will use a modified version of (2.6.4) to accommodate for the fact that we are now working in a discrete state setting rather than a continuous setting.

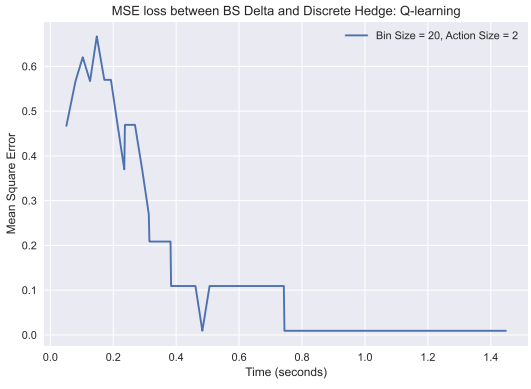
The way we achieve discrete states is to group intervals of the continuous stock price into bins of fixed size h. Each bin represents the new version of a state which we will call M_t. For example, for a bin size of 5, the interval $[0, 5)$ is state 0 and the interval $[5, 10)$ is state 1, and state n is represented by the interval $[nh, (n+1)h)$ where $h = 5$ in this case. As mentioned earlier, the reward function is slightly amended to make sure that the reward is constant for a given state transition and action. This is achieved by setting the first term of (2.6.4) to be $\Delta\tilde{M}_t = \tilde{M}_{t+1} - e^{r\Delta t}\tilde{M}_t$, where $\tilde{M}_t$ represents the midpoint of the bin of state M_t. The second term is a conditional expectation which is computed using kernel estimation using M_t as the conditioned variable. As expected, a smaller bin size and larger set of actions lead to better convergence but computation time is increased.

2.7.1 Q-Learning

The first discrete RL algorithm we will analyze for our problem is the famous Q-learning algorithm. We will work in a setting of only one discrete time step for brevity ($t = 0, 1$) and take the view of an European Call option seller with strike $K = 100$, $\sigma = 0.4$, $r = 0.05$ and $\Delta t = 0.1$. Our action space will consist of m uniformly spaced hedges between 0 and 1. We will evaluate the optimal hedge for different sized bins and number of actions. Moreover, we will analyze the efficiency of learning by finding the mean square error between the BS Delta computed at the midpoint of each state/bin and the optimal action of said state. In this particular case, the reward function is constant for a given state transition and action, hence, we can store the rewards in a look-up table and reuse the rewards when required, to save computation time.



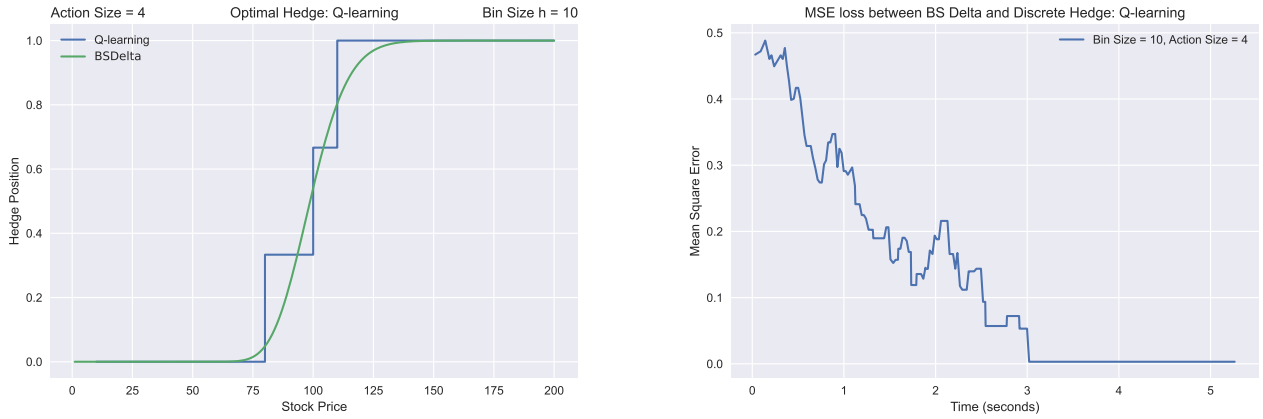
(a) Optimal Hedge over Spot price



(b) Mean square error between BS Delta and Discrete Optimal Action

Figure 2.1: Bin size of 20 and Number of Actions of 2

Figure 2.1 is a trivial example as there are only two actions $\{0, 1\}$, all the agent has to learn is to hedge if the spot is above the strike and liquidate all shares if the spot is below the strike. As the action space is small, we can use a large bin size of 20 which leads to quick computation and one can see from figure 2.1b, that convergence is achieved in around 0.8 seconds and required roughly 70 samples/episodes. Lastly, we work in a setting where $\mu = r$ with $\lambda = 10000000$ to make sure we work in a Black-Scholes limit so we can make a comparison with the BS-Delta.

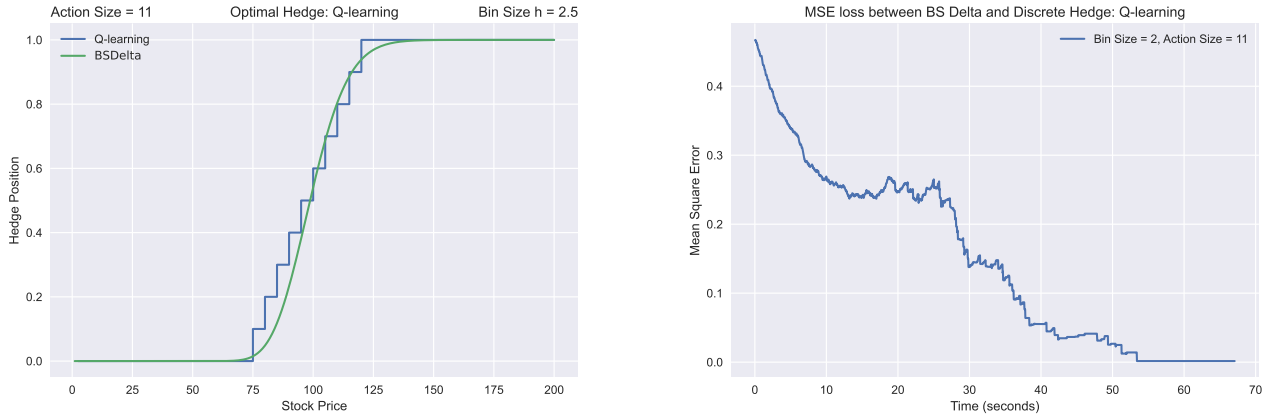


(a) Optimal Hedge over Spot price

(b) Mean square error of Discrete Action

Figure 2.2: Bin size of 10 and Number of Actions of 4

Figure 2.2a shows the optimal hedge with a bin size of 10 and four discrete actions available $\{0.0, 0.\dot{3}, 0.\dot{6}, 1.0\}$. One can see that the discrete actions is slowly taking the shape of the BS-Delta with convergence taking roughly 3 seconds requiring less than 400 samples/episodes.

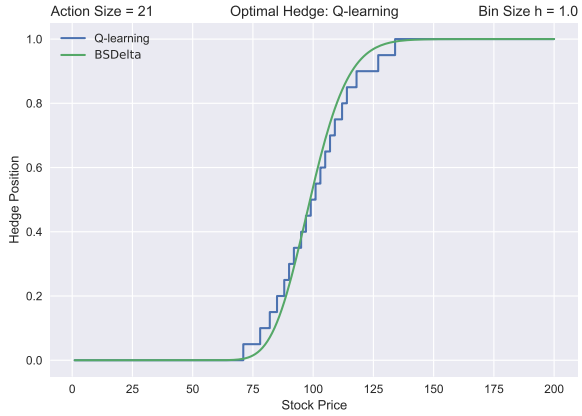


(a) Optimal Hedge over Spot price

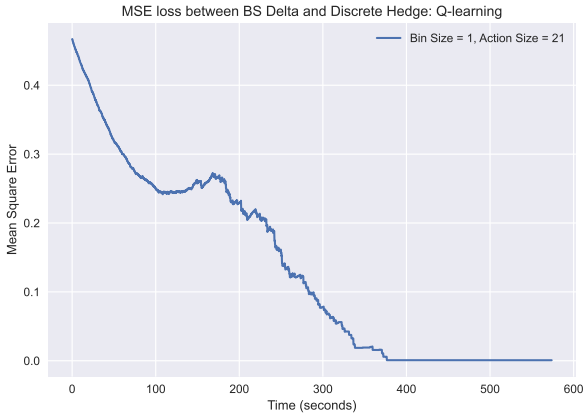
(b) Mean square error of Discrete Action

Figure 2.3: Bin size of 2.5 and Number of Actions of 11

In figure 2.3, we reduce the bin size and increase the action size significantly with a bin size of 2.5 and action space consisting of 11 actions, equally spaced by 0.1 between 0 and 1. The convergence takes roughly one minute with 2000 samples required. Even though the mean square error is low (0.00154), one can note that for spots slightly below the strike, the convergence to the BS-Delta is inefficient and the optimal action for these spots is far off the BS-Delta.



(a) Optimal Hedge over Spot price



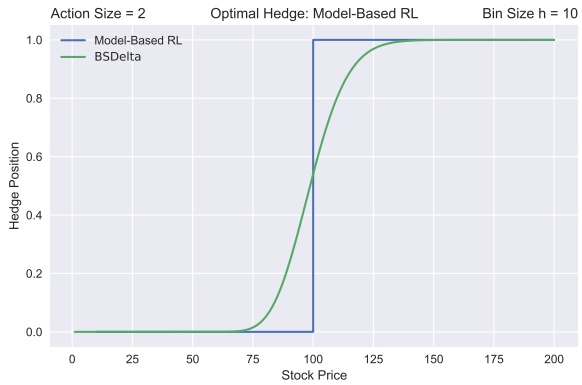
(b) Mean square error of Discrete Action

Figure 2.4: Bin size of 1 and Number of Actions of 21

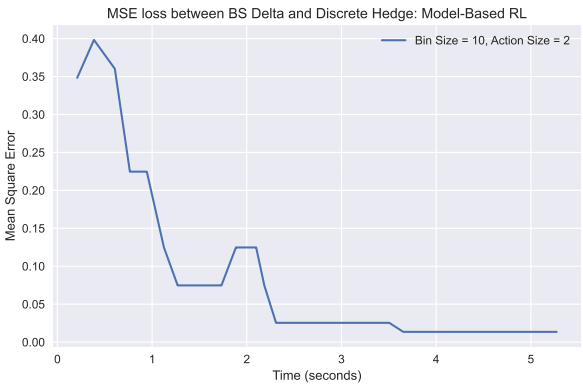
In figure 2.4a, the bin size is 1 and the action space has 21 different actions equally spaced by 0.05 from 0 to 1. In this particular case the convergence to the BS-delta is semi-efficient however convergence requires 10,000 samples and takes 10 minutes.

2.7.2 Model-Based RL: Tabular Form

We will now implement a model-based RL method described in section 1.4.1, where we try to approximate the MDP in a tabular form and then use value iteration to solve and find the optimal hedge. In our problem, the transition probability only depends on the current state and the next state and is independent of the action taken, whereas the reward function is dependent on all three. We will use the same sized bins and action spaces as we did in Q-learning and compare the efficiency of the two methods. Our set up is the same as the previous section with the agent taking the view of a call option seller with $K = 100$, $\Delta t = 0.1$, etc.



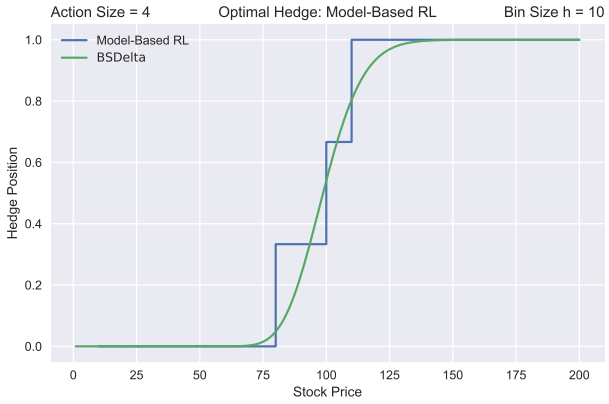
(a) Optimal Hedge over Spot price



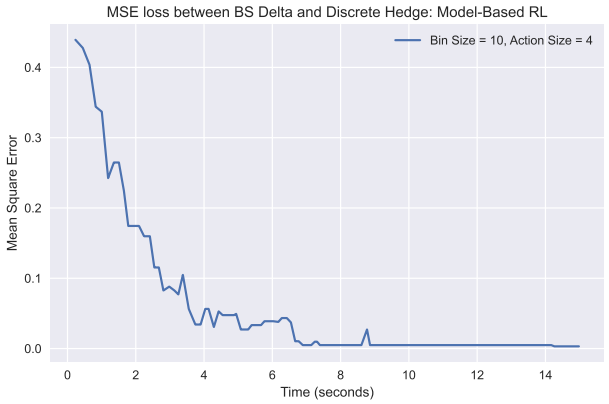
(b) Mean square error of Discrete Action

Figure 2.5: Model-Based RL: Bin size of 1 and Number of Actions of 21

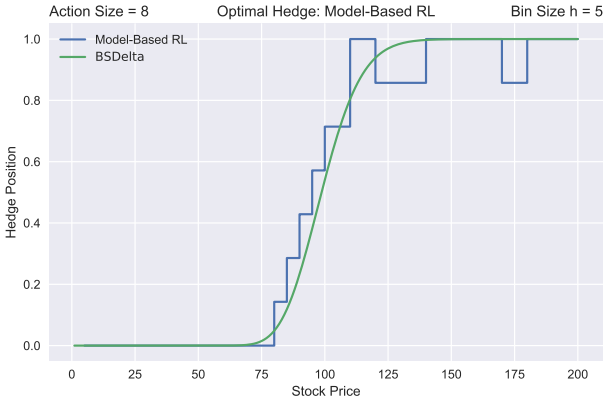
Figure 2.5 depicts the trivial case of only having two actions available $0, 1$ with a bin size of 10. Convergence takes roughly 4 seconds and is considerably slower than convergence achieved in a similar setting by Q-learning. We will look at a few more cases before discussing why this is the case.



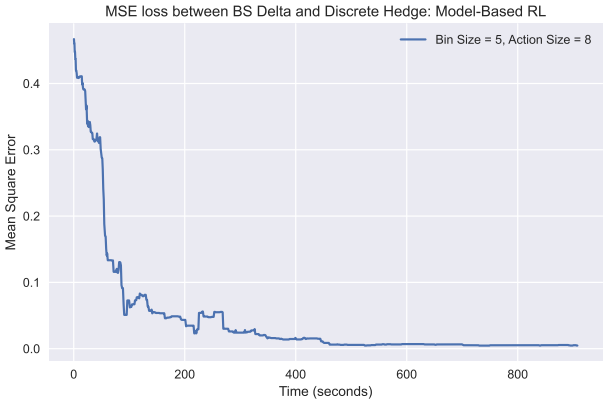
(a) Bin size 10, action size 4



(b) Bin size 10, action size 4 - MSE



(c) Bin size 5, action size 8



(d) Bin size 10, action size 4 - MSE

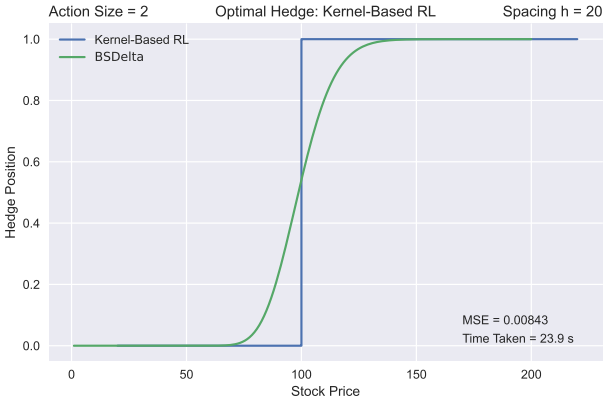
Figure 2.6: Discrete optimal hedge computed via Model-Based RL for the following parameters: $K = 100$, $\sigma = 0.4$, $r = 0.05$, $\mu = 0.7$ and $\Delta t = 0.01$

From the plots in figure 2.6, it is clearly obvious that the convergence via the model-based RL method is considerably less inefficient than by Q-learning. For a bin size of 5 with an action space of 8, convergence took over 800 seconds which was still inefficient and the actions did not converge well to the Black-Scholes delta. In actual fact, one can see from figure 2.6c, that around a spot price of 175, the optimal hedge decreases from 1.0 to 0.86 when the spot is well above the strike and should be 1.0.

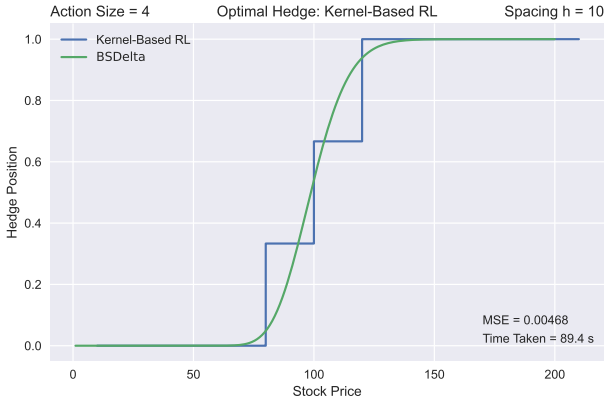
The fact Q-learning is a more efficient method in our problem than Model-Based RL is surprising. For our Frozen Gridworld example from section 1.4.1, this was certainly not the case with Model-Based RL being much more efficient in terms of computation. The reason why Q-learning fares much better in this problem rather than the Frozen Gridworld problem is because in the Frozen Gridworld environment, rewards are infrequently awarded (only if the agent reaches the goal state). Hence, in the Gridworld problem, the chance that the agent randomly reaches the goal state is low and so with Q-learning, the agent takes time to learn. However, in this financial setting, rewards are given out frequently (at every rebalancing step) and so the agent can learn efficiently with an on-line learning method such as Q-learning. Moreover, as the bin sizes get smaller, the number of states gets larger and for Model-Based RL, we require many more samples than Q-learning just to first learn the MDP before we can even apply the iterative step. This just shows that not one RL method is universally better than another and depends on the environment and problem at hand.

2.7.3 Kernel-Based Method

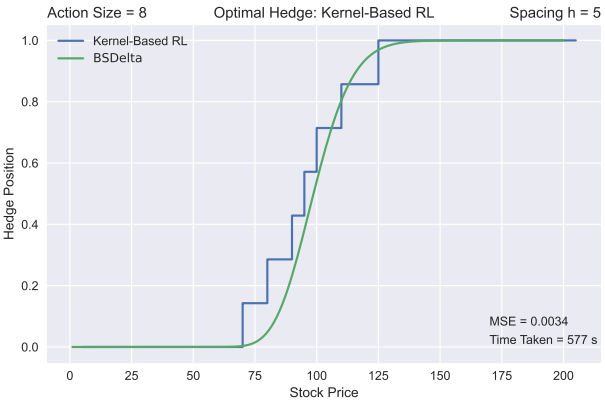
We will now implement and briefly discuss a Kernel-Based RL method as explained in section 1.4.2, in a similar setting in the previous two sections. We will analyse and compare this technique to Q-learning and Model-Based RL. As usual, we will take the view of a seller of a European Call Option with strike $K = 100$, $\Delta t = 0.1$ and $r = 0.05$. Technically, this method is implemented in a continuous state setting with discrete actions and does not fall in the same category as the previous two methods. We will be using Gaussian kernels with 100 samples to save computation time. As we are in a continuous setting, we will produce the optimal hedge for a discrete finite number of spot prices which are equally spaced by a length h and compare this to the previous two methods.



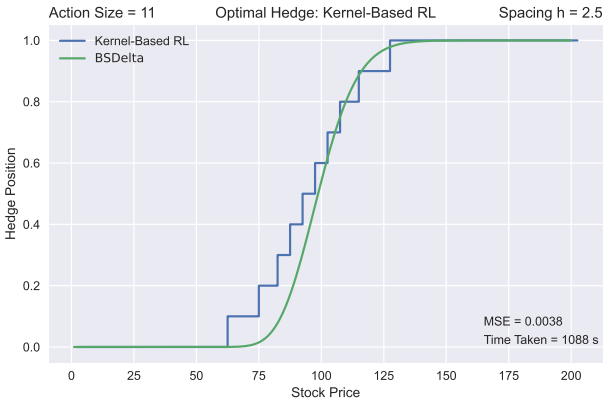
(a) 2 actions with spacing $h = 20$



(b) 4 actions with spacing $h = 10$



(c) 8 actions with spacing $h = 5$



(d) 10 actions with spacing $h = 2.5$

Figure 2.7: Kernel-Based RL estimate of discrete optimal hedge for the following parameters: $K = 100$, $\sigma = 0.4$, $r = 0.05$, and $\Delta t = 0.1$

From the plots in Figure 2.7, one can see that the convergence to the BS-delta is mediocre and takes considerably longer than the previous two methods. For example, 2.7d, shows the discrete optimal hedge in a setting of 11 actions with a spacing of 2.5, which took 1088 seconds and a mean square error of 0.0038. Whereas, a similar setting with the Q-learning method (Figure 2.3a) took a minute to converge and had a MSE of 0.00154. Thus, with Q-learning we achieved convergence much faster and with a smaller error. This suggests that Q-learning is a relatively efficient method in a discretized state-setting with discrete actions for our sequential maximisation problem.

2.8 Continuous States with Continuous Actions

We will now work in a continuous state-setting with continuous hedges just as in section 3.5 of Halperin [4]. We assume the agent has knowledge of equations (2.6.6) and (2.6.7). These equations can be solved in a backward recursive manner using N Monte-Carlo samples simultaneously. Yet again, the agent will take the view of a European Call option seller who is trying to hedge with a dynamic replicating portfolio consisting of the underlying stock and cash. In the first subsection, we will represent the optimal hedge and Q-value function with basis functions with the state variable X_t as the input. In the second subsection, we will use a neural network to represent the optimal hedge as a function of the stock price S_t rather than X_t. In the last part of this chapter, we will compute the optimal hedge using Kernel Estimation and compare this to the previous methods.

2.8.1 Basis Function Implementation

We will now use basis functions $\Phi_n(X_t)$ to represent both the optimal hedge $a_t(X_t)$ and the optimal Q-value function $Q_t^*(X_t, a_t^*)$. In this method, we only require numerical computation and some linear algebra. We can express both these terms as a summation of M number of basis functions with time-dependent coefficients ϕ_{nt} and ω_{nt} [4, equation 43]:

$$a_t^*(X_t) = \sum_{n=1}^{M} \phi_{nt}\Phi_n(X_t), \qquad Q_t^*(X_t, a_t^*) = \sum_{n=1}^{M} w_{nt}\Phi_n(X_t) \tag{2.8.1}$$

The time-dependent coefficients ϕ_{nt} and ω_{nt} are computed in a backward recursive manner from $t = T-1$ to $t = 0$. We first find the coefficients for the optimal hedge ϕ_{nt} and then use the optimal hedge to produce the Q-value function, $Q_t^*(X_t, a_t^*)$. Thus, just as we discussed earlier, we hedge first and then price second.

The way we find the coefficients for the optimal hedge is by maximising the Bellman Optimality equation (2.6.6) with the basis expansion from equation (2.8.1) substituted for the optimal hedge. The expectation of this optimality equation can be replaced with a summation over Monte-Carlo estimates. We will transform the minimisation into a maximisation by taking the negative of the expression and drop any terms independent of a_t as we are optimizing with respect to a_t. This leads to maximising the following with respect to the basis expansion of the optimal hedge [4, eqn 44]:

$$\sum_{k=1}^{N}\left(-\sum_{n=1}^{M}\phi_{nt}\Phi_n(X_t^k)\Delta S_t^k + \gamma\lambda\left(\hat{\Pi}_{t+1}^k - \sum_{n=1}^{M}\phi_{nt}\Phi_n(X_t^k)\Delta\hat{S}_t^k\right)^2\right) \tag{2.8.2}$$

Where the superscript k is the kth Monte-Carlo sample. The minimization of equation (2.8.2) with respect to the coefficients ϕ_{nt} leads to the following set of linear equations [4, eqn 45]:

$$\sum_{m=1}^{M} A_{nm}^{(t)}\phi_{mt} = B_n^{(t)}, \qquad n = 1, \ldots, M \tag{2.8.3}$$

Where $A_{nm}^{(t)}$ and $B_n^{(t)}$ are:

$$A_{nm}^{(t)} = \sum_{k=1}^{N}\Phi_n(X_t^k)\Phi_m(X_t^k)\left(\Delta\hat{S}_t^k\right)^2 \tag{2.8.4}$$

$$B_n^{(t)} = \Phi_n(X_t^k)\left[\hat{\Pi}_{t+1}^k\Delta\hat{S}_t^k + \frac{1}{2\gamma\lambda}\Delta S_t^k\right] \tag{2.8.5}$$

The above equations (2.8.4) and (2.8.5) are the element-wise expansions of matrix $\mathbf{A}_t$ and vector $\mathbf{B}_t$ respectively. The coefficients of the basis expansion of the optimal hedge ϕ_t in a vector form is [4, eqn 47]:

$$\phi_t^* = \mathbf{A}_t^{-1}\mathbf{B}_t \tag{2.8.6}$$

One can notice the similarity between equation (2.8.6) and the equation for the optimal hedge (2.6.7), where the expression for $B_n^{(t)}$ represents the numerator of equation (2.6.7) and the expression for $A_{nm}^{(t)}$ depicts the denominator.

Now that we have the basis expansion for the optimal hedge $a_t^*(X_t)$, we can use this to help us solve coefficients ω_{nt} for the Q-value functions. This can be achieved by treating the Bellman Optimality equation (2.6.5) as a regression problem [4, eqn 48]:

$$R_t(X_t, a_t^*, X_{t+1}) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) = Q_t^*(X_t, a_t^*) + \epsilon_t \tag{2.8.7}$$

Where ϵ_t is a random noise at time t with mean zero. Thus, under expectation, we indeed obtain the Bellman Optimality equation (2.6.5).As this is a regression problem, the coefficients ω_{nt} are found by solivng the following least-square minimization problem of the Monte-Carlo samples with the basis expansion of the Q-value function at time t:

$$\sum_{k=1}^{N}\left(R_t(X_t^k, a_t^*, X_{t+1}^k) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*(X_{t+1}^k, a_{t+1}) - \sum_{n=1}^{M}\omega_{nt}\Phi_n(X_t^k)\right)^2 \tag{2.8.8}$$

The minimization of this expression leads again to a set of linear equations with matrix $\mathbf{C}_t$ and vector $\mathbf{D}_t$ with elements:

$$C_{nm}^{(t)} = \sum_{k=1}^{N}\Phi_n(X_t^k)\Phi_m(X_t^k) \tag{2.8.9}$$

$$D_n^{(t)} = \sum_{k=1}^{N}\Phi_n(X_t^k)\left(R_t(X_t^k, a_t^*, X_{t+1}^k) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*(X_{t+1}^k, a_{t+1})\right) \tag{2.8.10}$$

The optimal coefficients ω_t^* for the Q-value function can be obtained in a vector form:

$$\omega_t^* = \mathbf{C}_t^{-1}\mathbf{D}_t \tag{2.8.11}$$

We can use equations (2.8.6) and 2.8.11 recursively from $t = T-1, \ldots, 0$ to obtain the optimal hedge and subsequently, the price of an option. Note again, that in this scenario, we assume the agent has knowledge of the MDP dynamics, mainly the reward function $R_t(X_t, a^*t, X_{t+1})$. These equations can be utilised in both a continuous and discrete state setting, but for now we will stick to a continuous setting for the rest of the paper.

However, there are still many questions which have been unanswered. One major question is the choice of basis functions and the number of functions required. For our continuous setting, a smooth basis function would be desirable. One obvious that comes to mind is polynomial functions which we will analyse and discuss later on. Another choice which may fare better than a polynomial basis is using trigonometric functions.

Polynomial Basis Function

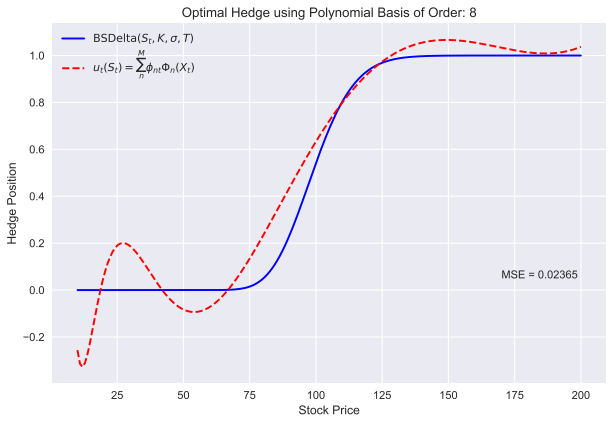
We will now implement polynomial basis function expansion for the optimal hedge and Q-value function. As usual, we will take the view of a European Call Option seller who has to hedge the option with a dynamic replicating portfolio consisting of the underlying stock and cash. We assume the stock price dynamics are under the discrete Black-Scholes model with parameters $\sigma = 0.4$, $\Delta t = 0.1$, $\mu = r = 0.05$ $K = 100$ and $\lambda = 10000$. Note how we have set $\mu = r$ so that in the limit $\Delta t \to 0$, we tend to the continuous BS-limit and we work in a setting of one discrete time step $t = 0, 1$ for brevity. We will assume we have access to $N = 10,000$ Monte-Carlo sample paths. We will analyse the convergence of our optimal hedge to the BS-delta under increasing degrees of polynomial functions.

More formally, the optimal hedge and Q-value function with polynomial basis functions can be expressed as:

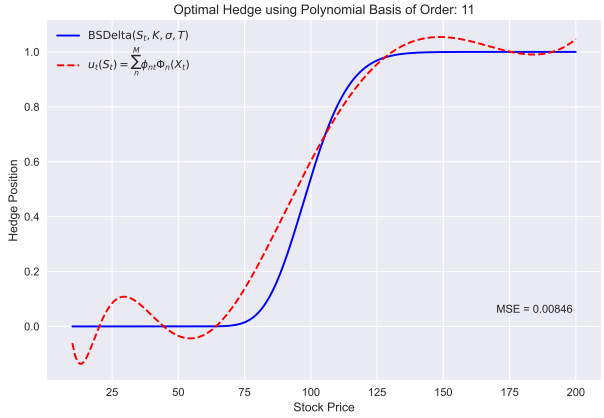
$$a_t^*(X_t) = \sum_{n=0}^{M} \phi_{nt}(X_t)^n, \qquad Q_t^*(X_t, a_t^*) = \sum_{n=0}^{M} w_{nt}(X_t)^n \tag{2.8.12}$$

Where M is the order of the polynomial. The first plot will represent the optimal hedge based on a cubic expansion.

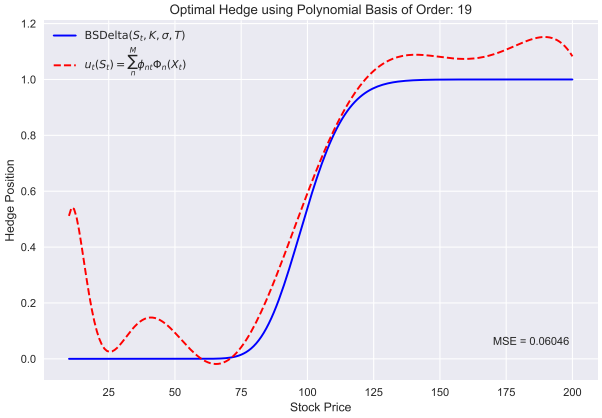
(a) Order of 3



(b) Order of 8



(c) Order of 11



(d) Order of 19

Figure 2.8: Optimal hedge computed via polynomial basis functions for the following parameters: $K = 100$, $\sigma = 0.4$, $r = 0.05$, and $\Delta t = 0.1$

Figure 2.8 shows plot of the optimal hedge computed using polynomial basis functions of varying orders. The plots are computed from coefficients which produced the smallest mean square error over the 10,000 samples over 100 different runs. From the above plots, one can see the mean square error does decrease as we increase the order from 3 to 11 but not much difference between an order of 11 and 19. The mean square error we obtain for an order of 11 is 0.00846 which is relatively small but not as good as Kernel Estimation. Thus, polynomial basis functions have a relatively poor convergence to the Black-Scholes Delta. Let us look at the mean squared error over different orders of polynomials.

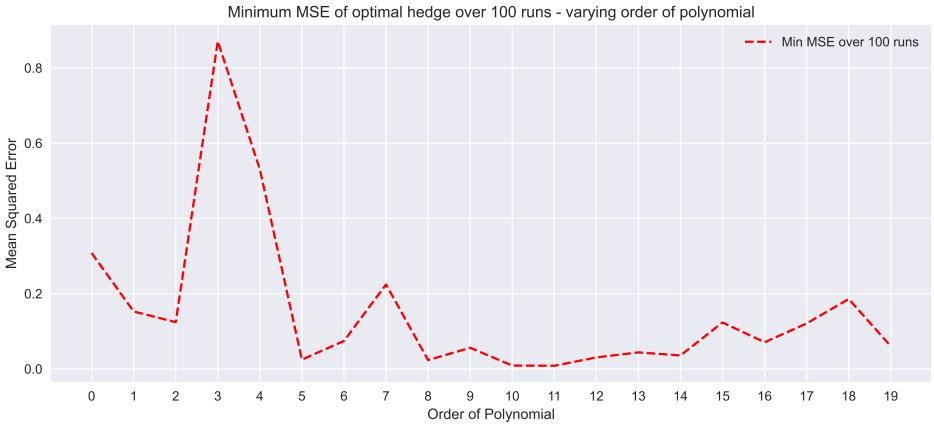


Figure 2.9: Mean square error between estimated hedge and BS Delta over 10,000 samples

Figure 2.9 depicts the smallest mean square error of the optimal hedge over 100 different runs for varying orders of the polynomial. A polynomial of order 11 has the smallest mean square error and the optimal hedge is represented in Figure 2.8c. It may seem surprising that a higher-ordered polynomial does not lead to a smaller mean squared error loss. This is due to the fact that we are minimizing the loss functional given by (2.8.2) rather than the mean squared error loss between our optimal hedge and the BS-Delta.

Let us compare the Q-value functions for orders of 3, 5, and 8. Now we have already seen that polynomial functions provide poor convergence for the optimal hedge. Let us assume we know the optimal hedge which in this case is the BS-delta and set $\lambda = 0$ so the negative Q-value function should be equal to the analytical BS-price.

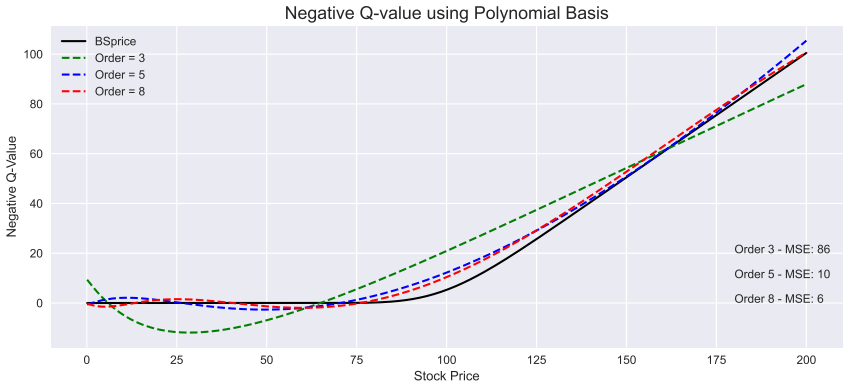


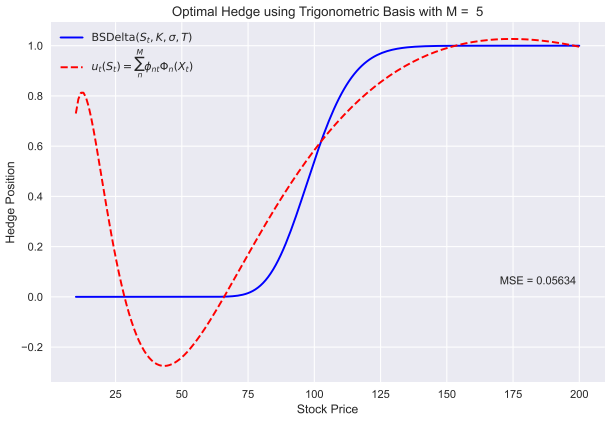
Figure 2.10: Negative Q-Value (price of option) at time $t = 0$ computed using polynomial basis

Figure 2.10 shows the Q-value functions computed using polynomial bases with $\lambda = 0$. As we increase the order of the polynomial, the mean-squared error between the BS-delta and the optimal hedge slowly decreases. However, recall that this was computed when we assumed that we knew the correct optimal hedge and not one produced using basis functions.

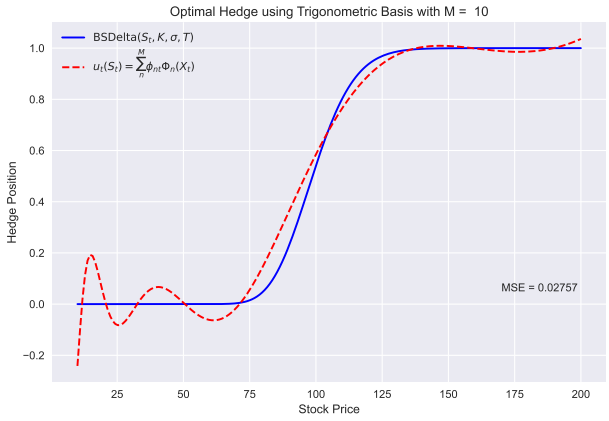
Trigonometric Basis Function

We will now implement a trigonometric function expansion for the optimal hedge in the same setting as the previous section. We will compare the efficiency against polynomial basis functions. As before, we take the view of a European Call Option seller with stock price dynamics following a discrete BS-model with one discrete time step. We use the same parameters as we did with polynomial basis functions and the same number of samples (10,000). We will now also check if the portfolio value at time 0 improves as we include more trigonometric functions. More formally, the basis expansion for the optimal hedge and Q-value function in a trigonometric setting can be expressed as:

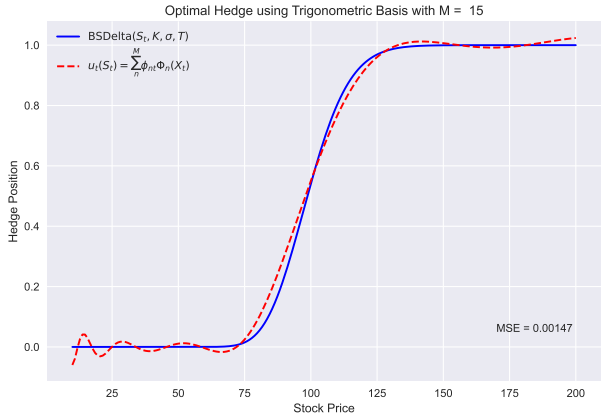
$$a_t^*(X_t) = \sum_{n=1}^{M} \phi_{nt}\big(\sin(nX_t)+\cos(nX_t)\big), \qquad Q_t^*(X_t, a_t^*) = \sum_{n=1}^{M} w_{nt}\big(\sin(nX_t)+\cos(nX_t)\big) \quad (2.8.13)$$



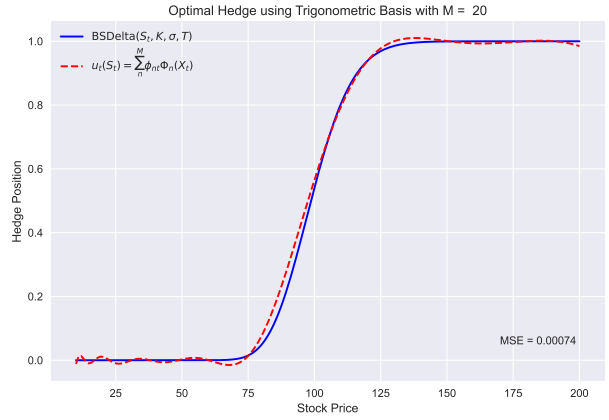
(a) M = 5



(b) M = 10



(c) M = 15



(d) M = 20

Figure 2.11: Optimal hedge computed via polynomial basis functions for the following parameters: $K = 100$, $\sigma = 0.4$, $\mu = r = 0.05$, and $\Delta t = 0.1$

Figure 2.11 is similar to figure 2.8 as it shows the optimal hedge computed using a trigonometric basis from the set of coefficients ϕ_{nt} which produced the smallest mean square error from 100 different runs. As you can see from the plots, as we increase M, the number of trigonometric functions, the mean square error between the optimal hedge and the BS-delta starts to decrease. The mean square error using trigonometric functions is much smaller than using polynomial basis functions. If we compare a polynomial of order 19 (Figure 2.8d) and trigonometric functions with $M = 20$ (Figure 2.11d), the mean square error using polynomial basis functions over 10,000 samples is 0.00846 whereas, with trigonometric functions, the mean square error is 0.000741, more than a factor of 10 smaller. Let us see how the mean squared error changes as we increase M.

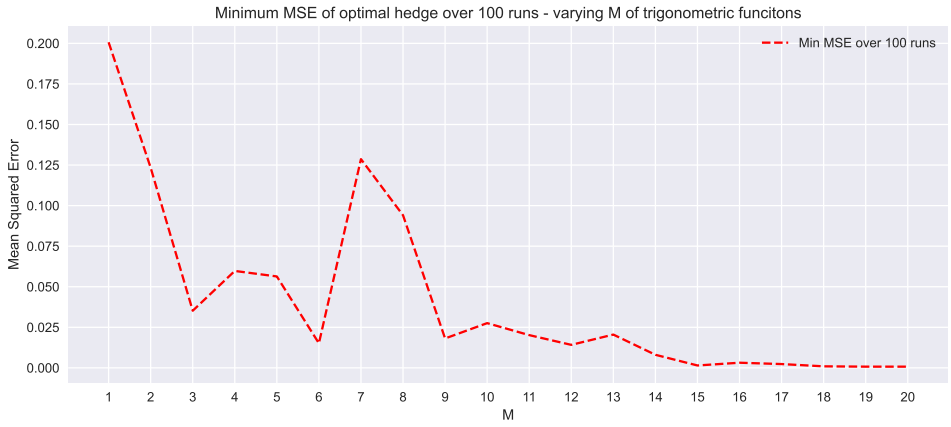


Figure 2.12: Mean square error between estimated hedge and BS Delta over 10,000 samples

Figure 2.12 is the trigonometric basis function version of Figure 2.9. The figure shows the smallest mean squared error achieved over 100 different runs for the optimal hedge computed using a trigonometric basis expansion. This figure is more promising than figure 2.9 as the mean square error is decreasing after M passes 15. Whereas for Figure 2.9, the MSE starts to increase after an order of 11. However, the mean-square error for the trigonometric basis expansion is not monotonically decreasing with M. This is due to the fact that we are minimizing the loss functional described by equation equation (2.8.2) and not the mean-square error loss between the optimal hedge and the BS-delta. Let us see how the actual loss functional changes as we increase M for the trigonometric basis expansion.

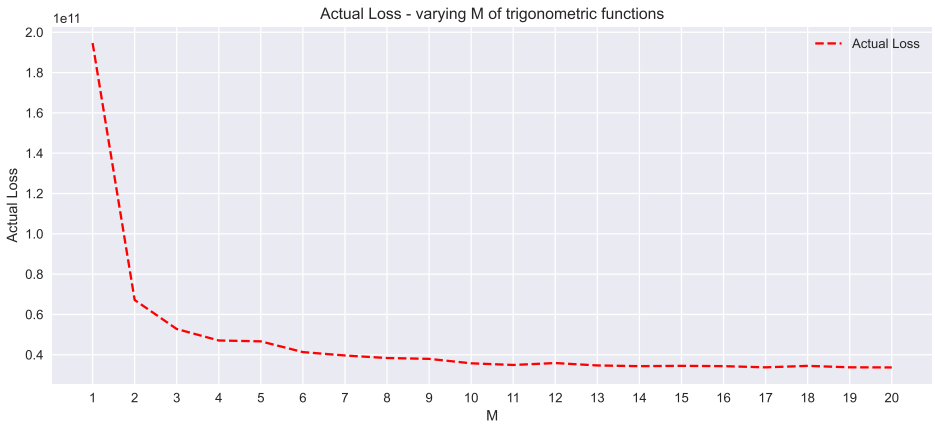


Figure 2.13: Actual Loss (2.8.2) using trigonometric basis functions

Figure 2.13 shows the actual loss function computed using equation (2.8.2) with trigonometric basis functions over different values of M using the same set of coefficients ϕ_{nt} as used in Figure 2.12. We use the sample paths for all values of M when computing the loss. Unlike the previous figure for the mean-squared error (Figure 2.12), the actual loss does indeed decrease as we increase the number of functions and the loss does not fluctuate. This is intuitive and expected as a larger value of M encompasses all possible combinations of functions of a smaller value of M. Hence, the loss should either stay the same or decrease as we increase the number of functions. Let us see how the Q-value function looks like with trigonometric basis functions in a setting where we assume we know the actual optimal hedge and set $\lambda = 0$ so we should obtain the BS-price.

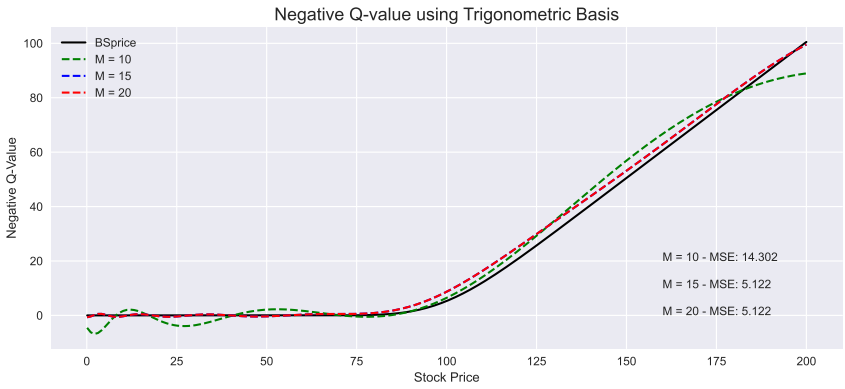


Figure 2.14: Negative Q-Value (price of option) at time $t = 0$ computed using trigonometric basis

Figure 2.14 shows the negative Q-value function as a basis expansion of trigonometric functions. We can see as we increase M, the negative Q-value slowly converges to the BS-price. The trigonometric basis functions provides a better fit to the price than polynomial basis functions. The trigonometric basis functions provide a quick and simple way to obtain the optimal hedge and price with decent convergence to the true hedge and price. We will look at how we can implement a neural network for both hedging and Q-value functions to try and create a system of efficient hedging and pricing.

2.8.2 Neural Network Implementation

Let us first define a feedforward neural network which we will employ to produce the optimal hedge and optimal Q-value function.

Definition 2.8.1 (Feedforward Neural Network)**.** [15] [16, definition: 2.1] Let $L \in \mathbb{N}$ and also $N_0 \dots N_L \in \mathbb{N}$. For any $i = 1 \dots L$, let $\sigma_i : \mathbb{R} \to \mathbb{R}$ and let $A_i : \mathbb{R}^{N_{i-1}} \to \mathbb{R}^{N_i}$ be affine functions such that:

$$A_i(\mathbf{x}) = W^i\mathbf{x} + \mathbf{b}^i \qquad \mathbf{x} \in \mathbb{R}^{N_{i-1}},\ W^i \in \mathbb{R}^{N_i \times N_{i-1}},\ \mathbf{b}^i \in \mathbb{R}^{N_i}$$

Where W^i is a matrix of weight parameters and $\mathbf{b}^i$ is a bias vector of parameters.

A function $f : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as:

$$f(x) = g_L(x) \circ \dots g_1(x) \qquad \text{where } g_i = \sigma_i \circ A_i$$

is called a feedforward neural network.

We will denote the class of such functions f by:

$$\mathcal{N}_L(N_0, \ldots, N_L; \sigma_1, \ldots, \sigma_L)$$

The value L denotes the number of layers of the neural network and N_i is the number of units in the i^{th} layer. These are the hyperparameters of the neural network whereas the elements of the matrices W^i and the bias vectors $\mathbf{b}^i$ are the actual parameters. $N_1, \ldots, N_{L-1}$ represent the dimensions of the hidden layers whilst N_0 and N_L are the dimensions of the input and output layer, respectively. One should also note that the activation functions σ_i are applied in a component-wise manner.

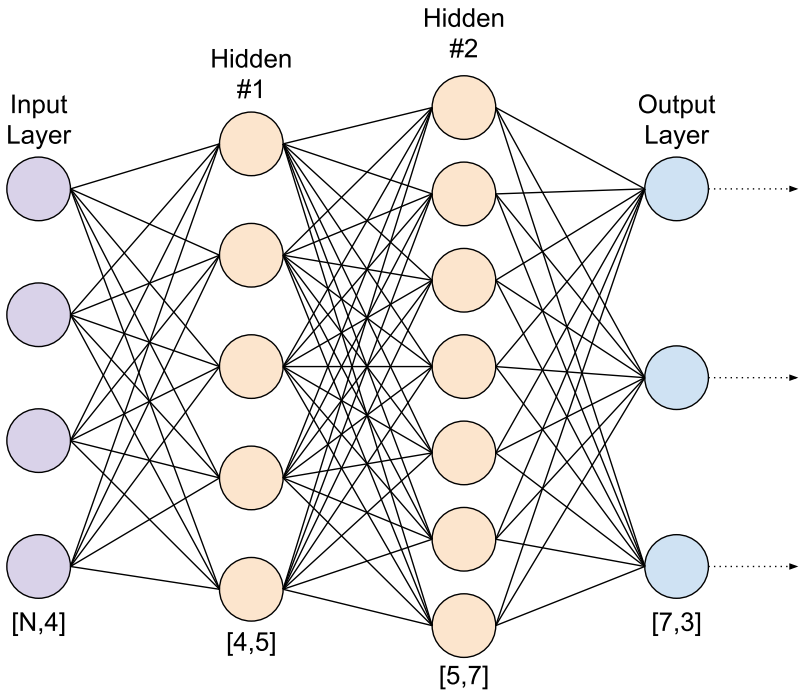


Figure 2.15: Graphical representation of feedforward neural network with $L = 3$, $N_0 = 4$, $N_1 = 5$, $N_2 = 7$ and $N_3 = 3$

[17]

Activation Functions

Let us briefly discuss some popular activation functions which we will use to solve for the optimal hedge and Q-value function.

Rectified Linear Unit (ReLU)

This is one of the most popular activation functions for the hidden layers. This is due to the simplicity of the function as the function is: $\mathbf{ReLU}(x) = \max(x, 0)$. The simplicity of the function means the derivative is simple to compute but it is undefined at zero, so we just set the derivative at zero to be 1. ReLu is numerically efficient and can still capture non-linear features of a function. Although, one of the major problems in using this activation function is that it does not allow any negative values to go through the units. This leads to zero gradients which is a problem for gradient-based learning algorithms and eventually leads to "dead" units that are never activated.

Exponential Linear Unit (ELU)

$$\textbf{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

The ELU function is similar to the ReLU function and has the same positive features but also deals with the dying ReLU problem by allowing negative values and the gradients are non-zero. Generally, we use a value between 0.1 and 0.3 for α. Using an exponential function introduces more complexity; this could lead to slower computation time, especially for large datasets. Moreover, the ELU function does not deal with exploding gradients, where the gradients are large and cause unstable updates to the weights.

Scaled Exponential Linear Unit (SELU)

$$\textbf{SELU}(x) = \begin{cases} \lambda x & \text{if } x \geq 0 \\ \lambda\alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

SELU is a relatively new activation function. The difference between ELU and SELU is the additional scaling parameter λ. SELU avoids both the exploding and vanishing gradient problem. The SELU activation function has an interesting property in that it is self-normalizing. In the paper by Klambauer - 2017 [18], it suggests to use an α and λ value of 1.673 and 1.0507, respectively.

Sigmoid and Tanh

The Sigmoid and Tanh functions use to be one of the most popular activation functions for the hidden layers before being surpassed by ReLu. This is because both of these functions are bounded and saturating. The boundedness is a problem for the hidden layers and negatively affects gradient-based learning. Nevertheless, they can serve well for the output layer, especially if we require a bounded result such as binary classification. In our case of hedging, we require our hedge to be bounded by 1, so the sigmoid function proves to be a good fit to achieve this result.

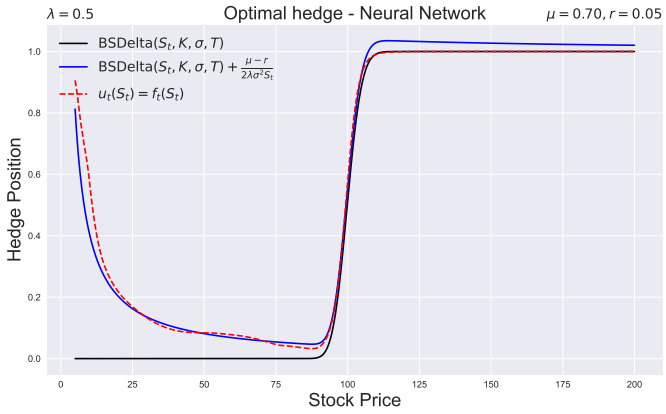
$$\textbf{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\textbf{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

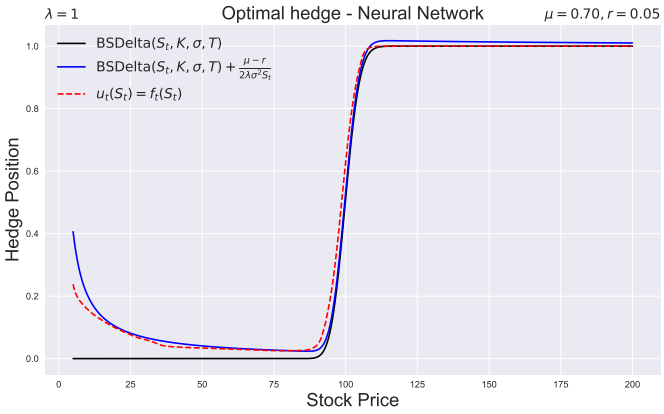
Let us now discuss the architecture of the neural network that we will employ to estimate the optimal hedge and how we are going to train this neural network. We will also revert to using the stock price S_t as the input rather than X_t. There is no significant difference in using either S_t and X_t, as the network should be able to learn the relationship between the two. Hence, we stick to using the stock price for the sake of ease. The purpose of our neural network is to minimize the loss depicted by equation 2.8.2, where we now substitute $u_t^*(S_t) = f(S_t)$ for the basis expansion $\sum_{n=0}^{M} \phi_{nt}(X_t)^n$ and dropping any terms independent of the optimal hedge.

We will implement the neural network using TensorFlow v2.0.0 and Keras v2.3.1. The neural network is implemented unconventionally, as we have a custom loss function with many different terms. Our network initially consists of 3 hidden layers with one input and four outputs, with 16 units at each hidden layer. We use either ReLU, ELU, or SELU as the activation functions for the hidden layers, while using Sigmoid as the output activation function. Our problem can be interpreted as a semi-supervised learning problem with our sampled data acting as labels and the stock price S_t is our input. Our labelled data consists of ΔS_t, $\hat{\Pi}_t$, $\Delta \hat{S}_t$ and an empty dummy array of zeros. The reason why we have 4 outputs rather than one is because the shape of the output must be the same as the labelled data which in our case is 4. The predicted output counterpart of the empty dummy array is our desired optimal hedge $u_t^*(S_t)$. Trivially, our input data is simply the stock price S_t.

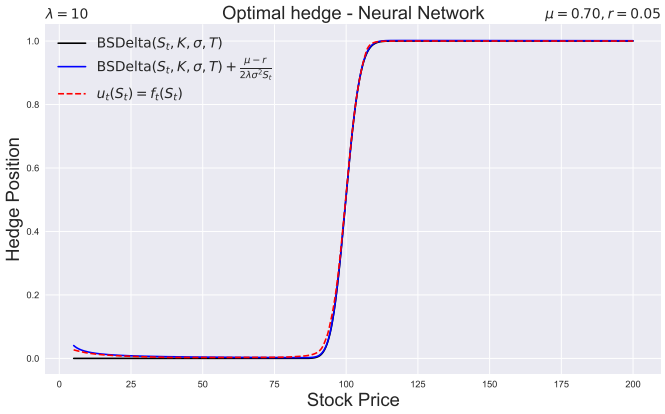
Let us now look at an example in which we utilize our neural network. In this particular example, we will produce stock price paths with $\mu \neq r$ and explore how the optimal hedge changes as we alter λ. As usual, we take the view of a European Call Option seller with strike $K = 100$, $\mu = 0.7$, $r = 0.05$ and $\sigma = 0.4$ with different values of λ and $\Delta t = 0.01$. We will compare our optimal hedge with the Taylor expanded hedge represented by equation (2.6.8). Our hedge should tend to the BS-delta as $\lambda \to \infty$. Let us see how well our neural network trains with 100,000 different samples with batch sizes of 100 using SELU as the hidden layer activation functions.



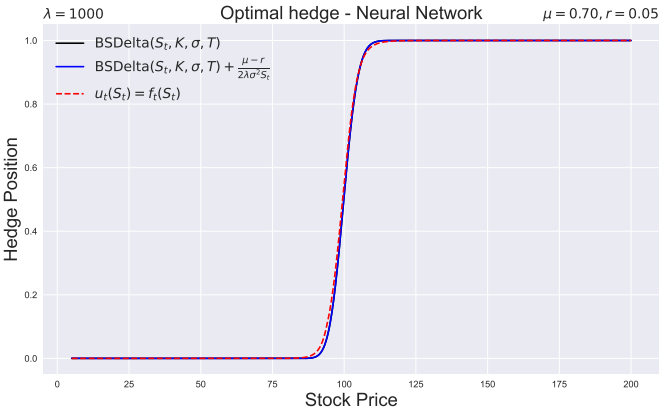
(a) $\lambda = 0.5$



(b) $\lambda = 1$



(c) $\lambda = 10$

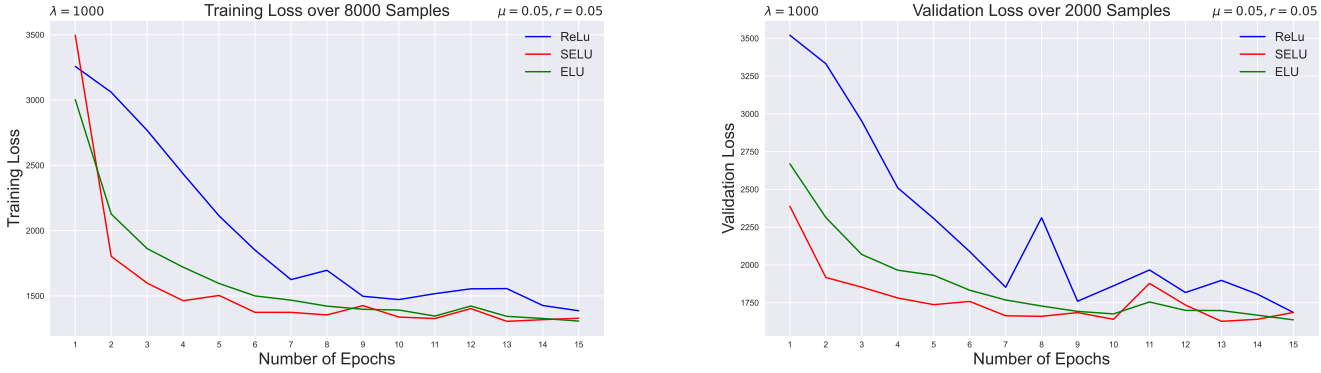


(d) $\lambda = 1000$

Figure 2.16: Optimal hedge over different risk-aversion factors for the following parameters: $K = 100$, $\sigma = 0.4$, $r = 0.05$, $\mu = 0.7$ and $\Delta t = 0.01$

Figure 2.16 shows four different plots of the optimal hedge over increasing levels of risk aversion λ. The red-dotted line depicts the optimal hedge computed by the neural network and the solid blue line is the Taylor expanded Black-Scholes delta eqn (2.6.8). The blue line represents the actual BS-delta and one can see that as the risk-aversion increases, our optimal hedge evaluated by the neural network tends to the BS-delta.

We will now compare the efficiency of using different activation functions for the hidden layers. We will now work in a setting where $\mu = r$ and so our neural network should converge to the BS-delta for an arbitrary value of λ which for our case, will be 1000. We will have access to 10,000 different sample paths and utilize 8000 for training and the other 2000 to compute the validation loss. We will train the neural network for each different type of activation using the same samples to make sure this is a fair test and compare the validation loss over 15 epochs. The activation functions to compare are ReLu, ELU, and SELU.



(a) Training loss

(b) Validation loss

Figure 2.17: Loss of network over different hidden layer activation functions - Optimal Hedge

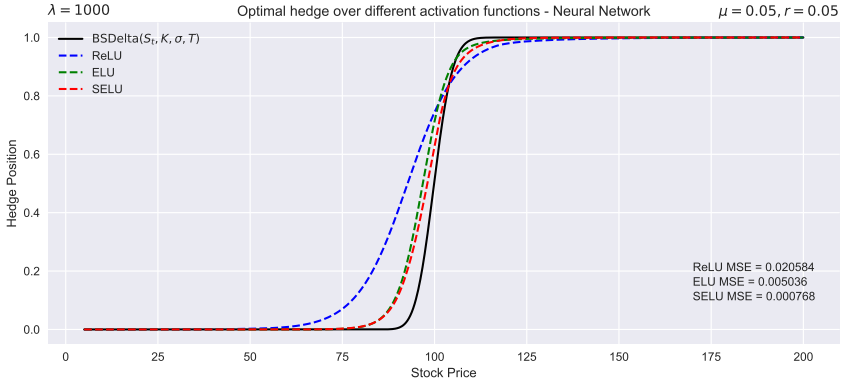
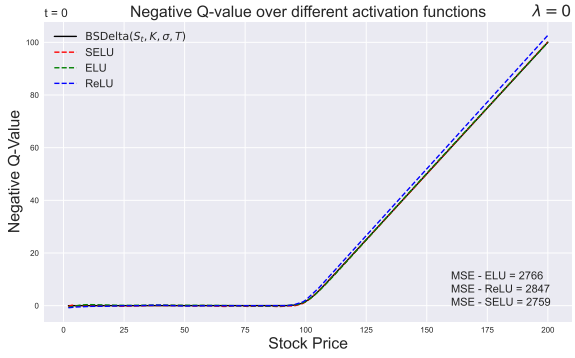


Figure 2.18: Optimal hedge with different activation functions

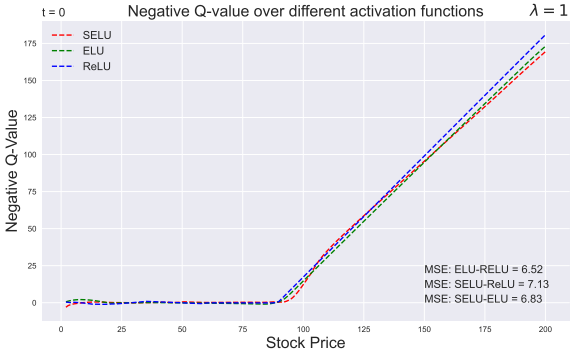
From Figure 2.17, we can see that having ReLU as the hidden layer activation function leads to a significantly larger training and validation loss compared to ELU or SELU. The use of ReLU can lead to "dead" units since negative values cannot pass through. This can be rectified by using ELU or SELU and consequently, the training is more efficient. We can see from Figure 2.17a, that the training loss of SELU is slightly lower than ELU over 15 epochs. The difference between these two activation functions is considerably small over a large number of epochs, however, SELU does learn quicker over a small number of epochs. The self-normalizing property of SELU can lead to better convergence.

Figure 2.18 shows the optimal hedge using the different hidden layer activation functions with the coloured dotted lines. The solid black line depicts the actual BS-delta. As we are working in a setting where $\mu = r$, we expect the optimal hedge to converge to the BS-delta. However, we are only working with 10,000 samples and our neural network needs more than 10x times more samples to obtain sufficient convergence to the BS-delta. Nonetheless, it is obvious to see from this figure that ReLU provides a much poorer convergence than either ELU or SELU, and the MSE between the hedge obtained by ReLU and the BS-delta is considerably larger than the counterparts of ELU and SELU. The mean-square error between the hedge obtained by SeLU and the BS-delta is smaller than the analogous error obtained by using ELU. It seems as if by solving the "dead" unit problem leads to a significant improvement in convergence as depicted by ELU, and thereafter, the self-normalizing property of SELU improves the convergence further.

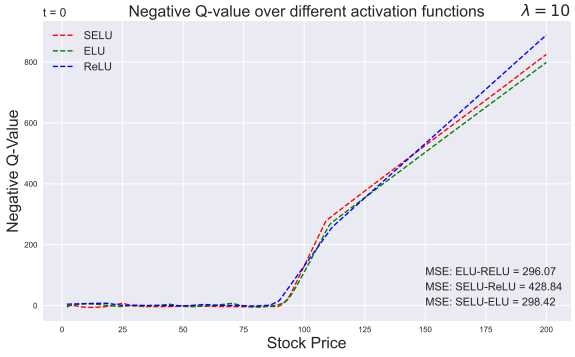
We can also compute the Q-value function with a neural network rather than the basis expansion. This is achieved by minimizing the loss depicted by equation (2.8.8). This is essentially a regression problem with a mean-square error loss function and is also semi-supervised. One can think of $R_t(S_t, a_t^*, S_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(S_{t+1}, a_{t+1})$ as our labels and our input is the stock price S_t. As before, we could use the variable X_t as our input but we use S_t instead for clarity. We will work in a setting where $\mu = r$ with varying λ ranging from 0 to 1000 and with access to 12,000 different samples. As we have already computed the optimal hedge in this setting, we will assume we know the hedge which in this setting is the BS-delta, and use this to help compute the Q-value function. We will compute the Q-value function using the three previous activation functions for the hidden layers and compare how the loss evolves over time. We will split the data set with 2,000 samples for validation and 10,000 for training. The architecture of our neural network will be similar to before with 3 hidden layers and the same hidden layer activation functions as before, with 16 units in each layer with one input and one output. However, the output activation function will simply be the identity function as a sigmoid would bound the output which is undesirable for the Q-value function.



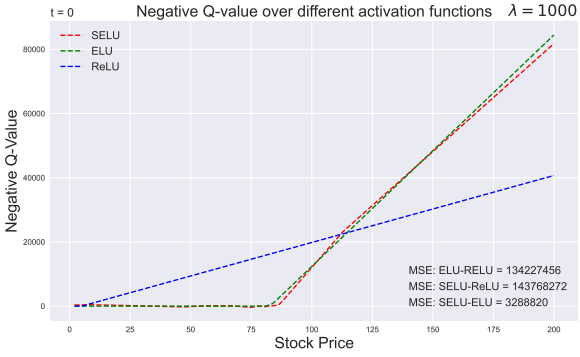
(a) $\lambda = 0$



(b) $\lambda = 1$



(c) $\lambda = 10$

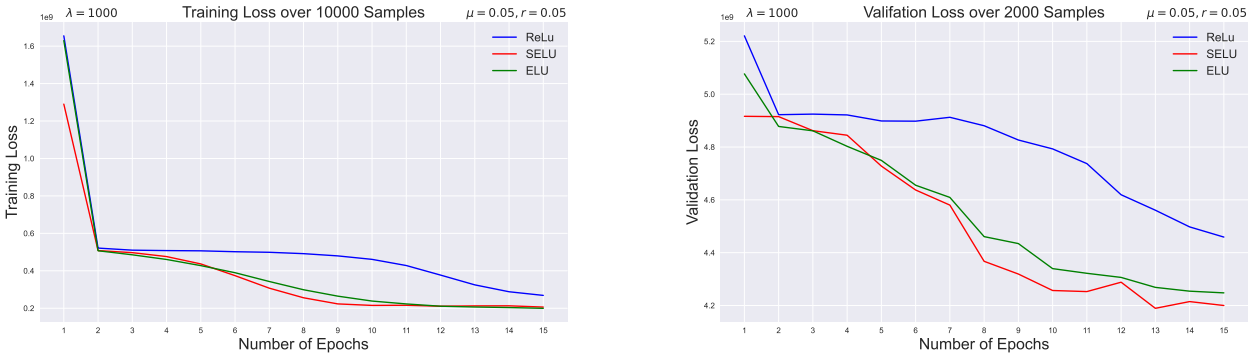


(d) $\lambda = 1000$

Figure 2.19: Negative Q-value function computed using neural network with different activation functions

In the bottom right-hand corner of figures 2.19b to 2.19d we have the mean-square distance between Q-values computed using the 10,000 samples between the neural networks implemented with each respective activation function. Generally, for the smaller values of λ, the neural networks powered by the different activation functions converge roughly to the same result, when $\lambda = 1$, the mean squared distance between SELU and ReLU is 7.13 while between SELU and ELU it is 6.83 so they are on the same order of magnitude. However, when $\lambda = 1000$, it is clear to see from Figure 2.19d that there is a discrepancy between ReLU and SeLU/ELU, the mean-squared distance between SeLU and ELU is 3288820, which is more than 10 times smaller than the distance between SELU and ReLU. So for large values of λ, ReLU struggles to train to what we perceive as the correct result set by SELU/ELU.

Figure 2.19a shows the negative Q-value function when $\lambda = 0$. This is a special case in which the negative Q-value should converge to the analytical Black-Scholes price for a Call Option. The solid black line depicts the Black-Scholes Call Price. The bottom-right hand corner of this figure shows the mean-square error between the Black-Scholes price and the negative Q-value computed using each respective activation function over the 12,000 samples. One can see clearly, that the neural network powered by ReLU provides the worst convergence with an MSE of 2847. By using ELU and thus eliminating the dead "unit" problem, this MSE improves by 2.85% to 2766. SELU then improves this further to a relatively lower value of 2759. Let us solidify this idea of ReLU providing poorer efficiency by looking at the training and validation loss for the case in figure 2.19d in which ReLU provides a significantly different result to SELU and ELU.



(a) Training loss

(b) Validation loss

Figure 2.20: Loss of network over different hidden layer activation functions: Q-Value Function

Figure 2.20 shows the training and validation loss for training the network to obtain the Q-value function over 15 epochs with $\lambda = 1000$. This is the training and validation loss from figure 2.19d where ReLU provided a very different solution to SeLU/ELU. We can see that SELU and ELU train more efficiently than ReLU which seems to plateau. The validation loss of ReLU is considerably larger than either SeLU or ELU over the 15 epochs with SELU generally having a lower validation loss than ELU. So it seems as if the solution obtained using ReLU is incorrect as the loss is considerably greater than the solution achieved by both SELU and ELU.

2.9 QLBS: Kernel Estimation

2.9.1 Nadaraya-Watson Estimator

In 1964, Nadaraya and Watson devised a non-parametric regression method to estimate a conditional expectation of a random variable Y given a random variable X using Kernel functions using a finite sample of data. Let us assume we have access to sampled data $(x_i)_{i=1}^n$ and $(y_i)_{i=1}^n$ of the dependent variable X and independent variable Y respectively. The Nadaraya-Watson estimator is:

$$\hat{\mathbb{E}}\big[Y \mid X = x\big] = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{i=1}^n K_h(x - x_i)} \tag{2.9.1}$$

Where K_h is a non-negative kernel function with smoothing parameter h.[19] [20]

We can utilize this estimator to compute the optimal hedge $u_t^*(S_t)$ using sampled data. The expression for the optimal hedge at time t consists of expectations of variables ΔS_t and Π_{t+1} conditioned on information $\mathcal{F}_t$. In our scenario, the only useful information available to us at time t is the stock price S_t so, we will condition on the underlying stock price S_t. Consequently, the optimal hedge is composed of conditional expectations with S_t as the dependent variable; this allows us to compute the optimal hedge using the Nadaraya-Watson estimator using our cross-sectional MC samples,$(S_t^k)_{k=0}^N, (\Delta S_t^k)_{k=0}^N$, and $(\Pi_{t+1}^k)_{k=0}^N$. We will be using Gaussian Kernels using Silverman's rule of thumb for the smoothing parameter h.[21]

$$K_h(x - x_i) \propto \exp\left(\left(\frac{x - x_i}{h}\right)^2\right) \tag{2.9.2}$$

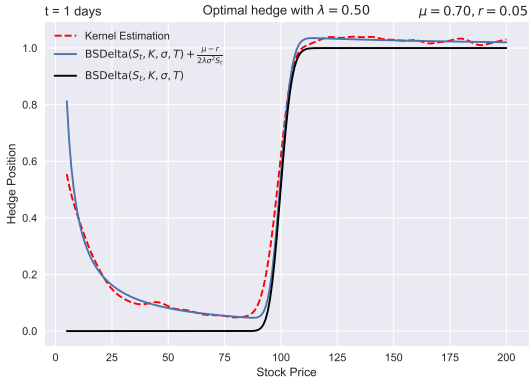
$$h = 0.9 * \min\left(\hat{\sigma}, \frac{IQR}{1.34}\right) n^{-0.2} \tag{2.9.3}$$

Where $\hat{\sigma}$ is the sample standard deviation of the sampled data $(x_i)_{i=1}^n$, IQR is the is the interquartile range of $(x_i)_{i=1}^n$ and n is the size of the sampled data.

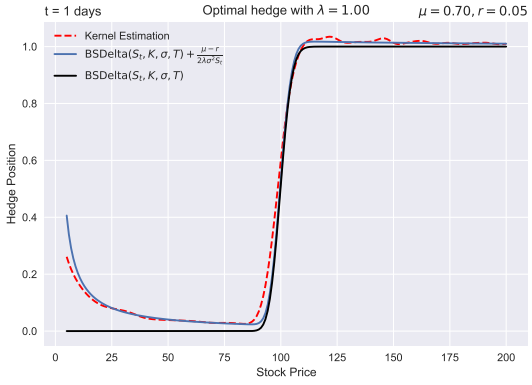
2.9.2 Hedging and Pricing using Kernel Estimation

We will now utilise the Nadaraya-Watson estimator to estimate the optimal hedge (2.6.7) in a QLBS setting. For this particular example, we will produce stock price paths in which $\mu \neq r$ and explore how the optimal hedge changes as we alter λ. From theory, we should see that as $\lambda \to \infty$, that for a small enough Δt, we should see convergence to the classical Black-Scholes Delta. We will also compare the efficiency of convergence of either setting $\mu = r$ or taking the limit $\lambda \to \infty$.

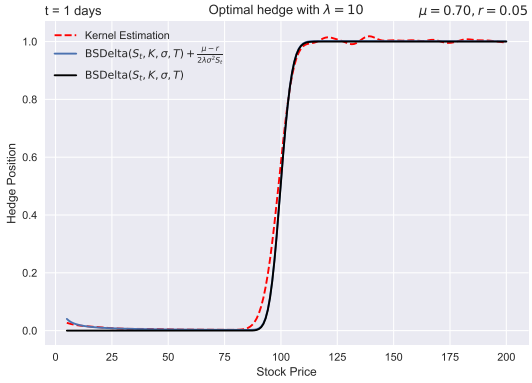
We will have a similar setting as the previous chapter and take the view of a European Call Option seller with two discrete time steps $t = 0, 1, 2$ and strike $K = 100$ with a $\Delta t = 0.01$ with $100,000$ sample paths. However, we will only consider the optimal hedge and Q-value at time 1 as the results are similar to time 0.



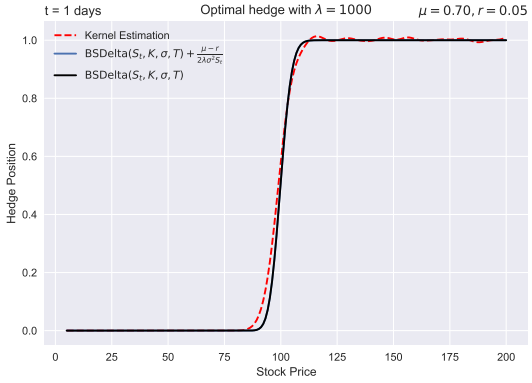
(a) $\lambda = 0.5$



(b) $\lambda = 1$



(c) $\lambda = 10$



(d) $\lambda = 1000$

Figure 2.21: Optimal hedge over different risk-aversion factors for the following parameters: $K = 100$, $\sigma = 0.4$, $r = 0.05$, $\mu = 0.7$ and $\Delta t = 0.01$

Figure 2.21 shows four different plots of the optimal hedge over increasing values for the risk-aversion. The red-dotted line represents our Kernel estimation while the blue line is the Taylor expanded hedge depicted by equation (2.6.8) . The solid black line is the classical Black-Scholes Delta. One can see that as the risk-aversion factor increases, the optimal hedge tends to the Black-Scholes Delta. Recall that we used the same number of samples to train the neural network. Computing the optimal hedge using Kernel Estimation is much faster and more efficient than training a neural network, however, from Figure 2.21 and Figure 2.16, we can see that there is not much considerable difference when we use 100,000 samples, we will investigate this further later.

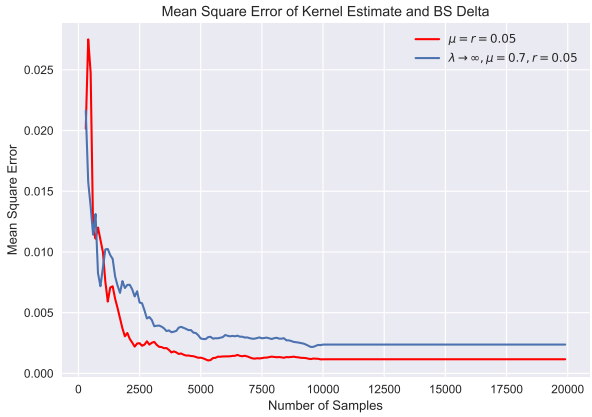
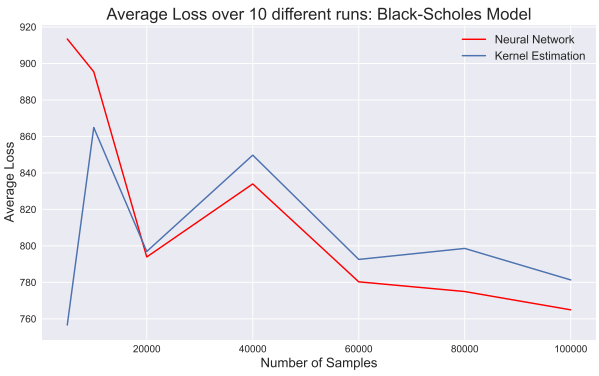


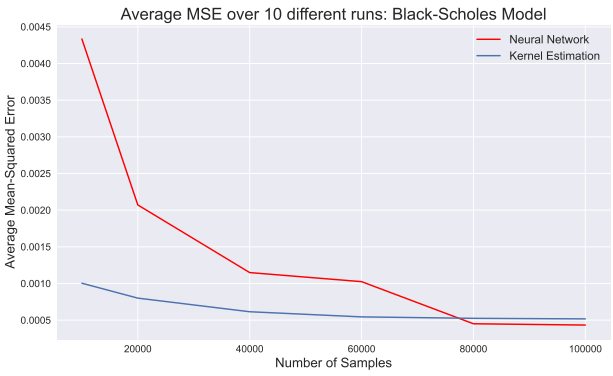
Figure 2.22: Mean square error between estimated hedge and BS Delta over 200 spot prices

Figure 2.22 shows the mean square error between the estimated hedge and the BS Delta over 200 different spot prices. As we mentioned earlier, theoretically, there is two methods to obtain convergence to the BS Delta. The red solid line depicts the method in which we set the drift of the stock equal to the risk-free rate with an arbitrary non-zero λ and the blue line represents the method which takes the limit $\lambda \to \infty$, which was achieved by setting $\lambda = 1 \times 10^1 1$. The convergence is a lot more efficient with the first method of setting $\mu = r$ as this completely eliminates the second term in the Taylor expansion (2.6.8).

Let us now compare the total loss and mean-square error of computing the optimal hedge using either the neural network or kernel estimation in the Black-Scholes limit as we increase the number of samples from 10,000 to 100,000. We will use the same set of sample paths for both methods to make this a fair test. Furthermore, we will take the average loss and mean-square error over 10 different sets of sample paths for each sample size. We will work in a setting where $\mu = r$ and with $\lambda = 1000$ so we are working in the BS-limit. We will use SELU as our hidden layer activation function for the neural network with the same architecture as in the previous section 2.8.2.



(a) Average loss over 10 different runs



(b) Average Mean-Squared Error

Figure 2.23: Loss and MSE over different sample sizes: BS-Model

Figure 2.23a shows the loss function computed using equation (2.8.2) averaged over 10 different sets of sample paths for each sample size. Note, that we use the same sample paths for both Kernel estimation and the neural network so we can directly compare the two. The loss function is dependent on the sample paths and so it is not an issue that the loss does not decrease as we increase the sample size. From this figure, we can see that for small sample sizes, the loss value by kernel estimation is significantly smaller than the loss produced by the neural network. This makes sense as neural networks require large amounts of data to train. As we increase the sample size to >20,000, the loss of the network improves rapidly and is now smaller than the loss generated via kernel estimation. Figure 2.23b shows the average mean-squared error between the optimal hedge and the BS-delta by both kernel estimation and the neural network. As expected, the neural network has a large value for the MSE, while kernel estimation has a much smaller value for small sample sizes (<10000). As we increase the sample size from 20,000 to 80,000, the MSE for kernel estimation improves slowly, whereas, for the neural network, the improvement is major but not enough to surpass kernel estimation. Once we have passed 80,000 samples, the network has a slightly smaller MSE than kernel estimation. From this result, we can imply that kernel estimation provides a relatively efficient convergence to the optimal hedge for small sample sizes. On the other hand, the neural network requires a much larger sample size (by a factor of 10) to obtain a sufficient result.

If we reduce the sample space to have stock prices in the range of [70,130] rather than [1,200], then kernel estimation converges much more efficiently than the neural network for large sample sizes as well as small sizes. Let us now investigate this further by computing statistics over 20 different runs with each run having a sample size of 100,000 and compare the statistics of both kernel estimation and the neural network to see which converges more efficiently.

	RMSE		MAE		MAPE		R^2	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
Kernel Estimation	0.00729	$8.25 \times 10^{-}4$	0.00552	$4.11 \times 10^{-}4$	5.21%	0.728%	0.9997	$6.57 \times 10^{-}5$
Neural Network	0.0342	0.0276	0.0221	0.0166	16.8%	14.5%	0.9894	0.162

Table 2.1: Statistics of convergence to BS-Delta over 20 different runs with sample size of 100,000 per run

Table 2.1 shows the mean and standard deviation of the error statistics over 20 different runs for 100,000 samples per run. (Note that the MAPE was calculated with all hedges above 0.01 as small hedges of zero cause division by zero). It is clear to see that errors produced by kernel estimation are significantly smaller and less noisy than the errors generated by the network.

Let us now conduct a paired t-test to compare the absolute errors of kernel estimation and the neural network (i.e. test to check if $|Network\ error| - |Kernel\ error|$ is significantly different to zero). Figure 2.24 shows a box plot of the 20 paired t-values of the absolute error over the 20 different runs. The blue dotted line is the 5% critical value for a two-tailed test. 19/20 of the t-values are far greater than this value which suggests that the absolute error of the network is greater than the error produced by kernel estimation.

Kernel estimation does not require the time-consuming training of a neural network and requires fewer data. A further downfall of the network is that the training stability is highly dependent on the initialization of the parameters. Poor initialization could potentially lead to a network that does not learn at all. The purpose of our neural network is to minimize the loss function with respect to the optimal hedge. For our QLBS setting, we can minimize this loss analytically and thereafter employ kernel estimation. Although, if we relax assumptions such as transaction costs, then a closed-form solution may no longer exist and we would require the use of a network to minimize this loss function.

Figure 2.24: Boxplot of 20 paired test t-values of absolute error of hedge obtained via neural network and kernel estimation: Black-Scholes Setting

Chapter 3

Extensions

3.1 Heston Model

The purpose of working in a Black-Scholes setting is the fact we can easily use the Black-Scholes model as a benchmark for our different methods of hedging and pricing. The Black-Scholes model provides simple stock price generation with closed-form analytical solutions which are not computationally taxing to evaluate and simple to reproduce results. We will now no longer assume that the volatility of the underlying asset is constant and work under the stochastic volatility model of Heston [22]. The Heston model is slightly more complex than the Black-Scholes model but still has closed-form solutions to pricing and hedging and does not require much more computation power to evaluate. Note, the formulae for the closed-form solution can be found in the appendix B.2.The stock price S_t and volatility process ν_t under the Heston is given below: [23, pg 861]

$$dS_t = \mu S_t dt + \sqrt{\nu_t} S_t dW_t^S \tag{3.1.1}$$
$$d\nu_t = \kappa(\theta - \nu_t)dt + \sigma_\nu \sqrt{\nu_t} dW_t^\nu \tag{3.1.2}$$

Where μ is the drift of the stock price, κ is the mean-reversion rate of the volatility, θ is the long-term mean of the volatility, σ_ν is the volatility of the volatility and W_t^S, W_t^ν are standard Brownian motions with correlation ρ. We can simulate a stock price process $(S_t)_{t=0}^T$ in a *discrete-time* setting under the Heston model in the following way:

$$S_{t+1} = S_t \exp\left[\left(r - \frac{\nu_t}{2}\right)\Delta t + Z_t^1 \sqrt{\nu_t \Delta t}\right]$$

$$\nu_{t+1} = \nu_t + \kappa(\theta - \nu_t)\Delta t + \left(\rho Z_t^1 + Z_t^2 \sqrt{1-\rho^2}\right)\sqrt{\nu_t \Delta t}$$

Where Z_t^1, Z_t^2 are standard independent normal variables. Once we have simulated the stock price paths, we can then produce the optimal hedge using either Kernel Estimation or the neural network. We can use (2.6.7) to compute the optimal hedge with kernel estimates. We can create a neural network and thereafter, minimize the loss given by equation (2.8.2), we obtain the optimal hedge. This is the same as the previous chapter, the only difference is the dynamics of the stock price process. We will compare the efficiency of both kernel estimation and the neural network in learning the optimal hedge within the Heston model. As before, we will work in a setting where we take the view of a European Call option seller with strike $K = 100$ and $\mu = r = 0.05$ with $\lambda = 1000$. For the parameters of the Heston model, we use parameters given in Kwon (2018) [24, Section 3] which is consistent with empirical results for a single stock give in Johannes et al.(2009) and Bakshi et al.(2010). [25] [26]. We will work in a discrete-time setting with one discrete time step $t = 0, 1$ with the following parameters:

$$\Delta t = 0.01,\ \mu = r = 0.05\ \kappa = 3.17,\ \theta = 3.17,\ \sigma_\nu = 0.3,\ \nu_0 = 0.2,\ \rho = -0.5 \tag{3.1.3}$$

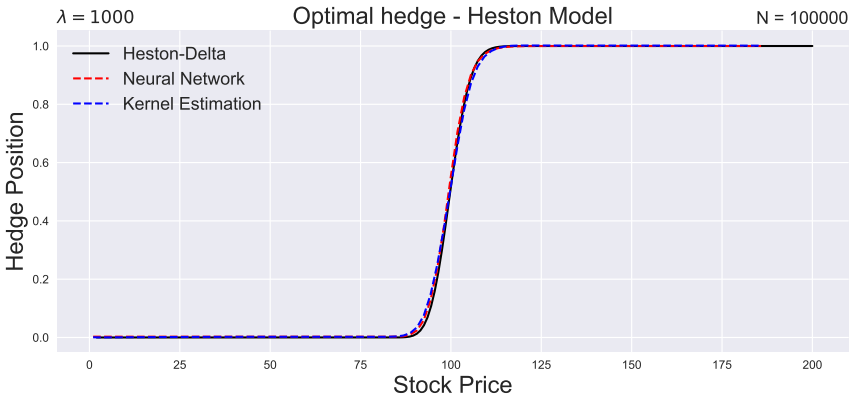
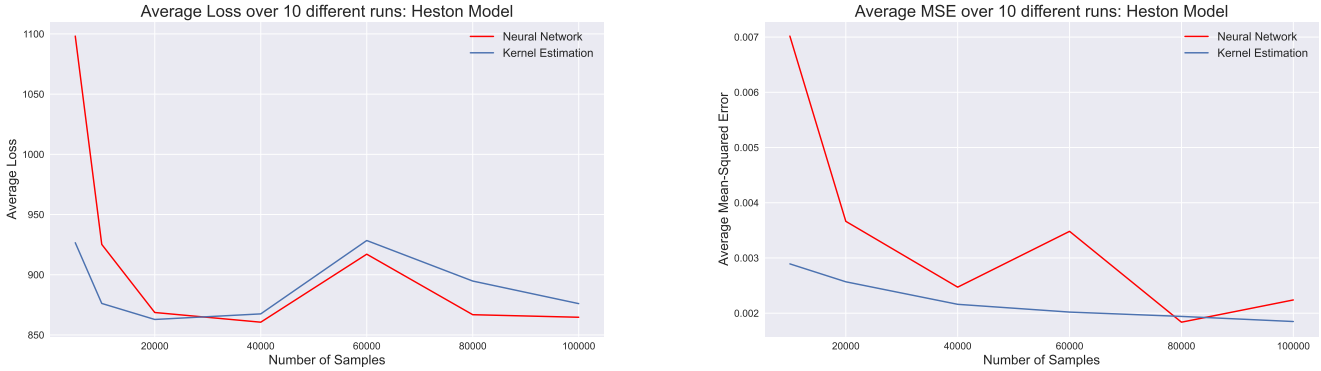


Figure 3.1: Optimal hedge under Heston Model



(a) Average loss over 10 different runs

(b) Average Mean-Squared Error

Figure 3.2: Loss and MSE over different sample sizes: Heston-Model

Figure 3.1 shows the optimal hedge at time $t = 0$ under the Heston model with parameters defined in (3.1.3) with 3 different methods including access to 100,000 different sample paths. The solid black line is the hedge computed using the closed-form solution for the delta in the Heston model. The red-dotted line is the hedge produced by a 3 layer neural network with 16 units in each layer using SELU in the hidden layers and sigmoid as the output activation function. From this figure, we can see that both Kernel Estimation and the Neural Network provide a reasonable convergence to the Heston-Delta.

Figure 3.2 compares the efficiency of Kernel estimation and the Neural Network just like we did in the previous chapter with Figure 2.23. In this case, we evaluate the hedges for initial stock prices in the domain [1,200]. Just like the Black-Scholes case, the neural network starts with an average loss much greater than that of kernel estimation. This is when the sample size is less than 20,000. For sample sizes greater than 40,000, the network has a smaller average loss than kernel estimation. As we increase the sample size, the average mean-squared error for kernel estimation is consistently improving but very slowly. For the network, the improvement is quicker but the error does fluctuate as we increase the sample size.

Let us now investigate this further as we did in the Black-Scholes case with a stock price domain of [70,130] and a sample size of 100,000 which we run 10 times. Let us see if kernel estimation provides better convergence than the network in this smaller domain.

	RMSE		MAE		MAPE		R^2	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
Kernel Estimation	0.00847	0.00104	0.00639	7.43×10^{-4}	6.58%	1.31%	0.9996	9.84×10^{-4}
Neural Network	0.0423	0.0184	0.0283	0.0111	16.5%	8.09%	0.988	0.00921

Table 3.1: Statistics of convergence to Heston Delta over 10 runs with sample size 100,000

The statistics in table 3.1 are very similar to those from 2.1. We can see here that the error produced by kernel estimation is far smaller than the errors generated by the neural network. Let us perform one last paired t-test of the absolute errors of the neural network and kernel estimation to see if there is any significant difference.

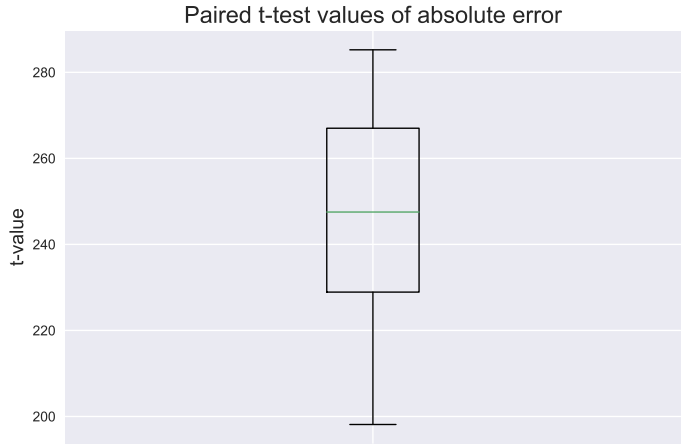


Figure 3.3: Boxplot of 10 paired test t-values of absolute error of hedge obtained via neural network and kernel estimation: Heston Setting

Figure 3.3 shows 10 t-values from the paired t-tests between the absolute error of the network and of kernel estimation. All 10 t-values are much greater than the 5% two-tailed critical value of 1.96. Just as in the Black-Scholes case, we can infer that hedge obtained by kernel estimation has a lower absolute error than the hedge obtained by the network in a Heston model setting.

3.2 GJR-GARCH: S&P500

We will now fit a GARCH(1,1,1) model to S&P 500 daily returns from 01/01/2018 to 30/08/2020 which corresponds to 669 return values. Let us first recall the volatility dynamics of a GARCH(1,1,1) which was originally introduced by Glosten et al.(1993) [27]:

$$r_t = \mu + \epsilon_t \tag{3.2.1}$$

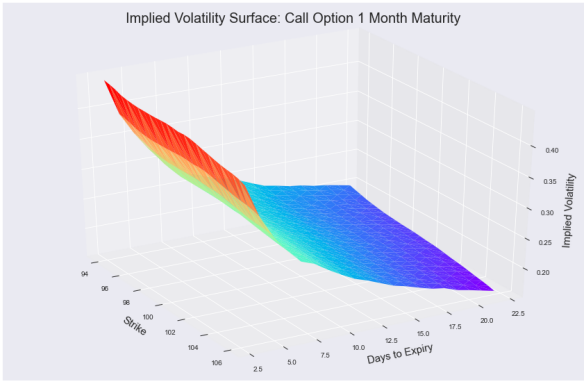
$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \gamma\epsilon_{t-1}^2 + \mathbb{1}_{\epsilon_{t-1}<0} + \beta\sigma_{t-1}^2 \tag{3.2.2}$$

$$\epsilon_t = \sigma_t Z_t \qquad Z_t \sim \mathcal{N}(0,1) \tag{3.2.3}$$

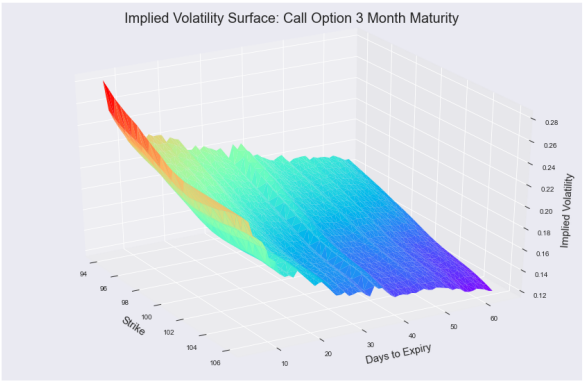
Where r_t is the returns, μ is the constant mean, and σ is the volatility of the stock. ω, α, β and γ are parameters of the model. After fitting our model to S&P500 returns, we obtain the following parameters:

$$\mu = 0.518,\ \omega = 0.0299,\ \alpha = 0.0201\ \gamma = 0.2664\ , \beta = 0.8252$$

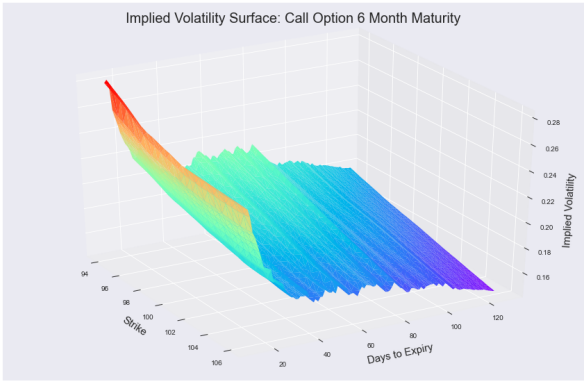
We then simulate 1000 stock price paths using our GARCH model and we use Kernel Estimation with $\lambda = 100000$ on equation (2.6.7) to hedge daily. Like before, we take the view of a European Call option seller now with 25 differing strikes, ranging from 94 to 106. Using our simulations and our hedge, we can obtain the portfolio value at time t using the backward-recursive formula (2.1.5). We can then use this as the "price" of the option and then feed this into the inverse equation for the BS-price to obtain the implied volatility. Below are the surfaces for maturities of 1 month, 3 months, 6 months and 1 year.



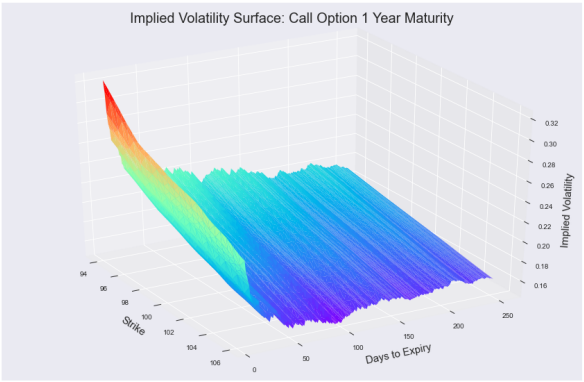
(a) 1 Month Maturity



(b) 3 Month Maturity



(c) 6 Month Maturity



(d) 1 Year Maturity

Figure 3.4: Implied volatility surfaces generated using GJR-GARCH and Kernel Estimation

Figure 3.4, is 4 surface plots of the implied volatility over, strike, and time to maturity. One can see the implied volatility skew/smile taking shape [23, Sec 49.5]. Note this was not computed using market prices of options. This used a GJR-GARCH model to produce samples of data in which we hedged using Kernel Estimation to generate the portfolio value at each time step. As stated in the Halperin paper on page 24, the volatility smile problem *does not exist* in Q-learning as it is data-driven and does not depend on a model. This could be an area of further interest and research. This could be an area of further interest and research

Chapter 4

Conclusion

In this paper, we applied theory set out in the paper by Igor Halperin[4]. This paper set out a model for discrete-hedging and pricing derivatives using Reinforcement Learning, in particular the famous and efficient Q-learning algorithm. In our paper, we focus on European Call options. This algorithm allows us to hedge in both a discrete-action and continuous-action space setting using only trading data. In the original paper by Halperin; one of the goals of the QLBS model was to provide a simple environment to test various Reinforcement Learning algorithms in discrete-time and discrete-space settings. We tested 3 different discrete-RL algorithms in this environment; Model-Based RL, Kernel-Based RL, and of course Q-learning. For this particular environment we found that Q-learning provided a relatively fast and efficient solution compared to the other algorithms. As we discussed earlier, the efficiency of the algorithm is dependent on the environment; for our Gridworld environment, Model-Based RL required much less computation time compared to Q-learning yet for our QLBS environment, Q-learning comes out on top. The major attractive feature of utilizing Q-learning in a financial setting is the capability of going model-free and being able to hedge without information on the model dynamics and requiring only data. The benchmark model used in this paper is the classical Black-Scholes model due to tractability and ease of computation of closed-form solutions.

In the Halperin paper, the optimal hedge was found through the optimization of the Bellman equations. The paper proposed to use a basis function expansion for the optimal hedge and Q-value functions. The optimization of the Bellman equations using basis functions led to the minimizing of equation (2.8.2), which is highly tractable and only requires linear algebra and Monte Carlo samples. However, one major problem is choosing an appropriate basis function and also the mathematical intuition behind adopting a particular function, especially if the model dynamics are unknown. This could potentially be an area of further research.

Nevertheless, one workaround to using conventional basis functions is to train a neural network to learn the optimal hedge by minimizing equation (2.8.2). This again, requires no information about the model dynamics, only requires market data to learn. However, the issue of choice still arises with neural networks, the ability to set up the architecture of a neural network is somewhat of a dark art. There is limited intuition as to why some functions are more proficient than others and how much data is required for the network to learn sufficiently.

Due to the quadratic nature of our Bellman Optimality equation (2.6.5), there exists an analytical solution in the form of conditional expectations, (2.6.7) which can be estimated using the Nadaraya-Watson estimator. We found that this estimator requires not many samples to provide a succinct result. We also found it to be far more efficient than the hedges predicted by our neural network. In the last section of our paper, we move away from a Black-Scholes model and test these methods within the Heston model to find that Kernel Estimation provides the most proficient solution yet again.

Further areas of interest include applying different algorithms such as the Deep Deterministic Policy Gradient (DDPG) algorithm [28] which utilizes both actor-critic and policy-gradient methods in a continuous-action space and is a model-free off-policy method. One could also look into including market externalities such as transaction costs, liquidity costs, funding costs, etc. As this model is data-driven, one could look into working in a multi-dimensional case and could use this environment to test many other models using only data.

Appendix A

Dynamic Programming (DP)

We will now evaluate a few algorithms which allow one to solve for the optimal policy when the model dynamics are *known*, i.e., probability transition matrix, and reward function are known. These algorithms utilize the Bellman equations.

A.1 Policy Evaluation Algorithm

We will first discuss what is known as the *prediction problem*. This algorithm allows one to compute the value function V_π for a given policy π. Recall that the value function can be expressed as:

$$V_\pi(s) = \mathbb{E}_t^\pi[G_t|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_\pi(s')]$$

The iterative algorithm is going to create a sequence of approximate value functions $V_\pi^0, V_\pi^1, V_\pi^2 \ldots$ until convergence. We arbitrarily initialize the approximate value function $V_\pi^0(s)$ for all s. The algorithm essentially uses the Bellman Expectation Equation (1.2.1) as an update rule for the $k + 1^{th}$ iteration:

$$V_\pi^{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_\pi^k(s')] \tag{A.1.1}$$

At each iteration, we update the value function for all states $s \in \mathcal{S}$. Technically, the value function only converges in the limit which is not possible computationally and must be cut short. One way to achieve this is to use a threshold and stop updating once the maximum of the difference between the value functions between iterations is sufficiently small, i.e. $\max_s |V_\pi^{k+1}(s) - V_\pi^k(s)| < \theta$ where θ is a small threshold.

Algorithm 2: Policy Evaluation Algorithm

Input: Policy π to be evaluated
Initialize $V_\pi(S)$ arbitrarily for all $s \in \mathcal{S}$
θ is a small threshold parameter for accuracy of approximation

while $\Delta < \theta$ **do**
  $\Delta \leftarrow 0$
  for *each* $s \in \mathcal{S}$ **do**
    $v \leftarrow V_\pi(S)$
    $V_\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_\pi(s')]$
    $\Delta \leftarrow \max\big(\Delta, |v - V_\pi(s)|\big)$
  end
end

We could amend amend this algorithm slightly to loop through every $s \in \mathcal{S}$ and every $a \in \mathcal{A}$ to evaluate the action-value function Q_π. The update rule would be based on equation (1.2.2) and works similarly to equation (A.1.1). This may be a better choice as it would allow one to perform policy improvement much more easily than using the state-value function.

A.2 Policy Improvement

The previous algorithm allowed us to compute the value functions for a given policy. However, it did not allow us to discover the optimal policy, which is the main goal of dynamic programming. We can use the previous technique of determining the value function to help us improve the policy. This is achieved by policy iteration in which we evaluate the policy and subsequently improve the policy, we repeat this process until we reach the optimal policy.

Theorem A.2.1 (Policy Improvement)**.** *Let π and π' be deterministic policies such that $Q_{\pi'}(s, \pi'(s)) \geq V_\pi(s)$, then $V_{\pi'}(s) \geq V_\pi(s)$.*

Proof.

$$
\begin{aligned}
V_\pi(s) &\leq Q_\pi\left(s, \pi'(s)\right) \\
&= \mathbb{E}\left[R_{t+1} + \gamma V_\pi\left(S_{t+1}\right) \mid S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}\pi'\left[R_{t+1} + \gamma V_\pi\left(S_{t+1}\right) \mid S_t = s\right] \\
&\leq \mathbb{E}\pi'\left[R_{t+1} + \gamma Q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) \mid S_t = s\right] \\
&= \mathbb{E}\pi'\left[R_{t+1} + \gamma \mathbb{E}\left[R_{t+2} + \gamma V_\pi\left(S_{t+2}\right) \mid S_{t+1}, A_{t+1} = \pi'\left(S_{t+1}\right)\right] \mid S_t = s\right] \\
&= \mathbb{E}\pi'\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 V_\pi\left(S_{t+2}\right) \mid S_t = s\right] \\
&\leq \mathbb{E}\pi'\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V_\pi\left(S_{t+3}\right) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}\pi'\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \cdots \mid S_t = s\right] \\
&= V_{\pi'(s)}
\end{aligned}
$$

[10][Section 4.2, pg 78] □

One way to select a new action is to act *greedily*, that is to select the action that produces the largest Q-value. The new greedily policy can be expressed as:

$$
\begin{aligned}
\pi'(s) &= \operatorname*{argmax}_a Q_\pi(s, a) && \text{(A.2.1)} \\
&= \operatorname*{argmax}_a \mathbb{E}_t[R_{t+1} + V_\pi(S_{t+1}) \mid S_t = s, A_t = a] && \text{(A.2.2)} \\
&= \operatorname*{argmax}_a \sum_{s'} \mathbb{P}(s', a \mid s, a)[r + V_\pi(s')] && \text{(A.2.3)}
\end{aligned}
$$

The policy iteration algorithm works by first evaluating an arbitrary deterministic policy and then improving the policy *greedily*, and we repeat this process until the policy is no longer changing thus has converged to the optimal policy. If $\pi(s)^{k+1} = \pi(s)^k$ (where the superscript represents the iteration) for all $s \in \mathcal{S}$, then the policy is no longer changing and has converged to the optimal policy π^*.

A.2.1 Value Iteration

One major downside of using policy iteration to obtain the optimal policy is the fact that policy evaluation has to be performed every iteration which can be computationally inefficient. Policy evaluation is an iterative process so exact convergence occurs only in the limit. In many cases, one does not need to wait for exact convergence to achieve the optimal policy. Value iteration improves the policy after each update of the value function. The update of the value iteration algorithm is based on the Bellman Optimality equation (1.2.6) in the following way with the superscript k as the iteration value:

$$V_*^{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*^k(s')] \tag{A.2.4}$$

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*(s')] \tag{A.2.5}$$

Algorithm 3: Value Iteration Algorithm

Initialize $V_*(S)$ arbitrarily for all $s \in \mathcal{S}$
θ is a small threshold parameter for accuracy of approximation

while $\Delta < \theta$ **do**
  $\Delta \leftarrow 0$
  for *each* $s \in \mathcal{S}$ **do**
    $v \leftarrow V_*(S)$
    $V_*(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*(s')]$
    $\Delta \leftarrow \max\big(\Delta, |v - V_*(s)|\big)$
  end
end
Output Policy: $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \gamma V_*(s')]$

Appendix B

Closed form solutions: Pricing and Hedging

B.1 Black-Scholes Model

We will now express the price and delta of a European Call option with strike K, spot S_t, volatility σ and risk-free rate r in the Black-Scholes model.

B.1.1 Pricing

The price of a European call option can be expressed as:

$$C = S_t N(d_1) - Ke^{-r(T-t)}N(d_2)$$

Where N$(\cdot)$ is the standard normal cdf and d_1, d_2 are:

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)\left(T-t\right)\right]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

B.1.2 Delta

The delta of a European Call Option in the Black-Scholes model can be expressed as:

$$\Delta = \frac{\partial C}{\partial S_t} = N(d_1)$$

B.2 Heston Model

We will now present the closed-form solutions to pricing and hedging in the Heston model as presented in Yang (2013) and Heston (1993) [29] [22]

B.2.1 Pricing

In 1993, Heston assumed that the price of a call option under the Heston model should be of the form:

$$C(S, \nu, t, T) = SP_1 - Ke^{-r(T-t)}P_2 \tag{B.2.1}$$

With spot price S at time t, strike K, constant risk-free rate r, initial volatility v and maturity T. P_1 and P_2 can be defined as a fourier inverse transform using characteristic functions f_1 and f_2 in the following way:

$$P_j = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left[\frac{e^{-i\varphi \ln K} f_j(x, v, \tau; \varphi)}{i\varphi}\right] d\varphi, \quad j = 1, 2 \tag{B.2.2}$$

Where $\tau = T - t$ and $x = \log(S_t)$
Heston assumed the solutions to the charcteristic functions as:

$$f_j\left(x, v, \tau; \varphi_j\right) = \exp\left\{C_j\left(\tau; \varphi_j\right) + D_j\left(\tau; \varphi_j\right) v + i\varphi_j x\right\}$$

Where

$$C_j\left(\tau; \varphi_j\right) = \mu\varphi_j\tau + \frac{a}{\sigma^2}\left\{\left(b_j - \rho\sigma\varphi_j i + d_j\right)\tau - 2\ln\left[\frac{1 - g_j e^{d,\tau}}{1 - g_j}\right]\right\}$$

$$D_j\left(\tau; \varphi_j\right) = \frac{b_j - \rho\sigma\varphi_j i + d_j}{b_j - \rho\sigma\varphi_j i - d_j}\left[\frac{1 - e^{d_j\tau}}{1 - g_j e^{d_j\tau}}\right]$$

$$g_j = \frac{b_j - \rho\sigma\varphi_j i + d_j}{b_j - \rho\sigma\varphi_j i - d_j}$$

$$d_j = \sqrt{\left(\rho\sigma\varphi_j i - b_j\right)^2 - \sigma^2\left(2u_j\varphi_j - \varphi_j^2\right)}$$

$$u_1 = 0.5, u_2 = -0.5, a = \kappa\theta$$

$$b_1 = \kappa + \lambda - \rho\sigma, b_2 = \kappa + \lambda$$

where κ is the mean-reversion rate and θ is the long-term mean.

B.2.2 Heston Delta

The Heston Delta can be expressed as:

$$\Delta = \frac{\partial C(S, v, t, T)}{\partial S} = P_1 + S\frac{\partial P_1}{\partial S} - K\frac{\partial P_2}{\partial S} \tag{B.2.3}$$

Where

$$\frac{\partial P_j}{\partial S} = \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left\{\frac{\partial\left[\frac{e^{-i\varphi \ln K} f_j(x, v, \tau; \varphi)}{i\varphi}\right]}{\partial S}\right\} d\varphi \qquad j = 1, 2$$

We can use the chain rule to express the above derivative as:

$$\frac{\partial P_j}{\partial S} = \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left\{\frac{\partial\left[\frac{e^{-i\varphi \ln K} f_j(x, v, \tau; \varphi)}{i\varphi}\right]}{\partial f_j(x, v, \tau; \varphi)} \cdot \frac{\partial f_j(x, v, \tau; \varphi)}{\partial S}\right\} d\varphi$$

Where

$$\frac{\partial f_j(x, v, \tau; \varphi)}{\partial S} = \frac{i\varphi}{S} \cdot f_j(x, v, \tau; \varphi)$$

Therefore,

$$\frac{\partial P_j}{\partial S} = \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left\{\frac{e^{-i\varphi \ln K} f_j(x, v, \tau; \varphi)}{i\varphi} \cdot i\varphi \cdot \frac{1}{S}\right\} d\varphi$$

$$= \frac{1}{\pi} \int_0^\infty \mathrm{Re} \left\{ \frac{e^{-i\varphi \ln K} f_j(x, v, \tau; \varphi)}{S} \right\} d\varphi \tag{B.2.4}$$

We can then substitute these derivatives from eqn (B.2.4) into eqn (B.2.3) to obtain the closed-form solution to the Heston Delta:

$$\Delta H = P_1 + \frac{S}{\pi} \int_0^\infty Re \left\{ \frac{e^{-i\varphi \ln K} f_1(x, v, \tau; \varphi)}{S} \right\} d\varphi - \frac{S}{\pi} \int_0^\infty Re \left\{ \frac{e^{-i\varphi \ln K} f_2(x, v, \tau; \varphi))}{S} \right\} d\varphi \tag{B.2.5}$$

Bibliography

[1] Chris Watkins and P K Dynan. Q-learning in machine learning. 1989.

[2] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science*, 3(12):1337–1344, 2017. PMID: 29296675.

[3] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *ArXiv*, abs/1911.08265, 2019.

[4] Igor Halperin. Qlbs: Q-learner in the black-scholes (-merton) worlds. *Econometrics: Mathematical Methods and Programming eJournal*, 2017.

[5] `https://gym.openai.com/`.

[6] Fischer Black and Myron S. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[7] `https://www.bis.org/publ/otc_hy1911.htm`.

[8] Paul Wilmott. Derivatives: The theory and practice of financial engineering. 1998.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

[10] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, second edition. *IEEE Transactions on Neural Networks*, 2018.

[11] Richard Bellman. Dynamic programming. *Science*, 1957.

[12] Nicholas K. Jong and Peter Stone. Kernel-based models for reinforcement learning. 2006.

[13] Marc Potters, Jean-Philippe Bouchaud, and Dragan Sestovic. Hedged monte-carlo: Low variance derivative pricing with objective probabilities. 2001.

[14] Alan Roy Stuart and Harry M. Markowitz. Portfolio selection: Efficient diversification of investments. *A Quarterly Journal of Operations Research*, 10:253, 1959.

[15] Catherine F. Higham and Desmond J. Higham. Deep learning: An introduction for applied mathematicians, 2018.

[16] Mikko Pakkanen. Deep learning: Lecture notes, 2019/2020.

[17] `https://www.doc.ic.ac.uk/~nuric/teaching/imperial-college-machine-learning-neural-`
`html`.

[18] Guenter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *ArXiv*, abs/1706.02515, 2017.

[19] E. A. Nadaraya. On estimating regression. 1964.

[20] Geoffrey Watson. Smooth regression analysis. 1964.

[21] Bernard W. Silverman. Density estimation for statistics and data analysis. 1986.

[22] S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 1993.

[23] Paul Wilmott. *Paul Wilmott on Quantitative Finance 2nd Edition*. John Wiley & Sons, 2006.

[24] Tae Kwon. Particle filtering of volatility dynamics for kospi200 and its sequential prediction. *Journal of Forecasting*, 37, 08 2018.

[25] M. Johannes, N. Polson, and Jonathan R. Stroud. Optimal filtering of jump diffusions: Extracting latent states from asset prices. *Review of Financial Studies*, 22:2759–2799, 2009.

[26] Gurdip Bakshi, Charles Cao, and Zhiwu Chen. Option pricing and hedging performance under stochastic volatility and stochastic interest rates. 2010.

[27] L. Glosten, R. Jagannathan, and D. Runkle. On the relation between the expected value and the volatility of the nominal excess return on stocks. 1993.

[28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[29] Y. Yang. Valuing a european option with the heston model. 2013.