UiO : Department of Mathematics
University of Oslo

Reinforcement learning and stochastics with applications in mathematical finance

Eva Steine Dahl

Master's Thesis, Spring 2020

Abstract

The topic of this thesis is stochastic optimal control and reinforcement learning. Our aim is to unify the theory and language used in the two fields. The thesis presents both frameworks and discuss similarities, differences and how the reinforcement learning framework can be extended to include elements from the Hamilton-Jacobi Bellman equations. In the second part of the thesis, this theory is used in order to price exotic options in energy markets. We also use the HJB-equations and the Q-learner as an update rule to look at problems from portfolio optimization.

Acknowledgements

First and foremost, I would like to thank my supervisor Kristina Rognlien Dahl. Her constant support, advise and optimism has not only been crucial for the creation of this thesis, it has also been a great source of comfort through the process of making it. I am very grateful for having had such a competent and kind supervisor. I would also like to thank Fred Espen Benth for sharing his knowledge during my summer project, and for introducing me to the exciting field of power markets.

I want to thank my family for their moral support and encouragements. A special thanks to my mother for allowing me to raid her fridge every other weekend, and for solving crosswords with me when the last thing I wanted to see was another equation.

Last, but far from least, I want to thank all the great people that I have met throughout my years at the faculty of mathematics. From the students that made my academic years fun, the professors who made them challenging, and the people at the administration office who made them possible. They all played their part in making my years at UiO memorable.

Contents

List of Figures

Introduction

The theme of the thesis

The topic of this thesis is control problems, studied in two different fields known as stochastic optimal control and reinforcement learning (RL). Both fields study problems of decision making in a stochastic environment over time, and both are based on the Bellman equations. Still, they differ in a number of ways, the main ones being:

1. The theory of stochastic optimal control is primarily studied by mathematicians and physicists, and the theoretical framework is therefore mathematically rigorous. Reinforcement learning, on the other hand lies in the intersection between informatics and statistics. It also has great applicational value in the field of robotics, and is thus also studied by engineers. This has lead to the RL-literature often being based on examples of successful application rather than mathematical proofs.

2. Reinforcement learning uses algorithms that performs updates in a learning environment.

3. Stochastic optimal control deals with continuous settings, while reinforcement learning is primarily dealing with discrete settings.

The goal of the thesis is to present the setting of reinforcement learning in a more formalized way, and to reconnect the field of reinforcement learning with the original mathematical framework from which it originally grew. A problem with many papers regarding RL is that they deal mostly in empirics. They show what might work, but many fail to explain under which conditions something *must* work. This makes for a field in which trial and error is a large part of the modus operandi.

Throughout the thesis, we look at examples involving mathematical finance and portfolio optimization. This is a field which has been studied using stochastic analysis for many years, but it has in recent years gained the interest of machine learning specialists looking for more data driven approaches. We will consider some simple examples involving optimization of portfolios, but also more complex cases of pricing financial products based on several underlying dynamics.

The work process

The work process has been as follows. Initially, we started by looking at the field of stochastic control, and in particular the Hamilton-Jacobi-Bellman equations (HJB equations), and their proofs. This was a completely new field for this author, and therefore turned out to be quite time consuming. The field of reinforcement learning was investigated next. Reinforcement learning is a wide field, and there are numerous approaches, with even more tweaks and variations, each assumed to be fitting to it's own branch of problems. Navigating this jungle of different approaches, each with their own variations of the notation, showcased both the strengths and weaknesses of the reinforcement learning framework. Lastly, we looked at applications within the field of finance.

There are three primary sources that has formed the basis for the underlying theory of this thesis. The stochastic analysis and optimal control is based on Øksendal's approach in [Øks03]. The theory of reinforcement learning is primarily based on Sutton's work in [SB18], and the pricing of exotic options leans on theory from Benth in [BBK08]. In addition, the method described in Chapter 5 is based on a paper by Halperin [Hal17]. We have also been looking at a number of papers, particularly on the subject of reinforcement learning. This was necessary, since continuous state models in reinforcement learning is a current field of study, and new result are still being produced.

Structure of the thesis

The thesis is structured as follows:

1. Chapter 1 contains the mathematical framework necessary for stochastic optimal control, and the framework of the HJB-equations.

2. Chapter 2 presents the framework of the HJB-equations, in addition to an example of application to finance.

3. Chapter 3 presents the field of reinforcement learning, and the algorithm known as Q-learning.

4. In Chapter 4, we discuss similarities and differences of the two approaches in more detail. We also discuss whether we can use the theoretical framework from HJB to formalize the convergence in RL.

5. Chapter 5 presents a case from the intersection of reinforcement learning and mathematical finance. Here we study portfolio optimization, a problem which has been solved by use of stochastic optimal control, but which in recent years also has been explored through machine learning.

My contributions

Through the thesis, we will use the symbol $\diamondsuit$ to signal the authors own contributions. The following list highlights the main contributions in each chapter.

Chapter 1:

- Example 1.2.5 and Example 1.4.11: Did computations and wrote code.

In addition, remarks and interpretations revolving the theorems in this chapter are my own.

Chapter 2:

- Example 2.4.1: Showed simple example of stopping time.
- Proof of Theorem 2.6.2 and Theorem 2.6.4: Filled in missing details, references and computations.
- Example 2.6.5: Added explanations, computations and details.

In addition, remarks and interpretations in this chapter are my own.

Chapter 3:

- Figure 3.1: Created timeline.
- Example 3.3.1: Wrote algorithm and code.
- Example 3.4.1: Created textual context to example by Sutton [SB18], and added interpretation.

In addition to this, I have interpreted and summarized theory from multiple sources in the rest of the sections.

Chapter 4:

This chapter is primarily my own analysis, summarizing and drawing connections between the two frameworks.

Chapter 5:

- Section 5.2: The analysis of Halperin's framework in relation to Q-learning is by me.
- Section 5.3: The setup of the dynamics of the exotic option is by me, by an idea from Fred Espen Benth.
- Section 5.4: Extended Halperin's approach [Hal17] for an exotic option type.
- Section 5.4: Added computation of the variance of the portfolio.
- Section 5.5: Did additional computations for $Q^*(\cdot)$, and added some details and remarks.
- Section 5.6: Wrote all code, and did the numerical analysis.
- Section 5.8: Did all computations.
- Section 5.8: Did all computations.

In addition to this, I have interpreted and summarized theory from multiple sources in the rest of the sections.

CHAPTER 1

Preliminaries

1.1 Introduction

We will start by introducing some concepts, definitions and theory from the field of stochastic analysis. *Stochastic analysis* is the study of stochastic processes, which means processes driven in part by random noise. The concepts introduced here will lay the foundation for the Hamilton-Jacobi-Bellman equations described in Chapter 2. All the definitions in this section are taken from Øksendal [Øks03].

We will present the general framework of the problems studied in stochastic analysis. These will give us a way of formalizing problems involving processes under uncertainty. For these types of problems, some essential questions needs to be answered:

1. What are the possible ways that the process can evolve, i.e. which states, or values, can the process take?
2. How does the process evolve over time?
3. What information regarding the states and their history is available to us at each time step?

The first part of this chapter will provide us with the tools to describe information regarding these questions in a formal way.

The second part will present the random noise through the introduction of the Brownian motion. We will also present how we can integrate over processes regarding uncertainty, by introducing the Itô integral. Here we will present the constraints underlying these integrals, and some tools for solving them. The last part will describe stochastic differential equations and their solutions. This will provide us with important tools for analyzing the stochastic processes, and for performing optimal decisions in stochastic environments.

1.2 Spaces, measurable sets and probability theory

Modeling a stochastic system

For any problem involving stochastic processes, we will need a way of formalizing what values, or states, the process is possibly going to take. To model the states of the world or system we are looking at, we will let Ω denote the (possibly infinite) set of all possible states. On this set, we define a σ-algebra:

Definition 1.2.1 (σ-algebra)**.** If Ω is a given set, then a *σ-algebra* $\mathcal{F}$ on Ω is a family of subsets on σ with the following properties:

1. $\emptyset \in \mathcal{F}$,
2. $F \in \mathcal{F} \implies F^c \in \mathcal{F}$, where $F^c = \Omega \backslash F$, and
3. $A_1, A_2, \dots \in \mathcal{F} \implies \bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

In the context of this thesis, $F \in \mathcal{F}$ can be viewed as events, or states of the world. In this way we can consider Ω as representing all the possible states of the system we are considering. The smallest σ-algebra that can be constructed from any given family can be defined as the union of all σ-algebras of the set. More typically, however, we will start with an event(or a family of events), and construct the σ-algebra by adding subsets until it satisfies 1. – 3. in Def. 1.1.1. For a set $\mathcal{U}$ of events in Ω, we therefore denote the smallest σ-algebra $\mathcal{H}_{\mathcal{U}}$, *the σ-algebra generated by $\mathcal{U}$*.

The *Borel σ-algebra* is the smallest σ-algebra of all open subsets of a topological space. The elements of this set are called the *Borel sets*.

Example 1.2.2 (Borel sets)**.** For the topological space $\mathbb{R}$ the Borel σ-algebra is the smallest σ-algebra that contain all open intervals.

A set with a σ-algebra assigned to it, $(\Omega, \mathcal{F})$, is called a *measurable space*. On these types of spaces we can introduce probability measures:

Definition 1.2.3 (Probability measure)**.** A *probability measure* P on a measurable space $(\Omega, \mathcal{F})$ is a function $P : \mathcal{F} \to [0, 1]$ such that

1. $P(\emptyset) = 0, P(\Omega) = 1$.
2. If $A_1, A_2, \dots \in \mathcal{F}$ and $\{A_i\}_{i=1}^{\infty}$ are disjoint (i.e. $A_i \cap A_j = \emptyset$ for $i \neq j$) then $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$.

Probability measures lets us assign probabilities to different sets of states. We call $(\Omega, \mathcal{F}, P)$ a *probability space*. It is called *complete* if $\mathcal{F}$ contains all subsets G of Ω with P outer measure zero, i.e.

$$P^*(G) := \inf\{P(F); G \subset F \in \mathcal{F}\} = 0.$$

In our case, complete implies that all the events with probability zero are contained in $\mathcal{F}$. We call the subsets of $\mathcal{F}$ the *$\mathcal{F}$-measurable sets*.

Let $X : \Omega \to \mathbb{R}^n$ be a function. The *σ-algebra generated by X* is the set of events in Ω that is mapped to the Borel sets by X.

We want to look at dynamic systems over time, where the information that is available to us increases over time. To model this, we introduce the concept of a filtration:

Definition 1.2.4 (Filtration)**.** A *filtration* on the probability space $(\Omega, \mathcal{F}, P)$ is a family $\mathcal{M} = \{\mathcal{M}_t\}_{t \geq 0}$ of σ-algebras $\mathcal{M}_t \in \mathcal{F}$ such that

$$0 \leq s < t \implies \mathcal{M}_s \subset \mathcal{M}_t. \tag{1.1}$$

We model the amount of information available to us at time t by $\mathcal{M}_t$, i.e. the σ-algebra containing all the events that are possible at this time. We call the subsets of $\mathcal{M}_t$ the *$\mathcal{M}_t$-measurable sets*.

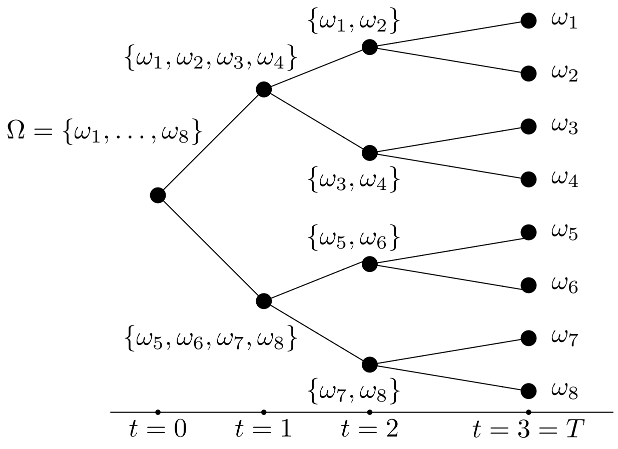

Figure 1.1: Scenario tree for t = 3.

Example 1.2.5 (◇ Measurable set)**.** If you own a stock, you are often interested in whether the price of the stock will go up or down in the future. This can be modeled using the framework described above. We look at a simplified case in discrete time. We denote a price increase by U and a price decrease by D. If we look at a time interval of three days we will have a probability space Ω consisting of the events:

$$\begin{array}{lll} \omega_1 = UUU, & \omega_4 = UDU, & \omega_7 = DUD, \\ \omega_2 = UUD, & \omega_5 = DUU, & \omega_8 = DDD, \\ \omega_3 = UDD, & \omega_6 = DDU, & \end{array}$$

so that we get $\Omega = \{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5,\omega_6,\omega_7,\omega_8\}$.

This can also be represented by a scenario tree, (see Figure 1.1), where we can see the branches as representation of the information in the filtration at each time $t = \{0, 1, 2, 3\}$. Here each node can be seen as representing the information we have about possible states at time t. At $t = 0$ we have not yet seen any stock movement, and therefore there is a positive probability for any of the 8 events. At $t = 1$ we have observed the first price change, and we are therefore able to tell whether we are in the lower or top branch in the tree. When we are at $t = 3$ all information is available to us, and we are in a deterministic state of complete information. This partition generates a filtration which contains the complete set of information we have at each time t.

1.3 Stochastic processes

We will use the concept of a stochastic process to model the change of a (stochastic) system over time.

Definition 1.3.1 (Stochastic process)**.** Let T be an interval on $[0,\infty)$. A *stochastic process* is a parametrized collection of random variables

$$\{X_t\}_{t\in T},$$

defined on the probability space $(\Omega, \mathcal{F}, P)$ and assuming values in $\mathbb{R}^n$.

The parameter space T represents the time interval on which we observe the stochastic process, and unless otherwise specified we will define T as $[0, \infty)$ in this thesis.

One type of such process, which will play an important role in the following chapter, is the Brownian motion:

Definition 1.3.2 (1-dimensional Brownian motion)**.** A stochastic process $\{B_t\}_{t\geq 0}$ on the probability space $(\Omega, \mathcal{F}, P)$ is called a (standard) *Brownian motion* iff

1. $B_0 = 0$ with probability 1 almost everywhere.
2. The paths $(t \mapsto B_t(\omega))$ are continuous.
3. B has independent increments, meaning that $B_{t_1} - B_{t_0}, B_{t_2} - B_{t_1}, \dots, B_{t_n} - B_{t_{n-1}}$ are independent for $0 \leq t_1 < t_2 \cdots < t_n$.
4. B has normal stationary increments, i.e. that $B_t - B_s$ has the same distribution as B_{t-s} and $B_{t-s} \sim N(0, t-s)$.

This definition can be generalized for the multi-dimensional case. It can also be shown that a standard Brownian motion has expectation 0 and variance t. The expected value is found by using Item 1 and Item 3:

$$E[B_t] = [E[B_t - B_0 + B_0] = E[B_t - B_0] + E[B_0] = 0 + B_0 = 0. \tag{1.2}$$

By using the standard formula for the variance in combination with the result above and 3., we get the result for the variance:

$$Var(B_t) = E[B_t^2] - E[B_t]^2 = E[B_t^2] = t, \tag{1.3}$$

(where we for the last equality use that Item 3 with $s = 0$ implies $E[B_t^2] = t$.) A plot of five paths of the Brownian motion can be seen in Figure 1.2, where the code can be found in Appendix A.1. For each time point the Brownian motion's increment is normally distributed with expectation equal to zero. By summing these increments we get the path of the Brownian motion.

A Brownian motion is a type of martingale:

Definition 1.3.3 (Martingales)**.** Let $\{\mathcal{F}_t\}_{t\geq 0}$ be a filtration on $(\Omega, \mathcal{F}, P)$. A stochastic process $X_t : \Omega \to \mathbf{R}$ is called a *martingale* if

$$X_t = E[X_s|\mathcal{F}_t], \quad \forall s > t. \tag{1.4}$$

The term was originally used by gamblers, who used the phrase "to play the martingale" on the strategy of always betting the double amount as your loss in coin-toss games [Man]. If the coin process truly is a martingale, one would assume that in the long run, the winnings and losses would even out, as a fair coin has a 50/50 probability of landing on either side. Most gamblers, however, greatly underestimated the amount of losses one would have to account for in the short run.

Example 1.3.4 (Brownian motion is a martingale $\diamondsuit$)**.** It can be shown that a Brownian motion is a martingale. Since $E[B_t - B_s] \sim N(0, t-s)$, we have that $E[B_t] < \infty$ and that

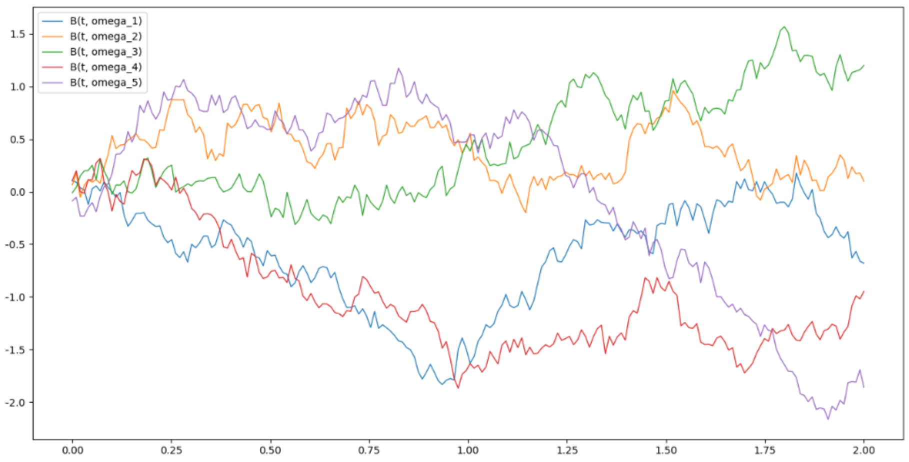

Figure 1.2: Plot of Brownian motions.

$$\begin{aligned}E[B_t|\mathcal{F}_s] &= E[B_t - B_s + B_s|\mathcal{F}_s]\\ &= E[B_t - B_s|\mathcal{F}_s] + E[B_s|\mathcal{F}_s]\\ &= B_s,\end{aligned}$$

where we for the third equality use that the Brownian motion have increments with expectation zero, and that $E[B_s|\mathcal{F}_s] = B_s$.

We can generate a filtration based on the Brownian motion in a rather intuitive way. The σ-algebra generated by the Brownian motion up to time t will be

$$\mathcal{F}_t = \{\sigma(B_s : s \leq t)\} \tag{1.5}$$

We see that $\{\mathcal{F}_t\}_{t\geq 0}$ will be increasing, so this family will be a filtration. This filtration will give us a way of representing the information we have based on the Brownian motion, and it implies that the Brownian motion at each time t is $\mathcal{F}_t$-measurable. Another way of saying this is that the Brownian motion is *adapted* to the filtration $\mathcal{F}_t$. In the discrete case, this information can also be represented as a scenario-tree, as seen in Figure 1.1. In this case, each branch would represent a state of the Brownian motion for some time step.

1.4 The stochastic integral

Looking at the plots of Brownian motions in Figure 1.2, it is natural to ask whether these types of processes are integrable and differentiable. To this end, we will now give a formal definition of the *Itô integral*, which for some Itô-integrable function f (to be specified) is denoted

$$\int_S^T f(t,\omega)dB_t(\omega). \tag{1.6}$$

It is natural to assume that this integral would be defined as the limit of the sum of a partition with mesh approaching zero:

$$\lim_{\Delta t \to 0} \{f(t_i, \omega)(B(t_{i+1}) - dB(t_i))\}.$$

However, it turns out that the paths of the Brownian motion has too big variation to define the integral without first restraining ourselves to a specific class of functions:

Definition 1.4.1 (Class of integrable functions)**.** Let $\mathcal{V} = \mathcal{V}(S,T)$ be the class of functions

$$f(t,\omega)\colon [0,\infty) \times \Omega \to \mathbb{R}, \tag{1.7}$$

such that

1. $(t,\omega) \to f(t,\omega)$ is $\mathcal{B} \times \mathcal{F}$-measurable, where $\mathcal{B}$ is the Borel σ-algebra on $[0,\infty)$.
2. $f(t,\omega)$ is $\mathcal{F}_t$-adapted.
3. $E[\int_S^T f(t,\omega)^2 dt] < \infty$.

It can be proved [Øks03] that there always will exist a sequence of elementary functions that can approximate functions of this class:

Definition 1.4.2 (Elementary functions)**.** A function $\phi \in \mathcal{V}$ is called *elementary* if it has the form

$$\phi(t,\omega) = \sum_j e_j(\omega) \mathbf{1}_{[t_j, t_{j+1})}(t), \tag{1.8}$$

where $\mathbf{1}_{[t_j,t_{j+1})}(t)$ is the indicator function which is 1 when t is in the interval $[t_j, t_{j+1})$ and 0 otherwise, and e_j is any $\mathcal{F}_{t_j}$-measurable function.

We will use the L^p-norm in order to describe the magnitude, i.e. the size, of a process:

Definition 1.4.3 (The L^p-norm)**.** If $X\colon \Omega \to \mathbb{R}^n$ is a random variable and $p \in [1,\infty)$ is a constant we define the L^p-norm of X, denoted $\|X\|_p$, by

$$\|X\|_p = \|X\|_{L^p(P)} = (\int_\Omega |X(\omega)|^p dP(\omega))^{\frac{1}{p}}.$$

If $p = \infty$, we set

$$\|X\|_\infty = \|X\|_{L^\infty(P)} = \inf\{N \in \mathbb{R}\colon |X(\omega)| \le N \quad a.s.\}.$$

We now have the framework needed to define the stochastic integral:

Definition 1.4.4 (The Itô integral)**.** Let $f \in \mathcal{V}(S,T)$. Then the *Itô integral* of f is defined by

$$\int_S^T f(t,\omega) dB_t(\omega) = \lim_{n\to\infty} \int_S^T \phi_n(t,\omega) dB_t(\omega), \tag{1.9}$$

where the limit is in the $L^2(P)$-norm, and where $\{\phi_n\}_{n=1}^{\infty}$ is a sequence of elementary functions such that

$$E[\int_S^T (f(t,\omega) - \phi_n(t,\omega))^2 dt] \to 0 \quad \text{as } n \to \infty. \tag{1.10}$$

A useful property of the Itô integral is the *Itô isometry*:

Theorem 1.4.5 (The Itô isometry)**.**

$$E[(\int_S^T f(t,\omega)dB_t(\omega))^2] = E[\int_S^T f^2(t,\omega)dt], \tag{1.11}$$

for all $f \in \mathcal{V}(S,T)$*.*

Theorem 1.4.6 (Properties of the Itô integral)**.** *Let* $\phi, \theta \in \mathcal{V}(0,T)$*,* α, β *be constants and* $0 \leq S < U < T$*. Then:*

1. $\int_S^T \phi(t,\omega)dB_t(\omega) = \int_S^U \phi(t,\omega)dB_t(\omega) + \int_U^T \phi(t,\omega)dB_t(\omega)$ *with probability* 1 *almost everywhere.*
2. $\int_S^T (\alpha\phi(t,\omega) + \beta\theta)dB_t(\omega) = \alpha \int_S^T \phi(t,\omega)dB_t(\omega) + \beta \int_S^T \theta dB_t(\omega)$ *with probability* 1 *almost everywhere,* $\alpha, \beta \in \mathbb{R}$*.*
3. $E[\int_S^T \phi(t,\omega)dB_t(\omega)] = 0$.
4. $\int_S^T \phi(t,\omega)dB_t(\omega)$ *is* $\mathcal{F}$*-measurable.*

One problem with Itô integrals is that they are not stable under smooth maps:

Definition 1.4.7 (Smooth maps)**.** For $n \in \mathbb{Z}^+ \cup \{\infty\}$, we let

$$C^n([0,\infty) \times \mathbb{R}) \tag{1.12}$$

be defined as the space of functions that are n times continuously differentiable on $[0,\infty) \times \mathbb{R}$, with continuous extensions of the partial derivative on $(0,\infty) \times \mathbb{R}$.

A mapping by a twice continuously differentiable function of a Itô integral is not necessarily an Itô integral. If we introduce the stochastic integral, also called an *Itô process*, we can ensure this. It is defined as follows:

Definition 1.4.8 (Itô processes)**.** A stochastic process $X_t : [0,\infty) \times \Omega \to \mathbb{R}$ is called a (1-dimensional) Itô process if

$$X_t = X_0 + \int_0^t u(s,\omega)ds + \int_0^t v(s,\omega)dB_s,$$

where $v \in \mathcal{V}_{\mathcal{H}}$ and u is $\mathcal{F}_t$-adapted, with $E[\int_0^t |u(s,\omega)|ds] < \infty$, $t \geq 0$.

The Itô process is the solution to the *stochastic differential equation* (SDE) defined by

$$\frac{dX_t}{dt} = u(t,\omega) + v(t,\omega)B_t, \tag{1.13}$$

where $u(s,\omega)$ and $v(s,\omega)$ is defined as in Definition 1.4.8. We will write this as its differential form $dX_t = udt + vdB_t$ when it does not create confusion.

One of the main tools for solving such stochastic differential equations is the *Itô's Lemma.*

Theorem 1.4.9 (Itô's lemma)**.** *Let X_t be an Itô process given by*

$$dX_t = udt + vdB_t, \tag{1.14}$$

and let $g \in C^2([0,\infty)\times\mathbb{R})$ (i.e g twice continuously differentiable on $(0,\infty)\times\mathbb{R}$ with continuous extensions of the partial derivative on $[0,\infty)\times\mathbb{R}$). Then $Y_t := g(t, X_t)$ is an Itô process with

$$Y_t = Y_0 + \int_0^t \frac{\partial}{\partial t} g(s,X_s)ds + \int_0^t \frac{\partial}{\partial x} g(s,X_s)dX_s + \frac{1}{2}\int_0^t \frac{\partial^2}{\partial x^2} g(s,X_s)v_s^2 ds, \tag{1.15}$$

where

$$\int_0^t \frac{\partial}{\partial x} g(s,X_s)dX_s = \int_0^t \frac{\partial}{\partial x} g(s,X_s)u_s ds + \int_0^t \frac{\partial}{\partial x} g(s,X_s)v_s dB_s. \tag{1.16}$$

That is, $dY_t = \tilde{u}dt + \tilde{v}dB_t$, where

$$\tilde{u}(s,\omega) = \frac{\partial}{\partial t} g(s,X_s(\omega)) + \frac{1}{2}\frac{\partial^2}{\partial x^2} g(s,X_s(\omega))v^2(s,\omega) + \frac{\partial}{\partial x} g(s,X_s(\omega))u(s,\omega), \tag{1.17}$$

$$\tilde{v}(s,\omega) = \frac{\partial}{\partial x} g(s,X_s(\omega))v(s,\omega). \tag{1.18}$$

Remark 1.4.10. We note that Itô's lemma can in some sense be seen as a stochastic version of the chain rule, where we need to add an extra correction term (the last part of Equation (1.15)) because of the large variation of the Brownian motion.

Example 1.4.11 (Geometric Brownian motion $\diamondsuit$)**.** In Figure 1.3 we see a plot of the path of a *geometric Brownian motion* for 5 different states. This is a stochastic process with drift, meaning that the expected value changes over time. It is the solution to the stochastic differential equation

$$dX(t) = \mu X(t)dt + \sigma X(t)dB(t).$$

The geometric Brownian motion in the plot have $\mu = 0.1$ and $\sigma = 0.1$. We have also plotted the expected value as the dotted line, given by $X(0)\exp(\mu t)$. The code can be found in Appendix A.2.

1.5 Itô diffusions

A stochastic process which solves a stochastic differential equation is called an *Itô diffusion.* In this section, we describe some properties of such solutions, and under which assumptions they apply.
We will first establish for which cases the solution of such a stochastic process exists, and for which cases they are unique:

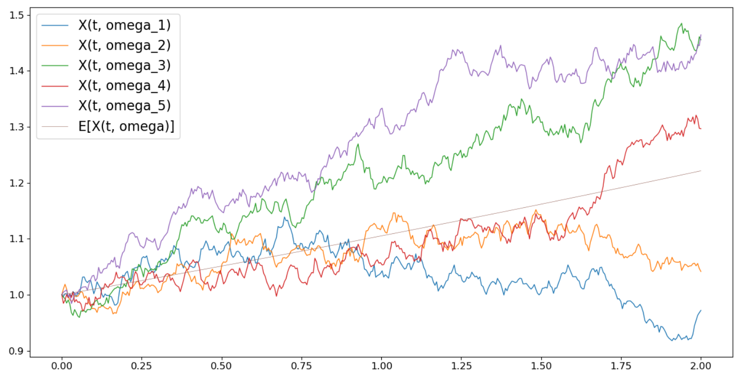

Figure 1.3: Plot of geometric Brownian motions with drift $\mu = 0.1$ and volatility $\sigma = 0.1$.

Theorem 1.5.1 (Existence and uniqueness theorem for SDE's). *Let $T > 0$, and $b(\cdot,\cdot) : [0,T] \times \mathbb{R}^n \to \mathbb{R}^n$ and $\sigma(\cdot,\cdot) : [0,T] \times \mathbb{R}^n \to \mathbb{R}^{n\times m}$ be measurable functions satisfying*

$$|b(t,x)| + |\sigma(t,x)| \leq C(1+|x|); \quad x \in \mathbb{R}^n, t \in [0,T], \tag{1.19}$$

for some constant C, (where $|\sigma|^2) = \sum |\sigma_{ij}|^2$) and such that

$$|b(t,x) - b(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq D|x-y|; \quad x,y \in \mathbb{R}^n, t \in [0,T], \tag{1.20}$$

for some constant D. Let Z be a random variable which is independent of the σ-algebra $\mathcal{F}_\infty^{(m)}$ generated by $B_s(\cdot)$, $s \geq 0$, and satisfying

$$E[|Z|^2] < \infty. \tag{1.21}$$

Then the stochastic differential equation

$$dX_t = b(t,X_t)dt + \sigma(t,X_t)dB_t, \quad 0 \leq t \leq T, \tag{1.22}$$
$$X_0 = Z \tag{1.23}$$

has a unique t-continuous solution $X_t(\omega)$ with the property that $X_t(\omega)$ is adapted to the filtration $\{\mathcal{F}_t^Z\}$ generated by Z and $B_s(\cdot)$, for $s < t$, and

$$E\Big[\int_0^T |X_t|^2 dt] < \infty. \tag{1.24}$$

Theorem 1.5.1 essentially says that if a stochastic differential equation satisfies the conditions of maximum linear growth (Equation (1.19)), and Lipschitz continuity (Equation (1.20)), then there will exist a solution, and this solution will be unique.

The time-homogeneous case of solutions is defined as follows:

Definition 1.5.2 (Itô diffusions)**.** A (time-homogeneous) Itô diffusion is a stochastic process $X_t(\omega) = X(t,\omega) : [s,\infty) \times \Omega \to \mathbb{R}^n$ satisfying a stochastic differential equation of the form

$$dX_t = b(X_t)dt + \sigma(X_t)dB_t, \quad t \geq s; X_s = x, \tag{1.25}$$

where B_t is an m-dimensional Brownian motion and $b : \mathbb{R}^n \to \mathbb{R}$ and $\sigma : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ satisfy the conditions from Theorem 1.5.1. In this case the conditions simplify to

$$|b(x) - b(y)| + |\sigma(x) - \sigma(y)| \leq D|x - y|, \quad x, y \in \mathbb{R}^n, \tag{1.26}$$

i.e. that $b(\cdot)$ and $\sigma(\cdot)$ are Lipschitz continuous.

Notation 1.5.3. We will use the notation $X_t^{s,x}$, for $t \geq s$, when denoting the solution to the SDE in Equation (1.25). When $X_t(\omega)$ is evaluated on $[0,\infty)$ we will simply write X_t^x.

We see that the functions b and σ no longer depend directly on t. This is essential for the next theorem, which shows that the process X_t now satisfies the *Markov property*:

Theorem 1.5.4 (Markov property of Itô diffusion)**.** *Let $\{\mathcal{F}_t\}_{t\geq 0}$ be the natural filtration of the path of a Brownian motion $\{B_t\}_{t\geq 0}$, and let X_t be an Itô diffusion. Then*

$$E[f(X_{t+h}^x)|\mathcal{F}_t](\omega) = E[f(X_h^y)]|_{y=X_t(\omega)},$$

for all $t, h \geq 0$ and bounded Borel functions $f : \mathbb{R}^n \to \mathbb{R}$.

The theorem shows that the Itô diffusion X_t is memoryless. It does not depend on the entire process from its initial state up to t, but only on the value at the time t. This property is essential for the HJB theorem which we will present in Chapter 2.

We can generalize this further, so that the Markov property also holds for a type of random time called a stopping time:

Definition 1.5.5 (Stopping time)**.** Let $\{\mathcal{N}_t\}_{t\geq 0}$ be an increasing family of σ-algebras on Ω. A function $\tau : \Omega \to [0,\infty]$ is called a (strict) *stopping time* with respect to $\{\mathcal{N}_t\}$ if

$$\{\omega : \tau(\omega) \leq t\}_{t\geq 0} \in \mathcal{N}_t \text{ for all } t > 0. \tag{1.27}$$

Thus $\mathcal{N}_t$ contains the history of the stopping time, and if we know $\mathcal{N}_t$, we know whether the stopping time has occurred or not. For such a stopping time, we have that the Markov property still holds, if we also assume that f is a Borel function on $\mathbb{R}^n$.

Having found the time-homogeneous Itô diffusion, we now define an operator associated with this diffusion:

Definition 1.5.6 (Generator of Itô diffusion)**.** Let $\{X_t\}$ be a (time-homogeneous) Itô diffusion in $\mathbb{R}^n$. The *(infinitesimal) generator* A of X_t is defined by

$$(Af)(x) = \lim_{t\downarrow 0} \frac{E^x[f(X_t)] - f(x)}{t}, \quad x \in \mathbb{R}^n. \tag{1.28}$$

We let $\mathcal{D}_A(x)$ denote the set of functions $f : \mathbb{R}^n \to \mathbb{R}$ such that the limit exists at x, and $\mathcal{D}_A$ denote the set of functions for which the limit exist for all $x \in \mathbb{R}^n$. Having the generator on this form is not very applicable in itself, but the next theorem will give an explicit expression for A:

Theorem 1.5.7 (Explicit form of generator)**.** *Let X_t be the Itô diffusion*

$$dX_t = b(X_t)dt + \sigma(X_t)dB_t. \tag{1.29}$$

If $f \in C_0^2(\mathbb{R}^n)$ then $f \in \mathcal{D}_A$, and

$$(Af)(x) = \sum_i b_i(x)\frac{\partial f}{\partial x_i} + \frac{1}{2}\sum_{i,j}(\sigma\sigma^T)_{i,j}(x)\frac{\partial^2 f}{\partial x_i \partial x_j}. \tag{1.30}$$

When f is sufficiently smooth, we can use a formula to find the expected value at a stopping time, τ:

Theorem 1.5.8 (Dynkin's formula)**.** *Let $f \in C_0^2(\mathbb{R}^n)$. Suppose τ is a stopping time, $E^x[\tau] < \infty$. Then*

$$E^x[f(X_\tau)] = f(x) + E^x[\int_0^\tau Af(X_s)ds]. \tag{1.31}$$

Dynkin's formula allows us to express the expected value of a function of stochastic processes by its generator.
The next lemma will allow us to move the limit inferior outside the integral while maintaining a lower bound:

Theorem 1.5.9 (Fatou's lemma)**.** *Assume that $\{f_n\}_{n\in\mathbb{N}}$ is a sequence of non-negative, measurable functions. Then*

$$\int_\omega \liminf_{n\to\infty} f_n(\omega)d\mu(\omega) \leq \liminf_{n\to\infty} \int_\omega f_n(\omega)d\mu(\omega). \tag{1.32}$$

Fatou's lemma allows us to interchange the expectation and the limit for measurable functions. This will be important for the proof of the Hamilton-Jacobi-Bellman equation in Section 2.6.

The definitions and theorems presented in this chapter is the basis for stochastic optimal control theory. This method allows us to find optimal decisions in stochastic environments, and will be the topic of the next chapter.

CHAPTER 2

Stochastic optimal control

2.1 Introduction

Stochastic optimal control aims at finding the optimal function, called the control, to maximize a given value function over time. It is based on the work of R.Bellman, who introduced the *Dynamic Programming principle* in the 1950's, see e.g. [Bel53]. It states that if a sequence of controls, or functions, are optimal, then the sequence will still be optimal if you remove the first control. In general, it therefore says that a subsequence starting at *any* time t will also be optimal.

There are two main ways of solving these types of problems. When the controls are Markov, as described in Section 1.5, we can use the Hamilton-Jacobi-Bellman (HJB) equation to find the optimal control. The other approach, called the *Pontyagin's maximum principle* makes fewer assumptions on the control, but is often more computationally demanding. This thesis will focus on the HJB-approach.

In this chapter we will describe the framework of the *Hamilton-Jacobi-Bellman equation*, which relies heavily on results from Itô calculus, and formulate the general case. We will follow Øksendals approach in [Øks03], build on his proofs, and end the section with an application from mathematical finance.

The main result of this chapter is the Hamilton-Jacobi-Bellman equation. It shows that for a stochastic differential equation satisfying some conditions, we can find an optimal control for the stochastic optimization by solving a much simpler maximization problem (if such an control exists). This is proven for a class of functions called *Markov controls.* It can, however, be shown that under some additional constraints, the Markov controls can perform just as good as any control adapted to the filtration at time t.

There are three main components of the optimization problem; the system, the control, and the evaluation function. We will discuss them separately, before considering the solution to the problem in the form of the HJB-equation.

2.2 The system

Let $(\Omega, \mathcal{F}, P)$ be a given probability space. We let the state of the system at time t be described by an Itô-process X_t of the form

$$dX(t) = dX(t;u) = b(t, X(t), u(t))dt + \sigma(t, X(t), u(t))dB(t), \tag{2.1}$$

where the functions are defined as follows:

$$\begin{aligned}
&b: \quad \mathbb{R} \times \mathbb{R} \times u \to \mathbb{R}^n, \\
&\sigma: \quad \mathbb{R} \times \mathbb{R} \times u \to \mathbb{R}^{n \times m}, \\
&u(t) \in U \subset \mathbb{R}^k, \\
&X(t) \in \mathbb{R}.
\end{aligned}$$

$B(t)$ in Equation (2.1) is an m-dimensional Brownian motion (Definition 1.3.2), and it will represent the "noisy" part of the problem, i.e. the part that we cannot model in a deterministic fashion.

The solution to the stochastic differential equation (2.1) will be denoted $\{X_h^{s,x}\}_{h \geq s}$. This denotes the process with initial starting point x, evaluated on the interval $[s, h]$. The solution can be written as

$$X_h^{s,x} = x + \int_s^h b(r, X_r^{s,x}, u_r) + \int_s^h \sigma(r, X_r^{s,x}, u_r) dB(r). \tag{2.2}$$

We let the probability of $X(t)$ being in each of the subsets of the filtration, starting at x for $t = s$, be denoted by $Q^{s,x}$, so that we can write

$$Q^{s,x}[X_{t_1} \in F_1, \dots, X_{t_k} \in F_k] = P^0[X_{t_1}^{s,x} \in F_1, \dots, X_{t_k}^{s,x} \in F_k],$$

for $s \leq t_i$, $F_i \subset \mathbb{R}^n$, $1 \leq i \leq k$, and $k = 1, 2, \dots$. This gives us a simpler way of describing the probability law the time-shifted process which will be introduced later in the chapter.

2.3 The control

Next, we want to model the control. This is the part that involves the agents ability to act on past information, and possibly influence subsequent processes. Since we assume that the agent observes the state at time t, we need the control to reflect the information available at this time. This implies that we need the control to be $\mathcal{F}_t^m$-measurable at time t, and stochastic for $t > 0$. To achieve this, we introduce

$$u = u(t, X_t(\omega)) \in U \tag{2.3}$$

as a parameter contained in the Borel set U for each t. This will be the action that controls the process $\{X_t\}_{t \geq 0}$. It depends on the time and state, and it affects the process. We note that it only depends on the state space $\{X_t\}_{t \geq 0}$ at time t, and not on the starting point, or any of the preceding states. We call this type of functions *Markov controls*, as the resulting stochastic process will be a Markov process.

2.4 The evaluating function

Our ultimate goal is to optimize decisions, so we want to find the best actions to make over a given time interval. We will evaluate the value of the state and action combinations through two functions. One will model the instant reward at each time point, discounted to present value. The other will model the reward at some terminal, and possibly stochastic, point in time. We define two (continuous) functions for this evaluation: The function f, which is the *profit rate function*, and the function g which evaluates the action at the first exit time of the process from a certain domain G :

$$\begin{aligned} f: &\quad \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}, \\ g: &\quad \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}. \end{aligned}$$

For a fixed domain $G \in \mathbb{R} \times \mathbb{R}^n$, we define $\hat{T}$ as the first exit time from G after some time point s of the process $\{X_r^{s,x}\}_{r \geq s}$. Written more formally:

$$\hat{T} = \hat{T}^{s,x}(\omega) = \inf\{r > s; (r, X_r^{s,x}(\omega)) \notin G\} \leq \infty.$$

This makes it possible to formulate problems where we look at an optimal control over an interval that is modeled to end when a particular event happen. It also allows for much simpler constructions, as shown in the following example.

Example 2.4.1 (Deterministic times are stopping times $\diamondsuit$)**.** If we define $G := T \times \mathbb{R}^n$, where $T \in \mathbb{R}$, $T > s$ is some constant, we get

$$\begin{aligned} \hat{T} &= \inf\{r > s : (r, X_r^{s,x}(\omega)) \notin G\} \\ &= \inf\{r > s\colon (r, X_r^{s,x}(\omega)) \notin T \times \mathbb{R}^n\} \\ &= T, \end{aligned}$$

where we in the second equality use that $X_r^{s,x}(\omega)$ will always be in $\mathbb{R}^n$. We are then left with the deterministic stopping time T.

For simplicity, we introduce

$$Y_t = (s + t, X_{s+t}^{s,x}), \qquad \text{for } t \geq 0, \quad Y_0 = (s, x). \tag{2.4}$$

we can then reformulate both the initial differential equation and the performance function at time s as

$$\begin{aligned} dY_t = dY_t^u &= b(Y_t, u_t)dt + \sigma(Y_t, u_t)dB(t), && (2.5) \\ J^u(y) &= E^y[\int_0^{\tau_G} f^{u_t}(Y_t)dt + g(Y_{\tau_G})\chi_{\{\tau_G < \infty\}}], && (2.6) \\ y &= (s, x). && (2.7) \end{aligned}$$

This function will be our way of evaluating the impact of our action on the rate of profit, and the terminal value. We evaluate the process from time 0, which can be interpreted as our "now", all the way up until the exit time through the function f, and the value of the state at the exit time is evaluated through the function g.

2.5 The objective

For each staring point, x, and each point in the interval, s, we are looking for two things. We aim to find a strategy of controls, defined by the function u, that represent the optimal course of action. We also need to find what "profit" we can expect by choosing this strategy. We denote this profit by the value function $\Phi(y)$, also called the optimal performance.

This can be formalized as stating that for each $y = (s, x) \in G$, our goal is to find the value function $\Phi(y)$, and the optimal control $u^* = u^*(y, t, \omega) \in \mathcal{A}$, such that

$$\Phi(y) := \underset{u(t,\omega)\in\mathcal{A}}{sup} \; J^u(y) = J^{u^*}(y), \tag{2.8}$$

where $\mathcal{A} \in U$ is the family of *admissible* controls.

Having described each of the components of our problem, we are ready to present the solution.

2.6 The Hamilton-Jacobi-Bellman Equation

We will consider Markov controls, given by $u = u(t, X_t(\omega))$ as described in Section 2.3. We can then formulate the dynamics of the system as

$$dY_t = b(Y_t, u(Y_t))dt + \sigma(Y_t, u(Y_t))dB(t), \tag{2.9}$$

where we have used the notation introduced at the end of Section 2.4. By defining the operator as

$$(L^v\phi)(y) = \frac{\partial\phi}{\partial s}(y) + \sum_{i=1}^{n} b_i(y, v)\frac{\partial\phi}{\partial x_i} + \sum_{i,j=1}^{n} a_{i,j}(y, v)\frac{\partial^2\phi}{\partial x_i\partial x_j}, \tag{2.10}$$

where $a_{i,j} = \frac{1}{2}(\sigma\sigma^T)_{ij}$, $y = (s, x)$ and $x = (x_1, \ldots, x_n)$, we can define the generator of the Itô diffusion as

$$(A\phi)(y) = (L^{u(y)}\phi)(y).$$

To formulate the HJB equation we first need the notion of a regular point:

Definition 2.6.1. A point $y \in \partial G$ is called *regular* for G (w.r.t. X_t) if

$$Q^y[\tau_G = 0] = 1.$$

Otherwise the point is called *irregular.*

In other words, any point on the boundary of G is called regular if the process almost surely will leave G at that point.

The next theorem presents the conditions under which a solution to the optimization problem exists. It also shows that given these conditions, it will be found as the solution to a partial differential equation:

Theorem 2.6.2 (The Hamilton-Jacobi-Bellman theorem, part I)**.** *Define*

$$\Phi(y) = \sup\{J^u(y) : u = u(Y) \textit{ is a Markov control}\}. \tag{2.11}$$

Suppose that $\Phi \in C^2(G) \cup C(\overline{G})$, the union of the twice continuously differentiable functions with its continuously differentiable closure, satisfies

$$E^y[|\Phi(Y_\alpha)| + \int_0^\alpha |L^v \Phi(Y_t)| dt] < \infty, \tag{2.12}$$

for all bounded stopping times $\alpha \leq \tau_G$, all $y \in G$ and all $v \in U$. Moreover, suppose that an optimal Markov control u^* exists, and that ∂G is regular for $Y_t^{u^*}$. Then the value function Φ (the solution to our optimization problem) satisfies

$$\sup_{v \in U} \{f^v(y) + (L^v \Phi)(y)\} = 0 \quad \text{for all } y \in G \tag{2.13}$$

and

$$\Phi(y) = g(y) \quad \text{for all } y \in \partial G. \tag{2.14}$$

The supremum in (2.13) *is obtained if $v = u^*(y)$ where $u^*(y)$ is optimal. In other words,*

$$f(y, u^*(y)) + (L^{u^*(y)} \Phi)(y) = 0, \quad \text{for all } y \in G. \tag{2.15}$$

By this theorem, we have that if there exists an optimal control, we know that it will be the function u^* that satisfies Equation (2.13) and Equation (2.14). Simply put the Hamilton-Jacobi-Bellman equation says that we can find the optimal control, the control that maximize the performance function J, as a solution to the partial differential equation $f(y, u^*(y)) + (L^v \Phi)(y)$ and the boundary condition $\Phi(y) = g(y)$.

Sketch of proof ◇

This section is based on the proof of Øksendal, see [Øks03], but has been extended with additional explanations. There are three things that need to be proved, mainly Equation (2.13), Equation (2.14) and Equation (2.15). To prove Equation (2.15) we will use a theorem stating the existence to a problem known as the combined Dirichlet-Poisson problem:

Theorem 2.6.3. *Assume that $\tau_G < \infty$ a.s. Q^x for all x. Let $\phi \in C(\partial G)$ be bounded and let $g \in C(D)$ satisfy*

$$E^x[\int_0^{\tau_G} |g(X_s)| ds] < \infty \text{ for all } x \in G. \tag{2.16}$$

Define

$$\omega(x) = E^x[\phi(X_{\tau_G})] + E^x[\int_0^{\tau_G} g(X_s) ds], \quad x \in G. \tag{2.17}$$

1. *Then*

$$\mathcal{A}\omega = -g \text{ in } G, \tag{2.18}$$

and

$$\lim_{t \uparrow \tau_G} \omega(X_t) = \phi(X_{\tau_G}) \text{ a.s. } Q^x, \text{ for all } x \in G. \tag{2.19}$$

2. *Moreover, if there exists a solution* $\omega_1 \in C^2(D)$*, and a constant* C *such that*

$$|\omega_1(x)| < C(1 + E^x[\int_0^{\tau_G} |g(X_s)|ds]), \quad x \in G, \tag{2.20}$$

and ω_1 *satisfies Equation* (2.18) *and Equation* (2.19)*, then* $\omega_1 = \omega$*.*

It can be proven that for L uniformly elliptic in G, and $g \in C^\alpha(D)$ for some $\alpha > 0$ we have

$$L\omega = -g \text{ in } G, \tag{2.21}$$

and

$$\lim_{x\to y, x\in G} \phi(y) \text{ for all regular } y \in \partial G. \tag{2.22}$$

This is exactly what we need to prove Equation (2.15), as we then have

$$(L^{u^*(y)}\Phi)(y) = -f(y, u^*(y)) \implies f(y, u^*(y)) + (L^{u^*(y)}\Phi)(y) = 0. \tag{2.23}$$

Next we want to prove Equation (2.14). Since ∂G is regular, we know that any process starting in this point will leave G immediately, and thus the stopping time τ_G will be 0. We therefore get that

$$\begin{aligned}
\Phi(y) = J^{u^*}(y) &= E^y\big[\int_0^{\tau_G} f^{u^*}(Y_t)dt + g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}}\big] \\
&= E^y\big[\int_0^{0} f^{u^*}(Y_t)dt + g(Y_0)\chi_{\{0<\infty\}}\big] \\
&= E^y[g(Y_0)] = E^y[g(s,x)] \\
&= E^y[g(y)].
\end{aligned}$$

The last thing to prove is Equation (2.13). We set $y = (s, x) \in G$, and choose any Markov control u. Let $\alpha < \tau_G$ be a bounded stopping time. We then have that

$$E^y[J(Y_\alpha)] = E^y\Big[E^{Y_\alpha}\big[\int_0^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}}\big]\Big] \tag{2.24}$$

$$= E^y\Big[E^y\big[\theta_\alpha(\int_0^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}})\big]\big|\mathcal{F}_\alpha\Big], \tag{2.25}$$

$$\tag{2.26}$$

where we have used the strong Markov property from Section 1.5, and a shift parameter defined by $(\theta_\alpha Y)(t) = Y(\alpha + t)$ "shifting" the function to start in α. The strong Markov property then gives us that for a measurable function η we have $E^{Y_\alpha}[\eta] = E^y[\theta_\alpha\eta|\mathcal{F}_\alpha]$.

We can further use that $\theta_\alpha \int_0^{\tau_G} \eta(Y_s)ds = \int_\alpha^{\tau_G} \eta(Y_s)ds$, which gives us

$$= E^y\Big[E^y\big[\theta_\alpha(\int_0^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}})\big]\big|\mathcal{F}_\alpha\Big] \tag{2.27}$$

$$= E^y\Big[E^y\big[(\int_\alpha^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}})\big]\big|\mathcal{F}_\alpha\Big]. \tag{2.28}$$

$$\tag{2.29}$$

Since we have the expectation of a conditional expectation, we can use the law of total expectation ($E[E[X|\mathcal{F}_s]] = E[X]$). If we also use that $\int_a^T f ds = \int_0^T f ds - \int_0^a f ds$, we get

$$= E^y[\int_0^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}} - \int_0^\alpha f^u(Y_r)dr] \tag{2.30}$$

$$= E^y[\int_0^{\tau_G} f^u(Y_r)dr - g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}}] - E^y[\int_0^\alpha f^u(Y_r)dr], \tag{2.31}$$

$$\tag{2.32}$$

which we recognize as

$$J^u(y) - E^y[\int_0^\alpha f^u(Y_r)dr]. \tag{2.33}$$

We have thus showed that

$$E^y[J(Y_\alpha)] = J^u(y) - E^y[\int_0^\alpha f^u(Y_r)dr]. \tag{2.34}$$

Rearranging this we see that

$$J^u(y) = E^y[\int_0^\alpha f^u(Y_r)dr] + E^y[J(Y_\alpha)]. \tag{2.35}$$

We now let $W \subset G$ be of the form $W = \{(r,z) \in G : r < t_1\}$ where $s < t_i$. Set $\alpha = \inf\{t \geq 0 : Y_t \neq W\}$, so that α is the first exit time from W. We suppose that an optimal control $u^*(y) = u^*(r,z)$ exists (as is stated in the theorem) and choose

$$u(r,z) = \begin{cases} v & \text{if } (r,z) \in W, \\ u^*(r,z) & \text{if } (r,z) \in G \backslash W, \end{cases} \tag{2.36}$$

where $G\backslash W$ is the set of of point that are in G but not in W. Since we have defined α as the first exit time from W we have that

$$\Phi(Y_\alpha) = J^{u^*}(Y_\alpha) = J^u(Y_\alpha). \tag{2.37}$$

If we combine this with the expression for $J^u(y)$ from Equation (2.35), and remember that $\Phi(y)$ is the supremum of $J^u(y)$, we get

$$\Phi(y) \geq J^u(y) = E^y[\int_0^\alpha f^v(Y_r)dr] + E^y[J(Y_\alpha)] \tag{2.38}$$

$$= E^y[\int_0^\alpha f^v(Y_r)dr] + E^y[\Phi(Y_\alpha)]. \tag{2.39}$$

Since $\Phi \in C^2(G)$ we can use Dynkin's formula, given in Theorem 1.5.8, and get that

$$E^y[\Phi(Y_\alpha)] = \Phi(y) + E^y[\int_0^\alpha (L^u\Phi)(Y_r)dr]. \tag{2.40}$$

Note that we can use this formula since we have chosen a control u, and thus Y_t is an Itô diffusion with generator given by $(L^{u(y)}\Phi)(y)$.
If we use this expression in Equation (2.38) we get

$$\Phi(y) \geq E^y[\int_0^\alpha f^v(Y_r)dr] + \Phi(y) + E^y[\int_0^\alpha (L^u\Phi)(Y_r)dr], \tag{2.41}$$

which implies that for all $W = \{(r, z) \in G : r < t_1\}$, we have

$$E^y[\int_0^\alpha f^v(Y_r) + (L^u\Phi)(Y_r)dr] \leq 0, \tag{2.42}$$

and since the stopping time α is always positive, we have

$$\frac{E^y[\int_0^\alpha f^v(Y_r) + (L^u\Phi)(Y_r)dr]}{E^y[\alpha]} \leq 0. \tag{2.43}$$

If we let t_1 go towards s, i.e. towards the starting point, we get that since $F^v(\cdot)$ and $(L^v\Phi)(\cdot)$ are continuous at y, $F^v(y) + (L^v\Phi)(y) \leq 0$.
By combining this with Equation (2.15), we get that

$$\underset{v\in U}{sup}\{f^v(y) + (L^v\Phi)(y)\} = 0, \quad \text{for all } y \in G, \tag{2.44}$$

which completes the proof. ■

The second part of the theorem states that if we have found a control such that $sup_{v\in U}\{f^v(y) + (L^v\Phi)(y)\} = 0$ for all $y \in G$, then we have found the optimal control:

Theorem 2.6.4 (The Hamilton-Jacobi-Bellman equation, part II)**.** *Let ϕ be a function in $C^2(G) \cap C(\overline{G})$ such that. for all $v \in U$,*

$$f^v(y) + (L^v\phi)(y) \leq 0, \quad y \in G, \tag{2.45}$$

with boundary values

$$\underset{t\to\tau_G}{lim}\, \phi(Y_t) = g(Y_{\tau_G})\chi_{\{\tau_G<\infty\}}, \qquad a.s.\ Q^y, \tag{2.46}$$

and such that

$$\{\phi^-(Y_\tau); \quad \tau \textit{ stopping time, } \tau \leq \tau_G\} \textit{ is uniformly} \tag{2.47}$$
$$Q^y\textit{-integrable for all Markov controls } u \textit{ and all } y \in G. \tag{2.48}$$

Then

$$\phi(y) \geq J^u(y) \textit{ for all Markov controls } u \textit{ and all } y \in G. \tag{2.49}$$

Moreover, if for each $y \in G$ we have found $u_0(y)$ such that

$$f^{u_0(y)}(y) + (L^{u_0(y)}\phi)(y) = 0, \tag{2.50}$$

and

$$\{\phi(Y_\tau^{u_0}); \tau \text{ is stopping time, } \tau \leq \tau_G\} \text{ is uniformly } Q^y\text{-integrable for all } y \in G, \tag{2.51}$$

then $u_0 = u_0(y)$ is a Markov control such that

$$\phi(y) = J^{u_0}(y).$$

Hence, if u_0 is admissible, then u_0 must be an optimal control and $\phi(y) = \Phi(y)$.

Sketch of proof

Assume that ϕ satisfies Equation (2.45) and Equation (2.46), and let u be a Markov control. We then define a new stopping time by

$$T_R = \min\{R, \tau_G, \inf\{t > 0\colon |Y_t| \geq R\}\},$$

for all $R < \infty$, i.e the first exit time from either G or the $R \times R$ square. Since by Equation (2.45) we have that $(L^u\phi) \leq -f^u$, we get by Dynkin's formula (Theorem 1.5.8) that

$$\begin{aligned} E^y[\phi(Y_{T_R})] &= \phi(y) + E^y[\int_0^{T_R} (L^u\phi)(Y_r)dr] \\ &\leq \phi(y) - E^y[\int_0^{T_R} f^u(Y_r)dr], \end{aligned}$$

for all $R \leq \infty$. This implies that

$$\phi(y) \geq E^y[\int_0^{T_R} f^u(Y_r)dr + \phi(Y_{T_R})],$$

for all $R \leq \infty$. Since this applies for all R, we can preserve the inequality when taking the limit

$$\phi(y) \geq \lim_{R\to\infty} E^y[\int_0^{T_R} f^u(Y_r)dr + \phi(Y_{T_R})].$$

Further, we can use Fatou's Lemma (Theorem 1.5.9) since we have from Equation (2.47) that both $\phi(Y_{T_R})$, $\phi(Y_{\tau_G})$ and $\int_0^{\tau_G} f(Y_r)dr$ are uniformly Q^y-integrable, and thus measurable. We can thus interchange the expectation and the limit:

$$\phi(y) \geq E^y[\lim_{R\to\infty}\{\int_0^{T_R} f^u(Y_r)dr + \phi(Y_{T_R})\}].$$

We note that when $R \to \infty$, $T_R \to \tau_G$, and thus we get by Equation (2.46)

$$\begin{aligned} \phi(y) &\geq E^y[\lim_{R\to\infty}\{\int_0^{T_R} f^u(Y_r)dr + \phi(Y_{T_R})\}] \\ &= E^y[\int_0^{T_R} f^u(Y_r)dr + g(Y_{\tau_g})\chi_{\{\tau_G<\infty\}}] \\ &= J^u(y). \end{aligned}$$

We have thus proved Equation (2.49). To prove Equation (2.51), we just consider u_0 such that Equation (2.50) and Equation (2.51) holds, and by the same computations we get equality. ■

We will end the chapter by showing how the HJB-theorem can be used to solve an optimization problem in finance.

Example 2.6.5 (HJB-equation in portfolio problem)**.** We will look at a classic example of portfolio selection, taken from [Øks03], where we consider an agent who has the choice between allocating his funds. He can put them in a stock, with price dynamics given by

$$\frac{dS(t)}{dt} = S(t)[\mu + \sigma B(t)],$$

with $\mu > 0$ representing the average price change of the stock, $\sigma > 0$ representing how volatile the stock is, and $B(t)$ denoting a Brownian motion. He can also put them in a bank account with a deterministic dynamics given by

$$dX(t) = \rho X(t)dt,$$

where ρ is the interest rate. We assume there is no possibility of lending money, so the total wealth of the agent is given as

$$Z(t) = S(t) + X(t).$$

If we define the action as the fraction of the wealth allocated in the stock, $u(t) = \frac{S(t)}{S(t)+X(t)}$, we get that $S(t) = u(t)Z(t)$ and $X(t) = Z(t)(1 - u(t))$, and we can therefore write the dynamics of the wealth as

$$\begin{aligned} dZ(t) &= dS(t) + dX(t) \\ &= Z(t)[(\rho(1 - u(t)) + \mu u(t))dt + \sigma u(t)dB(t)]. \end{aligned}$$

We assume that the agent starts with x amount of wealth, such that $Z(0) = x > 0$. Our agent will have to choose how much money to invest in the "risky" stock based on his utility, $U(Z(t_0))$, of the wealth at the end of some period, denoted t_0. We thus get the performance function

$$\begin{aligned} \Phi(s, x) &= \sup_u J^u(s, x) = J^{u^*}(s, x) \\ &= J^{u^*}(s, x) = E^{s,x}[U(Z^{u^*}_{t_0})]. \end{aligned}$$

Since we assume that the only payoff from our investment is the utility of the value of our portfolio at the end of our time period, we get that $f = 0$. For Φ to satisfy Equation (2.13) we need to find the action $v = u(t, x)$ that maximizes

$$\frac{\partial \Phi}{\partial t} + x(\rho + (\mu - \rho)v)\frac{\partial \Phi}{\partial x} + \frac{1}{2}\sigma^2 v^2 x^2 \frac{\partial^2 \Phi}{\partial x^2}. \tag{2.52}$$

Differentiating this with respect to v and setting equal to zero gives us that

$$v = -\frac{(\mu - \rho)\frac{\partial \Phi}{\partial \mathbf{x}}}{x\sigma^2 \frac{\partial^2 \Phi}{x^2}}. \tag{2.53}$$

When put back into Equation (2.52), this gives us that Φ needs to satisfy

$$\frac{\partial \Phi}{\partial t} + \rho x \frac{\partial \Phi}{\partial x} - \frac{(\mu-\rho)^2 (\frac{\partial \Phi}{\partial \mathbf{x}})^2}{2\sigma^2 \frac{\partial^2 \Phi}{x^2}} = 0, \text{ for } t < t_0, x > 0, \tag{2.54}$$

$$\Phi(t,x) = U(x), \text{ for } t = t_0 \text{ or } x = 0, \tag{2.55}$$

where the last equation comes from the boundary condition in Equation (2.14). We choose to solve this for the utility function $U(x) = x^\gamma$. We need to find a solution of the form $\Phi(x,t) = f(t)x^\gamma$. By putting this form into our expression in Equation (2.52) we get

$$f'(t)x^\gamma + x(\rho + (\mu-\rho)v)f(t)\gamma x^{\gamma-1} + \frac{1}{2}\sigma^2 v^2 x^2 f(t)\gamma(\gamma-1)x^{\gamma-2} \tag{2.56}$$

$$= f'(t)x^\gamma + (\rho + (\mu-\rho)v)\gamma + \frac{1}{2}\sigma^2 v^2 \gamma(\gamma-1))f(t)x^\gamma \tag{2.57}$$

$$= g_x'(t) + \lambda g_x(t), \tag{2.58}$$

for

$$g_x(t) = f(t)x^\gamma, \tag{2.59}$$

$$\lambda = (\rho + (\mu-\rho)v)\gamma + \frac{1}{2}\sigma^2 v^2 \gamma(\gamma-1). \tag{2.60}$$

We see that Equation (2.58) is just an ordinary differential equation, which has $e^{\lambda(t_0-t)}x^\gamma$ as its solution. We can now put this into Equation (5.78) we get

$$u^*(t,x) = -\frac{(\mu-\rho)e^{\lambda(t_0-t)}\gamma x^{\gamma-1}}{x\sigma^2 e^{\lambda(t_0-t)}\gamma(\gamma-1)x^{\gamma-2}} \tag{2.61}$$

$$= \frac{\mu-\rho}{\sigma^2(1-\gamma)}. \tag{2.62}$$

This can be put in the expression for λ, giving us

$$\lambda = \sup_{v\in\mathcal{A}}\{(\rho + (\mu-\rho)v)\gamma + \frac{1}{2}\sigma^2 v^2 \gamma(\gamma-1)\} = r\gamma + \frac{(\mu-\rho)^2\gamma}{2\sigma^2(1-\gamma)}. \tag{2.63}$$

We have now found a control that satisfies the conditions given in Theorem 2.6.2. By Theorem 2.6.4, this shows that if $u^*(t,x) \in (0,1)$, i.e. if u^* is an admissible control, then it is an optimal control.

We have in this chapter presented a way of solving optimization problems based on Itô calculus. This approach uses powerful mathematical results, and presents conditions which guarantees an optimal solution. In the next chapter we study a similar type of optimization problem, but from the viewpoint of an entirely different field. Reinforcement learning is a type of machine learning which uses iterative algorithms on data in order to find the optimal control, and the optimal value function in settings of decision making over time. Unlike with the HJB-theorem, reinforcement learning makes few assumptions on the underlying process and value function. We will see, however, that this also limits the convergence results, and makes it harder to provide clear prerequisites needed for the method to find the optimal values and controls.

CHAPTER 3

Reinforcement learning

3.1 Introduction

In recent years, there has been a surge in the use of machine learning, and many companies are now exploring how it can be used to utilize their data in new ways. This popularity can partly be attributed to the increase in computational power, but is also likely an effect of the availability of large amounts of data.

Machine learning and statistics share many methods, but in general their difference lies in the targeting of different goals. While statisticians are primarily focused on the validity of their models and making inference, machine learning is predominantly concerned with making predictions or clustering data. Even though statisticians sometimes do machine learning, and machine learners sometimes do statistics, neither are a sub-field of the other. This might also be some of the reason why many machine learning methods lacks the formal proofs that are more common in statistics.

Reinforcement learning is a type of machine learning, and has gained some fame for having been used to beat the best computers in the games of Go, chess and shogi, see [Sil+18]. These are all games that involve a vast number of states, and achieving them has been considered a giant leap towards truly intelligent machines.

Machine learning

Machine Learning (ML) is an application of artificial intelligence that focuses on the use of algorithms to enable machines to learn from data without explicitly stating the way in which the data should be utilized. Instead ML focuses on fitting data into predefined structures or patterns, and allowing the computer to find the best way to do so. The methods can range from simple methods of regression, where the goal is to find values of parameters in a linear equation, to deep neural networks, where the computer can model complex systems of equations mapping input to output.

The process of using ML can be generalized in the following way: We have some sort of available data. We use this data to construct a *learner*, which is a function that maps input data into some sort of output. When using machine learning algorithms, we need to be able to evaluate the performance of our model. The error estimate of our model will, for any random process, consist of a reducible error and a irreducible error. The reducible error is produced by our inability to correctly capture the true underlying model, while the irreducible error is produced by the random noise in the data. In order to get an accurate estimate of the error in our model it is essential that we evaluate the performance of our model on one data set, usually called the *test set*, while training the model on another set, called the *training set.* This way we avoid that the model tries to fit our data set, which in any non-trivial case will be a subset of our true population, too closely. Problems arising from this is known as *overfitting.*

There are four main classes of machine learning, and the differences between them revolves around how they deal with the training set:

1. *Supervised learning* has a training set which constitutes of independent variables, often denoted x, and a dependent variable, typically denoted y. The name "supervised" comes from the fact that we can imagine that there is a supervisor who knows the correct link between the y's and the x's in our training set. This supervisor then can give us feedback each time we try to infer a value of y from the values of x.

2. *Unsupervised learning* has a training set which has independent variables x, but no dependent variable. We therefore have no "correct" solution that we can get feedback on, and there is no supervisor telling us if we are right or wrong in our assessment. This type of learning tries to discover traits of the distribution, or discovering clusters, in the data.

3. *Semi-supervised learning* has some parts of the training set with both dependent and independent variables, and some parts with only independent variables.

4. *Reinforcement learning*(RL) solves a sequential decision-making problem by performing actions in an environment. You train your model on available data until some convergence criteria is met, and the convergence is our measure of the performance of the model. We therefore don't usually separate our data in training- and test-sets, but instead consider any data used until convergence as the training set.

The majority of machine learning algorithms, among others linear regression, boosting and trees, are supervised learning algorithms. A method does, however, not need to be limited to only one type of learning. Neural networks, for example, can be used both in supervised and unsupervised learning.

Reinforcement learning differs from other fields of machine learning in several ways. Most importantly, it focuses on problems involving sequential decision making, which differs from the more classic cases of prediction and classification performed by most ML algorithms. An implication of this is that RL algorithms also have to take into account the delayed reward of the actions performed, instead of considering the performance of the model on each training data individually and isolated. Lastly, it allows for active exploration performed on the data, which involves letting the learner use prior knowledge gained from earlier iterations to choose the data considered.

We will in this thesis focus on this fourth type of machine learning. This chapter will introduce reinforcement learning, present one of its most used algorithms called the Q-learner, and discuss limitations and possible improvements to this method.

3.2 Markov decision processes

The following section is based on Sutton [SB18]. In reinforcement learning, the primary goal is to find an action that achieves some sort of pre-specified goal, and possibly also what value this action generates. Simply put, the RL problem can be stated as follows:

- Ω is the environment.This represents the system our agent interacts with, and can for example be the stock market of an option or a stock. A specific observation of this system is represented by ω.
- $S_t(\omega, a_{t-1}) \in \mathcal{S}$ is the state of the system. The state space $\mathcal{S}$ can be any value or vector representing the relevant information of the system passed to the agent.
- $a_t(S_t) \in \mathcal{A}$ is the agents action performed in the environment.
- $\pi(S_t)$ is the function mapping any possible state S_t to an action a_t. In other words it is the strategy which the agent act according to.
- $R_{t+1}(S_t, a_t) \in \mathcal{R} \in \mathbb{R}$ is the reward perceived by the agent at time t+1. Note that this is the reward for the action taken at time t, not $t+1$.

When it does not create confusion, we will denote the state, action, policy and reward by S_t, a_t, π_t and R_{t+1}.

We call the learner and decision maker the *agent*, while everything the agent interacts with is called the *environment*. They interact continually, in a loop. At each time step, $t = 0, 1, 2, \ldots$ the agent receives some representation of the environment's *state*, $S_t(\omega) \in \mathcal{S}$, and uses this information to select an action, $a_t(S_t) \in \mathcal{A}$. At the next time step the agent receives a numerical reward $R_{t+1} \in \mathcal{R} \in \mathbb{R}$, representing a payoff from the action chosen at time t, and information of the new state S_{t+1}. We can represent this as a *trajectory* $S_0, a_0, R_1, S_1, a_1, R_2, S_2, a_2, R_3, \ldots$, and thus reduce a wide range of sequential decision making problems to three signals passing back and forth between an agent and an environment: the action a_t, the state S_t and the reward R_t. These types of processes are known as *Markov Decision Processes* (MDP). The goal of an MDP is to maximize the accumulated (discounted) rewards and it is necessary for the states to have the *Markov property*, as described in Section 1.5. This means that we need all relevant history to be represented in the current state, and the actions can only depend on the current state.

Some additional remarks on the flexibility of the framework should be noted:

- The time steps do not need to be fixed and can be of arbitrary length.
- There can be an infinite time horizon, or an end time that can be either deterministic or stochastic.
- The states can be low-level, for example the monetary value of a stock, but also high level with a higher degree of abstraction, as long as we can formalize it in some meaningful way as a state space $\mathcal{S}$.
- The actions are also very generally defined, and can be anything we want our agent to learn.

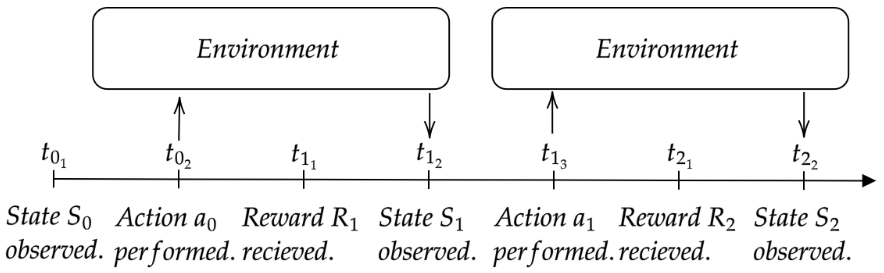

Figure 3.1: Timeline of trajectory.

The downside of this flexibility is that the choices of how we represent this framework, in essence how we choose $\mathcal{A}, \mathcal{S}$, and $\mathcal{R}$, will greatly influence our solution. These choices are subject to personal beliefs and a priori knowledge.

The Bellman equations

The goal of an MDP is to maximize the accumulated (discounted) rewards, which is expressed through a *value function*, $v(s)$. This function assigns a value to each state encountered. We are therefore looking for an optimal policy π, that is a set of actions, that maximize this value function. But since we are considering not only immediate rewards, but also future rewards, we need a formula that connects the value of some state at time t with the value of the following states. The Bellman equation does exactly this, and can be stated as follows [SB18]:

$$v_\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k} S | S_t = s] \tag{3.1}$$

$$= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in S} \sum_{r \in \mathcal{R}} p(s', r|s, a)\big(r + \gamma v_\pi(s')\big), \tag{3.2}$$

where $p(s', r|s, a)$ is the conditional probability of getting to state s' with a reward of r by using action a in state s, and $v(s')$ is the value of the state s'.

In many applications, the choice of action given a state is deterministic, and $\pi(a|s)$ will therefore be 1 for a single $a \in \mathcal{A}$, and 0 for all others. Equation (3.1) can then be seen as the sum of instant and (discounted) future rewards of going from state s to s', weighted by the probability of this transition actually taking place.

Since the action a is the only component we are able to control, the optimal value function will satisfy the Bellman optimality equations:

$$v^*(s) = \max_a \sum_{s' \in S} \sum_{r \in \mathcal{R}} p(s', r|s, a)\big(r + \gamma v^*(s')\big), \tag{3.3}$$

$$\pi^*(x) = \arg\max_a v(s), \tag{3.4}$$

where $v^*(\cdot)$ denotes the optimal value function. These equations are used in the mathematical optimization method *dynamic programming*, and are also the foundation for the theory discussed in Chapter 2.

The key point of Equation (3.3) is that it gives us a relation between the value function for state s, and the value function for the next state s'. Thus if we know the optimal value function for some time point t, we can find the optimal value for the immediately preceding state. This also gives us the optimal action, as it will be precisely the a_t that maximize this value function. As we will see, recursive relationships like this is a recurring theme in reinforcement learning, and the Bellman equation is the foundation for almost all learning algorithms used.

In the two following sections, we will discuss the policy π and the value function v from Equation (3.1) in greater detail.

The policy

The function that connects each possible state of the environment to an action is called a *policy.* This can be random, in which case we write $\pi(a|s)$ as the probability of performing action a, given state s. In all the settings in this chapter, however, the function mapping s to a will be deterministic, so we will write

$$\pi(S_t) = a_t. \tag{3.5}$$

When we are in a setting where the actions performed on the environment will affect the consecutive states, the policy chosen can have a large effect on which types of data, that is which states, the learner is exposed to. We therefore need to make sure that a policy both seeks to act in an optimal way, but also that it tries to explore and gain knowledge of the state space. In this perspective we can differentiate between a *greedy policy*, which will always seek to take the action that maximise the reward, while a less greedy policy will allow for more exploration. The greedy policy is defined as

$$\pi^*(S_t) = \arg\max_a V(S_t), \tag{3.6}$$

while a less greedy, and more exploring policy will choose Equation (3.6) most of the time, but also perform random actions in order to explore the state space.

One of the problems with greedy strategies is that if we are unlucky with our first data points, a trajectory of less optimal actions might be chosen, which can lead to the better actions being starved of data. We are essentially biased towards the data we already know something about, even though there might be other, and better strategies to be tried. One way of rectifying this without having to choose a less greedy strategy is to equip the actions with a strongly optimistic prior belief, so that we will need more data to discard the action. This way we can make the risk for eliminating an optimal action arbitrarily small [KLM96]. We will explain how this can be done in Section 3.3.

A couple of things should be noted here. The policy defining the action a_t is only dependent on S_t. The implication of this is that the policy has the Markov property, meaning that it is independent of the state (and the action) at times s_{t-i} for all i. We should also note that the optimum value of an action is less meaningful when seen outside the context of the policy. Since we assume that the value of a state is dependent on both the initial action, and the policy performed on the remaining states, whether an action is optimal or not will in most problems depend on the actions performed on future (but not past) states.

The value function

How well the action performs is evaluated by a value function, which we defined as a recursive relation in Equation (3.1). The value function takes into account both the immediate *reward* R_{t+1} and the long term reward of the action, which we denote by G_{t+1}. The $R_t \in \mathcal{R} \subseteq \mathbb{R}$, is a problem specific value.

By defining G_t as the instant reward plus the sum of discounted rewards, we can write

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots && (3.7)\\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^2 R_{t+3} + \dots) && (3.8)\\ &= R_{t+1} + \gamma G_{t+1} && (3.9)\end{aligned}$$

We can then write the value function as

$$V_t(s) = \mathbb{E}_\pi[G_t | S_t = s], \tag{3.10}$$

where the expectation is dependent on the policy. This follows from the fact that the rewards in G_t is dependent on what actions are taken in the future, in addition to also being dependent on the stochastic nature of the states.

The value function assigns a value to a state, given that a certain policy is followed in the future. It will be our way of quantifying the performance of our policy, and the goal of reinforcement learning is to find the policy which maximize this value function. To be able to formulate a problem as a reinforcement learning problem we therefore need to find expressions for the immediate and long term reward, and an algorithm that tells us how the agent can learn by observing the environment and the payoff from the value function.

The computation of the reward is considered external to the agent, as it is defined by the problem, and thus out of the agents control. Note that this has nothing to do with the amount of information the agent has of the state and the reward function. The agent can have complete information on both, but will still not have complete control over them.

3.3 The RL algorithms: learning from the environment

There are typically two ways in which we can find the optimal policy. Both is performed by first initializing arbitrary values of the value function for each value $s \in \mathcal{S}$, and for policy iteration also $\pi(s) \in \mathcal{A}$.

1. *Policy iteration* performs optimization based directly on the policy. It is performed by alternating between an evaluation step and an improvement step. In the evaluation step, we use the Bellman equations to update the value function for all $s \in \mathcal{S}$:

$$V_{k+1} = \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s], \tag{3.11}$$

where $k = 1, 2, \dots,$ is the index for the iterations. This update is performed until the change in update is sufficiently small, that is when

$$|V(s) - V_{\text{new}}(s)| < \epsilon, \tag{3.12}$$

for some error bound ϵ and all $s \in \mathcal{S}$. In the improvement step, we check if the policy is actually maximizing the Bellman equations used in step one. If so, the optimal policy is reached, if not, we go back to step one, now using the actions that did optimize the value functions.

2. *Value iteration* performs optimization indirectly by maximizing the value function, since the policy that maximizes this is the optimal set of actions. It updates the value function, for each $s \in \mathcal{S}$, by

$$V_{k+1} = \underset{a}{max} \mathbb{E}[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s, A_t = a], \tag{3.13}$$

   and does so until Equation (3.12) is satisfied.

In the first case we update the policy function for each iteration (through the improvement step), and the update of the value function is used as a way of evaluating the policy. In the second case we update the value function directly.

If we want to utilize prior beliefs, as discussed in Section 3.2, we could choose non-arbitrary initial values of $V(s)$ and $\pi(s)$. We would then instead be using values representing our prior beliefs, either for all $s \in \mathcal{S}$, or for some subset of the state space.[KLM96]. These prior beliefs could for example be represented by giving high values to states that are believed to be "good" in some sense.

When assessing the performance of our model we need to quantify exactly what we consider a good performance. Optimally, we would want an algorithm that quickly converges to the correct optimal strategy and value function. In reality, however, the need for efficiency is often at odds with the need to find the correct optimal value. Additionally, the optimal value is often either impossible to reach, or computationally unfeasible, and we therefore are more likely to try to converge fast to a "near optimal" value. In this case we have to specify what we define as near optimal, essentially choosing ϵ.[KLM96] This will be discussed more in Section 3.4 and Section 3.6.

A well explored alternative to reinforcement learning is Monte Carlo simulation. These methods have clear convergence proofs, and are therefore incorporated into many RL methods. We will therefore present the general framework of this approach next, before presenting how reinforcement learning algorithms extend on this method.

Monte Carlo simulation

Monte Carlo (MC) methods involves simulating by use of random numbers in order to create a sample set, or to evaluate some mathematical expression. Given a distribution, we can generate samples of any function of this distribution. We can then use averaging to infer properties of this function. The following example show one use of MC estimation.

Example 3.3.1 (MC simulations of GBM's (cont. from Example 1.4.11) $\diamondsuit$)**.** We can use Monte Carlo simulations in order to compute estimates of the properties of a geometric Brownian motion. The approach is written in pseudo-code below, and the full code is found in Appendix A.1.

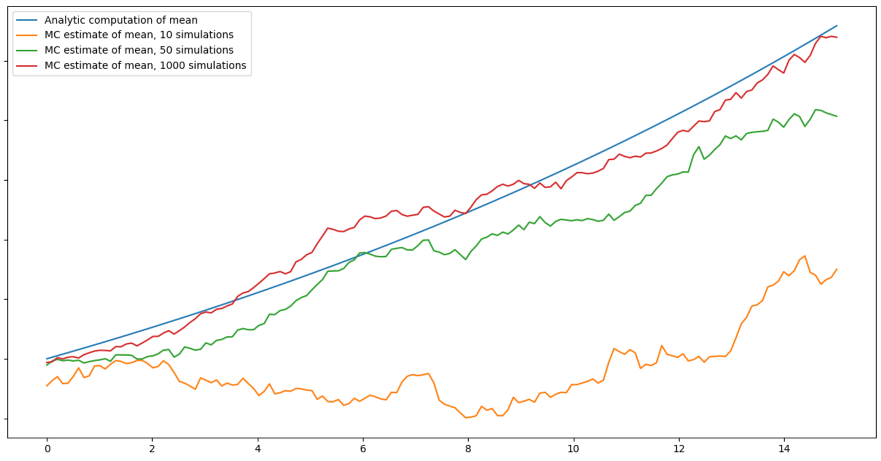

Figure 3.2: Plot of MC estimate of mean of geometric Brownian motion. Shown for 10, 50 and 1000 simulations. Numerical solution is also displayed.

Algorithm 1: Monte Carlo computation of GBM mean

```
Let $Nmc$ be the number of MC simulations, $n$ be the number of time
  steps in a (discrete) Brownian motion, B(t) is Brownian motion, and
  G(t) is geometric brownian motion.;
for $Nmc = 1, 2, \ldots, M$ do
    B(0) = 0 ;
    G(0) = initial condition;
    Generate n Brownian increments from distribution $N(0, t)$;
    for $t = 1, 2, \ldots, n$ do
        $\epsilon \sim N(0, t)$ ;
        $B(t) = B(t-1) + \epsilon$;
        $G(t) = G(0) \exp[(\mu - \frac{\sigma^2}{2})t + \sigma B(t)]$;
    end
    Compute the mean for each time point $t$ as;
    $\mathbb{E}[G(t)] = \frac{\sum_0^M G(t)}{M}$;
end
```

By using the code in Appendix A.2 for 1000 simulations, we get the estimate of the mean as portrayed in the plot in Figure 3.2. Here we see the estimates for the mean of a GBM for 10, 50 and 1000 simulations. As we see, a higher number of simulations cause the graph to closer approximate the graph of the true analytic solution [1].

Just like with exploration of the state space, MC simulation and estimation might also suffer from too large, or infinite, state spaces. The random number

[1]Note that a higher volatility, that is a higher σ leads to more variation in the paths, and thus a larger number of simulation needed for a good approximation. The paths in the plot has $\sigma = 0.2$.

generator used for the simulation may not be able to generate all points within a reasonable time, or at all.

The MC estimate will have sampling error, where some of the error will be irreducible error stemming from the variance of the data. Some of the error, however, will be variance that can be reduced by increasing the number of MC samples. Choosing the right size for the sample set is therefore essential for the MC estimate to be of value. Methods exist for estimating a confidence interval for the estimator:

$$(\hat{\theta} - I_1, \hat{\theta} + I_2). \tag{3.14}$$

The sample size can then be chosen such that the length of the confidence interval is shorter than some max value. When using MC simulations, the sample standard deviation (SD) is often reported alongside the results, as the order of the SD will give an upper bound on the precision of the estimate [Gen03].

Temporal difference: Where Monte Carlo meets dynamic programming

We will be using methods from a framework of reinforcement learning called *temporal difference* (TD). Temporal difference methods contain elements from two well explored methods in optimization, Monte Carlo (MC) estimation (Section 3.3) and dynamic programming (DP) (Section 3.2). Just like with Monte Carlo methods, it learns by directly observing the environment without needing a model of the underlying dynamics. And just like in dynamic programming, it uses iteratively updated learned estimates, i.e. it uses bootstrapping [SB18].

Both MC and DP uses experience to solve prediction problems, but they differ in what their target is. In the simplest of cases, the Monte Carlo method will update the value function[2] by

$$V(S_t) \to (1-\alpha)V(S_t) + \alpha G_t, \tag{3.15}$$

where G_t is the discounted future reward, and α is a step size parameter, while the TD method will update V by

$$V(S_t) \to (1-\alpha)V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1})], \tag{3.16}$$

or, equivalently, by

$$V(S_t) \to V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \tag{3.17}$$

The method above is called TD(0) or *one-step TD*, and is a special case of the more general *TD(λ)* (for more on this method, see [SB18]). We call $R_{t+1} + \gamma V(S_{t+1})$ the *TD target*. Temporal difference methods augment the existing estimate of the value function by a learning rate times the difference between the old estimate and the TD target. We see that TD bases its update in part on an existing estimate, i.e. it is a bootstrapping method. We have that

$$\begin{aligned} V(s) &= \mathbb{E}_\pi[G_t | S_t = s] && (3.18)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] && (3.19)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]. && (3.20) \end{aligned}$$

[2]note that V is still a function of the policy π.

We can say that MC methods uses an estimate of Equation (3.18) as target, while dynamic programming uses an estimate of Equation (3.20). They are however not estimates for the same reasons. While in the MC case it is an estimate because we do not know the true expected value and therefore have to use a sample return, in the DP case we assume complete knowledge of the model. In that case it is instead the fact that we don't know $V(S_{t+1})$ that creates the need for estimation, and forces us to use an estimate $V(\hat{S}_{t+1})$. The temporal difference method inherits both these traits. We use an estimate both because we use sampling, and because we don't know V [SB18].

When we are in a setting of not knowing the true underlying model of our data the advantages of TD methods over dynamic programming is clear. We can simply take our data as given, and do not need any knowledge of the transitional probabilities. We are simply observing samples from some unknown distribution, without making any assumption on what this distribution is.

As for Monte Carlo methods, the essential difference is that while MC methods must wait until the end of some trajectory to update V, TD methods can update it at time $t + 1$. The Monte Carlo method needs to wait until we have reached the end of the time period in our simulation, as it computes the expected value. Temporal difference algorithms, on the other hand, are doing bootstrapping, and can therefore use information as it is gathered to continually update its estimate. This can potentially speed up learning.

An inherent weakness to bootstrapping methods is the dependence on the initial estimate. The value function is being updated by using both some new data and the old estimate. If there are not enough data points this might lead to divergence, i.e. our learning algorithm might not converge. We do however know that for enough data, some TD algorithms are proved to converge. The Q-learner is one of these.

The Q-function

Q-learning is an algorithm which is used for choosing an optimal action by finding the optimal value of state and action trajectories.

The Q-function, also known as the *action-value function*, is defined by the following update-rule

$$Q(S_t, a_t) = (1 - \alpha)Q(S_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)], \tag{3.21}$$

where α is a learning-rate parameter and γ is the discounting factor. $Q(S_t, a_t)$ represents the value of taking the action a_t while being in state S_t. We can therefore interpret Equation (3.21) as updating the Q-function by a weighted sum of the old estimate and an estimate using the sample of R_{t+1} (and the estimate on future optimal value).

The following algorithm, taken from Sutton [SB18], describes the complete set-up:

Algorithm 2: Q-learner algorithm.

```
Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ ;
for $s \in \mathcal{S}_i$, where $\mathcal{S}_i$ is some trajectory, do
    for $t = 1, 2, \dots, n$ do
        Choose action $a$ from $S_t$ using policy derived from the Q-function;
        Take action $a$, observe $R$ and $S_{t+1}$;
        $Q(S_t, a_t) \to (1-\alpha)Q(S_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)]$;
    end
    until the terminal state is reached.
end
```

The Q-learner is *model-free*, which means that we are not making any assumptions on the underlying dynamics of the system. This makes the algorithm applicable to a large number of problems.

It is also an *off-policy* method. This means that the policy which is learned is not dependent on the policy used while learning. Specifically, it means that the learner will gain knowledge of the environment with a policy b, and use this to find the optimal value of the Q-function. It will however, not be learning this policy. Instead, it will learn a policy, π^*, which is the policy that maximizes the Q-function. In Section 5.2 we use the greedy Q-function to find a solution of a portfolio optimization problem.

Q-learning is one of the most used RL algorithms, but when applied to large or continuous state spaces it has some non-negligible short-comings. We will discuss these problems in more detail in Section 3.4 and Section 3.5.

3.4 Generalization of the state and action space

In a lot of settings, we are faced with an infinite number of actions, and possibly also an infinite number of states. Even in settings where there is a large, but finite number of actions or states the computational complexity will likely make it impossible to learn the model or the optimal policy. To deal with this we will need to generalize the state and action space. This can often be done with the use of a suitable basis or function approximation, under the assumption that similar actions on similar states will generate similar outcomes. In this case, the choice of weights is a supervised learning problem, and in general all methods used for supervised learning can be used for the function approximation [SB18]. Some are however more general, and in recent years, deep Q-learning have gained a lot of popularity. This is an extension of the traditional algorithms used in Q-learning, and the only correction is that our generalization of large state and action spaces are done by using neural networks (for more on this, see [Mni+15]).

The choice of generalization can be essential for the convergence of the learning algorithm, but there is no universal answer to what the best choice of method is. Choosing this often requires a great deal of insight to the problem, and often some trial and error. Linear function approximation has been the most used approach, mainly because of the simplicity of the approach, the fact that it generate quadratic errors, and that it has been shown to have stronger convergence properties in many cases.[KOT11] It also makes it easier to use least squares optimization, as seen in Section 5.2.

In general, function approximation poses problems for all off-policy methods. Unlike with on-policy methods, where the extension to a generalized state or action space is more direct, these methods require special care. The problem is connected with the distinction between a training policy used for exploration, and a target policy which we are trying to optimize. This potentially allows for artificially high values given to actions because of the anticipation of a future reward. While in the online setting these anticipations needs to be confirmed, this is not the case for the off-policy setting. This problem is still being studied, and even though some methods have shown promising results, none have managed to extend the proof of convergence in Q-learning to the generalized setting [SB18].

Example 3.4.1 (Divergence of Q-learning $\diamondsuit$)**.** As an example of the problems of off-policy learning, we might construct a simple example. This is based on an divergence-example from Sutton [SB18], but with an added context.

Let us assume an agent who is considering buying a risky stock, S_2, with funds from another stock, S_1, which she owns. She is under the assumption that this stock is twice as valuable as the stock she owns, so we set up the assumed parameterized values as

$$Q(S_1) = \omega \tag{3.22}$$
$$Q(S_2) = 2\omega. \tag{3.23}$$

As it turns out, both stocks are value-less, and gives a return of 0 for each t. This is however, still possible to represent exactly by our parameterized Q-functions. The corresponding Q-function will be

$$Q(S_t, a_t) \to Q(S_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, a_t)]. \tag{3.24}$$

We assume that a_t has no influence on the Q-function at all, and that we are simply trying to discover what value change we will get by going from owning stock 1 to stock 2. If our initial assumption is that $\omega = 1$, then for $t = 1, 2$, i.e. only two time periods, we get that the first iteration of Q-update is

$$\begin{aligned} \omega_{new} &= \omega + \alpha[R_{t+1} + \gamma 2\omega, a) - \omega] && (3.25)\\ &= \omega + \alpha[0 + \gamma 2\omega - \omega] && (3.26)\\ &= (1-\alpha)\omega + \alpha\gamma 2\omega && (3.27)\\ &= (1-\alpha) + 2\alpha\gamma, && (3.28) \end{aligned}$$

which is greater than one for $(2\gamma - 1)\alpha \geq 0$. For this case, the value of ω will in fact increase as long as the initial value is greater than 1. If we where to encounter this state multiple times, this effect would only get larger, and the weight would go towards infinity.

The key to this example is to understand the effect of assuming a different policy behaviour from the one performed while learning the Q-function. Even in the case where the policy does not affect the Q-function it can lead to divergence. The reason for this is that we assume that the learner is doing sub-optimal decisions when learning the Q-value. In this example, the owner of the stock assumes that he should not take into account the future disappointing return of the stock. Instead he assumes that this bad return is stemming from his exploratory policy not being optimal, not the fact that the stock is bad (which it is). Thus he will not update the weights of this bad state, even when future experience will show him the true return of the stock.

While this might seem like a very simple example, and it does not truly show that the weights are not updated at a later point, it captures the essential problem of off-policy learning. More complex examples with complete MDP's exist, see for example Sutton [SB18].

Choosing the class of function approximators

When using function approximators in RL algorithms, we are no longer trying to find the optimal value function, or Q-function. We are instead looking for some $\tilde{Q}^* \in F$, where F is the class of function approximators considered. This might be parameterized functions, for example linear functions, or non-parameterized functions such as neural networks. Given such a class of functions, we want to find the $\hat{Q}^*$ that approximates Q best. We therefore need a notion of the distance between Q and $\hat{Q}$. While the Euclidean norm might be fitting for some cases, often we are more concerned with some states than others. We would likely be more interested in minimizing the error for a state with large probability of being encountered, than the error of unlikely states. Sutton [SB18] therefore proposes to use a weighted distance measure, defined by

$$||Q||_\mu^2 = \sum_{s \in \mathcal{S}} \mu(s) Q(s,a)^2. \tag{3.29}$$

The weights $\mu(s)$ represents how much we "care" about each state instance s, and can be set using for example prior knowledge of the problem.

We can then write the *mean squared action-value error*, $\overline{AVE}$, given by

$$\overline{AVE}(\mathbf{w}) = \sum_{s \in \mathcal{S}} \mu(s)[Q_\pi(s,a) - \hat{Q}(s,a), \mathbf{w})]^2 \tag{3.30}$$

as

$$\overline{VE}(\mathbf{w}) = ||Q_\mathbf{w} - Q_\pi||_\mu^2. \tag{3.31}$$

This error will never converge to zero unless the function class chosen for the function approximation perfectly captures the true Q-values. We can therefore not use it as a formal definition of convergence. It does, however, give us a way of evaluating how well the class of functions approximates the Q-function. We are then faced with balancing the minimization of this error against the size of the class of functions. Choosing a too large class might make learning the approximated values computationally unfeasible. It might also lead to overfitting, as described in Section 3.1.

3.5 Reasons for divergence

Although proofs and guarantees of convergence are sparse in the field of reinforcement learning, we do know something about what causes the lack of convergence.

As described by Sutton [SB18], there are three components that, when combined, cause large problems for convergence. He calls the combination of these *the deadly triad*:

1. Off-policy algorithms use a different policy for learning than what it is trying to learn. This means that the distribution of the samples will most likely be different than the distribution of the actual on-policy distribution. Specifically, if the true transitional probabilities are represented by P, but our learner is only exposed to the transitional probabilities D, then this might lead to problems.

2. Using bootstrapping methods involve sampling from an existing estimate. Such methods can be vulnerable to bad initial estimates.

3. Function approximation leads to an incorrect representation of the true value or Q-function.

These elements are all present in Q-learning under continuous environments, and it is for this reasons that convergence fails.

Existing convergence proofs

As mentioned earlier, it has been proved that the Q-learning algorithm converge to the optimal value function, given that it is exposed to enough data (see [WD92] for complete proof). When generalization is introduced, of the form described in Item 3 in Section 3.5, we no longer have such a guarantee [SB18].

The convergence proofs that exist on Q-learning (without function approximation), and on temporal difference in general are based on showing that the learning-operator is a contraction (see for example [SB18], [MMR08] or [TV97]). These proofs provide some interesting insight into the problems with showing convergence of the Q-function. The proof of Q-function convergence assumes that all states are visited an infinite amount of times. In large state spaces, however, we might not even visit a state at all, and many will only be visited once. In the case of the continuous setting we are even worse off, as no state will be visited more than once. There exist methods to deal with this problem but none give the same assurance of convergence as the case of non-continuous Q-learning.

Another proof of interest is the proof of convergence of the $TD(\lambda)$-algorithm [TV97]. This method shares many similarities with the Q-function. While it is proved that this algorithm converges when using linear function approximators, this only applies for an on-policy setting. Further, a counter-example is provided, showing that an off-policy method will fail, keeping all other assumptions required for the proof intact. The literature seems to imply that if a convergence proof is to be gained, it will likely not happen without ensuring that the behavior policy and learned policy is closely related [TV97].

3.6 Computational complexity and efficient learning

When we are faced with large state or action spaces, the use of function approximators on these spaces can often massively improve the computational complexity and improve convergence speed. There are however also differences in the computational speed of different methods.

In general, value iteration is much faster per iteration, but policy iteration takes fewer iterations. The computational complexity of value iteration is $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$ steps per iteration, while for policy iterations it is $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2 +$

$|\mathcal{S}|^3$).[KLM96] The "expensive" part of policy iteration is solving for the exact value of V_π. The algorithm can therefore be sped up by using an alternative version that fix the policy over successive iterations and instead do value iteration in these steps. There are also several numerical-analysis techniques that can speed up the processes both for value and policy iterations, like *multigrid methods* and *state aggregation* [KLM96].

Another way of making the learning more efficient is to utilize prior knowledge to make for "smarter" exploration of the state space, as described briefly in Section 3.3.

When all the data is already present, for example when we are only looking at historical data, we can utilize what is known as *batch updating.* Batch updating means that we only update our Q-function after processing a batch of data, instead of updating it after each new data point, i.e. after each new observation of our environment. This will also speed up the learning.[LGR12]

In this chapter, we have introduced reinforcement learning, and highlighted some of its strengths and weaknesses. We have primarily focused on a type of RL algorithm called Q-learning, and shown under which circumstances the algorithm might fail to converge. We have chosen to focus on the Q-learner because it is one of the most used algorithms from the RL framework. It also has some strong convergence properties for the finite-state case. While this convergence no longer is guaranteed for the case of continuous underlying dynamics, this makes it an interesting focus for comparison with the HJB-equations discussed in Chapter 2. This comparison will be the topic of the next chapter. There, we will discuss the connections between the HJB-equations and RL. We will discuss similarities and differences between the two frameworks, and try to formalize prerequisites for the convergence of the RL algorithms, specifically the Q-learner.

CHAPTER 4

Reinforcement learning and stochastic optimal control

4.1 Introduction

The two previous chapters have shown two different approaches to optimizing over action- and value spaces. Both are based on the Bellman equation. The theory of stochastic optimal control (SOC) extends the approach to a continuous state and action setting through Itô calculus and the Hamilton-Jacobi-Bellman equation. In the reinforcement learning setting however, the applicability for the continuous case is more indirect, coming from a generalization of the state and action space. This also means that in addition to the approximality caused by the randomness of the paths, we will also get an error from our inability to represent the state- and action space accurately. While there are many examples of these approximations working well, there are few formal proofs, and in practice the use of these algorithms often involves a certain degree of trial and error.

This chapter will look at the differences and similarities between the two frameworks, and try to formalize what the differences might mean for the convergence of RL algorithms, specifically the Q-learner used in combination with function approximation. We will look at some assumptions needed for convergence of the approximated Q-function, and study whether any of the assumptions posed by the HJB-equation might help convergence in the Q-learner case.

4.2 Comparing the general framework

We will start by highlighting the common features of stochastic optimal control and reinforcement learning before turning to their differences. In both settings we are looking for an optimal trajectory, or a path. In addition to some deterministic dynamics, this path will be determined by three factors:

1. The starting state of the system.
2. The random noise.
3. The actions which are performed on the system.

Interpretation:		Parameter:	Restrictions:
State of the system	SOC	Y_t	Itô diffusion.
to be controlled.	RL	S_t	Markov property.
Actions performed	SOC	u	Markov property, $\mathcal{F}_t - measurable.$
in the system.	RL	a	Markov property, $\mathcal{F}_t - measurable.$
Short term reward.	SOC	$g(Y_{\tau_G})$	continuous.
	RL	R_t	
Long term reward.	SOC	$\int_0^{\tau_G} f^{u_t}(Y_t)dt$	continuous.
	RL	G_t	
Value function.	SOC	$J^u(y)$	$E^y[\lvert\Phi(Y_\alpha)\rvert + \int_0^\alpha \lvert L^v\Phi(Y_t)\rvert dt] < \infty$
	RL	$V_t(s) = \mathbb{E}_\pi[G_t \lvert S_t = s]$	

Table 4.1: Comparison of SOC and ML.

The first factor is known, but in most cases not controllable, the second is neither controllable nor known apart from some characteristics, and the last one is controllable, but might be a function of the partially unknown state space. The question posed in both stochastic optimal control and in reinforcement learning is how we can use the controllable action, which we denoted u in SOC and a in RL, to optimize the system. To optimize the system is just another way of saying that we want to find the optimal path of the value function.

In this chapter, we will generally use the term *control action* when we are simultaneously talking about the control u and the action a.

An important difference between the two approaches lies in the method they use. In stochastic optimal control we can use theory from differential equations to find the control that optimizes the value function. This is based on first using data or prior knowledge to find an expression for the dynamics of the system. In reinforcement learning, however, we are always performing our optimization by use of iteration. We are making no assumptions on the underlying model, but are instead discovering the dynamics through iteratively updating some value function or policy.

In Table 4.1 we have listed the main components of the two frameworks. We will discuss the components separately in the following sections.

The state

We will begin by addressing the first component in the table, the state of the system. In general, the states are less rigorously specified in the RL setting than within SOC. They are assumed to be some trajectory, or path, either discrete or continuous. The only assumption that needs to be satisfied is that the states have the Markov property.

In the SOC case, we assume for Y_t to be an Itô diffusion, i.e. a solution to the system dynamics

$$dX(t) = dX(t;u) = b(t, X(t), u(t))dt + \sigma(t, X(t), u(t))dB(t), \tag{4.1}$$

as described in Equation (2.1). The restrictions on $b(\cdot)$ and $\sigma(\cdot)$ ensures the existence of a solution X_t. This diffusion also has the Markov property. We note that this is not the same as assuming that the controls are Markov, which is a stronger restriction than just the states being Markov.

The control action

In both the RL and optimal control framework, the control action is assumed to have the Markov property (see Section 1.5). That is, in both cases we demand that the control action taken at time t is only based on information from state S_t or Y_t. When using the HJB-equations, we assume that the control is $\mathcal{F}_t$-measurable. This is usually not specified in the RL framework, but is assumed because of the way the agent is acquiring knowledge of the environment. When we use algorithms where all data is processed simultaneously, this assumption still needs to hold. We also note that in the stochastic optimal control setting, we know the relation between the control action and the state space, while in the RL case, this relation needs to be discovered.

The short and long term reward

The short and long term rewards described in Table 4.1 are defined somewhat differently in the two frameworks. Stochastic optimal control allows for a specific reward signal that applies for the final stage of the trajectory, that is the function $g(\cdot)$. This function does not need to have any connection to the function $f(\cdot)$, that describes how we evaluate the reward over time. In the case of the RL-scheme. we have only one way of expressing the reward, through the reward signal R.

In the general setting of RL, we put a great deal of importance on the initial reward of our action. This is because it is the only certain information we have at the next step, telling us whether we are on a good path to our goal or not. This can for example be seen in the update rule for the Q-function,

$$Q(S_t, a_t) = (1-\alpha)Q(S_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)], \tag{4.2}$$

where we take the sampling information gathered by the reward signal, R_t, and use it as part of the update. Future rewards do of course also matter, but we can separate the initial reaction from our action because we are not considering expected values, but how the value function is actually responding to our choice of action. This is important in a setting of sequential decision making because the instant reward is the only riskless reward we can consider. All other rewards are hypothetical, and may not actually be realized.

The value function

We will end by discussing the value function for the two approaches, as shown in Table 4.1. This function will evaluate the connection between the state and action control, and the short and long term reward. As discussed in Section 2.4, the value function in stochastic optimal control is expressed as

$$J^u(y) = E^y[\int_0^{\tau_G} f^{u_t}(Y_t)dt + g(Y_{\tau_G}\chi_{\{\tau_G<\infty\}})], \tag{4.3}$$

where we note that both f and g can be 0. In RL it is expressed as

$$V_t(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s], \tag{4.4}$$

where R is the instant reward from the last action, and G is the following discounted rewards.

The RL framework value function, Equation (4.4), is based on a simple observation: Any reward at a given time, is only a function of the current state, but an action might affect which states, and thus which rewards are gained in the future. Thus we get that an action might affect all future rewards, but these rewards are still only a function of the current states, and are as such Markov rewards.

In the stochastic optimal control value function Equation (4.3), we are looking at the long term reward from now until an end point, τ_G, and consider separately the reward gained at this 'exit time'. In the RL value function Equation (4.4), we are considering the instant reward, given by an action, and a future (uncertain) reward that is given as all future rewards. That is, for any action, we can look at what that action gives us immediately, and what it gives us in the future.

4.3 Formalizing the setting of RL

As shown in the previous section and Table 4.1, the general framework of reinforcement learning puts few assumptions on the components, such as the state-action space, or the value function. There are however some things that need to be satisfied in order to at least theoretically be able to reach an optimal value.

For any reinforcement learning problem, there are two clear assumptions that applies for all types of algorithms (for instance the Q-learning algorithm in Section 3.3):

1. We require that the problem can be formulated as a Markov decision problem, as described in Section 3.2.

2. We require availability of samples from the data distribution, or an environment where one can simulate in order to collect data.

If these two requirements are met, we are able to set up any reinforcement learning algorithm. However, we do not have any way of knowing whether this algorithm will converge or not.

Additional requirements for convergence Of TD algorithms

In the analysis of convergence by Melo et al. [MMR08], two additional requirements are presented as necessary for the convergence of an RL algorithm. The first, and most intuitive, requirement concerns the data acquired. We need a sufficient amount of information on the dynamics of the system in order to find an optimal policy and its corresponding value function. In general, we will need the data to have "explored" the state-action space, $\mathcal{S} \times \mathcal{A}$, enough. This means that if we let $(\Omega_S, \mathcal{F}_S, P_S)$ denote the probability space (see Section 1.2) of the sample set, and $(\Omega_D, \mathcal{F}_D, P_D)$ denote the probability space of the true underlying data, we require that

$$(\Omega_S, \mathcal{F}_S, P_S) \approx (\Omega_D, \mathcal{F}_D, P_D). \tag{4.5}$$

The less data the algorithm is exposed to, the more bias the model will have. When we are in an online situation, that is when the information is gathered "real-time", this is less of a problem. In that case we can define an exploratory, that is a less greedy, policy which will explore the state-action space as needed. If we are in a batch-learning setting (see Section 3.6), however, we will typically have a fixed amount of data available. In this case the agent's ability to find the optimal policy is dependent on the agent's access to information on the dynamics. If the samples of data does not contain remotely optimal actions, we can not expect to find these actions from the data either. To counter this problem, *growing batch learning* is sometimes used, where we alternate between performing batch learning, and using the derived policy to gain more samples from the state-action space.[LGR12]

The second assumption for convergence by Melo et al. [MMR08], regards the need for some sort of "smoothness" of the underlying value function. This is a direct consequence of the solution to our optimization only being approximate. It will be approximate because of randomness in our data, and because we are approximating the Q-function. We will therefore need that any approximate solution is also within some range of the underlying true optimal values. A small change to an action should not drastically change the course of the trajectory. In the convergence proofs by Melo et.al [MMR08] the weights of the value function are required to be Lipschitz continuous with respect to the action, although different assumptions have been made in other convergence analyses.

Interestingly, this is not too different from the assumption made on the value function J in stochastic optimal control, as we there assumed C^2-smoothness (see Theorem 2.6.2 and Table 4.1). As the Lipschitz continuity is a stronger assumption than the assumption put on J, one might question whether this requirement could be soften in the RL case.

The assumptions listed here are present in most proofs of convergence, but several examples [SB18], have shown that they are not sufficient, even when the optimal solution does exist. The remaining assumptions often differ, but we will discuss the ones that are present in proofs of convergence regarding the Q-learner.

Convergence of the Q-learner

In general, the Q-learner algorithm will converge when all states have an exact representation, given enough data, in addition to some other minor demands, namely:

1. $\sum_{i=0}^{\infty} \alpha_i = \infty$,
2. $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$,

3. The reward signal is finite i.e. $R_t < \infty$ for all t,

where α is the learning rate [SB18]. Item 1 and Item 2 basically ensures that the learning rate is not to large, nor too small. A too large learning rate can cause the learner to skip past the optimal value, while a too small learning rate will make in computationally unfeasible to reach an optimal value (see for example the proof in [WD92] [MMR08] for more on the use of these assumptions). These requirements are also present in the proofs of similar algorithms, such at the proof of $TD(\lambda)$ by Tsitsiklis [TV97].

4.4 Assumptions for the HJB equation

While the assumptions on the RL framework are quite loosely defined, the assumptions posed by the HJB equation are more rigid and formalized. Generally they consist of:

1. The action is Markov. This is basically the same demand as in the RL framework.
2. The value function needs to be twice continuously differentiable, or in the continuous closure of this space. This smoothness is an assumption on the function of both the starting point of the process, and the process itself. In other words, both small changes to the process and small changes to the initial value should not change the optimal solution too much. This is similar to the assumption of smoothness discussed in the last section. It does, however, give us a candidate for exactly how smooth this function is required to be.
3. We are demanding that the expected value of the value function (for any stopping time bounded by τ_G) and the integral of the operator is finite. The first will be true for any convergent RL algorithm. The second, however, does not have a clear counterpart in the RL setting.
4. We are assuming that the closure is regular, so that any process will leave the given domain G when it gets to the closure.
5. Lastly, we assume that an optimal control, u^*, actually exists.

Under these assumptions, we know that the solution will given by the differential equation defined in Equation (2.15), that is by

$$f(y, u^*(y)) + (L^{u^*(y)}\Phi)(y) = 0, \qquad \text{for all } y \in G, \tag{4.6}$$

and

$$\Phi(y) = g(y) \qquad \text{for all } y \in \partial G. \tag{4.7}$$

Remark 4.4.1. The assumption in Item 3 requires some additional comments. As far as this author can see, the assumption on finite expectation, shown under restrictions of the value function in Table 4.1, poses no additional restrictions on the function Φ. We have already assumed $\Phi \in C^2(G) \cap C(\overline{G})$. This will ensure that the partial derivatives in the operator L in Equation (2.10) will be finite. We have also assumed that $b(y, v)$ and $\sigma(y, v)$ are Lipschitz continuous (Equation (1.20)), so the operator will be finite on a finite interval $[0, \alpha]$.

We also note that in Example 2.6.5, we were able to find an exact solution to the optimization problem. This is, however, a quite unusual situation, and we often need to use approximations in order to solve Equation (4.6). This means that we are also facing questions of efficiency in the HJB setting.

4.5 Combining results from stochastic optimal control and RL

We have in the previous sections discussed the differences in the two frameworks of HJB theorems and the TD-algorithms in RL. One natural question is whether one could improve the converge rates or proofs of the TD algorithms by using the theory and assumptions from the HJB theorems. We will in this section briefly discuss some possible directions which could be explored, noting that most of these ideas goes beyond the scope of this thesis and has to be left for future work.

As noted in Item 2 in Section 4.4, the HJB equations requires that the value function satisfy some smoothness conditions. This assumption could potentially take care of a problem when using RL and function approximations. This is because it would mean that our approximated representations of the value- or Q-function remains within a certain bound of the functions true value. If this is not satisfied, we have no guarantee that the approximations used are not generalizing over too large areas of the state-action space. As noted in Section 4.3, several convergence results regarding $TD(\lambda)$ algorithms require this.

In this chapter, we have discussed differences and similarities in two ways of doing optimization, one rooted in mathematics and one rooted in informatics and statistics. While the mathematical framework has been used in finance for a long time, the RL framework has only recently gained popularity as a way of doing predictions and deciding prices in financial markets. In Chapter 5, we will look at some ways in which we can combine results from mathematical finance, which use models, and still end up with a Q-learning algorithm which is model-free. We will then use this method to price exotic options in power markets. We will also discuss whether this approach can be augmented to solve the problem discussed in Example 2.6.5, by using the optimality condition in Equation (4.6) and Equation (4.7) as an update rule.

CHAPTER 5

RL for pricing and hedging exotic options.

5.1 Introduction.

This section will look at applications in financial markets. These are markets where financial contracts are bought and sold, by agents that are either trying to reduce risk or gaining revenue. In this chapter we will focus on the use of reinforcement learning in the pricing of one type of contract known as an *option.* Options are financial instruments which allow the owner the opportunity, but not the obligation, to buy or sell an asset, $S(t)$, at a predetermined time, T, for a predetermined price, K. The asset $S(t)$ is typically modelled by an SDE, and we will in Section 5.3 describe a type of SDE that is used for financial assets. The most common type of options are *put options*, where the owner of the option has a profit of

$$\max\{K - S(T), 0\}, \tag{5.1}$$

and *call options*, where the owner has a profit of

$$\max\{S(T) - K, 0\}. \tag{5.2}$$

The predetermined price K, is commonly known as the *strike price.*

Owning an option reduces risk, as the agreed upon price protect him or her against large fluctuations in the market. Because of this, options are often used in markets where prices have high variation, or where prices fluctuate a lot over short time periods. For more on option theory, see [Ben03].

The subject of this section will be how we can use the model described by Halperin [Hal17] to price *exotic options*, that is options with more complex structures than the traditionally traded ones. Halperin named the method *QLBS* since his original goal was to use the Q-learner to compute the well-known Black-Scholes solution for the pricing of a put option. He did, however, note that the method allow for extensions well beyond simple put options. Our focus will be extending the method to allow for another (non-hedgeable) stochastic process influencing the price of the option. While Halperin presented the theoretical framework for the pricing and hedging of a standard put option, he did not present any numerical results. We will therefore also expand on Halperin's work by doing numerical experiments based on his method. The code for our implementation can be found in Appendix A.3.

Variable	Description:
K	Strike price for electricity.
L	Strike price for wind.
S(t)	Price of electricity.
W(t)	Wind speed.
σ	volatility of electricity prices.
η	volatility of wind speed.
$B(t)$	Brownian motion modeling stochastic part of electricity dynamics.
$\tilde{B}(t)$	(correlated) Brownian motion modeling stochastic part of wind dynamics.
$\hat{B}(t)$	(Uncorrelated) Brownian motion modeling stochastic part of wind dynamics.
ρ	Correlation between $B(t)$ and $\hat{B}(t)$.
μ	Long term mean of wind process.
α	Rate of mean reversion for the wind process.
$Z(t)$	Logarithm of the wind process.
$\Lambda(t)$	Function modelling the seasonal trend in the data.
$Z_1(t)$	First row of the multidimensional stochastic process $Z(t)$.
A	Matrix modelling the amount of autocorrelation.
$\Pi(t)$	Portefolio value at time t.
$u(t)$	Position in tradable stock at time t.
$I(t)$	Value in bank account at time t.
$e^{r\Delta t}$	Interest rate.
γ	Risk parameter.

Figure 5.1: List of notation for exotic options.

We will first describe how Halperin [Hal17] uses modern portfolio theory to find expressions for optimizing the price of options. We will then show that this method can be extended by introducing another process into the framework, and how this can be used in the reinforcement learning setting proposed by Halperin. We will end by discussing which conclusions can be made, and further extensions that possibly can be implemented.

5.2 Exotic options for power markets.

In energy markets, the price of electricity usually varies a great deal over short time periods, and a drop in temperature or a draught can greatly affect prices. Because of the uncertainty that this imposes on companies dealing with the supply or demand of large quantities of power, these agents are often willing to pay a higher price in order to insure, or hedge, against the fluctuation. This is the market of power options. In this market the agents can buy the right to purchase a certain amount of power, at a specified time or time interval, at a predetermined price. The price of these options need to reflect the uncertainty of the electricity price, which again is affected by factors such as temperature, rainfall and wind speed.

Halperin used his model for the case of a put option on a single asset with a single underlying stochastic process. We will extend on this, by showing that the same approach can be used on an option that is dependent on two correlated stochastic processes. One of them will represent the hedgeable stock, and the other will represent the non-hedgeable wind process.

We are putting ourselves in the position of a retailer of power. This retailer buys power at prices that varies greatly, and sells it to consumers at a price that is allowed far less fluctuations (they typically have to announce a price change at least a week before). Because of this, they stand to loose a lot of money if the price of power is low, or if the demand for power is low. This makes them interested in hedging their position, that is they want to buy an option to insure themselves against future losses. These future losses will depend on both the price of electricity and on temperatures or wind. The price of this option will be the topic of the remaining parts of this section.

We will consider a special case of option pricing based on the German market. In Germany the main source of energy is coal, but they also rely on wind power. In this market the sellers of coal energy are only allowed to sell when there is not enough wind energy to satisfy the demand for power. Therefore, the suppliers of coal energy stands to loose money if there is a lot of wind. We will therefore look at a put-call option defined by

$$\max\Big(K - S(T), 0\Big) \max\Big(W(T) - L, 0\Big), \tag{5.3}$$

where $S(T)$ is the price of power and $W(T)$ is the amount of wind. The owner of such an option will have the choice to sell power at a predefined price, K, called the *strike price*. In addition to this, however, the value of the option is also affected by the wind, and there is therefore an additional strike price, L, that represents the wind speed. This represents the buyer and seller's assumption of what can be considered "normal" wind speed, such that a large value of $W(T)$ increases the price of the option. The option is thus dependent on two stochastic processes, $S(t)$ and $W(t)$ and we will see that these processes are dependent.

RL in the QLBS model ◇.

Halperin's QLBS model uses the Q-function described in Section 3.3, but it does not utilize the whole flexibility allowed by the algorithm. Perhaps most noteworthy, it does not allow for the action, which in the intended QLBS case is the hedge, to affect the observed state space. This is essential for the batch learning part of the model, since we evaluate across all simulations simultaneously. It also means that the method can be implemented without use of tools such as *Tensorflow* or *Pytorch*, as we don't need to simulate a real-time environment. We also note that this reflects the fact that the hedges does not have any effect on the price of the option.

Since the hedge a_t does not affect the state of the asset we are also removing the need for *exploration* in our model. An important question in reinforcement learning is the balance between exploration and *exploitation*. By this we mean to what degree our method should utilize prior knowledge in order to learn versus using the new data which is gathered. When our agent is part of an interaction with his environment, making sub-optimal actions is an important part of gaining information of the environment. We will consider a *greedy* policy (see Section 3.2), meaning that we do no exploration. This also allows for the use of backward iteration when finding the optimal Q-function. If we were to consider a less greedy policy, this would not work, as we would need to know the value of V_t to calculate Q_{t+1}. One way of adding exploration to the model, could instead be to use consecutive batch learning. We would then use a policy in the simulation of a set of processes, perform QLBS on these, and use the computed action, but with some noise, in the simulation of the next.

The fact that the Q-learner is an off-policy method, as described in Section 3.3, transfers into the QLBS-model. This makes it possible to create an artificial data set of trading strategies, and use this with some data set of real stock trajectories. The hedges can be completely random, and our model will still discover to the correct option price, given that we feed our model enough data.

5.3 Modeling the dynamics ♢

The price of power, $S(T)$, will be modelled as a stochastic process. We will use a geometric Brownian motion as described in Example 1.4.11, but since $S(T)$ is a forward price, and not a spot price, we will have $r = 0$. We therefore get that the process is defined by

$$dS(t) = \sigma S(t) dB(t). \tag{5.4}$$

When modeling the wind dynamics, we will use two slightly different approaches, depending on whether we are using real or simulated data. We will consider the case of simulated data first, as this is the simplest approach. A list of all the notation can be found in Section 5.2.

Modelling the wind process using simulated data.

To simulate the stochastic process for the wind dynamics, $W(t)$, we use the exponential of a Ornstein-Uhlenbeck process, $Z(t)$. This process is typically used for modelling wind, temperature or rain water, among other things because of its mean-reverting property (see [BBK08]). It will be based on a different Brownian motion, $\tilde{B}$:

$$W(T) = \exp(Z(T)), \tag{5.5}$$

$$dZ(t) = (\mu - \alpha Z(t))dt + \eta d\tilde{B}(t), \tag{5.6}$$

$$d\tilde{B} := \rho dB + \sqrt{1-\rho^2} d\hat{B}, \tag{5.7}$$

where B is the Brownian motion used in Equation (5.4), $\hat{B}$ is a different Brownian motion independent on B, and $\rho \in (-1, 1)$ is the correlation factor between $\tilde{B}$ and B. The strike K will depend on the current spot price while the strike L will depend on what we constitute as "normal" wind.

This problem is essentially a problem of estimating a function of two uncorrelated Brownian motion. We see this because

$$\max(K - S(T), 0) \max(\exp(Z(T)) - L, 0) \tag{5.8}$$

can be written as a function of the two underlying Brownian motions:

$$f_1(B(T))f_2(\tilde{B}(T)), \tag{5.9}$$

where f_1 represents the first max-function and f_2 represents the second max-function. By using Equation (5.7) we can write this as

$$f_1\big(B(T)\big)f_2\big(\rho B(T) + \sqrt{1-\rho^2}\hat{B}(T)\big) \tag{5.10}$$

$$:= g\big(B(T), \hat{B}(T)\big). \tag{5.11}$$

Which shows that the option is a function of two uncorrelated Brownian motions.

While this stochastic process will capture the dynamics of the wind process when using simulation, real data requires a more complex model to capture seasonal effects.

Modelling the wind process using real data.

When modeling processes like wind, temperature or rain we need our process to reflect the seasonal attributes these processes display. Wind will typically fluctuate over the course of a year, with high wind speed being more common in some seasons than others. One simple way of addressing this trend in the data, described by Benth [BBK08], is to define the wind process as the sum of a seasonally dependent, deterministic function Λ, and a stochastic noise term Z_1:

$$\exp(W(t)) = \Lambda(t) + Z_1(t). \tag{5.12}$$

This gives us a parametric function, which captures both the periodic nature of the seasonal effects, and a possible evolution over time. Since we expect the seasonally dependent part to fluctuate between high and low values we can define a seasonality function by

$$\Lambda(t) := a_0 + a_1 t + a_2 \cos(2\pi(t - a_3)/365). \tag{5.13}$$

We can then perform a regression on true wind data using this function. This lets us find estimates for the parameters, a_0, a_1 and a_2. Then we can add it to our modeled stochastic part.

Another feature of the wind process is the presence of autocorrelation. The wind speed of today will often be correlated with the wind speed of the preceding days. In order for our model to keep its Markov property we need to be able to write our wind dynamics as a function of a non-correlated stochastic process. We can incorporate this into our model by using a *CAR(p)-model* to express the wind dynamics . It models a continuous autoregressive stochastic process, (CAR). Autoregressive means that there is correlation between the process at times t and $t + k$ for $k = 0, 1, \dots, p$. It is expressed by

$$dZ(t) = AZ(t)dt + e_p(t)\sigma(t)d\tilde{B}(t), \tag{5.14}$$

where, for the correlation factor $\rho \in (-1, 1)$,

$$d\tilde{B} := \rho dB + \sqrt{1-\rho^2} d\hat{B}, \tag{5.15}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \\ -\alpha_p & -\alpha_{p-1} & \alpha_{p-2} & \cdots & \alpha_1 \end{bmatrix}, \tag{5.16}$$

$Z(t)$ is the deseasonalized wind dynamics, and $\tilde{B}(t)$ is a random noise term modeled by a Brownian motion. B and $\hat{B}$ are as described in the former section. By defining the process in this way, we are still able to express the payoff-function as a function of a standard Brownian motion, which is necessary to use the framework of the QLBS-model. Also, $\max(K - S(T), 0) \max(\exp(Z(T)) - L, 0)$ can still be written as a function of two uncorrelated Brownian motions, by the same argument as in the previous section. To find the appropriate p it is necessary to look at relevant wind data, and study correlation between the time periods (see more in [BBK08]).

5.4 Defining the portfolio and the hedge

This section will consist of two parts. The first part will describe how we can formulate our portfolio optimization problem into a reinforcement problem, through the use of the Q-function. The second part will explain how we can find explicit expressions for the updates for this function, enabling us to use batch learning on all the paths of the stochastic process for each time step.

We will begin by describing the portfolio problem, and how a self-financing portfolio can be used to find the "fair" price. By fair we mean a price that ensures that neither the seller nor the buyer of the option is profiting, if we would be able to consider all the possible values of the option.

The portfolio $\diamondsuit$

This section follows the methods proposed by Halperin [Hal17]. While Halperin described the method for the classic Black-Scholes case of a put-option, we will extend it to the case of a put-call option dependent on two assets. As we will see, the inclusion of a second non-tradeable asset does not introduce much change to the original model, as the recursiveness of the model comes from the self-financing constraint in the replicating portfolio. Since the portfolio only consists of the bank account and the tradable asset we are left with the same expressions as in Halperins Black-Scholes case.

We are interested in valuation and hedging of an option on a given asset, for which we have some historic or simulated data points. These data points contain information on the asset's spot price for a given time period, at certain time intervals. The following sections are based on the theory of hedging by a replicating portfolio, a standard setting of mathematical finance [BBK08].

We start out by observing the price dynamics of the given stock. We consider a seller of a put-call option with terminal value

$$H_T(S_T, W_T) := \max\big(K - S_T, 0\big) \max\big(W_T - L, 0\big), \tag{5.17}$$

as described in Section 5.2, where S_t is the stock price and W_t is the exponential of either a geometric Brownian motion or for the case of available real wind data, a CAR(p)-process. K and L denote the strike prices. Since W_t represents the wind dynamics, we will only allow for trading in the stock, S_t. In order to hedge the option, the seller will set up a replicating portfolio that consists of the tradeable stock and a bank account I_t, which at any time $t \leq T$ will have the value

$$\Pi_t = u_t S_t + I_t, \tag{5.18}$$

where u_t is the position in the stock S_t and I_t is a risk-free bank account. This portfolio is closed at the time of maturity $t = T$, at which point our goal is to have its value perfectly replicating the value of the option. Closing the portfolio means setting u_T to 0. This gives us a condition for the portfolio at maturity:

$$\Pi_T = I_T. \tag{5.19}$$

Our goal is to make this as close (in the standard Euclidean norm) to the value of the option $H_T(S_T, W_T)$ as possible.

We note that this portfolio is the same as the one shown by Halperin [Hal17]. Even though the option in our case is based on two stochastic processes, the problem is reduced to the same mathematical expressions as Halperin's once we have set up the portfolio, since the portfolio is only based on the tradeable asset. In fact, because the updates to the Q-function used by Halperin is based on the replicating portfolio, any portfolio with only one tradeable asset will behave in exactly the same way. This means that we can use the method for a wide range of options. The rest of this section will therefore mimic the approach of [Hal17]. We note, however, that even the introduction of more tradable assets would follow the same approach, and could be incorporated at the cost of some computational complexity.

The recursive relation $\diamondsuit$

The main idea behind our pricing scheme is that if we are able to find a recursive way of expressing the value of the option, we can back trace from the terminal condition all the way to $t = 0$. We therefore need a way of expressing Π_t as a function of Π_{t+1}. To achieve this, we set a self-financing constraint. This means that we demand that all changes to the portfolio, i.e. all hedges, are financed by a set bank account. We do not allow for money to be invested in the portfolio after $t = 0$, so the change of (time-discounted) value in the portfolio must be equal to the change of (time-discounted) value in the stock. This can be expressed as:

$$e^{r\Delta t}\Pi_t(B_t) - \Pi_{t+1}(B_{t+1}) = e^{r\Delta t} u_t S_t - u_t S_{t+1}. \tag{5.20}$$

We will in the following sections use the notation

$$\gamma = e^{-r\Delta t}, \tag{5.21}$$

where r is the interest rate, so that γ discounts future values to their present value.

Using Equation (5.20), we can express Π in a recursive manner as:

$$\Pi_t(B_t) = \gamma[\Pi_{t+1} - u_t \Delta S_t], \quad \text{where } \Delta S_t = S_{t+1} - e^{r\Delta t} S_t. \tag{5.22}$$

Here, B_t is the Brownian motion representing the stochastic part of the asset S_t. As a general note, we will write all functions of S_t as functions of B_t to emphasise that this is a prerequisite to be able to formulate our problem as a Markov decision process. If we can't do this, we are no longer in the setting of reinforcement learning. We also note that $\Delta S_t = S_{t+1} - e^{r\Delta t} S_t$ is simply the change in value of the stock price S_t, and $e^{r\Delta t}$ is an interest rate factor setting the value of S_t to the present value of $t+1$.

The point of the hedge, or action, is to minimize the risk associated with the portfolio. Therefore, the optimal action is one that minimizes the variance of the portfolio value,

$$u^*(B_t) = \arg\min_u \text{Var}[\Pi_t | \mathcal{F}_t]. \tag{5.23}$$

This is the greedy policy that the Q-learning algorithm is trying to learn (see Section 3.2). Here, the history $\mathcal{F}_t$ is the history of the cross-sectional data, that is the data from *all* the paths of the stochastic process up to time t.[1] From the standard formula for variance,

$$\begin{aligned}
\text{Var}[\Pi_t|\mathcal{F}_t] &= \gamma^2 \mathbb{E}_t[(\Pi_t - \mathbb{E}[\Pi_t])^2] && (5.24)\\
&= \gamma^2 \mathbb{E}_t[(\Pi_{t+1} - u_t\Delta S_t - \mathbb{E}[\Pi_{t+1} - u_t \Delta S_t])^2] && (5.25)\\
&= \gamma^2 \mathbb{E}_t[(\hat{\Pi}_{t+1} - u_t \Delta \hat{S}_t)^2], && (5.26)
\end{aligned}$$

where we for simplification have used the notation

$$\hat{x} := x - \mathbb{E}[x]. \tag{5.27}$$

We can then find the optimal value of u_t by computing the derivative, solve for u, and set the equation equal to zero:

$$\begin{aligned}
\frac{\partial \text{Var}[\Pi_t|\mathcal{F}]}{\partial u} &= \gamma^2(-2\mathbb{E}_t[\Pi_{t+1}\Delta\hat{S}_t] + 2u_t \mathbb{E}_t[\Delta \hat{S}_t^2] && (5.28)\\
\implies u_t^* &= \frac{\mathbb{E}_t[\hat{\Pi}_{t+1}\Delta\hat{S}_t]}{\mathbb{E}_t[\Delta\hat{S}_t^2]} && (5.29)\\
&= \frac{\text{Cov}(\Pi_{t+1}, \Delta S_t|\mathcal{F}_t)}{\text{Var}(\Delta S_t|\mathcal{F}_t)}. && (5.30)
\end{aligned}$$

By defining the optimal action in this way, we have defined it as a way of controlling *all* the variance of the portfolio. In general, an owner of a portfolio would like to maximize his monetary value, and would not be concerned with the possibility of gaining a lot of money. One possibility could therefore be to instead use the *conditional value at risk* (CVaR) measure to define the optimal action (see for example [RUZ02]). We will however leave this idea to future research, and continue using Equation (5.23) as the optimal hedge.

[1]We note that the value of the hedge can only be dependent on history up to this point, as it is performed without knowledge of the future.

The action-value function

We will now present the value function, which represents the value of the portfolio. The optimal value of this function at time $t = 0$ will be the price of the option.

Halperin's [Hal17] choice of option price is based on an idea by Potters et al. [PBS01]. The idea is that the value of the option at time t is equal to the value of the portfolio $\Pi(t)$, plus the future risk associated with the portfolio. The price then also takes into account the risk faced by the seller and can be expressed by the value function:

$$V_t(B_t) := \mathbb{E}\left[\Pi_t + \lambda \sum_{t'=t}^{T} e^{-r(t'-t)} \mathrm{Var}[\Pi_{t'}|\mathcal{F}_{t'}]\middle|\mathcal{F}_t\right], \tag{5.31}$$

where λ is a parameter stating how risk averse the seller of the option is. Since the risk of the option can't be eliminated completely in a realistic setting, this gives us a way of quantifying how an increase in either risk, or risk aversion will affect the price of the option. The minimization of this value function will give the price of the option at time t, as we are trying to find the hedge that reduce future variance (i.e. risk) of the portfolio. By changing the sign, we can equivalently state the problem as a maximization of

$$V_t(B_t) = \mathbb{E}\left[-\Pi_t - \lambda \sum_{t'=t}^{T} e^{-r(t'-t)} \mathrm{Var}[\Pi_{t'}|\mathcal{F}_{t'}]\middle|\mathcal{F}_t\right]. \tag{5.32}$$

By moving the first part outside the summation, we then get

$$V_t(B_t) = \mathbb{E}\left[-\Pi_t - \lambda \mathrm{Var}[\Pi_t] - \lambda \sum_{t'=t+1}^{T} e^{-r(t'-t)} \mathrm{Var}[\Pi_{t'}|\mathcal{F}_{t'}]\middle|\mathcal{F}_t\right]. \tag{5.33}$$

We can then use the recursive nature of Π_t to write this expression as a function of V_{t+1}:

$$\begin{aligned} V_t(B_t) &= \mathbb{E}_t^{\pi}[\gamma a_t \Delta S_t(B_t, B_{t+1}) - \lambda \mathrm{Var}(\Pi_t|\mathcal{F}_t) + \gamma V_{t+1}(B_{t+1})] && (5.34)\\ &= \mathbb{E}_t[R_t(B_t, a_t, B_{t+1}) + \gamma V_{t+1}^{\pi}(B_{t+1})], && (5.35) \end{aligned}$$

where we have defined R as

$$R_t(B_t, a_t, B_{t+1}) := \gamma a_t \Delta S_t(B_t, B_{t+1}) - \lambda \mathrm{Var}(\Pi_t|\mathcal{F}_t). \tag{5.36}$$

We have now restated our initial value function into an expression involving one term representing the reward signal $R(\cdot)$, and one term representing the discounted future value $V(\cdot)$, as described in Section 3.2. We will use the notation $V^*(\cdot)$ for the optimal value function in the following sections. We also remark that the value function is dependent on the policy π.

In the framework of the QLBS-model, we allow for the dynamics of the system to be unknown. We will therefore not optimize the value function, but instead the similar Q-function, as described in Section 3.3. It is defined as the expected value of the value-function given the state of the system, $B_t = x$, and the action performed on this state, $a_t = a$, and conditioned on following a policy π' for the remaining time periods.[2] We then get that the RL algorithm is performed on the Q-function

$$Q_t(x,a) = \mathbb{E}_t[R_t(B_t, a_t, B_{t+1})|B_t = x, a_t = a] + \gamma \mathbb{E}_t[V^*_{t+1}(B_{t+1})|B_t = x]. \tag{5.37}$$

The optimized value function is the same as the optimized Q-function when maximizing a over the next time step:

$$V^*_{t+1}(B_{t+1}) = \max_{a_{t+1}\in\mathcal{A}} Q^*_{t+1}(B_{t+1}, a_{t+1}). \tag{5.38}$$

Therefore, Equation (5.37) can be formulated as

$$Q_t(x,a) = \mathbb{E}_t[R_t(B_t, a_t, B_{t+1}) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q^*_{t+1}(B_{t+1}, a_{t+1})|B_t = x, a_t = a]. \tag{5.39}$$

At $t = T$ the hedge is set to zero, and we get the terminal condition from Equation (5.31):

$$Q^*_T(B_T, a_T = 0) = -\Pi_T(B_T) - \lambda \mathrm{Var}(\Pi_T(B_T)). \tag{5.40}$$

We note that, from Equation (5.38), maximizing the Q-function with respect to a gives the optimal value function, and that the optimal strategy, i.e. the optimal set of hedges, is given as

$$\pi^*_t(B_t) = \arg\max_{a_t\in\mathcal{A}} Q^*_t(B_t, a_t). \tag{5.41}$$

The Q-function described in Equation (5.39) involves the expected values of processes which we don't know the distribution of. In the next section, we will therefore use the Monte Carlo paths to approximate them. This will allow us to use regression methods to find optimal values of Q and a.

5.5 Learning the price of the option and the hedge

We assume we have some set of basis functions $\{\Phi_n(x)\}$. This can for example be spline functions, polynomials, or a Fourier basis (see [SB18] for more on choosing basis functions). The number of basis function will need to be selected through either prior knowledge or numerical experiments. More complex dynamics will typically require a larger number of basis functions in order to accurately represent the trajectories. This increase will, however, lead to a quite significant increase in the computational complexity [SB18].

We can express the optimal action and Q-function as

$$a^*(B_t) = \sum_{n=0}^{M} \phi_{n,t} \Phi_n(B_t) \tag{5.42}$$

$$Q^*(B_t, a^*_t) = \sum_{n=0}^{M} \omega_{n,t} \Phi_n(B_t), \tag{5.43}$$

[2]We remember that we are assuming that the states are Markov, so this means that all relevant information on the future state B_{t+1} is present in x and a.

for $\phi \in \mathbb{R}$, $\omega \in \mathbb{R}$, and where M is the dimension of the basis, i.e. the number of basis functions.

Our goal is to perform regression on the Q-function in order to find values for the weights $\phi_{n,t}$ and $\omega_{n,t}$. We will start by considering the action, and it's weights $\omega_{n,t}$.

Finding the optimal action $\diamondsuit$

To find the values of $\omega_{n,t}$, we start with the expression for Q. By inserting the expression for R in Equation (5.36), and the expression for the optimal hedge in Equation (5.26) into Equation (5.39) we get:

$$Q^*_t(B_t, a_t) = \gamma \mathbb{E}_t[Q^*_{t+1}(B_{t+1}, a^*_{t+1}) + a_t \Delta S_t] - \lambda\gamma^2 \mathbb{E}_t[(\hat{\Pi}_{t+1} - u_t \Delta \hat{S}_t)^2]. \tag{5.44}$$

Writing out the last quadratic expression, we get:

$$Q^*_t(B_t, a_t) = \gamma \mathbb{E}_t[Q^*_{t+1}(B_{t+1}, a^*_{t+1}) + a_t \Delta S_t] - \lambda\gamma^2 \mathbb{E}_t\left[\hat{\Pi}^2_{t+1} - 2a_t \hat{\Pi}_{t+1}\Delta \hat{S}_t + a_t^2 (\Delta \hat{S}_t)^2\right]. \tag{5.45}$$

We then substitute the expectation with the MC estimate, as described in Section 3.3:

$$\begin{aligned} Q^*_t(B_t, a_t) = {} & \gamma \frac{1}{N_{MC}} \left(\sum_{k=1}^{N_{MC}} Q^{k,*}_{t+1}(B^k_{t+1}, a^*_{t+1}) + a_t \Delta S^k_t \right) \\ & - \lambda\gamma^2 \frac{1}{N_{MC}} \left(\sum_{k=1}^{N_{MC}} (\hat{\Pi}^k_{t+1})^2 - 2a_t \hat{\Pi}^k_{t+1} \Delta \hat{S}^k_t + a_t^2 (\Delta \hat{S}^k_t)^2 \right), \end{aligned} \tag{5.46}$$

where N_{MC} is the number of Monte Carlo paths. The number of MC paths will need to be determined by doing numerical experiments, or by using the theory on confidence bounds discussed in Section 3.3.

Since we are performing minimization on the weights of the expansion of a, we can remove the a_t-independent terms. This includes Q^*, as the action, equivalently the hedge, should not affect the *optimal* Q-value:

$$Q^*_t(B_t, a_t) = \sum_{k=1}^{N_{MC}} \left(a_t \Delta S^k_t - \lambda\gamma \left[(\hat{\Pi}^k_{t+1})^2 - 2a_t \hat{\Pi}^k_{t+1} \Delta \hat{S}^k_t + a_t^2 (\Delta \hat{S}^k_t)^2 \right] \right). \tag{5.47}$$

We can now substitute a_t with the basis expansion, and change the sign in order to make it into a minimization problem[3]:

$$\begin{aligned} Q^*_t(B_t, a_t) = \sum_{k=1}^{N_{MC}} \Bigg(& - \sum_{n=1}^{M} \phi_{n,t} \Phi_n(B^k_t) \Delta S^k_t \\ & + \lambda\gamma \left[(\hat{\Pi}^k_{t+1})^2 - 2 \sum_{n=1}^{M} \phi_{n,t} \Phi_n(B^k_t) \hat{\Pi}^k_{t+1} \Delta \hat{S}^k_t + \left(\sum_{n=1}^{M} \phi_{n,t} \Phi_n(B^k_t) \Delta \hat{S}^k_t \right)^2 \right] \Bigg) \end{aligned} \tag{5.48}$$

[3]Note that the minimization is with respect to the weights of the hedge, not the value function.

$$= \sum_{k=1}^{N_{MC}} \left(- \sum_{n=1}^{M} \phi_{n,t} \Phi_n(B_t^k) \Delta S_t^k + \lambda\gamma \left((\hat{\Pi}_{t+1}^k)^2 + \sum_{n=1}^{M} \phi_{n,t} \Phi_n(B_t^k) \Delta \hat{S}_t^k \right)^2 \right). \tag{5.49}$$

Having expressed Q^* as a function of the unknown basis weights and data from the samples, we now want to find the optimal weights. We therefore differentiate with respect to $\phi_{n,t}$, and set equal to zero:

$$\sum_{k=1}^{N_{MC}} \Bigg(- \Phi_n(B_t^k) \Delta S_t^k \tag{5.50}$$

$$+ \lambda\gamma \left[- 2\Phi_n(B_t^k) \hat{\Pi}_{t+1}^k \Delta \hat{S}_t^k + 2 \left(\sum_{n=1}^{M} \phi_{n,t} \Phi_n(B_t^k)^2 (\Delta \hat{S}_t^k)^2 \right] \right) \tag{5.51}$$

$$= 0. \tag{5.52}$$

Solving this for each $\phi_{n,t}$ gives us the following set of equations:

$$\sum_{k=1}^{N_{MC}} \sum_{n=1}^{M} \phi_{n,t} \Phi_n(B_t^k) \Phi_m(B_t^k) (\Delta \hat{S}_t^k)^2 \tag{5.53}$$

$$= \sum_{k=1}^{N_{MC}} \left(\Phi_n(B_t^k) \hat{\Pi}_{t+1}^k \Delta \hat{S}_t^k + \frac{1}{2\lambda\gamma} \sum_{k=1}^{N_{MC}} \Phi_n(B_t^k) \Delta S_t^k \right). \tag{5.54}$$

This can be written as

$$\boldsymbol{A}_t \boldsymbol{\phi}_t = \boldsymbol{B}_t \tag{5.55}$$

$$\implies \boldsymbol{\phi}_t = \boldsymbol{A}_t^{-1} \boldsymbol{B}_t, \tag{5.56}$$

where $\boldsymbol{A}_t$ is an $M * M$ matrix with elements

$$A_{n,m} = \sum_{k=1}^{N_{MC}} \Phi_n(B_t^k) \Phi_m(B_t^k) (\Delta \hat{S}_t^k)^2, \tag{5.57}$$

$\boldsymbol{B}_t$ is a vector with elements

$$B_n = \sum_{k=1}^{N_{MC}} \left[\Phi_n(B_t^k) \hat{\Pi}_{t+1}^k \Delta \hat{S}_t^k + \frac{1}{2\lambda\gamma} \sum_{k=1}^{N_{MC}} \Phi_n(B_t^k) \Delta S_t^k \right], \tag{5.58}$$

and $\boldsymbol{\phi_t}$ is a vector of the optimal weights.

Finding the Q-function.

Our next step is to find the optimal Q-function. This means that we need to find the weights to our basis function that maximizes the expression from Equation (5.39). To this end, we can express our optimality condition, that is our Bellman equation (see Section 3.2), as

$$R_t(B_t, a_t^*, B_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(B_{t+1}, a_{t+1}) = Q_t^*(B_t, a_t^*) + \epsilon_t, \tag{5.59}$$

where ϵ_t is the irreducible error (see Section 3.1) of our model estimate at time t. We assume this to be random noise, with $\mathbb{E}(\epsilon_t) = 0$. When $a_t = a_t^*$, the expectation of this is equivalent to the expectation of the Bellman equation defined in Equation (5.39). We can therefore find the optimal Q-value by performing least-squares minimization on

$$F_t(\omega) = \sum_{k=1}^{N_{MC}} \left(R_t(B_t, a_t^*, B_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(B_{t+1}, a_{t+1}) - \sum_{n=0}^{M} \omega_{n,t} \Phi_n(B_t^k) \right)^2. \tag{5.60}$$

Differentiating with respect to each ω_i and setting equal to zero gives us the following expression:

$$\begin{aligned} \frac{\partial F_t(\omega)}{\omega_i} = \sum_{k=1}^{N_{MC}} \big[&- 2(R_t(B_t, a_t^*, B_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(B_{t+1}, a_{t+1}))\Phi_i(B_t^k) \\ &+ 2\omega_i \Phi_i(B_t^k)\big] \\ &= 0. \end{aligned} \tag{5.61}$$

This implies that

$$\begin{aligned} \sum_{k=1}^{N_{MC}} \Big[(R_t(B_t, a_t^*, B_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(B_{t+1}, a_{t+1}))\Phi_i(B_t^k) \Big] \\ = \sum_{k=1}^{N_{MC}} \Big[\omega_i \Phi_i(B_t^k) \Big]. \end{aligned} \tag{5.62}$$

This can be expressed on matrix notation, as

$$\boldsymbol{C}_t \omega_t^* = \boldsymbol{D}_t, \tag{5.63}$$

where

$$C_{n,m}^{(t)} = \sum_{k=1}^{N_{MC}} \Phi_n(B_t^k)\Phi_m(B_t^k), \tag{5.64}$$

$$D_n^{(t)} = \sum_{k=1}^{N_{MC}} \Phi_n(B_t^k) \Big(R_t(B_t, a_t^*, B_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(B_{t+1}, a_{t+1}) \Big). \tag{5.65}$$

Here, C_t is an $M \times M$ matrix, and D_t is an M-dimensional vector. We then get that the optimal weights are given as:

$$\omega_t^* = \boldsymbol{C}_t^{-1} \boldsymbol{D}_t. \tag{5.66}$$

Inserting ω_t^* into Equation (5.43) gives us the optimal Q-function. Our initial goal was to maximize the Q-function in order to find the optimal fair asking price for the option. Since we changed the sign of the Q-function and made it into a minimization problem, we get that the negative of value of this function at time $t = 0$ gives us the optimal price for the option.

parameter:	value
σ	0.4
r	0.05
S0	100
K	100
μ_w	0.05
T	1
n	25
λ	0.00001
corr	-0.9

Figure 5.2: Parameter values for simulations.

We note that this method should work for both continuous and discrete time, although only the discrete case has been proved to converge to the true optimal option price (see Section 3.4). We also have a risk aversion parameter that measures the option sellers perceived risk. This allows for using the same method both for the no-arbitrage case, and for the more realistic case where there is some risk, and therefore some reward perceived by the sellers and buyers of the options.

5.6 Numerical experiments $\diamondsuit$

In this section we include a numerical experiment for the case of a put-call option which is dependent on both a tradable asset representing the energy price, and a correlated non-tradable asset representing the wind speed. This is an extension of Halperin's [Hal17] work, as he did not any numerics in his paper, and also because of the inclusion of a second process. The parameters for our simulations can be found in Figure 5.2.

We start by simulating the paths of the tradable asset S_t and the non-tradable wind process W_t. We will use 10000 simulations. The plots from the simulation of the two correlated Brownian motions, S_t and W_t can be found in Figure 5.3, where we see the asset and wind process in the lower row, while the upper row shows the Brownian motions. We create a third degree polynomial as a basis to represent our data points, and create a feature matrix of these points.

We can now start our computation of a_t and Π_t. This is done by computing our way back from the terminal value T, by using equations Equation (5.57), Equation (5.58), and Equation (5.56), and putting this into Equation (5.42). We can then find Π_t by putting the optimal hedge a_t^* into Equation (5.22). Plots of a_t and Π_t is pictured in Figure 5.4. The hedges (a) are all equal at $t = 0$, as they respond to the same initial value. We see that the value of the hedges has a larger spread as we move along the time-axis. This is because the

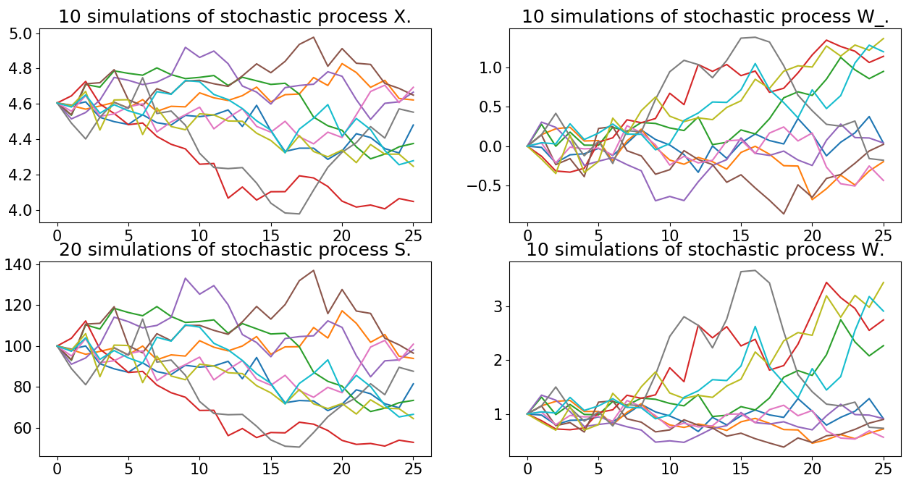

Figure 5.3: Plot of Brownian motions(upper row), and S and W (bottom row).

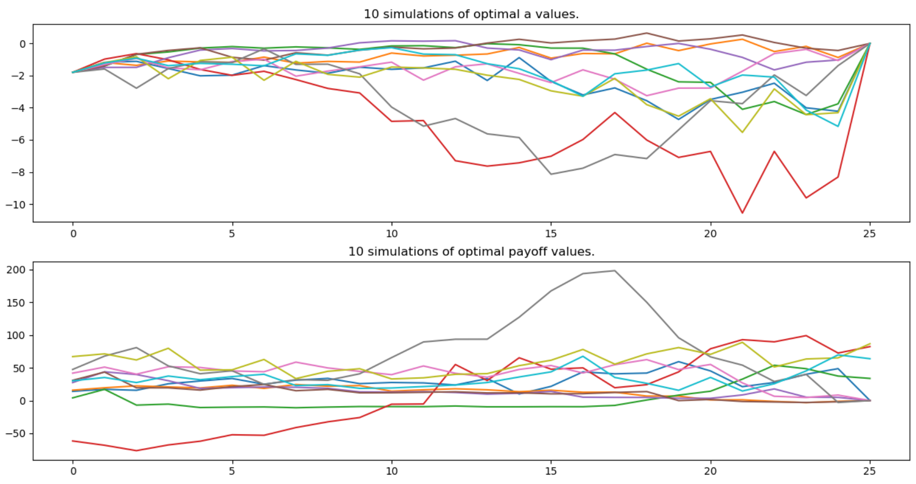

Figure 5.4: Plots of 10 paths of computed a values(hedges) and payoff values.

underlying stochastic processes also moves in a wider range as time passes, and the hedges needs to become bigger to control these variations. At $t = 25$ the portfolio closes, and the hedges are therefore all set to zero.

Having found the optimal value for a_t^*, we can now use Equation (5.64) and Equation (5.65) in Equation (5.66) to find the optimal Q-function. The negative of this function at time $t = 0$ will be the correct price. Figure 5.5 shows the paths of the Q-values. We get that the price of the option is 29.425. As a comparison, the Monte Carlo estimate for the option price is 31.944. The entire computation time is 1.43 seconds.

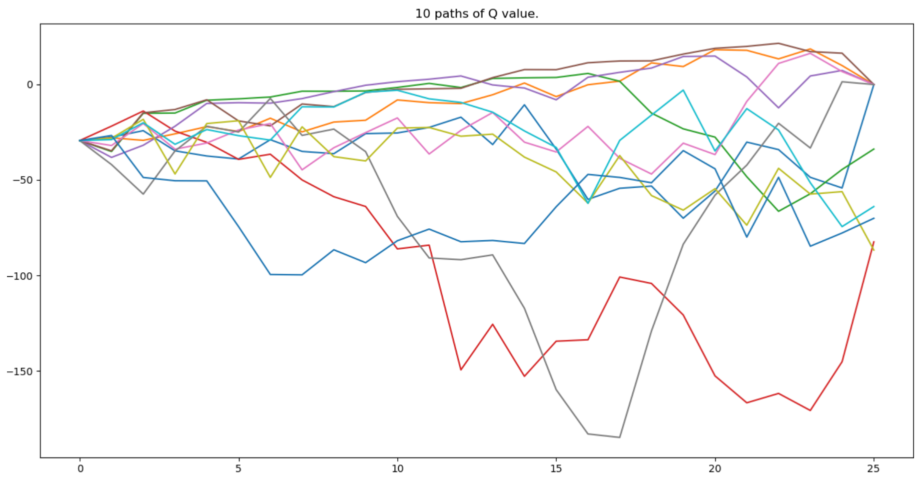

Figure 5.5: Plot of 10 paths of computed Q-function.

5.7 Discussion

In this section we have implemented the QLBS method for the case of two correlated stochastic processes, where only one of them is hedgeable. We have built on the work of Halperin, both by implemented the method which he proposes in his paper, and also by extending it to two processes. By doing this, we both show the validity of Halperins method, while also showcasing its usefulness in working with exotic options. The framework and code can easily be transformed into different options where only one of the assets are hedgeable. We only need to change the expression for our terminal value and make sure that this is a function of the underlying Brownian motions, the Q-function and our regression problem will stay the same. We have also noted that additional hedgeable assets can be incorporated, but this will lead to somewhat more complex computations of the matrices A_t, B_t, C_t and D_t. This implies a great flexibility compared to other methods of hedging and pricing which are more dependent on the type of option being considered. We are also extending the model without almost any delay in the numeric speed. The added stochastic process only influences the starting value of the Q-function, and will not add extra computations to the updating of the function. When simulating data we are only decreasing the computational speed by having to simulate another process and computing the option value for each simulation, while in the case of real data, only the latter of these will create any delay. This makes the method proposed in this report fairly efficient compared to other alternative approaches.

We are however still in the somewhat stylized setting of complete model knowledge. Because of this, the application of the QLBS-method described in this chapter does in many ways perform the same task as dynamic programming. It is in the case of incomplete knowledge of the model that the methods of reinforcement learning may contribute something completely new to the table. In the setting we have described, we have looked at the case where we observe some path of the asset price and wind dynamics, and where we are able to compute the reward as we work our way back from $t = T$. In the batch

Q-learning described by Halperin [Hal17], we are instead observing samples from the reward function and the action function. We have more data, but less prior knowledge of the model. It is in this setting that the QLBS-model might present completely new results.

Another way forward would be to consider real data of prices and wind, and compute the option price based on this. In this case, one could use the seasonal models proposed in section 3.1, and thus allow for seasonal variations in our stochastic wind process. The focus of this section has been to implement an extension of Halperin's approach, and we therefore leave the case of real historical data as a possibility for future work.

5.8 The control problem in the QLBS setting

In both Example 2.6.5, and the QLBS-approach to pricing exotic options in Chapter 5, we have studied the case of financial portfolios. We have used the two frameworks, HJB and RL, to predict future values of portfolios, and used hedges to optimize wealth. The two setups presented have many common traits. We will in this section discuss which necessary components are needed to be able to use the QLBS framework for an optimization problem. We will then describe the differences and similarities between the problem from Chapter 5 and Example 2.6.5, and formulate the example from Example 2.6.5 in the RL setting. We will end the section by discussing problems with our approach.

Necessary conditions for the QLBS model

In order to use the method presented in Chapter 5, there are four important aspects that needs to be present:

1. The risky process need to be a martingale, or we need to be able to transform it into a martingale.

2. The stochastic process that we are performing the optimization on (i.e. the portfolio) needs a recursive relation. This means that we need some way of linking the present and future reward of the q-function for each time point t, where $0 \geq t < T$.

3. We need to have some terminal condition, which provides us with the value of the Q-function for time T, for each of the samples.

4. The action (i.e. hedges) need to be quadratic, or on some other form that will provide us with a maximum solution. If this is not satisfied, we are no longer able to express the solution by Equation (5.56) and Equation (5.66).

. The first assumption is needed to remove the drift and make the mean constant. The second assumption is essential for all RL problems, and ensures that we can use the Bellman equations. Without it, we are no longer able to use future optimal values to find the present optimal value.

The third assumption is closely related to the Bellman conditions. We need this in order to have a "starting point" for our optimal chain of value functions. This is always satisfied for a finite trajectory in an RL framework, as the value function or Q-function will equal the reward signal, which is considered exogenous to the problem.

The assumption in Item 4 is a consequence of the use of least squares. In Chapter 5, we found an expression for the optimal action, in the example the hedge, which made the errors quadratic. The quadratic errors made for a rather simple computation, but essentially, the important part is that the errors made it possible to do least squares minimization on the Q-function. This is rather similar to a condition required for using the HJB-theorems in Chapter 2. Here, we see that if we can not have quadratic or other similarly solvable solutions, then the control u disappears when differentiating on the operator-expression in Equation (2.52), and we are not able to solve the problem by using differentiation.

Comparing the two portfolio problems

In this section, we will comment on the differences and similarities of the two examples.

In chapter two we used the HJB-theorem to maximize the utility of a portfolio. This portfolio consisted of a bank account

$$dX(t) = \rho X(t)dt, \tag{5.67}$$

and a risky asset

$$dS(t) = S(t)\mu dt + S(t)\sigma dB(t), \tag{5.68}$$

This setup is in fact very similar to the original problem presented by Halperin in [Hal17], where he used the framework to find the Black-Scholes solution. The stock S_t and bank account B_t is defined identically, but while Halperin defined the portfolio as

$$\Pi_t = u_t S_t + B_t, \tag{5.69}$$

meaning that u_t represents the amount of stocks owned at time t, we defined it in Example 2.6.5 as the fraction of wealth owned at time t. We could then directly describe the dynamics by

$$\begin{aligned} dZ(t) &= dS(t) + dX(t) && (5.70)\\ &= Z(t)[(\rho(1-u(t)) + \mu u(t))dt + \sigma u(t)dB(t)]. && (5.71)\end{aligned}$$

In Halperin's approach, we defined the optimal hedge, u_t as

$$u^*(B_t) = \arg\min_u \mathrm{Var}[\Pi_T|\mathcal{F}_t], \tag{5.72}$$

in other words, as the action that minimize the variance, conditional on the filtration generated by the samples of data. This gave us an analytic expression of u^*, and one of the important features of this definition was that it made the errors quadratic.

When optimizing u in Example 2.6.5, we are instead trying to find u^* given by

$$\begin{aligned} u^*(B_t) &= \arg\max_u (U(\Pi_t)|\mathcal{F}_t) && (5.73)\\ &= \arg\max_u ((\Pi_t)^\gamma|\mathcal{F}_t). && (5.74)\end{aligned}$$

This is not quadratic in u_t, which makes Halperin's scheme impossible to use, at least the version which produce the matrices in Equation (5.57) and Equation (5.58).

Using the HJB-equations as an Q-learner update rule

We will in this section discuss a possible approach to solving the portfolio problem, and discuss problems that arise with Halperin's [Hal17] method. We will use both the update and terminal condition given from Example 2.6.5, and try to find the optimal value for the utility of the portfolio, Z_t, given by $(Z_T)^\gamma$.

We should note that this is a type of reinforcement learning problem where the instant reward for almost all states are zero. The only states which have an instant reward are the terminal ones, that is Z_T. Because of this, we can't set up the same type of portfolio relation as the one used by Halperin. We are instead using the operator expression as an update. In essence this means that the value of each non-terminal state is initially zero. But as we work our way back, using the Equation (2.13),

$$\underset{v\in U}{sup}\{f^v(y) + (L^v\Phi)(y)\} = 0 \quad \text{for all } y \in G, \tag{5.75}$$

we will assign a higher value to states that lead to a larger terminal value of the portfolio.

We can use the HJB equations to provide both the update rule and the terminal conditions. They are, for the portfolio described above, given by

$$\underset{v}{sup}\{\frac{\partial\Phi}{\partial t} + x(\rho + (\mu-\rho)v)\frac{\partial\Phi}{\partial x} + \frac{1}{2}\sigma^2 v^2 x^2 \frac{\partial^2\Phi}{\partial x^2}\} = 0. \tag{5.76}$$

and

$$\Phi(t,x) = x^\gamma. \tag{5.77}$$

The first equation says that the optimal value will occur when we choose the highest value of v that makes the operator zero.

Using the expression for v, found in Example 2.6.5,

$$v = -\frac{(\mu-\rho)\frac{\partial\Phi}{\partial\mathbf{x}}}{x\sigma^2\frac{\partial^2\Phi}{x^2}}. \tag{5.78}$$

and inserting it back into Equation (5.76), gives us that Φ needs to satisfy

$$\frac{\partial\Phi}{\partial t} + \rho x\frac{\partial\Phi}{\partial x} - \frac{(\mu-\rho)^2(\frac{\partial\Phi}{\partial\mathbf{x}})^2}{2\sigma^2\frac{\partial^2\Phi}{x^2}} = 0, \text{ for } t < t_0, x > 0, \tag{5.79}$$

$$\Phi(t,x) = U(x), \text{ for } t = t_0 \text{ or } x = 0, \tag{5.80}$$

In order to make clear the connections between the control parts, and their RL counterparts, we will break it down in a list:

1. $J^u(s,x)$ is the value function, which will correspond to the Q-function $Q(a,s)$. We note that both are a function of both action and state.

2. $\Phi(s, x)$ is the optimal value function, that is our $Q^*(a, s)$. This is the function we are trying to find for each iteration in Halperin's scheme.

3. $\Phi(t_0, x) = U(x)$ is the terminal condition.

In the case of the control example we have an initial wealth, $Z_s = x$.

The setup of the problem

The general idea for our setup is as follows:

1. Simulate the sample data. This is done the same way as for the Chapter 5, except we are now generating paths of Equation (5.67) and Equation (5.68).

2. Create a basis and fill it with the data from the trajectories. This can for example be done as in Section 5.5, by using a polynomial basis of degree 3.

3. Define the final payoff as Z_T^γ. This will also be the final value of the Q-value, that is $Q(Z_T) = Z_T^\gamma$. We will set $u = 0$ for the final step, so that the $Z_T = X_T$.

4. Discretize the optimality conditions. We need to discretize Equation (5.76) in order to formalize the update. We will use central differencing in order to find numerical approximations for the derivatives.

5. Write the numerical approximation in Item 4 so that we have Φ_n on one side, and Φ_{n+1} on the other. This Φ-function represents the Q-function in the QLBS approach, so we now have an expression similar to Equation (5.39) in the exotic option example.

6. Replace u and Φ by their basis functions, and solve for the weights.

However, we will show that this, which can be seen as the most direct translation of the QLBS-model presented in this chapter (by which we mean the formulation of the problem that makes the fewest amount of new assumptions), will lead to some non-negligible problems.

Discretizing the operator

Since we are working with a numerical solution to the HJB-equations, we will need to discretize the operator defined in Equation (5.76). We start by defining the discrete version of the partial derivatives. For this we will use central differencing and combine it with Monte Carlo estimation. For each MC simulation of the wealth Z_{t+1} at time $t + 1$, we will define the partial derivative of sample i as

$$\frac{\partial \Phi^i_{t+1}}{\partial Z^i_{t+1}} \approx \frac{\Phi(Z^{i+1}_{t+1}) - \Phi(Z^{i-1}_{t+1})}{Z^{i+1}_{t+1} - Z^{i-1}_{t+1}} \tag{5.81}$$

$$\frac{\partial^2 \Phi}{\partial Z^2_{t+1}} \approx \frac{\Phi(Z^{i+1}_{t+1}) - 2\Phi(Z^i_{t+1}) + \Phi(Z^{i-1}_{t+1})}{(Z^{i+1}_{t+1} - Z^i_{t+1})(Z^i_{t+1} - Z^{i-1}_{t+1})}, \tag{5.82}$$

for each n. We have written this in terms of $t+1$ since we are going to compute the derivatives using back propagation. For the partial derivative of $t+1$, we will have to use forward difference. We then get that

$$\frac{\partial \Phi^i_{t+1}}{\partial t} \approx \frac{\Phi(Z^i_{t+1}) - \Phi(Z^i_t)}{\Delta t}. \tag{5.83}$$

Remark 5.8.1. We use forward difference in order to get an expression with Q_n, and because we don't have $\Phi(Z_{n+0.5\Delta t})$ or $\Phi(Z_{n-0.5\Delta t})$. In general we will never have the value of $\Phi(Z_{t-\theta})$ for any theta, because of the back propagation.

To simplify the notation, we will denote

$$\Phi^i_{t+1} = \Phi(Z^i_{t+1}), \tag{5.84}$$

$$a(Z^i_{t+1}, v^i_t) = Z^i_{t+1}(\rho + (\mu - \rho)v_i), \tag{5.85}$$

$$b(Z^i_{t+1}, v^i_t) = \frac{1}{2}\sigma^2 v_i^2 (Z^i_{t+1})^2. \tag{5.86}$$

We can then write the optimality condition in Equation (5.76) as

$$\begin{aligned}\frac{\Phi^i_{t+1} - \Phi^i_t}{\Delta t} &+ a(Z^i_{t+1}, v^i_{t+1})\frac{\Phi^{i+1}_{t+1} - \Phi^{i-1}_{t+1}}{Z^{i+1}_{t+1} - Z^{i-1}_{t+1}} \\ &+ b(Z^i_{t+1}, v^i_{t+1})\frac{\Phi^{i+1}_{t+1} - 2\Phi^i_{t+1} + \Phi^{i-1}_{t+1}}{(Z^{i+1}_{t+1} - Z^i_{t+1})(Z^i_{t+1} - Z^{i-1}_{t+1})} = 0.\end{aligned} \tag{5.87}$$

We are thus left with an equation of only $t+1$ dependent terms, and Φ^i_t. If we have the optimal "fraction of wealth" samples v^i_{t+1}, we can therefore solve for Φ^i_t, by using back-iteration. As we will see, however, the setup runs into some problems when we try to find the optimal v-values.

Formulating the HJB update ◇

Having defined discrete expressions for the HJB-optimality condition in Equation (4.6), we can use Equation (5.78), derived from the differentiation in Example 2.6.5, that is

$$v = -\frac{(\mu - \rho)\frac{\partial \Phi}{\partial \mathbf{x}}}{x\sigma^2 \frac{\partial^2 \Phi}{x^2}}. \tag{5.88}$$

By using the expressions from Section 5.8 with MC estimates from samples, and inserting them in Equation (5.78), we get

$$v^*_t = -\frac{(\mu - \rho)\sum_{i=1}^{Nmc}\left(\frac{\Phi^{i+1}_t - \Phi^{i-1}_t}{Z^{i+1}_t - Z^{i-1}_t}\right)\frac{1}{Nmc}}{\sum_{i=1}^{Nmc} Z^i_t \left(\sigma^2 \frac{\Phi^{i+1}_t - 2\Phi^i_t + \Phi^{i-1}_t}{(Z^{i+1}_t - Z^i_t)(Z^i_t - Z^{i-1}_t)}\right)\frac{1}{Nmc}}. \tag{5.89}$$

However, in this equation, v_t is *only* dependent on the value function Φ and portfolio Z at time t. Thus we can't express this in the recursive way, as was done in the method of Halperin [Hal17], and we are not able to find the optimal values.

Since we are unable to find these optimal v_t's, we are not able to find the optimal value function in Equation (5.87) either, and our approach therefore comes to a halt.

Discussion

As we have seen, the method presented by Halperin [Hal17] is flexible in regards to using different option types. However, as this section shows, it's way of formulating the recursive relation is closely related to the way the portfolio optimization is defined. By changing the expression for the optimal hedge, as defined originally in Equation (5.23), we loose the natural link to the Bellman equations (see Section 3.2), and are no longer able to find the analytic solutions.

The basic ideas presented in this section should still be possible to use in a more data driven method, one which perform the Q-learning algorithm without using the analytic expressions in the QLBS model. However, because of time limitations, this proved to be beyond the scope of this thesis.

In the next chapter, we will discuss some concluding remarks, and comment on possible future work and extensions.

CHAPTER 6

Concluding remarks

We have in this thesis studied stochastic optimal control and reinforcement learning. These two fields study the same problem, and are both based on the optimality condition proposed by Bellman (see Section 3.2), but also differs in their use of these equations. We have tried to unify some of the theory underlying the two fields, and also looked at how the theory can be used in applications involving portfolio optimization and option pricing.

In Chapter 4 we discussed the lack of convergence proofs concerning the Q-learner in a continuous action-state space setting. We studied some proofs of similar algorithms and specific cases, and noted which conditions that seem to be present in all the proofs. We also discussed whether additional assumptions made in some of these proofs have connections to the assumptions made in the HJB-equations. The convergence of the Q-learner in continuous action-state space settings is an ongoing field of exploration. In Chapter 4, we tried to address what assumptions might make up the bare minimum of the requirements for convergence.

One possible idea for future work would be to extend on the method from Chapter 5 by using the CVaR as a measure of the agents aversion to risk, as was briefly discussed in Section 5.4. By using the CVaR, we would both change the analytic expression of the optimal hedge (Equation (5.23)), and the future risk minimization in the value function (Equation (5.31)). It would also be interesting to look at true power market data in order to compare the option price produced by the model with real option prices. This is especially interesting to consider when comparing different values of the risk parameter λ, used in the model to quantify an agents aversion to risk. Since we assume that the agents in a market perceives some risk (they would not want to buy and sell options otherwise), this parameter is essential for the part of the model that brings the pricing method into the real, risky world of financial products.

Another idea is to allow for some degree of influence by the hedge on the state values. This was noted in Section 5.2, and would involve using consecutive batch learning. In portfolio theory, one usually assumes that the owner of the portfolio is too small to affect prices in the market by selling or buying stocks. In the power market discussed in Chapter 5, however, the market usually consists of fewer, and larger, power market suppliers, and one might therefore assume that some of these has a certain degree of market power.

In the second part of Chapter 5 we looked at ways to expand on Halperin's approach [Hal17], but by using the optimality demand from the HJB-equations as an update rule. While we did not manage to find the way to do so, we highlighted which prerequisites were needed to extend the model, and showed why the straightforward approach leads to expressions involving multiple unknowns.

APPENDIX A

Python code for numerical experiments.

A.1 Simulating Brownian motions

```
import matplotlib.pyplot as plt
import numpy as np

def bm(T, dt):
        #--------------------------------------------------------------------------
        """
        Description:    Function generating and ploting multiple uncorrelated Brownian
                                motions in addition to their mean value.

        Input:                  T = terminal time of Brownian motion.
                                dt = time step size.

        Output:                 Returns array of Brownian motion, as the
                        cumulated sum of the Brownian increments.
        """
        #--------------------------------------------------------------------------
        n = int(T/dt)
        tt = np.linspace(0, T, n)
        for i in range(1,6):
                dbm = np.random.standard_normal(n)              # Brownian increments.
                bm = np.cumsum(dbm)*np.sqrt(dt)                 #Brownian motion.
                bm[0] = 0
                mean = 0
                mean += np.mean(bm)

                plt.rcParams['figure.figsize'] = (10,8)
                plt.rcParams['lines.linewidth'] = 0.4
                plt.plot(tt, bm, label = 'B(t, omega_%i)' %i, linewidth = 1)
                print(np.mean(bm))
                #plt.plot(np.mean(bm))
                plt.legend()
        plt.plot(tt, np.mean(bm)*tt/tt, label = mean)
        plt.show()
        return(bm)

bm = bm(T, dt)
```

A.2 Simulating geometric Brownian motions with Monte Carlo estimation

```
import matplotlib.pyplot as plt
import numpy as np
import math as math

def gbm(T, dt, mu, sigma, ic, Nmc):
        #---------------------------------------------------------------------------
        """
        Description:    Function generating and ploting multiple uncorrelated geometric
                                Brownian motions in addition to their mean value.

        Input:                  T = terminal time of Brownian motion.
                                dt = time step size.
                                mu = drift of geometric Brownian motion.
                                sigma = volatility of geometric Brownian motion.
                                ic = start value.

        Output:                 returns array of geometric Brownian motion.
        """
        #---------------------------------------------------------------------------
        n = int(T/dt)   # number of time steps.
        tt = np.linspace(0, T, n)       # array of plotting points.
        gbm = np.zeros((n, Nmc))
        for i in range(1,Nmc):
                bm = np.zeros(n)
                dbm = np.random.standard_normal(n)      # Brownian increments.
                bm[0] = 0
                bm = np.cumsum(dbm)*np.sqrt(dt) #Brownian motion.

                gbm[:, i] = ic* np.exp((mu - 0.5*sigma**2)*tt + sigma * bm)     #GBM.
        return(tt, gbm)

#Parameters:
T = 15 # Time.
dt = 0.1 # Time step.
mu = 0.05 # Drift.
sigma = 0.2 # Volatility.
ic = 1. # Initial condition.

tt, gbm = gbm(T, dt, mu, sigma, ic, 100)

#Plots of 6 paths of GBM's:
for i in range(1,6):
        plt.rcParams['figure.figsize'] = (10,8)
        plt.rcParams['lines.linewidth'] = 0.4
        plt.plot(tt, gbm[:, i], label = 'X(t, omega_%i)' %i, linewidth = 1, linestyle="dashed")
        plt.legend()
plt.show()

#Plots of Monte Carlo estimates of GBM mean value:
plt.plot(tt, ic*np.exp(mu*tt), label = 'Analytic computation of mean', linewidth=1.5)
plt.plot(tt, np.mean(gbm[:, 0:10], axis=1),\
label = 'MC estimate of mean, 10 simulations', linewidth=1.5)
plt.plot(tt, np.mean(gbm[:, 0:50], axis=1),\
label = 'MC estimate of mean, 50 simulations', linewidth=1.5)
plt.plot(tt, np.mean(gbm[:, 0:1000], axis=1),\
label = 'MC estimate of mean, 1000 simulations', linewidth=1.5)
plt.legend()
plt.show()
```

A.3 Pricing exotic options

```
import matplotlib.pyplot as plt
import numpy as np
import math as m
from scipy.stats import norm
import pandas as pd
import scipy.interpolate as interpolate
from scipy.linalg import cholesky
import bspline
import bspline.splinelab as splinelab
import time
np.random.seed(15)

t0 = time.time() # Timer for computation time.

sigma = 0.4 # volatility of stock.
mu = 0.0 # Drift of Stock.
S0 = 100 # Initial value of Stock price.
K = S0 # Strike. We use the initial value of the stock.
r = 0.05 # Interest rate.
a = 0.5
etta = 0.6 # Volatility of Wind dynamics.
mu_w = 0.05 # Drift of wind dynamics.

T = 1. # Time of termination.
n= 25 # Number of time steps.
delta_t = T/n                         # time steps.
gamma = np.exp(-r*delta_t)      # discounting factor.
risk_aversion = 0.00001 # Risk aversion.
corr = -0.9 # Correlation of the two Brownian motions.
Nmc = 10000 # Number of simulations.

tt = np.linspace(0, T, n+1)
l = len(tt)
X = np.zeros((Nmc, n+1))
S = np.zeros((Nmc, n+1))

W= np.zeros((Nmc, n+1))
W_= np.zeros((Nmc,n+1))

for j in range(0, Nmc):
        dim_bm = 2
        norm_ = np.random.standard_normal((n+1,dim_bm)) # Two arrays of Brownian increments.
        R = np.array([[1, corr], [corr, 1]]) # corr is correlation.
        corr_chol = cholesky(R, lower=True)     # Make cholesky decomposition.
        norm_ = np.dot(corr_chol, norm_.T) # Create correlated Brownian motions.
        norm_ = norm_.T
        norm_[0, 0] = np.log(S0)
        norm_[0, 1] = 0
        norm_[1:,0] = norm_[0,0] + sigma*np.cumsum(norm_[1:,0])*np.sqrt(delta_t) # Sigma-scaled standard BM.
        norm_[1:,1] = norm_[0,1] + np.cumsum(norm_[1:, 1])*np.sqrt(delta_t) # Standard BM.
        X[j,:] = norm_[:,0]     # Brownian motion driving the asset.
        W_[j,:] = norm_[:,1] # Brownian motion driving the wind.
        S[j, :] = np.exp((mu - 0.5*sigma**2)*tt[:] + X[j,:]) # Stock process.
        W[j, :] = np.exp((mu_w - 0.5*etta**2)*tt[:] + W_[j,:]) # Wind process.

delta_S = S[:, 1:n+1] - m.exp(r*delta_t) * S[:,0:n] # Increments of stock process.
delta_S_hat = delta_S[:,:] - np.mean(delta_S[:,:], axis=0) # Difference to mean of increments of S.

""" We create a simple polynomial basis: """
basis_dim = 3
```

```
def poly_basis(x):
        """
        Input is a vector with dimension 1 times Nmc of state variable X_t.
        Output is matrix of Nmc times basis_dim.
        """
        pol = np.stack((np.ones(Nmc), x, np.square(x)))
        pol = np.polynomial.polynomial.polyvander(x, basis_dim-1).T
        return(pol.T)


""" We create a matrix of of all our datapoints by using the basis: """
A = np.zeros((n, basis_dim, basis_dim))
M_approx = np.zeros((n+1, Nmc, basis_dim))

for i in range(0, n+1):
        M_approx[i, :, :] = poly_basis(X[:,i])

""" We define the two functions from equation (52) in Halperin: """

def A_func(t, delta_S_hat, Nmc, M_approx):
        """ A is an n x basis_dim x basis_dim dimensional matrix,
        based on equation (52) in Halperin. """
        delta_S_hat2 = np.square(delta_S_hat[:,t])
        Y = np.dot(M_approx[t, :, :].T, M_approx[t, :, :] * delta_S_hat2.reshape(-1,1))
        return(Y)

def B_func(t, delta_S_hat, Nmc, M_approx, PI_hat_t):
# note that PI_t should be a vector of PI_t for each k.
        """ B is an  n x basis_dim dimensional matrix, , based on equation (52) in Halperin."""
        Y = np.dot(M_approx[t, :, :].T, PI_hat_t * delta_S_hat[:,t])\
    # + 1./(2*gamma * risk_aversion)*delta_S[:,t].T) #Risk part is set to zero.
        return(Y)

"""
We define our terminal payoff-function by using the expression for the option.
Here we allow for both the payoff-function used by Halperin,
and the payoff-function defining a put-call on energy futures.
  """
def term_payoff(S, W, K, L=1, put = "false"):
        if put == "true":
                return(np.maximum(K - S, 0))
        else:
                return(np.maximum(K-S, 0)*np.maximum(W-L, 0))

"""
We then compute values for the action a and the portfolio PI iteratively,
by using the recursivenes of PI.
"""
PI = np.zeros((Nmc, n+1))
phi = np.zeros((basis_dim, n+1)) # vector with weights for hedge.
a = np.zeros((Nmc, n+1)) # a is the hedge.
PI_hat = np.zeros((Nmc,n+1))

PI_hat_func = lambda x: x - np.mean(x, axis=0)
PI[:, n] = term_payoff(S[:, -1], W[:, -1], K)
phi[:,n] = 0
PI_hat[:, n] = PI_hat_func(term_payoff(S[:,-1], W[:, -1], K)) # terminal value of portfolio.

for t in range(n-1, -1, -1):
        A_ = A_func(t, delta_S_hat, Nmc, M_approx)
        B_ = B_func(t, delta_S_hat, Nmc, M_approx, PI_hat[:,t+1])
        phi[:,t] = np.dot(np.linalg.inv(A_), B_)
        a[:, t] = np.dot(M_approx[t,:,:], phi[:,t])
        PI[:,t] = gamma * (PI[:, t+1] - a[:, t] * delta_S[:, t])
        PI_hat[:,t] = PI_hat_func(PI[:,t])

""" Using the computed hedge values, we can now compute the reward R: """
R = np.zeros((Nmc, n+1))
R[:, n] = -risk_aversion * np.var(PI[:, -1])

def R_func(t, a):
        R[:, t] = gamma * a[:, t] * delta_S[:, t] - risk_aversion * gamma**2 *np.mean(PI_hat[:, t+1]**2 \
         - 2*a[:, t]*delta_S_hat[:, t]*PI_hat[:,t+1] +
        a[:, t]**2 * (delta_S_hat[:,t])**2)
        return(R[:, t])

""" We define the two functions from in (56) in Halperin.  """
def C_func(t, M_approx):
        Y = (M_approx[t, :, :].T).dot(M_approx[t, :, :])
        return(Y)

def D_func(t, M_approx, R, Q):
        Z = np.dot(M_approx[t, :, :].T, (R + gamma * Q))
        return Z

"""
We can then find the Q value, which is the optimal price of the option,
by working our way back from t=T, using the regression suggested by
Halperin. We find the weights for the basis expansion, and use these to
find Q:
"""
Q = np.zeros((Nmc, n+1))
Q[:,n] = -PI[:, n] - risk_aversion * np.var(PI[:, n]) # Start value for the Q-function.
def Q_func():
        for t in range(n-1, -1, -1):
                R[:, t] = R_func(t, a)
                C = C_func(t, M_approx)
                D = D_func(t, M_approx, R[:, t] , Q[:,t+1])
                omega = np.linalg.inv(C).dot(D)
                Q[:, t] = np.dot(M_approx[t, :, :], omega.T)
Q_func()

""" Functions for computing the Black-Scholes price:"""
def BS_put(t, S0, sigma, r, K, T):
        d1 = 1./(sigma* np.sqrt(T-t)) * (np.log(S0/K) + (r + 0.5*sigma**2)*(T-t))
        d2 = d1 - sigma*np.sqrt(T-t)
        price = norm.cdf(-d2)*K*np.exp(-r*(T-t)) - norm.cdf(-d1)*S0
        return price

def BS_call(t, S0, sigma, r, K, T):
        d1 = 1./(sigma* np.sqrt(T-t)) * (np.log(S0/K) + (r + 0.5*sigma**2)*(T-t))
        d2 = d1 - sigma*np.sqrt(T-t)
        price = norm.cdf(d1)*S0 - norm.cdf(d2)*K*np.exp(-r*(T-t))
        return price

print("Put option price is: %s" % (BS_put(0, S0, sigma, r, K, T)))
print('Halperin estimate of option price.: %s' % (-Q[0,0]))

""" For comparison we compute the expected value of the return by using the MC simulations: """
total_nmc_payoff = np.sum(term_payoff(S[:,-1], W[:, -1], K))
expected_payoff = np.exp(-r) * total_nmc_payoff/Nmc
print('Expected payoff by Monte Carlo simulation: %s' % expected_payoff)
print('Difference between QLBS and MC estimate: %s' % (-Q[0, 0] - expected_payoff))
print('Difference between QLBS and true estimate: %s' % (BS_put(0,S0,sigma,r,K,T) - (-Q[0,0])))
print('Difference between MC and true estimate: %s'%(BS_put(0,S0,sigma,r,K,T)-expected_payoff))

t1 = time.time()

print('Total computation time: %s ' % (t1-t0))

""" Plots """
# Plot of the 10 first simulations of X:
fig, axs = plt.subplots(2, 2, constrained_layout=False)
axs[1,0].plot(S[0:10, :].T)
axs[1,0].set_title('20 simulations of stochastic process S.')
axs[0,0].plot(X[0:10, :].T)
axs[0,0].set_title('10 simulations of stochastic process X.')
axs[1,1].plot(W[0:10, :].T)
axs[1,1].set_title('10 simulations of stochastic process W.')
axs[0,1].plot(W_[0:10, :].T)
axs[0,1].set_title('10 simulations of stochastic process W_.')
plt.show()

print(np.corrcoef(X[10,:], W_[10,:]))# Check correlation.
plt.plot(X[5,:], W[5,:], "o")
plt.title('Correlation, X and W.')
plt.show()

fig, axs = plt.subplots(2, constrained_layout=False)
axs[0].plot(a[0:10, :].T)
axs[0].set_title('10 simulations of optimal a values.')
axs[1].plot(PI[0:10, :].T)
axs[1].set_title('10 simulations of optimal payoff values.')
plt.show()

plt.plot(R[0:10,:].T)
plt.title('10 simulations of optimal R values.')
plt.show()
plt.plot(Q[0:11, :].T)
plt.title('10 paths of Q value.')
plt.show()
```

List of notation and symbols

Mathematics

$(\Omega, \mathbb{F})$	A The measurable space.
$(\Omega, \mathcal{F}, P)$	The probability space.
$(Af)(x)$	The generator A of X_t.
$\hat{T}$	The first exit time after s for $\{X_r^{s,x}\}_{r \geq s}$.
$\mathcal{F}$	The σ-algebra on Ω.
$\mathcal{H}_{\mathcal{U}}$	The σ-algebra generated by $\mathcal{U}$.
Ω	The set of possible states.
$\Phi(y)$	The maximum atainable value of the performance function.
τ	The stopping time.
$\{B_t\}_{t \geq 0}$	The Brownian motion.
$\{X_h^{s,x}\}_{h \geq s}$	The process starting in x, evaluated on $[s, h]$.
$\{X_t\}_{t \in T}$	The stochastic process.
$C^n([0, \infty) \times \mathbb{R})$	The space o n times continuously differentiable functions on $[0, \infty) \times \mathbb{R}$ with continuous extensions of the partial derivative on $(0, \infty) \times \mathbb{R}$.
$J^u(y)$	The performance function.
P	The probability measure.
$Q^{s,x}$	The probability law of X_t.
Y_t	The time shifted state-control pair.

Machine Learning

$\alpha \in [0, 1]$	The learning rate.
$\gamma \in [0, 1]$	The discount factor.

$\mathcal{A}$	The action space.
$\mathcal{S}$	The state space.
Ω	The environment.
π	The policy mapping states to actions.
$a_t(S_t, R_t) \in \mathcal{A}$	The action taken at time t.
b	The behaviour policy which drives the learning algorithm in off-policy learning.
$G_{t+1}(\pi, S_t)$	The discounted future reward.
$p(s, s_{t+1}, a)$	The transitional probability of going from state s_t to s_{t+1} when performing action a.
$Q_\pi(S_t, a_t)$	The action-value function, see Section 3.3
$R \in \mathcal{R} \subset \mathbb{R}$	The reward space.
$R_{t+1}(S_t, a_t, \omega) \in \mathcal{R}$	The reward received at time $t+1$, for the state at time t.
$S_t(S_{t-1}, a_{t-1}, \omega)$	The state of the environment at time t.
$V_\pi(S_t, a_t)$	The value function.

Bibliography

[BBK08] Benth, F. E., Benth, J. Š., and Koekebakker, S. *Stochastic Modeling of Electricity and Related Markets.* WORLD SCIENTIFIC, 2008. eprint: https://www.worldscientific.com/doi/pdf/10.1142/6811.

[Bel53] Bellman, R. *An introduction to the theory of dynamic programming.* Tech. rep. RAND CORP SANTA MONICA CA, 1953.

[Ben03] Benth, F. E. *Option theory with stochastic analysis: an introduction to mathematical finance.* Springer Science & Business Media, 2003.

[Gen03] Gentle, J. E. *Random number generation and Monte Carlo methods.* eng. New York, 2003.

[Hal17] Halperin, I. "QLBS: Q-learner in the Black-Scholes (-Merton) worlds". In: *arXiv preprint arXiv:1712.04609* (2017).

[KLM96] Kaelbling, L. P., Littman, M. L., and Moore, A. W. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[KOT11] Konidaris, G., Osentoski, S., and Thomas, P. "Value function approximation in reinforcement learning using the Fourier basis". In: *Twenty-fifth AAAI conference on artificial intelligence.* 2011.

[LGR12] Lange, S., Gabel, T., and Riedmiller, M. "Batch Reinforcement Learning". In: *Reinforcement Learning: State-of-the-Art.* Ed. by Wiering, M. and Otterlo, M. van. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 45–73.

[Man] Mansuy, R. "The origins of the word "martingale"". In: ().

[MMR08] Melo, F. S., Meyn, S. P., and Ribeiro, M. I. "An analysis of reinforcement learning with function approximation". In: *Proceedings of the 25th international conference on Machine learning.* 2008, pp. 664–671.

[Mni+15] Mnih, V. et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[Øks03] Øksendal, B. *Stochastic differential equations.* Sixth. Universitext. An introduction with applications. Springer-Verlag, Berlin, 2003, pp. xxiv+360.

[PBS01] Potters, M., Bouchaud, J.-P., and Sestovic, D. “Hedged Monte-Carlo: low variance derivative pricing with objective probabilities”. In: *Physica A: Statistical Mechanics and its Applications* 289.3-4 (2001), pp. 517–525.

[RUZ02] Rockafellar, R. T., Uryasev, S. P., and Zabarankin, M. “Deviation measures in risk analysis and optimization”. In: *University of Florida, Department of Industrial & Systems Engineering Working Paper* 2002-7 (2002).

[SB18] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning series. MIT Press, 2018.

[Sil+18] Silver, D. et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[TV97] Tsitsiklis, J. N. and Van Roy, B. “Analysis of temporal-diffference learning with function approximation”. In: *Advances in neural information processing systems.* 1997, pp. 1075–1081.

[WD92] Watkins, C. J. and Dayan, P. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.