Controlling the drift of a Brownian motion with statistical methods and reinforcement-learning

Masterarbeit

im Masterstudiengang Mathematik (1-Fach)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Christian-Albrechts-Universität zu Kiel
vorgelegt von

Philip Le Borne

Erstprüfer: Prof. Dr. Sören Christensen
Zweitprüfer: Prof. Dr. Jan Kallsen

Kiel, 8. März 2024

Contents

Zusammenfassung

In dieser Arbeit präsentieren wir vier Methoden, um den Drift einer Brownschen Bewegung zu kontrollieren, und zwar zwei basierend auf statistischen Methoden und zwei basierend auf Reinforcement Learning. Nach einer Einleitung in die Problemstellung in Kapitel 1 führen wir in den Kapiteln 2 und 3 einige Definitionen und Resultate über stochastische Differenzialgleichungen und über die Erneuerungstheorie ein. Diese benötigen wir, um die zu Grunde liegende Fragestellung zu formulieren. In Kapitel 4 wird das Szenario betrachtet, in dem wir alle Informationen über den stochastischen Prozess kennen, basierend auf der Publikation *Some Solvable Stochastic Control Problems* von Beneŝ, Shepp and Wittenhousen [BSW80].

Nachdem wir eine explizite Lösung zum stochastischen Kontrollproblem gegeben haben, beschäftigen wir uns im weiteren Verlauf der Arbeit mit dem Fall, dass nicht alle Informationen über den stochastischen Prozess gegeben sind, insbesondere falls die Werte vom Drift der Brownschen Bewegung nicht bekannt sind. Dies basiert auf dem nicht veröffentlichten Entwurf und Überlegungen von Prof. Dr. Sören Christensen (Universität Kiel), Prof. Dr. Stefan Ankirchner und Stefan Perko (beide Universität Jena). Die explizite Lösung gibt die Struktur der optimalen Kontrollfunktion an. Mit diesem Wissen werden dann in den Kapiteln 6 und 7 statistische Methoden verwendet, um die optimale Kontrollfunktion mit Hilfe von Schätzern zu approximieren. Dies führt zu Algorithmen, um den Drift einer Brownschen Bewegung zu kontrollieren. Wir präsentieren zwei solcher Algorithmen in den Kapiteln 6 und 7. Während für den ersten Algorithmus auch theoretische Ergebnisse gezeigt werden, führt der zweite Algorithmus in den numerischen Tests zu einer Verbesserung, dessen theoretische Analyse liegt jedoch noch nicht vor.

Anschließend präsentieren wir einen weiteren Ansatz, der auf Reinforcement Learning basiert. Dazu geben wir eine kurze Einführung in das Q-Learning und das Deep-Q-Learning. Die erste Methode beruht auf dem Reinforcement Learning der 80er Jahre, in dem der kontinuierliche Zustandsraum zuerst diskretisiert werden muss, bevor der klassische Q-Learning Algorithmus verwendet werden kann. Der modernere Ansatz, Deep-Q-Learning, wird anschließend präsentiert.

Für alle vier Methoden zeigen und diskutieren wir numerische Resultate.

1 Introduction

Imagine we have a trajectory, some moving object and something controlling the direction the object is moving in. The goal is for the moving object to stay, if possible, exactly on its given trajectory. This goal is easy to achieve if there is no need to control the direction of the moving object. Take for example a train moving with a constant speed on some railroad tracks (the trajectory). Since we can assume that the train will not deviate from the direction that the tracks set, there is no need for controlling the direction. If we consider a car, for example, driving on a road, then we would set the trajectory as the center of the car's lane. Now, contrary to the train, the car is not fixed to the trajectory and can freely move left or right of it. The control would then be to use the steering wheel to readjust the car's position back to the center of the lane. The car example can be simplified and formulated as a stochastic control problem where (very broadly speaking) the system's (car's) position X_t with respect to the time $t \in [0, \infty)$ is given as the solution of a stochastic differential equation with given initial value

$$dX_t = b(t, X_t)dt + dW_t, \qquad X_0 = x_0. \tag{1.1}$$

The random fluctuations of the the car's movement are modeled by a standard Brownian motion (Wiener process) W_t. These fluctuations can be caused by wind, road conditions, the car itself, or other attributes. The steering is modeled by the drift $b : [0, \infty) \times \mathbb{R} \to \mathbb{R}$ of the stochastic differential equation. Here, positive of negative drift would correspond to steering the car left or right. The trajectory is then the origin and the goal is to keep the system's position X_t as close to the origin as possible at all times ($t \in [0, \infty)$). There are many more things to consider in the car example. However the goal of this thesis is not to model specific examples but to discuss different methods for solving the general control problem given by the stochastic differential equation (1.1) above.

Other examples for stochastic control problems are supply/demand or controlling the direction of a satellite. For many applications, the simple model of the system's position (1.1) is not adequate and has to be refined. These examples are, however, useful in order to understand the theory and algorithms presented in the later sections.

These types of stochastic control problems have been extensively studied in the literature, for example in [Hen+19], [LP20] and [BSW80].

In this thesis, the main question is how to find a control function b that optimally controls the system given by (1.1), i.e., keep the system as close to the origin as possible. This type of control problem was considered by Beneŝ, Shepp and Wittenhousen in their paper *Some Solvable Stochastic Control Problems* [BSW80]. In this paper, they found explicit solution to three types of stochastic control problems, one of which provided the ground work for this thesis and is presented in an adapted form in Section 4. Finding an explicit solution of the initial control problem as presented in Section 4 relies on knowing certain properties about the drift function $b(\cdot, \cdot)$ in (1.1). This was then the motivation for Prof. Dr. Sören Christensen (University of Kiel), Prof. Dr. Stefan Ankirchner and Stefan Perko (both University of Jena) to see how control can still be achieved if we take away the knowledge of the drift function $b(t, x)$ in (1.1) that was needed to find an explicit solution.

The remainder of this thesis is structured into three parts. The first part consists of Sections 2 and 3 which briefly presents some theory that is needed later. In Section 2, we introduce some results for stochastic differential equations which were taken and adapted from the books [Øks98], [KS91], [BS96] and [Kal02]. In Section 3, we give a short introduction to renewal theory

in order to prove a key result needed to analyze the regret of statistical algorithms presented later, i.e., to analyze how good the algorithms for controlling the drift are. This section is based on the preprint book [Als10] and the key result is adapted from the paper [Lin83]. We corrected a few index mistakes in the proof of the result form [Lin83] and give a few more details.

The next two parts are the main focus of this thesis. In the second part of the thesis, Sections 4, 5, 6 and 7, we focus on using statistical methods to solve the stochastic control problem sketched above in (1.1). The underlying idea of both methods is to use the trajectories of the process as estimates for the drift. The idea of using the trajectories can also be found in [LP20], however, there is not much other literature regarding this stochastic control problem. These sections are based on the unpublished draft of Prof. Dr. Sören Christensen (University of Kiel), Prof. Dr. Stefan Ankirchner and Stefan Perko (both University of Jena). In Section 4 we present the initial control problem where we have all knowledge about the system (1.1) at hand and present the explicit solution that was found by Beneŝ, Shepp and Wittenhousen in [BSW80]. Then in Section 5 we give some properties of Brownian motions with drift in order to analyze the regret of the algorithms in the next two sections. In Section 6, we present and analyze an algorithm for estimating the drift of a Brownian motion, hence solving the control problem with an approximate solution. In the next Section 7, we present the idea for an improved algorithm. The main idea for the algorithm was taken from the unpublished draft. These, however, had to be heavily modified. This improved algorithm is presented in a working state, however, we had to put in some conditions and restrictions in order to ensure the approximation to converge. While working further on this algorithm, we reached a point where numeric results showed that it is reasonable to assume that this algorithm offers an improvement compared to the one presented in the previous Section 6. However, the proof for this remains an open question. At the end of each respective subsection for the two algorithms, we present their implementation and discuss their results.

The third part of this thesis, Section 8, is dedicated to using reinforcement learning methods for solving the stochastic control problem. We first give a short introduction to Q-Learning and show how Q-learning can be used to control the drift of a stochastic differential equation. We then give a brief introduction to Deep-Q-Learning and present and discuss the results of this method. The implementation of the Deep-Q-Learning method relies strongly on examples given in the book [Lap18] whereas the implementation of classic Q-Learning algorithm is my own contribution.

2 Stochastic differential equations

In the following we will assume $(\Omega, \mathcal{F}, P)$ to be a probability space and all random variables to be defined on this space. We base the definitions and results on the books [SP12], [Øks98], [KS91], [Kal21].

Let $(\mathcal{F})_{t\geq 0}$ be a filtration[1] on $(\Omega, \mathcal{F}, P)$. Recall that a *martingale* is an adapted stochastic process $X = (X_t, \mathcal{F}_t)_{t\geq 0}$ satisfying $E[|X_t|] < \infty$ and

$$E[X_t|\mathcal{F}_s] = X_s \tag{2.1}$$

for all $t > s$. If we have '$\geq$' or '$\leq$' instead of '$=$' in (2.1) we call X a *sub-* or *supermartingale*.

Definition 2.1. Let $(W_t)_{t\geq 0}$ be a d-dimensional Brownian motion. A filtration $(\mathcal{F}_t)_{t\geq 0}$ is called *admissible*, if

1. $\mathcal{F}_t^W \subset \mathcal{F}_t$ for all $t \geq 0$ and
2. $W_t - W_s$ is independent of $\mathcal{F}_s$ (independent increments).

If $\mathcal{F}_0$ contains all subsets of P null sets, $(\mathcal{F}_t)_{t\geq 0}$ is called a *complete admissible* filtration.

Note that the filtration $(\mathcal{F}_t^W)_{t\geq 0}$ generated by W is always an admissible filtration.

Many of the results presented in this section also hold for higher dimensions, we however will only consider the one-dimensional case.

A *stochastic differential equation* (SDE) is an equation of the form

$$dX_t = b(t, X_t)dt + \sigma(t, X_t)dW_t, \qquad X_0 = x_0, \tag{2.2}$$

where

- (W_t) is a d-dimensional Brownian motion on a filtered probability space $\big(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, P\big)$,
- x_0 is $\mathcal{F}_0$-measurable,
- $b : [0,T] \times \mathbb{R}^n \to \mathbb{R}^n$ and $\sigma : [0,T] \times \mathbb{R}^n \to \mathbb{R}^{n\times d}$ have some regularity properties specified case by case,
- $b(t, X_t)$ is predictable and
- the solution (X_t) is a d-dimensional continuous adapted process.

We distinguish between weak and strong solutions for equations of the form (2.2).

Definition 2.2. Let $(W_t, \mathcal{F}_t)_{t\geq 0}$ be a Brownian motion with an admissible filtration $(\mathcal{F}_t)_{t\geq 0}$ and let $b : [0,T] \times \mathbb{R}^n \to \mathbb{R}^n$ and $\sigma : [0,T] \times \mathbb{R}^n \to \mathbb{R}^{n\times d}$ be measurable functions. A progressively measurable process $(X_t, \mathcal{F}_t)$ with initial condition $X_0 = x_0$ and $X_t \in \mathbb{R}^n$ is called a *strong solution* of an SDE (2.2) with initial condition $X_0 = x_0$ if

$$X_t - X_0 = \int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s$$

holds almost surely for all $t \geq 0$.

[1] an increasing family of sub-σ-algebras of $\mathcal{F}$.

A solution X is called a strong solution because the version B_t of a Brownian motion is given in advance and the solution X constructed from it is $\mathcal{F}_t^Z$-adapted[2].

In contrast to the former definition of a strong solution, we speak of a *weak solution* if we are only given the initial distribution of X_0 and are looking for a probability space $(\Omega, \mathcal{F}, P)$ along with a Brownian motion $(W_t, \mathcal{F}_t)_{t\geq 0}$ and adapted process X that solves (2.2). Formally we define this as follows.

Definition 2.3. A *weak solution* to the SDE (2.2) consists of some Brownian motion $(W_t, \mathcal{F}_t^W)_{t\geq 0}$ and a progressively measurable process $(X_t, \mathcal{F}_t^W)_{t\geq 0}$ such that $(\mathcal{F}_t^W)$ is a complete admissible filtration and

$$X_t - X_0 = \int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dB_s, \qquad X_0 = x_0$$

holds almost surely for all $t \geq 0$.

Definition 2.4. We say that there is *uniqueness in law* for equations of the form (2.2), if given two weak solutions on any pair of spaces, their laws coincide.

For the setting of a weak solution, we are only given the functions $b(t, x)$ and $\sigma(t, x)$ and ask for a process $\big((X_t, B_t), \mathcal{H}_t\big)$ on a probability space $(\Omega, \mathcal{H}, P)$ such that equation (2.2) holds a.s..

A strong solution is of course also a weak solution, however the converse does not hold in general. An example for this is the so called Tanaka's equation

$$dX_t = sign(X_t)dB_t, \qquad X_0 = 0,$$

for which there exists a weak but no strong solution. We refer to Example 5.3.2 in [Øks98] for details.

Later we will be considering one-dimensional, time-homogeneous stochastic differential equation of the form

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t, \tag{2.3}$$

where we have constant noise $\sigma \equiv 1$ and bounded drift $b : \mathbb{R} \to \mathbb{R}$. For these equations there exist results on the existence of a strong solution. The following heorem is Proposition 5.17 in Chapter 5 of [KS91].

Proposition 2.5. *(Existence of strong solutions) Suppose the drift $b : \mathbb{R} \to \mathbb{R}$ is bounded and σ is Lipschitz-continuous with σ^2 bounded away from zero on every compact subset of $\mathbb{R}$. Then for every initial condition independent of the driving Brownian motion $W = (W_t, \mathcal{F}_t)_{t\in[0,\infty)}$, the one-dimensional, time-homogeneous stochastic differential equation (2.3) has a (non-exploding) unique strong solution.*

Remark 2.6. The statement of the previous Proposition 2.5 also holds for d-dimensional Brownian motions. For this we refer to 5.5.17 in [KS91]

The next theorem, the Girsanov theorem, is a fundamental theorem in the general stochastic analysis. Basically the theorem states, that if one changes the drift component of an Itô process, then the corresponding laws of the processes will not differ dramatically. This version is taken form the book [Øks98]

[2]Here $\mathcal{F}_t^Z$ denotes the filtration generated by Z.

Theorem 2.7. *(Girsanov) Let $(Y_t)_{t\in[0,\infty)}$ be an Itô process in $\mathbb{R}$ of the form*

$$dY_t = b(t, Y_t)dt + dW_t, \qquad t \leq T,\ Y_0 = 0,$$

where $T \in [0,\infty]$ is a given constant and W_t is a Brownian motion. Let

$$M_t := \exp\left\{\int_0^t b(s, Y_s)dX_s - \frac{1}{2}\int_0^t b(s, Y_s)^2 ds\right\}, \quad t \leq T$$

and assume that b satisfies Novikov's condition

$$E\left[\exp\left(\frac{1}{2}\int_0^T b^2(s, Y_s)ds\right)\right] < \infty$$

where $E = E_P$ is the expectation w.r.t. P. Define the measure Q on $(\Omega, \mathcal{F}_T)$ by

$$dQ = M_T dP. \tag{2.4}$$

Then Y_t is a Brownian motion w.r.t. Q for $t \leq T$.

Remark 2.8. Novikov's condition guarantees that M is a martingale w.r.t. the filtration $(\mathcal{F}_t)_{t\geq 0}$ and measure P for $t \leq T$ (c.f. [Øks98]). Since M_t is a martingale we have

$$M_T dP = M_t dP$$

on $\mathcal{F}_t$ for $t \leq T$. This is the case since for a bounded, $\mathcal{F}_t$ measurable function f we have

$$\begin{aligned}\int_\Omega f(\omega)M_T(\omega)dP(\omega) &= E[fM_T] = E\big[E[fM_T|\mathcal{F}_t]\big] \\ &= E\big[fE[M_T|\mathcal{F}_t]\big] = E[fM_t] = \int_\Omega f(\omega)M_t(\omega)dP(\omega)\end{aligned}$$

Remark 2.9. An equivalent way of expressing (2.4) is to say that Q is absolutely continuous w.r.t. P, i.e. $Q \ll P$ with Radon-Nikodym derivative

$$\frac{dQ}{dP} = M_T$$

on $\mathcal{F}_T$. This is the case since Theorem 2.7 states that for all Borel sets $B_1, \ldots, B_k \in \mathbb{R}$ and $t_1, \ldots, t_k \leq T$, $k \in \mathbb{N}$ we have

$$Q[Y_{t_1} \in B_1, \ldots, Y_{t_k} \in B_k] = P[W_{t_1} \in B_1, \ldots, W_{t_k} \in B_k].$$

Since $M_T > 0$ almost surely, we also have $P \ll Q$, hence the two measures Q and P are equivalent.

The following proposition is adapted from Proposition 5.3.6 of [KS91].

Proposition 2.10. *Let $T < \infty$ be fixed and consider the stochastic differential equation with initial distribution $X_0 \sim \delta_x$[3] and*

$$dX_t = b(t, X_t)dt + dW_t, \quad 0 \leq t \leq T, \tag{2.5}$$

where $b : [0,\infty) \times \mathbb{R} \to \mathbb{R}$ is a Borel-measurable bounded function, W_t is a Brownian motion and $x \in \mathbb{R}$ is given. Then there exists a weak solution to (2.5)

[3]Here δ_x denotes the Dirac-measure.

Proof. The idea is to construct a solution using Girsanov's Theorem 2.7. Let $x \in \mathbb{R}$ and let $(X_t)_{t\geq 0}$ be a Brownian motion on some filtered probability space $\big(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, P\big)$. Since b is bounded, Novikov's condition is satisfied, hence the process

$$M_t := \exp\left\{\int_0^t b(s, X_s)dX_s - \frac{1}{2}\int_0^t b(s, X_s)^2 ds\right\}$$

is a continuous martingale. Now let

$$dQ = M_T dP$$

on $\mathcal{F}_T$. Then Girsanov's Theorem 2.7 implies that under Q the process

$$W_t = X_t - x - \int_0^t b(s, X_s)ds$$

is a Brownian motion. Rearranging terms yields

$$X_t = x + \int_0^t b(s, X_s)ds + W_t$$

which shows that $\big(\Omega, \mathcal{F}_t, (\mathcal{F}_t)_{t\geq 0}, Q, W, X\big)$ is a weak solution of the SDE. □

Remark 2.11. Proposition 2.10 also holds for $0 \leq t < \infty$, we refer the reader to Proposition 5.3.6 in [KS91].

Definition 2.12. Let $(X_t)_{t\geq 0}$ be a (time-homogeneous) Itô diffusion in $\mathbb{R}$. The *(infinitesimal) generator* A of X_t is defined by

$$Af(x) = \lim_{t\downarrow 0} \frac{E^x[f(X_t)] - f(x)}{t}$$

for $x \in \mathbb{R}$ and $f \in C^0(\mathbb{R})$. The set of function $f : \mathbb{R} \to \mathbb{R}$ such that the limit exists at x is denoted by $\mathcal{D}_A(x)$, and $\mathcal{D}_A$ denotes the set of functions for which the limit exists for all $x \in \mathbb{R}$.

The intuition of generator is that it describes the movement of the process on the infinitesimal time interval. If we consider a one-dimensional diffusion process given by the SDE

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t, \qquad X_0 = x, \tag{2.6}$$

where W_t is a standard Brownian motion, b is assumed to be bounded and σ is assumed to be Lipschitz-continuous with σ^2 bounded away from zero on every compact subset of $\mathbb{R}$, such that by Proposition 2.5 there exists a strong solution. Now for $f \in C^2(\mathbb{R})$, an application of Itô's formula yields that the generator of a solution X_t of the SDE is given by

$$Af(x) = b(x)f'(x) + \frac{\sigma^2(x)}{2}f''(x). \tag{2.7}$$

Indeed, if we apply Itô's formula to $f(X_t)$ we have

$$f(X_t) = f(x) + \int_0^t \left(b(X_t)f'(X_t) + \frac{1}{2}\sigma^2(X_t)f''(X_t)\right)dt + \int_0^t f(X_t)\sigma(X_t)dB_t. \tag{2.8}$$

Now using the fact that the stochastic integral is a local martingale, the fundamental theorem of calculus and by taking the limits yields the desired form (2.7). If we wish to remove the drift in (2.8), we are seeking a function $s \in \mathcal{D}_A(x)$ satisfying $As \equiv 0$. This yields the following differential equation

$$bs' + \frac{\sigma}{2}s'' = 0.$$

By substituting $y = s'$ and solving the first-order homogeneous linear differential equation through standard methods, we get the solution

$$s(x) = \int_c^x \exp\left(-2\int_c^y \frac{b(\zeta)}{\sigma^2(\zeta)}d\zeta\right)dx$$

for any constant $c \in \mathbb{R}$. We call the function s *scale function* associated to X_t. We say that X_t is on *natural scale* if $s(x) = x$. Further we define the *speed measure* m as the measure on $\mathbb{R}$ with the density

$$x \mapsto \frac{2}{\sigma^2(x)s'(x)}.$$

The speed measure measures the speed at which the diffusion moves in the space. One can show that there exists an invariant probability measure if and only if the speed measure is finite. Then the invariant measure is unique and given by

$$\pi(A) = m(A)/m(\mathbb{R}) \tag{2.9}$$

(c.f. Lemma 33.19 in [Kal02] and [BS96]).

3 Introduction to renewal theory

The theorems in this chapter are based off on the book *Renewal, Recurrence and Regeneration* by Gerold Alsmeyer [Als10]. The goal of this section is to show that the moments of *coupling time* $\tau_c = \inf\{t \geq 0 : X_t = X'_t\}$ of two independent diffusion processes X and X' exist. This is one of the main results of the paper of T. Lindvall [Lin83] and will become of importance later when we analyze the regret of our algorithms.

Definition 3.1. Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of i.i.d. positive random variables with (positive) mean μ. The increasing sequence $(S_n)_{n\in\mathbb{N}_0}$ defined by $S_0 = X_0$ and

$$S_n = S_0 + \sum_{i=1}^{n} X_i$$

is called *renewal process*. We call the X_i *inter-arrival* times and the S_n *arrival epoch*. If $S_0 = 0$, we say that the renewal process is an *ordinary* renewal process and for $S_0 > 0$, we say it is a *delayed* renewal process[4].

Example 3.2. Suppose we are given an infinite supply of light bulbs and their lifetimes are given by the random variables $X_0, X_1, X_2, \ldots$, assumed to be i.i.d. with finite mean. Then the process

$$S_n = \sum_{i=1}^{n} X_i$$

denotes the time at which the n-th light bulb fails and needs to be replaced.

Some key questions in renewal theory are the following:

1. Is the number of renewals $N(t) = \sup\{n : S_n \leq t\}$ a.s. finite? What can be said about the expected number of renewals $EN(t)$?
2. Does $\lim_{t\to\infty} N(t)/t$ exist? And what can be said about its expectation?

In this section we will answer these questions.

Definition 3.3. The stochastic process $(N(t))_{t\geq 0}$ defined as

$$N(t) = \sup\left\{n \in \mathbb{N}_0 : S_n \leq t\right\}$$

is called the *renewal counting process* associated with $(S_n)_{n\in\mathbb{N}_0}$.

An equivalent definition is given by

$$N(t) = \sum_{n\geq 1} 1_{[0,t]}(S_n).$$

This form extends to measurable sets A of $\mathbb{R}_{\geq 0}$ by

$$N(A) := \sum_{n\geq 1} 1_A(S_n) = \sum_{n\geq 1} \delta_{S_n}(A)$$

[4]Note that X_0 and X_1 do not have to have the same distribution.

where δ_{S_n} denotes the Dirac measure at S_n. The intensity measure

$$U(A) := EN(A) = \sum_{n\geq 1} P(S_n \in A)$$

for $A \in \mathcal{B}(\mathbb{R}_{\geq 0})$ is called the *renewal measure* of S_n which measures the expected number of renewals in a set. Its distribution function

$$[0,\infty) \ni t \mapsto U(t) := U([0,t]) = \sum_{n\geq 1} P(S_n \leq t)$$

is called *renewal function* of $(S_n)_{n\geq 0}$.

The name "renewal process" comes from the property of the process starting over at each arrival epoch S_n. If, say the n-th arrival occurs at $S_n = T$, then the m-th arrival starting from $S_n = T$ is at $S_{n+m} - S_n$. This means that, if $S_n = T$ is given, then

$$\big\{N(T+t) - N(T) : t \geq 0\big\}$$

is a renewal counting process with i.i.d inter-arrival intervals having the same distribution as the original renewal process.

Remark 3.4. Suppose the underlying i.i.d. sequence X has finite positive mean $\mu > 0$. Let $\hat{S}_n = \frac{1}{n} S_n$. By SLLN we have $\hat{S}_n \to \mu$ a.s. Then $n\hat{S}_n = S_n \to \infty$, i.e.

$$\{\hat{S}_n \to \mu\} \subset \{S_n \to \infty\}.$$

From this it follows that

$$1 = P(\hat{S}_n \to \mu) \leq P(S_n \to \infty),$$

which implies that $N(t) < \infty$ a.s. for all $t \geq 0$.

Theorem 3.5. *If $(S_n)_{n\geq 0}$ is a renewal process with mean interrenewal time $0 < \mu \leq \infty$, then a.s.*

$$\lim_{t\to\infty} \frac{N(t)}{t} = \frac{1}{\mu}.$$

Proof. Since $N(t) \to \infty$ a.s. for $t \to \infty$, the SLLN implies for $(S_n)_{n\geq 0}$ that both $N(t)^{-1} S_{N(t)}$ and $N(t)^{-1} S_{N(t)+1}$ converge a.s. to μ for $t \to \infty$. The definition of $N(t)$ also implies

$$S_{N(t)} \leq t \leq S_{N(t)+1}$$

for all $t \geq 0$, i.e.

$$\frac{S_{N(t)}}{N(t)} \leq \frac{t}{N(t)} \leq \frac{S_{N(t)+1}}{N(t)}.$$

By letting $t \to \infty$ the claim follows. □

The corresponding assertion is also true for $U(t)/t$.

Theorem 3.6. *(Elementary renewal theorem) Let (S_n) be a renewal process with mean interrenewal time $0 < \mu \leq \infty$. Then*

$$\lim_{t\to\infty} \frac{U(t)}{t} = \frac{1}{\mu}.$$

We refer to [Als10] for the proof.

Lemma 3.7. *Let λ be a probability measure , $(X_t)_{t\geq 0}$ a diffusion process with distribution P_λ on the probability space $(\Omega, \mathcal{B}, P_\lambda)$ and let $\alpha \geq 1$ such that*

$$\int |x|^{1-\frac{1}{\alpha}}\, m(dx) < \infty$$

for the speed measure m. Then

1. *$E_\lambda[\tau_0^\alpha] < \infty$, if $\int |x|\, \lambda(dx) < \infty$,*
2. *$E_\pi[\tau_0^{\alpha-1}] < \infty$,*

where $\tau_0 := \inf\{t \geq 0 : X_t = 0\}$ is the first-hitting time of 0.

We leave out the proof due to its length and technicality and refer the reader to [Lin83]. The next two results are mostly adapted from [Lin83]. We corrected a few index mistakes in the proofs and give a few more details.

Lemma 3.8. *Let $Y_1, Y_2, \ldots$ be non-negative i.i.d. random variables with distribution F and $E[Y_1^\alpha] = \mu_\alpha < \infty$ for an $\alpha \geq 1$. With $S_n = \sum_{i=1}^n Y_i$, let $V(t) := \min\{S_n - t : S_n - t \geq 0\}$ be the overshoot at level $t \geq Y_1$ of the renewal process $(S_n)_n$. Then*

1. *there exists a constant C_1 such that $E[V(t)^\alpha] \leq C_1(1+t)$ for all $t \geq 0$; and*
2. *for each $\rho > 0$ there exists a constant C_2 such that $E[V(t)] \leq C_2 + \rho \cdot t$ for all $t \geq 0$.*

Proof. Let $V(t) = S_\tau - t$ be the overshoot at level t where $\tau = \inf\{n \geq 0 : S_n \geq t\}$. Note that the distribution of $Z := Y_1 - t$ is given by

$$F_Z(x) = P(Z \leq x) = P(Y_1 \leq x + t) = F(x + t).$$

Since

$$S_{n_0} - t = Y_{n_0} - t + \sum_{i=1}^{n_0 - 1} Y_i,$$

we apply the convolution formula for the independent random variables $Y_{n_0} - t$ and S_{n_0-1} (cf. A.2). Using this, it then holds for the distribution $F_{V(t)}$ of the overshoot that

$$F_{V(t)}(x) = \int_{[0,t)} F(t - s + x) dU(s).$$

Hence we have

$$\begin{aligned} E[V(t)^\alpha] &= \int x^\alpha dF_{V(t)}(x) = \int_{[0,t)} dU(s) \int_{x\geq 0} x^\alpha dF(t - s + x) \\ &\leq \int_{[0,t)} dU(s) \int_{x \geq t-s} x^\alpha dF(x) \leq \mu_\alpha U[0,t) \leq C_1(1+t) \end{aligned}$$

for some constant C_1 and for all $t \geq 0$ since $\lim_{t\to\infty} U[0,t)/t$ exists by Theorem 3.6. For the second claim, note that $S_{N(t)+1} = t + V(t)$ where $N(t)$ denotes the number of renewals in $[0,t)$. Then with Wald's identity (c.f. Theorem A.4) it follows that

$$E[V(t)] = E[S_{N(t)+1}] - t = \mu_1 \cdot E[N(t) + 1] - t = \mu_1(U[0,t) + 1) - t \leq C_2 + \rho t$$

for all $t \geq 0$ if C_2 is sufficiently large, since $U[0,t)/t \to 1/\mu_1$ a.s. □

We note that Lemma 3.8 can be extended to non-negative random variables. Indeed, if ξ is a non-negative random variable independent of $Y_1, Y_2, \ldots$ using Fubini it holds that $E[V(\xi)^\alpha] \leq C_1(1 + E[\xi])$ and $E[V(\xi)] \leq C_2 + \rho \cdot E[\xi]$.

In the following, let λ and μ be probability measures on $(-\infty, \infty)$ and let X, X' be independent diffusion processes with distributions P_λ[5] and P_μ on the probability space $(\Omega \times \Omega, \mathcal{B} \times \mathcal{B}, P_\lambda \times P_\mu)$. We also assume that the diffusions are regular[6] and in natural scale[7].

The proof of the next lemma can be found in [Lin83]. We have corrected an index mistake and adapted it to only include the for us relevant result.

Lemma 3.9. *Let $\alpha \geq 1$. Then $E[\tau_c^\alpha] < \infty$ where*

$$\tau_c := \inf\{t \geq 0 : X_t = X'_t\}$$

is the coupling time of the two processes $(X_t)_{t\geq 0}$ and $(X'_t)_{t\geq 0}$.

Proof. Let

$$\tau_0 = \inf\{t \geq 0 : X_t = 0\}$$
$$\tau'_1 = \inf\{t \geq 0 : X'_t = 1\}$$

be the first hitting times of 0 and 1 of the processes X and X' respectively. W.l.o.g. let $X'_0 \leq X_0$. The case $X'_0 > X_0$ follows analogously. Let $S_0 = \tau_0$ and recursively define

$$S_n := \inf\{s : s > S_{n-1}, X_s = 0, \exists t \text{ such that } S_{n-1} < t < s \text{ and } X_t = 1\}$$

for $n \geq 1$. The process $(S_n)_{n\geq 0}$ is a renewal process and by Lemma 3.7, its delay and recurrence times have finite α moments if $X_0 \sim \lambda$[8]. The same applies to the renewal process $(S'_n)_{n\geq 0}$ if $X'_0 \sim \mu$, where

$$S'_n := \inf\{s : s > S'_{n-1}, X'_s = 1, \exists t \text{ such that } S'_{n-1} < t < s \text{ and } X'_t = 0\}.$$

The recurrence times of the two processes have the same distribution, say F. Now define the stopping times $(\nu_i)_{i\geq 0}$ and $(\kappa_i)_{i\geq 0}$ though $S_{\kappa_0} = S_0$ and

$$S'_{\nu_0} = \inf\{S'_j : S'_j \geq S_{\kappa_0}\}, \qquad S_{\kappa_1} = \inf\{S_j : S_j \geq S'_{\nu_0}\},$$
$$S'_{\nu_1} = \inf\{S'_j : S'_j \geq S_{\kappa_1}\}, \qquad S_{\kappa_2} = \inf\{S_j : S_j \geq S'_{\nu_1}\}, \text{ etc.}$$

We now introduce an increasing sequence of σ-fields $(\mathcal{B}_i)_{i\in\mathbb{N}_0}$, defined by

$$\mathcal{B}_i = \sigma\{X_s,\ X'_t\ :\ s \leq \gamma_i,\ t \leq S'_{\nu_i}\},$$

with $\gamma_i = \inf\{s : s > S_{\kappa_i} \text{ and } X_s = 1\}$ and let $\eta = \min\{i \geq 1 : S'_{\nu_i} \leq \gamma_i\}$. Then, clearly η is a stopping time w.r.t. $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$. On the set $X'_0 \leq X_0$, we have for the coupling time that

[5] P_λ is the measure on $(\Omega, \mathcal{B})$ defined by $P_\lambda(A) = \int P_x(A)\lambda(dx)$, where P_x is the relevant Markov measure on $(\Omega, \mathcal{B})$ making $P_x(X_0 = x) = 1$.

[6] A diffusion process is said to be regular if the process hits each point $x \in \mathbb{R}$ almost surely cf. [Fre12] p. 106 ff.

[7] Any diffusion can be put into natural scale with a space transformation, cf. [Fre12] p.113 ff.

[8] if the process $(X_t)_{t\geq 0}$ has initial distribution λ

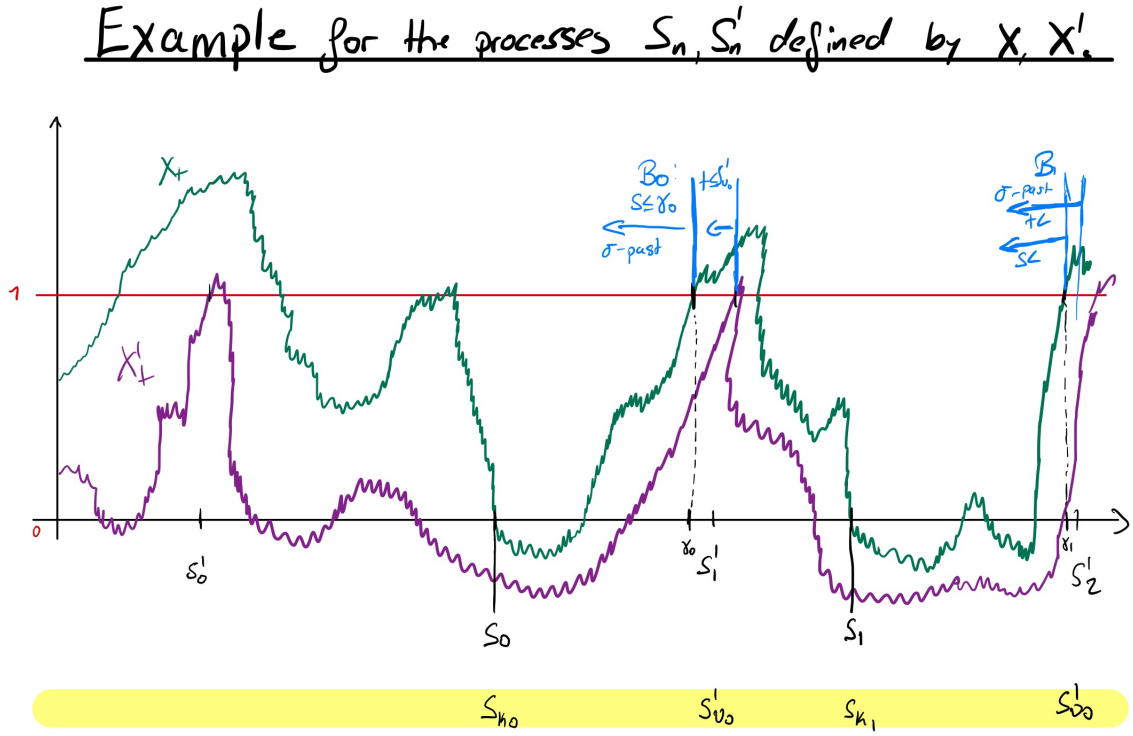

Figure 1: Example of the two processes $(X_t)_{t\geq 0}$ and $(X'_t)_{t\geq 0}$ and the construction of the stopping times in the proof of Lemma 3.9.

$\tau_c \leq S'_{\nu_\eta}$[9]. Then

$$\tau_c \leq S'_{\nu_0} + \sum_{i=1}^{\infty} U_i \cdot 1_{\{\eta \geq i\}} \tag{3.1}$$

where $U_i := S'_{\nu_i} - S'_{\nu_{i-1}}$. We now apply Lemma 3.8. We think of $S_{\kappa_i} - S'_{\nu_{i-1}}$ as an overshoot of the level $S'_{\nu_{i-1}} - S_{\kappa_{i-1}}$ for a renewal process starting at $S_{\kappa_{i-1}}$ for $i \geq 1$ and analogously of $S'_{\nu_i} - S_{\kappa_i}$ for $i \geq 1$. Using the fact that X and X' are independent, the first-moment result for overshoots at random levels implies

$$\begin{aligned} E_{\lambda,\mu}[(S_{\kappa_i} - S'_{\nu_{i-1}})^\alpha | \mathcal{B}_{i-1}] &= E_{\lambda,\mu}[(S_{\kappa_i} - S_{\kappa_{i-1}} - (S'_{\nu_{i-1}} - S_{\kappa_{i-1}}))^\alpha | \mathcal{B}_{i-1}] \\ &= E_{\lambda,\mu}[V(S'_{\nu_{i-1}} - S_{\kappa_{i-1}})^\alpha | \mathcal{B}_{i-1}] \\ &\leq C + C \cdot (S'_{\nu_{i-1}} - S_{\kappa_{i-1}}) \\ &\leq C + C \cdot U_{i-1} \end{aligned}$$

[9]Interpretation of S'_{ν_η}: Returning from 0 the process X hits 1 after X' does. Then on the set $\{X'_0 \leq X_0\}$ the processes must have crossed paths (compare Figure 1).

where C denotes a generic constant. We also have

$$E_{\lambda,\mu}[(S'_{\nu_i} - S_{\kappa_i})^\alpha | \mathcal{B}_i] \leq C + C \cdot U_{i-1}.$$

This combined with Jensen's inequality[10] and the tower property for conditional expectations yields

$$E_{\lambda,\mu}\big[U_i^\alpha | \mathcal{B}_{i-1}\big] \leq C + C \cdot U_{i-1}. \tag{3.2}$$

Because of (3.1), $E[S'^\alpha_{\nu_0}] < \infty$ and the Minkowski inequality, the claim follows if we show that

$$\sum_{i=1}^{\infty} E[U_i^\alpha \cdot 1_{\{\eta \geq i\}}]^{1/\alpha} < \infty. \tag{3.3}$$

It is well known that $V(t)$ converges in distribution as $t \to \infty$ to a limit with $[0, \infty)$ as its support and its density is $C \cdot (1 - F(x))$ with respect to the Lebesgue measure (cf. [Smi58] [AT90]). We also have $P(V(t) \leq x) > 0$ for all $t \geq 0$, $x > 0$ for our particular F. Hence $\inf_{t \geq 0} P(V(t) \leq x) > 0$ for all $x > 0$, which implies the existence of a $\gamma > 0$ such that

$$E[1_{\{\eta \geq i\}} | \mathcal{B}_{i-1}] \leq (1 - \gamma) \cdot 1_{\{\eta \geq i-1\}}$$

leading to $P(\eta \geq i) \leq (1 - \gamma)^i$. With (3.2), we have

$$\begin{aligned} E[U_i^\alpha \cdot 1_{\{\eta \geq i\}}] &= E[1_{\{\eta \geq i\}} \cdot E[U_i^\alpha | \mathcal{B}_{i-1}]] \\ &\leq E[1_{\{\eta \geq i\}} \cdot (C + C \cdot U_{i-1})] \\ &\leq C \cdot (1 - \gamma)^i + C \cdot E[U_{i-1} \cdot 1_{\{\eta \geq i\}}]. \end{aligned}$$

For (3.3) to hold, it remains to show $\sum_{i=2}^{\infty} E[U_{i-1} \cdot 1_{\{\eta \geq i\}}]^{1/\alpha} < \infty$. With the same reasoning as for (3.2), using Lemma 3.8 (2), we get the estimate

$$E[U_{i-1} | \mathcal{B}_{i-2}] \leq C + \rho \cdot U_{i-2}.$$

For $i \geq 2$ and setting $\rho = 1/2$, this yields

$$\begin{aligned} E[U_{i-1} \cdot 1_{\{\eta \geq i\}}] &\leq E[1_{\{\eta \geq i-1\}} \cdot E[U_{i-1} | \mathcal{B}_{i-2}]] \\ &\leq C \cdot (1 - \eta)^{i-1} + \frac{1}{2} \cdot E[1_{\{\eta \geq i-1\}} U_{i-2}] \\ &\leq C \cdot i \cdot \max(1 - \gamma, \frac{1}{2})^i \end{aligned}$$

and hence $\sum_{i=2}^{\infty} E[U_{i-1} \cdot 1_{\{\eta \geq i\}}]^{1/\alpha} < \infty$. □

[10] Since x^α is convex

4 An initial control problem

In this section we study moving systems that ideally stay on a fixed trajectory. However, in many applications moving objects such as for example cars or satellites, do not stay on a fixed trajectory like a train for example.

We first consider the case where the control parameters are given.

Let $\theta_0 < 0 < \theta_1$ be reals and U be the set of all Borel-measurable functions $b : [0, \infty) \times \mathbb{R} \to [\theta_0, \theta_1]$. Then by Proposition 2.10, there exists a weak solution $(\Omega, \mathcal{F}_\Omega, \mathbb{P}^x, (\mathcal{F}_t)_{t\geq 0}, W, X^b)$ to the stochastic differential equation

$$dX_t^b = b(t, X_t)dt + dW_t, \qquad X_0^b = x, \tag{4.1}$$

which is unique in law. It holds

$$\sup_{t\geq 0} E^x[|X_t|^3] < \infty,$$

which we will prove in Lemma 5.4 in the next section.

We can interpret the solution (X_t) of (4.1) as the position of our system at time t, with the optimal position being the origin. Our goal is to find the optimal drift b such that the quadratic deviation from the origin (the cost)

$$J(x, b) := \limsup_{T\to\infty} E^x \left[\frac{1}{T}\int_0^T (X_s^b)^2 ds\right] \tag{4.2}$$

is minimized for all initial values $x \in \mathbb{R}$. The corresponding value function is given by

$$V(x) = \inf_{b\in U} J(x, b). \tag{4.3}$$

We expect that the initial value x has no effect on J and V since in the long term it does not matter where the system starts.

Remark 4.1. To simplify the analysis of such systems, we do not account for the cost of steering. For example, in the case of keeping a satellite on a fixed trajectory, we might have to take into consideration the amount of "fuel" it takes to move the satellite back on track.

We also assume that there is no delay in control. This means that we allow the drift to be discontinuous. With other works, we can jump from one "extreme" control in one direction to the other "extreme" instantly. We will see that this type of "bang-bang" control is the optimal control.

We now introduce a common method for finding an optimal control b for (4.2).

4.1 Hamilton-Jacobi-Bellman equation

The Hamilton - Jacobi - Bellman (HJB) equation is a non-linear partial differential equation that provides necessary and sufficient conditions for optimality of a control with respect to a loss function. Its solution is the value function of the optimal control problem. We will briefly sketch the intuition behind the derivation of the equation for the value function

$$V(t, X_t) = \inf_{b\in U} E\left[\int_t^T u(X_s^b)ds \middle| \mathcal{F}_t\right], \qquad 0 \leq t \leq T$$

where we assume X_t^b to be given as the solution of the stochastic differential equation

$$dX_t^b = b(t, X_t^b)dt + dW_t$$

with initial condition $X_0^b = x_0 \in \mathbb{R}$ on the respective filtered probability space $\big(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, P\big)$. Let $\varepsilon > 0$ and $t_\varepsilon := t + \varepsilon$. The derivation relies on Bellman's principle of optimality [Bel57]:

"*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*"

By Bellman's principle of optimality, going from time t to time $t + \varepsilon$, we can write

$$V(t, X_t^b) = \inf_{b' \in \hat{U}} E\left[\int_t^{t_\varepsilon} u(X_s^{b'})ds + V(t_\varepsilon, X_{t_\varepsilon}^{b'})\big|\mathcal{F}_t\right],$$

where $\hat{U} \subset U$ denotes the set of all control functions b' that coincides with b up until time t. Now if we assume $V \in C^{1,2}(\mathbb{R})$ such that we can apply Itô's formula, we then have

$$V(t_\varepsilon, X_{t_\varepsilon}^b) = V(t, X_t^b) + \int_t^{t_\varepsilon} \left(\frac{\partial V}{\partial x}(s, X_s^b)dX_s + \int_t^{t_\varepsilon} \frac{\partial V}{\partial s}(s, X_s^b) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(s, X_s^b)\right) ds.$$

Combining the last two equations we get

$$\begin{aligned} 0 \geq V(t, X_t^b) - E\Bigg[&\int_t^{t_\varepsilon} u(X_S^b) + \frac{\partial V}{\partial s}(s, X_s^b) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(s, X_s^b)ds \\ &+ \int_t^{t_\varepsilon} \frac{\partial V}{\partial x}(s, X_s^b)dX_s^b + V(t, X_t^b)\big|\mathcal{F}_t\Bigg] \\ = -E\Bigg[&\int_t^{t_\varepsilon} u(X_s^b) + \frac{\partial V}{\partial s}(s, X_s^b) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(s, X_s^b)ds \\ &+ \int_t^{t_\varepsilon} \frac{\partial V}{\partial x}(s, X_s^b)dX_s^b\big|\mathcal{F}_t\Bigg]. \end{aligned}$$

and by using $dX_t^b = b(t, X_t^b)dt + dW_t$ we have

$$E\int_t^{t_\varepsilon} \frac{\partial V}{\partial x}(s, X_s^b)dX_s^b = E\int_t^{t_\varepsilon} b(s, X_s^b)\frac{\partial V}{\partial x}(s, X_s^b)ds$$

since the expectation of the Brownian integral is 0. Hence, if the expectation for $t_\varepsilon \to t$ is differentiable in t, we get the equation

$$0 \leq u(X_t^b) + b(t, X_t^b)\frac{\partial V}{\partial x}(t, X_t) + \frac{\partial V}{\partial t}(t, X_t) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(t, X_t),$$

where we have equality for the minimizing / optimal $b = b^*$. The Hamilton-Jacobi-Bellman differential equation then reads as

$$\begin{aligned} 0 &= \inf_{b\in U} u(x) + b(t, x)\frac{\partial V}{\partial x}(t, x) + \frac{\partial V}{\partial t}(t, x) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(t, x) \\ &= \inf_{b\in U} \mathcal{A}^b V(t, x) + \frac{\partial V}{\partial t}(t, x) + u(x) \end{aligned} \tag{4.4}$$

where

$$\mathcal{A}^b\varphi(t,x) := b(t,x)\frac{\partial\varphi}{\partial x}(t,x) + \frac{1}{2}\frac{\partial^2\varphi}{\partial x^2}(t,x),$$

for $t \in [0,\infty)$, $x \in \mathbb{R}$, and $\varphi \in C^{1,2}(\mathbb{R})$ defines the stochastic differentiation operator.

Remark 4.2. The assumptions that are made above are very restrictive, especially $V \in C^{1,2}(\mathbb{R})$. Some of these assumptions can be relaxed since it turns out that for applications we generally do not need to guarantee that all of the assumptions required for the derivation hold. Instead the typical scheme for solving stochastic control problems is the following:

1. Find a solution b^* of the HJB-equation (4.4) for the (still) unknown function V.
2. Inserting the minimizer b^* into the equation (4.4) yields a partial differential equation (PDE) for V, which we then solve.
3. Use a verification theorem to show that the solution of the PDE corresponds with the value function and the solution b^* is the optimal control.

Here we only considered the case for a fixed T. The ergodic case for $T \to \infty$ is a bit different. The idea is however similar, but since the derivation for this is out of the scope of this thesis, we refer to [Øks98] and [CDR08] for details. The main difference is that the time component disappears in (4.4) and the Hamilton-Jacobi-Bellman differential equation then reads as

$$0 = \inf_{b\in U} \mathcal{A}^b V(t,x) + u(x). \tag{4.5}$$

4.2 Solution to the optimal control problem

We now return to finding an optimal b for the cost (4.2). We will follow the following approach: We assume the cost to be of polynomial growth in the initial state, i.e. the integral part of (4.2) to be of the form

$$E^x\left[\int_0^T (X_s^b)^2 ds\right] \approx \varphi(x) + T\eta$$

where φ is of polynomial growth, describing the initial cost, and $T\eta$ (with $\eta \in \mathbb{R}_{>0}$) is the cost that accumulates over time. We will see that the initial cost does not influence the cost in the long term. Intuitively, it makes sense for φ to be a polynomial of degree two where the minimum is the minimal cost of the system, and the further the system begins from the optimum, the greater should be the cost term $\varphi(x)$. The Hamilton-Jacobi-Bellman equation is given by (4.5) with $u(x) = x^2 - \eta$, i.e.

$$0 = \inf_{b\in[\theta_0,\theta_1]} \mathcal{A}^b\varphi(t,x) + x^2 - \eta. \tag{4.6}$$

A solution to the HJB-equation consists of a pair $(\varphi,\eta) \in C^2(\mathbb{R}) \times \mathbb{R}$. We now heuristically solve the equation (4.6). Define

$$b^*(x) := \begin{cases} \theta_0, & \varphi(x) > 0 \\ \theta_1, & \varphi(x) \le 0. \end{cases}$$

Then

$$\inf_{b\in[\theta_0,\theta_1]} b\varphi'(x) = b^*(x)\varphi'(x),$$

hence (4.6) turns into two ODEs

$$\begin{aligned}
0 &= \inf_{u\in[\theta_0,\theta_1]} \mathcal{A}^u\varphi(t,x) + x^2 - \eta \\
&= \inf_{u\in[\theta_0,\theta_1]} u\varphi'(x) + \frac{1}{2}\varphi''(x) + x^2 - \eta \\
&= b^*(x)\varphi'(x) + \frac{1}{2}\varphi''(x) + x^2 - \eta \\
&= \begin{cases} \theta_0\varphi'(x) + \frac{1}{2}\varphi''(x) + x^2 - \eta, & \varphi(x) > 0 \\ \theta_1\varphi'(x) + \frac{1}{2}\varphi''(x) + x^2 - \eta, & \varphi(x) \le 0. \end{cases}
\end{aligned}$$

The general solution of (4.6) is given by

$$\varphi(x) = c + c'\frac{e^{-2xb^*(x)}}{2b^*(x)} + x\left(\frac{\eta}{b^*(x)} - \frac{1}{2b^*(x)^3}\right) + \frac{x^2}{2b^*(x)^2} - \frac{x^3}{3b^*(x)} \tag{4.7}$$

for constants $c, c' \in \mathbb{R}$. (Show/give details for the solution in the appendix.) By definition of b^* it holds that $-xb^*(x) > 0$ for large $|x|$. Since we are looking for a polynomial as the solution, it is reasonable for us to choose $c' = 0$. Further we assume there exists a unique $\delta \in \mathbb{R}$ such that $\varphi'(\delta) = 0$. It remains to determine η and ensure $\varphi \in C^2(\mathbb{R})$. Since we are constructing φ' to be continuous in δ, we have

$$\frac{\eta}{\theta_0} - \frac{1}{2\theta_0^3} + \frac{\delta}{\theta_0^2} - \frac{\delta^2}{\theta_0} = 0 = \frac{\eta}{\theta_1} - \frac{1}{2\theta_1^3} + \frac{\delta}{\theta_1^2} - \frac{\delta^2}{\theta_1}.$$

This leads to the equations

$$\eta - \frac{1}{2\theta_0^2} + \frac{\delta}{\theta_0} - \delta^2 = 0 = \eta - \frac{1}{2\theta_1^2} + \frac{\delta}{\theta_1} - \delta^2,$$

hence

$$\delta^2 - \frac{\delta}{\theta_0} + \frac{1}{2\theta_0^2} = \delta^2 - \frac{\delta}{\theta_1} + \frac{1}{2\theta_1^2}.$$

Solving for δ leads to

$$\delta = \frac{1}{2\theta_0} + \frac{1}{2\theta_1}.$$

Inserting δ into the equation above yields

$$\eta = \frac{1}{4\theta_0^2} + \frac{1}{4\theta_1^2}.$$

A simple calculation shows that φ'' is also continuous in δ with

$$\varphi''(\delta) = -\frac{1}{\theta_0\theta_1}.$$

It remains to show that φ is continuous in δ. To ensure continuity in δ, we can choose the constant c in (4.7). W.l.o.g. we assume $c = 0$ for $x > \delta$, i.e.

$$\varphi(x) = x\left(\frac{\eta}{\theta_0} - \frac{1}{2\theta_0}\right) + \frac{x^2}{2\theta_0^2} - \frac{x^3}{3\theta_0}, \quad x > \delta.$$

Then

$$\delta\left(\frac{\eta}{\theta_0}-\frac{1}{2\theta_0}\right)+\frac{\delta^2}{2\theta_0^2}-\frac{\delta^3}{3\theta_0}=c+\delta\left(\frac{\eta}{\theta_1}-\frac{1}{2\theta_1}\right)+\frac{\delta^2}{2\theta_1^2}-\frac{\delta^3}{3\theta_1}.$$

Solving for c, we obtain

$$c=\frac{(\theta_0-\theta_1)(\theta_0+\theta_1)^3}{24\theta_0^4\theta_1^4}.$$

Hence, $\varphi\in C^2(\mathbb{R})$ and the pair (η,φ) is a solution to the Hamilton-Jacobi-Bellman equation (4.6).

We now show that b^* is indeed the optimal control minimizing the cost (4.2).

Theorem 4.3. *A solution to the Hamilton-Jacobi-Bellman equation (4.5) is given by*

$$\varphi(x)=\begin{cases}x\left(\frac{\eta}{\theta_0}-\frac{1}{2\theta_0^3}\right)+\frac{x^2}{2\theta_0^2}-\frac{x^3}{3\theta_0}, & x>\delta,\\ x\left(\frac{\eta}{\theta_1}-\frac{1}{2\theta_1^3}\right)+\frac{x^2}{2\theta_1^2}-\frac{x^3}{3\theta_1}+\frac{(\theta_0-\theta_1)(\theta_0+\theta_1)^3}{24\theta_0^4\theta_1^4}, & x\le\delta,\end{cases}$$

for $x\in\mathbb{R}$ and

$$\eta=\frac{1}{4\theta_0^2}+\frac{1}{4\theta_1^2},\qquad \delta=\frac{1}{2\theta_0}+\frac{1}{2\theta_1}.$$

Further, with

$$b^*(x):=\begin{cases}\theta_0, & x>\delta,\\ \theta_1, & x\le\delta,\end{cases}\tag{4.8}$$

we have

$$\eta=V(x)=J(x,b^*)$$

for all $x\in\mathbb{R}$. In particular, b^* is the optimal control minimizing (4.2).

Proof. We have already shown that the pair (φ,η) constitutes a solution of the Hamilton-Jacobi-Bellman equation (4.6) and that $\varphi\in C^2(\mathbb{R})$. Since the solution X of (4.1) is an Itô process, we can use Itô's formula which yields

$$\varphi(X_T^b)=\varphi(x)+\int_0^T\left(b(s,X_s^b)\varphi'(X_s^b)+\frac{1}{2}\varphi''(X_s^b)\right)ds+\int_0^T\varphi'(X_s^b)dW_s.$$

Since φ' is a polynomial, $b\in U$ is bounded and Borel-measurable and the fourth moments are uniformly bounded (c.f. Lemma 5.4),

$$\left(\int_0^t\varphi'(X_s^b)dW_s\right)_{t\in[0,T)}$$

is a square integrable martingale. Hence we have

$$E\varphi(X_T^b)=\varphi(x)+E\left[\int_0^T b(s,X_s^b)\varphi'(X_s^b)+\frac{1}{2}\varphi''(X_s^b)ds\right].$$

We now show that η is a lower bound of the cost (4.2) by using the fact that the pair (φ, η) is a solution to the Hamilton-Jacobi-Bellman equation (4.6). We have

$$\begin{aligned}\frac{1}{T}\big(E\varphi(X_T^b) - \varphi(x)\big) + \eta &= \frac{1}{T}E\int_0^T (X_s^b)^2 ds - \frac{1}{T}E\int_0^T \Big(\mathcal{A}^b\varphi(s, X_s^b) + (X_s^b)^2 - \eta\Big)\, ds \\ &\leq \frac{1}{T}E\int_0^T (X_s^b)^2 ds.\end{aligned}$$

Since $b \in U$ and φ is a polynomial of degree 3 we have

$$\lim_{T\to\infty}\frac{1}{T}E\varphi(X_T^b) \lesssim \lim_{T\to\infty}\frac{1}{T}\sup_{t\geq 0}E\big(X_t^b\big)^3 = 0$$

(c.f. Lemma 5.4). Finally, taking the limes superior as $T \to \infty$, we have the lower bound

$$\eta \leq J(x, b)$$

for all initial values $x \in \mathbb{R}$. On the other hand, consider our optimal control candidate b^* defined as above (4.8). Note that $b^* \in U$. Then

$$\frac{1}{T}\big(E\varphi(X_T^b) - \varphi(x)\big) + \eta = \frac{1}{T}E\int_0^T (X_s^b)^2 ds$$

and hence

$$\eta = J(x, b^*).$$

This implies that b^* is the desired optimal control and

$$V(x) = \eta = \frac{1}{4\theta_0^2} + \frac{1}{4\theta_1^2}$$

is the value of the control problem for all initial points $x \in \mathbb{R}$. □

4.3 Implementation

We implement the situation where we know the values of θ_0 and θ_1. Theorem 4.3 provides the optimal control function b^* where we can calculate the optimal switching point δ using the maximal control in each respective direction. In the Simulation.py we have implemented methods to calculate the optimal switching point δ and to simulate a Brownian motion (with drift) using the Euler–Maruyama method. Figure 3 shows an example of a process of the form (4.1), where $\theta_0 = -6$ and $\theta_1 = 1$ are not symmetric, using the optimal control as proposed in Theorem 4.3.

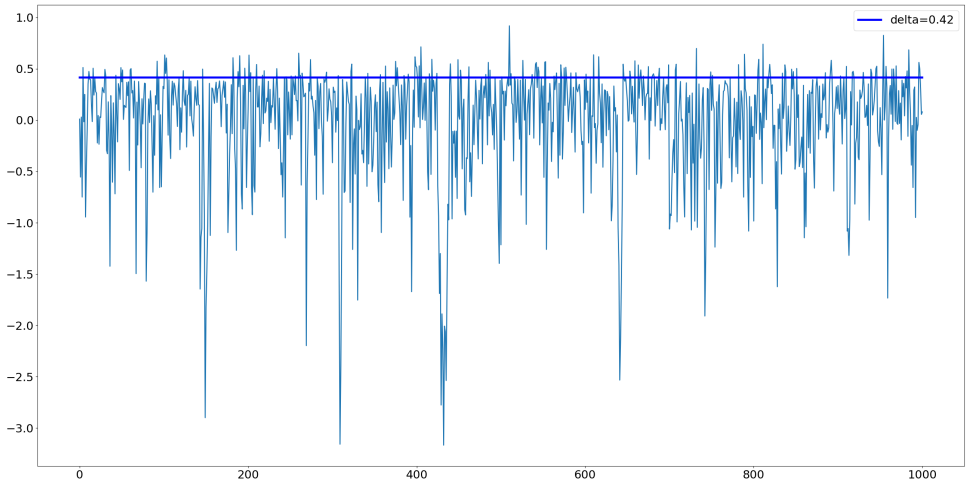

Figure 2: Process controlled with optimal switching point δ and control b^* as in Theorem 4.3.

5 Properties of a Brownian motion with broken drift

In Theorem 4.3, we have not only found the optimal control function b^* but also its structure. Given a switching point $z \in \mathbb{R}$, ideally the optimal point δ as previously defined in Theorem 4.3, we choose the corresponding drift (θ_0 or θ_1) that pushes the process back towards the point z after overshooting it. Note that in the symmetric case where $\theta_1 = -\theta_0$, the optimal switching point δ is the origin 0. If we can, however, move more in one direction than the other, i.e. $|\theta_i| < |\theta_j|$, the optimal point δ will be less or greater than 0. This is due to our goal of minimizing the long term expected cost (4.2), i.e. minimizing the area between the systems position and the origin. This motivates the following definition of SDEs where the drift function $b = b_z$ is of this form.

We now show a few properties of Brownian motions with broken drift. Let us first give a definition of the SDE's that we will be studying.

Definition 5.1. Let $\theta_0 < 0 < \theta_1$ be reals and define

$$b_z(x) := \begin{cases} \theta_0, & x > z, \\ \theta_1, & x \leq z, \end{cases}$$

with $b_\infty(x) = \theta_1$ and $b_{-\infty}(x) = \theta_0$ for all $x \in \mathbb{R}$. We call the (unique) solution $X^{z,s}$ of the family of stochastic differential equations

$$dX_t^{z,s}(\mu) = b_z(X_t^{z,s}(\mu))dt + dW_t, \quad X_s^{z,s}(\mu) \sim \mu \tag{5.1}$$

with initial distribution μ, a *Brownian motion with broken drift*. For $x \in \mathbb{R}$ we write $X_s^{z,s}(x) = X_s^{z,s}(\delta_x)$ where δ_x is the Dirac measure.

Note that since the function b_z is bounded, measurable and locally integrable and (5.1) is a time-homogeneous SDE, Proposition 2.5 provides the existence and uniqueness of a strong

solution. Note that we initial could only speak of weak solutions, since before Theorem 4.3 we did not know that the optimal control is of a "bang-bang" type and did not require the SDE to be time-homogeneous.

Definition 5.1 will provide the framework to define dynamical systems, where we will use estimators of θ_0, θ_1 and δ to find a control in the case that the values of θ_0 and θ_1 are not known.

The goal is to provide an estimate of the speed at which the solution to the SDE (5.1) converges to its stationary distribution. This will be useful to give bounds of our algorithm to estimate the optimal switching point δ for the control b_z.

Lemma 5.2. *The function*

$$f(x) = \frac{1}{\bar{\delta}} \begin{cases} e^{2\theta_0(x-z)}, & x > z, \\ e^{2\theta_1(x-z)}, & x \leq z \end{cases}$$

with $\bar{\delta} = \frac{1}{2}\left(\frac{1}{\theta_1} - \frac{1}{\theta_0}\right)$*, is the density of the stationary distribution of (5.1).*

Proof. We proceed by showing that the speed measure is finite and then use (2.9). The associated scale function to the solution of (5.1) is given by

$$s(x) = \int_z^x \exp\left(-2\int_z^y b_z(\zeta)d\zeta\right)dy = \begin{cases} \int_z^x \exp(-2(y-z)\theta_0)dy, & x \geq z, \\ \int_z^x \exp(-2(y-z)\theta_1)dy, & x < z. \end{cases}$$

The scale function is differentiable with its derivative given by

$$s'(x) = \exp(-2(x-z)b_z(x))$$

and the speed measure is given by the density

$$x \mapsto \frac{2}{\exp(-2(x-z)b_z(x))} = 2 \cdot \exp(2(x-z)b_z(x)),$$

i.e.

$$m(dx) = 2 \cdot \exp(2(x-z)b_z(x))dx.$$

Since

$$\begin{aligned} m(\mathbb{R}) &= 2\int_{\mathbb{R}} \exp(2(x-z)b_z(x))dx \\ &= 2\left[\int_z^\infty \exp(2(x-z)\theta_0)dx + \int_{-\infty}^z \exp(2(x-z)\theta_1)dx\right] \\ &= \frac{1}{\theta_1} - \frac{1}{\theta_0} < \infty \end{aligned}$$

is finite, we have that by (2.9) the density of the stationary distribution is then given by

$$f(x) = \frac{2}{m(\mathbb{R})}\exp(2(x-z)b_z(x)) = \frac{1}{\bar{\delta}}\exp(2(x-z)b_z(x)).$$

□

Lemma 5.3. *The second moment μ_z^2 of the stationary distribution exists and satisfies*

$$\mu_z^2 - \mu_\delta^2 = (z-\delta)^2$$

where $\delta = 1/2\theta_0 + 1/2\theta_1$ is defined as in the previous section.

Proof. With partial integration we have

$$\int x^2 e^{2\theta(x-z)} dx = e^{2\theta(x-z)} \left(\frac{x^2}{2\theta} - \frac{x}{2\theta^2} + \frac{1}{4\theta^3} \right) + C$$

for $C \in \mathbb{R}$. As above let $\bar{\delta} := \frac{1}{2}\left(\frac{1}{\theta_1} - \frac{1}{\theta_0}\right)$. Since

$$\begin{aligned} e^{2\theta(x-z)} \left(\frac{x^2}{2\theta} - \frac{x}{2\theta^2} + \frac{1}{4\theta^3} \right) &\underset{x\to\infty}{\longrightarrow} 0, \qquad \text{for } \theta < 0, \\ e^{2\theta(x-z)} \left(\frac{x^2}{2\theta} - \frac{x}{2\theta^2} + \frac{1}{4\theta^3} \right) &\underset{x\to-\infty}{\longrightarrow} 0, \qquad \text{for } \theta > 0, \end{aligned}$$

we have

$$\begin{aligned} \mu_z^2 &= \int_{-\infty}^{\infty} x^2 f(x) dx \\ &= \frac{1}{\bar{\delta}} \int_{-\infty}^{z} x^2 e^{2\theta_1(x-z)} dx + \frac{1}{\bar{\delta}} \int_{z}^{\infty} x^2 e^{2\theta_0(x-z)} dx \\ &= \frac{1}{\bar{\delta}} \left(\left(\frac{1}{2\theta_1} - \frac{1}{2\theta_0} \right) z^2 - \frac{1}{2} \left(\frac{1}{\theta_1^2} - \frac{1}{\theta_0^2} \right) z + \frac{1}{4\theta_1^3} - \frac{1}{4\theta_0^3} \right) \\ &= \frac{1}{\bar{\delta}} \left(\bar{\delta} z^2 - 2\bar{\delta}\delta z + \frac{(\theta_0 - \theta_1)(\theta_0^2 + \theta_1^2 + \theta_0\theta_1)}{4\theta_0^3\theta_1^3} \right) \\ &= z^2 - 2\delta z + \frac{\theta_0\theta_1(\theta_0 - \theta_1)(\theta_0^2 + \theta_1^2 + \theta_0\theta_1)}{2(\theta_0 - \theta_1)\theta_0^3\theta_1^3} \\ &= z^2 - 2\delta z + 2\eta + \frac{1}{2\theta_0\theta_1}. \end{aligned}$$

Now since

$$\mu_z^2 - \mu_\delta^2 = z^2 - 2\delta z - \delta^2 + 2\delta^2 = (z-\delta)^2,$$

the claim follows. □

We are now interested in the *coupling time*[11] of two processes X and Y satisfying (5.1) for a fixed $z \in \mathbb{R}$ and initial distributions λ and μ respectively. Consider the following system of stochastic differential equation

$$d\begin{pmatrix} X_t \\ Y_t \end{pmatrix} = \begin{pmatrix} b_z(X_t) \\ b_z(Y_t) \end{pmatrix} dt + \begin{pmatrix} 1 \\ 1 \end{pmatrix} dW_t, \quad \begin{pmatrix} X_0 \\ Y_0 \end{pmatrix} \sim \lambda \otimes \mu,$$

where λ and μ are probability measures on $\mathbb{R}$ with finite fourth moments. By Proposition 2.5 there exists a strong solution (X, Y) satisfying the strong Markov property. Since the processes

[11] The time it takes for two processes to touch each other, i.e. $\tau = \inf\{t \geq 0 : X_t = Y_t\}$.

X and Y have the same distribution on $\{t \geq \tau\}$, the Cauchy-Schwarz and Minkowski equality yield

$$\begin{aligned} E[X_t^2 - Y_t^2] &= E[(X_t^2 - Y_t^2)1_{\{t<\tau\}}] \\ &\leq \sqrt{E\left(X_t^2 - Y_t^2\right)^2} \cdot \sqrt{P(t<\tau)} \\ &\leq \left(\sup_{t\geq 0}\sqrt{E[X_t^4]} + \sup_{t\geq 0}\sqrt{E[Y_t^4]}\right)\sqrt{P(t<\tau)}. \end{aligned} \tag{5.2}$$

By proving that the right hand side of (5.2) is finite, we in turn show, by having the process $Y \sim \mu$ start with the stationary distribution of X, that X converges to its stationary distribution.

We begin by showing that the moments of a solution to (5.1) exist.

Lemma 5.4. *Let $X = X^{z,0}$ be the solution of (5.1) and θ_0, θ_1, b_z be as in Definition 5.1. Then it holds*

$$\sup_{t\geq 0} E[e^{c|X_t|}] < \infty$$

for $c < 2\min(|\theta_0|, \theta_1)$.

Proof. Consider $S : \mathbb{R} \to [0,\infty) \in C^2(\mathbb{R})$. Then, by Itô's formula it holds

$$S(X_t) = S(X_0) + \int_0^t \left(S'(X_s)b_z(X_s) + \frac{1}{2}S''(X_s)\right)ds + \int_0^t S'(X_s)ds.$$

If we assume that

$$b_z S' + \frac{1}{2}S'' = 0,$$

then $S(X)$ is a non-negative local martingale that is bounded from below and hence a supermartingale. Indeed, since $S(X)$ is a local martingale, there exists a sequence of stopping times (τ_n) with $\tau_n \to \infty$ such that the stopped process $M_t^{(n)} := S(X_{t\wedge\tau_n})$ is a martingale for each $n \in \mathbb{N}$. Then, by the conditional version of Fatou's Lemma, it holds

$$E[S(X_t)|\mathcal{F}_s] = E\left[\liminf_{n\to\infty} M_t^{(n)}|\mathcal{F}_s\right] \leq \liminf_{n\to\infty} E\left[M_t^{(n)}|\mathcal{F}_s\right] = S(X_s).$$

In particular we have

$$ES(X_t) \leq ES(X_0)$$

for $t \geq 0$. Substituting $y = S'$ we seek a solution of the ordinary differential equation

$$y'(x) = \begin{cases} -2\theta_0 y(x), & x \geq z, \\ -2\theta_1 y(x), & x < z. \end{cases}$$

One solution is given by

$$y(x) = \begin{cases} e^{2|\theta_0|(x-z)} - 1, & x \geq z, \\ 1 - e^{2\theta_1(z-x)}, & x < z. \end{cases}$$

Since $y(z+) = y(z-) = 0$ we have that y is continuous in z. Moreover, y is differentiable with continuous derivative. Now, if $\theta_1 \geq |\theta_0|$, we define

$$S(x) = \begin{cases} \frac{1}{2|\theta_0|}e^{2|\theta_0|(x-z)} - (x-z), & x \geq z, \\ -\delta + \frac{1}{2\theta_1}e^{2\theta_1(z-x)} + (z-x), & x < z, \end{cases}$$

and conversely for $\theta_1 < |\theta_0|$, we define

$$S(x) = \begin{cases} \delta + \frac{1}{2|\theta_0|} e^{2|\theta_0|(x-z)} - (x-z), & x \geq z, \\ \frac{1}{2\theta_1} e^{2\theta_1(z-x)} + (z-x), & x < z. \end{cases}$$

Then, S satisfies the desired conditions and we can conclude

$$E[e^{c|X_t|}] \lesssim E[S(X_t)] \leq E[S(X_0)]$$

for $c < 2\min(|\theta_0|, \theta_1)$ and all $t \geq 0$. □

We now turn to the right factor of (5.2). In the next Lemma 5.5, we will show the existence of an $\alpha \geq 1$ such that the coupling time satisfies $E[\tau^\alpha] < \infty$. Then, Markov's inequality yields

$$P(\tau \geq t) \leq \frac{E[\tau^\alpha]}{t^\alpha} < \infty$$

for all $t > 0$, in particular

$$\lim_{t \to \infty} t^\alpha P(\tau \geq t) < \infty.$$

The main idea to show the existence of such an α is to show that

$$\int |x|^{1-\frac{1}{\alpha}} \, m(dx) < \infty$$

for the speed measure m and then apply Lemma 3.9.

Lemma 5.5. *For every $\alpha \geq 1$, the coupling time satisfies $E[\tau^\alpha] < \infty$.*

Proof. As in the proof of Lemma 5.2, the associated scale function to the solution of (5.1) is given by

$$s(x) = \int_z^x \exp\left(-2\int_z^y b_z(\zeta)d\zeta\right) dy = \begin{cases} \int_z^x \exp(-2(y-z)\theta_0)dy, & x \geq z, \\ \int_z^x \exp(-2(y-z)\theta_1)dy, & x < z. \end{cases}$$

The scale function is differentiable with its derivative given by

$$s'(x) = \exp(-2(x-z)b_z(x))$$

and the speed measure is given by the density

$$x \mapsto \frac{2}{\exp(-2(x-z)b_z(x))} = 2 \cdot \exp(2(x-z)b_z(x)),$$

i.e.

$$m(dx) = 2 \cdot \exp(2(x-z)b_z(x))dx.$$

Then,

$$\int_{\mathbb{R}} e^{c|x|} m(dx) < \infty$$

for all $c < 2\min(|\theta_0|, \theta_1)$. In particular we have

$$\int_{\mathbb{R}} |x|^{1-\frac{1}{\alpha}} \, m(dx) < \infty$$

for any $\alpha \geq 1$. Note that, in order to apply Lemma 3.9, it is required that the underlying process is in natural scale. But since the speed measure m of X is also the speed measure of the process in natural scale $s(X)$, Lemma 3.9 yields the desired bound $E[\tau^\alpha] < \infty$ for all $\alpha \geq 1$. □

In conclusion, using Lemma 5.4 and Lemma 5.5 in the inequality (5.2), and having $Y_t \sim \mu$, where μ be the stationary distribution of X, yields the following.

Proposition 5.6. *Let $X^{z,s}$ be the solution of (5.1) and μ_z^2 the second moment of its stationary distribution. Then*

$$\begin{aligned} E\big[(X_t^{z,s})^2 - \mu_z^2\big] &\leq C \cdot \sqrt{E\big[\tau^\alpha\big]} \cdot (t-s)^{-\alpha/2} \\ &\lesssim (t-s)^{-\alpha/2} \end{aligned}$$

for constant C given by the uniformly bounded forth moments of the r.h.s. of (5.2) and for all $\alpha \geq 1$, $t \geq s$.

6 An explore first algorithm

Suppose we know that we can control the systems position X either with $\theta_0 < 0$ or $\theta_1 > 0$, we however do not know their respective values. With other words we know we can "steer the car either left or right" (θ_0 or θ_1), but we do not know the effect the steering will have. In the next sections we will present and compare different methods of *learning* or *estimating* the values of the drift θ_0 and θ_1, i.e. the effect of steering left or right.

We begin by presenting a simple algorithm that gives an estimate of the (optimal) switching point δ that we computed in Theorem 4.3.

We fix a radius $K > 0$ and a time horizon $T > 0$. The algorithm has two stages, first the *exploration* and second the *exploitation* stage. In the exploration stage we start our process at $-K$ with drift $\theta_1 > 0$. Once it hits K, we switch to drift $\theta_0 < 0$ until we hit $-K$. We repeat this until we have returned from one barrier to the other $n \in \mathbb{N}$ times. We define

$$\begin{aligned} \tau_{k+1} &= \inf\left\{t \geq \tau_k : \tilde{X}_t = (-1)^k K\right\} \wedge T, \\ \tilde{X}_t &= X_t^{(-1)^k \infty, \tau_k}\left((-1)^{k+1} K\right) \end{aligned} \tag{6.1}$$

for $t \in [\tau_k, \tau_{k+1})$, $k \in \mathbb{N}_0$ and $\tau_0 = 0$. Then the first hitting times τ_k represent the k-th time when $\tilde{X}_t$ has hit the barrier K (or $-K$) starting from the other barrier $-K$ (or K). Note that the notation (6.1) refers to the Definition 5.1, where the term $(-1)^k\infty$ sets the drift, i.e. for the control function we

$$b_{(-1)^k\infty}(x) = \begin{cases} \theta_0, & \text{if } \ k \mod 2 = 1, \\ \theta_1, & \text{if } \ k \mod 2 = 0, \end{cases}$$

where the process either starts in K or in $-K$.

In the second stage of the algorithm we use the dynamics of (5.1) with $z = \hat{\delta}$ where $\hat{\delta}$ is an estimate of the optimal switching point δ using the observed trajectory in the exploration stage.

6.1 Drift estimation

We now construct an estimator of the drift θ_i. Recall Theorem 4.3 which states that an estimate of the drift will also give an estimate of the optimal switching point δ. A reasonable estimator of θ_i using the trajectory of $\tilde{X}_t$ on a single interval $[\tau_k, \tau_{k+1})$ is then

$$\frac{\tilde{X}_{\tau_{k+1}} - \tilde{X}_{\tau_k}}{\tau_{k+1} - \tau_k} = (-1)^k \frac{2K}{\tau_{k+1} - \tau_k}$$

with $k \equiv (i+1) \mod 2$. Then, using Wald's equations (c.f. Theorem A.4)[12], we have

$$\theta_i E[\Delta\tau_k] = E[\Delta X_k] = (-1)^k 2K,$$

where $\Delta\tau_k := \tau_{k+1} - \tau_k$ hence

$$\frac{(-1)^k 2K}{E[\tau_{k+1} - \tau_k]} = \frac{\theta_i (-1)^k 2K}{(-1)^k 2K} = \theta_i,$$

[12]Note that [Hal70] extends the discrete-time version of Wald's equation to continuous time.

for $k \equiv (i+1) \mod 2$. We will consider the estimator

$$\hat{\theta}_i = (-1)^{i+1} \frac{2K}{\overline{\tau}_i}$$

of θ_i where we define the mean return times as

$$\overline{\tau}_1 := \frac{1}{n} \sum_{k=0}^{n-1} \Delta\tau_{2k} \quad \text{and} \quad \overline{\tau}_0 := \frac{1}{n} \sum_{k=0}^{n-1} \Delta\tau_{2k+1}.$$

It holds[13] that $E[\overline{\tau}_i] = \frac{2K}{|\theta_i|}$. Then, due to the fact that the $\Delta\tau_\ell$ possess the strong Markov property (cf. [Ibe13]), the $\Delta\tau_\ell$ are i.i.d. Note that the $\Delta\tau_\ell$ are the times that it takes the process $\tilde{X}_t$ to hit the respective barrier on the interval $[\tau_\ell, \tau_{\ell+1})$. Since the $\Delta\tau_\ell$ are also integrable random variables, the strong law of large numbers yields that

$$\overline{\tau}_i \to \frac{2K}{|\theta_i|}$$

a.s. as $n \to \infty$. Hence $\frac{1}{\hat{\theta}_i}$ is a consistent estimator of $\frac{1}{\theta_i}$, from which we get a consistent estimator

$$\hat{\delta} = \frac{1}{2\hat{\theta}_1} + \frac{1}{2\hat{\theta}_0}$$

of the optimal switching point δ (given by Theorem 4.3). We now compute the distribution of $\overline{\tau}_i$.

Lemma 6.1. *Let $X_t = B_t + ct$ be a Brownian motion with drift $c \in \mathbb{R}$. Then the density of the first hitting time $\tau_a := \inf\{t \geq 0 : X_t = a\}$ is given by*

$$f_{\tau_a}(t) = \frac{a}{\sqrt{2\pi t^3}} \exp\left(\frac{-(a-ct)^2}{2t}\right).$$

Proof. It is known that the density of the first hitting time of a standard Brownian motion is

$$f(t) = \frac{a}{\sqrt{2\pi t^3}} e^{-a^2/2t}$$

The key is to use Girsanov's theorem to construct a measure change

$$\left.\frac{dQ}{dP}\right|_{\mathcal{F}_t^W} = \exp\left(cW_t - \frac{1}{2}c^2 t\right) =: Z_t$$

[13] This is again an application of Wald's identity:

$$E\overline{\tau}_1 = \frac{1}{n} \sum_{k=0}^{n-1} E\Delta\tau_{2k} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{(-1)^{2k} 2K}{\theta_1} = \frac{2K}{\theta_1},$$

and analogously for $i = 0$ using $-1 \cdot \theta_0 = |\theta_0|$.

where W_t SBB, and $c \in \mathbb{R}$. Then, since on the set $\{\tau_a \leq t\} \in \mathcal{F}_t^W \cap \mathcal{F}_{\tau_a}^W = \mathcal{F}_{\tau_a \wedge t}^W$ we have $Z_{\tau_a \wedge t} = Z_{\tau_a}$, the OST implies

$$\begin{aligned} Q(\tau_a \leq t) &= E_Q\left[1_{\tau_a \leq t}\right] \\ &= E_P\left[1_{\tau_a \leq t} Z_t\right] \\ &= E_P\left[1_{\tau_a \leq t} E[Z_t | \mathcal{F}_{\tau_a \wedge t}^W]\right] \\ &= E_P\left[1_{\tau_a \leq t} Z_{\tau_a}\right] \\ &= E_P\left[1_{\tau_a \leq t} \exp\left(ca - \frac{1}{2}c^2 \tau_a\right)\right] \\ &= \int_0^t \exp(ca - \frac{1}{2}c^2 s) f(t) dt. \end{aligned}$$

The claim follows. □

The distribution of $\Delta\tau_{2k}$ (and of $\Delta\tau_{2k+1}$ respectively) is the same as the distribution of the first hitting time of a Brownian motion with drift θ_1 (and θ_0 respectively) of the level $2K$ (and $-2K$ respectively). With Lemma 6.1 it follows that the density of $\Delta\tau_{2k+(1-i)}$ is given by

$$f_{\Delta\tau_{2k+(1-i)}}(t) = \frac{2K}{\sqrt{2\pi t^3}} e^{-\frac{(2K-\theta_i t)^2}{2t}}. \tag{6.2}$$

The cumulative distribution function (CDF) of $\Delta\tau_{2k+(1-i)}$ can be written in terms of the CDF of the Normal distribution

$$\begin{aligned} F_{\Delta\tau_{2k+(1-i)}}(T) &= \int_0^T f_{\Delta\tau_{2k+(1-i)}}(t) dt \\ &= 1 - \Phi\left(\frac{2K - \theta_i T}{\sqrt{T}}\right) + \exp(8K\theta_i)\left(1 - \Phi\left(\frac{2K + \theta_i T}{\sqrt{T}}\right)\right). \end{aligned} \tag{6.3}$$

Note that for the CDF of the Normal distribution we have the identity $\Phi(-x) = 1 - \Phi(x)$ for all $x \in \mathbb{R}$. The representation (6.3) can be shown by taking the derivative of the right hand side with respect to T which we show in the Appendix A.5. We now show that $\Delta\tau_{2k}$ has finite moments.

Lemma 6.2. *The Laplace transformation of $\Delta\tau_{2k+(1-i)}$ at $\alpha \in \mathbb{R}$ is given by*

$$E\left[e^{-\alpha\Delta\tau_{2k+(1-i)}}\right] = \exp\left(2K\left(\theta_i - \sqrt{2\alpha + \theta_i^2}\right)\right)$$

in particular $\Delta\tau_\ell$ has finite absolute moments.

Proof. Using the density (6.2) we have

$$\begin{aligned} E\left[e^{-\alpha\Delta\tau_{2k+(1-i)}}\right] &= \int_0^\infty e^{-\alpha t} f_{\Delta\tau_{2k+(1-i)}}(t) dt \\ &= \int_0^\infty e^{-\alpha t} \frac{2K}{\sqrt{2\pi t^3}} e^{-\frac{(2K-\theta_i t)^2}{2t}} dt \\ &= \int_0^\infty \frac{2K}{\sqrt{2\pi t^3}} e^{2K\left(\theta_i - \sqrt{2\alpha+\theta_i^2}\right) - \frac{\left(2K - \sqrt{2\alpha+\theta_i^2}t\right)^2}{2t}} dt \\ &= e^{2K\left(\theta_i - \sqrt{2\alpha+\theta_i^2}\right)} \int_0^\infty \frac{2K}{\sqrt{2\pi t^3}} e^{-\frac{\left(2K - \sqrt{2\alpha+\theta_i^2}t\right)^2}{2t}} dt \end{aligned} \tag{6.4}$$

where we used

$$-\alpha t - \frac{(2K-\theta_i t)^2}{2t} = -\frac{(2K)^2 + (2\alpha+\theta_i)t^2 - 4K\theta_i t - 4Kt\sqrt{2\alpha+\theta_i^2} + 4Kt\sqrt{2\alpha+\theta_i^2}}{2t}$$
$$= 2K\left(\theta_i - \sqrt{2\alpha+\theta_i^2}\right) - \frac{\left(2K - \sqrt{2\alpha+\theta_i^2}t\right)^2}{2t}.$$

Now the integral term in (6.4) is the limit of the cumulative distribution of the first hitting time of the level $2K$ of a Brownian motion with drift $\tilde{\theta}_i = \sqrt{2\alpha+\theta_i^2}$. Now, by exchanging θ_i with $\tilde{\theta}_i$ in (6.3) and taking the limit $T \to \infty$ we have that the integral in (6.4) is equal to 1 and the claim follows. □

Remark 6.3. The previous Lemma 6.2 can also be shown by defining the Martingale

$$M_t = \exp\left(\sigma B_t - \frac{\sigma^2}{2}t\right) = \exp\left(\left(-c + \sqrt{c^2+2\alpha}\right)X_t - \alpha t\right)$$

where $X_t = B_t + ct$ is Brownian motion with drift $c \in \mathbb{R}$ and SBM B and an application of the optional sampling theorem. A sketch of this can be found in the Appendix (c.f. A.6).

Now since $(\Delta\tau_k)_{k\in\mathbb{N}_0}$ are independent, we have with Lemma 6.2 that

$$E\left[e^{-\alpha\overline{\tau}_i}\right] = \prod_{k=1}^{n} E\left[e^{-\frac{\alpha}{n}\Delta\tau_{2k+1-i}}\right]$$
$$= \exp\left(2Kn\left(\theta_i - \sqrt{2\frac{\alpha}{n}+\theta_i^2}\right)\right). \tag{6.5}$$

This following result.

Lemma 6.4. *The inverse Laplace transformation and thus density of $\overline{\tau}_i$ is given by*

$$f^n_{\overline{\tau}_i}(t) = 2K\sqrt{\frac{n}{2\pi t^3}}e^{-\frac{2(2K-\theta_i t)^2}{2t}}.$$

Now using the estimators of θ_i, we define

$$\hat{\delta} = \frac{1}{2}\left(\frac{1}{\hat{\theta}_0} + \frac{1}{\hat{\theta}_1}\right) = \frac{1}{4K}(\overline{\tau}_1 - \overline{\tau}_0). \tag{6.6}$$

The exploitation stage is then given by the dynamics

$$\tilde{X}_t = X_t^{\hat{\delta},\tau_{2n}}(-K)$$

with $t \in [\tau_{2n},\ T]$.

6.2 Regret

We define the regret of the dynamics as the difference of the cost of our learning algorithm $\tilde{X}$ and of the optimally controlled process $X^{\delta,0}$, with $\delta = 1/2\theta_0 + 1/2\theta_1$ defined as in Theorem 4.3. We let both processes begin in the same point $-K$. The expected regret at time T is then given by

$$R(T) := E\left[\int_0^T \tilde{X}_t^2 - \left(X_t^{\delta,0}(-K)\right)^2 dt\right]. \tag{6.7}$$

We also define the mean regret

$$\hat{R}(T) := \frac{1}{T} E\left[\int_0^T \tilde{X}_t^2 - \left(X_t^{\delta,0}(-K)\right)^2 dt\right].$$

Theorem 6.5. *Suppose the explore-first algorithm returns from the lower to the upper barrier K exactly $\left\lfloor \sqrt{T} \right\rfloor$ times during the exploration phase. Then there exists a constant $C > 0$, such that the expected and expected discounted regret of the explore-first algorithm satisfy*

$$R(T) \leq C\sqrt{T} \quad \text{and} \quad \hat{R}(T) \leq C\frac{1}{\sqrt{T}},$$

for all $T \geq 0$.

Proof. We begin by splitting the regret into the exploration and exploitation stages. Let $n := \left\lfloor \sqrt{T} \right\rfloor$ be number of times the explore-first algorithm returns form the lower to the upper barrier. We have

$$\begin{aligned} R(T) = E&\left[\int_0^{\tau_{2n}\wedge T} \tilde{X}_t^2 - \left(X_t^{\delta,0}(-K)\right)^2 dt\right] \\ &+ E\left[\int_{\tau_{2n}\wedge T}^T \left(X_t^{\hat{\delta},\tau_{2n}}(-K)\right)^2 - \left(X_t^{\delta,\tau_{2n}}(-K)\right)^2 dt\right]. \end{aligned} \tag{6.8}$$

First we find an upper bound of the first summand. Since $\tilde{X}_t$ is an Itô diffusion of the type

$$\begin{aligned} dX_t &= \theta_0 dt + dW_t, \qquad \text{on } [\tau_{2k+1}, \tau_{2k+2}) \\ dX_t &= \theta_1 dt + dW_t, \qquad \text{on } [\tau_{2k}, \tau_{2k+1}) \end{aligned}$$

we have by applying Itô's formula that

$$X_t^3 = X_s^3 + 3\theta_i \int_s^t X_\ell^2 d\ell + 3\int_s^t X_\ell d\ell + 2\int_s^t X_\ell^2 dW_\ell$$

for $0 \leq s \leq t < \infty$ and for which the last summand is a martingale[14]. Now rearranging terms and taking the expectation yields

$$E\left[\int_{\tau_{2k+1-i}}^{\tau_{2k+2-i}} X_\ell^2 d\ell\right] = \frac{1}{3\theta_i}(X^3_{\tau_{2k+2-i}} - X^3_{\tau_{2k+1-i}}) - \frac{1}{\theta_i} E\left[\int_{\tau_{2k+1-i}}^{\tau_{2k+2-i}} X_\ell d\ell\right].$$

[14] a local martingale in general. However since $E\left[\sup_{0\leq s\leq t} |X_s|\right] < \infty$ for each $t > 0$ (c.f. Lemma 5.4) it is a true martingale

Now for $E\left[\int_{\tau_{2k+1-i}}^{\tau_{2k+2-i}} X_\ell d\ell\right]$, applying Itô (to $x \mapsto x^2$) and rearranging terms as done above, yields

$$E\left[\int_{\tau_{2k+1-i}}^{\tau_{2k+2-i}} X_\ell^2 d\ell\right] = (-1)^{i+1}\frac{2K^3}{3\theta_i} + (-1)^{i+2}\frac{K^2}{\theta_i^2} + \frac{1}{2\theta_i^2}E\big[\Delta\tau_{2k+1-i}\big].$$

Since (6.5) implies $E[\overline{\tau}_i] < \infty$, this equation yields

$$\begin{aligned}
E\left[\int_0^{\tau_{2n}\wedge T} \tilde{X}_t^2 - \left(X_t^{\delta,0}(-K)\right)^2 dt\right] &\leq \sum_{k=0}^{n-1} E\int_{\tau_{2k}\wedge T}^{\tau_{2k+1}\wedge T} \tilde{X}_t^2 dt + E\int_{\tau_{2k+1}\wedge T}^{\tau_{2k+2}\wedge T} \tilde{X}_t^2 dt \\
&\leq n\cdot C + \frac{1}{2\theta_1^2}\sum_{k=0}^{n-1} E\big[\Delta\tau_{2k}\big] + \frac{1}{2\theta_0^2}\sum_{k=0}^{n-1} E\big[\Delta\tau_{2k+1}\big] \\
&= n\cdot C + \frac{n}{2\theta_1^2}E\big[\overline{\tau}_1\big] + \frac{n}{2\theta_0^2}E\big[\overline{\tau}_0\big] \\
&= n\left(C + K\left(\frac{1}{\theta_1^3} + \frac{1}{|\theta_0|^3}\right)\right) \\
&\lesssim n
\end{aligned} \tag{6.9}$$

for the constant

$$C = \left(\frac{2}{3}K^3\left(\frac{1}{\theta_1} - \frac{1}{\theta_0}\right) - K^2\left(\frac{1}{\theta_1^2} - \frac{1}{\theta_0^2}\right)\right).$$

Note that the second inequality in (6.9) is only a true inequality in the case $\tau_{2n} > T$, i.e. when the exploration stage is not completed.

To find an estimate of the regret in the exploitation stage, it is more convenient to start the processes in their respective stationary distributions and then apply Proposition 5.6. Correcting for the time it takes the process to become stationary, we have

$$\begin{aligned}
&\int_{\tau_{2n}\wedge T}^{T} \left(X_t^{\hat{\delta},\tau_{2n}}(-K)\right)^2 - \left(X_t^{\delta,\tau_{2n}}(-K)\right)^2 dt \\
&\quad= \int_{\tau_{2n}\wedge T}^{T} (\mu_{\hat{\delta}}^2 - \mu_\delta^2)dt + \int_{\tau_{2n}\wedge T}^{T} \left(\left(X_t^{\hat{\delta},\tau_{2n}}(-K)\right)^2 - \mu_{\hat{\delta}}^2\right) dt \\
&\qquad\qquad - \int_{\tau_{2n}\wedge T}^{T} \left(\left(X_t^{\delta,\tau_{2n}}(-K)\right)^2 - \mu_\delta^2\right) dt \\
&\quad= (T-\tau_{2n})^+(\mu_{\hat{\delta}}^2 - \mu_\delta^2) + \int_{\tau_{2n}\wedge T}^{T} \left(\left(X_t^{\hat{\delta},\tau_{2n}}(-K)\right)^2 - \mu_{\hat{\delta}}^2\right) dt \\
&\qquad\qquad - \int_{\tau_{2n}\wedge T}^{T} \left(\left(X_t^{\delta,\tau_{2n}}(-K)\right)^2 - \mu_\delta^2\right) dt
\end{aligned} \tag{6.10}$$

where μ_z^2 is the second moment of the stationary distribution of the solution to (5.1) for every $z \in \mathbb{R}$. Recall, that by Lemma 5.3 we have

$$\mu_{\hat{\delta}}^2 - \mu_\delta^2 = (\hat{\delta} - \delta)^2.$$

Using this, the estimate (6.6) of δ we have

$$\begin{aligned}
E\left[(T-\tau_{2n})^+(\mu_{\hat{\delta}}^2-\mu_\delta^2)\right] &\leq TE\left[\left(\frac{1}{4K}\overline{\tau}_1-\frac{1}{4K}\overline{\tau}_0-\frac{1}{2\theta_1}-\frac{1}{2\theta_0}\right)^2\right] \\
&\leq 2TE\left[\left(\frac{1}{4K}\overline{\tau}_1-\frac{1}{2\theta_1}\right)^2\right]+2TE\left[\left(\frac{1}{4K}\overline{\tau}_0-\frac{1}{2\theta_0}\right)^2\right] \\
&= \frac{2T}{16K^2}\big(\operatorname{Var}\overline{\tau}_1+\operatorname{Var}\overline{\tau}_0\big) \\
&= \frac{T}{8K^2}\left(\operatorname{Var}\frac{1}{n}\sum_{k=0}^{n-1}\Delta\tau_{2k}+\operatorname{Var}\frac{1}{n}\sum_{k=0}^{n-1}\Delta\tau_{2k+1}\right) \\
&= \frac{T}{8n^2K^2}\left(\sum_{k=0}^{n-1}\operatorname{Var}\left(\Delta\tau_{2k}\right)+\sum_{k=0}^{n-1}\operatorname{Var}\left(\Delta\tau_{2k+1}\right)\right) \\
&= \frac{1}{8K^2}\frac{T}{n}\big(\operatorname{Var}\Delta\tau_0+\operatorname{Var}\Delta\tau_1\big). \qquad (6.11)
\end{aligned}$$

Further, by Proposition 5.6, we have

$$\int_s^T E\left(\left(X_t^{z,s}(-K)\right)^2-\mu_z^2\right)dt \lesssim \int_0^T \frac{1}{t}dt = \ln T \qquad (6.12)$$

for all $z\in\mathbb{R}$ and $s\in[0,T]$. We now have the following result: In terms of n, the regret of the exploration phase is of order n (c.f. (6.9)) and the regret of the exploitation phase is of order $\frac{T}{n}$ (c.f. (6.11)). Since by assumption we have $n=\left\lfloor\sqrt{T}\right\rfloor$, both the exploration and the exploitation phase are of the same order.

Now taking the expectation of (6.10) while using (6.11) and (6.12) for the exploitation part of the regret (6.8) and using (6.9) for the exploration part, the regret of the two stages then is

$$R(T)\lesssim \underbrace{\sqrt{T}}_{\substack{\text{exploration}\\ \text{term (6.9)}}} + \underbrace{\sqrt{T}+\ln(T)+\ln(T)}_{\substack{\text{exploitation}\\ \text{(6.11) \& (6.12)}}} \lesssim \sqrt{T}$$

and

$$\hat{R}(T)=\frac{1}{T}R(T)\lesssim\frac{\sqrt{T}}{T}.$$

□

6.3 Implementation

For the implementation we make use of the simulation class (c.f. appendix [**TODO**]) which simulates the process $X_t=X_t^b$ defined as in (4.1). The SDE is simulated via the Euler–Maruyama method (c.f. appendix (**TODO**)) which is implemented in this class. Details can be found in Appendix B.

For the initialization we define a radius $K>0$ and pass an environment as `env` from the simulation class. We also define n as the number of returns from the lower to the upper barrier K of our process. As in Theorem 6.5, we set $n:=\left\lfloor\sqrt{T}\right\rfloor$ and initialize the sequence of stopping

times τ_k as tau. In the implementation we also assume for $t = k \cdot dt$ where $dt = 1 \times 10^{-m}$ is the *step size* for $k, m \in \mathbb{N}_0$.

```
class ExporeFirst:

    def __init__(self, K, env, record = True):
        self.K = K
        self.n = math.floor(math.sqrt(env.max_T))
        self.env = env
        self.record = record
        self.tau = np.zeros(2*self.n+1)
```

We now implement the exploration stage of the dynamics. Due to discretization of the time t, we cannot guarantee that the process X_t simulated in the environment env hits the barrier K or $-K$ exactly at a specific time step. Hence we relax the equality condition and set tau as the first passage time of barrier K (or $-K$), returning from $-K$ (or K).

```
def exploration(self):
        h = 1
        tau0, tau1, tau_temp= 0,0,0
        drift = 1
        X,_,_ = self.env.step(drift, self.record)
        while h <= 2*self.n: #We iterate until we have returned from one
            barrier to the other 2n times
            if drift == 1:
                if X >= self.K:
                    tau1 += self.env.t - tau_temp
                    tau_temp = self.env.t
                    drift = 0
                    h += 1
            else:
                if X <= -1.0 * self.K:
                    drift = 1
                    tau0 += self.env.t - tau_temp
                    tau_temp = self.env.t
                    h += 1

            X,_,_ = self.env.step(drift, self.record)
            self.tau0 = tau0
            self.tau1 = tau1
            if self.env.done:
                break
        return self.env.t
```

Once we initialized the first passage times tau, we can compute

$$\text{tau1} = n\overline{\tau}_1 = \sum_{k=0}^{n-1} \Delta\tau_{2k}, \qquad \text{tau0} = n\overline{\tau}_0 = \sum_{k=0}^{n-1} \Delta\tau_{2k+1}$$

and in turn estimate the switching point $\hat{\delta}$. This is implemented in the following:

```
def get_delta(self):
        return (self.tau1 - self.tau0) / (self.n * 4.0 * self.K)
```

Once we have an estimate of the optimal switching point δ we can begin the exploitation phase of the algorithm. This consists of mainly passing the estimated delta $\hat{\delta}$ to the environment and continuing the simulation with $\hat{\delta}$ until we reach our time horizon T.

```
def exploitation(self):
        done = self.env.done
        delta = self.get_delta()
        self.env.delta = delta

        while not done:
            _, _, done = self.env.step_d(record=True)
        return self.env.pos_cost
```

We now combine these three steps exploration, estimation and exploitation in the following method:

```
def exploitation(self):
        # print("Starting exploitation phase")
        done = self.env.done
        delta = self.get_delta()
        self.env.delta = delta

        while not done:
            _, _, done = self.env.step_d(record=True)
        return self.env.pos_cost
```

We can compute the mean cost of the exploitation stage with the eval_last_steps method in the simulation class. Theorem 6.5 gives us an upper bound of the regret[15], where we include the exploration stage. However the discounted cost of the exploitation stage is also of interest, since we can expect the exploration stages of different algorithms that we will cover later, like Q-learning for example, to differ significantly.

In Figure 3 we give an example of the explore first algorithm. At first this does not seem too promising, however by increasing the time horizon and decreasing the radius K, the cost of the exploitation stage improves significantly as can be seen in Figure 4. We chose these values to depict the two stages better and to display the intervals $[\tau_k, \tau_{k+1})$ for $k \in [0 : 2n-1]$ better.

A natural question arises to the choice of the radius $K > 0$. Ideally we choose K small enough such that the cost of exploration phase stays proportional to the cost of the process controlled by the optimal switching point δ, however, large enough such that we do not hinder the estimation of δ.

We can also implement the regret, defined as in (6.7), as a function of T. Let $T > 0$ be the time-horizon. We compute the estimated regret (6.7) via Monte-Carlo simulation, by running the implemented explore first algorithm $N \in \mathbb{N}$ times and then computing the average of the accumulated regrets (6.7). More specifically we compute

$$\tilde{R}(T) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\int_0^T \tilde{X}_t^2 dt - T\eta \right) \tag{6.13}$$

where $\tilde{X}_t$ is the process defined by (6.1) (i.e. the process that is defined/implemented by the explore-first algorithm) and η is the long time cost if had controlled the process with the optimal switching point δ and control function b_δ (as defined in Definition 5.1). Note that in (6.13) the

[15] i.e. difference of the cost of our learning algorithm $\tilde{X}$ and of the optimally controlled process $X^{\delta,0}$

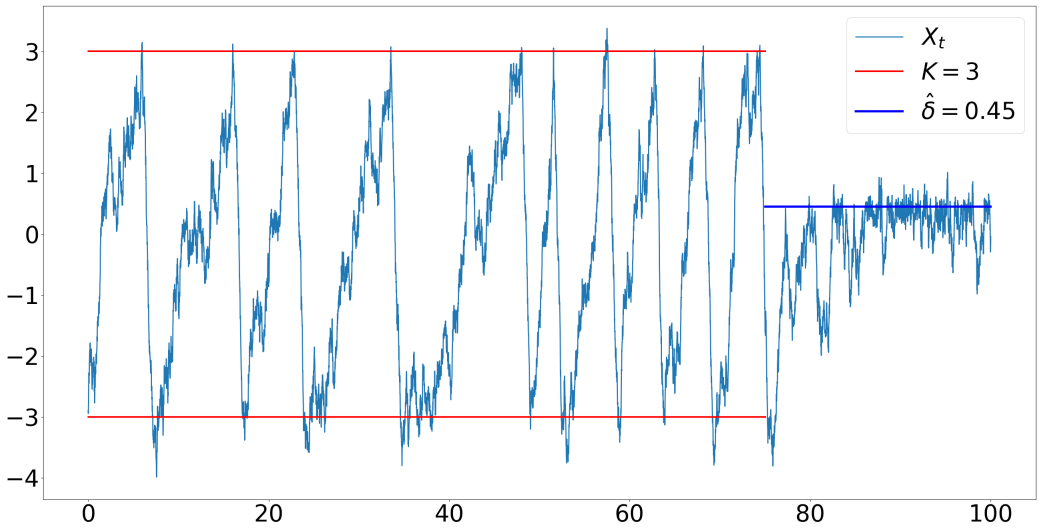

Figure 3: A simulation of the explore-first algorithm with time horizon $T = 100$ and step size $dt = 0.01$. For this example we chose $\theta_0 = -6$ and $\theta_1 = 1$. The cost of the exploitation stage is 0.85 (total cost being 2.79) whereas the optimal cost by Theorem 4.3 is 0.26.

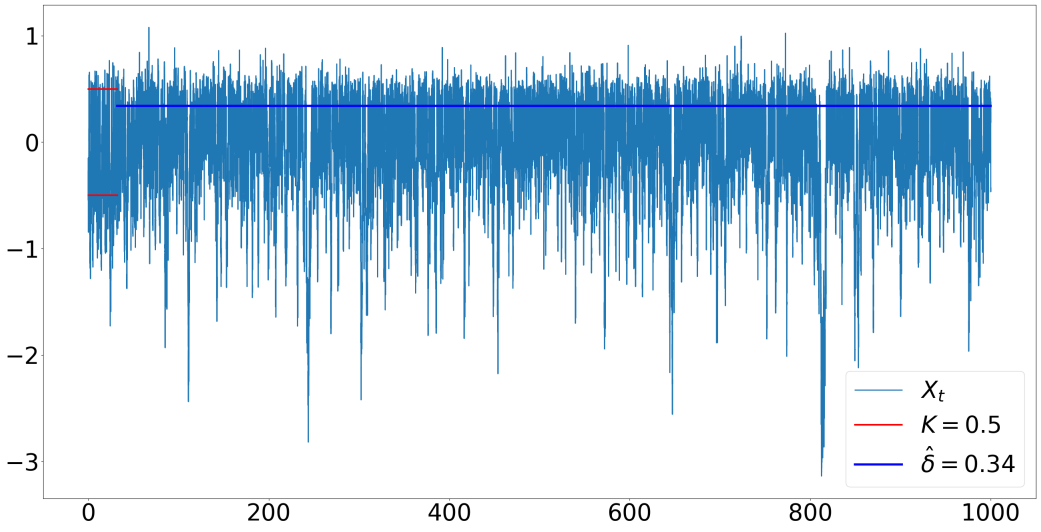

Figure 4: A simulation of the explore-first algorithm with time horizon $T = 1000$ and step size $dt = 0.01$. For this example we chose $\theta_0 = -6$ and $\theta_1 = 1$. The cost of the exploitation stage is 0.29 whereas the optimal cost by Theorem 4.3 is 0.26.

summands differ for each i, since $\tilde{X}_t$ is a diffusion process containing "randomness". The simulated regret (6.13) is depicted in Figure 5 for time-horizon $T = 1000$. The scales of Figure 5 are in log-scale, hence the implementation corresponds to the theoretical result of Theorem 6.5 where we proved that the regret grows in term of $\sqrt{T}$.

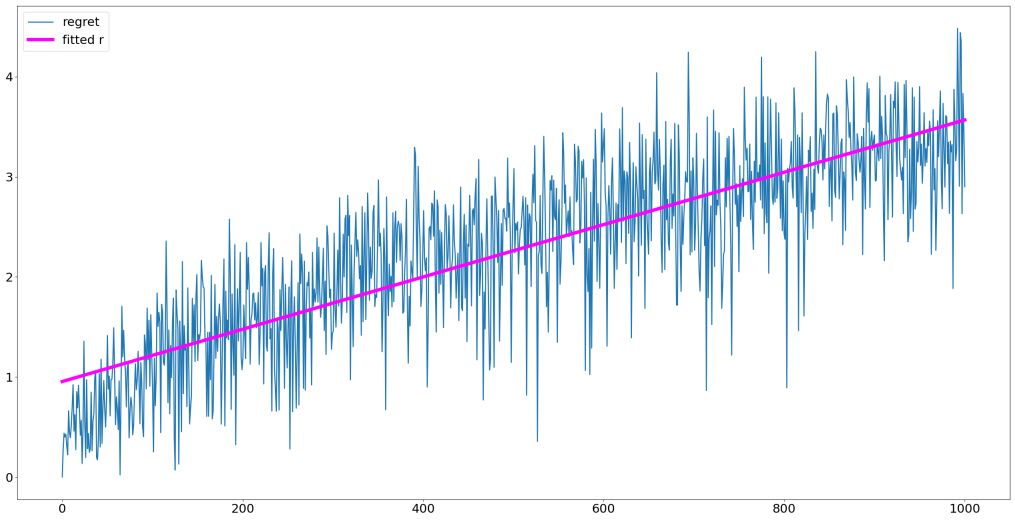

Figure 5: Regret in terms of time T of explore first algorithm in square root scale. Parameters: $K = 1$, $T = 100$, $dt = 0.001$, $(\theta_0, \theta_1) = (-6, 1)$, $iter = 100$, average cost ≈ 0.2748, optimal cost $\eta = 0.2569\overline{4}$.

7 An adaptive algorithm

In the previous section we introduced an exploration/exploitation algorithm that approximates the optimal switching point $\hat{\delta}$ with a regret of order $\sqrt{T}$. In Theorem 6.5 we see that the exploration phase is the main contribution factor for the regret. We also see in Theorem 6.5 that the regret of not being stationary in the exploitation phase is of order $\ln(T)$. Hence, if we are able to avoid the exploration and bound the regret of using an estimate of the optimal switching point δ instead of the actual δ, our hope is that we can improve the order of the regret to $\ln(T)$ versus $\sqrt{T}$. For this we introduce an adaptive algorithm. The idea is to adapt the barriers dynamically in such a way that the upper and lower barrier of the explore first algorithm are not fixed, but instead converge from above and below to δ (c.f. Figure 6).

7.1 Drift estimation

Previously we considered using the trajectories of the process between two barriers to estimate the drift. We will now generalize this. Consider a Brownian motion with drift $(X_t)_{t\in\mathbb{R}_+}$, given by

$$X_t = X_s + \theta(t-s) + W_t - W_s,$$

where $t > s$ and W is a standard Brownian motion. Then we have

$$\frac{X_t - X_s}{t-s} = \theta + \frac{W_t - W_s}{t-s}.$$

Now consider we are given an sequence stopping times $\tau_0, \dots, \tau_{2n}$ such that $E[\tau_k] < \infty$ and a sequence of Brownian motions with drift, given by

$$X_t^k = X_{\tau_{2k}} + \theta(t - \tau_{2k}) + W_t - W_{\tau_{2k}}$$

for $t \in (\tau_{2k}, \tau_{2k+1})$, with random initial values $X_{\tau_{2k}}$ which are $\mathcal{F}_{\tau_{2k}}$-measurable for all $k \in \{0, \dots, n\}$. Now consider the estimator

$$\hat{\theta}_n := \frac{\sum_{k=0}^{n-1} \Delta X^k_{\tau_{2k}}}{\sum_{k=0}^{n-1} \Delta\tau_{2k}} = \theta + \frac{\sum_{k=0}^{n-1} \Delta W_{\tau_{2k}}}{\sum_{k=0}^{n-1} \Delta\tau_{2k}} \tag{7.1}$$

with $\Delta\tau_k := \tau_{k+1} - \tau_k$ and let $\tau = \sigma(\tau_0, \dots, \tau_n)$. We set

$$Y_k = \Delta W_{\tau_{2k}}.$$

Since the τ_{2k} are a.s. finite, the optional sampling theorem (A.3) yields

$$E[Y_k] = E[\Delta W_{\tau_{2k}}] = 0$$

and since Y_ℓ is independent of Y_k for $\ell \neq k$ (strong Markov property) we have

$$\operatorname{Var}(Y_k) = E\big[\Delta\tau_{2k}\big].$$

Now, if we assume

$$\sum_{k=0}^{n} E\big[\Delta\tau_{2k}\big] \underset{n\to\infty}{\longrightarrow} \infty, \tag{7.2}$$

then with Chebyshev's inequality we have for every $\varepsilon > 0$ that

$$P\left(\frac{\sum_{k=0}^{n-1}\Delta W_{\tau_{2k}}}{\sum_{k=0}^{n-1}\Delta\tau_{2k}} > \varepsilon\right) \leq \frac{\operatorname{Var}\sum_{k=0}^{n-1}\Delta W_{\tau_{2k}}}{\varepsilon^2\left(\sum_{k=0}^{n-1}E\big[\Delta\tau_{2k}\big]\right)^2} = \frac{1}{\varepsilon^2\sum_{k=0}^{n-1}E\big[\Delta\tau_{2k}\big]} \longrightarrow 0$$

for $n \to \infty$. Hence with the condition (7.2) we have by (7.1) that $\hat{\theta}_n$ is a consistent estimator of θ.

7.2 The algorithm

We fix a decreasing sequence of radii $(K_k)_{k\in\mathbb{N}_0}$ in $\mathbb{R}_+$ that converge to 0. Let $\varepsilon > 0$ and let $k_0 \in \mathbb{N}_0$ such that for all $k > k_0$ we have $K_k < \varepsilon/2$. First we initialize the estimator $\hat{\delta}_k = 0$ for $k = 0, 1, 2$.

Now the algorithm proceeds as follows. We start in $-K_0 + \hat{\delta}_0$ with drift θ_1. Once we hit $K_1 + \hat{\delta}_1$ we switch to drift θ_0 until we return back to $-K_2 + \hat{\delta}_2$. We repeat this, while for $k_0 \geq k > 2$ we use the trajectories for the estimate $\hat{\delta}_k$ of δ. While we repeat this, if one of the following cases arises

$$-K_k + \hat{\delta}_k > K_{k+1} + \hat{\delta}_{k+1} \tag{7.3}$$

$$K_k + \hat{\delta}_k < -K_{k+1} + \hat{\delta}_{k+1} \tag{7.4}$$

after hitting the barrier $(-1)^s K_k + \hat{\delta}_k$ ($s = 1$ for the case (7.3) and $s = 2$ for (7.4)), we continue using the current drift and do not switch to the other until we hit $-K_{k+1} + \hat{\delta}_{k+1}$ in the case of (7.3) or until we hit $K_{k+1} + \hat{\delta}_{k+1}$ in the case of (7.4). This is to prevent the system of diverging to $+\infty$ in the case of (7.3) or to $-\infty$ in the case of (7.4).

For $k > k_0$ we fix the radius to the upper $K_u = K_{2k+1} = K_{k_0} + \hat{\delta}_{k_0}$ and lower $K_l = K_{2k} = -K_{k_0} + \hat{\delta}_{k_0}$ barriers.

Note that these technicalities are to ensure the condition (7.5). We cannot have the radii converge to zero since otherwise the sum in (7.5) would be bounded and we cannot ensure the consistency of the estimator of δ. We however choose ε to be small.

We define

$$\tau'_{k+1} = \inf\left\{t \geq \tau_k : \tilde{X}_t = (-1)^k K_{k+1} + \hat{\delta}_{k+1}\right\},$$

$$\tilde{X}_t = \begin{cases} X_t^{(-1)^{k+1}\infty,\tau'_k}\big((-1)^{k+1}K_k + \hat{\delta}_k\big), & \text{if (7.3) or (7.4) hold} \\ X_t^{(-1)^{k}\infty,\tau'_k}\big((-1)^{k+1}K_k + \hat{\delta}_k\big), & \text{otherwise} \end{cases}$$

for $t \in [\tau'_k, \tau'_{k+1})$ and $\tau'_0 = 0$. Now let τ_k be the stopping time defined by τ', where in the cases (7.3) and (7.4) we combine the time intervals $[\tau'_k, \tau'_{k+1}]$ and $[\tau'_{k+1}, \tau'_{k+2}]$, where $\tilde{X}_t$ has the same drift, to one time-interval $[\tau_k, \tau_{k+1}] = [\tau'_k, \tau'_{k+2}]$. Then, $\tilde{X}_t$ has drift θ_1 for $t \in [\tau_{2k}, \tau_{2k+1})$ and drift θ_0 for $t \in [\tau_{2k+1}, \tau_{2k+2})$, $k \in \mathbb{N}_0$. We also define the estimators

$$\hat{\delta}_k = \frac{1}{2}\left(\frac{1}{\hat{\theta}_0^k} + \frac{1}{\hat{\theta}_0^k}\right),$$

$$\hat{\theta}_i^k = \frac{\sum_{\ell=0}^{k-1}\Delta\tilde{X}_{\tau_{2\ell+(1-i)}}}{\sum_{\ell=0}^{k-1}\Delta\tau_{2\ell+(1-i)}}.$$

Note that

$$\begin{aligned}\Delta\tilde{X}_{\tau_{2\ell+(1-i)}} &= \tilde{X}_{\tau_{2\ell+1+(1-i)}} - \tilde{X}_{\tau_{2\ell+(1-i)}} \\ &= (-1)^{1-i}\big(K_{2\ell+2-i} + K_{2\ell+1-i}\big) + \hat{\delta}_{2\ell+2-i} - \hat{\delta}_{2\ell+1-i}. \end{aligned} \tag{7.5}$$

We now show the condition (7.5). Let $2\ell \geq k_0$. Then the distribution of $\Delta\tau_{2\ell+(1-i)}$ is the same as the distribution of the first hitting time of a Brownian motion with drift θ_i of the level $(-1)^{1+i} \cdot 2K_{k_0}$. As in the explore first algorithm we have for $2\ell \geq k_0$ that

$$E\left[\Delta\tau_{2\ell+(1-i)}\right] = \frac{2K_{k_0}}{|\theta_i|} > 0.$$

Hence for $2\ell \geq k_0$ we have

$$E\left[\sum_{\ell=k_0}^{k-1} \Delta\tau_{2\ell+(1-i)}\right] \underset{k\to\infty}{\longrightarrow} \infty$$

and we have that $\hat{\theta}_i^k$ is a consistent estimator of θ_i.

Remark 7.1. The idea of the Adaptive algorithm is that the barriers converge from above and below to an estimator of the optimal switching point δ. It is still an open question how to construct this algorithm in such a way that we can guarantee the barriers to converge to the optimal delta. This is the reason we stop updating the barriers after k_0 and do not continue decreasing the radii K_k towards 0.

7.3 Implementation

This adaptive algorithm is implemented in Adaptive.py and like the other algorithms, it uses the simulation environment that simulates a Brownian motion with drift. We initialize the class Adaptiv_A for the adaptive algorithm as follows:

```
class Adaptive_A:

    def __init__(self, K, env, record=True):
        self.K = K
        self.n = math.floor(math.sqrt(env.max_T))
        self.env = env
        self.record = record
        self.xData = []
```

Here, env is the environment that is passed and we initialize the barrier K ($=K_0$). The next method estimation implements the adaptive algorithm. It runs runs until the process $\tilde{X}_t$ reaches the set time horizon T and returns the most recent δ estimate.

```
def estimation(self):
        n_theta0, n_theta1, theta0, theta1 = 0,0,0,0
        d_theta0, d_theta1 = 1,1
        tau0, tau1, tau2 = 0, 0, 0
        n = 0
        k_v_light = self.K
        delta_est = 0
        drift = 1
        X0 = self.env.X
```

```
        X, _, _ = self.env.step(drift, self.record)

        while not self.env.done:

            if drift == 1:
                if X >= k_v_light + delta_est:
                  prev = k_v_light + delta_est
                    n += 1
                    X1 = self.env.X
                    n_theta1 += X1 - X0
                    X0 = X1
                    tau1 = self.env.t
                    d_theta1 += tau1 - tau0
                    tau0 = tau1
                    drift = 0
                    theta1 = n_theta1 / d_theta1

                    if theta1 != 0 and theta0 != 0:
                        delta_est = self.compute_delta(theta0, theta1)
                        if k_v_light > self.env.dt + 0.01:
                            k_v_light = 1.0 / n
                            if prev < -1.0 * k_v_light + delta_est:
                                drift = 1
            else:
                if X <= -1.0 * k_v_light + delta_est:
                  prev = -1.0 * k_v_light + delta_est
                    n += 1
                    X1 = self.env.X
                    n_theta0 += X1 - X0
                    X0 = X1
                    tau1 = self.env.t
                    d_theta0 += tau1 - tau0
                    tau0 = tau1
                    drift = 1
                    theta0 = n_theta0 / d_theta0
                    if theta1 != 0 and theta0 != 0:
                        delta_est = self.compute_delta(theta0, theta1)
                        if k_v_light > self.env.dt + 0.01:
                            k_v_light = 1.0 / n
                            if prev > k_v_light + delta_est:
                                drift = 0

            X, _, _ = self.env.step(drift, self.record)

        return delta_est
```

For the estimators $\hat{\theta}_i^k$ we calculate iteratively the sums of the numerator and denominator (denoted by n_theta and d_theta) and then calculate the estimators $\hat{\theta}_i^k$ (denoted by theta0 and theta1) for the k-th update. For the sequence of radii we set $K_k = 1/k$, and we stop decreasing the radii K_k and barriers once $K_k < 0.01$. This is due to the condition (7.2) and to numerical stability issues. Experiments have shown that if the sequence (K_n) is too small, that then the estimates of the θ_i will start to become inaccurate. Figure 6 depicts how the barriers converge towards an estimator of δ[16].

[16]Here we used a sequence of radii that converge quite slow compared to $1/n$ to 0 in order to depict the process better. More specifically we used the sequence $K_{k+1} = 0.9 \cdot K_k$, and $K_0 = 1$.

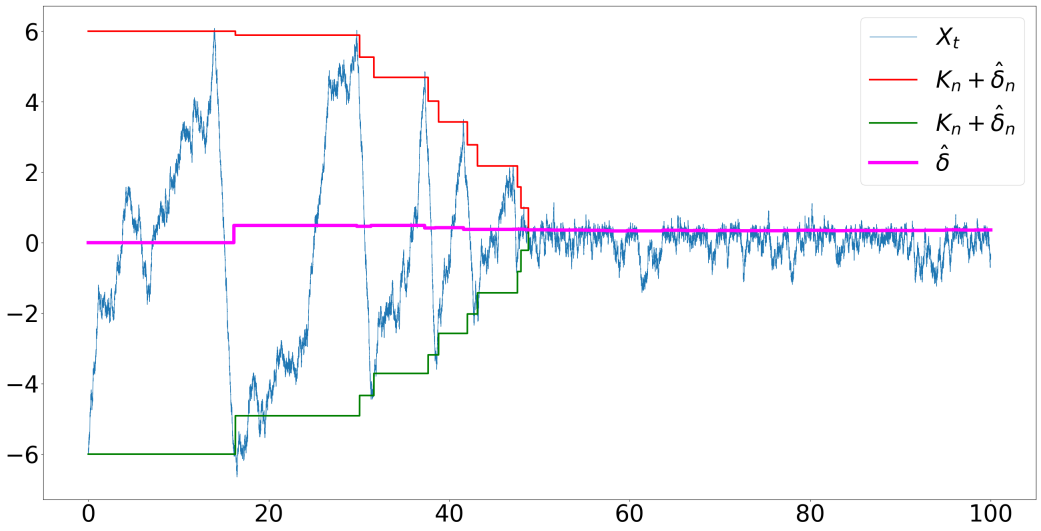

Figure 6: A simulation of the adaptive algorithm with time horizon $T = 100$ and step size $dt = 0.001$. For this example we chose $\theta_0 = -6$ and $\theta_1 = 1$. Generally it is better to choose K_0 to be small, the choice $K_0 = 6$ however provides a nice illustration of how the algorithm works. Note that the barriers $-K_n + \hat{\delta}_n$ and $K_n + \hat{\delta}_n$ converge for $n \to \infty$ to an estimator of δ.

As for the explore first implementation we use Monte-Carlo simulation to calculate the expected regret up to a time horizon $T > 0$. As before (6.13) we compute

$$\tilde{R}(T) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\int_0^T \tilde{X}_t^2 dt - T\eta \right) \tag{7.6}$$

where $\tilde{X}_t$ is the process defined by (6.1) (i.e. the process that is defined/implemented by the adaptive algorithm) and η is the long time cost if had controlled the process with the optimal switching point δ and control function b_δ (as defined in Definition 5.1). Figure 7 depicts the expected regret of the implementation of the adaptive algorithm (7.6) in log-scale for $T \in [0 : 1000]$. Here we see that the it seems plausible that the expected regret grows logarithmic, an improvement compared to the square-root growth in T of the regret 6.5 of the explore first algorithm. If this is the case in general is a open result and will be worked on in the future.

In Figure 8 we depict the error $\varepsilon = |\delta - \hat{\delta}|$ of the estimate $\hat{\delta}$ of the optimal switching point δ. Here we see that the error ε decreases very quickly, which in turn means that the estimate converges very quickly to the optimal switching point. The reason that the error ε does not continue decreasing towards 0 for $T > 400$ is that we stop decreasing the radii after a certain point (c.f. Remark 7.1).

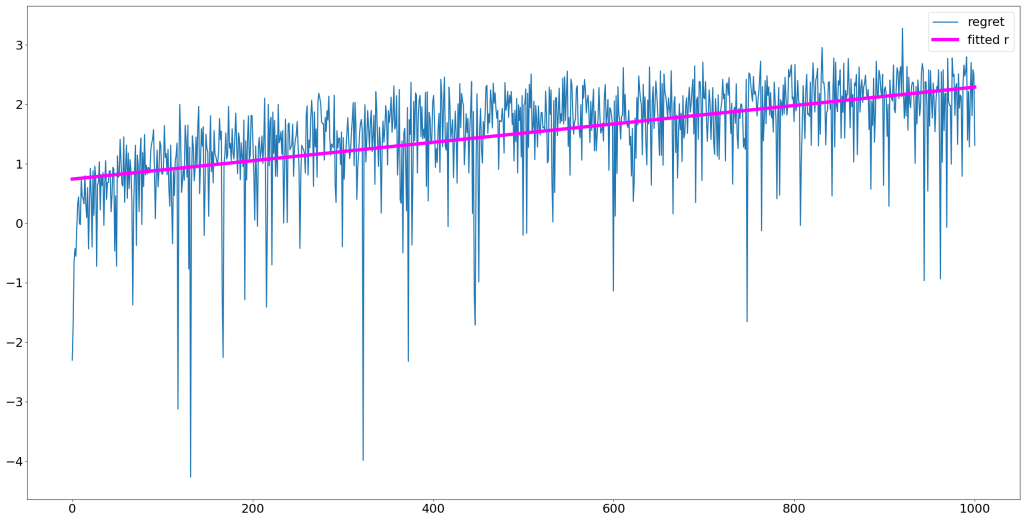

Figure 7: A simulation of the expected regret of the adaptive algorithm with time horizon $T = 1000$ and step size $dt = 0.001$ in log-scale. We chose $(\theta_0, \theta_1) = (-6, 1)$. Here we see that the regret grows proportional to $\ln(T)$, however, if this is actually the case is still an open question.

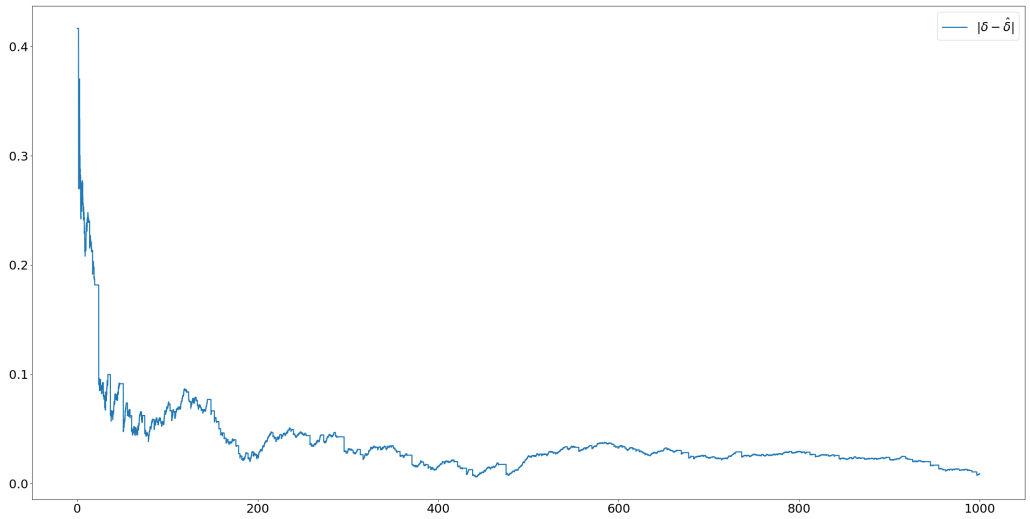

Figure 8: A simulation of the error w.r.t. $T \in [0, 1000]$ between the estimate $\hat{\delta}$ obtained by the adaptive algorithm and the optimal δ. The step size is $dt = 0.001$. We choose $(\theta_0, \theta_1) = (-6, 1)$. Here we see decreases very quickly and that taking ε^2 for the error $\varepsilon = |\delta - \hat{\delta}|$ represents the added cost by using the estimated switching point vs the optimal one.

8 A reinforcement-learning approach

There are three types of machine learning (c.f. [Lap18]):

1. **Supervised-learning:** uses pre-labeled data to train an algorithm, which then can make predictions on new unlabeled data.

2. **Unsupervised-learning:** finds patterns in unlabeled data-sets without any human guidance.

3. **Reinforcement-learning:** a general approach to solve reward based problems. Here an agent is trained / learns how to act in an environment in such a way to maximize the reward or minimize the cost.

In this chapter we will examine how we can apply reinforcement-learning for the problem at hand, controlling the drift of a Brownian motion. In most of the literature, one wishes to maximize the reward of taking certain actions. We however wish to minimize the cost of a system, as in the initial control problem (4.1), or more specifically (4.2). Because of this, in many definitions and equations that follow, we replace sup or max with inf or min.

The theory of reinforcement learning is very vast and out of the scope of this thesis. We refer to [Lap18] for details and begin by giving a brief introduction to reinforcement-learning and its applications.

Over the years reinforcement learning has become a popular method for solving complex problems, for example:

- Play many Atari games better than humans.
- Trade stocks.
- Autonomous driving.
- Robotics: Making a robot walk like a human.

The main two attributes in reinforcement learning are the *agent* and the *environment*. The agent can interact with the environment by choosing an action and will then in turn receive a reward together with a new state from the environment. Formally we model the environment as a Markov decision process (MDP). A Markov decision process is described as a stochastic process $(X_t)_{t\in\mathbb{N}_0}$ (with the Markov property, i.e. Markov chain) where we extend it with actions and rewards/costs, which describes the evolution of the states of a system. Formally we define the following. The distribution of such a system naturally depends on the chosen policy, however, for a fixed Markov type policy, the dependence structure is in the form of a Markov chain.

Definition 8.1. Let $X_0, X_1, X_2, \dots$ be a sequence of random variables with values in a finite set E. Then $(X_t)_{t\in\mathbb{N}_0}$ is called *Markov chain* if

$$P(X_{t+1} = y | X_0 = x_0, \dots, X_{t-1} = x_{t-1}, X_t = x) = P(X_{t+1} = y | X_t = x),$$

for all $t \in \mathbb{N}_0$ and all $x, y, x_0, x_1, \dots, x_{t-1} \in E$ with $P(X_0 = x_0, \dots, X_{t-1} = x_{t-1}, X_t = x) > 0$. We call $P(X_{t+1} = y | X_t = x) =: p_t(y|x)$ the *transition probabilities*.

The evolution of states is as follows: We start at time $t = 0$ in some state $x_0 \in E$ and transition to a state $x_1 \in E$ by $p_0(x_1|x_0) =: q_0$. Next, we move to state x_2 with transition probability $p_1(x_2|x_1) =: q_1$, and so on. We represent this dynamic as

$$x_0 \underset{q_0}{\longrightarrow} x_1 \underset{q_1}{\longrightarrow} x_2 \longrightarrow \cdots$$

Note that the distribution of a Markov chain is fully determined by the distribution of X_0 and the transition probabilities. Further, given any distribution η on E and, for each $x \in E$, $t \in \mathbb{N}_0$ and stochastic vector $(p_t(y|x))_{y\in E}$, there exists a Markov chain with initial distribution η and transition probabilities $p_t(y|x)$, $x, y \in E$, $t \in \mathbb{N}_0$.

Now, by including actions, rewards and discounting into a Markov process, we can formally define a Markov decision process.

Definition 8.2. A *Markov decision process* (MDP) is given by a tuple $M = \big(E, A, (A(x))_{x\in E}, p, r\big)$ where

- E is a finite set, called *state space*,
- A is a finite set, called *action space*,
- for each $x \in E$ we have a set $A(x) \subseteq A$, called the *admissible actions*,
- for each $x \in E$ and $a \in A$, the tuple $\big(p(y|x,a)\big)_{y\in E}$ is a stochastic vector, representing the *transition probabilities*,

- for each $x \in E$ and $a \in A$ the value $c(x,a) \in \mathbb{R}$ represents the *one-stage cost* of the system at state x if the action a is taken[17] and
- $\gamma \in (0,1)$ is called the *discount factor.*

The discount factor is some value in $(0,1)$ which quantifies the importance of future costs (or depending on the context, future rewards). Values closer to 0 lead to the agent preferring immediate rewards whereas values closer to 1 lead the agent to consider future rewards over immediate rewards.

Remark 8.3. In many applications the state space is not finite. In many cases the assumption of finite state-/action- space can be relaxed, however technical issues (like measurability, existence of maximizers etc.) have to be taken into account. For the underlying problem we have an infinite state space[18], but we will discuss a way to discretize the state space.

Now, the goal of reinforcement learning is that the agent does not just choose any actions over time but only actions that minimize the expected accumulated cost. With other words, our goal is that the agent chooses the optimal *policy.*

Definition 8.4.

1. For each $t \in \mathbb{N}_0$ we call a mapping $\pi_t : E \to A$ a *pure decision rule* if $\pi_t(x) \in A(x)$ for all $x \in E$. A *pure policy*, also called a *pure strategy*, is a sequence $\pi = (\pi_0, \pi_1, \pi_2, \ldots)$ of decision rules.
2. For each $t \in \mathbb{N}_0$ we call a mapping

$$\begin{aligned} \pi_t : E &\to \mathcal{P}(A) := \{q : q \text{ is a stochastic vector on } A\} \\ x &\mapsto (\pi_t(a|x))_{a \in A} \end{aligned}$$

(admissible) *decision rule*[19]. A (admissible) *policy*, also called (admissible) *strategy*, is a sequence $\pi = (\pi_0, \pi_1, \pi_2, \ldots)$ of (admissible) decision rules.

Now, given a Markov decision process $M = \big(E, A, (A(x))_{x \in E}, p, r\big)$ and an admissible policy π, we define the expected cost at time t given $X_t = x$ as

$$c_{t,\pi}(x) := \sum_{a \in A(x)} \pi_t(a|x) c(x,a).$$

For strategies we have the transition probabilities

$$p_{t,\pi} := \sum_{a \in A} \pi_t(a|x) p(y|x,a).$$

Definition 8.5. The *value function* of a policy π for the infinite time horizon with discounting is given by

$$V^\pi(x) = E\left[\sum_{t=0}^{\infty} \gamma^t c_{t,\pi}(X_t) \,\middle|\, X_0 = x \right].$$

[17] Most literature this is defined as the *one-stage reward* $r(x,a)$ of the system at state x if action a is taken. Since in our case the ideal reward is 0, c.f. (4.2), we regard the reward as a "cost" which we wish to minimize.

[18] for each $t \in [0,\infty)$ the position of the process $X_t^{z,s}$ considered in (5.1) is a state

[19] For a decision rule it holds $\sum_{a \in A(x)} \pi_t(a|x) = 1$ for all $x \in E$

We call

$$V^*(x) := \inf_\pi V^\pi(x),$$

with $x \in E$ the *value function* of the Markov decision process and call a policy π^* an *optimal policy* if

$$V^*(x) = V^{\pi^*}(x)$$

for all $x \in E$.

Definition 8.6. In a discounted infinite horizon problem, let π be a policy. Then the Q - *function* Q^π, also called the *state-action value*, is defined by

$$Q^\pi(x,a) := V^{\pi^a}(x) = E\left[\sum_{t=0}^{\infty} \gamma^t c_{t,\pi_t^a}(X_t) \,\middle|\, X_0 = x\right]$$

for all $a \in E$, $a \in A(x)$, where

$$\pi_t^a(x) := \begin{cases} \pi_t(x), & t \geq 1, \\ a, & t = 0. \end{cases}$$

The corresponding *optimal Q-function* Q^* is then defined by

$$Q^*(x,a) := \inf_\pi Q^\pi(x,a)$$

for all $x \in E$, $a \in A(x)$.

8.1 Introduction to Q-learning

One very popular value-iteration based reinforcement learning method is *Q-learning*. The foundation is given by the optimal Bellman equation

$$Q^*(x,a) = c(x,a) + \gamma \sum_{y \in E} \left(p(y|x,a) \min_{b \in A(x)} Q^*(y,b) \right) \tag{8.1}$$

and

$$V^*(x) = \min_{a \in A(x)} Q^*(x,a).$$

In this notation, $Q^*(x,a)$ represents the expected cost of taking action a in state x. On the right hand side of (8.1) $c(x,a)$ represents the cost of taking action a in state x whereas the second summand is the discounted value of the successor states. The advantage of Q-learning compared to other methods is that the minimization in (8.1) takes place in the expectation.

In the following we assume $A(x) = A$ for all $x \in E$ since for the problem that we are considering, the action space will be the same for each state.

In most cases, the transition probabilities $p(y|x,a)$ are not available. We then have to approximate them via Monte-Carlo simulation. For this, consider independent random variables $Y_n(x,a)$, $x \in E$, $a \in A$, $n \in \mathbb{N}$, such that $Y_n(x,a) \sim p(\cdot|x,a)$ for all n, x, a. For a given sequence $(\alpha_n)_{n \in \mathbb{N}}$ of positive reals, we approximate Q^* as follows: We start with some arbitrary function Q_0 (in most cases as $Q_0 = 0$) and define recursively

$$Q_{n+1}(x,a) = (1-\alpha_n) Q_n(x,a) + \alpha_n \left[c(x,a) + \gamma \min_{b \in A} Q_n(Y_{n+1}(x,a), b) \right]. \tag{8.2}$$

Here, the sequence $(\alpha_n)_{n\in\mathbb{N}}$is chosen to converge to 0, but not too fast, i.e.

$$\sum_{n\in\mathbb{N}} \alpha_n = \infty, \qquad \sum_{n\in\mathbb{N}} \alpha_n^2 < \infty.$$

Note that by (8.1) $Q^*(x,a)$ is the best predictor (in the L^2 sense) for

$$c(x,a) + \gamma \min_{b\in A(x)} Q^*\big(Y_{n+1}(x,a), b\big).$$

Then the term $c(x,a) + \gamma \min_{b\in A(x)} Q_n\big(Y_{n+1}(x,a), b\big) - Q_n(x,a)$ can be rewritten as

$$-\frac{\alpha_n}{2}\frac{d}{dq}|_{q=Q^*(x,a)} \left(c(x,a) + \gamma \min_{b\in A(x)} Q^*\big(Y_{n+1}(x,a), b\big) - q \right),$$

hence the Q-Learning algorithm can be regarded as a gradient-descent algorithm, which tries to minimize the quadratic error between the prediction $Q^*(x,a)$ and the target $c(x,a) + \gamma \min_{b\in A(x)} Q^*\big(Y_{n+1}(x,a), b\big)$.

Remark 8.7. This method is also called *tabular Q-Learning*, since the Q-Values are stored in a table. The algorithm can be summarized as follows (c.f. [Lap18] Ch. 6) :

1. Create a empty Table $Q(s,a)$.
2. By interacting with the environment we receive a tuple (s,a,c,s') (state, action, cost/reward, next state). Here, one has to decide how the action is chosen. We can either choose an action randomly or decide based off of what we have learned, i.e. the Q-Values. This is the "exploration" vs. "exploitation" dilemma. There is no generalized rule for this decision[20].
3. Update the Q-Values using the Bellman equation (8.2):

$$Q(x,a) \leftarrow (1-\alpha_n)Q(s,a) + \alpha_n \left[c(s,a) + \gamma \min_{b\in A} Q(s',b) \right].$$

4. Check if the convergence requirements are fulfilled. If not, return to step 2.

8.2 A Q-Learning based algorithm

In this section we present an algorithm based on the classic Q-learning algorithm[21] to estimate the optimal switching point δ. As in the adaptive algorithm, we have an *exploration* phase and an *exploitation* phase. In the exploration phase we recursively fill the Q-table by (8.2). After sufficient iterations, it seams reasonable that the approximation Q_n of Q^* will have the form as in Table 1[22]. We then estimate δ using the Q-Table and move on to the exploitation phase.

[20] Typically in practice *epsilon-decay* is used to balance "exploration" vs. "exploitation", where with probability ε we explore and exploit otherwise. Initially we set $\varepsilon = 1$ and every step we reduce ε until a minimal-exploration-probability $\varepsilon_{min} > 0$.

[21] Also known as *tabular Q-learning* since it utilizes a table where the Q-Vales are stored.

[22] Note that there are no results guaranteeing that the Q-Table will have the form of Table 1 and if the Q-Table is even correct when we stop iteration. More on this later in the next section.

The algorithm present here has no theoretical foundation, the goal is only to provide an insight on how the classic Q-Learning algorithm can be applied to the problem at hand.

We assume that we have two actions to chose from: negative drift $a_0 < 0$ or positive drift $a_1 > 0$. This corresponds to the choice between the respective maximal drifts θ_0 and θ_1 in the initial control problem (4.1) and in Definition 5.1. The first problem we encounter is that we do not have a discrete state space since each position X_t of the process at time $t \in [0, \infty)$ is a state. We can, however, aggregate the continuous state space $\mathbb{R}$ into $m+2 \in \mathbb{N}$ intervals of the form

$$(-\infty, y_1),\ [y_1, y_2), \ldots, [y_m, y_{m+1}),\ [y_{m+1}, \infty), \tag{8.3}$$

with $y_k \in \mathbb{R}$. We are mainly interested in the aggregated state-intervals $s_k := [y_k, y_{k+1})$ since we expect y_1 and y_n to be chosen in a way that if the process is in $(-\infty, y_1) =: s_0$ or $[y_m, \infty) =: s_{m+1}$ the actions to be clear, (a_1 or a_0 respectively).

In this setting we formulate the Markov decision process for our problem. We consider the state space $E = \{s_0, s_1, \ldots, s_m, s_{m+1}\}$, action space $A = \{a_0, a_1\}$ and $A(s) = A$ for all $s \in E$. We can then approximate the optimal Q^* function with (8.2) where we use

$$c(x, a) = \gamma \cdot x^2,$$

as initially given in (4.2). This choice for c is in order to compare the results of the Q-Learning Algorithm with the previous statistical results presented in Sections 4, 6 and 7. Let

$$s : \mathbb{R} \to E, \qquad x \mapsto \begin{cases} s_0, & x \in s_0, \\ s_1, & x \in s_1, \\ \quad\vdots & \\ s_m, & x \in s_m, \\ s_{m+1}, & x \in s_{m+1}, \end{cases}$$

be the *state-transformation* mapping that assigns the position of our system to an interval, i.e. a state. The dynamics of the exploration stage are then defined recursively by

$$\begin{aligned} \tau_k &= \inf\left\{ t \geq \tau_{k-1} : \tilde{X}_t \notin s\big(\tilde{X}_{\tau_{k-1}}\big) \right\}, \\ j_k &= \arg\min_j Q\big(s(\tilde{X}_{\tau_k}, a_j)\big), \\ \tilde{X}_t &= X_t^{(-1)^{j_k+1}\infty, \tau_k}\big(\tilde{X}_{\tau_k}\big), \end{aligned}$$

where $t \in [\tau_k, \tau_{k+1})$, $k \in \mathbb{N}_0$, $\tau_0 = 0$, and $\tilde{X}_0 = 0$. In each time-step of this stage we update the Q-values via (8.2). Since Q_n converges to the optimal Q^* value, we assume that for an n_0 we have the following for $n \geq n_0$: There exists a $k \in \{0, 1, 2, \ldots, m, m+1\}$ such that

$$\begin{cases} Q(s(x), a_0) < Q(s(x), a_1), & \text{for } x \in s_{\ell+1} \text{ for all } \ell \leq k, \\ Q(s(x), a_0) > Q(s(x), a_1), & \text{for } x \in s_\ell \text{ for all } \ell > k, \end{cases} \tag{8.4}$$

i.e. after a sufficient number of time-steps, the Q-table should have the form as in Table 1. Then we assume that the optimal switching point δ should lie in the interval $s_k \cup s_{k+1}$[23]. A

States	s_0	s_1	$\cdots$	s_m	s_{m+1}
	$(-\infty, y_1)$	$[y_1, y_2)$	$\cdots$	$[y_m, y_{m+1})$	$[y_{m+1}, \infty)$

	$Q(s_i, a_0)$		$Q(s_i, a_1)$
s_{m+1}	$q_{m+1,0}$	$<$	$q_{m+1,1}$
s_m	$q_{m,0}$	$<$	$q_{m,1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_{k+1}	$q_{k+1,0}$	$<$	$q_{k+1,1}$
s_k	$q_{k,0}$	$>$	$q_{k,1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_1	$q_{1,0}$	$>$	$q_{1,1}$
s_0	$q_{0,0}$	$>$	$q_{0,1}$

Table 1: The optimal switching point δ should be in the interval $s_k \cup s_{k+1}$.

natural estimate of δ is then the mean of the upper/lower bounds of $s_k \cup s_{k+1}$[24], i.e.

$$\hat{\delta} := \frac{\sup(s_{k+1}) + \inf(s_k)}{2}$$

for $k \in [1 : m]$. We also assume that the choice of y_1 and y_{m+1} to be such that $y_1 \leq \delta \leq y_{m+1}$. Naturally this cannot always be guaranteed since δ is unknown, but the algorithm can be adapted to decrease or increase the lower or upper boundary, such that the "switch" (in Table 1) occurs in between the state-intervals $s_1, \ldots, s_m$.

Remark 8.8. An important note is, like mentioned before, that there are no theoretical foundations to the presented method of finding an estimate of δ using Q-Learning. This is just an idea how to use the basic Q-Learning algorithm of the 1990s to find a control of the system X_t with unknown drift coefficients θ_0 and θ_1. More specifically, the assumption (8.4) and that the Q-Table has the form of Table 1 have no theoretical foundation and are only assumptions that seam reasonable. We present some important limitations and problems that can occur the the section 8.2.2.

Now the dynamics of the exploitation phase is given by

$$\tilde{X}_t = X_t^{\hat{\delta},s}(\tilde{X}_s)$$

with $s = n_0$. Here, n_0 denotes the time where the Q-table has reached the form as described above (as in Table 1) and we stop the exploration phase.

If the cost of the exploitation phase is not acceptable, we can repeat the exploration phase, where we choose $\tilde{y}_1 := \inf s_k$ and $\tilde{y}_{m+1} := \sup s_{k+1}$ and have the new refined state space

$$(-\infty, \tilde{y_1}),\ [\tilde{y}_1, \tilde{y}_2), \ldots, [\tilde{y}_m, \tilde{y}_{m+1}),\ [\tilde{y}_{m+1}, \infty).$$

The criteria for when to preform a refinement is to be set depending on the application. Since the optimal cost is unknown, there is no fixed measurement for an acceptable exploration cost.

[23] This is a assumption we make and should not be taken as a fact. It could happen that we do not iteration enough and the true δ lies in a different state-interval.

[24] Note that by construction $\inf(s_k \cup s_{k+1}) = \inf(s_k)$ and $\sup(s_k \cup s_{k+1}) = \sup(s_{k+1})$.

8.2.1 Implementation and results

We implement this simple Q-Learning algorithm in the SimpleQ class. As before we make use of the simulation class for all simulations of the process $X_t = X_t^b$ defined as in (4.1).

For the implementation we fix a time-horizon $T > 0$ and the number of iteration iter ($\in \mathbb{N}$) we wish to preform. This is a difference to the other implementations, since here we reset the systems position back to its initial state X_0 after reaching the time-horizon T and then start the next iteration starting at X_0 again. In the other implementations we instead only fix a horizon T and do not reset the environment. The reason for this is due to convergence of the Q-Table (c.f. Table 1) and will be discussed later.

We begin by initializing the class with an environment env, the number of states (i.e. intervals) num_of_states between $-\infty$ and ∞ and the lower (lowerb) and upper (upperb) bounds of the state-intervals (8.3).

```
class SimpleQ:

    def __init__(self, ita, env, num_of_states, upperb, lowerb):
        self.ita = ita
        self.env = env
        self.num_of_states = num_of_states
        self.upperb = upperb
        self.lowerb = lowerb
```

The main method is the following function, which implements the recursion (8.2) for the Q-Values. Here for the sake of brevity, we only show the main lines of the code. The parts that are left out can be found in the Appendix B.4.

```
def q_learn(self, start, Q):
    # initialize learn-rate and gamma (see Appendix for full code).
      [...]

        Q_old = Q
        for i in progressbar(range(self.ita)):
            # we initialize the first state of the episode
            current_state = self.get_state(self.num_of_states, start)

            while not self.env.done:
                # The environment runs the chosen action and returns
                # the next state, a reward and true if the episode is ended.

                action = self.action(current_state, exploration_proba, Q)
                X, reward, _ = self.env.step(action, True)
                next_state = self.get_state(self.num_of_states, X)

                Q[current_state, action] = ((1 - lr) * Q[current_state, action]
                             + lr * (reward + gamma * min(Q[next_state, :])))

                current_state = next_state

            # We update the exploration proba using exponential decay formula
            t_reward = self.env.total_cost
            self.env.reset(start)
            exploration_proba = max(min_exploration_proba, np.exp(-
                exploration_decreasing_decay * i))
      [...]
```

The function get_state, called in line 16, returns the index k of the state-interval $s_k = [y_k, y_{k+1})$ for which $\tilde{X}_t \in s_k$ holds at time t. Line 18 is where the Q-values are updated (c.f. (8.2)) and the main part of this method.

Since the iteration (8.2) is preformed over all state-action pairs (x, a), we have to ensure that the agent "visits" all states frequently. This means that having too many states would lead to longer training times. Because of this we set the size of the state space to 5, i.e. 5 time intervals (7 including the two boundary intervals). We set the learn-rate to $lr = 0.1$ and discount factor to $\gamma = 0.5$. Instead of choosing $\gamma = 0.99$ we choose a smaller value, since we have a small state space consisting of 5 intervals (7 including $(-\infty, -2)$ and $(2, \infty)$). We list the values used in Table 2.

lr	γ	num_of_states	dt	upperb	lowerb	iter	T
0.01	0.5	5	0.01	2	-2	5000	100

Table 2: Values for the Q-Learning algorithm.

As for the previous implementations we set $\theta_0 = -6$ and $\theta_1 = 1$. Then after 5000 iterations (a total of 5×10^7 time-steps) the Q-Table (depicted in Table 3) has the desired form of Table 1.

States	s_0	s_1	$\cdots$	s_5	s_6
	$(-\infty, -2)$	$[-2, -1.2)$	$\cdots$	$[1.2, 2)$	$[2, \infty)$

	$Q(s_i, a_0)$		$Q(s_i, a_1)$
s_6	3.40×10^{-3}	$<$	1.24×10^{-2}
s_5	1.12×10^{-3}	$<$	3.32×10^{-3}
s_4	8.75×10^{-4}	$<$	1.02×10^{-3}
s_3	1.49×10^{-3}	$>$	1.20×10^{-3}
s_2	7.79×10^{-2}	$>$	5.38×10^{-3}
s_1	4.68×10^{0}	$>$	4.13×10^{-2}
s_0	9.12×10^{1}	$>$	1.25×10^{0}

Table 3: The optimal switching point δ should be in the interval $s_3 \cup s_4$.

In Table 3 we can also see that the learner spent more time in s_0 and s_1 than the other state-intervals. This makes sense since the "downward" drift $\theta_0 = -6$ is grater than the "upward" drift $\theta_1 = 1$.

Now, either using the Q-Table or the switching point

$$\hat{\delta} = \frac{\sup(s_4) + \inf(s_3)}{2} = 0.4 \approx 0.41\overline{6} = \delta,$$

we obtain an already good approximation for the optimal switching point δ. Note that the cost of using the estimate $\hat{\delta}$ instead of the optimal δ increases the long-term cost by ε^2, where $\varepsilon := |\hat{\delta} - \delta|$ is the error of the estimate $\hat{\delta}$.

We can now refine the intervals or stop the approximation with the Q-leaning algorithm and continue the exploitation with the estimate $\hat{\delta}$. Here we do not continue with the refinement of the intervals due to convergence issues. In practice it becomes apparent that when the intervals

are too small, the Q-learning algorithm has difficulties gathering enough information in each state. This is due to the process spending most of the time "above" or "below" the point δ and thus, if the intervals are too small, the learner will mostly observe the process in the state-intervals s_0 or s_{m+1} and rarely spend time in between the two.

There is also no (known) method of finding a upper bound for the regret, as in the previously discussed algorithms.

8.2.2 Limitations of using Q-Learning

One of the biggest downsides of the Q-Learning approach is that the Q-Learner requires a lot of data in order to learn. Even when experimenting with various different values for the learn-rate and discounting factor, it was not possible to achieve stable results for fewer than 5000 iterations, i.e., 50 million time-steps. Depending on the step-size dt, this can take a considerable amount of time. For $dt = 0.01$, it takes about 4 minutes to train on a Mac Studio equipped with an M2 Ultra , whereas the run-times of the statistical methods in the same setting $T = 100$ and $dt = 0.01$ are within milliseconds.

Another limitation is the inability of learning "on the fly". We cannot keep the system running while it is exploring and learning but have to reset it to the initial starting point X_0. We also note that the choice of the step-size dt is important. If dt is too small, then the cost function x^2 is not a good choice since the cost of controlling with one of the two actions will be indistinguishable to the learner.

An open question is the expected convergence time of the Q-table and of course the correctness of the table once it has the form as in Table 1. It is known that the Q-values will converge to the optimal Q-function Q^*, but the number of iterations n that are needed such that $|Q_n(a, x) - Q^*(a, x)| < \varepsilon$ holds for any $\varepsilon > 0$ and all actions $a \in \{a_0, a_1\}$ and states $x \in E$ is unknown.

The big advantage of this method is that we hardly need any theory. We can define an algorithm using the general Q-Learning equation given by (8.2) and with sufficient steps/data we have a good control of our system. There is still potential for optimization. One way of optimizing this method is to reduce the compute time and to increase the "data-throughput" by means of parallel processing. The main idea is to only update the values of the Q-function if there is an improvement. Since in practice it takes a considerable amount of iterations for the cost of control to decrease, we can run multiple iterations in parallel and synchronize the Q-values after one or more processes returns an improvement. In the case where more than one process returns an improvement we can either take the best one, i.e., the one that has the least iteration cost, or combine them in a suitable way. With long time horizons, this could reduce the training time and increase the amount of data the agent can observe.

We have experimented with different learning rates and discounting factors but have not gone into any rigorous optimization since the previous algorithms using statistics yielded more promising results and since the next method, Deep-Q-Learning, seemed to better fit the setting of having a continuous state space.

8.3 Deep-Q-Learning

As we have previously seen, we had to preform state-aggregation in order to utilize Q-Learning. Since the regular Q-Learning algorithm does not work very well, if we have too many state-intervals or if they are chosen too small, we present a method where instead of directly computing

these we approximate the Q-values using function approximation, more specifically by using neural-networks.

The idea is to replace the quantities $Q(x,a)$ of (8.2) with suitable functions

$$Q_w : E \to \mathbb{R}^{|A|}$$

with $w \in U \subset \mathbb{R}^d$ and where $|A|$ denotes the number of actions. The goal then is to find $w \in U$ that approximates $Q^*(x,a)$ as accurately as possible over $Q_w(X)_a$. Therefore it is advisable to choose the family Q_w in such a way that $U = \mathbb{R}^d$ and such that $Q_w(x)$ is differentiable w.r.t. w for all $x \in E$. By interpreting the Q-Learning algorithm as a gradient descent algorithm, it can be generalized for the case of function approximation as follows:

We begin by choosing an arbitrary w. Then at each point in time t, a decision a_t is made according to a policy π_t based on which a reward R_{t+1} and a new state X_{t+1} are obtained. Then, the parameter w is updated via

$$\begin{aligned} w &\leftarrow w - \frac{\alpha_t}{2} \nabla_z|_{z=w} \left(R_{t+1} + \gamma \max_a Q_z\big(X_{t+1}\big)_a - Q_z\big((X_s)_{a_t}\big) \right)^2 \\ &= w + \left(R_{t+1} + \gamma \max_a Q_z\big(X_{t+1}\big)_a - Q_w\big(X_t\big)_{a_t} \right) \nabla_w Q_w\big((X_s)_{a_t}\big). \end{aligned}$$

The direct approach would be to choose the class Q_w where $w \in \mathbb{R}^d$ as neural networks and apply the previously presented algorithm. A few problems arise with the direct approach. The usual optimization algorithms for the parameter w (usually stochastic gradient descent) only work well if i.i.d. (or at least approximately i.i.d.) data are trained. In practice however, successive states are usually strongly correlated. This is circumvented by extending the state space with a *replay buffer* which stores the last n steps. Then, instead of training on the current observation, a *mini-batch* is randomly sampled from the replay buffer in each time-step on which training occurs. The size of the replay buffer n varies from problem to problem. For our problem we use $n \in [10000 : 50000]$.

Another problem arises from the nonlinearity of the approximation of Q. In every iteration we update w such that $Q_w(X_t)_{a_t}$ better approximates the target value

$$R_{t+1} + \gamma \max_a Q_w(X_{t+1})_a.$$

Here instabilities in the resulting dynamics can occur since the target value itself changes in a nonlinear manner. This problem can be circumvented by introducing an auxiliary network $Q_{\widehat{w}}$. The target value is then set as

$$R_{t+1} + \gamma \max_a Q_{\widehat{w}}(X_{t+1})_a.$$

and we update $\hat{w} \leftarrow w$ and the target value from time to time. The resulting algorithm is the standard *Deep-Q-Learning* algorithm.

The Deep-Q-Learning algorithm can be summarized as follows (c.f. [Lap18] Ch. 6.4) :

1. Initialize the parameters for $Q(s,a)$ and $Q(s',a')$ with random weights, set the exploration probability $\varepsilon = 1.0$ and initialize an empty replay buffer.

2. Choose with probability ε an action randomly or choose the action $a = \arg\min_a Q(s,a)$.

3. Evaluate the action a in the environment and receive the cost/reward of taking the action a and the next state s'.

4. Add the transition (s, a, r, s') to the replay buffer.
5. Then (randomly) sample a *mini-batch* of transitions from the replay buffer.
6. For all transitions in the mini-batch, compute the target value $y = r$ if the episode ends in this step. Otherwise compute the target value $y = c + \gamma \min_{a'} \widehat{Q}(s', a')$.
7. Compute the loss $\mathcal{L} = \big(Q(s, a) - y\big)^2$.
8. Update $Q(s, a)$ using stochastic gradient descent in order to minimize the loss $\mathcal{L}$ with regard to the model parameters.
9. After every N steps, copy the weights of Q to the auxiliary network $\widehat{Q}$.
10. Repeat starting from Step 2 until convergence occurs.

The implementation strongly relies on the examples provided in the book of Maxim Lapin [Lap18] and is presented in the Appendix B.

8.3.1 Results

Here we present a few results obtained by using the Deep-Q-Learning algorithm for controlling the drift of a Brownian motion. For the first example we set the drift parameters, as in the other implementations, to $\theta_0 = -6$ and $\theta_1 = 1$. For $T = 1000$, $dt = 0.001$ and $frameskip = 10$, the process using Deep-Q-Learning to control the drift can be seen in Figure 9. We introduce the parameter $frameskip$ to enhance stability in training. If the step size dt is too small, then the learner will have difficulties differentiating the cost of taking action a_0 or action a_1.

In Figure 9 we see how the agent is "exploring" before figuring out how to control the system. It first dips down before overshooting the origin. This can be explained by the drift coefficients $\theta_0 = -6$ and $\theta_1 = 1$. If we choose randomly between the two, we will tend to move towards $-\infty$ more than towards ∞. Then the agent learned that choosing the drift θ_1 was the better option, hence we overshoot the origin and tend towards $+\infty$. After this S-type curve, the agent learns how to "balance" the usage of the two actions and we see how the system carefully moves in a controlled manner towards the origin.

This S-curve does not always have to occur, as seen in Figure 10. In Figure 10 we have the same settings as in Figure 9 except for setting $frameskip = 1$. The agent here has 10 times more decision-points than in Figure 9. Here, in Figure 10 we also see how instabilities can occur. We could conclude after $t = 200$ time steps (i.e. after $200 \cdot dt$ time steps) the agent has learned to control the system. However, as can be seen in the Figure 10, some fluctuations up or down still occur (as seen in Figure 10 for $t \in [400, 600]$). It would be good to know why these fluctuation happen and how to prevent these, this is however one of the downsides of Deep-Reinforcement learning. These types of stability or convergence results are very scarce.

For different drift values θ_0 and θ_1 the agent might also need more (or fewer) time-steps in order to learn and control the system. This can be seen in Figure 11. Here we set $\theta_0 = -0.2$ and $\theta_1 = 0.9$, and do not change any of the agents parameters used in Figure 9 except for increasing the time-horizon to $T = 3000$. Then by Theorem 4.3 the optimal switching point is $\delta = \approx 6.56$. In Figure 11 we see that the agent takes longer to control the system. At time $t = 1000$, which is the terminal time in Figure 9, we see that in Figure 11 the systems position is still far from the origin, i.e. nowhere near of being in the optimal range.

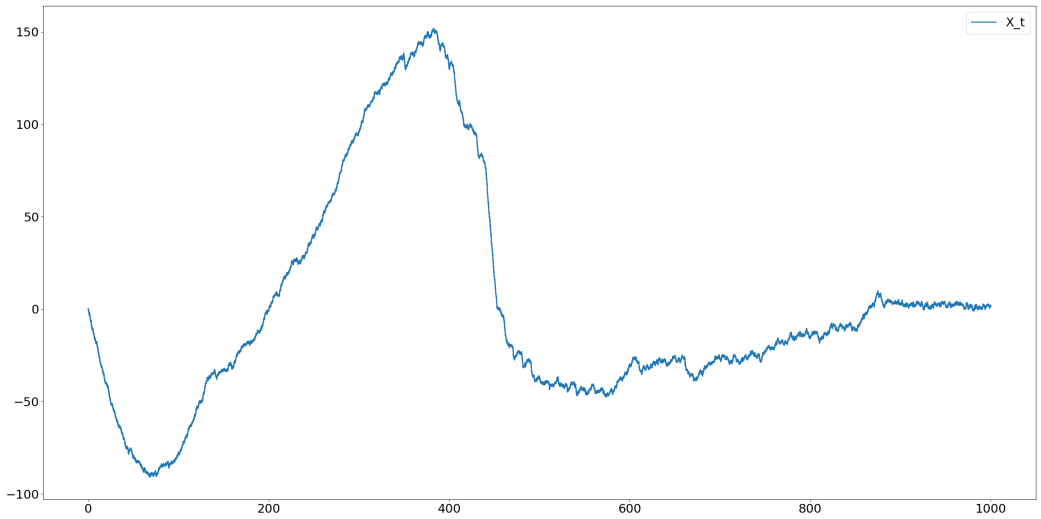

Figure 9: A simulation of the Deep-Q-learning algorithm with time horizon $T = 1000$ and step-size $dt = 0.001$ and *frameskip* $= 10$. This means that the agent has a step-size $dt = 0.01$ whereas the process is simulated with a step-size of $dt = 0.001$. We chose $\theta_0 = -6$ and $\theta_1 = 1$. Here we see that the agent has learned to control the process quite well after $t = 800$.

8.3.2 Limitations of using Deep-Q-Learning

The Deep-Q-Learning offers good results once the agent has been trained using significantly fewer time-steps than the algorithm using classic Q-Learning (50 times less in our experiments). Both reinforcement learning algorithms have very high exploration/learning costs. One needs a lot of iterations, i.e., data, and the other (Deep-Q) shoots towards $-\infty$ or ∞ depending on the drifts θ_0 and θ_1 before correcting its course. In our setting this does not matter since we considered the long term expected cost where what happens in the beginning does not matter in the long run. In practice though, these high initial costs can be detrimental. It could happen that the system moves to an unrecoverable state. For example, if the car drives off the road and crashes while exploring, we cannot recover. Another example is the control of the number of species in a population w.r.t. food. If we are given two actions

$$a_0 \mathrel{\hat{=}} \textit{“give no food”},$$
$$a_1 \mathrel{\hat{=}} \textit{“give max amount of food”},$$

then the population will decrease if action a_0 is taken, and vise versa increase with action a_1. The cost is then the (quadratic) deviation from the target population number. Note that we do not know how fast the population reacts to these two actions, i.e., whether the population is very sensitive to not being fed or how fast it reproduces or recovers given that there is sufficient food. This corresponds to the setting where we know that there exists two values for the drift, θ_0 and θ_1, however, we do not know their values. If we now use the Deep-Q-Learning algorithm to “learn on the fly” the control to keep the population at its target value, it could happen that while learning and exploring the effects of each action, we starve the population to a point

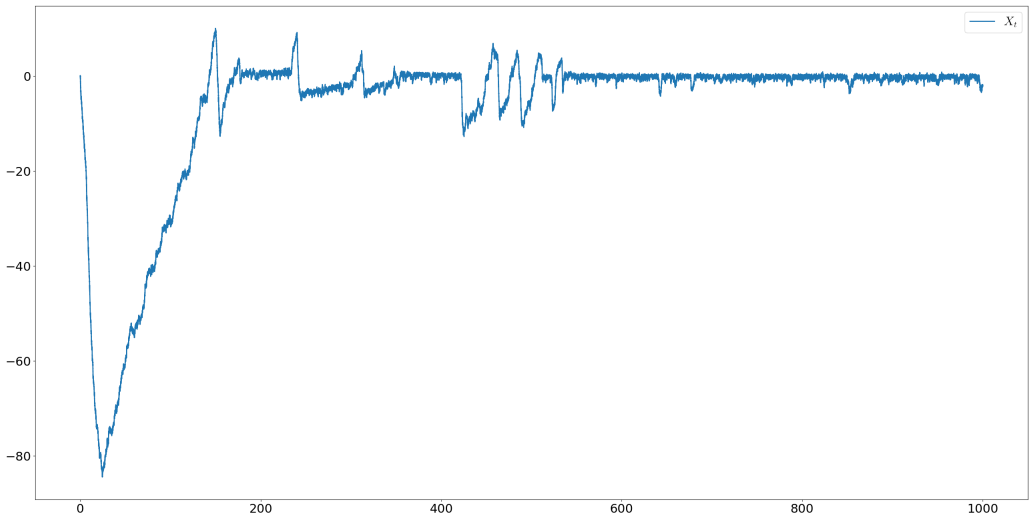

Figure 10: A simulation of the Deep-Q-learning algorithm with time horizon $T = 1000$ and step-size $dt = 0.001$ and no *frameskip*. Here we see that the agent has learned to control the process quite well after $t = 200$ and the agent did not explore "steering up" as it did in other cases.

Figure 11: A simulation of the Deep-Q-learning algorithm with time horizon $T = 1000$ and step-size $dt = 0.01$ and no *frameskip*. Here we the actual drift values $\theta_0 = -0.2$ and $\theta_1 = 0.9$ differ from the previous examples. The optimal switching point in this example is $\delta \approx 6.56$.

where it cannot recover. With other words, we have starved it to extinction. If we look at Figure 9 and suppose the y-axis denotes the deviation of the target population and the x-axis the time, then if we begin with a target population size smaller than 80, the population would die out.

One possibility is to limit the agent's ability to explore. If the system's position is too large in terms of its absolute value, then we force the agent to choose the respective drift θ_i. This solution would only be advisable if we can ensure that the optimal switching point δ lies in between the set bounds. If not, it could happen that the agent will never get the chance to explore and optimally control the system.

Another big limitation is that we do not know how well the reinforcement learning methods work for different (unknown to the agent) drift coefficients θ_0 and θ_1. With different experiments for various values for θ_0 and θ_1 the reinforcement learning methods worked quite well as can be seen in Figures 9 and 10, however, we are not able to say that without finetuning of the reinforcement learning model that we can expect the same results as seen in these experiments.

Here, like previously, the results rely solely on empirical tests. This is of course a limitation because we would like to ensure the convergence of the system and find bounds for the regret as we did in Theorem 6.5. However such results on bounds of the expected regret do not really exist for Deep-Q-Learning. The Deep-Q-Learning serves more as a type of "black-box" which can be applied to many different problems, for example our problem: "Controlling the drift of a Brownian motion".

9 Conclusion and Outlook

After laying the groundwork in the first part of this thesis (sections 2 and 3) and giving an explicit solution to the control problem if we have all information at hand, in the second and third parts we presented four methods for controlling the drift of a Brownian motion, two algorithms using statistical methods and two using reinforcement learning methods. All four methods succeed in providing a control $b_z(x)$ for the system's position $X_t^{z,s}$ w.r.t. the time $t \in [0, \infty)$ given by the stochastic differential equation

$$dX_t^{z,s}(\mu) = b\big(X_t^{z,s}(\mu)\big)dt + dW_t, \quad X_s^{z,s}(\mu) \sim \mu$$

(c.f. Definition 5.1), but their executions, efficiency (if applicable), stability and adaptability differ significantly. In the second part of this thesis we used statistics to estimate the optimal switching point δ, which we then used to provide an estimate of the optimal control function $b_\delta(x)$. We recall that the optimal control was a "bang-bang" type of control where we switch between using drift θ_0 or θ_1 whenever the system's position lies above or below the optimal switching point δ respectively.

Of the two statistical methods, the explore-first algorithm provides very good estimations of the optimal switching point and as of now, the underlying theory is more mature. Not only have we shown that in the exploration stage the estimation of the optimal switching point δ is consistent, but also that the regret (defined as a function of T) is proportional to $\sqrt{T}$ for time-horizon $T > 0$. On the other hand, the adaptive algorithm provides promising improvements compared to the explore-first algorithm, however, this algorithm is still in a developing state. We can speculate with numeric results that the regret is proportional to $\ln(T)$ for time-horizon $T > 0$, however, the proof for this is left as an outlook for further work on this topic. We were also not able to let the barriers completely converge towards the estimated switching point $\hat{\delta}_k$ (c.f. section 7.2) but had to stop the radii K_k from decreasing after falling below an ε, i.e., after $K_k < \varepsilon$ for an $\varepsilon > 0$. The idea with the adaptive algorithm is to have barriers $K_k + \hat{\delta}_k$ and $-K_k + \hat{\delta}_k$ that dynamically adapt to an estimate $\hat{\delta}$ of the optimal switching point δ. Since the radii are decreasing, the intuition is that for $k \to \infty$ the barriers converge towards an estimate of δ. Ensuring convergence for $k \to \infty$ is, however, not that easy and fixing this remains as an open problem for future work.

We also presented two reinforcement learning methods in the third part of this thesis, a classic Q-learning algorithm and a Deep-Q learning algorithm. The "classic" Q-Learning algorithm is different from the other three presented algorithms since the implementation is based off of iteration over a time-horizon while training/exploring. As apposed to the other three algorithms, this method does not allow for "on the spot" learning. The classical Q-Learning algorithm would for example be suited in the setting of a self-driving car where in the development stage we can train a model for controlling the side to side movement of the car. This is to be understood as having a "test road" of fixed length where we let the car (the system X_t) drive and the agent learn from the data points generated by these iterations. After the agent has found an acceptable control, we can then deploy the car to the market and use the trained model from then on. Conversely deploying the car with the untrained model and having the agent learn while users use the "self-driving" feature would most likely lead to failure.

The second method, Deep-Q-Learning, is fairly modern. Deep learning methods were first proposed in the early 2000s and their first industrial applications began appearing around 2010. Up until then one had to perform state aggregation in order to use "classic" reinforcement

learning methods. Deep-Q-Learning provided a break though for reinforcement leaning applications because it does not rely on state aggregation but instead uses function approximation with the help of neural networks. This is better suited for very large or even continuous state spaces like in our case. As seen in the experiments in section 8.3 the advantage of using the Deep-Q-Learning method is that it learns to control the system quicker (with fewer time-steps) and can be deployed while training. Remember that we had to first train the classic Q-learning algorithm by iterating over a fixed time-horizon and then resetting the system's position after each iteration. Another difference between the two methods is that if the Q-table converges correctly, then using the Q-table as the control function yields lower total cost (in the sense of 4.2) than the Deep-Q-learning method. This is plausible since in each step we perform function approximation in the Deep-Q-Learning method, whereas the Q-table would already closely resemble the optimal control function. The big assumption with the classic Q-learning is, however, that the Q-table converges correctly. While testing the algorithm, it often occurs that after 5000 iteration, i.e., 500.000 time-steps, the Q-table is still not correct. Since it is more common to use Deep-Q-Learning for these types of applications with continuous state spaces, we did not further look into optimizing the classic Q-learning algorithm.

While fine tuning and experimenting with reinforcement learning algorithms, a key part is the training/execution time. For the agent (Deep-Q-Learning) to learn to control the system, it took in general 30 minutes to execute the method for 100.000 time-steps, replay buffer size 10.000 and a mini-batch size of 1024 on a Mac Studio with an M2 Ultra chip (60 core GPU and 64gb of ram). Increasing the time-steps of course also increases the run-time, however, the statistical methods perform better and run in the milliseconds in general. The Deep-Q-Learning method can still be optimized to run faster, for example by storing the replay buffer on the GPU utilizing the unified memory architecture of the M2 Ultra chip, but these optimization are left for future work.

In summary, we presented four methods and ideas for controlling the drift of a Brownian motion in this thesis. Some of them work better in this setting than others, but all have their advantages and disadvantages, w.r.t. adaptability, training time, stability, consistency, etc. There are also still some open questions, for example the regret of the adaptive algorithm, or controlling the drift in higher dimensional Brownian motions.

References

[Als10] Gerold Alsmeyer. "Renewal, recurrence and regeneration". In: *Preprint version is available at http://www.math.uni-muenster.de/statistik/alsmeyer* (2010).

[AT90] G. K. Agrafiotis and M. Tsoukalas. "Excess-Time Renewal Theory with Applications". In: *The Journal of the Operational Research Society* 41.1 (1990), pp. 69–82. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/2582940 (visited on 02/09/2024).

[Bel57] Richard Bellman. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684. ISSN: 00959057, 19435274. URL: http://www.jstor.org/stable/24900506 (visited on 02/06/2024).

[BS96] A.N. Borodin and P. Salminen. *Handbook of Brownian Motion: Facts and Formulae*. Bioelectrochemistry, Principles and Practice. Birkhäuser Verlag, 1996. ISBN: 9783764354633. URL: https://books.google.de/books?id=c0S7Cpa8pHQC.

[BSW80] Václav E Beneš, Larry A Shepp, and Hans S Witsenhausen. "Some solvable stochastic control problemst". In: *Stochastics: An International Journal of Probability and Stochastic Processes* 4.1 (1980), pp. 39–83.

[BWng] Rabi Bhattacharya and Edward C. Waymire. "A Basic Course in Probability Theory". In: *Analysis* (forthcoming). DOI: 10.1007/978-0-387-71939-9.

[CDR08] Patrick Cattiaux, Paolo Dai Pra, and Sylvie Roelly. "A Constructive Approach to a Class of Ergodic HJB Equations with Unbounded and Nonsmooth Cost". In: *SIAM J. Control and Optimization* 47 (Jan. 2008), pp. 2598–2615. DOI: 10.1137/070698634.

[E18] Luis H. R. Alvarez E. *A Class of Solvable Stationary Singular Stochastic Control Problems*. 2018. arXiv: 1803.03464 [math.OC].

[Fre12] D. Freedman. *Brownian Motion and Diffusion*. Springer New York, 2012. ISBN: 9781461565741. URL: https://books.google.de/books?id=uxQGCAAAQBAJ.

[Hal70] W. J. Hall. "On Wald's Equations in Continuous Time". In: *Journal of Applied Probability* 7.1 (1970), pp. 59–68. ISSN: 00219002. URL: http://www.jstor.org/stable/3212148 (visited on 02/23/2024).

[Hen+19] Alexandru Hening, Dang H. Nguyen, Sergiu C. Ungureanu, and Tak Kwong Wong. "Asymptotic harvesting of populations in random environments". In: *Journal of Mathematical Biology* 78.1 (2019), pp. 293–329. DOI: 10.1007/s00285-018-1275-1. URL: https://doi.org/10.1007/s00285-018-1275-1.

[Ibe13] Oliver C. Ibe. "3 - Introduction to Markov Processes". In: *Markov Processes for Stochastic Modeling (Second Edition)*. Ed. by Oliver C. Ibe. Second Edition. Oxford: Elsevier, 2013, pp. 49–57. ISBN: 978-0-12-407795-9. DOI: https://doi.org/10.1016/B978-0-12-407795-9.00003-7. URL: https://www.sciencedirect.com/science/article/pii/B9780124077959000037.

[Kal02] O. Kallenberg. *Foundations of Modern Probability*. Probability and Its Applications. Springer New York, 2002. ISBN: 9780387953137. URL: https://books.google.de/books?id=L6fhXh13OyMC.

[Kal21] O. Kallenberg. *Foundations of Modern Probability*. Probability Theory and Stochastic Modelling. Springer International Publishing, 2021. ISBN: 9783030618711. URL: https://books.google.de/books?id=LNwaEAAAQBAJ.

[KS91] I. Karatzas and S. Shreve. *Brownian Motion and Stochastic Calculus*. Graduate Texts in Mathematics (113) (Book 113). Springer New York, 1991. ISBN: 9780387976556. URL: https://books.google.de/books?id=ATNy_Zg3PSsC.

[Lap18] M. Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing, 2018. ISBN: 9781788839303. URL: https://books.google.de/books?id=xKdhDwAAQBAJ.

[Lin83] Torgny Lindvall. "On Coupling of Diffusion Processes". In: *Journal of Applied Probability* 20.1 (1983), pp. 82–93. ISSN: 00219002. URL: http://www.jstor.org/stable/3213722 (visited on 02/06/2024).

[LP20] Antoine Lejay and Paolo Pigato. "Maximum likelihood drift estimation for a threshold diffusion". In: *Scandinavian Journal of Statistics* 47.3 (2020), pp. 609–637. DOI: https://doi.org/10.1111/sjos.12417. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/sjos.12417. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/sjos.12417.

[Øks98] Bernt Øksendal. *Stochastic Differential Equations*. Springer Berlin, Heidelberg, 1998. ISBN: 978-3-662-03620-4. eprint: 1905.08539 (cs.LG).

[Smi58] Walter L. Smith. "Renewal Theory and Its Ramifications". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 20.2 (1958), pp. 243–302. ISSN: 00359246. URL: http://www.jstor.org/stable/2983891 (visited on 02/09/2024).

[SP12] René L. Schilling and Lothar Partzsch. *An Introduction to Stochastic Processes*. Berlin, Boston: De Gruyter, 2012. ISBN: 9783110278989. DOI: doi:10.1515/9783110278989. URL: https://doi.org/10.1515/9783110278989.

Appendix A Collection of some useful results

Proposition A.1. *(Fubini, stochastic version) Let X_1 and X_2 be random variables defined on probability spaces $(\Omega_1, \mathcal{A}_1, P_1)$ and $(\Omega_2, \mathcal{A}_2, P_2)$. Further let $f : \Omega_1 \times \Omega_2 \to \mathbb{R}$ be measurable with*

$$\int_{\Omega_1 \times \Omega_2} |f(x,y)|\, d(x,y) < \infty.$$

Then

$$\begin{aligned} Ef(X_1, X_2) &= \int_{\Omega_1} \int_{\Omega_2} f(x_1, x_2) P^{X_2}(dx_2) P^{X_1}(dx_1) \\ &= \int_{\Omega_1} E\big[f(x_1, X_2)\big] P^{X_1}(dx_1) \end{aligned}$$

where P^X denotes the distribution of the random variable X.

The following Proposition is used in the proof of Lemma 3.8. This is a standard result and can be found in [Kal21].

Proposition A.2. *(Convolution) Let X_1 and X_2 be independent random defined on the probability space $(\Omega, \mathcal{A}, P)$.*

1. *The distribution of $Z = X_1 + X_2$ is given by*

$$F_Z(t) = P(X_1 + X_2 \le t) = \int_\Omega P(X_1 \le t - x_2) P^{X_2}(dx_2).$$

2. *If X_1 and X_2 have densities f_1 and f_2 respectively, then $Z = X_1 + X_2$ also has a density f given by*

$$f(t) = \int f_1(t - x) f_2(x) dx.$$

Proof. The first claim is a simple application of Fubini's Theorem A.1

$$P(Z \le t) = E[1_{\{Z \le t\}}] = \int_\Omega E[1_{\{X_1 + x_2 \le t\}}] P^{X_2}(dx_2) = \int_\Omega P(X_1 \le t - x_2) P^{X_2}(dx_2).$$

The second claim can also be easily shown using Fubini, when densities are given. □

Theorem A.3. *(Optional Sampling Theorem) Let M be a right continuous martingale. For each stopping time τ, it holds that:*

1. *If τ is bounded (i.e., there exists $N \in \mathbb{N}$ such that $\tau(\omega) < N$ for all ω), then:*

$$EM_\tau = E[M_0].$$

2. *If $P(\tau < \infty) = 1$ and $X_{\tau \wedge t}$ for $(t \in [0.\infty)$ is uniformly integrable. Then*

$$EM_\tau = E[M_0].$$

The proofs of Theorem A.3 can be found in [KS91] and [BWng].
The next Theorem is one of the main tools for constructing estimators in this thesis.

Theorem A.4. *(Wald equations) Let $X_n = \xi_1 + \cdots + \xi_n$ be an $\mathcal{F}$-random[25] walk with $E[\xi_1] = \mu < \infty$ and $\mathrm{Var}(\xi_1) = \sigma^2 < \infty$, and let τ be a stopping time on $\mathcal{F}$. Then*

1. *for $\xi_1 \in L^1$ and $E[\tau] < \infty$, it holds*

$$E\big[X_\tau\big] = \mu E\big[\tau\big].$$

2. *for $\mu = 0$, $\sigma^2 < \infty$, and $E[\tau] < \infty$, it holds*

$$E\big[X_\tau^2\big] = \sigma^2 E\big[\tau\big].$$

We refer to [Kal21] for the proof. If we consider a Brownian motion with drift $(X_t)_{t\in\mathbb{R}_+}$, given by

$$X_t = x_0 + \theta \cdot t + W_t,$$

where $t > 0$, $x_0, \theta \in \mathbb{R}$ and W is a standard Brownian motion. Then the increments

$$\xi_k := \Delta X_k = X_{k+1} - X_k$$

are i.i.d. with $E[\xi_1] = \theta$ and $\mathrm{Var}(\xi_1) = 1$ and define random walk and we can apply Wald's equations A.4 to $S_n = \xi_1 + \cdots + \xi_n$. In [Hal70] it is shown that this result (Theorem A.4) can be extended to continuous-time random walks.

The density of the the first hitting time $\Delta\tau_{2k+(1-i)}$ of a Brownian motion with drift θ_1 (and θ_0 respectively) of the level $2K$ (and $-2K$ respectively) is given by

$$f_{\Delta\tau_{2k+(1-i)}}(t) = \frac{2K}{\sqrt{2\pi t^3}} e^{-\frac{(2K-\theta_i t)^2}{2t}}. \tag{A.1}$$

Proposition A.5. *The cumulative distribution function (cdf) of $\Delta\tau_{2k+(1-i)}$ can be written in terms of the the cdf of the Normal density*

$$\begin{aligned} F_{\Delta\tau_{2k+(1-i)}}(T) &= \int_0^T f_{\Delta\tau_{2k+(1-i)}}(t)dt \\ &= 1 - \Phi\left(\frac{2K - \theta_i T}{\sqrt{T}}\right) + \exp(8K\theta_i)\left(1 - \Phi\left(\frac{2K + \theta_i T}{\sqrt{T}}\right)\right). \end{aligned} \tag{A.2}$$

Note that for the Normal cdf we have the identity $\Phi(-x) = 1 - \Phi(x)$ for all $x \in \mathbb{R}$.

Proof. We have $F_{\Delta\tau_{2k+(1-i)}}(0) = 0$. Now deriving (A.2) w.r.t. T we have

$$\frac{\partial}{\partial T}\Phi\left(-\frac{2K - \theta_i T}{\sqrt{T}}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(2K-\theta_i T)^2}{2T}} \left(-\frac{-\theta_i\sqrt{T} - (2K - \theta_i T)\frac{1}{2}T^{-\frac{1}{2}}}{T}\right), \tag{A.3}$$

$$\exp(8K\theta_i)\frac{\partial}{\partial T}\Phi\left(-\frac{2K + \theta_i T}{\sqrt{T}}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(2K-\theta_i T)^2}{2T}} \left(-\frac{\theta_i\sqrt{T} - (2K + \theta_i T)\frac{1}{2}T^{-\frac{1}{2}}}{T}\right). \tag{A.4}$$

[25] (discrete-time)

Now adding (A.3) and (A.4) yields

$$\frac{\partial}{\partial T}\Phi\left(-\frac{2K-\theta_i T}{\sqrt{T}}\right)+\exp(8K\theta_i)\frac{\partial}{\partial T}\Phi\left(-\frac{2K+\theta_i T}{\sqrt{T}}\right)=\frac{1}{\sqrt{2\pi}}e^{-\frac{(2K-\theta_i T)^2}{2T}}\cdot\frac{2K}{\sqrt{T^3}}.$$

□

For a standard Brownian motion, we have the following result.

Lemma A.6. *Let B be a standard Brownian motion and $a \in \mathbb{R}$. Then the first hitting time $\tau_a = \inf\{t \geq 0 : B_t = a\}$ is a.s. finite with Laplace transformation*

$$E\left[e^{-\alpha\tau_a}\right] = e^{-\sqrt{2\alpha}|a|},$$

for all $\alpha \in [0, \infty)$.

Since this is a standard result, we only give the idea of the proof and refer to Remark 2.8.3 from [KS91] for details. The main idea in the proof is to apply the optional sampling theorem to the martingale

$$M_t := \exp\left(\sigma B_t - \frac{\sigma^2}{2}t\right) \tag{A.5}$$

and $\tau_a \wedge n$ for some fixed constant $\sigma \neq 0$ and $n \in \mathbb{N}$. We now consider a Brownian motion $X_t = B_t + ct$ with drift $c \in \mathbb{R}$. Then $B_t = X_t - ct$ and the martingale (A.5) become

$$\begin{aligned} M_t &= \exp\left(\sigma B_t - \frac{\sigma^2}{2}t\right) \\ &= \exp\left(\sigma X_t - \left(c\sigma + \frac{\sigma^2}{2}\right)t\right) \\ &= \exp\left(\left(-c + \sqrt{c^2+2\alpha}\right)X_t - \alpha t\right) \end{aligned}$$

where $\alpha := \left(c\sigma + \frac{\sigma^2}{2}\right)$. Now following the same steps as in the proof of Lemma A.6, we get the Laplace transformation at $\alpha \in \mathbb{R}$ of $\Delta\tau_{2k}$ given by

$$E\left[e^{-\alpha\Delta\tau_\ell}\right] = \exp\left(2K\left(\theta_i - \sqrt{2\alpha+\theta_i^2}\right)\right)$$

Theorem A.7. *Let $E \neq \emptyset$ be an at most countable set, μ a probability mass function on E, Π a stochastic matrix on E. Then there is a Markov chain $(X_n)_{n\in\mathbb{N}}$ with state space E, initial distribution μ and transition matrix Π. The law P^X of $X = (X_n)_{n\in\mathbb{N}}$ on $\left(E^{\mathbb{N}}, \mathcal{P}(E)^{\mathbb{N}}\right)$ is uniquely determined by μ and Π.*

Appendix B Implementation

B.1 Simulation class

The environment consists of the following parameters:

- *delta*: Represents either the optimal or estimated switching point.
- *max_T*: The time horizon.
- *cost*: The cost function.
- *frameskip*: The number of frames or steps we skip. Default is set to 1.
- *dt*: The step size.
- *start*: The starting point of the system.
- *theta0* and *theta1*: The drift values.

```
def __init__(self, delta, max_T, cost=np.square, frameskip=1, dt=10.0 ** (-5),
    start=0.0, theta0=-1, theta1=1):
        self.delta = delta
        self.max_T = max_T
        self.cost = cost
        self.frameskip = frameskip
        self.dt = dt
        self.X = start
        self.theta0 = theta0
        self.theta1 = theta1

        self.done = False
        self.pos_cost = 0
        self.total_cost = 0
        self.t = 0
        self.trajectory = []
        self.regrets = np.zeros(int(max_T))
```

```
def step(self, action, record=True):
        X, pos_cost, trajectory, opt_trajectory = self.simulate(dt=self.dt,
                                                    cost=self.cost,
                                                    steps=self.frameskip,
                                                    start=self.X,
                                                    action=action,
                                                    record=record)

        self.X = X
        self.t += self.dt * float(self.frameskip)
        self.pos_cost = pos_cost
        self.total_cost += pos_cost
        if record:
            self.trajectory.append(trajectory)
        if self.t > self.max_T:
            self.done = True

        return X, pos_cost, self.done
```

```
def simulate(self, dt, cost, steps, start, action, record):
        effort = 0
        X = start
        pos_cost = 0
        trajectory = []

        delta = self.optimal_delta()

        for i in range(steps):
            dY = np.random.normal(loc=0.0, scale=np.sqrt(dt))
            Y = X + self._drift_action(action) * dt + dY
            if record:
                trajectory.append(Y)
            X = Y
            pos_cost = dt * cost(X) * dt

        return X, pos_cost, trajectory, opt_trajectory
```

Notes on implementation and Monte-carlo sim of expectation:
Outline:

- Introduce the Euler-M method for simulating a Brownian motion
- Calculation / Implementation of Regret for time T with cpu acceleration using parallel processing
- Usage of replay buffer and deep Q-Learning
- Environment implementation (Basically Euler method)

B.2 Explore first algorithm

```
class ExporeFirst:

    def __init__(self, K, env, record = True):
        self.K = K
        self.n = math.floor(math.sqrt(env.max_T))
        self.env = env
        self.record = record
        self.tau0 = 0
        self.tau1 = 0
        self.xData = []
```

```
    def exploration(self):
        h = 1
        tau0, tau1, tau_temp= 0,0,0
        drift = 1
        X,_,_ = self.env.step(drift, self.record)

        while h <= 2*self.n: #We iterate until we have returned from one
            barrier to the other 2n times
            if drift == 1:
                if X >= self.K:
                    tau1 += self.env.t - tau_temp
                    tau_temp = self.env.t
```

```
                    drift = 0
                    h += 1
            else:
                if X <= -1.0 * self.K:
                    drift = 1
                    tau0 += self.env.t - tau_temp
                    tau_temp = self.env.t
                    h += 1

            X,_,_ = self.env.step(drift, self.record)
            self.tau0 = tau0
            self.tau1 = tau1
            if self.env.done:
                break
        return self.env.t
```

```
    def get_delta(self):
        return (self.tau1 - self.tau0) / (self.n * 4.0 * self.K)
```

```
def exploitation(self):
        # print("Starting exploitation phase")
        done = self.env.done
        delta = self.get_delta()
        self.env.delta = delta

        while not done:
            _, _, done = self.env.step_d(record=True)
        return self.env.pos_cost
```

```
    def run(self, output):
        T = self.env.max_T

        self.exploration()
        t = self.env.t
        self.exploitation()

        trajectory = list(chain.from_iterable(self.env.trajectory))
        temp = int(round(T - t))
        if output:
            self.xData = []
            costs = []

            pool = multiprocessing.Pool()
            pool.map_async(self.expect_cost_sim, range(int(T)), callback = self
                .log_result)
            pool.close()
            pool.join()
            costs = [self._aux_mult2(a, b) for a,b in self.xData[0]]
            regrets = [self._aux_mult(a, b) for a,b in self.xData[0]]
            regrets = np.sqrt(regrets)

            t2 = np.arange(0.1, len(regrets), 1)

            fittedParameters, pcov = curve_fit(func, t2, regrets)
            modelPredictions = func(t2, *fittedParameters)

```

```
            K_line0 = [self.K]*int(t)
            K_line1 = [-1.0*self.K]*int(t)
            fig1, ax1 = plt.subplots(1, 1, figsize=(32, 16))
            fig3, ax3 = plt.subplots(1, 1, figsize=(32, 16))
            ax1.plot(np.linspace(0., int(T), len(trajectory)), trajectory, lw
                =2, label="$X_t$")
            ax1.plot(np.linspace(0., int(t), int(t)), K_line0, color='r', lw=3,
                 label="$K=$"+str(self.K))
            ax1.plot(np.linspace(0., int(t), int(t)), K_line1, color='r', lw=3)
            if temp >= 0:
                d_line = [self.get_delta()]*temp
                ax1.plot(np.linspace(int(t), int(T), temp), d_line, color='b',
                    lw=4, label="$\hat{\delta}=$"+str(round(self.get_delta(),2))
                    )
            ax3.plot(np.linspace(0., len(regrets), len(regrets)), (regrets), lw
                =2, label="regret")
            ax3.plot(np.linspace(0., len(regrets), len(modelPredictions)), (
                modelPredictions), color ='magenta' , lw=6, label="fitted r")
            ax3.legend()
            ax1.legend()
            plt.rcParams['text.usetex'] = True
            plt.show()

        cost = self.eval_last_steps(trajectory, temp)
        tot_cost = self.eval_last_steps(trajectory, T)

        return cost, tot_cost, self.get_delta(), self.get_theta(0), self.
            get_theta(1), t
```

B.3 Adaptive algorithm

```
class Adaptive_A:

    def __init__(self, K, env, record=True):
        self.K = K
        self.n = math.floor(math.sqrt(env.max_T))
        self.env = env
        self.record = record
        self.xData,self.yData = [], []
```

```
def estimation(self):
        n_theta0, n_theta1, theta0, theta1 = 0,0,0,0
        d_theta0, d_theta1 = 1,1
        tau0, tau1, tau2 = 0, 0, 0
        n = 0
        k_v_light = self.K
        delta_est = 0
        drift = 1
        X0 = self.env.X
        X, _, _ = self.env.step(drift, self.record)

        while not self.env.done:

            if drift == 1:
                if X >= k_v_light + delta_est:
```

```
                    prev = k_v_light + delta_est
                      n += 1
                      X1 = self.env.X
                      n_theta1 += X1 - X0
                      X0 = X1
                      tau1 = self.env.t
                      d_theta1 += tau1 - tau0
                      tau0 = tau1
                      drift = 0
                      theta1 = n_theta1 / d_theta1

                      if theta1 != 0 and theta0 != 0:
                          delta_est = self.compute_delta(theta0, theta1)
                          if k_v_light > self.env.dt + 0.01:
                              k_v_light = 1.0 / n
                              if prev < -1.0 * k_v_light + delta_est:
                                  drift = 1
              else:
                  if X <= -1.0 * k_v_light + delta_est:
                      prev = -1.0 * k_v_light + delta_est
                      n += 1
                      X1 = self.env.X
                      n_theta0 += X1 - X0
                      X0 = X1
                      tau1 = self.env.t
                      d_theta0 += tau1 - tau0
                      tau0 = tau1
                      drift = 1
                      theta0 = n_theta0 / d_theta0
                      if theta1 != 0 and theta0 != 0:
                          delta_est = self.compute_delta(theta0, theta1)
                          if k_v_light > self.env.dt + 0.01:
                              k_v_light = 1.0 / n
                              if prev > k_v_light + delta_est:
                                  drift = 0

              X, _, _ = self.env.step(drift, self.record)

          return delta_est
```

```
  def _aux_expcost(self, i):
          self.reset()
          self.estimation()
          c,_,_ = self.env.eval_steps()
          return c

      def log_result(self, result):
      # This is called whenever pool(i) returns a result.
      # result_list is modified only by the main process, not the pool workers.
          self.xData.append(result)

      def expect_cost_sim(self, T):
          ita = 100
          exp_cost = 0
          max_T = self.env.max_T
          self.env.max_T = T
          for i in range(ita):
```

```
18
19              self.reset()
20              self.exploration_light()
21              c,u_c,_ = self.env.eval_steps()
22
23              exp_cost += c
24          exp_cost =  exp_cost / ita
25          self.env.max_T = max_T
26          return exp_cost, T
```

```
1       def run(self, output):
2           T = self.env.max_T
3
4           delta_est, d, K_listu, K_listl = self.exploration()
5           cost, u_cost, _ = self.env.eval_steps()
6           print("total␣cost␣env␣=␣", cost, u_cost, self.env.max_T)
7           t = self.env.t
8           r = 0
9
10          trajectory = list(chain.from_iterable(self.env.trajectory))
11
12          if output:
13              odelta = self.optimal_delta()
14              d_regret = list(map(lambda x: math.fabs(x - odelta), d))
15              print("starting␣Evaluation")
16              cost, u_cost,_ = self.env.eval_steps()
17              r, regrets = self.env.regret()
18
19              regrets = []
20              opt_cost = self.optimal_cost()
21
22              self.xData = []
23
24              pool = multiprocessing.Pool()
25              pool.map_async(self.expect_cost_sim, range(int(env.max_T)),
                    callback = self.log_result)
26              pool.close()
27              pool.join()
28              print('results␣=␣', self.xData)
29              costs = [self._aux_mult2(a, b) for a,b in self.xData[0]]
30              regrets = [self._aux_mult(a, b) for a,b in self.xData[0]]
31              regrets[0] = 0.1
32              regrets = np.log(regrets)
33              np.save(filename_regrets, regrets)
34
35
36              for t in progressbar(range(100)):
37                  opt_cost_t = opt_cost
38                  expect_cost = self.expect_cost_sim(100, t)
39                  r = expect_cost - opt_cost_t
40                  regrets.append(t*math.fabs(r))
41
42              t1 = np.arange(0.1, len(regrets), self.env.dt)
43              t1 = np.exp(t1)
44              t2 = np.arange(0.1, len(regrets), 1)
45              t3 = np.arange(0.1, len(d_regret), 1)
46
```

```
            self.xData = t1
            self.yData = regrets
            geneticParameters = self.generate_Initial_Parameters()
            fittedParameters, pcov = curve_fit(func, t2, regrets)
            fittedParameters2 ,_ = curve_fit(func2, t3, d_regret)
            modelPredictions = func(t2, *fittedParameters)
            print(fittedParameters)
            np.save(filename_modelPredictions, modelPredictions)

            K_line0 = [self.K] * int(t)
            K_line1 = [-1.0 * self.K] * int(t)
            fig1, ax1 = plt.subplots(1, 1, figsize=(32, 16))
            fig2, ax2 = plt.subplots(1, 1, figsize=(32, 16))
            fig3, ax3 = plt.subplots(1, 1, figsize=(32, 16))
            ax1.plot(np.linspace(0., int(T), len(trajectory)), trajectory, lw
                =1, label="$X_t$")
            ax1.plot(np.linspace(0., int(T), len(K_listu)), K_listu, color='r',
                 lw=3, label="$K_n + \hat{\delta}_n$")
            ax1.plot(np.linspace(0., int(T), len(K_listl)), K_listl, color='g',
                 lw=3, label="$K_n + \hat{\delta}_n$")
            ax1.plot(np.linspace(0., int(T), len(d)), d, lw=6, label="$\hat{\
                delta}$", color='magenta')
            ax2.plot(np.linspace(0., int(T), len(d_regret)), d_regret, lw=2,
                label="$|\delta - \hat{\delta}|$")
            ax2.plot(np.linspace(0., int(T), len(modelPredictions2)),
                modelPredictions2, lw=2, label="fitted")
            ax3.plot(np.linspace(0., len(regrets), len(regrets)), (regrets), lw
                =2, label="regret")
            ax3.plot(np.linspace(0., len(regrets), len(modelPredictions)), (
                modelPredictions), color ='magenta' , lw=6, label="fitted r")

            ax3.plot(t1, np.log(t1), lw=2, color = 'r', label="log(t)")
            ax3.set_ylim([0,10])
            ax1.legend()
            ax2.legend()
            ax3.legend()
            plt.show()

        return cost, delta_est, t, r
```

B.4 Q-Learning

B.5 Deep-Q-Learning