Acknowledgments

I would like to thank...

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 The March of the Machines

1.2 Artificial Intelligence and Finance

1.3 Structure

Chapter 2

Discrete-Time Stochastic Optimal Control

2.1 Sequential Decision Problems

In sequential decision problems, an agent interacts with an environment by selecting a series of actions in order to complete a specific task. During this interaction, the agent receives a numerical reward from the environment and his goal is to find the best strategy in order to maximize a certain measure of his performance. The environment evolves stochastically and may be influenced by the interaction with the agent, so that each action taken by the agent may influence the circumstances under which future decisions will be made. Therefore, the agent must balance his desire to obtain a large reward today by acting greedily and the opportunities that will be available in the future. While this setting appears quite simple, it is general enough to encompasses a wide range of applications in different fields. A classical example is portfolio management, where an investor must allocate his capital so as to maximize his long-term profits. Another standard example is chess, where two players successively move pieces around the chessboard to checkmate the opponent's king.
The purpose of the following sections is to introduce the notation that will be used in the rest of this work and to recall the fundamental concepts and results of the discrete-time stochastic optimal control theory, which is the standard framework to study sequential decisions problems in mathematical terms. Since our discussion will be far from being comprehensive, we refer the reader to the extensive literature on the subject, such as [13], [66], [12].

2.2 Markov Decision Processes

A sequential decision problem under uncertainty can be schematized as in Figure 2.1: at a given time t, the agent (also known as decision maker or controller) observes the state s_t of the system (also know as environment) and subsequently performs an action a_t. Following this action, the agent receives an immediate reward r_{t+1} (or incurs an immediate cost) and the system evolves to a new state according to a probability distribution which depends on the action selected by the agent. At the subsequent time $t+1$, the agent selects a new action given the new state of the system and the process repeats. This interaction can be modeled rigorously using Markov decision processes.

Definition 2.2.1 (Markov Decision Process)**.** *A Markov decision process (MDP) is a stochastic dynamical system specified by the tuple* $< \mathbb{S}, \mathbb{A}, \mathcal{P}, \mathcal{R}, \gamma >$, *where*

i) $(\mathbb{S}, \mathcal{S})$ *is a measurable space, called the state space.*
ii) $(\mathbb{A}, \mathcal{A})$ *is a meausrable space, called the action space.*
iii) $\mathcal{P} : \mathbb{S} \times \mathbb{A} \times \mathcal{S} \to \mathbb{R}$ *is a Markov transition kernel, i.e.*
  a) for every $s \in \mathbb{S}$ *and* $a \in \mathbb{A}$, $B \mapsto \mathcal{P}(s, a, B)$ *is a probability distribution over* $(\mathbb{S}, \mathcal{S})$.
  b) for every $B \in \mathcal{S}$, $(s, a) \mapsto \mathcal{P}(s, a, B)$ *is a measurable function on* $\mathbb{S} \times \mathbb{A}$.
iv) $\mathcal{R} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ *is a reward function.*
v) $\gamma \in (0, 1)$ *is a discount factor.*

Typically, the state space (and similarly the action space) will be either finite, i.e. $\mathbb{S} = \{s_1, \ldots, s_d\}$, or continuous, i.e. $\mathbb{S} \subseteq \mathbb{R}^{D_s}$. The kernel $\mathcal{P}$ describes the random evolution of the system: suppose that at time t the system is in state s and that the agent takes action a, then, regardless of the previous history of the system, the probability to find the system in a state belonging to $B \in \mathcal{S}$ at time $t+1$ is given by $\mathcal{P}(s, a, B)$, i.e.

$$\mathcal{P}(s, a, B) = \mathbb{P}\left(S_{t+1} \in B | S_t = s, A_t = a\right) \tag{2.1}$$

Following this random transition, the agent receives a stochastic reward R_{t+1}. The reward function $\mathcal{R}(s, a)$ gives the expected reward obtained when action a is taken in state s, i.e.

$$\mathcal{R}(s, a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] \tag{2.2}$$

This setting can be easily generalized to the following cases

1. The initial state of the system is a random variable $S_0 \sim \mu$.
2. The actions that an agent can select depend on the state of the system.

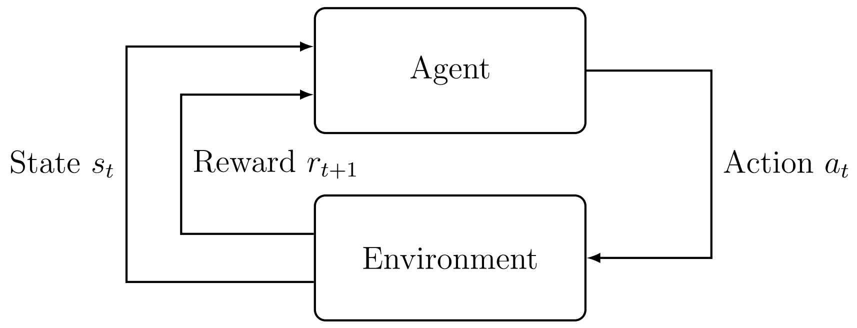

Figure 2.1: Agent-environment interaction in sequential decision problems.

2.3 Policies

At any time step, the agent selects his actions according to a certain policy.

Definition 2.3.1 (Policy). *A policy is a function $\pi : \mathbb{S} \times \mathcal{A} \to \mathbb{R}$ such that*

i) for every $s \in \mathbb{S}$, $C \mapsto \pi(s, C)$ is a probability distribution over $(\mathbb{A}, \mathcal{A})$.

ii) for every $C \in \mathcal{A}$, $s \mapsto \pi(s, C)$ is a measurable function.

Intuitively, a policy represents a stochastic mapping from the current state of the system to actions. Deterministic policies are a particular case of this general definition. We assumed that the agent's policy is stationary and only depends on the current state of the system. We might in fact consider more general policies that depends on the whole history of the system. However, as we will see, we can always find an optimal policy that depends only on the current state, so that our definition is not restrictive. A policy π and an initial state $s_0 \in \mathbb{S}$ determine a random state-action-reward sequence $\{(S_t, A_t, R_{t+1})\}_{t\geq 0}$ with values on $\mathbb{S} \times \mathbb{A} \times \mathbb{R}$ following the mechanism described above.

Definition 2.3.2 (History). *Given an initial state $s_0 \in \mathbb{S}$ and a policy π, a history (or equivalently trajectory or roll-out) of the system is a random sequence $H_\pi = \{(S_t, A_t)\}_{t\geq 0}$ with values in $\mathbb{S} \times \mathbb{A}$, defined on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$, such that for $t = 0, 1, \dots$*

$$\begin{cases} S_0 = s_0 \\ A_t \sim \pi(S_t, \cdot) \\ S_{t+1} \sim \mathcal{P}(S_t, A_t, \cdot) \end{cases}$$

we will denote by $(\mathbb{H}, \mathcal{H})$ the measurable space of all possible histories.

Moreover, we observe that

i) the state sequence $\{S_t\}_{t\geq 0}$ is a Markov process $< \mathbb{S}, \mathcal{P}_\pi >$

ii) the state-reward sequence $\{(S_t, R_t)\}_{t\geq 0}$ is a Markov reward process $< \mathbb{S}, \mathcal{P}_\pi, \mathcal{R}_\pi, \gamma >$

where we denoted

$$\begin{aligned}\mathcal{P}_\pi(s, s') &= \int_{\mathbb{A}} \pi(s, a)\mathcal{P}(s, a, s')da\\ \mathcal{R}_\pi(s) &= \int_{\mathbb{A}} \pi(s, a)\mathcal{R}(s, a)da\end{aligned}$$

In stochastic optimal control, the goal of the agent is to find a policy that maximizes a measure of the agent's long-term performance. In the next sections we discuss some objective functions that are commonly used in the infinite horizon framework.

2.4 Risk-Neutral Framework

In the risk-neutral setting, the agent is only interested in maximizing his reward, without considering the risk he needs to take on to achieve it. In an infinite horizon task, the agent's performance is typically measured either as the total discounted reward or as the average reward obtained at each time step. These two approaches are radically different both from a theoretical and an algorithmic point of view. For this reason, they will be always treated separately.

2.4.1 Discounted Reward Formulation

In the discounted reward formulation, the agent's performance is measured as the expected return obtained following a specific policy.

Definition 2.4.1 (Return). *The return is the total discounted reward obtained by the agent starting from t*

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1}$$

where $0 < \gamma < 1$ *is the discount factor.*

In some domains, such as economics, discounting can be used to represent interest earned on rewards, so that an action that generates an immediate reward will be preferred over one that generates the same reward some steps into the future. Discounting thus models the trade-off between immediate and delayed reward: if $\gamma = 0$ the agent selects his actions in a myopic way, while if $\gamma \to 1$ he acts in a far-sighted manner. There are other possilble reasons for discounting future rewards. The first is because it is mathematically convenient, as it avoids infinite returns and it solves many convergence issues. Another interpretation is that it models the uncertainty about the future, which may not be fully represented. Indeed, the discount factor could be seen as the probability that the world does not stop at a given time step. Since the return is stochastic, we consider its expected value.

Definition 2.4.2 (State-Value Function)**.** *The state-value function $V_\pi : \mathbb{S} \to \mathbb{R}$ is the expected return that can be obtained starting from a state and following policy π*

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right] \tag{2.3}$$

where the subscript in $\mathbb{E}_\pi$ indicates that all the actions are selected according to policy π. The state-value function measures how good it is for the agent to be in a given state and follow a certain policy. Similarly, we can introduce an action-value function that measures how good it is for the agent to be in a state, take a certain action and then follow the policy.

Definition 2.4.3 (Action-Value Function)**.** *The action-value function $Q_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the expected return that can be obtained starting from a state, taking an action and then following policy π*

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] \tag{2.4}$$

We have the following relationship between V_π and Q_π

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s, a) Q_\pi(s, a) da \tag{2.5}$$

Almost all reinforcement learning algorithms are designed to estimate these value functions and are typically based on the Bellman equations.

Proposition 2.4.1 (Bellman Expectation Equations)**.**

$$V_\pi(s) = \mathcal{R}_\pi(s) + \gamma T_\pi V_\pi(s) \tag{2.6}$$

$$Q_\pi(s, a) = \mathcal{R}(s, a) + \gamma T_a V_\pi(s) \tag{2.7}$$

where we denoted by T_a (resp. T_π) the transition operator for action a (resp. for policy π)

$$T_a F(s) = \mathbb{E}\left[F(S_{t+1}) | S_t = s, A_t = a\right] = \int_{\mathbb{S}} \mathcal{P}(s, a, s') F(s') ds'$$

$$T_\pi F(s) = \mathbb{E}_\pi\left[F(S_{t+1}) | S_t = s\right] = \int_{\mathbb{A}} \pi(s, a) \int_{\mathbb{S}} \mathcal{P}(s, a, s') F(s') ds' da$$

If we introduce the Bellman expection operator B_π, defined as

$$B_\pi V_\pi(s) = \mathcal{R}_\pi(s) + \gamma T_\pi V_\pi(s)$$

Then Eq. (2.6) can be written as a fixed-point equation

$$V_\pi(s) = B_\pi V_\pi(s)$$

which, under some simple assumptions on the reward functions, admits a unique solution by the contraction mapping theorem. A similar argument applies to Eq. (2.7). The agent's goal is to select a policy π_* that maximizes his expected return in all possible states. Such a policy is called *optimal.*

Definition 2.4.4 (Optimal State-Value Function)**.** *The optimal state-value function $V_* : \mathbb{S} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state*

$$V_*(s) = \sup_\pi V_\pi(s) \tag{2.8}$$

Definition 2.4.5 (Optimal Action-Value Function)**.** *The optimal action-value function $Q_* : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state and taking an action*

$$Q_*(s,a) = \sup_\pi Q_\pi(s,a) \tag{2.9}$$

The optimal value functions satisfy the following Bellman equations.

Proposition 2.4.2 (Bellman Optimality Equations)**.**

$$V_*(s) = \sup_a Q_*(s,a) = \sup_a \{\mathcal{R}(s,a) + \gamma T_a V_*(s)\} \tag{2.10}$$

$$\begin{aligned} Q_*(s,a) &= \mathcal{R}(s,a) + \gamma T_a V_*(s) \\ &= \mathcal{R}(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') \sup_{a'} Q_*(s',a') ds' \end{aligned} \tag{2.11}$$

Again, these two equations are fixed-point equations and the existence and uniqueness of a solution is guaranteed, under some technical assumptions, by the contraction mapping theorem. Starting from the optimal value functions, we can easily derived an optimal policy. Let us define a partial ordering in the policy space

$$\pi \succeq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in \mathbb{S} \tag{2.12}$$

Then the optimal policy $\pi_* \succeq \pi$, $\forall \pi$. We have the following result

Theorem 2.4.1 (Optimal Policy). *For any Markov decision process,*

i) It exists an optimal policy π_* such that $\pi_* \succeq \pi$, $\forall \pi$.

ii) $V_{\pi_*}(s) = V_*(s)$.

iii) $Q_{\pi_*}(s,a) = Q_*(s,a)$.

An optimal policy can be found by acting greedily with respect to Q_*, i.e. in each state s the agent selects the action that maximizes the action-value function

$$a_* = \arg\sup_a Q_*(s,a) \tag{2.13}$$

We see that this policy is deterministic and only depends on the current state of the system.

2.4.2 Average Reward Formulation

Most of the research in RL has studied a problem formulation where agents maximize the cumulative sum of rewards. However, this approach cannot handle infinite horizon tasks, where there are no absorbing goal states, without discounting future rewards. Clearly, discounting is only necessary in cyclical tasks, where the cumulative reward sum can be unbounded. More natural long-term measure of optimality exists for such cyclical tasks, based on maximizing the average reward per action. For a more in-depth presentation, the reader may again refer to the extensive literature on the subject, such as [4], [51] and the references therein. In the average reward setting, also known as long-run reward or ergodic reward, the goal of the agent is to find a policy that maximizes the expected reward per step.

Definition 2.4.6 (Average Reward). *The average reward ρ_π associated to a policy π is defined as*

$$\begin{aligned}\rho_\pi &= \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} R_{t+1}\right] \\ &= \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi}}\left[\mathcal{R}(S,A)\right] \\ &= \int_{\mathbb{S}} d_\pi(s) \int_{\mathbb{A}} \pi(s,a)\mathcal{R}(s,a) da ds\end{aligned} \tag{2.14}$$

where d_π is the stationary distribution of the Markov process induced by π.

The agent aims to find an *average optimal* policy

$$\pi_* = \arg\sup_\pi \rho_\pi \tag{2.15}$$

In this setting, we introduce the *average adjusted* value and action-value functions.

Definition 2.4.7 (Average Adjusted State-Value Function)**.** *The average adjusted state-value function $V_\pi : \mathbb{S} \to \mathbb{R}$ is the expected residual return that can be obtained starting from a state and following policy π*

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (R_{t+1} - \rho_\pi) \middle| S_0 = s \right] \tag{2.16}$$

The term $V_\pi(s)$ is usually referred to as the *bias* value, or the *relative* value, since it represents the relative difference in total reward gained starting from a state s as opposed to a generic state. ρ_π serves as a baseline that allows to avoid divergence in the value function definition.

Definition 2.4.8 (Average Adjusted Action-Value Function)**.** *The average adjusted action-value function $Q_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the expected residual return that can be obtained starting from a state, taking an action and then following policy π*

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (R_{t+1} - \rho_\pi) \middle| S_0 = s, A_0 = a \right] \tag{2.17}$$

We have the following relation between the state-value function and the action-value function

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s, a) Q_\pi(s, a) \tag{2.18}$$

The value functions satisfy the following Bellman equation

Proposition 2.4.3 (Bellman Expectation Equations)**.**

$$V_\pi(s) = \mathcal{R}_\pi(s) - \rho_\pi + T_\pi V_\pi(s) \tag{2.19}$$

$$Q_\pi(s, a) = \mathcal{R}(s, a) - \rho_\pi + T_a V_\pi(s) \tag{2.20}$$

Again, by introducing opportune Bellman operators, these equations can be rewritten as fixed-point equations. In the discrete case, where the transition operator correspond to matrices, these Bellman equations become linear systems that can be solved to obtain the value functions.

2.5 Risk-Sensitive Framework

In many application, in addition to maximizing the average reward, the agent may want to control risk by minimizing some measure of variability in rewards. In the risk-sensitive framework, the goal of the agent is to find the policy that optimally solves the trade-off between reward and risk. Although risk-sensitive sequential decision-making has a long history in operations research and finance, it has only recently grabbed the attention of the machine learning community. Hence, the literatures offers many reference which approach the risk-sensitive control problem from the traditional stochastic optimal control perspective. On the other hand, there are only few references that attack the problem in the reinforcement learning setting. Again, we can consider the discounted formulation or the average formulation.

2.5.1 Discounted Reward Formulation

A standard way to measure the risk associated with a policy π is the variance of the total discounted reward obtained starting from a given state s

$$\Lambda_\pi(s) = \mathrm{Var}_\pi\left(G_t | S_t = s\right) \tag{2.21}$$

This approach is the one considered in [76]. The variance can be decomposed as

$$\Lambda_\pi(s) = U_\pi(s) - V_\pi(s)^2 \tag{2.22}$$

where we denoted by U_π the square state-value function.

Definition 2.5.1 (Square State-Value Function)**.** *The square state-value function $U_\pi(s)$ is the second moment of the return that can be obtained under policy π starting from a state s*

$$U_\pi(s) = \mathbb{E}_\pi\left[G_t^2 | S_t = s\right] \tag{2.23}$$

As for the risk-neutral formulation, it comes in handy to introduce a square action-value function

Definition 2.5.2 (Square Action-Value Function)**.** *The square action-value function $W_\pi(s)$ is the second moment of the return that can be obtained starting from a state s, taking action a and then following policy π*

$$W_\pi(s,a) = \mathbb{E}_\pi\left[G_t^2 | S_t = s, A_t = a\right] \tag{2.24}$$

Clearly, the two square value functions are related by the following equation

$$U_\pi(s) = \int_{\mathbb{A}} \pi(s,a) W_\pi(s,a) da \tag{2.25}$$

We would like to obtain a Bellman expectation equation for $U_\pi(s)$ and $W_\pi(s,a)$, in order to piggyback on the discussion about the standard value functions. To do so, let us introduce the square reward function

Definition 2.5.3 (Square Reward Function)**.** *The square reward function $\mathcal{M}(s,a)$ is the second moment of the reward that can be obtained in state s when taking action a*

$$\mathcal{M}(s,a) = \mathbb{E}_\pi\left[R_{t+1}^2 | S_t = s, A_t = a\right] \tag{2.26}$$

let us also define the state-reward function $\mathcal{M}_\pi(s)$

$$\mathcal{M}_\pi(s) = \mathbb{E}_\pi\left[R_{t+1}^2 | S_t = s\right] = \int_{\mathbb{A}} \pi(s,a)\mathcal{M}(s,a)da \tag{2.27}$$

Moreover, we will need the reward-return state-covariance function

Definition 2.5.4 (Reward-Return State-Covariance Function)**.** *The reward-return state-covariance function $C_\pi(s)$ is the covariance between the first day reward and the successive returns starting from state s and then following policy π*

$$C_\pi(s) = \mathrm{Cov}_\pi\left(R_{t+1}, G_{t+1} | S_t = s\right) \tag{2.28}$$

again, it comes in handy to also introduce the reward-return action-covariance function

Definition 2.5.5 (Reward-Return Action-Covariance Function)**.** *The reward-return action-covariance function $C_\pi(s,a)$ is the covariance between the first day reward and the successive returns starting from state s, taking action a and then following policy π*

$$C_\pi(s,a) = \mathrm{Cov}_\pi\left(R_{t+1}, G_{t+1} | S_t = s, A_t = a\right) \tag{2.29}$$

Then, it is easy to show that the square state-value function satisfies the following Bellman equation

Proposition 2.5.1 (Bellman Expectation Equation)**.**

$$U_\pi(s) = \mathcal{K}_\pi(s) + \gamma^2 T_\pi U_\pi(s) \tag{2.30}$$

$$W_\pi(s,a) = \mathcal{K}_\pi(s,a) + \gamma^2 T_a U_\pi(s) \tag{2.31}$$

where we denoted

$$\mathcal{K}_\pi(s) = \mathcal{M}_\pi(s) + 2\gamma\mathcal{R}_\pi(s)T_\pi V_\pi(s) + 2\gamma C_\pi(s) \tag{2.32}$$

$$\mathcal{K}_\pi(s,a) = \mathcal{M}(s,a) + 2\gamma\mathcal{R}(s,a)T_a V_\pi(s) + 2\gamma C_\pi(s,a) \tag{2.33}$$

Proof. Let us prove the Bellman expectation equation for the square action-value function. The proof for the square state-value function is analogous.

$$\begin{aligned} W_\pi(s,a) &= \mathbb{E}_\pi\left[G_t^2 | S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[(R_{t+1} + \gamma G_{t+1})^2 | S_t = s, A_t = a\right] \\ &= \mathcal{M}(s,a) + 2\gamma \mathbb{E}_\pi\left[R_{t+1} G_{t+1} | S_t = s, A_t = a\right] + \gamma^2 T_a U_\pi(s) \end{aligned}$$

By the definition of covariance

$$C_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} G_{t+1} | S_t = s, A_t = a\right] - \mathcal{R}(s,a) T_a V_\pi(s)$$

Plugging in the first equation leads to the result. □

These equation are analogous to the Bellman expectation equations for the state-value function, with a synthetic reward function $\mathcal{K}_\pi(s)$ and a discount factor γ^2. It is possible to combine the Bellman expectation equation for $V_\pi(s)$ and for $U_\pi(s)$ to obtain a Bellman equation for the return variance $\Lambda_\pi(s)$.

Proposition 2.5.2 (Bellman Expectation Equation)**.**

$$\Lambda_\pi(s) = \mathcal{V}_\pi(s) + 2\gamma C_\pi(s) + \gamma^2 \operatorname{Var}_\pi\left(V_\pi(S_{t+1}) | S_t = s\right) + \gamma^2 T_\pi \Lambda_\pi(s) \quad (2.34)$$

where $\mathcal{V}_\pi(s)$ denotes the conditional variance of the reward

$$\mathcal{V}_\pi(s) = \operatorname{Var}_\pi\left(R_{t+1} | S_t = s\right) = \mathcal{M}_\pi(s) - \mathcal{R}_\pi(s)^2 \quad (2.35)$$

Proof. The result can be proved starting from Eq. (2.22) and exploiting the Bellman equations for U_π and V_π. Here we follow an alternative way based on the law of total variance.

$$\begin{aligned} \Lambda_\pi(s) &= \operatorname{Var}_\pi\left(G_t | S_t = s\right) \\ &= \operatorname{Var}_\pi\left(R_{t+1} + \gamma G_{t+1} | S_t = s\right) \\ &= \mathcal{V}_\pi(s) + 2\gamma \operatorname{Cov}_\pi\left(R_{t+1}, G_{t+1} | S_t = s\right) + \gamma^2 \operatorname{Var}_\pi\left(G_{t+1} | S_t = s\right) \end{aligned}$$

Applying the law of total variance we obtain

$$\begin{aligned} \operatorname{Var}_\pi\left(G_{t+1} | S_t = s\right) &= \mathbb{E}_\pi\left[\operatorname{Var}_\pi\left(G_{t+1} | S_{t+1}\right) | S_t = s\right] + \operatorname{Var}_\pi\left(\mathbb{E}_\pi\left[G_{t+1} | S_{t+1}\right] | S_t = s\right) \\ &= T_\pi \Lambda_\pi(s) + \operatorname{Var}_\pi\left(V_\pi(S_{t+1}) | S_t = s\right) \end{aligned}$$

Plugging it into the first equality yields the result

$$\Lambda_\pi(s) = \operatorname{Var}_\pi\left(R_{t+1} | S_t = s\right) + \operatorname{Var}_\pi\left(V_\pi(S_{t+1}) | S_t = s\right) + \gamma^2 T_\pi \Lambda_\pi(s)$$

□

Even if appealing from a theoretical point of view, these equations cannot be easily exploited to derive practical algorithms in a continuing environment. Indeed, the covariance term between the first day reward and the future return appearing in the synthetic reward would be hard to estimate when the lifespan of the experiment can be infinite. This doesn't necessarily represent a problem in an episodic environment, where the experiments have a finite (possibly random) lifespan. Even more problematic is the variance of the state-value function, which typically is unknown. For this reason, we will not develop any reinforcement learning algorithm in this framework and we will instead focus on the average reward formulation.
In [83] and [65], the authors circumvent this issue by implicitly assuming that the reward R_{t+1} is conditionally independent from the future rewards given the current state. Under this assumption, the covariance terms are null and the previous Bellman equation become

Corollary 2.5.1 (Bellman Equations Under Independence Assumption).

$$U_\pi(s) = \mathcal{M}_\pi(s) + 2\gamma\mathcal{R}_\pi(s)T_\pi V_\pi(s) + \gamma^2 T_\pi U_\pi(s) \tag{2.36}$$

$$W_\pi(s,a) = \mathcal{M}(s,a) + 2\gamma\mathcal{R}(s,a)T_a V_\pi(s) + \gamma^2 T_a U_\pi(s) \tag{2.37}$$

$$\Lambda_\pi(s) = \mathcal{V}_\pi(s) + \gamma^2 \operatorname{Var}_\pi\left(V_\pi(S_{t+1})|S_t = s\right) + \gamma^2 T_\pi \Lambda_\pi(s) \tag{2.38}$$

2.5.2 Average Reward Formulation

In [65], the authors consider the long-run variance as a measure of the risk associated to a policy π

Definition 2.5.6 (Long-Run Variance). *The long-run variance Λ_π under policy π is defined as*

$$\Lambda_\pi = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi\left[\sum_{t=0}^{T-1}(R_{t+1} - \rho_\pi)^2\right] \tag{2.39}$$

The long-run variance can be decomposed as follows

$$\Lambda_\pi = \eta_\pi - \rho_\pi^2 \tag{2.40}$$

where η_π is the average square reward per step

Definition 2.5.7 (Average Square Reward).

$$\begin{aligned}\eta_\pi &= \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi\left[\sum_{t=0}^{T-1} R_{t+1}^2\right] \\ &= \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi}}\left[\mathcal{M}(S,A)\right] \\ &= \int_{\mathbb{S}} d_\pi(s) \int_{\mathbb{A}} \pi(s,a)\mathcal{M}(s,a)\end{aligned} \tag{2.41}$$

where we denoted by $\mathcal{M}(s,a)$ the square reward function

$$\mathcal{M}(s,a) = \mathbb{E}\left[R_{t+1}^2 | S_t = s, A_t = a\right] \tag{2.42}$$

As before, we introduce the residual state-value and action-value functions associated with the square reward under policy π

Definition 2.5.8 (Average Adjusted Square State-Value Function)**.** *The average adjusted square state-value function $U_\pi : \mathbb{S} \to \mathbb{R}$ is the expected square residual return that can be obtained starting from a state and following policy π*

$$U_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\left(R_{t+1}^2 - \eta_\pi\right)\middle| S_0 = s\right] \tag{2.43}$$

Definition 2.5.9 (Average Adjusted Square Action-Value Function)**.** *The average adjusted square action-value function $Q_\pi : \mathbb{S}\times\mathbb{A} \to \mathbb{R}$ is the expected residual square return that can be obtained starting from a state, taking an action and then following policy π*

$$W_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\left(R_{t+1}^2 - \eta_\pi\right)\middle| S_0 = s, A_0 = a\right] \tag{2.44}$$

The following relation between square state-value function and the square action-value holds

$$U_\pi(s) = \int_{\mathbb{A}} \pi(s,a) W_\pi(s,a) \tag{2.45}$$

The average adjusted square value functions satisfy the following Bellman equations

Proposition 2.5.3 (Bellman Expectation Equations)**.**

$$U_\pi(s) = \mathcal{M}_\pi(s) - \eta_\pi + T_\pi U_\pi(s) \tag{2.46}$$

$$\begin{aligned} W_\pi(s,a) &= \mathcal{M}(s,a) - \eta_\pi + T_a U_\pi(s) \\ &= \mathcal{M}(s,a) - \eta_\pi + \int_{\mathbb{S}} \mathcal{P}(s,a,s') \int_{\mathbb{A}} \pi(s',a') W_\pi(s',a') da' ds' \end{aligned} \tag{2.47}$$

In the risk-sensitive setting, the agent wants to find a policy that solves the following mean-variance optimization problem

$$\begin{cases} \max_\pi \rho_\pi \\ \text{subject to } \Lambda_\pi \le \alpha \end{cases} \tag{2.48}$$

for a given $\alpha > 0$. Using the Lagrangian relaxation procedure, we can recast (2.48) to the following uncostrained problem

$$\max_{\lambda} \min_{\pi} L(\pi, \lambda) = -\rho_\pi + \lambda(\Lambda_\theta - \alpha) \tag{2.49}$$

Alternatively, the agent may want to optimize the Sharpe ratio, a risk-sensitive performance measure commonly used in finance

$$\mathrm{Sh}_\pi = \frac{\rho_\pi}{\sqrt{\Lambda_\pi}} \tag{2.50}$$

2.6 Dynamic Programming Algorithms

The Bellman equations provide the basis for the *value iteration* and *policy iteration* algorithms [80], which simply consist in iteratively applying the Bellman operators to an approximation of the value functions. As we will see, these methods require knowledge of the MDP dynamics and typically they are only applicable to finite state MDPs, for which the value functions can be represented as vectors and the Bellman operators becomes matrices. However, these simple algorithm provide some useful insight that can be exploited to develop more advanced reinforcement learning algorithm

2.6.1 Value Iteration

Value iteration is an iterative method to compute the optimal state-value function $V_*(s)$. Starting from an arbitrary function $V_0(s)$, the algorithm iteratively updates the approximation by applying the Bellman optimality operator in a fixed-point scheme

$$V_{k+1}(s) = B_* V_k(s) = \sup_a \left\{ \mathcal{R}(s,a) + \gamma T_a V_*(s) \right\}$$

This algorithm is guaranteed to converge to the optimal value function V_* by the contraction mapping theorem. Let us notice that the transition operator T_a requires knowledge of the MDP dynamics, which is not available in the typical reinforcement learning framework. Moreover, the update involves an optimization with respect to all possible actions which can be carried out efficiently only in the case of finite action spaces. A drawback of this algorithm is that it does not provide an explicit representation of the optimal policy, which in many control problems is what we are looking for.

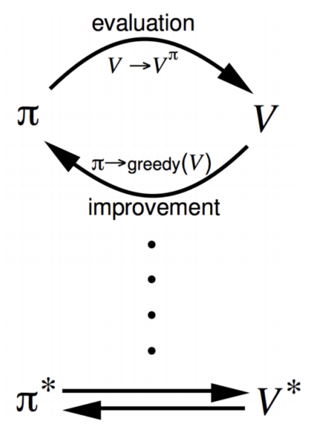

Figure 2.2: Policy iteration algorithm [78].

2.6.2 Policy Iteration

Policy iteration is an iterative method to approximate both the optimal state-value function V_* and the optimal policy π_*. This algorithm alternates an evaluation step, in which the current policy is evaluated using the state-value function, and an improvement step, in which the policy is improved by acting greedily with respect to the action-value function computed in the evaluation step. In the standard version of policy iteration, the state-value function for the current policy is evaluated starting from an arbitrary function $V_0(s)$ and iteratively applying the Bellman expectation operator in a fixed-point iteration scheme

$$V_{k+1}(s) = B_{\pi_n} V_k(s)$$

This algorithm is guaranteed to converge to V_{π_n}. The new policy is then computed as

$$\pi_{n+1} = \text{greedy}(V_{\pi_n})$$

and we go back to the evaluation step for this new policy. The algorithm is guaranteed to converge to V_* and π_*. Let us notice that it is not necessary to perfectly evaluate to policy π_n before performing the improvement step and the evaluation procedure can be stopped before convergence. This algorithm suffers from the same problem as above. However it provides the basic structure for most of the value-based reinforcement learning methods. In particular, in *generalized policy iteration* the evaluation step is performed in an arbitrary way which does not necessarily employ the Bellman operator. This scheme is illustrated in Figure 2.2,

Chapter 3

Reinforcement Learning

The standard way to solve Markov decision processes is through dynamic programming, which simply consists in solving the Bellman fixed-point equations discussed in the previous chapter. Following this approach, the problem of finding the optimal policy is transformed into the problem of finding the optimal value function. However, apart from the simplest cases where the MDP has a limited number of states and actions, dynamic programming becomes computationally infeasible. Moreover, this approach requires complete knowledge of the Markov transition kernel and of the reward function, which in many real-world applications might be unknown or too complex to use. Reinforcement Learning (RL) is a subfield of Machine Learning which aims to turn the infeasible dynamic programming methods into practical algorithms that can be applied to large-scale problems. RL algorithms are based on two key ideas: the first is to use samples to compactly represent the unknown dynamics of the controlled system. The second idea is to use powerful function approximation methods to compactly estimate value functions and policies in high-dimensional state and action spaces. In the following sections, we present the reinforcement learning problem in more depth and discuss how it relates to the standard discrete-time stochastic optimal control theory and to the other subfields of machine learning, such as supervised learning. In particula, we will see how RL extends ideas from optimal control theory and stochastic approximation to address the broader goal of artificial intelligence. This quick overview of RL is propaedeutic to the following chapters, where we will present in more detail a particular class of algorithms, called policy gradient methods, which are well suited for continuous action spaces. For a more thorough presentation, the reader may consult [78], [80] or [88].

3.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) is a general class of algorithms in the field of machine learning that allows an agent to learn how to behave in a stochastic and possibly unknown environment, where the only feedback consists of a scalar reward signal. In order to maximize the long-run reward, the agent must learn which actions are the most profitable by trial-and-error. Therefore, RL algorithms can be seen as computational methods to solve Markov decision processes by directly interacting with the environment, for which a model may or may not be available. Trial-and-error search and a delayed reward signal can be seen as the most characteristic features of reinforcement learning.

Compared to supervised learning, one of the main branches of machine learning, the feedback the learner receives is much less. In supervised learning, the agent is provided with examples of the correct or expected behavior by a knowledgeable external supervisor and his goal is to learn how to replicate these examples as well as possible and possibly generalize this knowledge to new examples. In reinforcement learning, the agent only receives a numerical reward that only gives a partial feedback of the goodness of his actions. Therefore, this feedback system is evaluative rather than instructive and it is much more difficult for the agent to learn how to behave in uncharted territory without any external guidance.

This particular framework generates some challenges that are not present in other kinds of learning. The first one is the trade-off between exploration and exploitation. In order to maximize his reward, an agent would greedily select actions that he has already tried in the past and found to be effective in producing rewards. However, to find these actions, he has to test actions that haven't been selected before in order to evaluate their potential. Clearly, this might result in worse performance in the short-term because the actions might be suboptimal. However, without trying them, the agent might not be able to find possible improvements. Thus, an agent must exploit what he already knows to obtain rewards, but he also needs to explore to select better actions in the future. A second challenge is the credit assignment problem. Since rewards might be delayed in time, it will be difficult for the agent to understand which actions are mostly responsible for the outcome.

3.2 Model-Free RL Methods

In Section 2.6, we discussed the policy iteration method for computing an optimal policy for an MDP in a finite state and action spaces. This algorithm belongs to the class of *model-based* methods, since it requires perfect

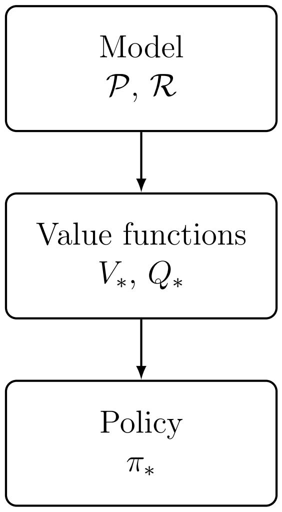

Figure 3.1: Solution process for the control problem.

knowledge of the Markov transition kernel and reward function, i.e. the model of the MDP. RL is primarily concerned with how to obtain an optimal policy when such a model is not available. In this section, we discuss some classes of *model-free* methods which do not rely on the MDP model. The lack of a model generates the need to sample the MDP to gather statistical knowledge about this unknown model. In the control setting, the goal is to approximate the optimal policy, which depends on the optimal value, which in turn depends on the model of the MDP, as shown in Figure 3.1. Indeed, we have already seen that a policy which is greedy with respect to the optimal action-value funtion Q_*, i.e. such that

$$\int_{\mathbb{A}} \pi_*(s,a) Q_*(s,a) da = \sup_a Q_*(s,a) \tag{3.1}$$

is optimal. Therefore, we can derive the following three methodologies that differ in which part of the solution process is approximated

i) *Model-approximation* algorithms approximate the MDP model and compute an estimate of the optimal policy by dynamic programming.
ii) *Value-approximation* algorithms use samples to directly approximate V_* or Q_*, from which an estimate of π_* can be derived by acting greedily.
iii) *Policy-approximation* algorithms directly try to estimate the optimal policy.

It should be noticed that these approaches are not mutually exclusive and can be combined to derive hybrid algorithms. In the following sections we discuss these three classes of algorithm in more detail, following closely [88].

3.2.1 Model Approximation

Model-approximation algorithms approximate the MDP model and compute an optimal policy by dynamic programming, using the techniques discussed in the previous chapter. Since $\mathbb{S}$, $\mathbb{A}$ and γ are assumed to be known, these methods are based on learning an approximation of the Markov transition kernel $\mathcal{P}$ and the reward function $\mathcal{R}$. Thanks to the Markov property, these quantities only depend on the current state and action, so that their approximation corresponds to a density estimation problem and a regression problem respectively, which are fairly standard supervised learning problems. Learning the model may not be trivial, but it is in general easier than learning the value of a policy or optimizing the policy directly. The major drawback of model-based algorithms in continuous-state MDPs is that, even if the model is available, it is in general infeasible to compute the value functions by dynamic programming and to extract an optimal policy for all states by acting greedily. Alternatively, a transition model estimate may be used to generate sample trajectories from the MDP, which can then be used to estimate the value function or directly improve the policy. However, the value function and the policy estimated using these samples can only be as accurate as the learned model, so that in many cases it may be easier to directly approximate the value function or the policy using the methods described below.

3.2.2 Value Approximation

Value-approximation algorithms use samples from the MDP to approximate V_* or Q_* directly and then derive an estimate of the optimal policy by acting greedily with respect to Q_*. Typically, when the state and action spaces are large, the value functions are estimated using a parametric function approximator, whose parameters are iteratively updated given the observed samples. Many reinforcement learning algorithms fall into this category and they can be distinguishes based on whether they are on-policy or off-policy and whether they update the value function estimates online or offline. *On-policy* algorithms approximate the state-value function V_π or the action-value function Q_π from samples of the MDP obtained by following the same policy π to be evaluated. Although the optimal policy π_* is initially unknown, such algorithms can eventually approximate the optimal value functions V_* or Q_* from which an approximation of the optimal policy can be derived. On the other hand, *off-policy* algorithms can learn the value of a policy different from the one use for obtaining the MDP samples. *Online* algorithms adapt their value approximation after each observed sample while *Offline* algorithms operate on batches of samples. Online algorithms typically require less computation per sample but their convergence is slower. Online on-policy algorithms include temporal-difference (TD) algorithms, such as TD-learning, Sarsa. Offline on-policy algorithms include least-squares approaches, such least-squares temporal difference (LSTD), least-squares policy evaluation (LSPE) and least-squares policy iteration (LSPI). The most known model-free online off-policy algorithm is Q-learning.

3.2.3 Policy Approximation

Policy-approximation algorithms only store a policy and update this policy to maximize a given performance measure and eventually converge to the optimal policy. Since these algorithms only use a policy estimate and don't rely on a value function approximation, they are also referred to as *direct policy-search* or *actor-only* algorithms. Algorithms that store both a policy and a value function are commonly known as *actor-critic* algorithms [79], [44]. Policy gradient algorithms, the most studied policy-approximation methods, will be discussed in much more detail in the next chapter.

Chapter 4

Policy Gradient

Policy gradient methods directly store and iteratively improve a parametric approximation of the optimal policy. This policy is commonly referred to as an *actor* and methods that directly approximate the policy without exploiting an approximation of the optimal value function are called *actor-only*. Algorithms that combine an approximation of the optimal policy with an approximation of the value function are commonly called *actor-critic* methods. As discussed in Chapter 2, an optimal policy can be obtained by simply acting greedily with respect to the optimal action-value function. However, in large or continuous action spaces, this leads to a complex optimization problem that is computationally expensive to solve. Therefore, it can be beneficial to store an explicit estimation of the optimal policy from which we can select actions. Policy gradient methods have other advantages compared to standard value-based approaches. In many applications, a good policy has a more compact representation than the value function so that it may be easier to approximate. Moreover, the policy parameterization can be chosen so as to be relevant for the task and to directly incorporate prior knowledge. Another advantage is that these methods can learn stochastic policies and not only deterministic ones, which might be useful in multiplayer frameworks or in partially observable environments. Finally, these methods have better convergence properties and are guaranteed to converge at least to a local optimum, which may be good enough in practice. On the other hand, policy gradient methods are typically characterized by a large variance which may hinder the converge speed.
In the following sections, we present the basics of policy gradient methods closely following the thorough overviews provided in [64]. Then, we discuss some state-of-the-art policy gradient algorithms and we propose some original extensions to these methods.

4.1 Basics of Policy Gradient Methods

In policy gradient methods, the optimal policy is approximated using a parametrized policy $\pi : \mathbb{S} \times \mathcal{A} \times \Theta \to \mathbb{R}$ such that, given a parameter vector $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$, $\pi(s, B; \theta) = \pi_\theta(s, B)$ gives the probability of selecting an action in $B \in \mathcal{A}$ when the system is in state $s \in \mathbb{S}$. The general goal of policy optimization in reinforcement learning is to optimize the policy parameters $\theta \in \Theta$ so as to maximize a certain objective function $J : \Theta \to \mathbb{R}$

$$\theta^* = \arg\max_{\theta \in \Theta} J(\theta)$$

In the following, we will focus on gradient-based and model-free methods that exploit the sequential structure of the the reinforcement learning problem. The idea of policy gradient algorithms is to update the policy parameters using the gradient ascent direction of the objective function

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J\left(\theta_k\right) \tag{4.1}$$

where $\{\alpha_k\}_{k \geq 0}$ is a sequence of learning rates. Typically, the gradient of the objective function is not known and its approximation is the key component of every policy gradient algorithm, a general setup of which is outlined in Algorithm 4.1. It is a well-know result from stochastic optimization [45] that, if the gradient estimate is unbiased and the learning rates satisfy the *Robbins-Monro conditions*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{4.2}$$

the learning process is guaranteed to converge at least to a local optimum of the objective function. Before describing various methods to approximate the gradient, let us give an overview of some standard objective functions and the situation in which they are commonly used.

Algorithm 4.1 General setup for a policy gradient algorithm.

```
Input: Initial parameters $\theta_0$, learning rate $\{\alpha_k\}$
Output: Approximation of the optimal policy $\pi_*$
1: repeat
2:     Approximate policy gradient $\widehat{g} \approx \nabla_\theta J\left(\theta_k\right)$
3:     Update parameters using gradient ascent $\theta_{k+1} = \theta_k + \alpha_k \widehat{g}$
4:     $k \leftarrow k + 1$
5: until converged
```

4.1.1 Risk-Neutral Framework

In the risk-neutral setting, the agent is only interested in maximizing his reward, without considering the risk he needs to take on to achieve it. In an episodic environment where the system always starts from an initial state s_0, the typical objective function is the start value.

Definition 4.1.1 (Start Value)**.** *The start value is the expected return that can be obtained starting from the start state $s_0 \in \mathbb{S}$ and following policy π_θ*

$$J_{start}(\theta) = V_{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}\left[G_0 | S_0 = s_0\right] \tag{4.3}$$

In a continuing environment, where no terminal state exists and the task might go on forever, it is common to use either the average value or the average reward per time step.

Definition 4.1.2 (Average Value)**.** *The average value is the expected value that can be obtained following policy π_θ*

$$J_{avV}(\theta) = \mathbb{E}_{S\sim d^\theta}\left[V_{\pi_\theta}(S)\right] = \int_{\mathbb{S}} d^\theta(s) V_{\pi_\theta}(s) ds \tag{4.4}$$

where d^θ is the stationary distribution of the Markov chain induced by π_θ.

Definition 4.1.3 (Average Reward per Time Step)**.** *The average reward per time step is the expected reward that can be obtained over a single time step by following policy π_θ*

$$J_{avR}(\theta) = \rho(\theta) = \mathbb{E}_{\substack{S\sim d^\theta \\ A\sim\pi_\theta}}\left[\mathcal{R}(S, A)\right] = \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi_\theta(s, a)\mathcal{R}(s, a) da ds \tag{4.5}$$

where d^θ is the stationary distribution of the Markov chain induced by π_θ.

4.1.2 Risk-Sensitive Framework

In the risk-sensitive framework, in addition to maximizing his reward, the agent also wants to control the risk he takes on to achieve it. It is thus necessary to introduce a function $\Lambda : \Theta \to \mathbb{R}$ such that $\Lambda(\theta)$ measures the risk associated with the policy π_θ. In an episodic framework where the system always starts from the same initial state s_0, the variance of the total return can be used [83]

Definition 4.1.4 (Start Variance)**.** *The start variance is the variance of the return that can be obtained starting from the start state $s_0 \in \mathbb{S}$ and following policy π_θ*

$$\Lambda_{start}(\theta) = \mathrm{Var}_{\pi_\theta}\left(G_0 \mid S_0 = s_0\right) = U_{\pi_\theta}(s_0) - V_{\pi_\theta}(s_0)^2 \tag{4.6}$$

In a continuing environment, we might consider the long-run variance [65] defined in Section 2.5

Definition 4.1.5 (Long-Run Variance). *The long-run variance Λ_π under policy π is defined as*

$$\Lambda_{long\text{-}run}(\theta) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} (R_{t+1} - \rho(\theta))^2\right] \tag{4.7}$$

In order to formalize this trade-off between reward and risk from a mathematical point of view, we borrow from the financial literature two standard risk-sensitive performance measures: the mean-variance criterion [52] and the Sharpe ratio [74].

Definition 4.1.6 (Mean-Variance Criterion). *The mean-variance criterion is defined as*

$$J_{MV}(\theta) = J(\theta) - \chi\Lambda(\theta) \tag{4.8}$$

where $\chi > 0$ is a constant that controls the trade-off between reward and risk.

Definition 4.1.7 (Sharpe Ratio). *The Sharpe ratio is defined as*

$$\mathrm{Sh}(\theta) = \frac{J(\theta)}{\sqrt{\Lambda(\theta)}} \tag{4.9}$$

In a risk-sensitive policy gradient algorithm, we try to approximate the optimal parameters that maximize these objective functions by updating the parameters following the gradient ascent directions. In the average-reward formulation, the ascent directions are given by

$$\nabla_\theta J_{\mathrm{MV}}(\theta) = \nabla_\theta \rho(\theta) - \chi \nabla_\theta \Lambda(\theta) \tag{4.10}$$

for the mean-variance criterion, where

$$\nabla_\theta \Lambda(\theta) = \nabla_\theta \eta(\theta) - 2\rho(\theta)\nabla_\theta\rho(\theta) \tag{4.11}$$

and by

$$\nabla_\theta \,\mathrm{Sh}(\theta) = \frac{\eta(\theta)\nabla_\theta\rho(\theta) - \frac{1}{2}\rho(\theta)\nabla_\theta\eta(\theta)}{\Lambda(\theta)\sqrt{\Lambda(\theta)}} \tag{4.12}$$

for the Sharpe ratio. Hence, in the risk-sensitive framework it is necessary to estimate to different gradients: $\nabla_\theta\rho(\theta)$ and $\nabla_\theta\eta(\theta)$. Let us notice that the equivalent expressions for the episodic setting can be obtained by replacing $\rho(\theta)$ (resp. $\eta(\theta)$) with $V_\theta(s_0)$ (resp. $U_\theta(s_0)$). In the next sections, we will discuss several techniques to estimate these gradients.

4.2 Finite Differences

The most straightforward way to estimate the objective function gradient consists in replacing the partial derivatives with the corresponding finite differences. In other words, the k-th gradient component is approximated by

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon e_k) - J(\theta)}{\epsilon} \tag{4.13}$$

where ϵ is a small constant and $e_k \in \mathbb{R}^{D_\theta}$ is the k-th element of the canonical basis. Hence, to compute the gradient it is sufficient to estimate the objective function for $D_\theta + 1$ different parameters combinations. Since the objective function is unknown, it must be estimated from sample trajectories simulated following the policies associated to the parameterizations appearing in the finite differences. For this reason, this method is computationally demanding and the gradient estimate obtained in this way is extremely noisy, which may slow down or even prevent the convergence of the algorithm. Still, this approach is sometimes effective and works for arbitrary, even non-differentiable, policies. Moreover, finite differences are often used to check gradient estimates during debugging.

4.3 Likelihood Ratio Methods

In this section, we describe several methods based on the *likelihood ratio* technique of stochastic optimization. Let Z be a random variable with a parametric probability density p_θ and assume that p_θ is known, explicitly computable and differentiable with respect to the parameters. The likelihood ratio technique is used to compute the following gradient

$$\nabla_\theta \mathbb{E}_{Z\sim p_\theta}\left[f(Z)\right] = \nabla_\theta \int p_\theta(z) f(z) dz$$

This type of problem appears in many domains, such as when computing the greeks of a derivative product in computational finance [61]. Under some regularity assumptions on the function f and on the probability distribution p_θ, this gradient can be rewritten in a more amenable way

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{Z\sim p_\theta}\left[f(Z)\right] &= \int \nabla_\theta p_\theta(z) f(z) dz \\
&= \int p_\theta(z) \frac{\nabla_\theta p_\theta(z)}{p_\theta(z)} f(z) dz \\
&= \int p_\theta(z) \nabla_\theta \log p_\theta(z) f(z) dz \\
&= \mathbb{E}_{Z\sim p_\theta}\left[\nabla_\theta \log p_\theta(Z) f(Z)\right]
\end{aligned}$$

Thus, the likelihood ratio technique simply consists in the following equality

$$\nabla_\theta \mathbb{E}_{Z\sim p_\theta}\left[f(Z)\right] = \mathbb{E}_{Z\sim p_\theta}\left[\nabla_\theta \log p_\theta(Z) f(Z)\right] \tag{4.14}$$

where $\nabla_\theta \log p_\theta(Z)$ is usually called likelihood score. This result provides a simple way to approximate the gradient using a Monte Carlo estimate: let $\{Z^{(m)}\}_{m=1}^M$ be i.i.d. samples from p_θ, then

$$\nabla_\theta \mathbb{E}_{Z\sim p_\theta}\left[f(Z)\right] \approx \frac{1}{M}\sum_{m=1}^{M} \nabla_\theta \log p_\theta(Z^{(m)}) f(Z^{(m)}) \tag{4.15}$$

As we will see in the following sections, this technique provides the basis for some of the most common policy gradient algorithms in the reinforcement learning literature.

4.3.1 Monte Carlo Policy Gradient

Let $h = \{(s_t, a_t)\}_{t\geq 0} \in \mathbb{H}$ be a given trajectory of the MDP and let us denote by $p_\theta(h) = \mathbb{P}_{\pi_\theta}\left(H = h\right)$ the probability of obtaining this trajectory under policy π_θ. Let $G(h)$ denote the expected return obtained on trajectory h

$$G(h) = \mathbb{E}\left[G_0 | H = h\right] = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)$$

Let us consider the average value objective function, which can be rewritten as an expectation over all possible trajectories

$$J_{\text{avV}}(\theta) = \mathbb{E}_{H\sim p_\theta}\left[G(H)\right]$$

Applying the likelihood ratio technique, we obtain

$$\nabla_\theta J(\theta) = \mathbb{E}_{H\sim p_\theta}\left[\nabla_\theta \log p_\theta(H) G(H)\right] \tag{4.16}$$

The crucial point is that $\nabla_\theta \log p_\theta(H)$ can be computed without knowledge of the transition probability kernel $\mathcal{P}$. Indeed, by a recursive application of the Markov property, we have

$$p_\theta(h) = \mathbb{P}\left(S_0 = s_0\right) \prod_{t=0}^{\infty} \pi_\theta(s_t, a_t) \mathcal{P}(s_t, a_t, s_{t+1})$$

from which we obtain

$$\log p_\theta(h) = \log \mathbb{P}\left(S_0 = s_0\right) + \sum_{t=0}^{\infty} \log \pi_\theta(s_t, a_t) + \sum_{t=0}^{\infty} \log \mathcal{P}(s_t, a_t, s_{t+1})$$

Since the only term depending on the parameters θ is the policy term,

$$\nabla_\theta \log p_\theta(H) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{4.17}$$

Therefore, we do not need the transition model to compute the $\nabla_\theta \log p_\theta(H)$. Moreover, it is easy to prove that

$$\mathbb{E}_{H\sim p_\theta}\left[\nabla_\theta \log p_\theta(H)\right] = 0 \tag{4.18}$$

Hence, a constant baseline $b \in \mathbb{R}$ can always be subtracted in Eq. 4.16 without changing the gradient value

$$\nabla_\theta J(\theta) = \mathbb{E}_{H\sim p_\theta}\left[\nabla_\theta \log p_\theta(H)(G(H) - b)\right] \tag{4.19}$$

Intuitively, the baseline b should measure the expected return under the policy, so that better (resp. worse) than average returns would produce a positive (resp. negative) gradient. A simple approach is to use a moving average of the returns observed while improving the policy. A more elaborate approach is to compute the baseline that minimizes the variance of the gradient estimator.
In an episodic environment, we can derive an estimate of the gradient by sampling M trajectories $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)})\}_{t=0}^{T^{(m)}}$ under policy π_θ and by approximating the expected value via Monte Carlo

$$\widehat{g}_{\mathrm{RF}} = \frac{1}{M}\sum_{m=1}^{M}\left[\sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)})\right]\left[\sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b\right] \tag{4.20}$$

This method, synthetized in Algorithm 4.2, is known in the literature as the REINFORCE algorithm and is guaranteed to converge to the true gradient at a pace of $O(M^{-1/2})$. In practice, we can obtain an approximation of the gradient using only one sample which leads to a stochastic gradient ascent method

$$\widehat{g}_{\mathrm{SRF}} = \left[\sum_{i=0}^{T} \nabla_\theta \log \pi_\theta(s_i, a_i)\right]\left[\sum_{j=0}^{T} \gamma^j r_{j+1} - b\right] \tag{4.21}$$

This method is very easy and works well on many problems. However, the gradient estimate is characterized by a large variance which can hamper the convergence rate of the algorithm. A first approach to address this issue is to optimally set the baseline to reduce the gradient variance.

Algorithm 4.2 Episodic REINFORCE policy gradient estimate

```
Input: Policy parameterization $\theta$, number of trajectories $M$
Output: REINFORCE policy gradient estimate $\widehat{g}_{RF} \approx \nabla_\theta J(\theta)$
1: Sample $M$ trajectories of the MDP following policy $\pi_\theta$
2: For all $k$, Compute the optimal baseline $b_k^*$ according to Eq. (4.22)
3: Compute $\widehat{g}_{\mathrm{RF}}$ according to Eq. (4.20)
```

4.3.1.1 Optimal Baseline

A standard variance reduction technique, called control variate, consists in setting the baseline so as to minimize the gradient estimate variance. More in detail, the optimal baseline for the k-th gradient component $\widehat{g}_k$ solves

$$b_k^* = \arg\min_b \operatorname{Var}\left(\widehat{g}_k\right)$$

It is easy to show that

$$b_k^* = \frac{\mathbb{E}\left[G(H)\left(\partial_{\theta_k}\log p_\theta(H)\right)^2\right]}{\mathbb{E}\left[\left(\partial_{\theta_k}\log p_\theta(H)\right)^2\right]}$$

which can be approximated by

$$\widehat{b}_k^* = \frac{\sum_{m=1}^{M}\left[\sum_{i=0}^{T^{(m)}}\partial_{\theta_k}\log\pi_\theta\left(s_i^{(m)},a_i^{(m)}\right)\right]^2\sum_{j=0}^{T^{(m)}}\gamma^j r_{j+1}^{(m)}}{\sum_{m=1}^{M}\left[\sum_{i=0}^{T^{(m)}}\partial_{\theta_k}\log\pi_\theta\left(s_i^{(m)},a_i^{(m)}\right)\right]^2} \tag{4.22}$$

4.3.2 GPOMDP

The Monte Carlo policy gradient estimate is typically characterized by a large variance, which may slow the method's convergence. To improve the estimate, it is sufficient to notice that future actions and past rewards are independent, unless the policy has been changed. Therefore, combining this observation with Eq. (4.18) yields

$$\mathbb{E}_{\pi_\theta}\left[\nabla_\theta\log\pi_\theta(S_t,A_t)\mathcal{R}(S_s,A_s)\right] = 0 \qquad \forall t > s$$

Plugging this equation in Eq. (4.19) leads to the well-known GPOMDP algorithm [7] for generating an estimate of the gradient of the objective function

$$\widehat{g}_{\mathrm{GPOMDP}} = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=0}^{T^{(m)}}\nabla_\theta\log\pi_\theta(s_i^{(m)},a_i^{(m)})\left(\sum_{j=i}^{T^{(m)}}\gamma^j r_{j+1}^{(m)} - b\right) \tag{4.23}$$

By removing almost half of the cross-products, this estimate typically has a smaller variance than the trivial REINFORCE policy gradient and can be further reduced by computing an optimal baseline. In the following sections we will see how this algorithm can be easily derived from a more general result: the policy gradient theorem.

4.3.3 Risk-Sensitive Monte Carlo Policy Gradient

For an episodic environment, the REINFORCE algorithm can be easily extended to the risk-sensitive framework described above. Indeed, it is sufficient to adapt its derivation for the average return to the second-moment of return

$$U(\theta) = \mathbb{E}_{H \sim p_\theta}\left[G(H)^2\right]$$

Applying the likelihood ratio technique, we obtain

$$\nabla_\theta U(\theta) = \mathbb{E}_{H \sim p_\theta}\left[\nabla_\theta \log p_\theta(H) G(H)^2\right]$$

From which we can easily derive a Monte Carlo estimate. Since the problems we will consider in the next chapters are not episodic, we will not investigate this approach further and we refer the interested reader to the extensive work of Tamar et Al. [83], [81], [84], [82], [24].

4.3.4 Stochastic Policies

The likelihood ratio technique is a powerful tool and it allows to derive some powerful policy gradient algorithms. However, this approach only works if the policy is stochastic. In most cases this is not a big problem, since stochastic policies are needed anyway to ensure sufficient exploration of the state-action space. Moreover, stochastic policies might be beneficial in partially observable MDP to avoid state aliasing and in adversarial learning where the Pareto optimal strategies of different players are randomized, such as in the classical game “rock-paper-scissors”. We introduce now two standard examples of stochastic policies for discrete and continuous action spaces respectively.

4.3.4.1 Boltzmann Exploration Policy

In discrete action spaces, the Boltzmann exploration policy, also known as softmax policy, is a common choice. In state $s \in \mathbb{S}$, this policy selects an action $a \in \mathbb{A}$ with probability

$$\pi_\theta(s, a) = \frac{e^{\theta^T \Phi(s,a)}}{\sum_{b \in \mathbb{A}} e^{\theta^T \Phi(s,b)}} \tag{4.24}$$

where $\Phi(s,a) \in \mathbb{R}^{D_\theta}$ is a given feature vector corresponding to state s and action a. the likelihood score for this policy is thus given by

$$\nabla_\theta \log \pi_\theta(s,a) = \Phi(s,a) - \sum_{b\in\mathbb{A}} \pi_\theta(s,a)\Phi(s,b) \tag{4.25}$$

4.3.4.2 Gaussian Exploration Policy

In continuous action spaces, a Gaussian exploration policy is commonly used. According to this policy, in a state s, actions are sampled from a Gaussian distribution with a parametric state-dependent mean $\mu_\psi(s) \in \mathbb{R}^{D_a}$, with $\psi \in \mathbb{R}^{D_\psi}$, and a covariance matrix $\Sigma \in \mathbb{R}^{D_a\times D_a}$. The policy parameters consist of $\theta = \psi, \Sigma$ and the likelihood score are thus given by

$$\nabla_\psi \log \pi_\theta(s,a) = \left(\frac{\partial \mu_\psi(s)}{\partial \psi}\right)^T \Sigma^{-1}(a - \mu_\psi(s)) \tag{4.26}$$

$$\nabla_\Sigma \log \pi_\theta(s,a) = \frac{1}{2}\left[\Sigma^{-1}\left(a - \mu_\psi(s)\right)\left(a - \mu_\psi(s)\right)^T \Sigma^{-1} - \Sigma^{-1}\right] \tag{4.27}$$

where $\frac{\partial \mu_\psi(s)}{\partial \psi}$ denotes the Jacobian matrix of μ_ψ with respect to ψ.

4.3.5 Policy Gradient with Parameter Exploration

In Monte Carlo Policy Gradient, trajectories are generated by sampling at each time step an action according to a stochastic policy π_θ and the objective function gradient is estimated by differentiating the policy with respect to the parameters. However, sampling an action from the policy at each time step leads to a large variance in the sampled histories and therefore in the gradient estimate, which can in turn slow down the convergence of the learning process. To address this issue, in [72] the authors propose the *policy gradient with parameter-based exploration* (PGPE) method, in which the search in the policy space is replaced with a direct search in the model parameter space. We start by presenting the episodic case and we will later extend this approach to the infinite horizon setting.

4.3.5.1 Episodic PGPE

Given an episodic MDP, PGPE considers a deterministic controller $F : \mathbb{S} \times \Theta \to \mathbb{A}$ that, given a set of parameters $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$, maps a state $s \in \mathbb{S}$ to an action $a = F(s;\theta) = F_\theta(s) \in \mathbb{A}$. The policy parameters are drawn from a probability distribution p_ξ, with hyper-parameters $\xi \in \Xi \subseteq \mathbb{R}^{D_\xi}$. Combining these two hypotheses, the agent follows a stochastic policy π_ξ defined by

$$\forall B \in \mathcal{A},\ \pi_\xi(s, B) = \pi(s, B; \xi) = \int_\Theta p_\xi(\theta) \mathbb{1}_{F_\theta(s) \in B} d\theta$$

The advantage of this approach is that the controller is deterministic and therefore the actions do not need to be sampled at each time step, with a consequent reduction of the gradient estimate variance. Indeed, It is sufficient to sample the parameters θ once at the beginning of the episode and then generate an entire trajectory following the deterministic policy F_θ. As an additional benefit, the parameter gradient is estimated by direct parameter perturbations, without having to backpropagate any derivatives, which allows to use non-differentiable controllers.
The hyper-parameters ξ will be updated by following the gradient ascent direction of the gradient of the expected reward, which can be rewritten as

$$J(\xi) = \mathbb{E}_{\substack{\theta \sim p_\xi \\ H \sim p_\theta}} [G(H)] = \int_\Theta \int_\mathbb{H} p_\xi(\theta, h) G(h) dh d\theta \tag{4.28}$$

By remarking that h is conditionally independent from ξ given θ, so that $p_\xi(\theta, h) = p_\xi(\theta) p_\theta(h)$, and applying the likelihood ratio technique, we obtain

$$\nabla_\xi J(\xi) = \mathbb{E}_{\substack{\theta \sim p_\xi \\ H \sim p_\theta}} [\nabla_\xi \log p_\xi(\theta) G(H)] \tag{4.29}$$

Again, we can subtract a constant baseline $b \in \mathbb{R}$ from the total return

$$\nabla_\xi J(\xi) = \mathbb{E} \left[\nabla_\xi \log p_\xi(\theta) \left(G(H) - b\right)\right] \tag{4.30}$$

In an episodic environment, the gradient can be approximated via Monte Carlo by first drawing M samples $\theta^{(m)} \sim p_\xi$ and then, for each combination of parameters, generating a trajectory $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)})\}_{t \geq 0}$ where actions are selected according to the deterministic controller $F_{\theta^{(m)}}$. This leads to the following estimate

$$\widehat{g}_{\text{PGPE}} = \frac{1}{M} \sum_{m=1}^{M} \nabla_\xi \log p_\xi \left(\theta^{(m)}\right) \left[G\left(h^{(m)}\right) - b\right] \tag{4.31}$$

In order to further reduce the estimate variance, an optimal baseline can be computed similarly to the REINFORCE case [91]. The episodic PGPE algorithm obtained in this way is reported in Algorithm 4.3. In the following paragraphs, we discuss some possible choices for the policy parameters distribution p_ξ, which is the last component of the algorithm.

Algorithm 4.3 Episodic PGPE algorithm

```
Input: Initial hyper-parameters $\xi_0$, learning rate $\{\alpha_k\}$
Output: Approximation of the optimal policy $F_{\xi^*} \approx \pi_*$
1: repeat
2:     for $m = 1, \dots, M$ do
3:         Sample controller parameters $\theta^{(m)} \sim p_{\xi_k}$
4:         Sample trajectory $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)})\}_{t \geq 0}$ under policy $F_{\theta^{(m)}}$
5:     end for
6:     Approximate policy gradient $\nabla_\xi J(\xi_k) \approx \widehat{g}_{\text{PGPE}}$ using Eq. (4.31)
7:     Update hyperparameters using gradient ascent $\xi_{k+1} = \xi_k + \alpha_k \widehat{g}_{\text{PGPE}}$
8:     $k \leftarrow k + 1$
9: until converged
```

Independent Gaussian Parameter Distribution A simple approach is to assume that all the components of the parameter vector θ are independent and normally distributed with mean μ_i and variance σ_i^2, i.e. $\theta_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, the gradient with respect to the hyper-parameters $\xi = (\mu_1, \dots, \mu_{D_\theta}, \sigma_1, \dots, \sigma_{D_\theta})^T$ is given by

$$\begin{aligned} \frac{\partial \log p_\xi(\theta)}{\partial \mu_i} &= \frac{\theta_i - \mu_i}{\sigma_i^2} \\ \frac{\partial \log p_\xi(\theta)}{\partial \sigma_i} &= \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3} \end{aligned} \tag{4.32}$$

Using a constant learning rate $\alpha_i = \alpha \sigma_i^2$, the gradient updates takes the following form

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \left[G(h) - b\right] (\theta_i - \mu_i) \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \left[G(h) - b\right] \frac{(\theta_i - \mu_i)^2}{\sigma_i} \end{aligned} \tag{4.33}$$

where b can be computed as a moving average of the past returns. Intuitively, if $G(h) > b$ we adjust ξ so as to increase the probability of θ while if $G(h) < b$ we do the opposite.

Gaussian Parameter Distribution A more elaborate approach is to assume a generic dependence among the controller parameters, i.e. $\theta \sim \mathcal{N}(\mu, \Sigma)$. However, if we directly used the covariance matrix Σ as an hyper-parameter, it would be computationally difficult to enforce that it remains well-defined, i.e. symmetric and semidefinite positive, during the gradient ascent iterations. A simpler approach is to parameterize the distribution using the Cholesky factor Σ, i.e. the matrix C such that $\Sigma = C^T C$. This choice has two advantages: first, C makes explicit the $n(n+1)/2$ independent parameters determining the covariance matrix Σ; in addition, $C^T C$ is by construction a well-defined covariance matrix. Hence, the hyper-parameters are $\xi = \{\mu, C\}$. The likelihood score for this distribution doesn't have a simple expression, but it becomes extremely easy in the Natural version of PGPE which will be discussed in the next sections.

Symmetric Sampling and Gain Normalization In some settings, comparing the gain with a baseline can be misleading. In their original work, the authors propose a symmetric sampling technique similar to antithetic variates that further improves the convergence of the method. More in detail, a more robust gradient estimate can be obtained by measuring the difference in reward between two symmetric samples on either side of the current mean. That is, we sample a random perturbation $\epsilon \sim \mathcal{N}(0, \Sigma)$, where $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_{D_\theta}^2)$, and we define $\theta^+ = \mu + \epsilon$ and $\theta^- = \mu - \epsilon$. Denoting by G^+ (resp. G^-) the gains obtained on the trajectory associated to θ^+ (resp. θ^-), the objective function gradient can be approximated with

$$\begin{aligned} \nabla_{\mu_i} J(\xi) &\approx \frac{\epsilon_i (G^+ - G^-)}{2\sigma_i^2} \\ \nabla_{\sigma_i} J(\xi) &\approx \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i^3} \left(\frac{G^+ + G^-}{2} - b \right) \end{aligned} \tag{4.34}$$

Hence, by choosing $\alpha_i^k = 2\alpha\sigma_i^2$, we have the following update rules

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \epsilon_i (G^+ - G^-) \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i} \left(G^+ + G^- - 2b \right) \end{aligned} \tag{4.35}$$

In addition, the authors propose to normalize the gains in order to make the updates independent of the scale of the rewards. For instance, we could modify the hyperparameters updates as follows

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \epsilon_i \frac{(G^+ - G^-)}{2m - G^+ - G^-} \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i} \frac{(G^+ + G^- - 2b)}{m - b} \end{aligned} \tag{4.36}$$

where m might be the maximum gain the agent can receive, if known, or alternatively the maximum gain achieved so far. Symmetric sampling and gain normalization can drastically improve the gradient estimate quality and consequently the convergence time.

4.3.5.2 Infinite Horizon PGPE

While in the episodic PGPE the parameters θ are sampled only at the beginning of each episode, in *Infinite Horizon PGPE* (IHPGPE) [71] the parameters and learning are carried out simultaneously, while interacting with the environment. Let $0 < \varepsilon < 1$ the probability of updating the policy parameters, the parameters θ_t can be sampled consecutively as follows

$$p_\xi(\theta_{i,t+1}) = \varepsilon\mathcal{N}(\mu_{i,t}, \sigma_{i,t}^2) + (1-\varepsilon)\delta_{\theta_{i,t}} \tag{4.37}$$

In practice, ε should be chosen so that the expected frequency of changing a single parameter is coherent with the typical episode length in the episodic framework. Alternatively, one could sample all the parameters at a certain time step simultaneously

$$p_\xi(\theta_{t+1}) = \varepsilon\mathcal{N}(\mu_t, \Sigma_t) + (1-\varepsilon)\delta_{\theta_t} \tag{4.38}$$

This is equivalent to splitting the state-action space into artificial episodes. However, updating parameters asynchronously changes the policy only slightly thus introducing less noise in the process. Again, parameters can be updated at every time step by gradient ascent

$$\begin{aligned} \mu_{i,t+1} &= \mu_{i,t} + \alpha\left[G_t(h) - b\right](\theta_{i,t} - \mu_{i,t}) \\ \sigma_{i,t+1} &= \sigma_{i,t} + \alpha\left[G_t(h) - b\right]\frac{(\theta_{i,t} - \mu_{i,t})^2}{\sigma_{i,t}} \end{aligned} \tag{4.39}$$

Similarly to the episodic case, we can improve the gradient estimate by symmetric sampling and gain normalization.

4.4 Risk-Neutral Policy Gradient Theorem

In the previous sections, we saw that the core of a policy gradient algorithm consist in the approximation of the objective function gradient. In this section we present the *policy gradient theorem* [79], which shows that the gradient can be rewritten in a form suitable for estimation from experience aided by an approximate action-value or advantage function. First, we will prove the theorem in the risk-neutral setting for which it was originally proposed. In particular, we will see how the GPOMDP algorithm can be easily derived by this result via a Monte Carlo approximation. Moreover, we will derive a class of actor-critic algorithms [44] that, in addition to a parametric approximation of the policy, also exploit an approximation of the action-value function or of an advantage function to reduce the variance of the gradient estimate. In particular, we review the powerful idea of compatible function approximation [79] which assures the convergence to a local optimum of the objective function. Finally, we discuss the natural policy gradient idea [41], which forms the basis of many state-of-the-art algorithms. The extension to the risk-sensitive setting will be done in the next section.

4.4.1 Theorem Statement and Proof

Theorem 4.4.1 (Risk-Neutral Policy Gradient)**.** *Let π_θ be a differentiable policy. The policy gradient for the average reward formulation is given by*

$$\nabla_\theta \rho(\theta) = \mathbb{E}_{\substack{S\sim d^\theta \\ A\sim\pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S,A) Q_\theta(S,A)\right] \tag{4.40}$$

where d^θ is the stationary distribution of the Markov chain induced by π_θ. The policy gradient for the start value formulation is given by

$$\nabla_\theta J_{start}(\theta) = \mathbb{E}_{\substack{S\sim d^\theta_\gamma(s_0,\cdot) \\ A\sim\pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S,A) Q_\theta(S,A)\right] \tag{4.41}$$

where $d^\theta_\gamma(s_0,\cdot)$ is the γ-discounted visiting distribution over states starting from the initial state s_0 and following policy π_θ

$$d^\theta_\gamma(s,x) = \sum_{k=0}^{\infty} \gamma^k \mathcal{P}^{(k)}_\theta(s,x) \tag{4.42}$$

Proof. We first prove the result for the average-reward formulation and then for the start state formulation. From the basic relation between state-value function and action-value function, we have

$$\begin{aligned}
\nabla_\theta V_\theta(s) &= \nabla_\theta \int_{\mathbb{A}} \pi_\theta(s,a) Q_\theta(s,a) da \\
&= \int_{\mathbb{A}} \left[\nabla_\theta \pi_\theta(s,a) Q_\theta(s,a) + \pi_\theta(s,a) \nabla_\theta Q_\theta(s,a)\right] da
\end{aligned}$$

Using the Bellman expectation equation for Q_θ

$$\begin{aligned}
\nabla_\theta Q_\theta(s,a) &= \nabla_\theta \left[\mathcal{R}(s,a) - \rho_\theta + \int_{\mathbb{S}} \mathcal{P}(s,a,s') V_\theta(s') ds'\right] \\
&= -\nabla_\theta \rho_\theta + \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta V_\theta(s') ds'
\end{aligned}$$

Hence, plugging in the first equation

$$\nabla_\theta V_\theta(s) = \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s,a) Q_\theta(s,a) da - \nabla_\theta \rho_\theta + \int_{\mathbb{A}} \pi_\theta(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta V_\theta(s') ds'$$

Integrating both sides with respect to the stationary distribution d^θ and noting that, because of stationarity,

$$\int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta V(s') ds' da ds = \int_{\mathbb{S}} d^\theta(s) \nabla_\theta V_\theta(s) ds$$

we obtain the result

$$\begin{aligned}
\nabla_\theta \rho_\theta &= \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s,a) Q_\theta(s,a) da ds \\
&= \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) Q_\theta(s,a) da ds \\
&= \mathbb{E}_{\substack{S \sim d^\theta \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S,A) Q_\theta(S,A) \right]
\end{aligned}$$

Let us now prove the theorem for the start state formulation. The first step is exactly the same. Hence, using the Bellman expectation equation for Q_θ

$$\begin{aligned}
\nabla_\theta Q_\theta(s,a) &= \nabla_\theta \left[\mathcal{R}(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') V_\theta(s') ds' \right] \\
&= \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta V_\theta(s') ds'
\end{aligned}$$

we obtain

$$\begin{aligned}
\nabla_\theta V_\theta(s) &= \int_{\mathbb{A}} \left[\nabla_\theta \pi_\theta(s,a) Q_\theta(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta V_\theta(s') ds' \right] da \\
&= \int_{\mathbb{S}} \sum_{k=0}^{\infty} \gamma^k \mathcal{P}_\theta^{(k)}(s,x) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(x,a) Q_\theta(x,a) da dx
\end{aligned}$$

after unrolling $\nabla_\theta V_\theta$ infinite times and denoting by $\mathcal{P}_\theta^{(k)}(s,x)$ the probability of going from state s to state x in k steps under policy π_θ. Defining the γ-discounted visiting distribution of state x starting from state s as

$$d_\gamma^\theta(s,x) = \sum_{k=0}^{\infty} \gamma^k \mathcal{P}_\theta^{(k)}(s,x)$$

we have the result

$$\begin{aligned}
\nabla_\theta V_\theta(s) &= \int_{\mathbb{S}} d_\gamma^\theta(s,x) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(x,a) Q_\theta(x,a) da dx \\
&= \mathbb{E}_{\substack{S \sim d_\gamma^\theta(s_0,\cdot) \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S,A) Q_{\pi_\theta}(S,A) \right]
\end{aligned}$$

$\square$

The action-value function is typically unknown and needs to be approximated. As for the REINFORCE algorithm, we can subtract a state-dependent baseline from the action-value function without changing the value of the expectation. Indeed

$$\begin{aligned}\mathbb{E}_{\substack{S\sim d^\theta\\ A\sim\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(S,A)B_\theta(S)\right] &= \int_{\mathbb{S}} d^\theta(s)\int_{\mathbb{A}} \pi_\theta(s,a)\nabla_\theta \log\pi_\theta(s,a)B_\theta(s)dads\\ &= \int_{\mathbb{S}} d^\theta(s)B_\theta(s)\int_{\mathbb{A}} \nabla_\theta\pi_\theta(s,a)dads\\ &= \int_{\mathbb{S}} d^\theta(s)B_\theta(s)\nabla_\theta \underbrace{\int_{\mathbb{A}}\pi_\theta(s,a)da}_{=1}\,ds = 0\end{aligned}$$

Hence, we can rewrite the policy gradient for the average reward formulation as

$$\nabla_\theta\rho(\theta) = \mathbb{E}_{\substack{S\sim d^\theta\\ A\sim\pi_\theta}}\left[\nabla_\theta\log\pi_\theta(S,A)\left(Q_{\pi_\theta}(S,A) - B_\theta(S)\right)\right] \tag{4.43}$$

and similarly for the start state formulation. This result can be used as the starting point to derive several policy gradient methods that use different approximation of the action-value function.

4.4.2 GPOMDP

For an episodic MDP, the action-value function can be estimated with the total return obtained on a sample trajectory

$$Q_\theta(s_0,a_0) \approx \sum_{t=0}^{T^{(m)}} \gamma^t r_{t+1}^{(m)}$$

Combining this remark with a Monte Carlo approximation of Eq. (4.43), we obtain the GPOMDP gradient estimate (4.23). Therefore, the GPOMDP algorithm is a simple application of the policy gradient algorithm, which is much more general than the likelihood ratio technique used to derive the algorithm in the previous sections. In the next subsections we present other ways to exploit the policy gradient theorem to design efficient learning algorithms.

4.4.3 Actor-Critic Policy Gradient

A baseline should ideally measure the typical return obtained by an agent in a certain state when following a certain policy. Therefore, it becomes natural

Algorithm 4.4 Generic structure for an online actor-critic algorithm.

Input:
- Initial actor parameters θ_0,
- Initial critic parameters ψ_0,
- Learning rate $\{\alpha_k\}$

Output: Approximation of the optimal policy $\pi_{\theta^*} \approx \pi_*$

```
1: repeat
2:     Observe tuple $< s_k, a_k, r_{k+1}, s_{k+1} >$ sampled from the MDP.
3:     Update critic parameters $\psi_{k+1}$ using a value-based method.
4:     Estimate policy gradient as $\widehat{g}_k^{\mathrm{AC}} = \nabla_\theta \log \pi_{\theta_k}(s_k, a_k) \widehat{A}_{\psi_{k+1}}(s_k, a_k)$
5:     Update actor by gradient ascent $\theta_{k+1} = \theta_k + \alpha_k \widehat{g}_k^{\mathrm{AC}}$.
6:     $k \leftarrow k + 1$
7: until converged
```

to use the state-value function as a benchmark for the action-value function. Eq. (4.43) thus becomes

$$\nabla_\theta \rho(\theta) = \mathbb{E}_{\substack{S \sim d^\theta \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S, A) A_\theta(S, A) \right] \tag{4.44}$$

where we introduced the advantage function

$$A_\theta(s, a) = Q_\theta(s, a) - V_\theta(s) \tag{4.45}$$

which measures how good is to take action a in state s compared to simply following the policy π_θ. The advantage function is unknown and should estimated from samples of the MDP. *Actor-Critic* algorithms consist in all those methods that employ an approximation of the advantage function or of the value function, also known as critic, to estimate the policy gradient. These methods thus maintain two sets of parameters: a *critic* that updates the action-value function parameters ψ and an *actor* that updates the policy parameters θ in the direction suggested by the critic. The general structure for an online actor-critic algorithm is reported in Algorithm 4.4.

There are two possible approaches to estimate the advantage function: the first one is to estimate both the state-value function $\widehat{V}_\psi \approx V_\theta$ and the action-value function $\widehat{Q}_\xi \approx Q_\theta$ and derive an estimate for the advantage function $\widehat{A}_{\psi,\xi} = \widehat{Q}_\xi - \widehat{V}_\psi \approx A_\theta$. The second approach directly approximates the advantage function $\widehat{A}_\psi \approx A_\theta$ and is usually preferred, since it allows us to maintain only one critic. Actor-critic algorithms typically use a temporal-difference algorithm to update an approximation of the value function $\widehat{V}_\psi \approx V_\theta$, from which we can derive an approximation for the advantage function.

Assume that the true state-value function V_θ is given. Then the TD error

$$\delta_\theta = R - \rho_\theta + V_\theta(S') - V_\theta(S) \tag{4.46}$$

is an unbiased estimate of the advantage function. Indeed

$$\begin{aligned}\mathbb{E}\left[\delta_\theta | S = s, A = a\right] &= \mathbb{E}\left[R - \rho_\theta + V_\theta(S') | S = s, A = a\right] - V_\theta(s)\\ &= Q_\theta(s,a) - V_\theta(s)\\ &= A_\theta(s,a)\end{aligned}$$

By a simple conditioning argument, the policy gradient can be rewritten as

$$\nabla_\theta \rho(\theta) = \mathbb{E}_{\substack{S\sim d^\theta\\ A\sim\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(S,A)\delta_\theta\right] \tag{4.47}$$

In practice, we can use an approximate TD error

$$\widehat{\delta}_k = r_{k+1} - \widehat{\rho}_k + \widehat{V}_{\psi_k}(s_{k+1}) - \widehat{V}_{\psi_k}(s_k) \tag{4.48}$$

where $\widehat{\rho}_k$ is an estimate of the average reward. Hence, we can easily obtain an approximation of the policy gradient by

$$\widehat{g}^k_{\text{TD}(0)} = \nabla_\theta \log \pi_\theta(s_k, a_k)\widehat{\delta}_k \tag{4.49}$$

which can be used to update the critic parameter in the gradient ascent direction. On the other hand, the critic parameters can be updated using a TD(0) temporal difference scheme

$$\psi_{k+1} = \psi_k + \alpha_k \widehat{\delta}_k \nabla_\psi \widehat{V}_{\psi_k}(s_k) \tag{4.50}$$

The discussion can be easily extended to more complex value-based methods for estimating the critic, such as the backward-view TD(λ). This method employs eligibility traces to assign credit for the rewards obtained by the agent to all previous states and actions. More formally, the policy parameters update rule becomes

$$\theta_{k+1} = \theta_k + \alpha_k \widehat{\delta}_k e_k \tag{4.51}$$

where the eligibility trace e_k is defined by the following recursive equation

$$e_{k+1} = \lambda e_k + \nabla_\theta \log \pi_{\theta_k}(s_k, a_k) \tag{4.52}$$

$0 \leq \lambda \leq 1$ is a parameter that manages the amount of bootstrap: for $\lambda = 0$ the method is the standard temporal difference scheme while for $\lambda = 1$ the method is equivalent to Monte Carlo. The complete scheme is reported in Algorithm 4.5.

Algorithm 4.5 TD(λ) policy gradient algorithm.

```
Input:
    • Initial actor parameters $\theta_0$,
    • Initial critic parameters $\psi_0$,
    • Learning rate $\{\alpha_k\}$
Output: Approximation of the optimal policy $\pi_{\theta^*} \approx \pi_*$
 1: Initialize $k = 0$ and the eligibility trace $e_{-1} = 0$
 2: repeat
 3:     Observe tuple $< s_k, a_k, r_{k+1}, s_{k+1} >$ sampled from the MDP.
 4:     Compute TD error $\widehat{\delta}_k = r_{k+1} - \widehat{\rho}_k + \widehat{V}_{\psi_k}(s_{k+1}) - \widehat{V}_{\psi_k}(s_k)$
 5:     Update critic parameters $\psi_{k+1} = \psi_k + \alpha_k \widehat{\delta}_k \nabla_\psi \widehat{V}_{\psi_k}(s_k)$
 6:     Update eligibility trace $e_{k+1} = \lambda e_k + \nabla_\theta \log \pi_{\theta_k}(s_k, a_k)$
 7:     Update actor parameters $\theta_{k+1} = \theta_k + \alpha_k \widehat{\delta}_k e_k$.
 8:     $k \leftarrow k + 1$
 9: until converged
```

4.4.4 Compatible Function Approximation

In the previous sections we saw that a critic may reduce the variance of thew policy gradient estimate. However, this is achieved at the cost of introducing a bias in the approximation which, in certain cases, may endanger the convergence of the method to a good solution. Luckily, the *compatible function approximation* [79] avoids introducing any bias by a careful choice of the critic. This technique consists in estimating the advantage function with a linear regression on a family of suitable basis functions consisting of the gradient of the likelihood score, i.e.

$$\widehat{A}_\theta(s, a) = \psi^T \nabla_\theta \log \pi_\theta(s, a) \tag{4.53}$$

The power of these basis functions becomes apparent in the following theorem

Theorem 4.4.2 (Compatible Function Approximation). *If*

i) the advantage function approximator is compatible to the policy, i.e.

$$\nabla_\psi A_\psi(s, a) = \nabla_\theta \log \pi_\theta(s, a) \tag{4.54}$$

ii) the advantage function parameters ψ minimize the mean square error

$$\theta = \arg\min_\psi \mathbb{E}_{\pi_\theta}\left[\left(A_{\pi_\theta}(S, A) - \widehat{A}_\psi(S, A)\right)^2\right] \tag{4.55}$$

Then the policy gradient is exact

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(S, A) \widehat{A}_\psi(S, A)\right] \tag{4.56}$$

Hence, the true advantage function in the policy gradient formula can be replaced by the compatible function approximation without changing the policy gradient value. A linear regression may seem a too simple representation of the advantage function. However, one must bear in mind that this approximation is only used to reduce the variance of the policy gradient estimate and should not be expected to provide an accurate representation of the true-value function. This technique has proved useful in many application.

4.4.5 Natural Policy Gradient

Despite all the advances in the variance reduction techniques, these methods still tend to perform surprisingly poorly. Even when applied to simple examples with rather few states, where the gradient can be determined very accurately, they turn out to be quite inefficient. Typically, one of the reasons of this behavior is the presence of large plateaus in the expected return landscape where the gradients are small and often do not point directly towards the optimal solution. In this context, the steepest ascent with respect to the Fisher information metric, called the *natural policy gradient*, turns out to be significantly more efficient than normal gradients for such plateaus. This technique was first proposed in the reinforcement learning setting in [41] and later applied to actor-critic algorithms in in [64]. The following properties of natural policy gradient make it one of the more reliable policy-based methods,

i) Convergence to a local minimum is guaranteed.
ii) By choosing a more direct path to the optimal solution in parameter space, the natural gradient typically has faster convergence and avoids premature convergence of "vanilla gradients".
iii) The natural policy gradient can be shown to be covariant, i.e. independent of the coordinate frame chosen for expressing the policy parameters.
iv) As the natural gradient analytically averages out the influence of the stochastic policy (including the baseline of the function approximator), it requires fewer data points for a good gradient estimate than "vanilla gradients".

4.4.5.1 Formalism of Natural Policy Gradients

Policy gradient methods improve the policy π_θ by iteratively applying "small" changes $\Delta\theta$ to the policy parameters θ. However, the meaning of "small" is ambiguous. For instance, when working with an Euclidean metric, the size of this update $\|\Delta\theta\| = \sqrt{\Delta\theta^T \Delta\theta}$ and therefore the update size depends on

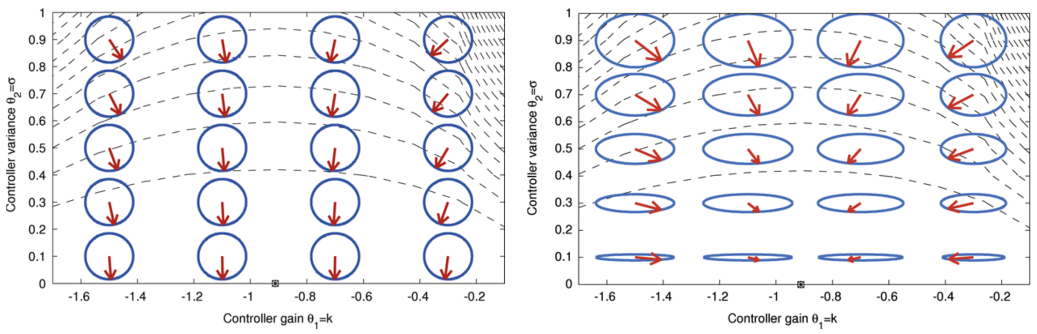

Figure 4.1: "Vanilla" policy gradient (left) vs. natural policy gradient (right). The main difference is how the two approaches punish the change in parameters. This distance is indicated by the blue ellipses in the contour plot while the dashed lines show the expected return. Obtaining a gradient then corresponds to finding a vector pointing from the center of the ellipses to the location with maximum expected return on the ellipse. A vanilla policy gradient (a) considers a change in all parameters as equally distant, thus, it is a search for a maximum on a circle while the natural gradient (b) uses scales determined by the Fisher information which results in a reduction in exploration. The slower reduction in exploration results into a faster convergence to the optimal policy [64].

the parameterization of the policy, which often results in unnaturally slow learning even if higher-order gradient methods were employed. This problem poses the question whether we can achieve a covariant gradient descent, i.e. a gradient descent with respect to an invariant measure of the closeness between the current policy and the updated policy based upon the distribution of the paths generated by each of these. Standard measures of distance between probability distributions are the Kullbach-Leibler divergence $d_{\mathrm{KL}}(p_\theta(h) \mid\mid p_{\theta+\Delta\theta}(h))$ and the Hellinger distance. These two distances can be approximation in first instance by a second-order Taylor expansion

$$d_{\mathrm{KL}}(p_\theta(h) \mid\mid p_{\theta+\Delta\theta}(h)) \approx \frac{1}{2}\Delta\theta^T F_\theta \Delta\theta$$

where F_θ is the Fischer information matrix

$$\begin{aligned} F_\theta &= \int_{\mathbb{H}} p_\theta(h)\nabla_\theta \log p_\theta(h) \nabla_\theta \log p_\theta(h)^T dh \\ &= \mathbb{E}_{H\sim p_\theta}\left[\nabla_\theta \log p_\theta(H)\nabla_\theta \log p_\theta(H)^T\right] \end{aligned} \tag{4.57}$$

The goal is to find the optimal update $\Delta\theta$ for the policy parameters so as to maximize the objective function, under the constraint that the new policy must be in a radius ϵ from the previous policy with respect to the Kullbach-Leibler divergence, i.e.

$$\begin{cases} \max_{\Delta\theta} J(\theta + \Delta\theta) \approx J(\theta) + \Delta\theta^T \nabla_\theta J(\theta) \\ \text{s.t. } d_{\text{KL}}(p_\theta(h) \mid\mid p_{\theta+\Delta\theta}(h)) \approx \frac{1}{2}\Delta\theta^T F_\theta \Delta\theta < \epsilon \end{cases}$$

The optimal solution is given by

$$\Delta\theta = \alpha_n F_\theta^{-1} \nabla_\theta J(\theta)$$

with

$$\alpha_n = \sqrt{\frac{\epsilon}{\nabla_\theta J(\theta)^T F_\theta^{-1} \nabla_\theta J(\theta)}}$$

The direction $\widetilde{\nabla}_\theta J(\theta) = \Delta\theta/\alpha_n$ is called the natural gradient and learning algorithms that use this gradient instead of the standard one are called natural policy gradient algorithms. The strongest theoretical advantage of this approach is that its performance no longer depends on the parameterization of the policy and it is therefore safe to use for arbitrary policies. In practice, the learning process converges significantly faster in most practical cases and requires less manual parameter tuning of the learning algorithm. The only remaining point to discuss is how to compute the inverse of the Fischer information matrix. This is not an easy task and the matrix will usually need to be estimated from sample trajectories. However, we will see that this matrix can be computed analytically after reformulating the policy gradient theorem for the parameter-based search methods such as PGPE. The formal derivation will be presented in the last section of this chapter and this result will lead to the *Natural PGPE* (NPGPE) algorithm [53].

4.5 Risk-Sensitive Policy Gradient Theorem

In this section the policy gradient theorem is extended to the risk-sensitive framework. In the average reward formulation of the control problem, the theorem and its derivation are analogous to those presented in the last section for the risk-neutral framework. This will allow us to derive in a trivial way the risk-sensitive versions of all the learning algorithms seen above. The risk-sensitive policy gradient theorem for the average-reward formulation was first derived in [65] and the presentation of this section closely follows the original article. Obtaining an equivalent theorem for the discounted reward formulation is more challenging. In the article cited above, the authors prove the theorem under a very strong assumption on the dependence of rewards obtained at different time steps which is not verified in many applications, among which is the asset allocation problem that we will consider in the next chapter. We will discuss the problems arising in the discounted setting and why it is not easy to derive a policy gradient theorem in this case. In the original article the authors mostly considered the mean-variance criterion since all the algorithms they propose are easily adapted to the Sharpe ratio criterion. Here we take the opposite direction and present the algorithms for the Sharpe ratio, referring to their article for the mean-variance counterparts. Let us consider a family of parametrized policies π_θ, with $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$. The optimization problem then becomes

$$\max_\theta J(\theta) = \mathrm{Sh}(\theta) = \frac{\rho(\theta)}{\sqrt{\Lambda(\theta)}} \tag{4.58}$$

where we denoted by $\rho(\theta) = \rho_{\pi_\theta}$ and similarly for the other quantities. Using a policy gradient approach, the policy parameters are updated following the gradient ascent direction.

4.5.1 Average Reward Formulation

In the average reward formulation the gradient of the Sharpe ratio is

$$\nabla_\theta J(\theta) = \frac{\eta(\theta)\nabla_\theta\rho(\theta) - \frac{1}{2}\rho(\theta)\nabla_\theta\eta(\theta)}{\Lambda(\theta)\sqrt{\Lambda(\theta)}} \tag{4.59}$$

Hence, to compute the update direction, it is sufficient to estimate the various quantities appearing in this formula. For instance, the average reward $\rho(\theta)$, the average square reward $\eta(\theta)$ and the reward variance $\Lambda(\theta)$ can be approximated using exponentially weighted moving averages. On the other hand, the gradient of the average reward $\nabla_\theta\rho(\theta)$ is given by the standard policy gradient theorem 4.4.1. The only term remaining is the gradient of the average square reward $\nabla_\theta\eta(\theta)$, which is provided by the risk-sensitive policy gradient theorem

Theorem 4.5.1 (Risk-Sensitive Policy Gradient). *Let π_θ be a differentiable policy. The policy gradient for the average square reward is given by*

$$\nabla_\theta\eta(\theta) = \mathbb{E}_{\substack{S\sim d_\theta \\ A\sim\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(S,A) W_\theta(S,A)\right] \tag{4.60}$$

where d^θ is the stationary distribution of the Markov chain induced by π_θ.

Proof. The proof is analogous to that of the risk-neutral version of theorem. From the basic relation between state-value function and action-value function, we have

$$\begin{aligned}\nabla_\theta U_\theta(s) &= \nabla_\theta \int_{\mathbb{A}} \pi_\theta(s,a) W_\theta(s,a) da \\ &= \int_{\mathbb{A}} \left[\nabla_\theta \pi_\theta(s,a) W_\theta(s,a) + \pi_\theta(s,a) \nabla_\theta W_\theta(s,a)\right] da\end{aligned}$$

Using the Bellman expectation equation for W_θ

$$\begin{aligned}\nabla_\theta W_\theta(s,a) &= \nabla_\theta \left[\mathcal{M}(s,a) - \eta_\theta + \int_{\mathbb{S}} \mathcal{P}(s,a,s') U_\theta(s') ds'\right] \\ &= -\nabla_\theta \eta_\theta + \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta U_\theta(s') ds'\end{aligned}$$

Hence, plugging in the first equation

$$\nabla_\theta U_\theta(s) = \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s,a) W_\theta(s,a) da - \nabla_\theta \eta_\theta + \int_{\mathbb{A}} \pi_\theta(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta U_\theta(s') ds'$$

Integrating both sides with respect to the stationary distribution d^θ and noting that, because of stationarity,

$$\int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s') \nabla_\theta U(s') ds' da ds = \int_{\mathbb{S}} d^\theta(s) \nabla_\theta U_\theta(s) ds$$

we obtain the result

$$\begin{aligned}\nabla_\theta \eta_\theta &= \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s,a) W_\theta(s,a) da ds \\ &= \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) W_\theta(s,a) da ds \\ &= \mathbb{E}_{\substack{S\sim d^\theta \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S,A) W_\theta(S,A)\right]\end{aligned}$$

□

The result is identical to that for the average reward, provided that we replace the action-value function $Q_\theta(s,a)$ by the square action-value function $W_\theta(s,a)$. As in the standard risk-neutral case, a state-dependent baseline can be introduced in the gradient without changing the result. In particular, by using the average-adjusted square value function as baseline, we can replace the average adjusted action-value functions with the following square advantage function

$$B_\theta(s,a) = W_\theta(s,a) - U_\theta(s) \tag{4.61}$$

Thus, the gradients can be written as

$$\nabla_\theta \eta(\theta) = \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi}} \left[\nabla_\theta \log \pi_\theta(S, A) B_\theta(S, A)\right] \tag{4.62}$$

From this result, following the exact same reasoning used in the risk-neutral framework, we can design a variety of risk-sensitive actor-critic algorithms, which employ an approximation of the standard and of the square advantage functions to obtain a more accurate estimate of the objective function.

4.5.2 Risk-Sensitive Actor-Critic Algorithm

In [65], starting from Eqs. (4.44) and (4.62), the authors propose a TD(0) risk-sensitive actor-critic algorithm for the average reward setting. The algorithm maintains two critics that estimate the average adjusted value-functions $V_\theta(S)$ and $U_\theta(S)$ respectively and are updated via a TD(0) temporal difference scheme. Let δ_n^A and δ_n^B be the TD errors for residual value and square value functions

$$\begin{aligned} \delta_t^A &= R_{t+1} - \widehat{\rho}_{t+1} + \widehat{V}(S_{t+1}) - \widehat{V}(S_t) \\ \delta_t^B &= R_{t+1}^2 - \widehat{\eta}_{t+1} + \widehat{U}(S_{t+1}) - \widehat{U}(S_t) \end{aligned} \tag{4.63}$$

where $\widehat{V}$, $\widehat{U}$, $\widehat{\rho}$ and $\widehat{\eta}$ are unbiased estimate of V_θ, U_θ, $\rho(\theta)$ and $\eta(\theta)$ respectively. It is easy to show that δ_t^A and δ_t^B are unbiased estimates of the advantage functions.

$$\begin{aligned} \mathbb{E}_\theta\left[\delta_t^A | S_t = s, A_t = a\right] &= A_\theta(s, a) \\ \mathbb{E}_\theta\left[\delta_t^B | S_t = s, A_t = a\right] &= B_\theta(s, a) \end{aligned}$$

An estimate of the gradients can be obtained by replacing the advantage functions with the TD errors

$$\begin{aligned} \nabla_\theta \rho(\theta) &\approx \nabla_\theta \log \pi_\theta(S_t, A_t)\delta_t^A \\ \nabla_\theta \eta(\theta) &\approx \nabla_\theta \log \pi_\theta(S_t, A_t)\delta_t^B \end{aligned} \tag{4.64}$$

The value functions are linearly approximated using some veatures vectors $\Phi_V : \mathbb{S} \to \mathbb{R}^{D_V}$ and $\Phi_U : \mathbb{S} \to \mathbb{R}^{D_U}$ as follows

$$\begin{aligned} \widehat{V}(s) &= \psi_V^T \Phi_V(s) \\ \widehat{U}(s) &= \psi_U^T \Phi_U(s) \end{aligned} \tag{4.65}$$

Combining all these ingredients lead to the Risk-Sensitive Average Reward Actor-Critic algorithm (RSARAC), the pseudocode of which is reported in

Algorithm 4.6 Risk-Sensitive Average Reward Actor-Critic algorithm

```
Input:
    • Initial actor parameters $\theta^0$
    • Initial critics parameters $\psi_V^0$ and $\psi_U^0$
    • Actor learning rate $\{\alpha_k\}$
    • Critics learning rate $\{\beta_k\}$
    • Averages learning rate $\{\gamma_k\}$
Output: Approximation of the optimal policy $\pi_{\theta^*} \approx \pi_*$
1: repeat
2:     Observe tuple $< s_k, a_k, r_{k+1}, s_{k+1} >$ sampled from the MDP.
3:     Update averages
           $\widehat{\rho}_{k+1} = (1-\gamma_k)\widehat{\rho}_k + \gamma_k r_{k+1}$
           $\widehat{\eta}_{k+1} = (1-\gamma_k)\widehat{\rho}_k + \gamma_k r_{k+1}^2$
4:     Compute TD errors
           $\delta_k^A = r_{k+1} - \widehat{\rho}_{k+1} + (\psi_V^k)^T \Phi_V(s_{k+1}) - (\psi_V^k)^T \Phi_V(s_k)$
           $\delta_k^B = r_{k+1}^2 - \widehat{\eta}_{k+1} + (\psi_U^k)^T \Phi_U(s_{k+1}) - (\psi_U^k)^T \Phi_U(s_k)$
5:     Update critic parameters
           $\psi_V^{k+1} = \psi_V^k + \beta_k \delta_k^A \Phi_V(s_k)$
           $\psi_U^{k+1} = \psi_U^k + \beta_k \delta_k^B \Phi_U(s_k)$
6:     Update actor parameters $\theta^{k+1} = \theta^k + \alpha_k \widehat{\mathrm{Sh}}(\theta_k)$.
7:     $k \leftarrow k+1$
8: until converged
```

Algorithm 4.6. Let us notice that the algorithm is a three time-scale stochastic approximation algorithm, where the learning rates, in addition to the usual Robbins-Monro conditions, should satisfy

$$\alpha_k < \beta_k < \gamma_k$$

4.5.3 Discounted Reward Formulation

In the discounted reward formulation, we need to adapt the definition of the Sharpe ratio associated to a policy. Let us suppose that the system always start in the same initial state s_0, then we can introduce the start-state Sharpe ratio that can be achieved following policy π_θ as

$$\mathrm{Sh}(\theta) = \frac{V_\theta(s_0)}{\sqrt{\Lambda_\theta(s_0)}} \tag{4.66}$$

where V_θ and Λ_θ are the state-value function and the variance-function introduced in Chapter 2. Hence, the gradient of the Sharpe ratio is given by

$$\nabla_\theta \mathrm{Sh}(\theta) = \frac{U_\theta(s_0)\nabla_\theta V_\theta(s_0) - \frac{1}{2}V_\theta(s_0)\nabla_\theta U_\theta(s_0)}{\Lambda_\theta(s_0)\sqrt{\Lambda_\theta(s_0)}} \tag{4.67}$$

The gradient of the state-value function $\nabla_\theta V_\theta(s_0)$ is given by the risk-neutral policy gradient theorem. Therefore, in order to approximate the gradient of the Sharpe ratio, we need to estimate the value functions $V_\theta(s_0)$, $U_\theta(s_0)$, the variance $\Lambda_\theta(s_0)$, and the gradient of the square state-value function $U_\theta(s_0)$. The first three terms might be easily approximated using moving averages. On the other hand, estimating $\nabla_\theta U_\theta(s_0)$ in the same spirit of the policy gradient theorem is much more delicate.

Theorem 4.5.2 (Risk-Sensitive Policy Gradient)**.** *Let π_θ be a differentiable policy. The policy gradient for the square state-value function is given by*

$$\begin{aligned}\nabla_\theta U_\theta(s_0) = \mathbb{E}_{\substack{S\sim d^\theta_{\gamma^2}(s_0,\cdot)\\ A\sim\pi_\theta(S,\cdot)\\ S'\sim\mathcal{P}(S,A,\cdot)}}&\left[\nabla_\theta \log\pi_\theta(S,A)W_\theta(S,A)\right.\\ &\left.+ 2\gamma\mathcal{R}(S,A)\nabla_\theta V_\theta(S') + 2\gamma C_\theta(S,A)\right]\end{aligned} \tag{4.68}$$

where $d^\theta_{\gamma^2}(s_0,\cdot)$ is the γ^2-discounted visiting distribution over states starting from the initial state s_0 and following policy π_θ

$$d^\theta_{\gamma^2}(s_0,x) = \sum_{k=0}^{\infty}\gamma^{2k}\mathcal{P}^{(k)}_\theta(s_0,x) \tag{4.69}$$

Proof. From the basic relation between square state-value function and the square action-value function, we have

$$\begin{aligned}\nabla_\theta U_\theta(s) &= \nabla_\theta \int_{\mathbb{A}} \pi_\theta(s,a)W_\theta(s,a)da\\ &= \int_{\mathbb{A}} \left[\nabla_\theta\pi_\theta(s,a)W_\theta(s,a) + \pi_\theta(s,a)\nabla_\theta W_\theta(s,a)\right]da\end{aligned}$$

Hence, using the Bellman expectation equation for W_θ

$$\begin{aligned}\nabla_\theta W_\theta(s,a) &= \nabla_\theta\left[\mathcal{M}(s,a) + 2\gamma\mathcal{R}(s,a)T_aV_\pi(s) + 2\gamma C_\theta(s,a) + \gamma^2 T_aU_\theta(s)\right]\\ &= 2\gamma\mathcal{R}(s,a)\nabla_\theta T_aV_\theta(s) + 2\gamma\nabla_\theta C_\theta(s,a) + \gamma^2\nabla_\theta T_aU_\theta(s)\\ &= \int_{\mathbb{S}}\mathcal{P}(s,a,s')\left[2\gamma\mathcal{R}(s,a)\nabla_\theta V_\theta(s') + 2\gamma\nabla_\theta C_\theta(s,a) + \nabla_\theta U_\theta(s')\right]ds'\end{aligned}$$

where we assumed to be able to exchange the gradient and the integral. Plugging in the first equation and exploiting the fact that $\int_{\mathbb{S}} \mathcal{P}(s,a,s')ds' = 1$, we obtain

$$\begin{aligned}\nabla_\theta U_\theta(s) = \int_{\mathbb{A}} \pi_\theta(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s')[&\nabla_\theta \log \pi_\theta(s,a) W_\theta(s,a) \\ &+ 2\gamma \mathcal{R}(s,a)\nabla_\theta V_\theta(s') + 2\gamma \nabla_\theta C_\theta(s,a) + \nabla_\theta U_\theta(s')]ds'da\end{aligned}$$

Unrolling $\nabla_\theta U_\theta$ infinite times and denoting by $\mathcal{P}_\theta^{(k)}(s,x)$ the probability of going from state s to state x in k steps under policy π_θ, we obtain

$$\begin{aligned}\nabla_\theta U_\theta(s) = \sum_{k=0}^{\infty} \gamma^{2k} \int_{\mathbb{S}} \mathcal{P}_\theta^{(k)}(s,x) \int_{\mathbb{A}} \pi_\theta(x,a) \int_{\mathbb{S}} \mathcal{P}(x,a,x')[&\nabla_\theta \log \pi_\theta(x,a) W_\theta(x,a) \\ &+ 2\gamma \mathcal{R}(x,a)\nabla_\theta V_\theta(x') + 2\gamma \nabla_\theta C_\theta(x,a)]dx'dadx\end{aligned}$$

Defining the γ^2-discounted visiting distribution of state x starting from state s as

$$d_{\gamma^2}^\theta(s,x) = \sum_{k=0}^{\infty} \gamma^{2k} \mathcal{P}_\theta^{(k)}(s,x)$$

we have the result

$$\begin{aligned}\nabla_\theta U_\theta(s) = \int_{\mathbb{S}} d_{\gamma^2}^\theta(s,x) \int_{\mathbb{A}} \pi_\theta(x,a) \int_{\mathbb{S}} \mathcal{P}(x,a,x')[&\nabla_\theta \log \pi_\theta(x,a) W_\theta(x,a) \\ &+ 2\gamma \mathcal{R}(x,a)\nabla_\theta V_\theta(x') + 2\gamma \nabla_\theta C_\theta(x,a)]dx'dadx\end{aligned}$$

$\square$

Compared to risk-sensitive policy gradient theorem in the average reward formulation, we have two additional terms: one term depending on the gradient of the state-value function at the next state and another term depending on the covariance between the one-step reward and the successive return. These terms are very difficult to approximate online in a continuing environment. Therefore, the result is of practical interest only for an episodic environment where the experiments have a finite (possible random) lifespan. Since the application we considered does not fall in this category, we won't discuss any risk-sensitive algorithm for the discounted formulation.

4.6 Parameter-Based Policy Gradient

The learning algorithms derived from the policy gradient theorems look for a set of optimal controller parameters by perturbing the control signal. Indeed, the policy gradient theorem holds only for stochastic policies, which explore in the action space. However, a significant problem with this sampling strategy is that the high variance in their gradient estimates leads to slow convergence. On the other hand, parameter-based algorithms such as PGPE look for an optimal solution to the control problem by directly perturbing the controller parameters. These algorithm thus employ a deterministic controller and explore in the parameters space. However, in its original version, PGPE was conceived for episodic environments and extended to continuing environment by artificially truncating the experiments lifespan. In this section we propose a parameter-based version of the policy gradient theorem, which can be used to derive online learning algorithms. The following discussion starts from and generalizes the results presented in [53], where the authors propose an online natural PGPE algorithm. Given the difficulties to extend the policy gradient theorem to the risk-sensitive discounted formulation, here we consider only the averge reward formulation.

4.6.1 Risk-Neutral Setting

Let us consider a deterministic controller $F : \mathbb{S} \times \Theta \to \mathbb{A}$ that, given a set of parameters $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$, maps a state $s \in \mathbb{S}$ to an action $a = F(s;\theta) = F_\theta(s) \in \mathbb{A}$. The policy parameters are drawn from a probability distribution p_ξ, with hyper-parameters $\xi \in \Xi \subseteq \mathbb{R}^{D_\xi}$. Combining these two hypotheses, the agent follows a stochastic policy π_ξ defined by

$$\forall B \in \mathcal{A},\ \pi_\xi(s,B) = \pi(s,B;\xi) = \int_\Theta p_\xi(\theta) \mathbb{1}_{F_\theta(s)\in B} d\theta \tag{4.70}$$

In the risk-neutral setting, the goal of the agent is to find a set of hyper-parameters ξ_* that maximizes the average reward obtained over a single time-step

$$\xi_* = \arg\max_\xi \rho(\xi)$$

Following the policy gradient approach, we iteratively update the parameters in the gradient ascent direction

$$\xi_{k+1} = \xi_k + \alpha_k \nabla_\xi \rho(\xi_k)$$

Theorem 4.6.1 (Risk-Neutral Parameter-Based Policy Gradient). *Let p_ξ be differentiable with respect to ξ, then the gradient of the average reward is given by*

$$\nabla_\xi \rho(\xi) = \mathbb{E}_{\substack{S\sim d^\xi \\ \theta\sim p_\xi}} \left[\nabla_\xi \log p_\xi(\theta) Q_{\pi_\xi}(S,\theta)\right] \tag{4.71}$$

where we denoted $Q_\xi(S,\theta) = Q_\xi(S, F_\theta(S))$.

Proof. Thanks to the likelihood ratio technique, we have

$$\begin{aligned}\nabla_\xi \pi_\xi(s,a) &= \int_\Theta \nabla_\xi p_\xi(\theta) \mathbb{1}_{F_\theta(s)=a} d\theta \\ &= \int_\Theta p_\xi(\theta) \nabla_\xi \log p_\xi(\theta) \mathbb{1}_{F_\theta(s)=a} d\theta \\ &= \mathbb{E}_{\theta\sim p_\xi}\left[\nabla_\xi \log p_\xi(\theta) \mathbb{1}_{F_\theta(s)=a}\right]\end{aligned}$$

hence, the policy π_ξ is differentiable with respect to ξ and the standard policy gradient theorem holds

$$\begin{aligned}\nabla_\xi \rho(\xi) &= \mathbb{E}_{\substack{S\sim d^\xi \\ A\sim\pi_\xi}}\left[\nabla_\theta \log \pi_\theta(S,A) Q_\theta(S,A)\right] \\ &= \int_{\mathbb{S}} d^\xi(s) \int_{\mathbb{A}} \pi_\xi(s,a) \nabla_\xi \log \pi_\xi(s,a) Q_{\pi_\xi}(s,a) da ds \\ &= \int_{\mathbb{S}} d^\xi(s) \int_{\mathbb{A}} \nabla_\xi \pi_\xi(s,a) Q_{\pi_\xi}(s,a) da ds\end{aligned}$$

Plugging the first equation in the inner integral and exchanging the integral over the action space with the expectation yields

$$\begin{aligned}\int_{\mathbb{A}} \nabla_\xi \pi_\xi(s,a) Q_{\pi_\xi}(s,a) da &= \mathbb{E}_{\theta\sim p_\xi}\left[\nabla_\xi \log p_\xi(\theta) \int_{\mathbb{A}} \mathbb{1}_{F_\theta(s)=a} Q_{\pi_\xi}(s,a) da\right] \\ &= \mathbb{E}_{\theta\sim p_\xi}\left[\nabla_\xi \log p_\xi(\theta) Q_{\pi_\xi}(s, F_\theta(s))\right] \\ &= \mathbb{E}_{\theta\sim p_\xi}\left[\nabla_\xi \log p_\xi(\theta) Q_{\pi_\xi}(s, \theta))\right]\end{aligned}$$

Finally, plugging this equation in the policy gradient theorem, we obtain the result

$$\nabla_\xi \rho(\xi) = \mathbb{E}_{\substack{S\sim d^\xi \\ \theta\sim p_\xi}}\left[\nabla_\xi \log p_\xi(\theta) Q_{\pi_\xi}(S,\theta)\right]$$

□

This expression is very similar to the original policy gradient theorem, but the expectation is taken over the controller parameters instead of the action space and we have the likelihood score the controller parameters distribution instead of that of the stochastic policy. Thus, we might interpret this result as if the agent directly selected the parameters θ according to a policy p_ξ, which then lead to an action through the deterministic mapping F_θ. Therefore, it is as if the agent's policy was in the parameters space and not in the control space. As in the standard policy gradient methods, we can add a state-dependent baseline $B_\xi(S)$ to the gradient without increasing the bias

$$\nabla_\xi \rho(\xi) = \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta) \left(Q_{\pi_\xi}(S,\theta) - B_\xi(S)\right)\right] \tag{4.72}$$

Algorithm 4.7 Actor-Critic PGPE

```
Input:
    • Deterministic controller $F : \mathbb{S} \times \Theta \to \mathbb{A}$
    • Initial hyper-parameters $\xi_0$
    • Initial critic parameters $\psi_0$
    • Learning rate $\{\alpha_k\}$
    • Momentum parameter $\lambda$
Output: Approximation of the optimal hyper-parameters $\xi_*$
 1: Initialize $k = 0$, average reward $\widehat{\rho}_0 = 0$ and eligibility trace $e_{-1} = 0$
 2: repeat
 3:     Observe current state $s_k$
 4:     Simulate controller parameters $\theta_k \sim p_{\xi_k}$
 5:     Perform action $a_k = F_{\theta_k}(s_k)$ and receive reward $r_{k+1}$
 6:     Update average reward estimate $\widehat{\rho}_{k+1} = \widehat{\rho}_k + \alpha_k(r_{k+1} - \widehat{\rho}_k)$
 7:     Compute TD error $\widehat{\delta}_k = r_{k+1} - \widehat{\rho}_{k+1} + \widehat{V}_{\psi_k}(s_{k+1}) - \widehat{V}_{\psi_k}(s_k)$
 8:     Update critic parameters $\psi_{k+1} = \psi_k + \alpha_k \widehat{\delta}_k \nabla_\psi \widehat{V}_{\psi_k}(s_k)$ .
 9:     Update eligibility trace $e_k = \lambda e_{k-1} + \nabla_\xi \log p_\xi(\theta_k)$
10:     Update hyper-parameters $\xi_{k+1} = \xi_k + \alpha_k \widehat{\delta}^k_{\psi_k} e_k$.
11:     $k \leftarrow k + 1$
12: until converged
```

Indeed,

$$
\begin{aligned}
\mathbb{E}\left[\nabla_\xi \log p_\xi(\theta) B(S)\right] &= \int_{\mathbb{S}} d^\xi(s) \int_\Theta p_\xi(\theta) \nabla_\xi \log p_\xi(\theta) B_\xi(s) d\theta ds \\
&= \int_{\mathbb{S}} d^\xi(s) B_\xi(s) ds \underbrace{\int_\Theta \nabla_\xi p_\xi(\theta) d\theta}_{\nabla_\xi 1 = 0} = 0
\end{aligned}
$$

This result can be used to design several actor-only or actor-critic algorithms that are the parameter-based equivalents of the traditional control-based policy-gradient algorithms discussed in the previous sections. Algorithm 4.7 reports the pseudocode for a parameter-based TD(0) actor-critic algorithm, which is one of the learning methods that will be used in the numerical applications.

4.6.2 Parameter-Based Natural Policy Gradient

Given the interpretation of p_ξ as a parameter-based stochastic policy, we can easily generalize the natural policy gradient methods by introducing the parameter-based Fischer information matrix

$$F_\xi = \mathbb{E}_{\theta\sim p_\xi}\left[\nabla_\xi \log p_\xi(\theta)\nabla_\theta \log p_\xi(\theta)^T\right] \tag{4.73}$$

Natural parameter-based policy gradient methods update the hyper-parameters following the natural gradient direction

$$\widetilde{\nabla}_\xi \rho(\xi) = F_\xi^{-1}\nabla_\xi \rho(\xi) \tag{4.74}$$

The advantage of this approach is that, for some particular probability distributions, the inverse of the Fisher information matrix can be computed analytically, as described in the next section.

4.6.2.1 NPGPE

In [53], the authors show that if the controller parameters are normally distributed the Fisher information matrix and its inverse can be computed analytically. This leads to an efficient online algorithm called *Natural Policy Gradient with Parameter-Based Exploration* (NPGPE). More in detail, suppose that

$$\theta \sim \mathcal{N}(\mu, \Gamma^T\Gamma)$$

where $\mu \in \mathbb{R}^n$ is the controller parameters mean and $\Gamma \in \mathbb{R}^{n\times n}$ denotes the Cholesky factor of the distribution covariance matrix Σ. The hyper-parameters are thus given by

$$\xi = (\mu, \Gamma^T_{1:n,1}, \Gamma^T_{2:n,2}, ..., \Gamma^T_{n:n,n}) \in \mathbb{R}^{\frac{n^2+3n}{2}}$$

For this distribution, the parameter-based policy gradient is given by

$$\nabla_\mu \log p_\xi(\theta) = \Sigma^{-1}(\theta-\mu) \tag{4.75}$$

$$\nabla_{\Gamma_k} \log p_\xi(\theta) = \begin{bmatrix}0 & I_{\bar{k}}\end{bmatrix}\left(\Gamma^{-1}Y - \text{diag}\left(\Gamma^{-1}\right)\right)e_k \tag{4.76}$$

where $\Gamma_k = \Gamma^T_{k,k:n}$ and $Y = \Gamma^{-T}(\theta-\mu)(\theta-\mu)^T\Gamma^{-1}$. These formulas might be used in a standard PGPE algorithm. However, the inversion of Γ is computationally expensive except for some very simple cases, for instance when *Sigma* is a diagonal matrix. In [77], the authors proved that the Fisher information matrix for this distribution is a block diagonal matrix $F_\xi = \text{diag}\{B_0, B_1, ..., B_n\}$ where

$$\begin{cases} B_0 &= \Sigma^{-1} \\ B_k &= \begin{bmatrix}0 & I_{\bar{k}}\end{bmatrix}\Gamma^{-1}(e_k e_k^T + I)\Gamma^{-T}\begin{bmatrix}0\\ I_{\bar{k}}\end{bmatrix} \end{cases}$$

Algorithm 4.8 NPGPE

```
Input:
    • Deterministic controller $F : \mathbb{S} \times \Theta \to \mathbb{A}$
    • Initial hyper-parameters $\xi^0$
    • Learning rate $\{\alpha_k\}$
    • Momentum parameter $\lambda$
Output: Approximation of the optimal hyper-parameters $\xi_*$
 1: Initialize $k = 0$, average reward $\widehat{\rho}_0 = 0$ and eligibility trace $e_{-1} = 0$
 2: repeat
 3:     Observe current state $s_k$
 4:     Draw $\zeta_k \sim \mathcal{N}(0, I_n)$
 5:     Compute controller parameters $\theta_k = \mu_k + \Gamma^T \zeta_k$
 6:     Perform action $a_k = F_{\theta_k}(s_k)$ and receive reward $r_{k+1}$
 7:     Update average reward estimate $\widehat{\rho}_{k+1} = \widehat{\rho}_k + \alpha_k (r_{k+1} - \widehat{\rho}_k)$
 8:     Compute natural policy gradients
            $\widetilde{\nabla}_\mu \log p_{\xi_k}(\theta_k) = \theta_k - \mu_k$
            $\widetilde{\nabla}_\Gamma \log p_{\xi_k}(\theta_k) = \left(\operatorname{triu}(\zeta_k \zeta_k^T) - \frac{1}{2}\operatorname{diag}(\zeta_k \zeta_k^T) - \frac{1}{2} I\right)\Gamma$
 9:     Update eligibility trace $e_k = \lambda e_{k-1} + \nabla_\xi \log p_{\xi_k}(\theta_k)$
10:     Update hyper-parameters $\xi_{k+1} = \xi_k + \alpha_k (r_{k+1} - \widehat{\rho}_k) e_k$
11:     $k \leftarrow k + 1$
12: until converged
```

where e_k is the k-th element of the canonical basis of $\mathbb{R}^n$ and $I_{\bar{k}}$ is $n - k + 1$-dimensional identity matrix. In [2], the authors found the following analytical expression for the inverse of each block

$$\begin{cases} B_0^{-1} &= \Sigma \\ B_k^{-1} &= \begin{bmatrix} 0 & I_{\bar{k}} \end{bmatrix} \Gamma^T \left(\begin{bmatrix} 0 & 0 \\ 0 & I_{\bar{k}} \end{bmatrix} - \frac{1}{2} e_k e_k^T \right) \Gamma \begin{bmatrix} 0 \\ I_{\bar{k}} \end{bmatrix} \end{cases}$$

Multiplying the policy gradients by the inverse Fisher informatio matrix $F_\xi^{-1} = \operatorname{diag}\{B_0^{-1}, B_1^{-1}, ..., B_n^{-1}\}$, we obtain the following natural policy gradients

$$\widetilde{\nabla}_\mu \log p_\xi(\theta) = \theta - \mu \tag{4.77}$$

$$\widetilde{\nabla}_\Gamma \log p_\xi(\theta) = \left(\operatorname{triu}(Y) - \frac{1}{2}\operatorname{diag}(Y) - \frac{1}{2} I\right)\Gamma \tag{4.78}$$

Algorithm 4.9 Natural Actor-Critic PGPE

Input:

- Deterministic controller $F : \mathbb{S} \times \Theta \to \mathbb{A}$
- Initial hyper-parameters ξ^0
- Initial critic parameters ψ_0
- Actor learning rate $\{\alpha_k\}$
- Critics learning rate $\{\beta_k\}$
- Momentum parameter λ

Output: Approximation of the optimal hyper-parameters ξ_*

```
1: Initialize $k = 0$, average reward $\widehat{\rho}_0 = 0$ and eligibility trace $e_{-1} = 0$
2: repeat
3:     Observe current state $s_k$
4:     Draw $\zeta_k \sim \mathcal{N}(0, I_n)$
5:     Compute controller parameters $\theta_k = \mu_k + \Gamma^T \zeta_k$
6:     Perform action $a_k = F_{\theta_k}(s_k)$ and receive reward $r_{k+1}$
7:     Update average reward estimate $\widehat{\rho}_{k+1} = \widehat{\rho}_k + \alpha_k(r_{k+1} - \widehat{\rho}_k)$
8:     Compute TD error $\widehat{\delta}_k = r_{k+1} - \widehat{\rho}_{k+1} + \widehat{V}_{\psi_k}(s_{k+1}) - \widehat{V}_{\psi_k}(s_k)$
9:     Update critic parameters $\psi_{k+1} = \psi_k + \alpha_k \widehat{\delta}_k \nabla_\psi \widehat{V}_{\psi_k}(s_k)$ .
10:    Compute natural policy gradients
           $$\begin{aligned} \widetilde{\nabla}_\mu \log p_{\xi_k}(\theta_k) &= \theta_k - \mu_k \\ \widetilde{\nabla}_\Gamma \log p_{\xi_k}(\theta_k) &= \left( \operatorname{triu}(\zeta_k \zeta_k^T) - \frac{1}{2} \operatorname{diag}(\zeta_k \zeta_k^T) - \frac{1}{2} I \right) \Gamma \end{aligned}$$
11:    Update eligibility trace $e_k = \lambda e_{k-1} + \widetilde{\nabla}_\xi \log p_{\xi_k}(\theta_k)$
12:    Update hyper-parameters $\xi_{k+1} = \xi_k + \alpha_k \widehat{\delta}_k e_k$.
13:    $k \leftarrow k + 1$
14: until converged
```

The natural policy gradients can be computed in $O(n^3)$, which is a sensible improvement compared to the $O(n^6)$ complexity for the standard PGPE updates. Let us remark that for an independent multi-variate Gaussian distribution, i.e. a diagonal covariance matrix Σ, the updates can be performed in $O(n)$. However, this approach neglects the dependences among parameters and could lead to suboptimal performances. The resulting NPGPE algorithm is reported in Algorithm 4.8. Given the parameter-based policy gradient algorithm, we can easily combine the actor-critic approach and the natural gradient technique in a parameter-based natural actor-critic algorithm, which is outlined in Algorithm 4.9.

4.6.3 Risk-Sensitive Setting

In the risk-sensitive setting, the agent tries to find a policy that maximizes the Sharpe ratio. Following the same reasoning of the control-based approach, we simply need to estimate the gradient of the average square reward $\eta(\xi)$. The parameter-based policy gradient generalizes without any additional effort

Theorem 4.6.2 (Risk-Sensitive Parameter-Based Policy Gradient)**.** *Let p_ξ be differentiable with respect to ξ, then the gradient of the average square reward is given by*

$$\nabla_\xi \eta(\xi) = \mathbb{E}_{\substack{S\sim d^\xi \\ \theta\sim p_\xi}} \left[\nabla_\xi \log p_\xi(\theta) W_{\pi_\xi}(S,\theta)\right] \tag{4.79}$$

where we denoted $W_\xi(S,\theta) = W_\xi(S, F_\theta(S))$.

Again, we can add a state-dependent baseline $B_\xi(S)$ to the gradient without increasing the bias

$$\nabla_\xi \eta(\xi) = \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta)\left(W_{\pi_\xi}(S,\theta) - B_\xi(S)\right)\right] \tag{4.80}$$

4.6.3.1 Risk-Sensitive NPGPE

Chapter 5

Financial Applications of Reinforcement Learning

5.1 Bibliographical Survey

5.2 Asset Allocation

The asset allocation problem consists of determining how to dynamically invest the available capital in a portfolio of different assets in order to maximize the expected total return or another relevant performance measure. Let us consider a financial market consisting of $I+1$ different stocks that are traded only at discrete times $t \in \{0, 1, 2, \dots\}$ and denote by $Z_t = (Z_t^0, Z_t^1, \dots, Z_t^I)^T$ their prices at time t. Typically, Z_t^0 refers to a riskless asset whose dynamic is given by $Z_t^0 = (1+X)^t$ where X is the deterministic risk-free interest rate. The investment process works as follows: at time t, the investor observes the state of the market S_t, consisting for example of the past asset prices and other relevant economic variables, and subsequently chooses how to rebalance his portfolio, by specifying the units of each stock $n_t = (n_t^0, n_t^1, \dots, n_t^I)^T$ to be held between t and $t+1$. In doing so, he needs to take into account the transaction costs that he has to pay to the broker to change his position. At time $t+1$, the investor realizes a profit or a loss from his investment due to the stochastic variation of the stock values. The investor's goal is to maximize a given performance measure.

5.2.1 Reward Function

Let W_t denote the wealth of the investor at time t. The profit realized between t and $t+1$ is simply given by the difference between the trading results and the transaction costs payed to the broker. More formally

$$\Delta W_{t+1} = W_{t+1} - W_t = \mathrm{PNL}_{t+1} - \mathrm{TC}_t$$

where PNL_{t+1} denotes the profit due to the variation of the portfolio asset prices between t and $t+1$

$$\mathrm{PNL}_{t+1} = n_t \cdot \Delta Z_{t+1} = \sum_{i=0}^{I} n_t^i (Z_{t+1}^i - Z_t^i)$$

and TC_t denotes the fees payed to the broker to change the portfolio allocation and on the short positions

$$\mathrm{TC}_t = \sum_{i=0}^{I} \delta_p^i \left| n_t^i - n_{t-1}^i \right| Z_t^i - \delta_f W_t \mathbb{1}_{n_t \neq n_{t-1}} - \sum_{i=0}^{I} \delta_s^i (n_t^i)^- Z_t^i$$

The transaction costs consist of three different components. The first term represent a transaction cost that is proportional to the change in value of the position in each asset. The second term is a fixed fraction of the total value of the portfolio which is payed only if the allocation is changed. The last term represents the fees payed to the broker for the shares borrowed to build a short position. The portfolio return between t and $t+1$ is thus given by

$$X_{t+1} = \frac{\Delta W_{t+1}}{W_t} = \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \tag{5.1}$$

where

$$X_{t+1}^i = \frac{\Delta Z_{t+1}^i}{Z_t^i}$$

is the return of the i-th stock between t and $t+1$,

$$a_t^i = \frac{n_t^i Z_t^i}{W_t}$$

is the fraction of wealth invested in the i-th stock between time t and $t+1$ and finally

$$\tilde{a}_t^i = \frac{n_{t-1}^i Z_t^i}{W_t} = \frac{a_{t-1}^i (1 + X_t^i)}{1 + X_t}$$

is the fraction of wealth invested in the i-th stock just before the reallocation. We assume that the agent invests all his wealth at each step, so that W_t can be also interpreted as the value of his portfolio. This assumption leads to the following constraint on the portfolio weights

$$\sum_{i=0}^{I} a_t^i = 1 \qquad \forall t \in \{0, 1, 2, \ldots\} \tag{5.2}$$

We notice that we are neglecting the typical margin requirements on the short positions, which would reduce the available capital at time t. Considering margin requirements would lead to a more complex constraint on the portfolio weights which would be difficult to treat in the reinforcement learning framework. Plugging this constraint into Eq. (5.1), we obtain

$$X_{t+1} = X + \sum_{i=1}^{I} a_t^i (X_{t+1}^i - X) - \sum_{i=0}^{I} \left[\delta_i \left|a_t^i - \tilde{a}_t^i\right| - \delta_s^i (a_t^i)^-\right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \tag{5.3}$$

which highlights the role of the risk-free asset as a benchmark for the portfolio returns. The total profit realized by the investor between $t = 0$ and T is

$$\Pi_T = W_T - W_0 = \sum_{t=1}^{T} \Delta W_t = \sum_{t=1}^{T} W_t X_t$$

The portfolio return between $t = 0$ and T is given by

$$X_{0,T} = \frac{W_T}{W_0} - 1 = \prod_{t=1}^{T} (1 + X_t) - 1$$

In order to cast the asset allocation problem in the reinforcement learning framework, we consider the log-return of the portfolio between $t = 0$ and T

$$R_{0,T} = \log \frac{W_T}{W_0} = \sum_{t=1}^{T} \log(1 + X_t) = \sum_{t=1}^{T} R_t \tag{5.4}$$

where R_{t+1} is the log-return of the portfolio between t and $t + 1$

$$R_{t+1} = \log \left\{ 1 + \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left|a_t^i - \tilde{a}_t^i\right| - \delta_s (a_t^i)^-\right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \right\} \tag{5.5}$$

The portfolio return and log-return can be used as the reward function of a RL algorithm, either in a offline or in an online approach.

5.2.2 States

At each time step, the agent observes the state of the system to make his decisions. First, we consider the $P+1$ past returns of all risky assets, i.e. $\{X_t, X_{t-1}, \dots, X_{t-P}\}$. These input variables may be used to construct more complex features for example using some deep learning techniques, such as a deep auto-encoder. In order to properly incorporate the effects of transaction costs into his decision process, the agent must keep track of its current position $\tilde{a}_t$. Finally, we might consider some external variables Y_t that may be relevant to the trader, such as the common technical indicator used in practice. Summing up, we assume that the state of the system is composed in the following way

$$S_t = \{X_t, X_{t-1}, \dots, X_{t-P}, \tilde{a}_t, Y_t, Y_{t-1}, \dots, Y_{t-P}\} \tag{5.6}$$

5.2.3 Actions

The agent, or trading system, only specifies the portfolio weights $a_t = (a_t^0, \dots, a_t^I)^T$ and therefore determines the allocation for the time interval $[t, t+1)$. If we assume that the agent invests all of his capital at each time step and that short-selling is not allowed, then the portfolio weights should satisfy, for every $t \in \{0, 1, \dots\}$, the following constraints

$$\begin{cases} a_t^i \geq 0 \ , \ \forall i \in \{0, \dots, I\} \\ \sum_{i=0}^I a_t^i = 1 \end{cases} \tag{5.7}$$

This constraint can be easily enforced by considering a parametric softmax policy. If short-selling is allowed, weights might also be negative. A simple approach is to assume that, for every $t \in \{0, 1, \dots\}$, the weights satisfy

$$\begin{cases} a_t^i \in \mathbb{R} \ , \ \forall i \in \{1, \dots, I\} \\ a_t^0 = 1 - \sum_{i=1}^I a_t^i \end{cases} \tag{5.8}$$

Since a_t^0 is uniquely determined by the other weights, it is enough to define a policy that specifies the allocation in the risky assets, e.g. a Gaussian policy in $\mathbb{R}^I$. The obvious shortcoming of this approach is that the agent might enter in huge short positions, which is not realistic. A first observation is that, if the stochastic policy is concentrated around the origin, huge long or short positions would have very small probabilities of being selected. Moreover, we notice that the agent would pay large fees for entering into large short positions, independently of the trading profits. Therefore, we expect the agent to learn that short positions are very expensive and would therefore try to

avoid them.
Working in a continuous action space is computationally difficult and only few reinforcement learning algorithm are well-suited to this setting, e.g. policy gradient methods. A simpler approach is to reduce the action space to a discrete space. For instance, in the two assets scenario we might assume that $a_t \in \{-1, 0, +1\}$. Thus the agent may be long $(+1)$, neutral (0) or short (-1) on the risky-asset. Working in a discrete action space is more simple, and standard value-based approaches might also be employed.

Chapter 6

Numerical Results for the Asset Allocation Problem

In this chapter we present the numerical results of some of the policy gradient algorithms discussed in Chapter 4 for the asset allocation problem. Two different type of markets are analyzed: a market with only one risky asset and a market where multiple risky assets are available, for which finding a trading strategy is more difficult since the state and action spaces are much larger. The learning algorithms are first applied both in their risk-neutral version and risk-sensitive formulation to synthetically generated data, which present profitably tradable features. Once the behavior of these algorithms is validated in this controlled environment, the various methods are also applied to historical price series.

6.1 Synthetic Risky Asset

To test the different reinforcement learning methods in a controlled environment, we generated log-price series for the risky asset as random walks with autoregressive trend processes. The two-parameter model is thus given by

$$\begin{aligned} z_t &= z_{t-1} + \beta_{t-1} + \kappa\epsilon_t \\ \beta_t &= \alpha\beta_{t-1} + \nu_t \end{aligned}$$

We then define the synthetic price series as

$$Z_t = \exp\left(\frac{z_t}{\max_t z_t - \min_t z_t}\right)$$

This model is often taken as a benchmark test in the automated trading literature, see for instance [57], because the price series generated in this

way present some patterns that can be profitably exploited. Moreover the model is stationary and therefore the policy learned on the training set should generalize well on the test set, also known as backtest in the financial jargon. Thus we would expect our learning algorithms to perform well on this test case. If this wasn't the case, we should go back and improve the learning algorithms.
In this setting, we compare three the results of three long-short strategies obtained with ARAC, PGPE and NPGPE in both the risk-neutral and risk-sensitive framework. This means that the agent can either go long on the risky asset (i.e. $a_t^1 = 1$) or short the security (i.e. $a_t^1 = -1$) and invest the proceedings in the risk-less asset. Given the current conditions of the financial markets, we always assume a risk-free rate $X = 0$. Let us describe in more detail the the choice we made for each of the algorithms.

6.1.1 Learning Algorithm Specifications

6.1.1.1 ARAC

For the ARAC algorithm we considered a Boltzmann exploration policy on the two actions $a_t^1 \in \{-1, 1\}$ and a linear critic in which the features coincide with the agent's observation of the system state. This critic is extremely simple and there is surely some work to be done to improve it.

6.1.1.2 PGPE

For the PGPE algorithm we considered a binary deterministic controller

$$F_\theta(s) = \mathrm{sign}(\theta \cdot s)$$

where the parameters and the state also include a bias term. The controller parameters are sampled from a multi-variate Gaussian distribution

$$\theta \sim \mathcal{N}(\mu, \mathrm{diag}(\sigma))$$

6.1.1.3 NPGPE

In NPGPE, we used the same controller as for PGPE but we assumed that the controller parameters are sampled from a Gaussian distribution parameterized by its mean and Cholesky factor

$$\theta \sim \mathcal{N}(\mu, C^T C)$$

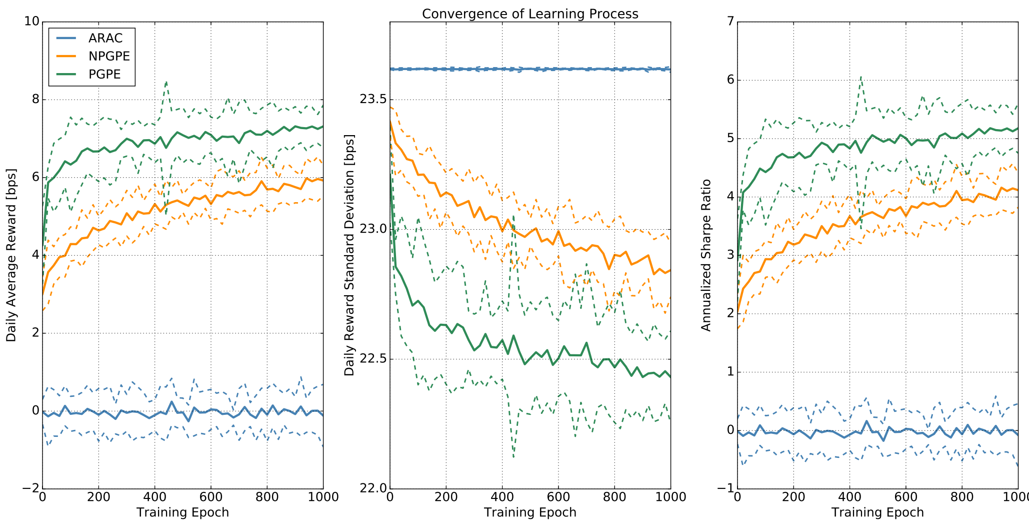

Figure 6.1: Risk-neutral learning process for the asset allocation problem with one synthetic risky asset.

6.1.2 Experimental Setup

All the algorithms were tested on the same price series of size 9000, generated from the process described above using $\alpha = 0.9$ and $\kappa = 3$. The learning process consisted of 500 training epochs on the first 7000 days of the series with a learning rate that decreased at each epoch according to a polynomial schedule. The trained agents were subsequently backtested on the final 2000 days, during which the agents kept learning online in order to try to adapt to the changing environment. The results that we present are the average of 10 independent experiments that used slightly different random initialization of the policy parameters.

6.1.3 Risk-Neutral Framework

6.1.3.1 Convergence

Let us first discuss the case with no transaction costs. Figure 6.1 shows the learning curves for the three risk-neutral algorithms in terms of average daily reward, which is the quantity being maximized by the algorithms, the daily reward standard deviation and the annualized Sharpe ratio. The first thing we observe is the ARAC algorithm seems not to be improving the trading strategy as the training epochs go by. The average reward obtained is close to zero and will be surely be negative once transaction costs are introduced. On the other hand, NPGPE slowly converges to a profitable strategy which

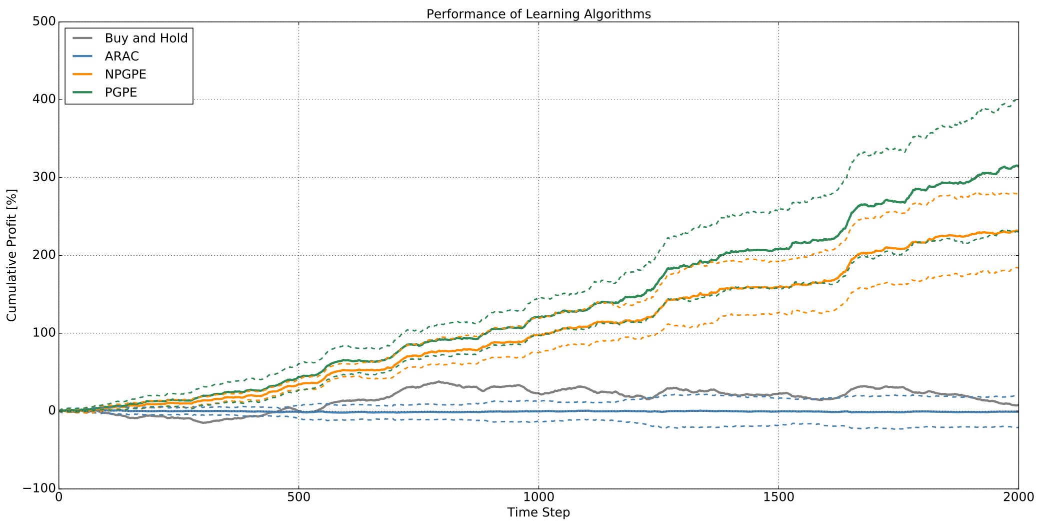

Figure 6.2: Backtest performance of trained trading systems for the asset allocation problem with one synthetic risky asset.

is however suboptimal compared to the one found by PGPE, that is better in all three measures considered. It is interesting to notice that PGPE and NPGPE yield a learning curve for the Sharpe ratio very similar to the one for the average reward. Even if the algorithm is risk-neutral, it manages to improve a risk-senitive measure at the same time of the average reward. This might be simply a peculiarity of the very simple model assumed for the synthetic risky asset. Moreover, since the price process is stationary, the trading strategy learned on the training set perfectly generalizes to the test set.

6.1.3.2 Performances

Figure 6.2 compares the backtest performances of the three learned policies and a Buy and Hold strategy, which simply consists in investing all the available capital in the risky asset. Let us repeat that the solid lines are the averages of 10 independent experiments, which allows us to determine the 95% confidence intervals represented with the dashed lines. We clearly see that NPGPE and PGPE easily beat the market, realizing a total profit of 231.63% and 314.34% respectively against the 7.81% profit of the Buy and Hold strategy over the same period. More statistics of the trading strategies are reported in Table 6.1.

Describe statistics meaning

	Buy and Hold	ARAC	NPGPE	PGPE
Total Return	7.81%	-0.86%	231.63%	314.34%
Daily Sharpe	0.27	-0.02	4.13	4.95
Monthly Sharpe	0.19	-0.07	2.90	3.26
Yearly Sharpe	0.23	-0.10	1.55	1.76
Max Drawdown	-22.35%	-12.60%	-3.72%	-3.27%
Avg Drawdown	-1.75%	-1.81%	-0.49%	-0.43%
Avg Up Month	2.87%	1.14%	2.47%	2.74%
Avg Down Month	-2.58%	-1.10%	-0.73%	-0.67%
Win Year %	40.00%	44.00%	98.00%	100.00%
Win 12m %	56.36%	48.00%	100.00%	100.00%
Reallocation Freq	0.00%	50.01%	19.99%	15.43%
Short Freq	0.00%	50.13%	41.59%	44.25%

Table 6.1: Backtest statistics of the risk-neutral trading strategies for the asset allocation problem with one synthetic risky asset.

6.1.3.3 Impact of Transaction Costs

In the algorithmic trading literature there are many examples of strategies based on the prediction of future rewards starting from more or less complex indicators [42], [67], [50]. However, as pointed out in [29], the performances of these methods quickly degrade when transaction costs for changing the portfolio composition or for shorting a security are considered. Indeed, these methods simply invest based on the prediction of the future returns, without explicitly taking into account transaction costs. On the other hand, reinforcement learning algorithms should learn to avoid frequent reallocations or shorts thanks to the feedback mechanism between the learning agent and the system, thus generating better trading performances. In this section we analyze how the strategies learned by PGPE and by NPGPE change when gradually increasing the proportional transaction costs and the short-selling fees. Intuitively, we expect a progressive reduction of the frequency of reallocation and of shorting the risky asset.
Figure 6.3 shows the impact of proportional transaction costs on the trading strategies learned by PGPE and by NPGPE. As expected, the frequency of reallocation for both strategies quickly drops to zero as the transaction costs increase, converging to the profitable buy and hold strategy. It is peculiar that the reallocation frequency for the PGPE strategy initially drops more quickly than for the NPGPE strategy, but then slows down and even increases when $\delta_P = 20$ bps. In summary, both algorithms are able to identify real-

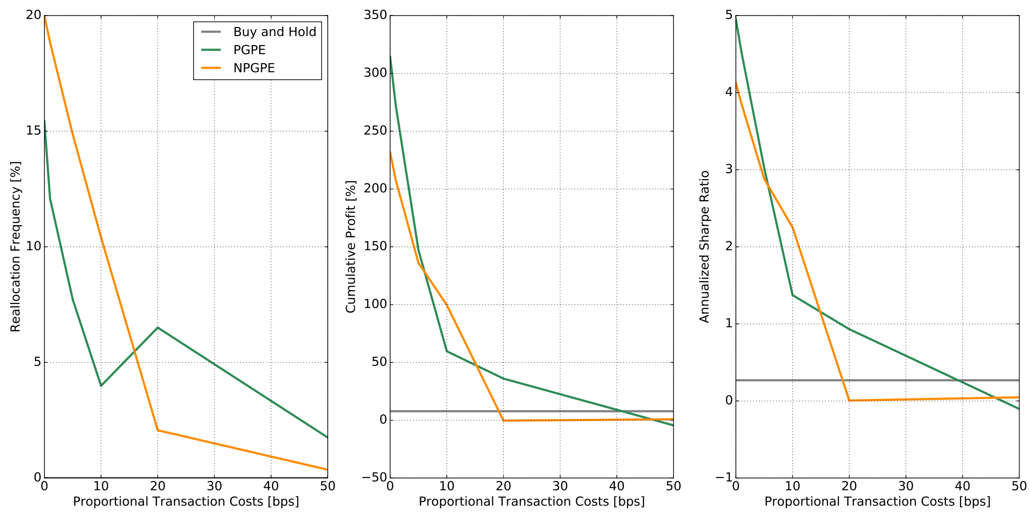

Figure 6.3: Impact of proportional transaction costs on the trading strategies learned by PGPE and NPGPE.

location as the cause for lower rewards and to subsequently reduce the rate of reallocation, converging towards the simple yet profitable buy and hold strategy. Figure 6.4 shows the impact of short-selling fees on the trading strategies learned by PGPE and NPGPE. Both algorithms behave as expected, displaying a progressive reduction of the frequency of short positions as the fees increase. For large values of short-selling fees, both strategies converge to the profitable buy and hold strategy, which completely avoids paying the fees. In particular, PGPE quickly replicates the buy and hold strategy. On the other hand, NPGPE is not able to exactly reproduce the buy and hold strategy but it seems to converge to it for very large values of the short-selling fee.

6.1.4 Risk-Sensitive Framework

In this section we present the results in the risk-sensitive framework, in which all the algorithms optimize the Sharpe ratio of the policy. Figure 6.5 shows the learning curves for the three risk-sensitive algorithms RSARAC, RSPGPE and RSNPGPE, which show similar behaviors to their risk-neutral counterparts. However, it is surprising that the strategies learned by RSPGPE and RSNPGPE have smaller Sharpe ratio than in the risk-neutral version of these algorithms, which optimize a different quantity. Figure 6.6 shows the backtest performances for the three risk-sensitive trading strategies. Again, RSPGPE and NPGPE beat the market even if in smaller measure than in the risk-neutral setting.

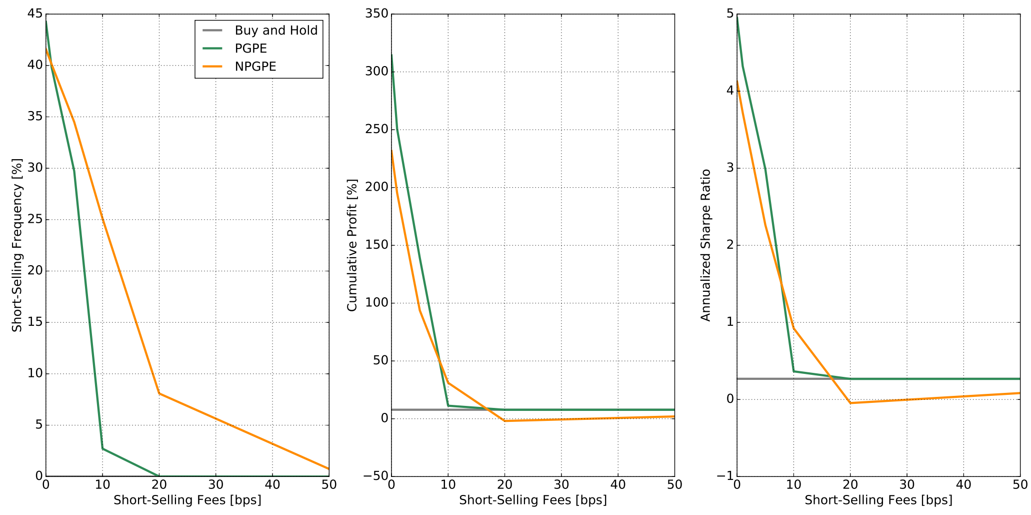

Figure 6.4: Impact of short-selling fees on the trading strategies learned by PGPE and NPGPE.

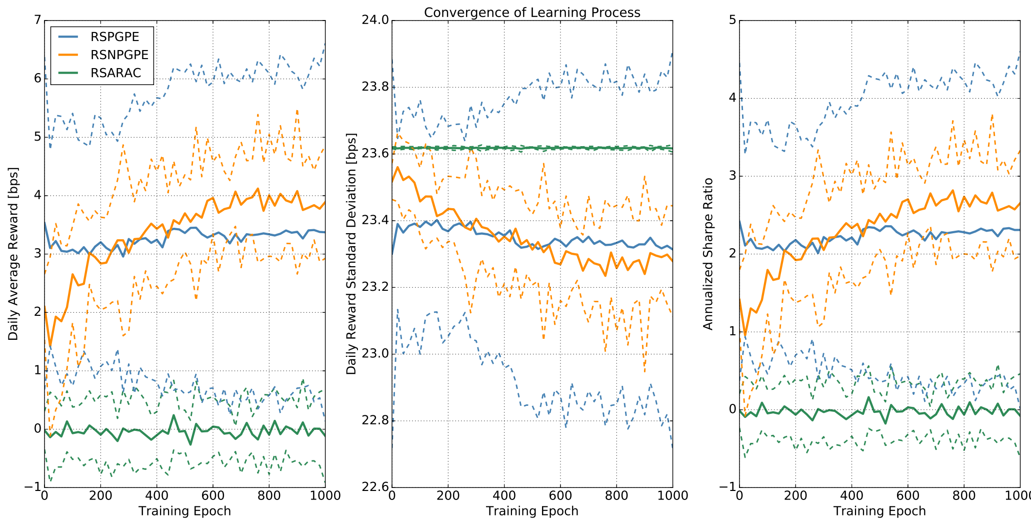

Figure 6.5: Risk-sensitive learning process for the asset allocation problem with one synthetic risky asset.

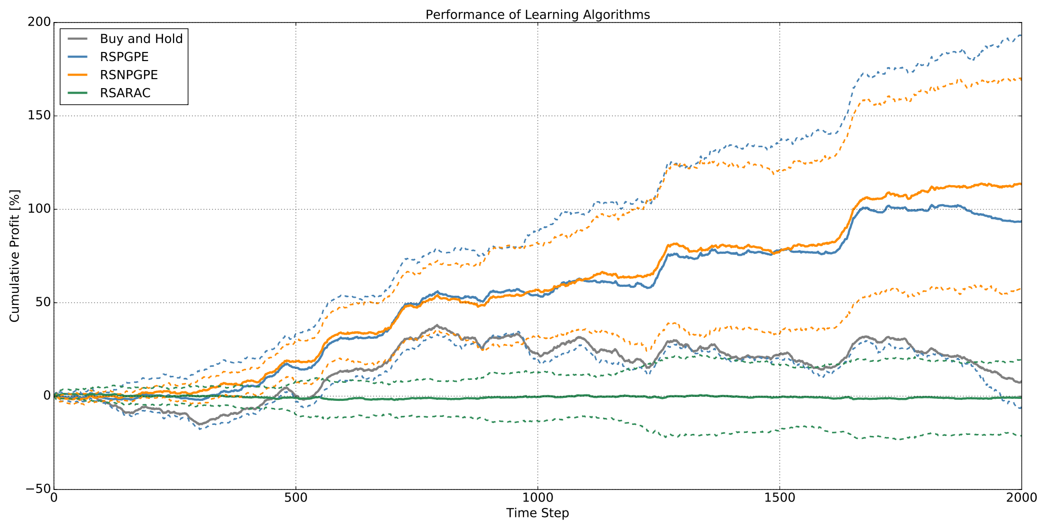

Figure 6.6: Backtest performance of the risk-sensitive trading strategies for the asset allocation problem with one synthetic risky asset.

	Buy and Hold	RSARAC	RSNPGPE	RSPGPE
Total Return	7.81%	-0.88%	113.42%	93.51%
Daily Sharpe	0.27	-0.03	2.55	2.15
Monthly Sharpe	0.19	-0.07	1.77	1.32
Yearly Sharpe	0.23	-0.09	1.05	0.78
Max Drawdown	-22.35%	-12.62%	-5.95%	-9.26%
Avg Drawdown	-1.75%	-1.77%	-0.63%	-0.95%
Avg Up Month	2.87%	1.14%	2.22%	2.59%
Avg Down Month	-2.58%	-1.11%	-1.00%	-1.35%
Win Year %	40.00%	44.00%	92.00%	80.00%
Win 12m %	56.36%	48.00%	98.55%	91.64%
Reallocation Freq	0.00%	49.99%	35.78%	16.15%
Short Freq	0.00%	50.12%	33.17%	21.84%

Table 6.2: Backtest statistics of the risk-sensitive trading strategies for the asset allocation problem with one synthetic risky asset.

6.2 Historic Risky Asset

6.3 Multiple Synthetic Risky Assets

6.4 Historic Multiple Risky Assets

Chapter 7

Conclusions

7.1 Summary

7.2 Further Developments

Bibliography

[1] A. Agarwal, P. Bartlett, and M. Dama, *Optimal allocation strategies for the dark pool problem*, arXiv preprint arXiv:1003.2245, (2010).

[2] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi, *Bidirectional relation between cma evolution strategies and natural evolution strategies*, in International Conference on Parallel Problem Solving from Nature, Springer, 2010, pp. 154–163.

[3] R. Almgren and N. Chriss, *Optimal execution of portfolio transactions*, Journal of Risk, 3 (2001), pp. 5–40.

[4] A. Arapostathis, V. S. Borkar, E. Fernández-Gaucherand, M. K. Ghosh, and S. I. Marcus, *Discrete-time controlled markov processes with average cost criterion: a survey*, SIAM Journal on Control and Optimization, 31 (1993), pp. 282–344.

[5] A. Basu, T. Bhattacharyya, and V. S. Borkar, *A learning algorithm for risk-sensitive cost*, Mathematics of Operations Research, 33 (2008), pp. 880–898.

[6] N. Bäuerle and U. Rieder, *Markov decision processes with applications to finance*, Springer Science & Business Media, 2011.

[7] J. Baxter and P. L. Bartlett, *Infinite-horizon policy-gradient estimation*, Journal of Artificial Intelligence Research, 15 (2001), pp. 319–350.

[8] S. D. Bekiros, *Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach*, Journal of Economic Dynamics and Control, 34 (2010), pp. 1153–1170.

[9] F. Bertoluzzo and M. Corazza, *Testing different reinforcement learning configurations for financial trading: Introduction and applications*, Procedia Economics and Finance, 3 (2012), pp. 68–77.

[10] ——, *Q-learning-based financial trading systems with applications*, University Ca'Foscari of Venice, Dept. of Economics Working Paper Series No, 15 (2014).

[11] ——, *Reinforcement learning for automated financial trading: Basics and applications*, in Recent Advances of Neural Network Models and Applications, Springer, 2014, pp. 197–213.

[12] D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 1, Athena Scientific, Belmont, 1995.

[13] D. P. Bertsekas and S. E. Shreve, *Stochastic optimal control: the discrete-time case*, vol. 23, Academic Press New York, 1978.

[14] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Optimization and neural computation series, Athena Scientific, Belmont, 1 ed., 1996.

[15] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, *Natural actor–critic algorithms*, Automatica, 45 (2009), pp. 2471–2482.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[17] V. S. Borkar, *A sensitivity formula for risk-sensitive cost and the actor–critic algorithm*, Systems & Control Letters, 44 (2001), pp. 339–346.

[18] ——, *Q-learning for risk-sensitive control*, Mathematics of operations research, 27 (2002), pp. 294–311.

[19] V. S. Borkar and S. P. Meyn, *Risk-sensitive optimal control for markov decision processes with monotone cost*, Mathematics of Operations Research, 27 (2002), pp. 192–209.

[20] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, vol. 39, CRC press, 2010.

[21] P. X. Casqueiro and A. J. Rodrigues, *Neuro-dynamic trading methods*, European Journal of Operational Research, 175 (2006), pp. 1400–1412.

[22] N. Chapados and Y. Bengio, *Cost functions and model combination for var-based asset allocation using neural networks*, IEEE Transactions on Neural Networks, 12 (2001), pp. 890–906.

[23] M. Choey and A. S. Weigend, *Nonlinear trading models through sharpe ratio maximization*, International Journal of Neural Systems, 8 (1997), pp. 417–431.

[24] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, *Risk-sensitive and robust decision-making: a cvar optimization approach*, in Advances in Neural Information Processing Systems, 2015, pp. 1522–1530.

[25] M. Corazza and A. Sangalli, *Q-learning vs. sarsa: comparing two intelligent stochastic control approaches for financial trading.*

[26] J. Cumming, D. Alrajeh, and L. Dickens, *An investigation into the use of reinforcement learning techniques within the algorithmic trading domain*, (2015).

[27] M. A. Dempster and V. Leemans, *An automated fx trading system using adaptive reinforcement learning*, Expert Systems with Applications, 30 (2006), pp. 543–552.

[28] M. A. H. Dempster and Y. S. Romahi, *Intraday FX trading: An evolutionary reinforcement learning approach*, Springer, 2002.

[29] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, *Deep direct reinforcement learning for financial signal representation and trading*, IEEE Transactions on Neural Networks and Learning Systems, (2016).

[30] Y. Deng, Y. Kong, F. Bao, and Q. Dai, *Sparse coding inspired optimal trading system for hft industry*, IEEE Transactions on Industrial Informatics, 11 (2015), pp. 467–475.

[31] X. Du, J. Zhai, and K. Lv, *Algorithm trading using q-learning and recurrent reinforcement learning*, positions, 1, p. 1.

[32] T. Elder, *Creating algorithmic traders with hierarchical reinforcement learning*, (2008).

[33] L. A. Feldkamp, D. V. Prokhorov, C. F. Eagen, and F. Yuan, *Enhanced multi-stream kalman filter training for recurrent networks*, in Nonlinear Modeling, Springer, 1998, pp. 29–53.

[34] K. Ganchev, Y. Nevmyvaka, M. Kearns, and J. W. Vaughan, *Censored exploration and the dark pool problem*, Communications of the ACM, 53 (2010), pp. 99–107.

[35] C. Gold, *Fx trading via recurrent reinforcement learning*, in IEEE 2003 IEEE International Conference on Computational Intelligence for Financial Engineering. Proceedings, 2003.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Book in preparation for MIT Press, 2016.

[37] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer, 2009.

[38] D. Hendricks and D. Wilcox, *A reinforcement learning extension to the almgren-chriss framework for optimal trade execution*, in IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), 2014, pp. 457–464.

[39] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach*, GMD-Forschungszentrum Informationstechnik, 2002.

[40] M. S. Joshi, *C++ design patterns and derivatives pricing*, vol. 2, Cambridge University Press, 2008.

[41] S. Kakade, *A natural policy gradient.*, in NIPS, vol. 14, 2001, pp. 1531–1538.

[42] K. Kamijo and T. Tanigawa, *Stock price pattern recognition-a recurrent neural network approach*, in IEEE 1990 IJCNN International Joint Conference on Neural Networks, 1990.

[43] M. Kearns and Y. Nevmyvaka, *Machine learning for market microstructure and high frequency trading*, High-Frequency Trading–New Realities for Traders, Markets and Regulators, (2013), pp. 91–124.

[44] V. R. Konda and J. N. Tsitsiklis, *Actor-critic algorithms.*, in NIPS, vol. 13, 1999, pp. 1008–1014.

[45] H. Kushner and G. G. Yin, *Stochastic approximation and recursive algorithms and applications*, vol. 35, Springer Science & Business Media, 2003.

[46] P. L.A. and M. Ghavamzadeh, *Actor-critic algorithms for risk-sensitive mdps*, in Advances in Neural Information Processing Systems 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., Curran Associates, Inc., 2013, pp. 252–260.

[47] S. Laruelle, C.-A. Lehalle, and G. Pages, *Optimal split of orders across liquidity pools: a stochastic algorithm approach*, SIAM Journal on Financial Mathematics, 2 (2011), pp. 1042–1076.

[48] ——, *Optimal posting price of limit orders: learning by trading*, Mathematics and Financial Economics, 7 (2013), pp. 359–403.

[49] H. Li, C. H. Dagli, and D. Enke, *Short-term stock market timing prediction under reinforcement learning schemes*, in 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 233–240.

[50] J. Liang, W. Song, and M. Wang, *Stock price prediction based on procedural neural networks*, Adv. Artif. Neu. Sys., 2011 (2011), pp. 1–11.

[51] S. Mahadevan, *Average reward reinforcement learning: Foundations, algorithms, and empirical results*, Machine learning, 22 (1996), pp. 159–195.

[52] H. Markowitz, *Portfolio selection*, The journal of finance, 7 (1952), pp. 77–91.

[53] A. Miyamae, Y. Nagata, I. Ono, and S. Kobayashi, *Natural policy gradient methods with parameter-based exploration for control tasks*, in Advances in neural information processing systems, 2010, pp. 1660–1668.

[54] J. Moody and M. Saffell, *Learning to trade via direct reinforcement*, Neural Networks, IEEE Transactions on, 12 (2001), pp. 875–889.

[55] J. Moody, M. Saffell, Y. Liao, and L. Wu, *Reinforcement learning for trading systems*, in Decision Technologies for Computational Finance: Proceedings of the fifth International Conference Computational Finance, vol. 2, Springer Science & Business Media, 2013, p. 129.

[56] J. Moody and L. Wu, *Optimization of trading systems and portfolios*, in Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr), 1997, pp. 300–307.

[57] J. Moody, L. Wu, Y. Liao, and M. Saffell, *Performance functions and reinforcement learning for trading systems and portfolios*, Journal of Forecasting, 17 (1998), pp. 441–470.

[58] Y. Nevmyvaka, Y. Feng, and M. Kearns, *Reinforcement learning for optimized trade execution*, in Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 673–680.

[59] J. Nocedal and S. Wright, *Numerical optimization*, Springer Science & Business Media, 2006.

[60] J. O, J. Lee, J. W. Lee, and B.-T. Zhang, *Adaptive stock trading with dynamic asset allocation using reinforcement learning*, Information Sciences, 176 (2006), pp. 2121–2147.

[61] G. Pages, *Introduction to Numerical Probability for Finance*, LPMA-Université Pierre et Marie Curie, 2016.

[62] J. Peters, K. Mülling, and Y. Altun, *Relative entropy policy search.*, in AAAI, Atlanta, 2010.

[63] J. Peters and S. Schaal, *Policy gradient methods for robotics*, in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, IEEE, 2006, pp. 2219–2225.

[64] ——, *Reinforcement learning of motor skills with policy gradients*, Neural networks, 21 (2008), pp. 682–697.

[65] L. Prashanth and M. Ghavamzadeh, *Actor-critic algorithms for risk-sensitive reinforcement learning.*, arXiv preprint arXiv:1403.6530, (2014).

[66] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 1994.

[67] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, *Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks*, IEEE Transactions on Neural Networks, 9 (1998), pp. 1456–1470.

[68] C. Sanderson, *Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments*, (2010).

[69] M. Sato and S. Kobayashi, *Variance-penalized reinforcement learning for risk-averse asset allocation*, in Intelligent Data Engineering and Automated Learning—IDEAL 2000. Data Mining, Financial Engineering, and Intelligent Agents, Springer, 2000, pp. 244–249.

[70] ——, *Average-reward reinforcement learning for variance penalized markov decision problems*, in Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, San Francisco, CA, USA, 2001, Morgan Kaufmann Publishers Inc., pp. 473–480.

[71] F. Sehnke et al., *Parameter exploring policy gradients and their implications*, PhD thesis, Technische Universität München, 2012.

[72] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber, *Policy gradients with parameter-based exploration for control*, in Artificial Neural Networks-ICANN 2008, Springer, 2008, pp. 387–396.

[73] ——, *Parameter-exploring policy gradients*, Neural Networks, 23 (2010), pp. 551–559.

[74] W. F. Sharpe, *The sharpe ratio*, The journal of portfolio management, 21 (1994), pp. 49–58.

[75] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *Deterministic policy gradient algorithms*, in ICML, 2014.

[76] M. J. Sobel, *The variance of discounted markov decision processes*, Journal of Applied Probability, (1982), pp. 794–802.

[77] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, *Efficient natural evolution strategies*, in Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 2009, pp. 539–546.

[78] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, vol. 135, MIT Press Cambridge, 1998.

[79] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al., *Policy gradient methods for reinforcement learning with function approximation.*, in NIPS, vol. 99, 1999, pp. 1057–1063.

[80] C. Szepesvári, *Algorithms for reinforcement learning*, Synthesis lectures on artificial intelligence and machine learning, 4 (2010), pp. 1–103.

[81] A. Tamar, D. D. Castro, and S. Mannor, *Temporal difference methods for the variance of the reward to go*, in Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 495–503.

[82] A. Tamar, Y. Chow, M. Ghavamzadeh, and S. Mannor, *Policy gradient for coherent risk measures*, in Advances in Neural Information Processing Systems, 2015, pp. 1468–1476.

[83] A. Tamar, D. Di Castro, and S. Mannor, *Policy gradients with variance related risk criteria*, in Proceedings of the 29th International Conference on Machine Learning, 2012, pp. 387–396.

[84] A. Tamar and S. Mannor, *Variance adjusted actor critic algorithms*, arXiv preprint arXiv:1310.3697, (2013).

[85] Z. Tan, C. Quek, and P. Y. Cheng, *Stock trading with cycles: A financial application of anfis and reinforcement learning*, Expert Systems with Applications, 38 (2011), pp. 4741–4755.

[86] R. S. Tsay, *Analysis of financial time series*, vol. 543, John Wiley & Sons, 2005.

[87] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, Proceedings of the IEEE, 78 (1990), pp. 1550–1560.

[88] M. Wiering and M. Van Otterlo, *Reinforcement Learning: State-of-the-Art*, vol. 12 of Adaptation, Learning, and Optimization, Springer, 1 ed., 2012.

[89] R. J. Williams and D. Zipser, *A learning algorithm for continually running fully recurrent neural networks*, Neural computation, 1 (1989), pp. 270–280.

[90] S. Yang, M. Paddrik, R. Hayes, A. Todd, A. Kirilenko, P. Beling, and W. Scherer, *Behavior based learning in identifying high frequency trading strategies*, in IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012.

[91] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama, *Analysis and improvement of policy gradient estimation*, in Advances in Neural Information Processing Systems, 2011, pp. 262–270.

[92] T. Zhao, G. Niu, N. Xie, J. Yang, and M. Sugiyama, *Regularized policy gradients: Direct variance reduction in policy gradient estimation*, in Proceedings of The 7th Asian Conference on Machine Learning, 2015, pp. 333–348.