BACHELOR

Convergence of several reinforcement learning algorithms

Mohamed, Abdulwahab

Award date:
2018

Bachelor Final Project (2WH40)

Convergence of several reinforcement learning algorithms

Author: Abdulwahab Mohamed (0965250)

Supervisor: dr. J.W. Portegies

Summary

First, we give a brief introduction to finite state Markov decision processes. After that we show how to translate a well-known problem, the mountain car problem, in terms of Markov decision process. Several algorithms from reinforcement learning will be used to find the optimal policy. The algorithms we consider are dynamic programming, Monte Carlo and temporal-difference learning. We also give either heuristics or a proof of their convergence properties. Finally we apply the algorithms to the mountain car problem to see how the algorithms perform.

Contents

1 Introduction

Every Friday morning, Mary is dropped off at her little lamb by her mother. Mary then takes her little lamb from the sheepfold and they go for a walk. After the walk, Mary and her little lamb play hide and seek outside the sheepfold until Mary's mother arrives. When the mother arrives, it is always very hard to get the little lamb in the sheepfold, she wants to play and not to be alone. One day Mary gave the little lamb a candy after it got in. From that day, Mary felt that the lamb actually liked the candy. Mary gave the candy every time and the little lamb gets smoothly in its sheepfold. Now Mary does not have to say anything to her lamb, because it gets in its sheepfold after she sees the mother's car arriving from distance.

Apparently, Mary's little lamb has learned to go to its sheepfold without a struggle. It has learned to do that due to the reward, namely the candy, it gets. Its good behavior is reinforced. Such type of learning process is called *reinforcement learning*. It is a broad type of learning process where there is a reinforcer which gives reward or penalty to stimulate learning. It comes in many different varieties, from learning to get into the sheepfold calmly for getting candies to learning how to win chess for getting money.

One possible way to deal with reinforcement learning mathematically is through *Markov decision processes*. That is the route to reinforcement learning that we have taken in this report. The theory of Markov decision processes is well-developed. Textbooks such as the ones by M. Puterman in [1] and U. Rieder et al in [2] define Markov decision processes and show how to use it for reinforcement learning. The idea is to find a policy that the agent in question should follow. We use the book of Sutten and Barto [3] to get algorithms for finding, in some sense good, policies.

Although the book of Sutton and Barto is readable and clear for non-mathematicians, it lacks proofs why the algorithms actually work. However, in other books as [4], particular algorithms as *dynamic programming* are treated mathematically rigorously. There are still algorithms that are broadly used, while a rigorous treatment of such algorithms is not easy to find. For example *Q-learning with constant learning-rate* is only showed to "converge" under a strict assumption [5].

In this report we investigate what is currently known about the convergence of the most used reinforcement learning algorithms for finite state Markov decision processes. We focus our attention on the algorithms discussed in the first six chapters of [3]. We discuss *dynamic programming*, *Monte Carlo methods* and *temporal-difference learning*. We have also chosen to consider finite state and finite action Markov decision processes to make the question handleable in a bachelor final project.

The report is organized as follows. In Section 2 we give a definition of Markov decision processes. We also introduce some performance measures for policies. After that we give a possible way to translate the mountain car problem in terms of a Markov decision process. That is a decision problem where a car is placed between two mountains and the goal is to achieve the mountaintop. To make it interesting, the car's engine is so weak such that the car is not able to get to the mountaintop by just moving forward. It is the task for the driver to find out a way to swing between the two mountains and get to the mountaintop.

The mountain car problem serves also as a benchmark problem for the performance of the reinforcement learning algorithms that we will discuss. It is chosen for its simplicity. It is at the same time not trivial, which means that it is quite representative for reinforcement learning problems.

In Section 3 we discuss the convergence of dynamic programming, in particular, value iteration and policy iteration. Dynamic programming algorithms are in general well-studied. In fact, they are usually included in books on Markov decision processes. Different variants of the algorithms are proved to converge in [1] and [6]. The convergence results that we give are mostly inspired by [6].

We discuss Monte Carlo algorithms in Section 4. We only focus on two Monte Carlo algorithms, namely Monte Carlo exploring-starts and Monte Carlo first-visit. Convergence of algorithms with a different goal, but similar to the Monte Carlo algorithms that we consider, is given in [7]. Also in [3], a heuristic of the convergence of the algorithms is given. We will elaborate on both results.

Finally, in Section 5 we elaborate on the convergence of temporal-difference algorithms, in particular two versions of Q-learning. In [5] and [8], the authors showed convergence of the Q-learning algorithm. Moreover, there are also convergence rates known for the convergence [9]. Although the main subject in these articles is convergence of Q-learning, the results rely on different assumptions. To understand the core of their argument, we gave a proof of the convergence in a simpler situation.

2 Markov Decision Processes

Consider an autonomous agent which sees the environment at time k as being S_k. After that it performs an action A_k, possibly randomly, according to a policy. Finally it receives a reward R_{k+1}. This autonomous agent wants to maximize the rewards it gets in some sense. Informally speaking, this is the idea of a Markov decision process.

We will now make the above description more formal. To that end, we first define what we mean when we talk about a Markov decision process. The definition that we give here is very similar as the definition in [2]. However the one that we give here is less general.

2.1 The mathematical framework

Definition 1. A Markov decision process (MDP) is the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, r)$ where each object is specified as follows:

1. $\mathcal{S}$ is the state space. It is a finite set and is endowed with the σ-algebra $2^{\mathcal{S}}$.
2. $\mathcal{A}$ is the action space. It is a finite set and is endowed with the σ-algebra $2^{\mathcal{A}}$.
3. $\mathcal{R} \subset \mathbb{R}$ is the reward space. It is a finite set and is endowed with the σ-algebra $2^{\mathcal{R}}$.
4. $P : 2^{\mathcal{S}} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a mapping satisfying: for a fixed $(s, a) \in \mathcal{S} \times \mathcal{A}$, $P(\cdot|s, a)$ is a probability measure on $2^{\mathcal{S}}$. The probability measure $P(\cdot|s, a)$ gives probabilities for transitions.
5. $r : \mathcal{S} \to \mathcal{R}$ is a measurable function. It is called the reward function; $r(s)$ has the interpretation of the reward for the agent after getting to state s.

Definition 2. Let $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, r)$ be a Markov decision process. Then one can define a policy π and a distribution μ as follows:

- The policy $\pi : 2^{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ is a mapping satisfying: for a fixed $s \in \mathcal{S}$, $\pi(\cdot|s)$ is a probability measure on $2^{\mathcal{A}}$.
- μ is the initial distribution on $\mathcal{S}$.

Definition 3. A policy π is said to be *deterministic* if the probability measure $\pi(\cdot|s)$ is a Dirac measure. A policy π is said to be *soft* if $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The set of all policies is denoted by Π and the set of all deterministic policies is denoted by Π_D. Also if $\pi \in \Pi_D$, then we abuse notation, and consider π as a mapping from $\mathcal{S}$ to $\mathcal{A}$, which maps each state s to the action a which has probability 1.

Remark 4. There are a couple of remarks that should be made.

- By $P(s'|s, a)$ we actually mean $P(\{s'\}|s, a)$ and the same is true for other probability measures.
- The probability measure $P(\ |\ )$ does not always correspond to a conditional probability distribution, even though the bar suggests otherwise.
- Given a MDP and a policy π one immediately gets a new kind of transition probability $p : 2^{\mathcal{S}\times\mathcal{A}} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which is defined as follows:

  $$p(S \times A|s, a) := \sum_{s' \in S} P(s'|s, a)\pi(A|s'). \tag{1}$$

  These objects turn out to be very useful and therefore they will get a name, namely: *state-action transition kernels*. The name will be clear later.

To a Markov decision process we can associate a Markov chain which has the interpretation of a stochastic process describing the movement of an agent through the state space $\mathcal{S}$. To that end, we first define a couple of concepts.

Definition 5. Let $(E, \mathcal{G})$ be a measurable space with E a finite set. A function $\lambda : \mathcal{G} \times E \to \mathbb{R}$ is called a *Markov kernel* if:

- For each $x \in E$ the mapping $B \mapsto \lambda(B, x)$ is a probability measure on $(E, \mathcal{G})$.
- For each $B \in \mathcal{G}$ the function $\lambda(B, \cdot)$ is a measurable function from $(E, \mathcal{G})$ to $(\mathbb{R}, \mathcal{B})$.

Theorem 6. Consider a Markov kernel λ on $(E, \mathcal{G})$. Let α be an initial distribution on $(E, \mathcal{G})$. Then there exists a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and a time-homogeneous Markov chain $(Y_k)_{k \geq 0}$ on that probability space taking values in E with initial distribution α and transition probabilities from i to j given by $\lambda(i, j)$.

Proof. The existence of the Markov chain given transition probabilities is a standard result in the theory of Markov chains; it is Corollary 14 in Chapter 1 of [10]. $\square$

Proposition 7. The state-action transition kernel p defined in (1) is a Markov kernel.

Proof. The proof follows from the way P and π are defined and the fact that $\mathcal{S}$ is a finite set. Therefore the proof will be omitted. $\square$

It seems that we are close to applying Theorem 6, but we need to define an initial distribution on $\mathcal{S} \times \mathcal{A}$ first. Define a probability measure ν on $(\mathcal{S} \times \mathcal{A}, 2^{\mathcal{S} \times \mathcal{A}})$ as follows:

$$\nu(S \times A) := \sum_{s \in S} \mu(s)\pi(A|s). \tag{2}$$

where μ is the initial distribution on $\mathcal{S}$. The probability measure ν will be our initial distribution on $\mathcal{S} \times \mathcal{A}$.

Now applying Theorem 6 with p being the Markov kernel and ν being the initial distribution gives us a Markov chain $(S_k, A_k)_{k \geq 0}$ on some measure space, say $(\Omega, \mathcal{F}, \mathbb{P})$, taking values in $\mathcal{S} \times \mathcal{A}$. With this theorem we see that the short description in the introduction actually makes sense.

2.2 Action-value function

The Markov chain $(S_k, A_k)_{k \geq 0}$ induces another stochastic process $(r(S_{k+1}))_{k \geq 0}$ which is the reward process. We define a sequence of random variables, namely the *discounted return* $(G_k)_{k \geq 0}$, which is defined as $G_0 = 0$ and for $k \geq 1$ as follows:

$$G_k := \sum_{j=0}^{\infty} \gamma^j r(S_{k+j}). \tag{3}$$

where $\gamma \in (0, 1)$ is called the *discount factor*. We do not consider the case $\gamma = 0$, because it is not really interesting. The discount factor makes sure that G_k is well-defined as a proper random variable, namely one has

$$|G_k| \leq \sum_{j=0}^{\infty} \gamma^j |r(S_{k+j})| \leq \sum_{j=0}^{\infty} \gamma^j \sup_s |r(s)| = \frac{\sup_s |r(s)|}{1 - \gamma} < \infty,$$

where we have used that $\{|r(s)| : s \in \mathcal{S}\}$ is finite which means that the supremum is attained. Since the random variable G_k is bounded, it is also integrable, i.e. the expectation of G_k is well-defined.

The random variable G_k is the sum of the discounted rewards after the k-th time step. One needs to say something about the expectation of G_k. One might want to answer: "What policy will maximize the mean of G_{k+1}, say, given that I'm now at $(S_k, A_k) = (s, a)$?".This question can be answered with the use of a so-called action-value function (see also [3]).

Definition 8. Consider an MDP and a policy π. Now one can define the *action-value function* $q_\pi \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ as follows. Take an initial distribution λ on $\mathcal{S}\times\mathcal{A}$ such that $\lambda(s,a)=1$ so that one gets a Markov chain $(S_k, A_k)_{k\geq 0}$. We then define $q_\pi(s,a)$ as

$$q_\pi(s,a) := \mathbb{E}[G_1 \mid S_0 = s, A_0 = a]. \tag{4}$$

The interpretation $q_\pi(s,a)$ has is the expected discounted return given one is in position s and does action a and follows policy π afterwards.

In a similar way one can define the *value function* $v_\pi \in \mathbb{R}^{\mathcal{S}}$ for each $s \in \mathcal{S}$ as follows:

$$v_\pi(s) := \mathbb{E}[G_1 \mid S_0 = s]. \tag{5}$$

As one might expect, q_π and v_π are related to each other. The important relations are gathered in one lemma.

Lemma 9. The functions q_π and v_π satisfy the following relations:

(i) q_π *is a non-trivial affine transformation of itself*:

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S}} r(s')P(s'|s,a) + \gamma \sum_{(s',a')\in\mathcal{S}\times\mathcal{A}} q_\pi(s',a')p(s',a'|s,a)$$

(ii) v_π *is a non-trivial affine transformation of itself*:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} v_\pi(s')P(s'|s,a)$$

(iii) q_π *is an affine transformation of* v_π:

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S}} P(s'|s,a)\left[r(s') + \gamma v_\pi(s')\right]$$

(iv) v_π *is an affine transformation of* q_π:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} q_\pi(s,a)\pi(a|s)$$

Proof. All of the equations above can be obtained by the same method. Therefore only the proof of (i) is given. Recall that we have a Markov chain $(S_k, A_k)_{k\geq 0}$ which we have used to define q_π. Notice that

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[r(S_1) + \gamma G_2 \mid S_0 = s, A_0 = a] \\ &= \sum_{s'\in\mathcal{S}} r(s')P(s'|s,a) + \gamma \sum_{(s',a')\in\mathcal{S}\times\mathcal{A}} \mathbb{E}[G_2 \mid S_1 = s', A_1 = a']p(s',a'|s,a), \end{aligned}$$

where we have used the Markov property. Moreover the last summation runs over all $(s',a') \in \mathcal{S}\times\mathcal{A}$ such that $p(s',a'|s,a) > 0$ to make the conditional expectation well-defined. Similarly for the first summation. Also notice that the conditional distribution of G_2 given $\{S_1 = s', A_1 = a'\}$ is equal to the conditional distribution $\hat{G}_1$ given $\{\hat{S}_0 = s', \hat{A}_0 = a'\}$ where $(\hat{S}_k, \hat{A}_k)_{k\geq 0}$ is a Markov chain induced by a different initial distribution, namely one for which there is positive probability for the event $\{\hat{S}_0 = s', \hat{A}_0 = a'\}$. But the expectation of that is $q_\pi(s',a')$, hence

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S}} r(s')P(s'|s,a) + \gamma \sum_{(s',a')\in\mathcal{S}\times\mathcal{A}} q_\pi(s',a')p(s',a'|s,a).$$

□

Now we can compare different policies, but we need to have a notion to what we call the optimal policy which will be given in the following definition.

Definition 10. A policy π is called optimal if for all policies π'

$$v_{\pi'}(s) \leq v_{\pi}(s) \quad \text{for all } s \in \mathcal{S}. \tag{6}$$

Although an optimal policy might not be unique, one denotes it with π_*.

Remark 11. The optimal value function will be written as v_* and it satisfies

$$v_*(s) = \max_{\pi} v_{\pi}(s) \quad \text{for all } s \in \mathcal{S}. \tag{7}$$

and the optimal action-value function will be written as q_* and it satisfies

$$q_*(s,a) = \max_{\pi} q_{\pi}(s,a) \quad \text{for all } (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{8}$$

How to find such policy? Does it exist? Its existence is a standard result [1]. Finding the optimal policy is a little bit harder than finding v_* and q_* as we can see from the following theorem.

Theorem 12. The functions v_* and q_* satisfy the so-called Bellman equations. That is, they satisfy

$$v_*(s) = \max_{a \in \mathcal{A}} \mathbb{E}[r(S_1) + \gamma v_*(S_1) \mid S_0 = s, A_0 = a] \quad \text{for all } s \in \mathcal{S}, \tag{9}$$

and

$$q_*(s,a) = \mathbb{E}[r(S_1) + \gamma \max_{a'} q_*(S_1, a') \mid S_0 = s, A_0 = a] \quad \text{for all } (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{10}$$

Proof. For a proof of this theorem, see for instance Section 5.4 in [1]. $\square$

This theorem gives us a finite system of equations for the optimal (action-) value function. For example, the system of equations for q_* consists of $|\mathcal{S} \times \mathcal{A}|$ equations. It might still be hard to solve for them. However these equations can help finding the solution and prove its existence through Banach fixed-point theorem.

To that end we define the operator $T : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ as follows, given $q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$, we define $T[q]$ by

$$T[q](s,a) = \mathbb{E}[r(S_1) + \gamma \max_{a'} q(S_1, a') \mid S_0 = s, A_0 = a]. \tag{11}$$

Clearly q_* is a fixed point of T by the Bellman equations. Moreover we have the following Lemma.

Lemma 13. The operator T defined in (11) is a contraction with contraction constant γ with respect to the supremum norm on $\mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ (which is denoted by $\|\cdot\|_\infty$).

Proof. We start by writing the definition of the maximum norm and using the triangle inequality for integrals afterwards. We proceed as follows:

$$\begin{aligned} \|Tq_1 - Tq_2\|_\infty &= \gamma \max_{s,a} |\mathbb{E}[\max_{a_1} q_1(S_1, a_1) - \max_{a_2} q_2(S_1, a_2) \mid S_0 = s, A_0 = a]| \\ &\leq \gamma \max_{s,a} \mathbb{E}[|\max_{a_1} q_1(S_1, a_1) - \max_{a_2} q_2(S_1, a_2)| \mid S_0 = s, A_0 = a]. \end{aligned}$$

Now notice that

$$|\max_{a_1} q_1(S_1, a_1) - \max_{a_2} q_2(S_1, a_2)| \leq \max_{s'} |\max_{a_1} q_1(s', a_1) - \max_{a_2} q_2(s', a_2)| \quad \text{a.s.}$$

Hence

$$\begin{aligned}\|Tq_1 - Tq_2\|_\infty &\leq \gamma \max_{s,a} \mathbb{E}[\max_{s'} |\max_{a_1} q_1(s', a_1) - \max_{a_2} q_2(s', a_2)| \mid S_0 = s, A_0 = a] \\ &= \gamma \max_{s'} |\max_{a_1} q_1(s', a_1) - \max_{a_2} q_2(s', a_2)|.\end{aligned}$$

Assume without loss of generality that $\max_{a_1} q_1(s, a_1) \geq \max_{a_2} q_2(s, a_2)$. Choose $a_0 \in \mathcal{A}$ arbitrarily. Then one has

$$\begin{aligned}|\max_{a_1} q_1(s, a_1) - \max_{a_2} q_2(s, a_2)| &= \max_{a_1} q_1(s, a_1) - \max_{a_2} q_2(s, a_2) \\ &= \max_{a_1}\{q_1(s, a_1) - \max_{a_2} q_2(s, a_2)\} \\ &\leq \max_{a'}\{q_1(s, a') - q_2(s, a')\} \\ &\leq \max_{a'} |q_1(s, a') - q_2(s, a')|.\end{aligned}$$

Therefore

$$\|Tq_1 - Tq_2\|_\infty \leq \gamma \max_{s'} \max_{a'} |q_1(s', a') - q_2(s', a')| = \gamma \|q_1 - q_2\|_\infty.$$

We conclude T is a contraction with contraction constant $\gamma \in (0, 1)$. □

Now one gets the existence of q_* by Banach fixed-point theorem. Once one has q_*, then an optimal policy π_* can be found by setting:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a'} q_*(s, a'), \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

This policy clearly satisfies $q_{\pi_*} = q_*$. It is not clear yet that it is optimal with respect to the value function as well. From part (iv) of Lemma 9, we know that for an arbitrary policy π that the following holds:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} q_\pi(s, a)\pi(a|s) \leq \sum_{a \in \mathcal{A}} q_{\pi_*}(s, a)\pi(a|s) \leq \max_{a \in \mathcal{A}} q_{\pi_*}(s, a) = \sum_{a \in \mathcal{A}} q_{\pi_*}(s, a)\pi_*(a|s) = v_{\pi_*}(s). \tag{13}$$

This holds for all s, hence π_* is optimal with respect to the value function as well.

These are the definitions we need. After this the theory will be applied to a particular, rather simple, example which illustrates the theory very well.

2.3 Mountain car problem

We will apply the theory of Markov decision processes on a very well-known problem from reinforcement learning, the *mountain car problem.* There are many formulations of the problem, but they all boil down to the same thing. In this section we give one particular formulation of the problem.

Let $f : [-1, 1] \to \mathbb{R}$ be an even function which is sufficiently many times continuously differentiable, strictly increasing on $(0, 1]$ and satisfying $f(0) = 0$. The graph of this function is considered as a valley between two mountains. A car is placed in the middle of the valley, namely at the point $(0, 0)$. The goal is to let the driver "learn" how to get to the most right-end point $(1, f(1))$. To make the problem interesting the engine of the car is set to be so weak that the car cannot get to $(1, f(1))$ by just moving forwards.

We will not solve the problem in this general form. We consider a particular choice of the function f, namely $f(x) = x^2$. The performance of the engine of the car is a parameter in terms of the maximum acceleration of the car.

2.3.1 Dynamics of the car

The dynamics of the car is important for finding the transition probabilities P (which turn out to be deterministic). For example, one needs to find out where the car will be next if one presses the gas pedal. The main focus is the dynamics on the x-axis, since the dynamics on the y-axis can be found through $y = f(x)$. We will use some results from physics in order to make the dynamics of the car realistic. To that end we first assume that the car has mass m. Let $\beta : [0, \infty) \to \mathbb{R}$ be a function that translates the action of the driver is making in terms of the car's acceleration. Then one can define the vectorial force that is coming from the car, $F_\beta : [0, \infty) \to \mathbb{R}^2$ as follows:

$$F_\beta(t) := m\frac{\beta(t)}{\sqrt{1 + [f'(x(t))]^2}} \begin{pmatrix} 1 \\ f'(x(t)) \end{pmatrix},$$

where $x(t)$ is the x-position of the car at time t. For example, if the driver steps on the gas pedal to go forward at time t then $\beta(t)$ is a positive number whose magnitude corresponds with how how far the driver is pushing down the pedal. This vectorial function F_β and β will be used to determine the position of the car.

Let $G : [0, \infty) \to \mathbb{R}^2$ be the gravitational force on time t and $N : [0, \infty) \to \mathbb{R}^2$ be the normal force on time t. Clearly the following holds:

$$G(t) = m \begin{pmatrix} 0 \\ -g \end{pmatrix},$$

where g is the gravitational acceleration. Moreover N is orthogonal to F_β, so

$$N(t) = m\frac{n(t)}{\sqrt{1 + [f'(x(t))]^2}} \begin{pmatrix} -f'(x(t)) \\ 1 \end{pmatrix},$$

for some function $n : [0, \infty) \to \mathbb{R}$. The net force F satisfies then $F = G + N + F_\beta$. By Newton's Second Law, we have

$$F(t) = m \begin{pmatrix} \ddot{x}(t) \\ \ddot{y}(t) \end{pmatrix}.$$

One has $y(t) = f(x(t))$, i.e. the car stays on the surface. The second derivative of $y(t)$, is then $\ddot{y}(t) = \ddot{x}(t)f'(x(t)) + (\dot{x}(t))^2 f''(x(t))$. Notice that $F = G + N + F_\beta$ is a system of two equations where the function $n(t)$ is still unknown. Using the second component of the equation $F = G + N + F_\beta$, we find

$$n(t) = \sqrt{1 + [f'(x(t))]^2}\ \left[\ddot{x}(t)f'(x(t)) + (\dot{x}(t))^2 f''(x(t)) + g\right] - \beta(t)f'(x(t)).$$

Now we find a differential equation for $x(t)$ using the first component of the equation $F = G + N + F_\beta$, namely:

$$\ddot{x}(t) = \frac{\beta(t)}{\sqrt{1 + [f'(x(t))]^2}} - \frac{f'(x(t))\left[g + (\dot{x}(t))^2 f''(x(t))\right]}{1 + [f'(x(t))]^2}.$$

Let $v = \dot{x}$ then we have

$$\frac{d}{dt}\begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \frac{\beta(t)}{\sqrt{1+[f'(x(t))]^2}} - \frac{f'(x(t))\left[g+(v(t))^2 f''(x(t))\right]}{1+[f'(x(t))]^2} \end{pmatrix}. \tag{14}$$

This ordinary differential equation determines the exact position of the car after taking the "actions" $\beta(t)$ and starting on position $x(0)$ with velocity $v(0)$.

2.3.2 MDP formulation of the problem

We see that the definition of MDP assumes a finite state space. It is convenient to describe the state of the car in terms of $(x, \dot{x})$ with $x \in [-1, 1]$. However, the environment as described above is uncountable. The other thing is that the variable t that is in the ODE in Equation (14) is continuous time while the MDP is only defined for discrete time steps. So it is necessary to do some preparation work before we can construct an MDP.
Consider the following state space:

$$\mathcal{S} = \{\xi_0, ..., \xi_N\} \times \{\eta_0, ..., \eta_N\}.$$

This set is a discretization version of the set $[-1, 1] \times \mathcal{V}$ where $\mathcal{V} \subset \mathbb{R}$ is the set that $\dot{x}$ can take. The set $\{\eta_0, ..., \eta_N\}$ is bounded and discrete version of $\mathcal{V}$ which will be chosen appropriately. The discretization is done so that the distance between neighboring points is equal. We also require the point $(0, 0)$ to be in there, which is guaranteed by choosing N to be an even number. This discretization has two parameters, namely N and η_N.

The actions that the driver can choose are: driving backward, staying still or driving forward; these actions get the values -1, 0 and 1 respectively. Therefore the set of actions will be $\mathcal{A} = \{-1, 0, 1\}$.

Now we give a procedure to find the function P. If the driver is in state S_k and chooses to do action $A_k \in \mathcal{A}$, then we will assume that from time step k to $(k + 1)$ the force is constant. In terms of F_β, that means that $\beta(t) \equiv \beta A_k$ with $\beta > 0$. Notice the abuse of notation when we defined β as a number, it is the constant magnitude which is a parameter. To find the next state, we solve the ODE in Equation (14) with initial values $(x(0), \dot{x}(0)) = S_k$ and get the values $(x(\tau), \dot{x}(\tau))$ where $\tau > 0$. The number τ is the real time that corresponds to the discrete time between k and $(k + 1)$ and we will leave it as a parameter.

The remaining problem is that $(x(\tau), \dot{x}(\tau))$, which appears to be the next state, is not an element of $\mathcal{S}$ in general. An easy way to get over this is to set:

$$S_{n+1} = \text{argmin}_{s \in \mathcal{S}} \|(x(\tau), \dot{x}(\tau)) - s\|_2,$$

which is by definition an element of $\mathcal{S}$. Using that, we don't get $(x(\tau), \dot{x}(\tau))$ as the next state, but an element of $\mathcal{S}$ that is closest to it. Actually, the formulation must be in terms of P, but it is redundant as long it is clearly described how S_{k+1} is distributed given $(S_k, A_k) = (s, a)$.

Finally the reward must be specified. The set of rewards is $\mathcal{R} := \{0, 1\}$. The car gets a reward of 0 all the time except if it goes to $(1, \eta)$ for all $\eta \in \{\eta_0, ..., \eta_N\}$, then it is equal to 1. In other words, $r(1, \eta) = 1$ and zero otherwise. The car will be forced to go to the most right-end point by giving the largest reward if it gets there. That is the goal and therefore for every state $S_n = (1, \eta)$ the next state will be $S_{n+1} = (1, \eta)$, this means that the car stays in there after achieving its goal. What could physically happen is that the car goes down again, but that is permitted.

Now the mountain car problem is formulated in terms of MDP. What remains now is finding the optimal policy. In the next sections we consider methods to find the optimal policy. We will apply these methods to the mountain car problem. The parameters that will be used are given in Table 1.

Table 1: Model parameters

Parameter	**Value**
N	20
η_N	3
β	2.5
τ	0.1
γ	0.5

With this choice of parameters, in particular β, the car can not get to the mountaintop in one run. As we can see from Figure 1, the car can get to $(x, y) = (0.27, 0.075)$ if the driver drives forward starting at position

$(0, 0)$. There are oscillations as we would expect. Also the role of τ is a little bit visible from the figure, namely one has $x(\tau) = 0.0123$. That means that in the first step of the MDP, the car can get to position $(x, y) = (0.0123, 1.51 \cdot 10^{-4})$.

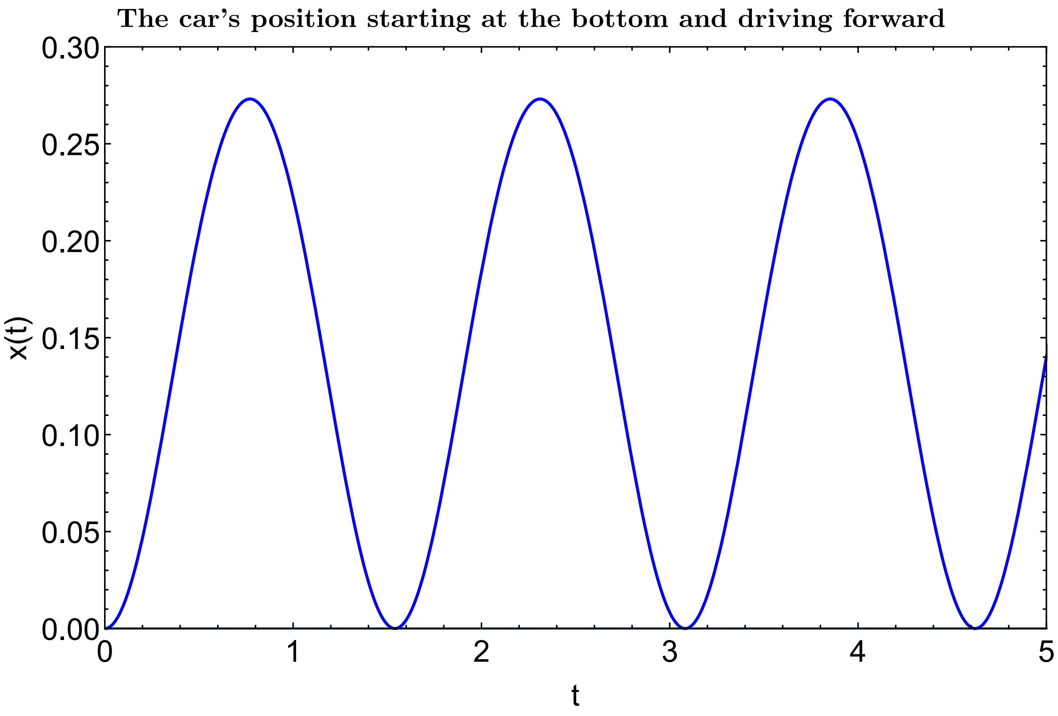


Figure 1: Numerical solution of the differential equation describing the movement of the car in case that the car starts at position $(0, 0)$ and drives forward.

In this section we have given a formal definition of an MDP. There are two types of measure of performance of policies which will be used throughout the whole report. After that, we have discussed an example, the mountain car problem, in terms of an MDP. In the next sections we give various algorithms to solve for the optimal policy. At last we apply them on the mountain car problem to test how the algorithms perform.

3 Dynamic Programming

There are several algorithms that assume the agent knows the dynamics of the environment plus the reward that it will be given. That means that the agent knows the function P and r. The algorithms that we consider are called *dynamic programming.* The goal, in those algorithms, is to find v_* and the optimal policy π_*. In the case where P and r are known functions, the Bellman equations are only unknown in v_* (see Theorem 12). So one could theoretically solve for v_*.

To illustrate the idea, define the operator $J : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ as follows:

$$J[v](s) = \max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s') + \gamma v(s') \right] \right\}. \tag{15}$$

Notice that v_* is a fixed point of J because of Theorem 12, i.e. the Bellman equations. From Proposition 14, which is stated below, one knows that J is a contraction with parameter γ. So for an arbitrary $V_0 \in \mathbb{R}^{\mathcal{S}}$ the iteration $V_n = JV_{n-1}$ has limit v_* as $n \to \infty$ by the Banach fixed-point theorem.

After that one finds the optimal policy by the relation of q_* and v_* given in part (iii) of Lemma 9. This algorithm is called *value iteration.* This is the first algorithm that we will consider.

Thereafter we discuss another algorithm, which is called *policy iteration.* It uses the fact that there exists a deterministic optimal policy. One also knows that there are $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies. So the idea is to use the fact that there is only a finite set of policies that should be considered and the Bellman equations.

For a policy $\pi \in \Pi_D$ we define the operator $J^\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ as follows for each $s \in \mathcal{S}$:

$$J^\pi[v](s) = \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s)) \left[r(s') + \gamma v(s') \right]. \tag{16}$$

Notice that we use π as a mapping from $\mathcal{S}$ to $\mathcal{A}$. By the second part of Lemma 9 we deduce that for a deterministic policy π, the value function v_π is a fixed point of J^π. This fact plays an important role for what will come later. The next proposition summarizes the important properties of the operator J and J^π.

Proposition 14. Consider the operators J and J^π defined in (15) and (16) respectively. Then both operators are:

(i) monotone, i.e. for $v \leq w$ one has $Jv \leq Jw$ and $J^\pi v \leq J^\pi w$.

(ii) a contraction with contraction constant γ with respect to the supremum norm on $\mathbb{R}^{\mathcal{S}}$. In particular, the operators are continuous.

Proof. The proof will be given only for the operator J. For (i), let $v \leq w$ be given. Then using that P is a non-negative function and $\gamma \geq 0$, one concludes $\gamma v(s')P(s'|s,a) \leq \gamma w(s')P(s'|s,a)$. Summing over $s' \in \mathcal{S}$ one gets $\gamma \sum_{s'} v(s')P(s'|s,a) \leq \gamma \sum_{s'} w(s')P(s'|s,a)$. Adding $\sum_{s'} r(s')P(s'|s,a)$ and taking the maximum over $a \in \mathcal{A}$ on both sides yields the desired inequality: $Jv \leq Jw$.

For (ii) one needs to remark that $|\max_x f(x) - \max_y g(y)| \leq \max_x |f(x) - g(x)|$, which is proven in the proof of Lemma 13. Using the previous observation, we can write

$$\begin{aligned} \|Jv - Jw\|_\infty &= \max_{s \in \mathcal{S}} \left| \max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s') + \gamma v(s') \right] \right\} - \max_{a' \in \mathcal{A}} \left\{ \sum_{s'' \in \mathcal{S}} P(s''|s,a) \left[r(s'') + \gamma w(s'') \right] \right\} \right| \\ &\leq \gamma \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} (v(s') - w(s'))P(s'|s,a) \right|. \end{aligned}$$

Using the triangle inequality one gets

$$\|Jv - Jw\|_\infty \leq \gamma \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} |v(s') - w(s')| P(s'|s,a).$$

We take the maximum of $|v(s') - w(s')|$ to get it out from the sum and we proceed as follows:

$$\begin{aligned}\|Jv - Jw\|_\infty &\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \max_{s''\in\mathcal{S}} \sum_{s'\in\mathcal{S}} |v(s'') - w(s'')| P(s'|s,a) \\ &\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \max_{s''\in\mathcal{S}} |v(s'') - w(s'')| \sum_{s'\in\mathcal{S}} P(s'|s,a) \\ &= \gamma \|v - w\|_\infty \cdot 1.\end{aligned}$$

This proves that J is a contraction. Similar arguments can be used for J^π. □

3.1 Value iteration

Value iteration is an algorithm that is very accurate in the sense that one can know exactly how accurate the approximation is depending on the number of iterations. Therefore we use this algorithm as a baseline for all other algorithms in the mountain car problem. The algorithm, as taken from [3], is shown below.

```
Initialization:
$V(s) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}$. Take $\theta > 0$ small (measure for accuracy) and set $\Delta = \theta$

Value iteration:
repeat
    for $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'\in\mathcal{S}} P(s'|s,a)\,[r(s') + \gamma V(s')]$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
    end
until $\Delta < \theta$

Policy Output:
$\pi(s) \leftarrow \text{argmax}_a \sum_{s'\in\mathcal{S}} P(s'|s,a)\,[r(s') + \gamma V(s')]$ for all $s \in \mathcal{S}$
```

Algorithm 1: Value iteration

Remark 15. Such algorithm can also be used to find v_π for some policy π. We have to consider the recurrence relation $V_{n+1} = J^\pi V_n$. The results that we will see holds also for this iteration. We do not discuss this in full detail, but we will use it in Section 4.3 and Section 5.3 to compare policies.

First of all it is clear that this algorithm terminates due to the Banach fixed-point theorem, since the value iteration is just a fixed-point iteration. Now we discuss the convergence properties of this algorithm. Mathematically, the algorithm is basically given by $V_{n+1} = JV_n$ where n runs in each step of value iteration.

The next lemma tells us that if a value function v_π for a policy $\pi \in \Pi_D$ is really close to v_*, then it is the optimal value function itself. This result is especially due to the fact that Π_D is a finite set.

Lemma 16. There exists $\zeta > 0$, such that for every policy $\pi \in \Pi_D$ satisfying

$$\|v_\pi - v_*\|_\infty < \zeta, \tag{17}$$

implies that π is an optimal policy.

Proof. Notice that there are a finite number of policies $\pi \in \Pi_D$. Now let $C^* := \{\pi \in \Pi_D \mid v_\pi \neq v_*\}$. Since there exists an optimal policy $\pi_* \in \Pi_D$, one knows that $C^* \neq \Pi_D$. If $C^* = \emptyset$ we are done. On the other hand, if $C^* \neq \emptyset$, then $\min_{\pi\in C^*} \|v_\pi - v_*\|_\infty > 0$ for being a minimum over a set with finite elements which are all positive. Take $\zeta = \min_{\pi\in C^*} \|v_\pi - v_*\|_\infty > 0$. Then every $\pi \in \Pi_D$ satisfying $\|v_\pi - v_*\|_\infty < \zeta$ must be not in C^* meaning that $v_\pi = v_*$. □

Now we introduce a theorem which tells us how far the true value is from the real value. The theorem tells us that we indeed get an approximation of the optimal value function using the algorithm.

Theorem 17. If the value iteration terminates at time step n with approximation V_n and policy π, then

$$\|V_n - v_*\|_\infty \leq \frac{\gamma}{1-\gamma}\theta, \tag{18}$$

and

$$\|v_\pi - v_*\|_\infty \leq \frac{2\gamma^2\theta}{(1-\gamma)^2}. \tag{19}$$

Moreover, if $\theta < \zeta\frac{(1-\gamma)^2}{2\gamma^2}$, where $\zeta > 0$ satisfies the conditions of Lemma 16, then the policy π is optimal.

Proof. Notice that using the fact that J is contraction by Proposition 14, we get

$$\begin{aligned}
\|V_n - v_*\|_\infty &= \|JV_{n-1} - Jv_*\|_\infty \\
&\leq \gamma\|V_{n-1} - v_*\|_\infty \\
&\leq \gamma\|V_{n-1} - V_n + V_n - v_*\|_\infty \\
&\leq \gamma\|V_{n-1} - V_n\|_\infty + \gamma\|V_n - v_*\|_\infty.
\end{aligned}$$

If the algorithm terminates at time step n, then we must have $\|V_n - V_{n-1}\|_\infty \leq \theta$, hence

$$\|V_n - v_*\|_\infty \leq \frac{\gamma\theta}{1-\gamma},$$

proving the first inequality.

For the second inequality, first we notice that π is chosen greedily with respect to V_n which implies that $J^\pi V_n = JV_n$. The rest of the proof is basically applying the triangle inequality several times. We proceed as follows:

$$\begin{aligned}
\|v_\pi - v_*\|_\infty &= \|J^\pi v_\pi - J^\pi V_n + JV_n - Jv_*\|_\infty \\
&\leq \|J^\pi v_\pi - J^\pi V_n\|_\infty + \|JV_n - Jv_*\|_\infty \\
&\leq \gamma\|v_\pi - V_n\|_\infty + \gamma\|V_n - v_*\|_\infty \\
&\leq \gamma\|v_\pi - v_*\|_\infty + \gamma\|v_* - V_n\|_\infty + \gamma\|V_n - v_*\|_\infty \\
&\leq \gamma\|v_\pi - v_*\|_\infty + \frac{2\gamma^2\theta}{1-\gamma}.
\end{aligned}$$

Taking $\|v_\pi - v_*\|_\infty$ to the left-hand side finishes the proof of the inequality. Finally the fact that π is optimal under the additional condition on θ is a result from Lemma 16. □

The same technique can be used to obtain a bound depending on n. At the n-th iteration one has

$$\|V_n - v_*\|_\infty \leq \frac{\gamma^n}{1-\gamma}\|V_1 - V_0\|_\infty. \tag{20}$$

This tells us how many iterations we need to get a desired accuracy. This is the reason we have chosen this algorithm as a baseline. We can know how many iterations we need to get a good approximation beforehand.

3.2 Policy iteration

A pseudocode for the policy iteration algorithm, as shown in [3], is given below.

```
Initialization:
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$. Take $\theta > 0$ small and set $\Delta = \theta$

Policy Evaluation (PE):
repeat
    for $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \left[r(s') + \gamma V(s')\right]$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
    end
until $\Delta < \theta$

Policy Improvement (PI):
policy-stable$\leftarrow$ true
for $s \in \mathcal{S}$ do
    old-action$\leftarrow a(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[r(s') + \gamma V(s')\right]$
    if old-action $\neq \pi(s)$ then
        policy-stable $\leftarrow false$
    end
end
if policy-stable then
    return $V$ and $\pi$
else
    go to Policy Evaluation
end
```
Algorithm 2: Policy iteration

It might seem that the algorithm must terminate somewhere, but in fact there are many subtleties. It might be the case that the algorithm switches between two or more optimal policies all the time so that it never terminates [3]. That is easily fixed by adding additional flags. In other words by making use of the history of the sequence of policies, but those cases make the algorithm a little bit complicated. Therefore we decided to leave it out. If one adds a code which keeps track of the history of the policies then the algorithm terminates, since the set Π_D is finite.

First we show that, if the algorithm terminates, possibly with additional flags, then it gives an approximation of v_* with an approximation of its corresponding optimal policy π_*. Secondly, we show that the algorithm gives policies which get better than the previous ones with respect to the value function. After that, we show that the algorithm actually gives an exact value of v_* and π_*, if we would run the algorithm without the termination code, i.e. for infinitely long time. The termination code is the code that lets the algorithm terminate, in other words, the line where it says "**if** *policy-stable* **then**...".

Before going into the properties of the algorithm, we need a mathematical formulation of the algorithm. We start with π_0 and V_0 and get then the following recursion. In each step of policy evaluation, abbreviated by PE, we get V_{n+1} and π_{n+1}, i.e. the index increases. The same will happen in each step of policy improvement, abbreviated by PI. Indeed having V_n and π_n, we get the following in the next step, one either does PE:

$$\begin{cases} V_{n+1} &= J^{\pi_n} V_n, \\ \pi_{n+1} &= \pi_n. \end{cases} \tag{21}$$

or does PI and sets for each $s \in \mathcal{S}$:

$$\begin{cases} \pi_{n+1}(s) & = \operatorname{argmax}_a \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s') + \gamma V_n(s') \right], \\ V_{n+1}(s) & = V_n(s). \end{cases} \tag{22}$$

Notice that each PE loop terminates after we have found an n such that $\|V_n - V_{n-1}\|_\infty < \theta$. But is such n guaranteed to exist? Yes, notice that during PE there is only one policy that will be followed, say π, then the recursion given for V_n converges to the fixed point of J^π. That implies that V_n is a Cauchy sequence proving that the difference between two values gets arbitrarily small. This immediately implies that the algorithm switches between PE and PI for infinitely many times, if run without the termination code.

The following proposition gives a bound on the difference of the approximate value function and the exact value function.

Proposition 18. Let time step n be the index of the last step in a PE loop, then one has

$$\|V_n - v_{\pi_n}\|_\infty \leq \frac{\theta\gamma}{1-\gamma}. \tag{23}$$

Proof. We know that n was the last index that we have got. Since we were in a PE loop at the last step, we know that $\pi_n = \pi_{n-1}$. Hence one has $v_{\pi_n} = v_{\pi_{n-1}}$ which leads to:

$$\begin{aligned} \|V_n - v_{\pi_n}\|_\infty &= \|J^{\pi_{n-1}} V_{n-1} - J^{\pi_{n-1}} v_{\pi_{n-1}}\|_\infty \\ &\leq \gamma \|V_{n-1} - v_{\pi_{n-1}}\|_\infty \\ &\leq \gamma \|V_{n-1} - V_n + V_n - v_{\pi_n}\|_\infty \\ &\leq \gamma \|V_{n-1} - V_n\|_\infty + \gamma \|V_n - v_{\pi_n}\|_\infty \\ &\leq \gamma\theta + \gamma \|V_n - v_{\pi_n}\|_\infty. \end{aligned}$$

Hence

$$\|V_n - v_{\pi_n}\|_\infty \leq \frac{\theta\gamma}{1-\gamma}.$$

□

One can always choose V_0 such that $V_0 \leq J^{\pi_0} V_0$. Let $e : \mathcal{S} \to \mathbb{R}$ such that $e(s) = 1$ for all $s \in \mathcal{S}$. Now we set $V_0 = -\beta e$ for β large enough. Because one has

$$J^{\pi_0} V_0 = J^{\pi_0}[-\beta e] = -\gamma\beta e = -\beta e + (1-\gamma)\beta e = V_0 + (1-\gamma)\beta e.$$

For $\beta \to \infty$ one has: $(1-\gamma)\beta e(s) \to \infty$ meaning that for β large enough $(1-\gamma)\beta e(s) \geq 0$. Hence for such β one gets $J^{\pi_0} V_0 \geq V_0$. From now on, we assume that V_0 satisfies $V_0 \leq J^{\pi_0} V_0$. It makes the proof of the results that will be mentioned later easier, although it turns out that it is not always needed.

Before we get into the important theorems, we first introduce a couple of lemmas.

Lemma 19. If $V_0 \leq J^{\pi_0} V_0$, then for all $n \in \mathbb{N}$ one has $V_n \leq V_{n+1} \leq J^{\pi_{n+1}} V_{n+1}$.

Proof. The proof is inspired by a proof in [6] where the author uses it to prove a convergence property of a more general policy iteration algorithm. That being said, we prove it by induction. Let $n \in \mathbb{N}$ and assume that $V_n \leq J^{\pi_n} V_n$ holds. Then we show that $V_n \leq V_{n+1} \leq J^{\pi_{n+1}} V_{n+1}$ holds.

There are two cases, either the next step, i.e. the next index $n+1$, is PE or PI.

- *PE is done next.* One has $V_{n+1} = J^{\pi_n} V_n$, hence $V_n \leq V_{n+1}$. During a PE loop, the policy will be kept fixed, i.e. $\pi_n = \pi_{n+1}$, hence $J^{\pi_{n+1}} V_{n+1} = J^{\pi_n} V_{n+1}$. Also one has $J^{\pi_n} V_n \leq J^{\pi_n} V_{n+1}$ by the monotonicity of J^{π_n}. These imply that $V_{n+1} = J^{\pi_n} V_n \leq J^{\pi_n} V_{n+1} = J^{\pi_{n+1}} V_{n+1}$. This proves the claim for the case PE is done next.

- *PI is done next.* One has $V_{n+1} = V_n$ in this step. Notice that for each $s \in \mathcal{S}$ we have

$$\begin{aligned} J^{\pi_n}[V_n](s) &= \sum_{s' \in \mathcal{S}} P(s'|s, \pi_n(s)) \left[r(s') + \gamma V_n(s') \right] \\ &\leq \max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[r(s') + \gamma V_n(s') \right] \right\} \\ &= J[V_n](s), \end{aligned}$$

implying $J^{\pi_n} V_n \leq J V_n$. Finally notice that the policy π_{n+1} is found such that $J V_n = J^{\pi_{n+1}} V_n$. Now together with the induction hypothesis, we conclude: $V_{n+1} = V_n \leq J^{\pi_n} V_n \leq J V_n = J^{\pi_{n+1}} V_n = J^{\pi_{n+1}} V_{n+1}$.

We have proved the claim in both cases. □

As a consequence, the estimated value function is always bounded by the true value function.

Corollary 20. If $V_0 \leq J^{\pi_0} V_0$, then $V_n \leq v_{\pi_n}$ for all $n \in \mathbb{N}$.

Proof. Let $n \in \mathbb{N}$, then $V_n \leq V_{n+m}$ for $m \geq 0$. Assume one would never get in the PI loop after time step n, then that would mean that $\pi_{n+m} = \pi_n$ is constant for $m \geq 0$. Then one knows that $V_{n+m} \to v_{\pi_n}$ due to the PE loop and Banach fixed-point theorem. □

It is important to notice that the previous lemma does not say that the policies get better. It only says that the approximations of the value function gets better. But do the policies get better? In other words, do the value functions of the policies π_n satisfy $v_{\pi_n} \leq v_{\pi_{n+1}}$? One would question whether that is true, since the approximate value functions V_n are a little bit close to their true value v_{π_n}. That is the case, but in order to have such result we need to get the approximate value functions V_n close to v_{π_n} by choosing the accuracy parameter θ small enough. That is summarized in the next theorem.

Before we get into the formulation of the theorem, we define the set C as follows:

$$C := \{(s, \pi, \pi') \in \mathcal{S} \times \Pi_D \times \Pi_D \mid v_\pi(s) \neq v_{\pi'}(s)\}. \tag{24}$$

If this set is empty, then we do not need an algorithm to find the optimal policy, because all policies are optimal. However if this set is not empty, then it is a finite set for being a subset of the finite set $S \times \Pi_D \times \Pi_D$. That means that for all $(s, \pi, \pi') \in C$, one has $|v_\pi(s) - v_{\pi'}(s)| > 0$, so the minimum of such quantities is strictly positive. That leads to the definition of the constant $\theta_0 > 0$:

$$\theta_0 := \frac{1-\gamma}{\gamma} \min_{(s,\pi,\pi') \in C} |v_\pi(s) - v_{\pi'}(s)|. \tag{25}$$

Theorem 21. If $V_0 \leq J^{\pi_0}$ and $\theta < \theta_0$, then the policy iteration gives a sequence of policies π_n that get better with the time step, i.e. $v_{\pi_n} \leq v_{\pi_{n+1}}$ for all $n \in \mathbb{N}$.

Proof. Notice that during PE the policies remain the same. However, if one gets in the PI loop in step $n+1$, then π_n might be different from π_{n+1}. So assume that the $(n+1)$-th step is a policy iteration step. Now we prove by contradiction. Assume that for some $s \in \mathcal{S}$, one has $v_{\pi_n}(s) > v_{\pi_{n+1}}(s)$. Notice that one has $V_n \leq V_{n+1} \leq V_{n+2} \leq v_{\pi_{n+2}} = v_{\pi_{n+1}}$, because of the fact $\pi_{n+1} = \pi_{n+2}$ by the PE step that comes after the PI step and Lemma 19. Now one has

$$\begin{gathered} V_n(s) \leq v_{\pi_{n+1}}(s) \\ V_n(s) - v_{\pi_n}(s) \leq v_{\pi_{n+1}}(s) - v_{\pi_n}(s). \end{gathered}$$

Using Proposition 18, one gets

$$-\frac{\theta\gamma}{1-\gamma} \leq V_n(s) - v_{\pi_n}(s).$$

That implies

$$0 < v_{\pi_n}(s) - v_{\pi_{n+1}}(s) \leq \frac{\theta\gamma}{1-\gamma}. \tag{26}$$

By construction $(s, \pi_n, \pi_{n+1}) \in C$, hence

$$|v_{\pi_n}(s) - v_{\pi_{n+1}}(s)| < \min_{(s',\pi,\pi')\in C} |v_\pi(s') - v_{\pi'}(s')|. \tag{27}$$

This is clearly a contradiction. Therefore the conclusion: $v_{\pi_n} \leq v_{\pi_{n+1}}$. $\square$

Now we can formulate a theorem which roughly says that if a policy is obtained twice in the policy iteration algorithm, then that policy is very close to the optimal policy. Under some conditions we know that the policies improve, so getting one policy twice would intuitively mean that there is no possibility for improvement.

Theorem 22. If $\theta < \theta_0$ and the initialization of the algorithm satisfies $V_0 \leq J^{\pi_0}V_0$, then if a policy is obtained for the second time in time step n, then the following hold:

$$\|v_{\pi_n} - v_*\|_\infty \leq \frac{2\gamma^2\theta}{(1-\gamma)^2}, \tag{28}$$

and

$$\|V_n - v_*\|_\infty \leq \frac{\gamma(1+\gamma)\theta}{(1-\gamma)^2}. \tag{29}$$

In particular, if $\theta < \min\{\zeta\frac{(1-\gamma)^2}{2\gamma^2}, \theta_0\}$, where $\zeta > 0$ is from Lemma 16, then the policy π_n is optimal.

Proof. The assumption in the theorem says that there exists $m < n$ such that $v_{\pi_m} = v_{\pi_n}$. But from Theorem 21 one knows $v_{\pi_m} \leq v_{\pi_{m+1}} \leq ... \leq v_{\pi_n} = v_{\pi_m}$. Hence $v_{\pi_{n-1}} = v_{\pi_n}$. Now one gets

$$\begin{aligned}
\|v_{\pi_n} - v_*\|_\infty &= \|v_{\pi_n} - Jv_{\pi_n} + Jv_{\pi_n} - v_*\|_\infty \\
&= \|v_{\pi_{n-1}} - Jv_{\pi_{n-1}} + Jv_{\pi_n} - Jv_*\|_\infty \\
&\leq \|v_{\pi_{n-1}} - Jv_{\pi_{n-1}}\|_\infty + \|Jv_{\pi_n} - Jv_*\|_\infty \\
&\leq \|v_{\pi_{n-1}} - Jv_{\pi_{n-1}}\|_\infty + \gamma\|v_{\pi_n} - v_*\|_\infty.
\end{aligned}$$

We conclude

$$\|v_{\pi_n} - v_*\|_\infty \leq \frac{1}{1-\gamma}\|v_{\pi_{n-1}} - Jv_{\pi_{n-1}}\|_\infty. \tag{30}$$

Notice that we defined π_n to be the policy satisfying $JV_{n-1} = J^{\pi_n}V_{n-1}$. Now one can bound the remaining terms as follows:

$$\begin{aligned}
\|v_{\pi_{n-1}} - Jv_{\pi_{n-1}}\|_\infty &= \|v_{\pi_{n-1}} - J^{\pi_n}V_{n-1} + JV_{n-1} - Jv_{\pi_{n-1}}\|_\infty \\
&\leq \|v_{\pi_{n-1}} - J^{\pi_n}V_{n-1}\|_\infty + \|JV_{n-1} - Jv_{\pi_{n-1}}\|_\infty \\
&= \|v_{\pi_n} - J^{\pi_n}V_{n-1}\|_\infty + \|JV_{n-1} - Jv_{\pi_{n-1}}\|_\infty \\
&\leq \|J^{\pi_n}v_{\pi_n} - J^{\pi_n}V_{n-1}\|_\infty + \|JV_{n-1} - Jv_{\pi_{n-1}}\|_\infty \\
&\leq \gamma\|v_{\pi_n} - V_{n-1}\|_\infty + \gamma\|V_{n-1} - v_{\pi_{n-1}}\|_\infty \\
&= \gamma\|v_{\pi_{n-1}} - V_{n-1}\|_\infty + \gamma\|V_{n-1} - v_{\pi_{n-1}}\|_\infty \\
&= 2\gamma\|V_{n-1} - v_{\pi_{n-1}}\|_\infty.
\end{aligned}$$

In time step $n-1$, then the PE loop was finished, hence using Proposition 18 one gets

$$\|v_{\pi_n} - v_*\|_\infty \le \frac{1}{1-\gamma}\|v_{\pi_{n-1}} - Jv_{\pi_{n-1}}\|_\infty \le \frac{2\gamma}{1-\gamma}\|V_{n-1} - v_{\pi_{n-1}}\|_\infty = \frac{2\gamma^2\theta}{(1-\gamma)^2}. \tag{31}$$

Now if $\theta < \min\{\zeta\frac{(1-\gamma)^2\zeta}{2\gamma^2}, \theta_0\}$, then one has $\|v_{\pi_n} - v_*\|_\infty < \zeta$. The claim that π_n is optimal follows from Lemma 16. For the second inequality, we first notice that $V_n = V_{n-1}$ due to the PI step, hence

$$\|V_n - v_*\|_\infty = \|V_{n-1} - v_*\|_\infty \le \|V_{n-1} - v_{\pi_{n-1}}\|_\infty + \|v_{\pi_n} - v_*\| \le \frac{\theta\gamma}{1-\gamma} + \frac{2\gamma^2\theta}{(1-\gamma)^2} = \frac{\gamma(1+\gamma)\theta}{(1-\gamma)^2}. \tag{32}$$

□

From this theorem we understand that if the policy iteration terminates, either by seeing the same policy in a row, or by recording which policies we have seen before, then the algorithm has terminated for "a good reason" (as long as θ is small enough of course).

From the previous results, we see that the approximation can be arbitrarily close to the true values by choosing the "right" value for θ. However, there is a stronger result which states that the value function approximate V_n gets close to v_* as we run the algorithm for an infinite number of steps, i.e. without the termination code.

Theorem 23. Consider a policy iteration algorithm without the termination code. If $\theta < \min\{\zeta\frac{(1-\gamma)^2}{2\gamma^2}, \theta_0\}$ and the initialization satisfies $V_0 \le J^{\pi_0}V_0$, then one has $\lim_{n\to\infty} V_n = v_*$ in $(\mathcal{S}, \|\cdot\|_\infty)$. Moreover there exists some K such that for all $n > K$, the policy π_n is optimal.

Remark 24. Before going to the proof, let us give some remarks.

- The condition $V_0 \le J^{\pi_0}V_0$ and $\theta < \min\{\zeta\frac{(1-\gamma)^2}{2\gamma^2}, \theta_0\}$ is not needed for the result as it can be seen in [4]. However, these conditions make the proof a little bit less technical. That means that we should not have to worry much about the initialization in practice.
- The statement that we get an optimal policy, does not mean that the policy that we get is constant, i.e. $\pi_n = \pi_{n+1}$ for all $n > K$. Rather, it might switch between multiple optimal policies all the time.
- In the case that there is only one deterministic optimal policy, this theorem says that the policy iteration algorithm as given in the pseudocode terminates at some point.

Proof. This proof is mainly due to [6]. Notice that by Corollary 20, we have $V_n \le v_{\pi_n}$. But then we conclude $V_n \le v_*$ for all $n \in \mathbb{N}$. Moreover for each $s \in \mathcal{S}$ one has $V_n(s) \le V_{n+1}(s) \le v_*(s)$ by Lemma 19. An increasing sequence which is bounded from above has a limit, so $V(s) := \lim_{n\to\infty} V_n(s)$ exists. Since $\mathcal{S}$ is finite, this pointwise convergence can be improved to uniform convergence. So $\lim_{n\to\infty} V_n = V$. Since the operator J is continuous, we get $\lim_{n\to\infty} JV_n = JV$. The limiting V satisfies

$$V_n \le V \le JV. \tag{33}$$

One has $V \le JV$, but if we prove $V = JV$ then we are done. Assume indirectly that there is some $s \in \mathcal{S}$ such that $V(s) < J[V](s)$. Since J is continuous there is some n_* such that for $n_* < n$ one has $V(s) < J[V_n](s)$. Now we take $n' > n_*$ such that at time step n' PI is done. That means that we have a policy $\pi_{n'}$ such that $J^{\pi_{n'}}V_{n'-1} = JV_{n'-1}$. Then at time step $n'+1$ we get in a PE loop and get $V_{n'+1} = J^{\pi_{n'}}V_n$. Due to monotonicity one has $J^{\pi_{n'}}V_{n'-1} \le J^{\pi_{n'}}V_{n'}$, hence

$$V_{n'+1}(s) \ge J^{\pi_{n'}}[V_{n'-1}](s) = J[V_{n'-1}](s) > V(s). \tag{34}$$

This inequality contradicts (33). So we must have $V(s) = J[V](s)$ for all $s \in \mathcal{S}$ meaning that V is the fixed point of J. The fixed point is unique, so

$$\lim_{n\to\infty} V_n = V = v_*.$$

For the second part notice that $\zeta - \frac{\theta\gamma}{1-\gamma} > 0$ by the choice of θ. Moreover there exists $n_0 > 0$ such that for $n > n_0$ one has $\|V_n - v_*\|_\infty < \zeta - \frac{\theta\gamma}{1-\gamma}$. There exists $K > n_0$ such that at time index K we are in the last step of a PE loop. Therefore using Proposition 18 one gets

$$\|v_{\pi_K} - v_*\|_\infty \leq \|v_{\pi_K} - V_K\|_\infty + \|V_K - v_*\|_\infty < \frac{\theta\gamma}{1-\gamma} + \zeta - \frac{\theta\gamma}{1-\gamma} = \zeta. \tag{35}$$

So the policy π_K must be optimal by Lemma 16. One also knows that $v_{\pi_K} \leq v_{\pi_n}$ for all $n > K$ by Theorem 21. That means that the policy π_n is optimal for all $n > K$. $\square$

The policy iteration algorithm is in general good to use. However, it does not always terminate as it is given in the pseudocode. Moreover adding additional flags to make sure that the algorithm terminates, may need a very large memory to store the history of the policies. Also there is no estimation for the right value of θ. However, by the last theorem, we know that the choice of θ is not a big issue, since running the algorithm for long enough also gives a good approximation.

3.3 Results

As mentioned in the beginning, we want to use the value iteration as a baseline. It is indeed the case that we cannot run the algorithm for infinitely long time, but we can make the error as small as machine precision. From Theorem 17 and Theorem 22 we know that the approximate optimal value function is $\frac{2\theta\gamma^2}{(1-\gamma)^2}$ away from the optimal value function. With the parameters chosen, that quantity is equal to 2θ (see Table 1 for the parameters). Therefor $\theta = 10^{-16}$ is a convenient choice. This choice holds for both algorithms, namely value iteration and policy iteration.

The initial function V_0 is the zero function. Notice that since the reward function takes the values 0 and 1, we may conclude that $V_0 \leq J^{\pi_0} V_0$ is satisfied for all initial policies π_0. We have chosen the initial policy π_0 to be the always-go-forward policy.

The number of iterations needed in value iteration is 56, while the number of iterations in policy iteration is 80 and 7 for PE and PI respectively. The policy iteration that we have used is without keeping track of the complete history of policies. In [3], it is claimed that policy iteration is usually faster than value iteration. Our result is not a contradiction anyways, but one would question why in this specific case we get something unusual. It might be the case that there are multiple policies which are optimal and at one point one of them is chosen twice in a row.

Denote V_{VI} and V_{PI} for the approximate value functions obtained from value iteration and policy iteration respectively. Since there are many values of order 10^{-17} and values of order 1, it is very hard to make a good visualization of the approximate value functions in terms of a contour plot for instance. A standard way to fix such problem is by taking the logarithm of the functions, so we consider $\log V_{VI}$ and $\log V_{PI}$ instead. The advantage of the logarithm is the fact it is strictly monotone increasing. Contour plots of $\log V_{VI}$ and $\log V_{PI}$ can be seen in Figure 2 and Figure 3 respectively. The use of contour plot is convenient since we need two components to describe the state space $\mathcal{S}$ in this particular example, namely the x-position and the x-velocity.

The reason why the logarithm of the value function looks like it does, is rather tricky. Several factors play a role. First of all a good understanding of the ordinary differential equations in Equation (14) is needed. Secondly, one has to take the rounding into account. At last, one has to be aware of the fact that each step in the MDP corresponds to a real time τ, which we have chosen to be 0.1. The second and last facts may mean that the car in some cases stays at the same x-position (or x-velocity) a couple of times, if the driver takes a particular action which is not necessarily staying still. For instance, the low values of the value function around the points $(x, \dot{x}) = (0.5, 1.5)$ might be very unintuitive at first glance. However, one thing that is happening there is that the car is stuck at one position. It is clear that the engine of the car is really weak from Figure 1. So, there is a kind of "fight" between the gravity and the fairly fast driving car. The rounding compromises both by rolling the situation back as it was when it started.

The figures look very much the same, in fact, one has

$$\|V_{VI} - V_{PI}\|_{\infty} \approx 2.78 \cdot 10^{-17}. \tag{36}$$

This is as small as machine precision. We may consider the two value functions equal.

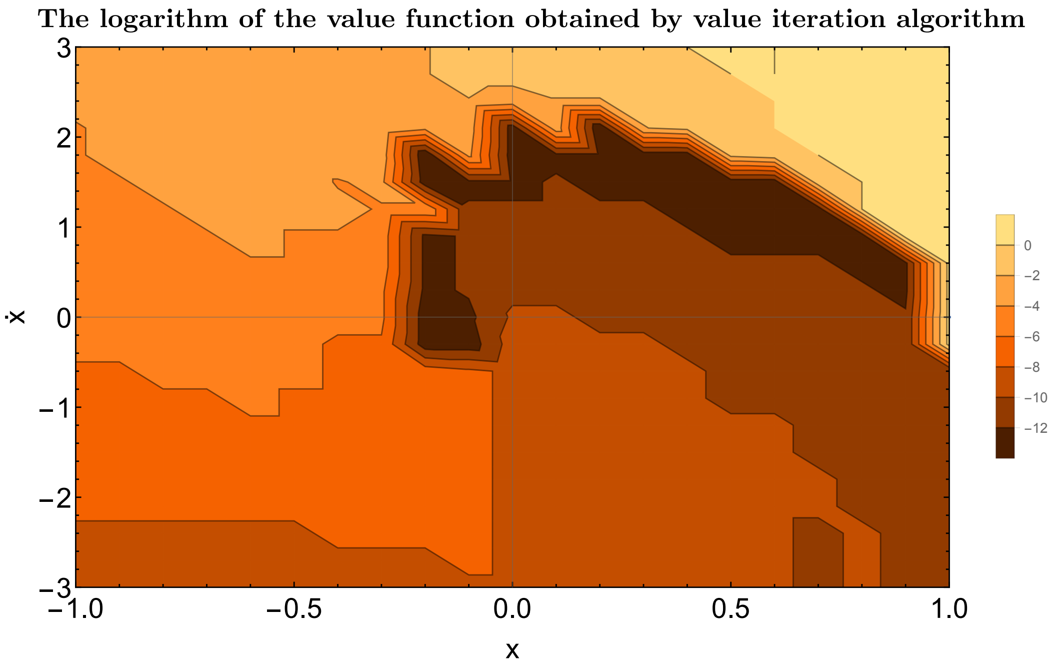


Figure 2: Contour plot of $\log V_{VI}$

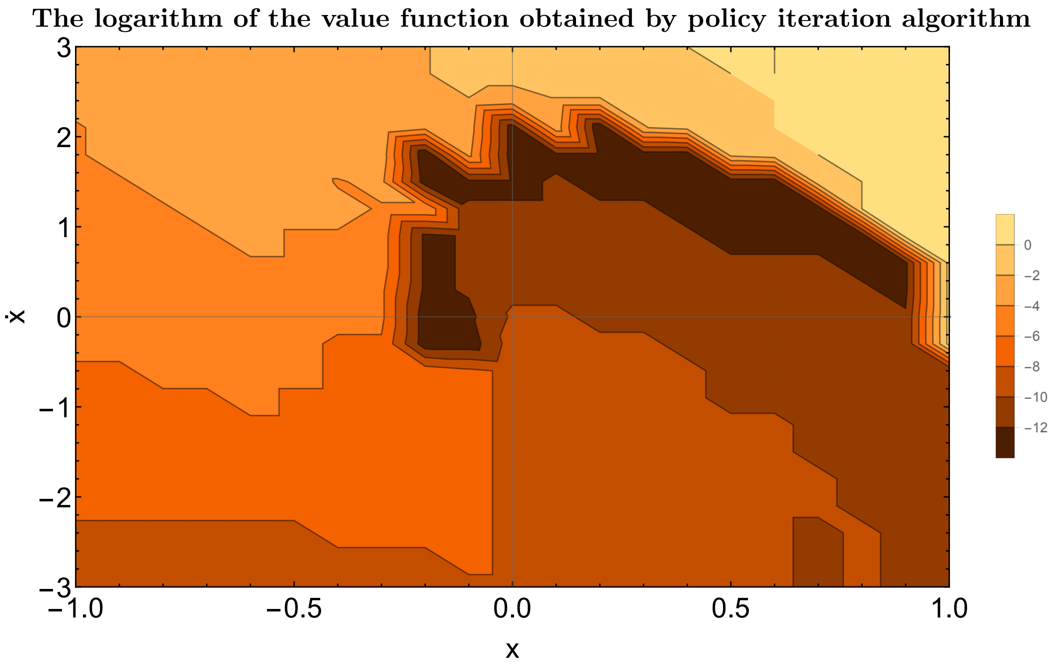


Figure 3: Contour plot of $\log V_{PI}$

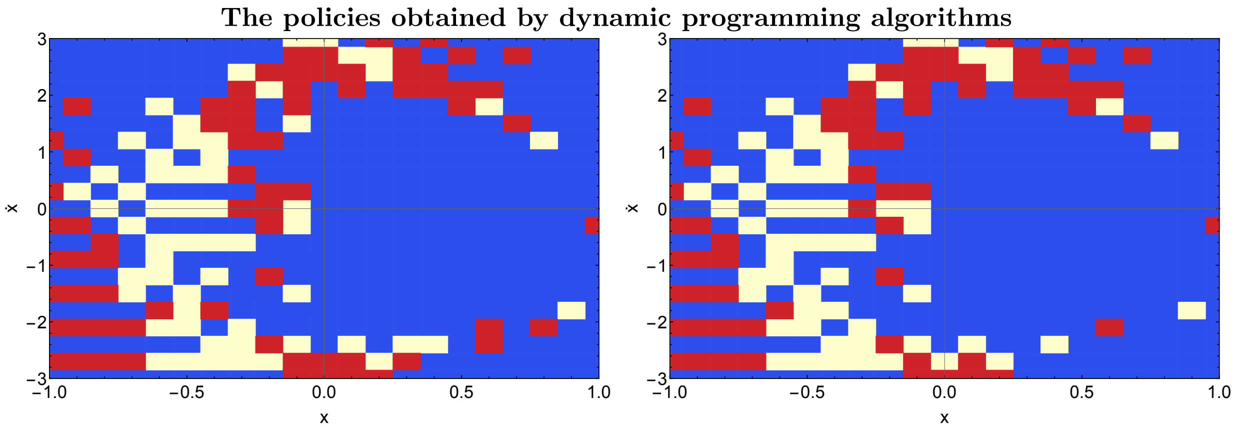
Let π_{VI} and π_{PI} be the deterministic policies we have obtained from value iteration and policy iteration respectively. If we define ρ as

$$\rho := \frac{|\{s \in \mathcal{S} \mid \pi_{VI}(s) = \pi_{PI}(s)\}|}{|\mathcal{S}|}, \tag{37}$$

then we get

$$\rho = \frac{433}{441} \approx 0.98.$$

So the policies are not the same, although they are both very likely to be optimal. The policies can be seen in Figure 4. If we look closely, we see where the difference lie. Using these figures, one can see how the driver should act in each state $(x, \dot{x})$. With both policies one can get to the right mountaintop in 16 movements using the policies obtained.



(a) Value iteration (b) Policy iteration

Figure 4: The policies from dynamic programming. The legend: red="forward", white="stay still" and cyan="backward"

Although it seems that the driver should always drive backwards, that is not actually the case. The reason for that is explained as follows. When taking the argmax, always the first argument is chosen if there is a tie. Since the first argument is driver backwards, driving backwards is chosen if there is a tie.

To actually see what is going on, we had implemented another argmax-function which gives the largest argument, namely driving forwards, if there is a tie. If we compare both implementations we can see where the type of action is actually relevant (see Figure 5). There are on two states where one should stay still, those states are given with arrows.

We see that the action that the driver should start with is driving backwards, since the action at that state is relevant. The policy is actually a little bit logic now. Now we see that if the driver is going forwards with high velocity, then he should keep going. The policy is a little bit chaotic, but that is due to the rounding and all other factors that we have mentioned that play a role.

As we can see both algorithms are performing well. Value iteration seems to outperform policy iteration in this particular case. We also conclude that we may see that the policies are machine-precise optimal. However, the policies are not the same. We have also seen that the policy iteration terminated without having to add additional flags. In the next part we give a small discussion on results obtained on finer discretization of the state space.

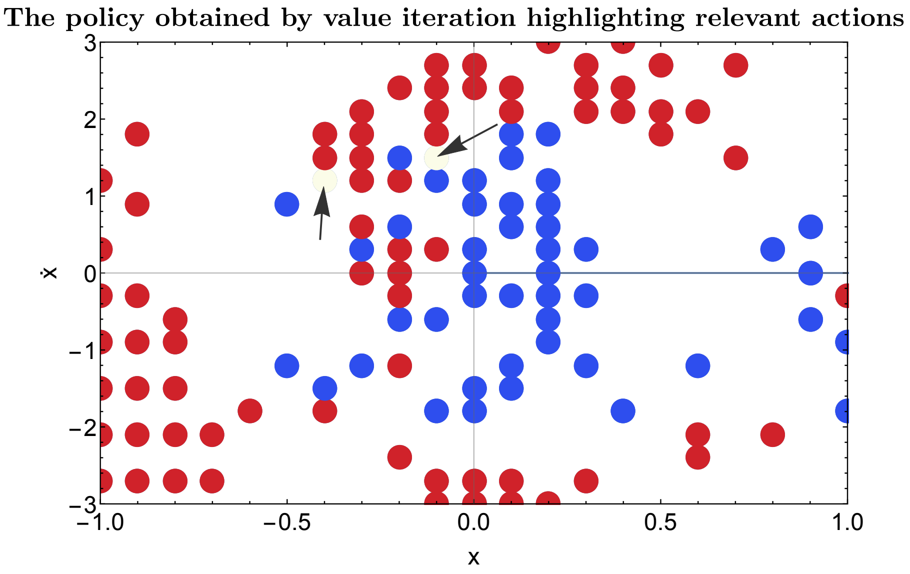


Figure 5: The policy obtained by value iteration highlighting where the type of action is relevant. The legend: red="forward", white="stay still" and cyan="backward"

3.3.1 Finer grid

This subsection is intended to get some understanding about how the car is moving towards its goal. We have taken a finer grid here, namely $N = 300$. We use value iteration to get the approximations. We immediately get a problem with the implementation. Because $\beta = 2.5$ suddenly does not work anymore; the car can not get to the mountaintop. That is actually very unintuitive. The only logical way to explain that is that when $\beta = 2.5$ and $N = 20$ there are large errors due to the rounding. In particular, that causes the car to go up, while it would never be able to go up if the rounding was not so large.

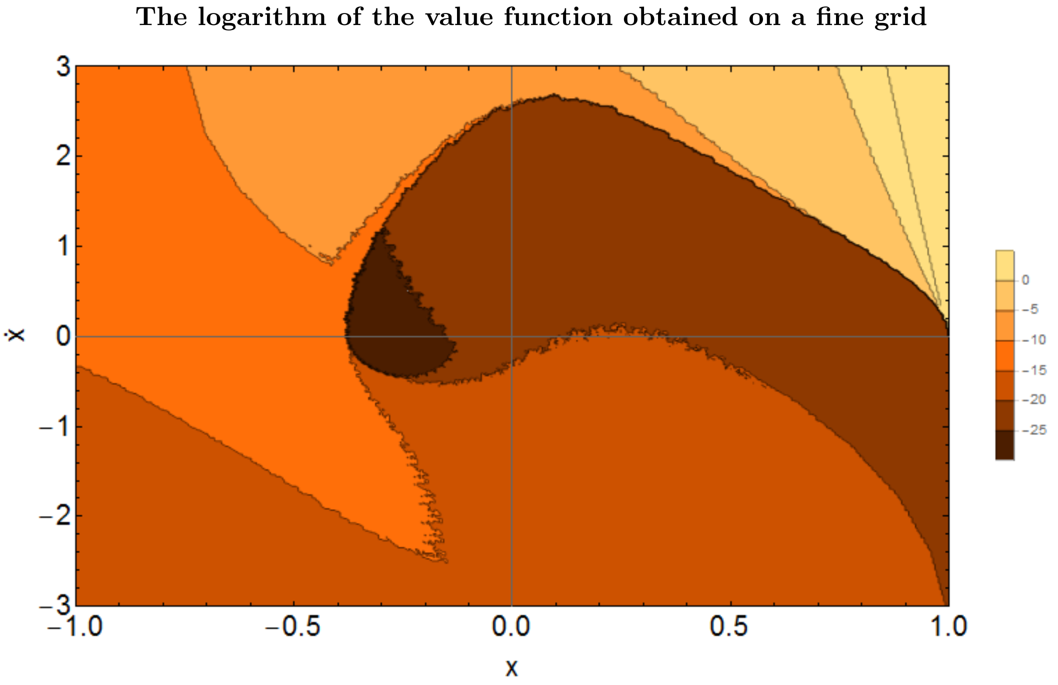


Figure 6: Contour plot of logarithm of the value function on a fine grid

Anyways, we have taken $\beta = 4.5$ since values below 4 did not work as well. The logarithm of the value function can be seen in Figure 6. It is very similar to what we have seen in Figure 2. We have explained that there were probably something going on with the rounding, but it seems that there are more serious problems. When $\beta = 2.5$ did not work well, we should have known that there is something deep happening here which we can not explain.

To obtain the optimal policy, we have again checked if there is a tie between going backwards or going forwards. The actions that were chosen without ties are highlighted. The policy can be seen in Figure 7. Surprisingly, this looks like the one in Figure 5 in some sense. It is not clear why there are ties between states where one should go backwards following the pattern.

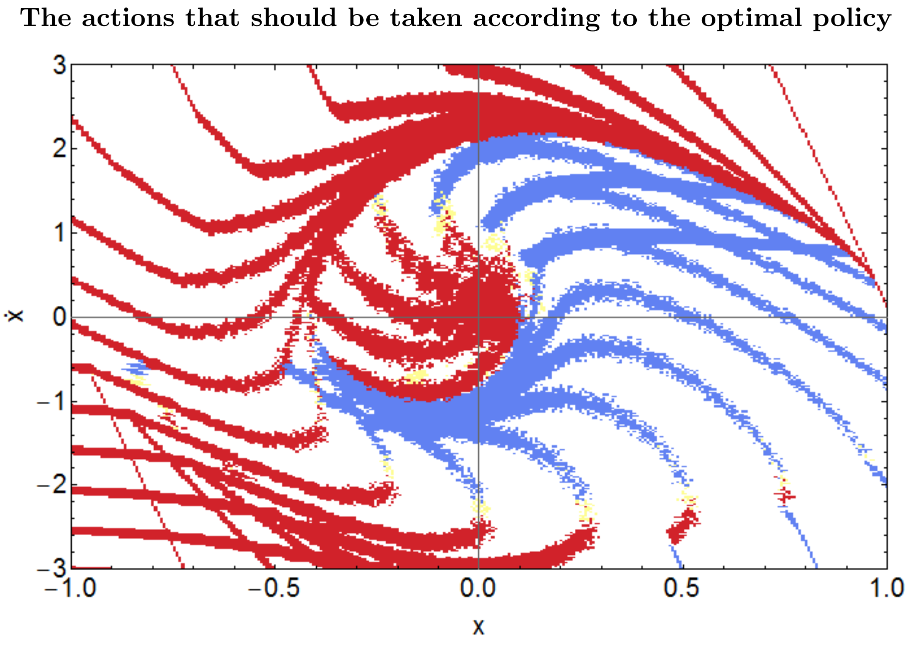


Figure 7: Highlighted policy. The legend: red="forward", yellow/white="stay still" and cyan="backward"

We were tending to keep the discussion whether the policy or the value function is understandable short. We should notice that it is nice to understand what is actually going on, but that is not really what we are using the mountain car problem for. For us it is enough that the car cannot get to the mountaintop by just going forwards for the case $N = 20$, because then we can check whether the algorithms work. This last part of the results is not relevant for the rest of the report. However, it is included to make an attempt for understanding the mountain car problem better.

4 Monte Carlo

In this section we provide another algorithm to solve for the optimal policy. The algorithms that we will consider here are the *Monte Carlo* algorithms. These algorithms might be more advantageous than the previous algorithms, since we do not need to assume the complete knowledge of the environment. In many practical situations, one does not know the environment dynamics.

However this algorithm's convergence properties are different compared to the dynamic programming algorithms, because its convergence is an open problem [11]. Therefore this section has a heuristic flavor rather than a formal flavor.

First we give a little motivation of the algorithm. We first show how to approximate $q_\pi(s, a)$ using the Monte Carlo algorithm. Such algorithm is shown to converge in [7] under the condition that we have a *finite horizon* decision task with no discounting, i.e. $\gamma = 1$. A finite horizon task means that the interaction terminates after some fixed time [3]. In particular, that means that the return is a finite sum of rewards which makes the analysis a bit easier. Since we do not have $\gamma = 1$ nor a finite horizon task, that result does not apply for our setting.

However, the same idea from [7] can be used to show why a similar result might hold. The idea is to approximate $q_\pi(s, a)$ as follows. Let us first construct n independent Markov chains $\{(S_{k,i}, A_{k,i})_k\}_{i=1}^n$ with initial distribution ν such that $\nu(s, a) = 1$ and policy π. With each of these Markov chains one gets a discounted return G_1 which is defined in Equation (3). So we get a sequence of i.i.d. random variables $G_{1,1}, ..., G_{1,n}$ of discounted returns. Now by the strong law of large numbers one gets

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} G_{1,i} = \mathbb{E}[G_{1,1}] \quad \text{a.s.} \tag{38}$$

For an event B occurring with probability 1 one has $\mathbb{E}[\,\cdot \mid B] = \mathbb{E}[\,\cdot\,]$. That means that $\mathbb{E}[G_{1,1}] = \mathbb{E}[G_{1,1} \mid S_0 = s, A_0 = a] = q_\pi(s, a)$. Therefore we conclude

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} G_{1,i} = q_\pi(s, a) \quad \text{a.s.} \tag{39}$$

In a simulation, we can get realizations of $r(S_{k,i}, A_{k,i})$, but we can only get a finite number of them. That means that we must estimate $q_\pi(s, a)$ through the following quantity:

$$\hat{G}_i = \sum_{k=0}^{N} \gamma^k r(S_{k+1,i}),$$

with $N \in \mathbb{N}$. These random variables can get quite close to $G_{1,i}$, namely we have the following upper bound for their difference:

$$|\hat{G}_i - G_{1,i}| \le \sum_{k=N+1}^{\infty} \gamma^k |r(S_{k+1,i})| \le \sup_{s'} |r(s')| \frac{\gamma^{N+1}}{1-\gamma}, \tag{40}$$

which can be made arbitrarily small by taking N large enough.

We simulate n Markov chains ending at time step N and consider the quantity

$$\frac{1}{n} \sum_{i=1}^{n} \hat{G}_i$$

as an estimate for $q_\pi(s, a)$. By the Strong Law of Large Numbers, we get the following:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \hat{G}_i = \hat{q}_\pi(s, a) \quad \text{a.s.,}$$

for some function $\hat{q}_\pi \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$. Then we have

$$|q_\pi(s,a) - \hat{q}_\pi(s,a)| = \lim_{n\to\infty}\left|\frac{1}{n}\sum_{i=1}^n G_{0,i} - \frac{1}{n}\sum_{i=1}^n \hat{G}_i\right| \le \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^n\left|\hat{G}_i - G_{0,i}\right| \le \sup_{s'}|r(s')|\frac{\gamma^{N+1}}{1-\gamma}.$$

This says roughly that if we simulate a large number of Markov chains ending at time step N, then we can get a fairly good estimate for $q_\pi(s,a)$. This is the underlying idea of Monte Carlo algorithms. One sees that we must repeat the whole simulation for each $(s,a) \in \mathcal{S}\times\mathcal{A}$. Here is where Monte Carlo extends the idea described above by splitting the random variable $\hat{G}_i$ in several pieces. We omit more details on that.

The previous paragraphs made a heuristic argument how one could estimate $q_\pi(s,a)$. However, it did not give arguments for finding the optimal policy. The idea is choosing π greedily with respect to the current estimate of q_π. More on this will be seen later.

We consider two algorithms. The first is called *Monte Carlo exploring-starts (ES)* and the other one is called *Monte Carlo first-visit*. Both rely on different assumptions which will be explained in the next subsections. However, both have the same goal, namely to find an approximation for q_*. As we know, from q_* we can deduce an optimal policy π_*.

4.1 Monte Carlo exploring-starts

For this algorithm one assumes that the MDP might start anywhere. In particular, one assumes that the initial distribution μ is uniform on $\mathcal{S}$. That explains the name *exploring starts*.

That assumption is actually not satisfied in the mountain car problem, but one could see this algorithm as the same problem where one is teaching the driver how to drive from an arbitrary point to the right-end point. After all, it is an algorithm for which we want to investigate its performance through a simple example.

The algorithm will be described in a pseudocode as given in [3] and then a motivation of the algorithm will be given.

```
Initialization:
$\pi(s) \in \mathcal{A}$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

Monte Carlo ES:
Loop for each episode:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}$ such that all pairs have positive probability
    Generate an episode from $S_0, A_0$, following $\pi : S_0, A_0, r(S_1), ..., S_{T-1}, A_{T-1}, r(S_T)$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, ..., 0$:
        $G \leftarrow \gamma G + r(S_{t+1})$
        if the pair $S_t, A_t$ does not appear in $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$ then
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)
            $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$
        end
    EndLoop
EndLoop
```
Algorithm 3: Monte Carlo ES

A simple heuristic for why this algorithm might be reasonable relies on a theorem called *policy improvement theorem* which is stated below [3].

Theorem 25. Let $\pi, \pi' \in \Pi_D$. If for all $s \in \mathcal{S}$ one has

$$q_\pi(s, \pi'(s)) \geq v_\pi(s), \tag{41}$$

then the policy π' is as good as, or better than, the policy π, i.e. $v_\pi \leq v_{\pi'}$.

Proof. The proof can be obtained by iteratively applying the third part of Lemma 9. The full proof can be found in [3]. $\square$

Now we show that taking the new policy greedily with respect to the action-value function will eventually lead to convergence to the optimal policy, if the action-value function is calculated exactly. Unfortunately that is not the case in the algorithm.

Assume that in each step of the algorithm one finds q_{π_k} exactly, where π_{k+1} is found as follows:

$$\pi_{k+1}(s) = \operatorname*{argmax}_{a \in \mathcal{A}} q_{\pi_k}(s, a). \tag{42}$$

Therefore we have

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \operatorname*{argmax}_{a \in \mathcal{A}} q_{\pi_k}(s, a)) \\ &= \max_{a \in \mathcal{A}} q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)). \end{aligned}$$

Notice that by part (iv) of Lemma 9, we get

$$v_{\pi_k}(s) = q_{\pi_k}(s, \pi_k(s)). \tag{43}$$

Together with the previous observation, this implies

$$q_{\pi_k}(s, \pi_{k+1}(s)) \geq v_{\pi_k}(s). \tag{44}$$

Applying the previous theorem to conclude that π_{k+1} is better or at least as good as π_k. Now we have a sequence of policies such that $v_{\pi_k} \leq v_{\pi_{k+1}}$. If $v_{\pi_k} = v_{\pi_{k+1}}$ for some k, then that means, using part of (iii) of Lemma 9, that

$$\begin{aligned} v_{\pi_{k+1}}(s) &= \max_{a \in \mathcal{A}} q_{\pi_k}(s, a) \\ &= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)[r(s') + \gamma v_{\pi_k}(s)] \\ &= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)[r(s') + \gamma v_{\pi_{k+1}}(s). \end{aligned}$$

Hence $v_{\pi_{k+1}}$ satisfies the Bellman equations, which implies that $v_{\pi_{k+1}} = v_*$ and that π_{k+1} is optimal. If $v_{\pi_k} \neq v_{\pi_{k+1}}$, then there is $s \in \mathcal{S}$ such that $v_{\pi_k}(s) < v_{\pi_{k+1}}(s)$. There are a finite number of deterministic policies, so at some point we get the optimal policy.

Surely, this says not much about the convergence about the actual algorithm. However it shows that one might expect that this algorithm is somehow reasonable. First heuristics of the estimation of the action value were given. After that we have seen that the updating rule for the policy does actually make sense in the case that every calculation is done exactly.

4.2 Monte Carlo first-visit

Here, one has the assumption that there is an arbitrary initial distribution for the MDP. That is what is going on in the mountain car problem, the car starts at position $x = 0$ with velocity $\cdot x = 0$. That gives the initial distribution $\mu(0, 0) = 1$ for example. However that immediately causes a problem, since that might cause the Markov chain $(S_k, A_k)_{k\geq 0}$ to never visit some pairs (s, a). For these pairs (s, a) one can not find an estimation of $q_\pi(s, a)$.

If we find a *soft policy*, i.e. a policy which has nonzero probability for every action, then there is a possibility for more exploration. That means that more pairs (s, a) in $\mathcal{S} \times \mathcal{A}$ can be encountered. That leads to the use of the ε-greedy policies in the following algorithm, that is a policy which satisfies $\pi(a|s) = \varepsilon/|\mathcal{A}|$ for all actions a, but the one that maximizes the approximate action-value function. This algorithm is also from [3] and it is given below.

```
**Initialization:**
Algorithm parameter small $\varepsilon > 0$
$\pi \leftarrow$ an arbitrary $\varepsilon-$soft policy
$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

**Monte Carlo first-visit:**
**Loop** *for each episode:*
    Choose $S_0 \in \mathcal{S}$ as the initial state and $A_0 \in \mathcal{A}$ following policy $\pi$
    Generate an episode following $\pi : S_0, A_0, r(S_1), ..., S_{T-1}, A_{T-1}, r(S_T)$
    $G \leftarrow 0$
    **Loop** *for each step of episode,* $t = T - 1, T - 2, ..., 0$:
        $G \leftarrow \gamma G + r(S_{t+1})$
        **if** *the pair* $S_t, A_t$ *does not appear in* $S_0, A_0, S_1, A_1, ..., S_{t-1}, A_{t-1}$ **then**
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \text{argmax}_a Q(S_t, a)$
            $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = A^* \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a \neq A^* \end{cases}$
        **end**
    **EndLoop**
**EndLoop**
```

Algorithm 4: Monte Carlo first-visit

In the same way one can show that this algorithm gives a policy which is optimal among all ε-greedy policies, if one calculates the action-value function in each step exactly [3]. However, that does not mean that it converges to an optimal policy.

The Monte Carlo algorithms that we have considered have many issues in practice. For instance, we don't know for how many episodes we should run the algorithm. Since its convergence property is still an open problem, it is difficult to know how the algorithm behaves with respect to the number of episodes.

4.3 Results

It is important to notice that every time we run the algorithm, we might get different results. Indeed, because we are in fact simulating a Markov chain each time which is random. Due to time limitations, we have chosen to run both algorithms 50 times each. In each simulation we have taken 10000 episodes which last 55 time steps. Furthermore the initialization used can be seen below (see Table 2).

Table 2: Initialization for Monte Carlo

Quantity	**Exploring starts**	**First visit**
ε	NA	0.3
Q_0	zero-function	zero-function
π_0	always-go-forward	uniform measure on $\mathcal{A}$

Before giving the results, we give a remark about an issue.

Remark 26. Consider an MDP with state space $\mathcal{S}$ and an initial distribution μ. Take a soft policy π. It might be the case that the induced Markov chain $(S_k, A_k)_{k\geq 0}$ never takes particular values in $\mathcal{S} \times \mathcal{A}$. However, there exists another state space $\mathcal{S}'$ such that $(S_k, A_k)_{k\geq 0}$ takes all values in $\mathcal{S}' \times \mathcal{A}$.

The remark only says that theoretically we might get such $\mathcal{S}'$. However, in practice, it is hard to find such $\mathcal{S}'$. In this particular example, we have constructed the transitions using the results from physics. To find which transitions are possible leads to a problem which is not in the scope of this report.

We can at least know from Monte Carlo first-visit which states belong to $\mathcal{S}'$. Since Monte Carlo exploring-starts can get to any pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, it is not fair to compare the values of the approximate action-value function $Q(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. For that reason and since we run the simulation for 50 times, which is not even so many to make statistical statements, we have decided to make only statements about the states $(s, a) \in \mathcal{S} \times \mathcal{A}$ which are visited in all of the simulations.

4.3.1 Notational conventions

We define notations for the concepts we were just talking about. For algorithm X, define the following set:

$$\mathcal{S}^X := \{s \in \mathcal{S} \mid s \text{ is visited in algorithm } X \text{ in all simulations}\}. \tag{45}$$

Also we use the following notation for norm:

$$\|f - g\|_{(\mathcal{S}^X,\infty)} := \sup_{x\in\mathcal{S}^X} |f(x) - g(x)|. \tag{46}$$

If we write the norm without the subscript $\mathcal{S}^X$ then we mean the norm over the whole set $\mathcal{S}$, and usually it is clear from the context.

We have mentioned in Section 3 that we use value iteration as a baseline. Therefore when we are talking about error, we usually mean the error compared to the value function from value iteration, namely V_{VI}. For algorithm X, we define the absolute error as the function $|V_X(s) - V_{VI}(s)|$ having domain $\mathcal{S}^X$. The maximum absolute error is then

$$\|V_X - V_{VI}\|_{(\mathcal{S}^X,\infty)}.$$

Moreover the relative error is defined as the mapping on $\mathcal{S}^X$

$$s \mapsto \left|\frac{V_X(s) - V_{VI}(s)}{V_{VI}(s)}\right|. \tag{47}$$

Fortunately $V_{VI}(s) \neq 0$ for all $s \in \mathcal{S}$, which means that the relative error is well-defined. The maximum relative error is then the maximum over $\mathcal{S}^X$.

In each simulation we get a policy. We make the ε-greedy policies deterministic such that it gives probability 1 for the action it gave the highest probability to. That makes sense, because if an ε-greedy policy gives a high probability for some action then that means that action maximizes the approximate action-value function. But then we could perform *that* action all the time as well. We use the notation $\pi_X^{(j)}$ for the policy that we have obtained with algorithm X at the j-th simulation. In our case we have $j \in \{1, 2, ...50\}$. Moreover we can get an approximation for the value function of the policy $\pi_X^{(j)}$. The method that we have used is described in Remark 15 and it gives us a very accurate value function approximation if we choose $\theta = 10^{-16}$. We denote the value functions we obtain by $V_X^{(j)}$.

We use $X = ES$ for Monte Carlo exploring-starts and $X = FV$ for Monte Carlo first-visit. The cardinality for the states visited are given in Table 3. Fortunately, we have enough points to compare the approximate value functions with each other.

Table 3: The size of the states visited in all Monte Carlo simulations

Quantity	**Cardinality**	**Proportion of total**
$\mathcal{S}^{ES}$	430	97.5%
$\mathcal{S}^{FV}$	290	65.8%
$\mathcal{S}^{ES} \cap \mathcal{S}^{FV}$	287	65.1%

4.3.2 Value function approximation

Recall that we have an approximation for v_* by the value iteration algorithm, while we have an approximation for q_* here. We want to compare the results, so it seems convenient to use the formula $\max_{a\in\mathcal{A}} q_*(s,a) = v_*(s)$ to translate the action-value function into a value function. That means that the algorithms give an approximation of v_* as well. Let us denote V_{ES} and V_{FV} for the value function we obtain by Monte Carlo exploring-starts and Monte Carlo first-visit respectively. A contour plot of the logarithm value function can be seen in Figure 8 and Figure 9. Notice, when comparing both results, that the legends are a bit different.

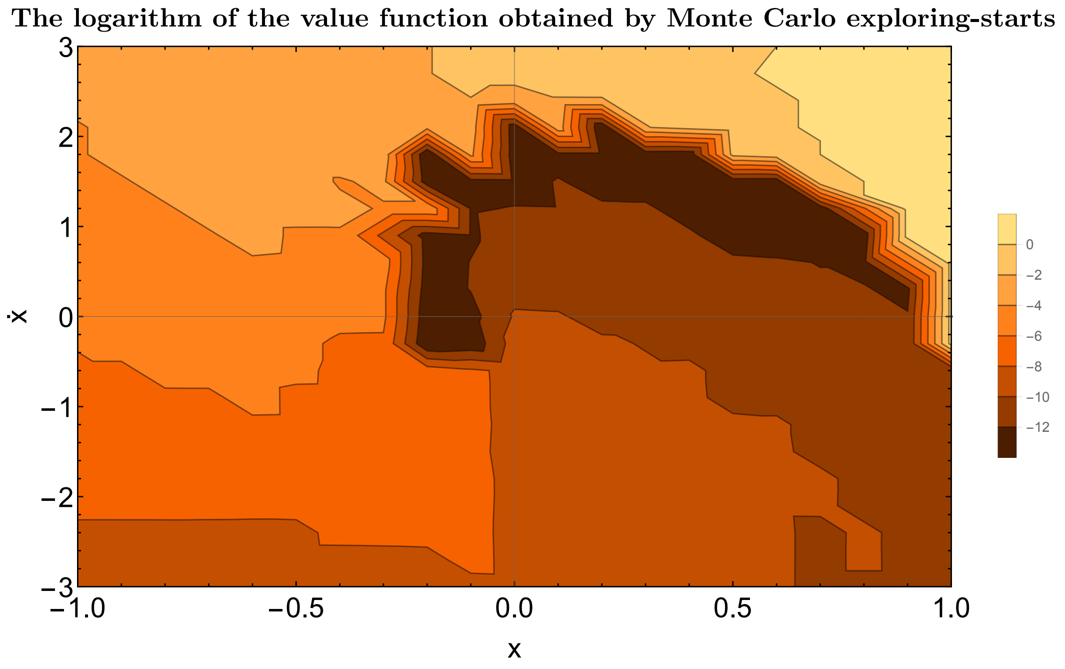


Figure 8: Contour plot of $\log V_{ES}$

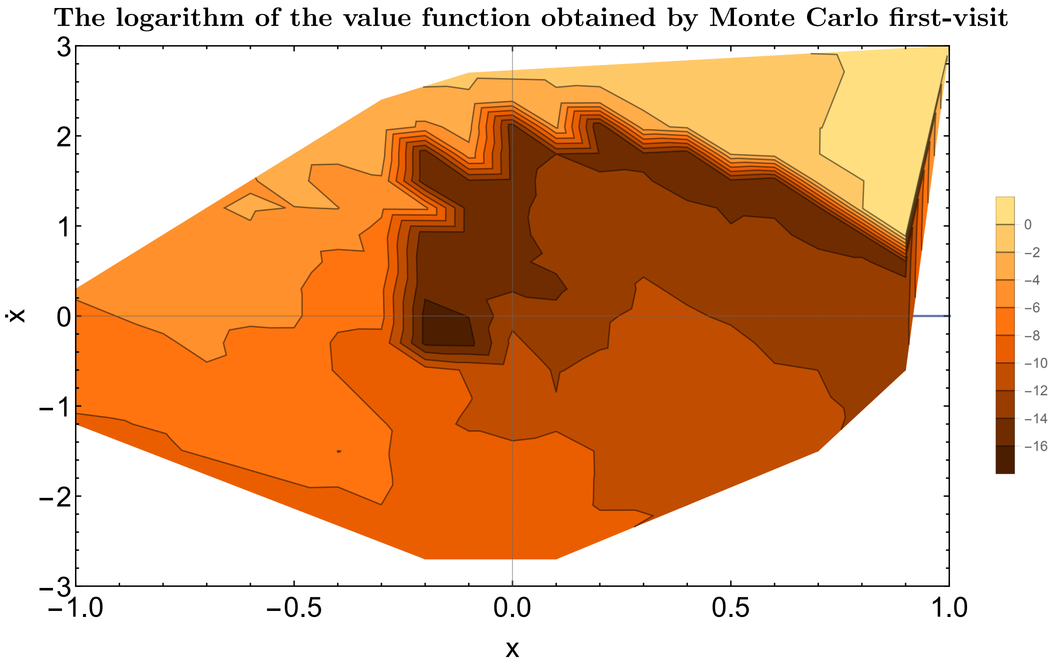


Figure 9: Contour plot of $\log V_{FV}$

The white color in the corners is due to the fact that not all states are visited. The only corner that is not white is the north-east corner. The reason for the last observation is that the algorithm finds a way to get to the goal, namely the points $(1, \eta) \in \mathcal{S}$ with η is positive meaning the car is driving fast to the right direction. Probably, in each simulation the driver finds out that it has to go backward and slide with high speed to get to the right-end point of the parabola (i.e. the mountain).

It is not easy to see the difference of the approximate value functions obtained here and the value function V_{VI} by looking at the plots. We can see the errors with respect to the baseline value function V_{VI} in Table 4. It is clear from Table 4 that Monte Carlo exploring-starts is performing much better Monte Carlo first-visit.

It seems that both algorithms give quite accurate approximations based on the absolute errors. However, the relative errors are rather large. Since the discount factor $\gamma = \frac{1}{2}$, we know that the value function is between 0 and 2. Furthermore, the value function is really close to zero for most of the arguments since the the return decays exponentially. Therefore small absolute errors yield large relative errors.

Table 4: Absolute and relative error of Monte Carlo algorithms

	Absolute error		**Relative error**	
Algorithm	Mean	Maximum	Mean	Maximum
Exploring starts	$1.49 \cdot 10^{-4}$	0.0260	0.0361	0.501
First visit	0.00595	0.0770	0.654	0.984

4.3.3 Optimality of policy

We check whether a policy from the algorithm is optimal. Therefore we consider how far $V_{ES}^{(j)}$ or $V_{FV}^{(j)}$ are from V_{VI}. If the maximum absolute difference is small enough, i.e. as small as machine precision, we consider the policy that belongs to the approximate value function optimal. We get the following bounds:

$$2.78 \cdot 10^{-17} \leq \|V_{ES}^{(j)} - V_{VI}\|_{(\mathcal{S}^{ES}, \infty)} \leq 0.0313, \tag{48}$$

and

$$1.22 \cdot 10^{-4} \leq \|V_{FV}^{(j)} - V_{VI}\|_{(\mathcal{S}^{FV}, \infty)} \leq 0.250. \tag{49}$$

Both upper and lower bound are attained. The mean absolute error is 0.00181 and 0.0174. It is clear that none of the policies of Monte Carlo first-visit might be considered as optimal.

If we consider the error in the whole domain $\mathcal{S}$, then we see that the maximum absolute error is less than 10^{-16} for 17 policies obtained from Monte Carlo exploring-starts. We consider these policies optimal. However, the maximum absolute error in the whole domain for Monte Carlo first-visit is equal to 2.00 for all policies. That is remarkable, but the reason is the following. If one starts at a state which leads to the right-end point of the parabola after one step, then the return in such case is

$$\sum_{i=0}^{\infty} \gamma^i \cdot 1 = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = 2.$$

But we knew that Monte Carlo first-visit did not always succeed to get to all states of the form $(1, \eta) \in \mathcal{S}$. The policies do not give the right choice of action which leads to missing the return which would be equal to 2. That means that none of the policies from Monte Carlo first-visit are optimal.

Table 5: The number of steps needed to get to the goal using the policies from Monte Carlo algorithms

	Number of steps needed to get to the right-end point		
Algorithm	Minimum (frequency)	Mean	Maximum (frequency)
Exploring starts	16 (48)	16.14	22 (1)
First visit	16 (40)	17.12	23 (1)

Although it seems that the policies from Monte Carlo first-visit are bad, they perform well giving the mountain car starts still at position $x = 0$. We can see from Table 5 that on average the car needs 16.14 steps and 17.12 using the policies from Monte Carlo exploring-starts and Monte Carlo first-visit respectively.

4.3.4 Convergence speed

We have run a special simulation to determine the convergence speed with respect to the number of episodes. In that simulation only a few values are stored to speed up the running time. We have run 50 simulations with 40000 episodes. We again considered the errors compared with V_{VI} in 2-norm over the set $\mathcal{S}^{ES} \cap \mathcal{S}^{FV}$. We see that the Monte Carlo exploring-starts converges much faster than Monte Carlo first-visit from Figure 10. It is not clear whether the error in Monte Carlo first-visit gets close to zero, since it seems to get very flat after the 30000-th episode.

However, that is not strange after all. We know from the discussion in Section 4.2 that the action-value function converges to an action-value function optimal among all ε-greedy action-value functions if the action-value function is calculated exactly in each Monte Carlo step. Even if that is true, since we have an approximation of the action-value function in each step, we can not expect the final approximation to be close to the optimal action-value function. There is no guarantee that there exists an optimal ε-greedy policy. What the figure below shows might be an illustration of what we have just mentioned.

The results that we have seen suggest that Monte Carlo exploring-starts is much better than Monte Carlo first-visit. That is indeed what one would expect. The main reason for that is the fact that one visits almost all pairs (s, a) in Monte Carlo exploring-starts. In both algorithms the driver seems to learn how to get to the mountaintop which was the actual goal. However, the driver does not always succeed to do it optimally, namely in 16 movements.

Also it is important to keep Remark 26 in mind, namely the possibility that the Markov chain $(S_k, A_k)_{k\geq 0}$ being not able to get to every pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ starting from a particular position. That is especially applicable for Monte Carlo first-visit. Comparing that algorithm with Monte Carlo exploring-starts or with value iteration might not be fair. That is also confirmed in the results that we have seen. Monte Carlo first-visit performs pretty well if we only consider the errors for the states that it has visited, namely the errors on $\mathcal{S}^{FV}$.

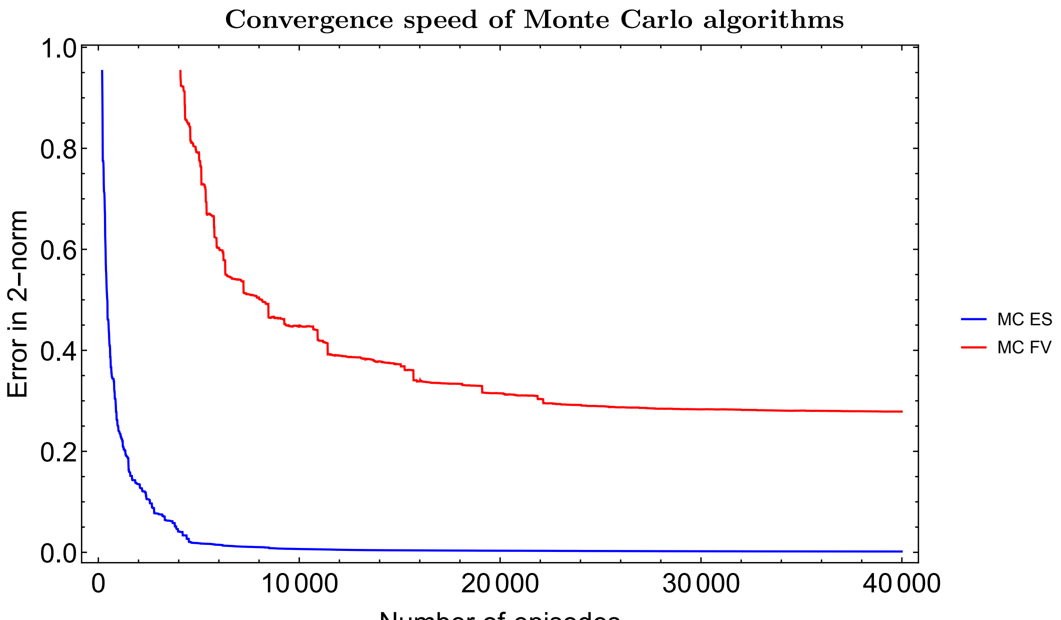


Figure 10: Convergence speed of Monte Carlo algorithms with respect to the number of episodes

5 Temporal-Difference Learning

Temporal-difference learning stands for a whole class of algorithms. However, we only discuss two of them here. The names might differ in literature, but we call them *Q-learning* and *Q-learning with learning policies*. As Monte Carlo, there is no need to know the dynamics of the MDP. The algorithms are intended for finding q_* which explains the name Q-learning.

5.1 Q-learning

The Q-learning algorithm finds q_* using an arbitrary soft. For the algorithm, one needs to have episodes, which might end after a fixed time of steps or by observing a particular state. The algorithm, as taken from [5], is shown below.

```
**Initialization:**
$Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ arbitrarily
$\pi(s|a) > 0$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ arbitrarily
Step size $\alpha \in (0,1)$

**Q-learning:**
**Loop** *for each episode:*
    Initialize $S$
    **Loop** *for each step of episode until $S$ is terminal*
        Choose $A$ from $S$ using $\pi$
        Take action $A$, observe $r(S')$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[r(S') + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    **EndLoop**
**EndLoop**
```

Algorithm 5: Q-learning

It is clear from the algorithm that we are simulating a Markov chain $(S_k, A_k)_{k\geq 0}$. That Markov chain induces a sequence of estimates of some action-value function which appears to be an approximation for q_*. Let us call that sequence $(Q_k)_{k\geq 0}$ where Q_0 is the arbitrary initialization. The index k increases at the second loop in the algorithm (independently of which episode). The sequence $(Q_k)_{k\geq 0}$ is a sequence of random variables, more precisely Q_k are $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$-valued random variables. The update rule for $Q_k(s,a)$ is as follows:

$$Q_{k+1}(s,a) = \begin{cases} (1-\alpha)Q_k(s,a) + \alpha\left[r(S_{k+1}) + \gamma \max_{a'} Q_k(S_{k+1},a')\right] & \text{if } (s,a) = (S_k, A_k), \\ Q_k(s,a) & \text{if } (s,a) \neq (S_k, A_k). \end{cases} \tag{50}$$

The updating rule for Q_k can be written more "compactly". To that end we first define α_k as a $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$-valued random variable as follows:

$$\alpha_k(s,a) := \alpha \mathbf{1}_{(s,a)=(S_k,A_k)}. \tag{51}$$

and then one can write

$$Q_{k+1}(s,a) = (1-\alpha_k(s,a))Q_k(s,a) + \alpha_k(s,a)\left[r(S_{k+1}) + \gamma \max_{a'} Q_k(S_{k+1},a')\right]. \tag{52}$$

Let us abuse the notation for q_* such that $q_*(s,a)$ is a degenerate random variable living on the same space as Q_k which satisfies $T[q_*] = q_*$ a.s.. Then one can define a sequence which measures how far Q_k is from q_*, namely ε_k which is also a $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$-valued random variable and is defined as follows:

$$\varepsilon_k(s,a) := Q_k(s,a) - q_*(s,a). \tag{53}$$

Notice that we have many stochastic processes which are all completely determined by the stochastic process $(S_k, A_k)_{k\geq 0}$.

This algorithm is shown in several ways to be reasonable for estimating q_*. For instance in [5], it is shown that for all $\delta > 0$ there exists $\alpha > 0$ and K_δ such that for all $k > K_\delta$ one has $\mathbb{E}[\|\varepsilon_k\|_\infty] < \delta$ under some conditions. This means that the error can be made arbitrary close to zero by choosing α appropriately. However the author assumed that there is some number $L \in \mathbb{N}$ such that in L time steps on average every pair $(s,a) \in \mathcal{S}\times\mathcal{A}$ is visited at least once. It is very difficult to tell if such assumption is satisfied in an MDP. A good thing about the author's approach is that he finds an upper bound for $\mathbb{E}[\|\varepsilon_k\|_\infty]$ in terms of k.

Also in [8] a stronger type of convergence is proved, namely $Q_k \to q_*$ a.s. That is proved under the assumption that α is non-constant, but rather a nonnegative decreasing sequence, α_k (not to be confused with $\alpha_k(s,a)$ as defined above) which satisfies

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty. \tag{54}$$

It is clear that such condition can never be satisfied with constant α. This result is actually very promising, since implementing the algorithm with such sequence α_k is not difficult. An example for such sequence is $(\frac{1}{k})_{k\geq 1}$.

In this report, we prove a stronger notation of convergence, in particular $Q_k \to q_*$ a.s. with constant α, in return for additional assumptions on the MDP. These assumptions might be rather restrictive, but they make the proof easier and they are satisfied for a big class of problems. Even if some of them are not satisfied, then sometimes there are ways to go around them (see for instance Remark 26 to go around the last assumption).

Assumptions 1.

- The probability measure $P(\cdot|s,a)$ is degenerate, i.e. there is some $y(s,a) \in \mathcal{S}$ such that $P(y(s,a)|s,a) = 1$.
- The policy π (which does not depend on k) has the property $\pi(a|s) > 0$ for all $a \in \mathcal{A}$.
- For all $s, s' \in \mathcal{S}$ there exists $i > 1$, $s_1, ..., s_i \in \mathcal{S}$ and $a_1, ..., a_{i-1} \in \mathcal{A}$ such that:
  - $s_1 = s$, $s_i = s'$,
  - $\prod_{j=2}^{i} P(s_j \mid s_{j-1}, a_{j-1}) > 0$.

Proposition 27. With the last two assumptions in assumptions 1, the Markov Chain $(S_k, A_k)_{k\geq 0}$ is irreducible.

Proof. In order to prove that, let $(s,a), (s',a') \in \mathcal{S}\times\mathcal{A}$ be given. There exists $i > 1$, $s_1, ..., s_i \in \mathcal{S}$ and $a_1, ..., a_{i-1} \in \mathcal{A}$ such that: $s_1 = s$, $s_i = s'$ and $\prod_{j=2}^{i} P(s_j \mid s_{j-1}, a_{j-1}) > 0$. Let a_i be just an arbitrary action, then

$$\prod_{j=2}^{i} p(s_j, a_j \mid s_{j-1}, a_{j-1}) = \prod_{j=2}^{i} P(s_j \mid s_{j-1}, a_{j-1})\pi(a_j \mid s_j) = \left(\prod_{j=2}^{i} P(s_j \mid s_{j-1}, a_{j-1})\right)\left(\prod_{j=2}^{i} \pi(a_j \mid s_j)\right) > 0,$$

since we have a product of positive terms. □

Recall the operator T defined in Equation (11),

$$T[q](s,a) = \mathbb{E}[r(S_1) + \gamma \max_{a'} q_(S_1, a') \mid S_0 = s, A_0 = a]. \tag{55}$$

By Assumptions 1 the Bellman equations (Theorem 12) boil down to

$$q_*(s,a) = r(y(s,a)) + \gamma \max_{a'} q_*(y(s,a),a') \qquad \text{for all } (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{56}$$

In that case the operator T becomes:

$$T[q](s,a) = r(y(s,a)) + \gamma \max_{a'} q(y(s,a),a'), \tag{57}$$

and, most importantly, the update rule for Q_k becomes:

$$Q_{k+1}(s,a) = (1 - \alpha_k(s,a))Q_k(s,a) + \alpha_k(s,a)T[Q_k](s,a). \tag{58}$$

This is the reason why the proof of the convergence that we will give actually works out perfectly. We introduce some important results before getting to the main proof.

Lemma 28. There exists a real number M such that $\|\varepsilon_k\|_\infty \leq M$ a.s..

Proof. The lemma can be proved by proving a similar bound for $\|Q_k\|_\infty$ by induction on k. Boundedness of $\|Q_k\|_\infty$ implies the boundedness of $\|\varepsilon_k\|_\infty$, since $\|\varepsilon_k\|_\infty \leq \|Q_k\|_\infty + \|q_*\|_\infty$. The complete proof is left out since it is not very illuminating, but the full details can be found in [12]. □

Unfortunately, this lemma does not say about how small M is. However, the fact that the error is bounded is useful as we will see later.

Now we define the following sequence of random variables $(X_k)_k$ recursively as follows:

$$\begin{cases} X_0 &= 0, \\ X_1 &= \inf\{j \in \mathbb{N} \mid \forall_{(s,a)\in\mathcal{S}\times\mathcal{A}} : \exists_{i\leq j} : (S_i, A_i) = (s,a)\}, \\ X_{k+1} &= \inf\{j \in \mathbb{N} \mid \forall_{(s,a)\in\mathcal{S}\times\mathcal{A}} : \exists_{i\leq j} : (S_{i+1+X_k}, A_{i+1+X_k}) = (s,a)\}. \end{cases} \tag{59}$$

The interpretation of X_k is explained as follows: $X_{k+1} - X_k$ is the number of steps that is needed to see each pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ at least once for the $(k+1)$-th time. The X_k's are called the k-th cover time of a Markov chain.

Proposition 29. The random variables X_k defined above satisfy $|\mathcal{S} \times \mathcal{A}| \leq X_k < \infty$ almost surely for all $k \in \mathbb{N}$.

Proof. The first inequality is trivial. For the second inequality, we define the following random variables:

$$Y_{s,a} := \inf\{j \in \mathbb{N} \mid (S_j, A_j) = (s,a)\}. \tag{60}$$

Now notice that

$$\{X_1 > x\} \subset \bigcup_{(s,a)\in\mathcal{S}\times\mathcal{A}} \{Y_{s,a} > x\}.$$

Since the Markov chain (S_k, A_k) is irreducible taking values in a finite space, we know that the Markov chain is positive recurrent [13]. Since $Y_{s,a}$ is the hitting time for (s,a) it means that it is finite a.s. This means

$$\lim_{x\to\infty} \mathbb{P}(Y_{s,a} > x) = \mathbb{P}(Y_{s,a} = \infty) = 0.$$

Therefore

$$\begin{aligned}\mathbb{P}(X_1 = \infty) &= \lim_{x\to\infty} \mathbb{P}(X_1 > x)\\ &\leq \lim_{x\to\infty} \sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \mathbb{P}(Y_{s,a} > x)\\ &= \sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \lim_{x\to\infty} \mathbb{P}(Y_{s,a} > x) = 0.\end{aligned}$$

We have used the fact that $\mathcal{S}\times\mathcal{A}$ is finite to interchange the limit and summation. That means in other words $X_1 < \infty$ a.s.. We proceed by induction, so assume $X_k < \infty$ a.s., then we have

$$\begin{aligned}\mathbb{P}(X_{k+1} > x) &= \sum_{m=0}^{\infty} \mathbb{P}(X_{k+1} > x \mid X_k = m)\mathbb{P}(X_k = m)\\ &= \sum_{m=0}^{\infty} \mathbb{P}(\inf\{j\in\mathbb{N} \mid \forall_{(s,a)\in\mathcal{S}\times\mathcal{A}} : \exists_{i\leq j} : (S_{i+1+m}, A_{i+1+m}) = (s,a)\} > x)\mathbb{P}(X_k = m).\end{aligned}$$

We define some more random variables

$$Y_{s,a}^m := \inf\{j\in\mathbb{N} \mid (S_{j+1+m}, A_{j+1+m}) = (s,a)\}.$$

One has $Y_{s,a}^m < \infty$ a.s., since this is also a hitting time for the shifted Markov chain $(S_{k+m+1}, A_{k+m+1})_{k\geq 0}$. This implies

$$\lim_{x\to\infty} \mathbb{P}(\inf\{j\in\mathbb{N} \mid (S_{j+1+m}, A_{j+1+m}) = (s,a)\} > x) \leq \lim_{x\to\infty} \sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \mathbb{P}(Y_{s,a}^m > x) = 0.$$

Hence using the Dominated Convergence Theorem we get

$$\begin{aligned}\lim_{x\to\infty} \mathbb{P}(X_{k+1} > x) &= \lim_{x\to\infty} \sum_{m=0}^{\infty} \mathbb{P}(\inf\{j\in\mathbb{N} \mid \forall_{(s,a)\in\mathcal{S}\times\mathcal{A}} : \exists_{i\leq j} : (S_{i+1+m}, A_{i+1+m}) = (s,a)\} > x)\mathbb{P}(X_k = m)\\ &= \sum_{m=0}^{\infty} \lim_{x\to\infty} \mathbb{P}(\inf\{j\in\mathbb{N} \mid \forall_{(s,a)\in\mathcal{S}\times\mathcal{A}} : \exists_{i\leq j} : (S_{i+1+m}, A_{i+1+m}) = (s,a)\} > x)\mathbb{P}(X_k = m)\\ &= 0,\end{aligned}$$

proving $\mathbb{P}(X_{k+1} = \infty) = 0$, i.e. $X_{k+1} < \infty$ a.s.. □

Now we have the important tools for proving the important theorem, we may finally introduce it.

Theorem 30. With the Assumptions 1, the error made in Q-learning will vanish a.s., formally $\|\varepsilon_k\|_\infty \to 0$ a.s.. Moreover $\mathbb{E}[\|\varepsilon_k\|_\infty^p] \to 0$ for $p \geq 1$, which means that $\|\varepsilon_k\|_\infty \to 0$ in L^p.

Proof. First we derive a recurrence relation for ε_k:

$$\begin{aligned}\varepsilon_{k+1}(s,a) &= Q_{k+1}(s,a) - q_*(s,a)\\ &= (1-\alpha_k(s,a))Q_k(s,a) + \alpha_k(s,a)T[Q_k](s,a) - q_*(s,a)\\ &= (1-\alpha_k(s,a))Q_k(s,a) + \alpha_k(s,a)T[Q_k](s,a) - (1-\alpha_k(s,a)+\alpha_k(s,a))q_*(s,a)\\ &= (1-\alpha_k(s,a))(Q_k(s,a) - q_*(s,a)) + \alpha_k(s,a)(T[Q_k](s,a) - q_*(s,a))\\ &= (1-\alpha_k(s,a))\varepsilon_k(s,a) + \alpha_k(s,a)(T[Q_k](s,a) - T[q_*](s,a)).\end{aligned}$$

Taking absolute value of each side and using the triangle inequality afterwards, we get

$$\begin{aligned}|\varepsilon_{k+1}(s,a)| &\leq (1-\alpha_k(s,a))|\varepsilon_k(s,a)| + \alpha_k(s,a)|T[Q_k](s,a) - T[q_*](s,a)| \\ &\leq (1-\alpha_k(s,a))\|\varepsilon_k\|_\infty + \alpha_k(s,a)\|TQ_k - Tq_*\|_\infty \\ &\leq (1-\alpha_k(s,a))\|\varepsilon_k\|_\infty + \gamma\alpha_k(s,a)\|Q_k - q_*\|_\infty \\ &= (1-\alpha_k(s,a))\|\varepsilon_k\|_\infty + \gamma\alpha_k(s,a)\|\varepsilon_k\|_\infty \\ &= (1-\alpha_k(s,a)(1-\gamma))\|\varepsilon_k\|_\infty.\end{aligned}$$

This implies the somehow weaker inequality $\|\varepsilon_{k+1}\|_\infty \leq \|\varepsilon_k\|_\infty$. Now $\epsilon_k(S_k, A_k)$ is again a stochastic process, and in particular it satisfies

$$|\varepsilon_{k+1}(S_k, A_k)| \leq c\|\varepsilon_k\|_\infty. \tag{61}$$

where $c := 1 - \alpha(1-\gamma) < 1$, since $\alpha_k(S_k, A_k) = \alpha$ by definition. Now define $\tau_n := \sum_{k=1}^n X_k$, where X_k is defined in Equation (59). Using Proposition 29 we get $n|\mathcal{S} \times \mathcal{A}| \leq \tau_n < \infty$ a.s., since τ_n is a finite sum of almost surely finite random variables. Hence $\tau_n \to \infty$ a.s.. We have by the definition of X_k

$$\|\varepsilon_{\tau_{n+1}}\|_\infty \leq c\|\varepsilon_{\tau_n}\|_\infty \leq ... \leq c^n\|\varepsilon_{\tau_1}\|_\infty \leq c^n M, \tag{62}$$

where the last bound is from Lemma 28. This implies $\|\varepsilon_{\tau_n}\|_\infty \to 0$ almost surely as $n \to \infty$.

Now we want to emphasize the fact that $\|\varepsilon_k\|_\infty$ is a sequence of random variables on $(\Omega, \mathcal{F}, \mathbb{P})$, the same probability space as the Markov chain (S_k, A_k) lives. So for each outcome $\omega \in \Omega$ we write $\|\varepsilon_k(\omega)\|_\infty$ and this is not to be confused with the other arguments that ε_k has as a function on $\mathcal{S} \times \mathcal{A}$. Also notice that the random variables τ_n are random variables on $(\Omega, \mathcal{F}, \mathbb{P})$.

There is a set $B \in \mathcal{F}$ such that $\mathbb{P}(B) = 1$ and for all $\omega \in B$ one has $\|\varepsilon_{k+1}(\omega)\|_\infty \leq \|\varepsilon_k(\omega)\|_\infty$ and $\|\varepsilon_{\tau_n(\omega)}(\omega)\|_\infty \to 0$. There is a set $C \in \mathcal{F}$ such that $\mathbb{P}(C) = 1$ and $\tau_n(\omega) \to \infty$. Notice that $\mathbb{P}(B \cap C) = 1$. Let $\omega \in B \cap C$, then $\|\varepsilon_{\tau_n(\omega)}(\omega)\|_\infty \to 0$. By definition $\tau_n(\omega)$ is strictly increasing. Thus, we can take $k_n := \tau_n(\omega)$ so that $\|\varepsilon_{k_n}(\omega)\|_\infty$ is a subsequence of $\|\varepsilon_k(\omega)\|_\infty$ which converges to zero. But $\|\varepsilon_k(\omega)\|_\infty$ is just a sequence of numbers, which is decreasing and nonnegative, i.e. bounded from below. From analysis, we know that it converges. However we know that there is a subsequence that converges to 0. So the whole sequence $\|\varepsilon_k(\omega)\|_\infty$ must converge to 0 as well. Since this holds for all $\omega \in B \cap C$, we conclude $\|\varepsilon_k\|_\infty \to 0$ almost surely.

Finally $\|\varepsilon_k\|_\infty^p \leq M^p$, so that by the Bounded Convergence Theorem one gets $\mathbb{E}[\|\varepsilon_k\|_\infty^p] \to 0$. □

This theorem assures that the algorithm, if run for infinitely long time, i.e. with infinitely many episodes, we eventually get an exact value for q_*. This means that if we run the algorithm for long enough we get $Q_k \approx q_*$.

It is unfortunate that it is not easy to get a useful bound on the error in terms of k. With our approach, that would mean that we need to find bounds for cover times of a Markov chain which does not appear to be easy for a general Markov chain. The reason for that is that we are fully relying on the random times τ_n. For instance in [14], many kind of useful upper bounds is given, but most of them hold for reversible Markov chains. Such assumption is hardly satisfied in the Markov chain we have got from an MDP.

Moreover the convergence speed is dependent on the choice of the initial policy π. Since the distribution of the Markov chain $(S_k, A_k)_{k\geq 0}$ is dependent on π by construction. That means of course that τ_n is also different for different policies. The good choice for π is of course hard to get, but intuitively the uniform policy on $\mathcal{A}$ might be on the less risky side.

We did not say anything about how we must pick the policy yet. In Q-learning, where we stop at time step k, we can get an approximation of the optimal policy by setting:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q_k(s,a). \tag{63}$$

Since we have $Q_k \approx q_*$ we expect to have $\pi \approx \pi_*$, since $\pi_*(s) = \operatorname{argmax}_a q_*(s,a)$.

5.2 Q-learning with learning policies

The second temporal-difference learning method that we discuss is the same algorithm, but now we update the policy throughout the algorithm. We make the policy greedy with respect to the approximation of q_* at each time step.

It is good to notice that the algorithm that we will consider is called *off-policy TD(0) control* in [3]. However, the pseudocode that they have given is ambiguous, it might be interpreted in several ways. It can be interpreted as Q-learning which is previously discussed or as the one we have given in the box below.

```
Initialization:
$Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ arbitrarily
$\pi(s|a) > 0$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ arbitrarily
Step size $\alpha \in (0,1)$, small $\varepsilon > 0$

Q-learning:
Loop for each episode:
    Initialize $S$
    Loop for each step of episode until $S$ is terminal
        Choose $A$ from $S$ using $\pi$
        Take action $A$, observe $r(S')$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[r(S') + \gamma \max_a Q(S',a) - Q(S,A)]$
        $A^* \leftarrow \text{argmax}_a Q(S,a)$
        $\pi(a|S) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = A^* \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a \neq A^* \end{cases}$
        $S \leftarrow S'$
    EndLoop
EndLoop
```

Algorithm 6: Q-learning with learning policies

The only difference with Q-learning is that the policy gets updated in each step in an episode. The policy is sent towards the optimal policy throughout the algorithm. Intuitively this means that the approximation of q_* converges faster. We omit many details on this algorithm, since it is very technical. It is also hard to find literature about this algorithm.

However, there is a very similar algorithm. It is exactly the same as the previous one, but where it says $\max_a Q(S',a)$ is changed to $Q(S',A')$ where A' is the action taken under the policy π given the agent is in state S'. This algorithm is called *SARSA(0)* and it is also discussed in [3]. The SARSA(0) algorithm is worth mentioning, since it is shown to converge to q_π where π is the optimal policy among all ε-greedy policies [15].

The convergence of SARSA(0) can be improved by putting extra conditions on the parameters of the algorithm. The extra conditions give us almost sure convergence of Q_k to q_* [15]. The α must be a sequence satisfying Equation (54). Moreover ε should be a decreasing sequence of functions on $\mathcal{S}$ that satisfies some conditions. For example, $\varepsilon_k(s) = c/n_k(s)$ where $0 < c < 1$ and $n_k(s)$ is the number that state s is visited until time step k is a good choice. The policies also converge to an optimal policy which happens to be deterministic since $\varepsilon_k \to 0$.

Unfortunately, we do not discuss SARSA(0) in the results although it is similar to Q-learning with learning policies. The further performance of Q-learning with learning polices will be discussed through the results we will obtain.

5.3 Results

We consider a similar approach as we have done in Section 4.3. We have again run 50 simulations with 10000 episodes each to make the results comparable to what we have seen Section 4.3. The initializations we have used are given in the table below.

Table 6: Initialization for Q-learning

Quantity	**without learning policies**	**with learning policies**
α	0.5	0.5
ε	0.3	0.3
Q_0	zero-function	zero-function
π_0	uniform measure on $\mathcal{A}$	uniform measure on $\mathcal{A}$

Recall the notations introduced in Section 4.3.1. We use the symbol QL for Q-learning and QLP for Q-learning with learning policies as subscripts. The size of the states visited can be seen in Table 7. The proportion of states visited is less than the proportion we have seen from the Monte Carlo algorithms. Monte Carlo exploring-starts is actually an exception since it is possible there to see all states. The proportions are quite close to each other if we leave Monte Carlo exploring-starts out. At last, the high proportion obtained by Q-learning is due to the fact that the policy remains uniform throughout the simulation. That leads to more exploration.

Table 7: The size of the states visited in all Q-learning simulations

Quantity	**Cardinality**	**Proportion of total**
$\mathcal{S}^{QL}$	342	77.6%
$\mathcal{S}^{QLP}$	257	58.3%
$\mathcal{S}^{QL} \cap \mathcal{S}^{QLP}$	249	56.5%

5.3.1 Value function approximation

We know that we can get an approximation for the value function once we have an approximation for the action-value function. So we get two approximate value functions from the simulations, namely V_{QL} and V_{QLP}. The logarithm of the approximate value functions can be seen in Figure 11 and Figure 12. We see that the Q-learning algorithm gives a good approximation for the value function, since it is close to Figure 2 on the positions where it is not white.

As usual we compare the results with the value function from value iteration, V_{VI}. The errors are given in Table 8. It seems that Q-learning is performing slightly better than Q-learning with learning policies. These numbers may also suggest that both algorithms are better than Monte Carlo first-visit, since it had a maximum relative error of 0.984.

However, the absolute errors of both Q-learning algorithms are larger than the errors in Monte Carlo. The absolute errors are quite large, since the Q-learning algorithm seems to converge slower than Monte Carlo algorithms. The convergence speed will be discussed later.

Table 8: Absolute and relative error of Q-learning algorithms

	Absolute error		**Relative error**	
Algorithm	Mean	Maximum	Mean	Maximum
without learning policies	0.00538	0.114	0.0570	0.631
with learning policies	0.0179	0.125	0.372	0.874

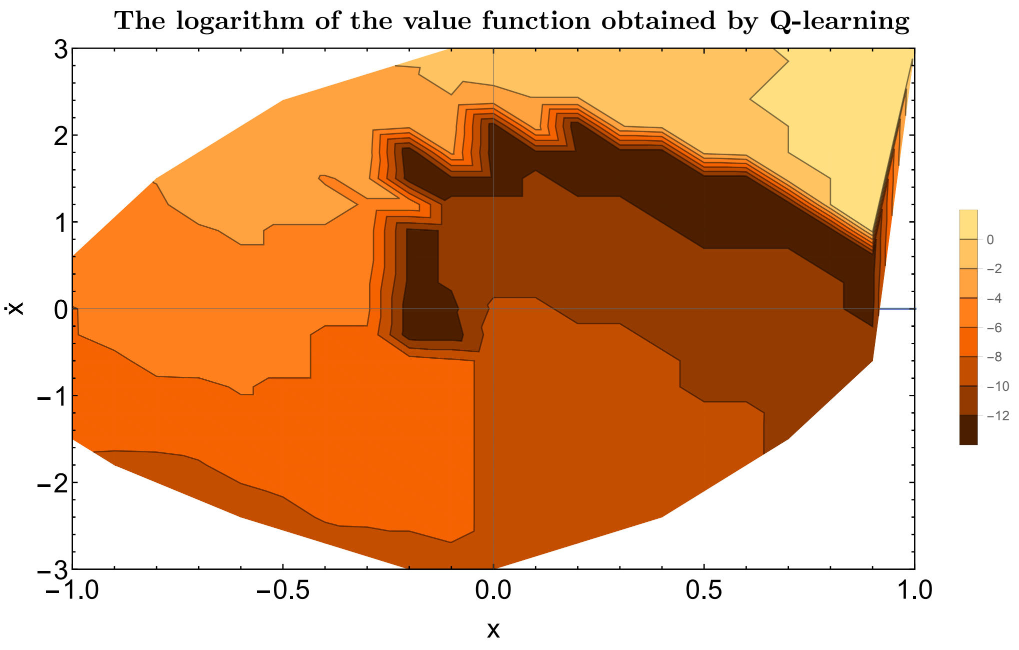


Figure 11: Contour plot of $\log V_{QL}$

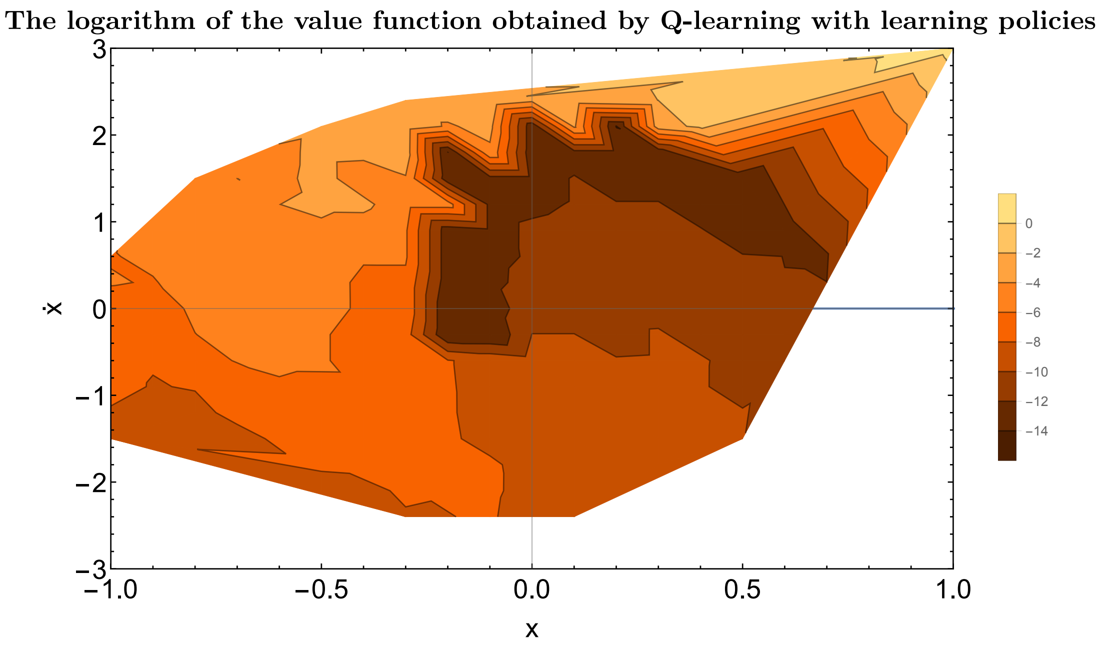


Figure 12: Contour plot of $\log V_{QLP}$

5.3.2 Optimality of policy

It is remarkable that all policies of the Q-learning algorithm are optimal (machine-precision sense). We know from Section 4.3.1 that we can get very accurate value functions $V_{QL}^{(j)}$ of the j-th simulation's policy. Those have maximum absolute difference of order 10^{-17}. In other words all policies obtained from Q-learning give an optimal value function. It is immediate that the number of steps the car needs is 16.

Unfortunately, the Q-learning with learning policies performs very badly. First of all, following five of the obtained policies do not succeed to let the car get to the mountaintop in less than 9999 steps (see Table 9). Only 19 of the policies can do it in 16 steps. One also has

$$\|V_{QLP}^{(j)} - V_{VI}\|_\infty = 2.00 \quad \text{for all } j = 1, ..., 50. \tag{64}$$

A possible explanation for such phenomenon is already given in Section 4.3 for Monte Carlo first-visit. We conclude that none of the policies obtained from Q-learning with learning policies are optimal.

Table 9: The number of steps needed to get to the goal using the policies from Q-learning algorithms

	Number of steps needed to get to the right-end point		
Algorithm	Minimum (frequency)	Mean	Maximum (frequency)
without learning policies	16 (50)	16	16 (50)
with learning policies	16 (19)	NA	<9999 (5)

5.3.3 Convergence speed

We have run 50 simulation with 40000 episodes. We compare the convergence speed of Monte Carlo algorithms with temporal-difference algorithms. We considered the 2-norm of the difference of each value function and the value function from value iteration. We first considered the 2-norm on the whole state space $\mathcal{S}$. The results can be seen in Figure 13. We immediately see that Monte Carlo exploring-starts outperforms all other algorithms as we could expect. We also see that both Monte Carlo algorithms outperform the temporal-difference learning algorithms.

Furthermore, we see that all algorithms but Monte Carlo exploring-starts, get flatter after the episode 20000. That is due to the hypothesis that the fixed initial state of the Markov chain $(S_k, A_k)_{k\geq 0}$ does not communicate with all states in $\mathcal{S}\times\mathcal{A}$. In other words, we do not know whether the Markov chain $(S_k, A_k)_{k\geq 0}$ is irreducible. The value function of the states that can not be visited from the fixed initial state does not get an approximation. That explains why the error in all algorithms but Monte Carlo exploring-starts do not get below some value, which appears to be around 8.24. It should also be clear that the figure is not contradicting Theorem 30.

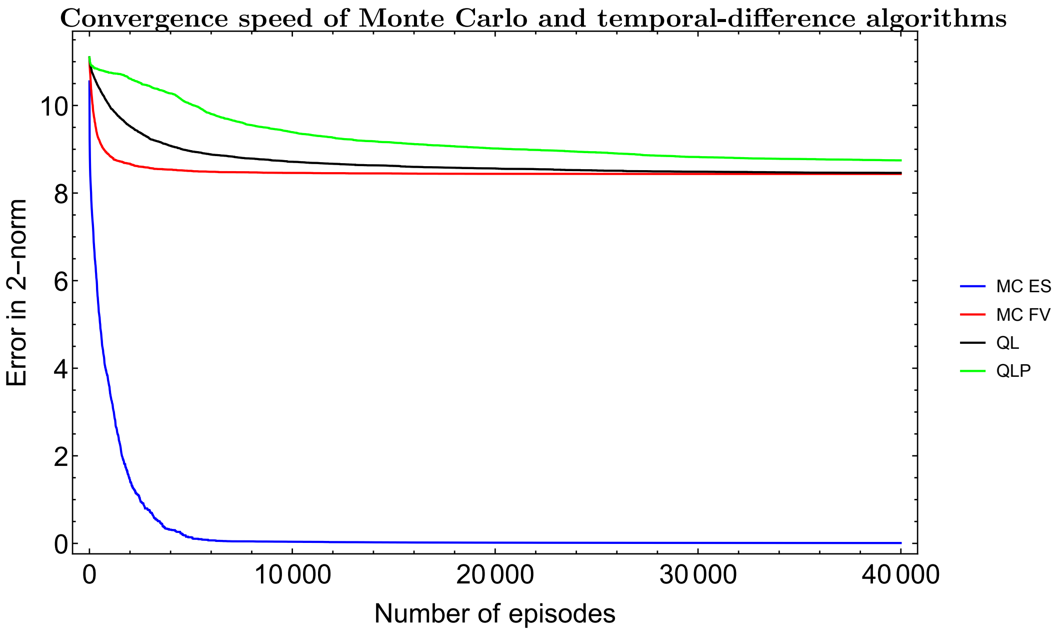


Figure 13: Convergence speed of the Monte Carlo and temporal-difference learning with respect to the number of episodes

It is indeed not very fair to consider the error over the whole state space $\mathcal{S}$. Now we focus on the set of states $s \in \mathcal{S}$ that are visited in all algorithms, i.e. the set $\mathcal{S}^{ES} \cap \mathcal{S}^{FV} \cap \mathcal{S}^{QL} \cap \mathcal{S}^{QLP}$. We call that set *the overlap set*. The cardinality of that set is 301, that is 68.3% of the whole state space. We get another picture of the convergence of the algorithms (see Figure 14). Both Monte Carlo algorithms start decaying fast in the beginning compared to the temporal-difference learning algorithms. However, the temporal-difference learning algorithms catch up with Monte Carlo first-visit eventually.

As we have seen in Section 4.3 it is not strange that the error of Monte Carlo first-visit does not vanish, because the algorithm is very likely to give a value function of a policy which is only optimal among ε-greedy policies. Something similar might be happening to Q-learning with learning policies (see discussion after the pseudocode of Q-learning with learning policies), although the graph suggests it is decaying.

Finally, recall Remark 26 where we said, avoiding extra notations and technical details, that there is a state space $\mathcal{S}' \subset \mathcal{S}$, such that the Markov chain $(S_k, A_k)_{k\geq 0}$ on $\mathcal{S}' \times \mathcal{S}$ is irreducible. Since $\mathcal{S}^{QL} \subset \mathcal{S}'$, we actually have an illustration of Theorem 30 in Figure 14. Of course, that observation should be taken with grain of salt, although it can be written more formally.

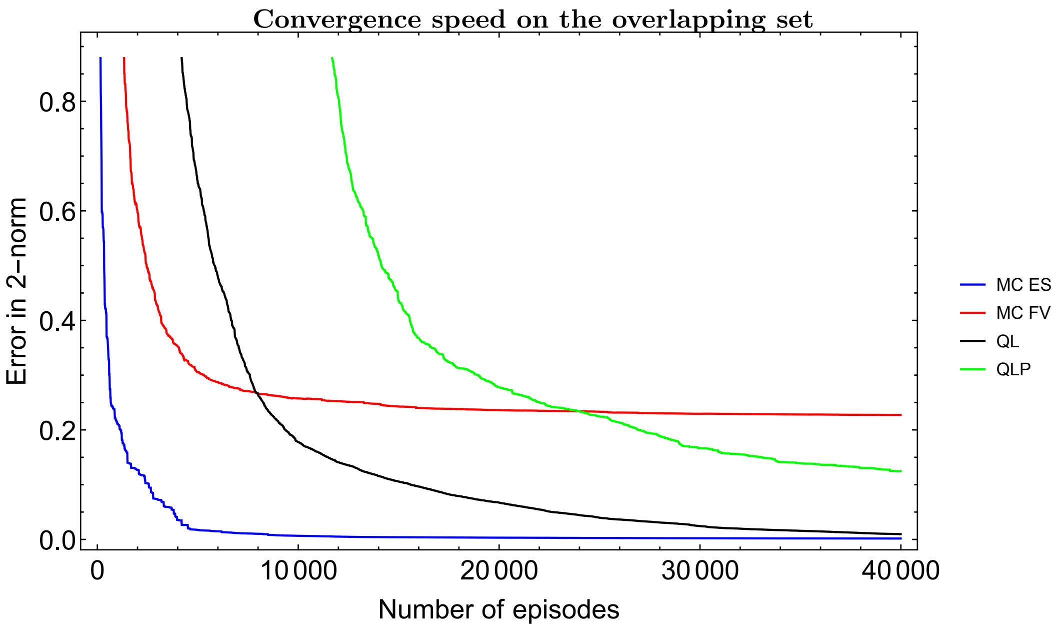


Figure 14: Convergence speed of the Monte Carlo and temporal-difference learning with respect to the number of episodes where only the values in the overlap set $\mathcal{S}^{ES} \cap \mathcal{S}^{FV} \cap \mathcal{S}^{QL} \cap \mathcal{S}^{QLP}$ is considered

We see that among the temporal-difference algorithms the Q-learning outperforms the Q-learning with learning policies. When we compare the temporal-difference algorithms with Monte Carlo algorithms, it seems that Monte Carlo algorithms are better in approximating the value function. The Monte Carlo algorithms give a good approximation overall. But when we only compare how each algorithms performs in the overlap domain, then we see that temporal-difference algorithms catch up with Monte Carlo first-visit.

The policies we have obtained by Q-learning were all optimal. However, some policies from Q-learning with learning policies failed badly. The car did not succeed to get to the goal in less than 9999 steps. Comparing the policies, we conclude that Q-learning is better than Monte Carlo algorithms and that Q-learning with learning policies is the worst among all those algorithms. This might indicate that the interpretation we had about the Q-learning with learning policies as written in [3] was actually wrong. Now it also makes sense why it was hard to find literature about that algorithm.

6 Conclusion

We have investigated what is currently known about the convergence properties of some of the most used reinforcement learning algorithms. The dynamic programming algorithms, value iteration and policy iteration, converge. The convergence proofs that we have given were relying on a less general definition of Markov decision processes. However the proofs can be easily generalized. We have shown that the dynamic programming algorithms in [3] actually terminate. Furthermore both algorithms give an approximation of the optimal value function with an approximation of an optimal policy which can be made arbitrarily close to the exact values. Combining the results obtained in this report with the results in [6] make clear that dynamic programming has very strong convergence properties.

We have also shown that Q-learning converges under extra assumptions on the Markov chain. We have obtained a stronger type of convergence than the type mentioned in [5]. Unfortunately, we could not make similar statements about Q-learning with learning policies, since it was hard to analyze. It might be the case that the interpretation we had about that algorithm was incorrect as discussed in the results. However, an algorithm which is similar to it converges which means that Q-learning with learning policies might actually converge.

Unfortunately, we conclude that it is still not known whether Monte Carlo algorithms converge. However, we have seen that all algorithm perform well on the mountain car problem. That means that there is hope for showing the convergence of Monte Carlo algorithms as they are or maybe under extra assumptions.

6.1 Further research

As a next step, I would investigate the performance of reinforcement learning algorithms for Markov decision processes with uncountable state space. These problems come with many additional challenges. The definitions become more subtle. Moreover the algorithms we have mentioned can not be used for an infinite state space MDP. Finally, one needs to do more work in the implementation. For instance, if the state space consists of possible camera images, then those images need some preparation work in order to use them as an element from the state space. This leads to many mathematical challenges.

References

[1] Martin L. Puterman. *Markov decision processes : discrete stochastic dynamic programming.* Wiley-Interscience, 2005.

[2] Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance.* Universitext. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement Learning : An Introduction.* 2018.

[4] Dimitri P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, Belmont, Mass. :, 1995.

[5] C. L. Beck and R. Srikant. Error bounds for constant step-size Q-learning. *Systems and Control Letters*, 61(12):1203–1208, 2012.

[6] Dimitri P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, Belmont, Mass :, 2005.

[7] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158, 1996.

[8] John N. Tsitsiklis. Asynchronous Stochastic Approximation and Q-Learning. *Machine Learning*, 16(3):185–202, 1994.

[9] Csaba Szepesvári. The asymptotic convergence-rate of Q-learning. In *Advances in Neural Information Processing Systems*, pages 1064–1070, 1998.

[10] Richard. Serfozo. *Basics of applied stochastic processes.* Springer, 2009.

[11] John N Tsitsiklis. On the Convergence of Optimistic Policy Iteration. *J. Mach. Learn. Res.*, 3:59–72, 2003.

[12] Abhijit Gosavi. Boundedness of iterates in Q-Learning. *Systems and Control Letters*, 55(4):347–349, 2006.

[13] Geoffrey R Grimmett and David Stirzaker. Probability and Random Processes, 2001.

[14] Andrei Z. Broder and Anna R. Karlin. Bounds on the cover time. *Journal of Theoretical Probability*, 2(1):101–120, 1989.

[15] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, 38(3):287–308, 2000.