INFO8003 - Optimal decision making for complex problems

Reinforcement Learning in a Discrete Domain (Part 2)

Prof. Ernst Damien
TA: Samy Aittahar - Bardhyl Miftari

Debor Antoine	Antoine.Debor@student.uliege.be
Navez Pierre	Pierre.Navez@student.uliege.be

Academic year 2020-2021

1 Implementation of the domain

The corresponding Python code can be found in the file `section1.py`.

A class named `Domain` has been created. This class contains four attributes and four methods, allowing to get information about the dynamics, the reward as well as the state and action spaces. The four attributes are the reward matrix (as given in the statement), the type of the domain (deterministic or stochastic), the value of the discount factor and the stochastic threshold. The domain instance represented by the reward matrix has been written in a `csv` file which is loaded when the domain is instantiated.

When launching the program, one can add arguments to specify the value of several parameters. In particular, in order to change the type of the domain in which the agent will take actions, one just needs to add `--stochastic` in the command line of the terminal. Otherwise, the deterministic domain is selected by default.

The rule-based policy that has been chosen consists in always going to the right. In other words, the action selected by the intelligent agent in the action space will always be (0, 1).

Starting from the state $x_0 = (0, 3)$ and following the policy described here above, the 10 steps trajectory followed by the agent in the deterministic domain is depicted in FIGURE 1.1, each step being written as (x_t, u_t, r_t, x_{t+1}), with u_t the action performed at time t and r_t the corresponding reward. The trajectory in the stochastic domain is presented in FIGURE 1.2.

The trajectory is computed thanks to the function `trajectory` of the code. This function takes as parameters the type of domain, the initial state and the number of steps the agent will effectuate. Then, using the methods of the `Domain` class, along with the defined policy, one can easily simulate the journey of the agent in the environment.

```
Deterministic domain chosen (default)

Initial state of the agent : (3, 0)

Instance of the domain:
[[ -3   1  -5   0  19]
 [  6   3   8   9  10]
 [  5  -8   4   1  -8]
 [  6  -9   4  19  -5]
 [-20 -17  -4  -3   9]]

Followed trajectory for 10 steps (Deterministic domain):
(x_0, u_0, r_0, x_1) : ((3, 0), (0, 1), -9, (3, 1))
(x_1, u_1, r_1, x_2) : ((3, 1), (0, 1), 4, (3, 2))
(x_2, u_2, r_2, x_3) : ((3, 2), (0, 1), 19, (3, 3))
(x_3, u_3, r_3, x_4) : ((3, 3), (0, 1), -5, (3, 4))
(x_4, u_4, r_4, x_5) : ((3, 4), (0, 1), -5, (3, 4))
(x_5, u_5, r_5, x_6) : ((3, 4), (0, 1), -5, (3, 4))
(x_6, u_6, r_6, x_7) : ((3, 4), (0, 1), -5, (3, 4))
(x_7, u_7, r_7, x_8) : ((3, 4), (0, 1), -5, (3, 4))
(x_8, u_8, r_8, x_9) : ((3, 4), (0, 1), -5, (3, 4))
(x_9, u_9, r_9, x_10) : ((3, 4), (0, 1), -5, (3, 4))
(x_10, u_10, r_10, x_11) : ((3, 4), (0, 1), -5, (3, 4))
```

FIGURE 1.1: 10 steps trajectory in the deterministic domain

```
Stochastic domain chosen

Initial state of the agent : (3, 0)

Instance of the domain:
[[ -3   1  -5   0  19]
 [  6   3   8   9  10]
 [  5  -8   4   1  -8]
 [  6  -9   4  19  -5]
 [-20 -17  -4  -3   9]]

Followed trajectory for 10 steps (Stochastic domain):
(x_0, u_0, r_0, x_1) : ((3, 0), (0, 1), -3, (3, 1))
(x_1, u_1, r_1, x_2) : ((3, 1), (0, 1), 4, (0, 0))
(x_2, u_2, r_2, x_3) : ((0, 0), (0, 1), 1, (0, 1))
(x_3, u_3, r_3, x_4) : ((0, 1), (0, 1), -3, (0, 0))
(x_4, u_4, r_4, x_5) : ((0, 0), (0, 1), 1, (0, 1))
(x_5, u_5, r_5, x_6) : ((0, 1), (0, 1), -5, (0, 0))
(x_6, u_6, r_6, x_7) : ((0, 0), (0, 1), 1, (0, 0))
(x_7, u_7, r_7, x_8) : ((0, 0), (0, 1), -3, (0, 1))
(x_8, u_8, r_8, x_9) : ((0, 1), (0, 1), -3, (0, 2))
(x_9, u_9, r_9, x_10) : ((0, 2), (0, 1), 0, (0, 0))
(x_10, u_10, r_10, x_11) : ((0, 0), (0, 1), -3, (0, 1))
```

FIGURE 1.2: 10 steps trajectory in the stochastic domain

2 Expected return of a policy

The corresponding Python code can be found in the file `section2.py`.

Using the stationary policy stating that the agent will always go to the right, one can approximate the expected return of this policy when starting from the initial state $x_0 = x$ by the following recurrence equation

$$J_N^\mu(x) = E\{r(x, \mu(x), w) + \gamma J_{N-1}^\mu(f(x, \mu(x), w))\}, \quad \forall N \geq 1 \tag{2.1}$$

with $J_0^\mu = 0$ and $\forall x$. This equation is based on the reward signal r associated with state x and the stationary policy $\mu(x)$. The decay factor γ is also taken into account at each iteration step.

For the deterministic case, one can get rid of the mathematical expectation over the disturbance w. In the case where the domain is stochastic, however, the dynamics of the domain differ from the deterministic case and one needs to take w into account, as for $w > 0.5$ a "jump" to state (0,0) is performed.

From the dynamic programming theory, one knows that for a infinite number N of iterations, the approximation of equation 2.1 tends to the true one in infinite norm[1]. Additionally, the theory gives a bound on the approximation error; this bound is expressed as a function of the discount factor γ, the number of iterations N, and the maximum reward signal B_r. Mathematically, this bound writes

$$\|J_N^\mu - J^\mu\|_\infty \leq \frac{\gamma^N}{1-\gamma} B_r. \tag{2.2}$$

[1]Alternatively, one can say that the infinite norm error tends towards 0.

N	Bound
10	1718.33
100	695.46
1000	0.082
3000	1.53×10^{-10}
10000	4.27×10^{-41}

TABLE 2.1: Value of the bound for different number of iterations

The number of iterations N should be chosen large enough so as to reach a reasonably small value for this bound, in order to well approximate the actual expected return, but not too large to avoid useless computation time. TABLE 2.1 illustrates the value of this bound depending on the number of iterations.

It has been decided to use $N = 3000$ as the magnitude of the error is sufficiently small to be negligible in that case, while the number of iterations is still at the order of the thousand and leads to a quite low computing time.

As a result, $J^{\mu}_{3000}(x)$, for each initial state x, in the deterministic domain, is displayed in the TABLE 2.2, with the upper left cell corresponding to state $(0, 0)$. One can observe that the expected return of the policy is maximum for initial state (4,0), where the reward signal is the largest and equal to 19. One can also note that, in a given line, all cells present scores close to each other; this is related to the chosen policy, which prevents the agent from changing its line during its trajectory.

1839.62	1857.19	1881	1900	1900
990.04	997.01	999	1000	1000
-779.3	-779.09	-791	-800	-800
-471.57	-467.24	-476	-500	-500
849.37	875.12	888	900	900

TABLE 2.2: Expected return of the stationary policy over all the state space in the deterministic domain

The resulting expected return in the stochastic domain is completely different, as it can be seen in TABLE 2.3.

-72.02	-71.46	-64.25	-54.75	-54.75
-67.78	-64.91	-64.16	-63.66	-63.66
-77.41	-73.25	-76.99	-81.49	-81.49
-75.35	-68.08	-66.52	-78.52	-78.52
-82.34	-74.12	-70.65	-64.65	-64.65

TABLE 2.3: Expected return of the stationary policy over all the state space in the stochastic domain

In the stochastic case, the reward signal has a value of -3 when a "jump" back to state (0 0) occurs, *i.e.* when w, which is uniformly distributed on $[0, 1]$, is greater than 0.5. This can explain the low (negative) scores observed in this case.

3 Optimal policy

The corresponding Python code can be found in the file `section3.py`.

In order to derive the optimal policy, one first needs to estimate the parameters of the Markov decision process (MDP) associated with the domain. An MDP is based on some transition probabilities $p(x'|x,u)$ and a reward function $r(x,u)$. The transition probabilities express the probability of reaching the state x' from the state x taking the action u. The reward function is simply the reward that the agent will acquire by taking the action u from the state x.

In the deterministic domain, the transition probability associated to a future state x' and an action u performed from a state x is either 1 or 0, as there is only one state x' reachable from the state x through the action u. Therefore, one can again get rid of the mathematical expectation over w for this case. On the contrary, in the stochastic domain, there is an uncertainty on the reached state due to the stochastic dynamics of the domain, and the conditional probabilities are derived taking into account the probability of "jumping" to the state (0,0).

Similarly, the reward is either perfectly known in the deterministic domain, in which case one can again get rid of the mathematical expectation over w, or it is a random variable if the domain is stochastic.

Then, the Q-function is computed using the recurrence equation

$$Q_N(x,u) = E\{r(x,u,w) + \gamma \max_{u' \in U} Q_{N-1}(f(x,u,w),u')\}, \quad \forall N \geq 1 \tag{3.1}$$

with $Q_0(x,u) = 0$. The state-action value functions $Q_N(x,u)$ are computed for each initial state x and each possible action u by the computation of N iterations of the recurrence equation.

It can be shown that when N goes to infinity, $Q_N(x,u)$ tends to the Q-function in infinite norm[2], defined as the unique solution of the Bellman equation which allows to find the optimal control policy $\mu^*(x)$, as a control policy $\mu^*(x)$ is optimal if and only if $\mu^*(x) \in \arg\max_{u \in U} Q(x,u)$. The associated value function can be found as well, using

$$J_{\mu^*} = \max_{u \in U} Q(x,u). \tag{3.2}$$

A theoretical upper bound on the suboptimality of μ_N^* with respect to μ^* allows to choose a value of N large enough to provide a good approximation, but not excessively large in order to avoid useless computation time. Equation (3.3) defines this bound, and it is computed in Table 3.1 for different values of N.

$$\|J^{\mu_N^*} - J^{\mu^*}\|_\infty \leq \frac{2\gamma^N}{(1-\gamma)^2} B_r. \tag{3.3}$$

N	Bound
10	343665.18
100	193092.29
1000	16.41
3000	3.05×10^{-8}
10000	8.55×10^{-39}

Table 3.1: Value of the bound for different number of iterations

[2] Alternatively, one can say that the infinite norm error tends towards 0.

Again, as in the previous section, a value $N = 3000$ is chosen, as it leads to a good accuracy while implying acceptable computing time.

For the deterministic case, the results are presented in Table 3.2 and Table 3.3.

1842.03	1857.19	1881	1900	1900
1854.58	1870.28	1881.09	1891	1900
1842.03	1855.58	1870.28	1881.09	1881
1828.61	1849.01	1863.65	1863.28	1864.09
1816.32	1826.52	1849.01	1863.65	1842.01

Table 3.2: Expected return of the optimal policy in the deterministic domain for all states

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
↑	→	→	↑	↑
↑	→	↑	↑	←

Table 3.3: Optimal policy in the deterministic domain for all states

One can observe that, whatever its initial state, the optimal policy brings the agent towards the upper right state, that is the one corresponding to the highest reward, and makes it stay there afterwards. It is also obvious that the expected returns are way better than for a naive rule-based policy as used in the previous sections.

For the stochastic case, the results are presented in Table 3.4 and Table 3.5.

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.9	167.63	172.13
159.45	160.14	163.05	167.21	167.63
159.26	162.2	167.21	162.2	167.21
159.26	155.71	162.2	167.21	162.23

Table 3.4: Expected return of the optimal policy in the stochastic domain for all states

↓	↓	↓	→	→
→	→	→	→	↑
↑	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

Table 3.5: Optimal policy in the stochastic domain for all states

One can observe that the optimal policy is different from the deterministic one, and no longer necessarily brings the agent to the upper right state. This could be explained by the fact that, since at any step the agent can "jump" to the state (0,0) (which has a negative associated reward), the

optimal behaviour consists in trying to maximize the short-term[3] return rather than planning a long-term strategy to reach the best state in terms of reward. It is also obvious that the expected returns are way better than for a naive rule-based policy as used in the previous sections. One however observes that, quite logically, these expected returns are smaller than for the deterministic case, as there is a 0.5 probability to jump to the state (0,0) at any step, and therefore to get a -3 reward.

To determine the smallest value of N, N_{min}, such that a greater value does not change the policy inferred from the last Q-function of the sequence, one repeats the previously described process for finding the optimal policy for growing values of N, starting from $N = 1$. Once N is such that the derived optimal policy is not different anymore from the one derived for $N - 1$, one has found $N_{min} = N - 1$.

The results of this experience can be found in TABLE 3.6. One has observed that for $N > 7$, the same policy is obtained as for $N = 7$; hence, for both types of domain, $N_{min} = 7$. Indeed, one can compare TABLE 3.3 with TABLE 3.6m, as well as TABLE 3.5 regarding TABLE 3.6n and see that there is no difference of policy for each state as soon as $N = 7$. For each type of domain, the derived policy is thus the optimal one for $N \geq 7$.

[3]Related to the next few steps.

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↑	↓	↑
←	←	→	←	←
↑	→	↑	↑	↓

(a) N=1, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↑	↓	↑
←	←	→	←	←
↑	→	↑	↑	↓

(b) N=1, stochastic domain

↓	↓	→	→	→
←	→	→	→	↑
↓	→	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(c) N=2, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(d) N=2, stochastic domain

↓	↓	→	→	→
→	→	→	→	↑
↓	→	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(e) N=3, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(f) N=3, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↓	↑	↑
←	→	→	←	←
↑	→	↑	↑	←

(g) N=4, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(h) N=4, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
→	→	→	↑	←
↑	→	↑	↑	↓

(i) N=5, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(j) N=5, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
←	→	→	↑	↑
↑	→	↑	↑	←

(k) N=6, deterministic domain

↓	↓	↓	→	→
→	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(l) N=6, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
↑	→	→	↑	↑
↑	→	↑	↑	←

(m) N=7, deterministic domain

↓	↓	↓	→	→
→	→	→	→	↑
↑	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(n) N=7, stochastic domain

TABLE 3.6: On the left, the evolution of the optimal policy in the deterministic domain for N=1 to 7. On the right, the evolution of the optimal policy in the stochastic domain for N=1 to 7

4 System identification

The corresponding Python code can be found in the file `section4.py`.

The goal is to retrieve the equivalent Markov decision process given a trajectory

$$h_t = (x_0, u_0, r_0, x_1, u_1, r_1, ...x_{t-1}, u_{t-1}, r_{t-1}, x_t).$$

In other words, from h_t the agent will infer an approximation of $p(x'|x,u)$ an $r(x,u)$. As explained in the first part of this work, these are the probability of reaching the state x' from the state x taking action u, along with the reward function associated with the action u performed from the state x.

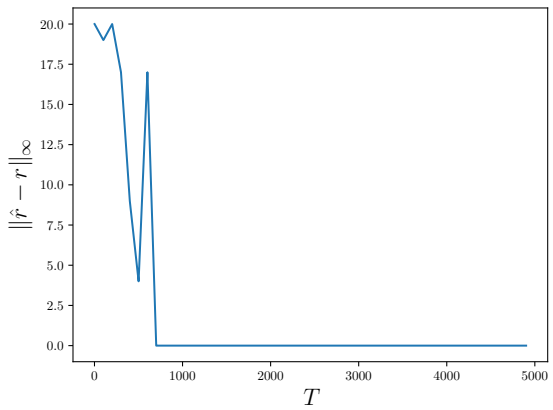
The first step is to obtain $\hat{r}$, the approximation of r. Given the overall trajectory, $\hat{r}(x,u)$ is defined as the mean of the random variable $r(x,u,w)$ acquired over the subset of tuples $(x_k, u_k) = (x,u)$ of the trajectory. If $A(x,u)$ denotes the set such that $A(x,u) = \{k \in 0,1,..,t-1 | (x_k,u_k) = (x,u)\}$ and $\#A(x,u)$ denotes the cardinality of the set, to each value k of the set, is associated an occurrence r_k of the random variable $r(x,u,w)$. One can thus express $\hat{r}(x,u)$ as the following unbiased estimator

$$\hat{r}(x,u) = \frac{\sum_{k \in A(x,u)} r_k}{\#A(x,u)}. \tag{4.1}$$

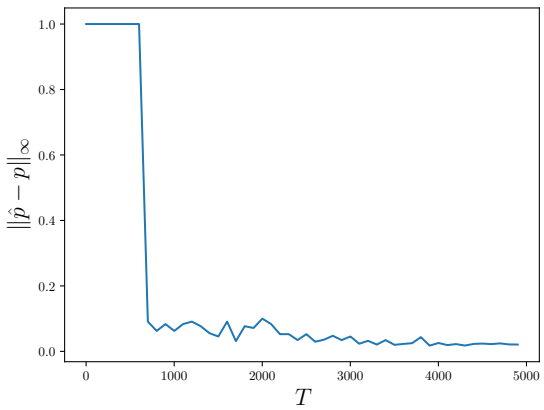
A slightly different reasoning applies to compute $\hat{p}(x'|x,u)$. In that case, one will consider the random variable $I_{\{x'=f(x,u,w)\}}$ which is basically the indicator function, that takes either a value of 1 or 0. There are $\#A(x,u)$ of them and for each possible state x' of the domain, regarding the overall trajectory and the subset $A(x,u)$, the estimator is

$$\hat{p}(x'|x,u) = \frac{\sum_{k \in A(x,u)} I_{\{x_{k+1}=x'\}}}{\#A(x,u)}. \tag{4.2}$$

Obviously, the greater the number of steps in the trajectory, the most accurate will be the estimator. One can illustrate the convergence of the estimators $\hat{r}(x,u)$ and $\hat{p}(x'|x,u)$ to the true functions $r(x,u)$ and $p(x'|x,u)$ by plotting their difference in infinite norm as a function of the number of steps. These plot are depicted on FIGURE 4.1 for the deterministic domain, and on FIGURE 4.2 for the stochastic domain.



(a) Convergence speed of $\hat{r}(x,u)$ in the deterministic domain

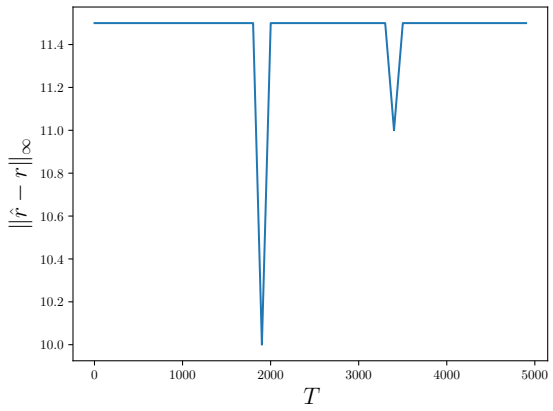


(b) Convergence speed of $\hat{p}(x'|x,u)$ in the deterministic domain

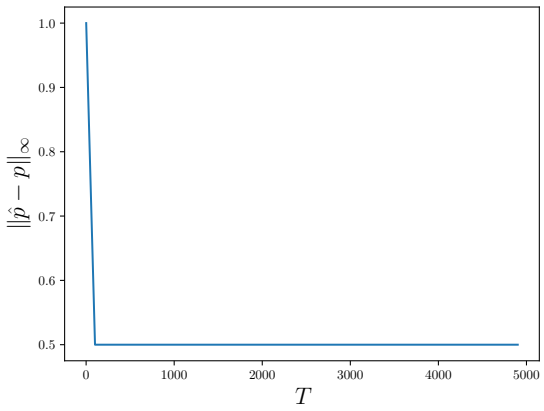
FIGURE 4.1: Convergence speed of $\hat{r}(x,u)$ and $\hat{p}(x'|x,u)$ in the deterministic domain

In the deterministic domain, the reinforcement learning agent can infer precisely $r(x, u)$ in less than 1000 steps in its trajectory. After about 750 steps, the error converges towards 0. Moreover, one can observe that the difference between r and $\hat{r}$ is bounded by the maximum reward value because the reward function has been initialized with 0 all over the state action space.

Concerning the approximation $\hat{p}$, the first observation is that there is a minimal number of steps to be taken before decreasing the error. Indeed, on FIGURE 4.1b, the error is maximal until crossing the 500 first steps. Clearly, the agent needs to explore the state action space a minimum before being able to learn a probability distribution. After that has happened, the error oscillates between 0 and 0.1 but still decreases towards 0 as the number of steps gets bigger.



(a) Convergence speed of $\hat{r}(x, u)$ in the stochastic domain



(b) Convergence speed of $\hat{p}(x'|x, u)$ in the stochastic domain

FIGURE 4.2: Convergence speed of $\hat{r}(x, u)$ and $\hat{p}(x'|x, u)$ in the stochastic domain

In the stochastic domain, however, the results show that the error does not converge towards 0, neither for the approximation of $r(x, u)$ nor for the approximation of $p(x'|x, u)$. On FIGURE 4.2a one can see that the value of convergence is about 11,4. Indeed, the randomness of the dynamics makes the agent unable to learn precisely the reward function, taking a particular action. Furthermore, there is two decreases in the error at two particular number of steps, but these decreases are not so important and happen randomly.

The convergence of $p(x'|x, u)$ depicted on FIGURE 4.2b comes very much faster than in the deterministic domain as the agent only needs about a hundred steps before the infinite norm error reaches its asymptotic value of 0.5, which corresponds to the probability for the agent to "jump" at the state $(0, 0)$ at any move. To conclude, the agent is unable to learn an accurate probability distribution due to the randomness of its dynamics.

Using these estimators, one can go one step further and estimate the Q-function by the recurrence equation defined hereunder

$$\hat{Q}_N(x, u) = \hat{r}(x, u) + \gamma \sum_{x' \in X} \hat{p}(x'|x, u) \max_{u' \in U} \hat{Q}_{N-1}(x', u'), \quad \forall N \geq 1 \tag{4.3}$$

With $\hat{Q}_0(x, u) = 0$ and $\forall (x, u)$. One can measure the convergence of $\hat{Q}(x, u)$ towards the real Q-function by computing their difference in infinite norm for several sizes of the trajectory. With $N = 1000$ iterations of the recurrence equation, the results are displayed on TABLE 4.1.

Number of steps	$\|\hat{Q}_N(x,u) - Q_N(x,u)\|_\infty$
10	1899.92
100	1899.92
1000	132.25
10000	20.46
100000	1.76
1000000	0.19

(a) Deterministic domain

Number of steps	$\|\hat{Q}_N(x,u) - Q_N(x,u)\|_\infty$
10	438.84
100	172.12
1000	440.72
10000	172.12
100000	167.2
1000000	162.22

(b) Stochastic domain

TABLE 4.1: Infinite norm of the error for both deterministic and stochastic domains

One can see that, in the deterministic case, the infinite norm of the error decreases towards a numerical zero with the number of steps increasing. For an horizon large enough, typically greater than around $T = 1000$ according to TABLE 4.1a, one observes a convergence for the approximated Q-function. In the stochastic case however, as the agent was unable to well estimate the Markov decision process, $\hat{Q}$ does not converge towards Q. This is illustrated in TABLE 4.1b.

One can use $\hat{Q}_N$ to derive the approximation of the optimal policy $\hat{\mu}_N^*$ which is given by the following relation

$$\hat{\mu}_N^* = \arg\max_{u \in U} \hat{Q}_N(x,u) \quad \forall x \in X. \tag{4.4}$$

The length of the trajectory is fixed to 5000. This is a trade-off between computation time and performance. The optimal policy of each domain is illustrated on TABLE 4.2.

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
↑	→	→	↑	↑
↑	→	↑	↑	←

(a) Deterministic domain

↓	↓	↓	↓	↓
→	↓	←	←	↓
↓	↓	↑	←	←
←	←	↓	→	↑
↓	↓	↓	↓	↓

(b) Stochastic domain

TABLE 4.2: Approximation of the optimal policies for both types of domain

The results corroborates what was found when discussing the convergence of $\hat{Q}$. Indeed, in the deterministic domain, one can retrieve a policy that is exactly the same than the true optimal one, found in the first part of this assignment. In the stochastic domain, as expected, the approximation of the optimal policy does not correspond to the real one.

With $\hat{Q}$ at disposal, the expected return of the approximation of the optimal policy $J_{\hat{\mu}^*}^N$ can be computed as well, and compared with respect to the expected return $J_{\mu^*}^N$ of the true optimal policy. The results of these computations are tabulated in TABLE 4.3 and TABLE 4.4.

1841.95	1857.11	1880.91	1899.92	1899.92
1854.49	1870.20	1881.01	1890.92	1899.91
1841.95	1855.49	1870.19	1881.01	1890.92
1828.53	1848.92	1863.56	1863.20	1864.01
1816.24	1826.44	1848.93	1863.56	1841.93

(a) Expected return $J^N_{\mu^*}$ of the true optimal policy in the deterministic domain for all states

1841.95	1857.11	1880.91	1899.92	1899.92
1854.49	1870.20	1881.01	1890.92	1899.91
1841.95	1855.49	1870.19	1881.01	1890.92
1828.53	1848.92	1863.56	1863.20	1864.01
1816.24	1826.44	1848.93	1863.56	1841.93

(b) Expected return $J^N_{\hat{\mu}^*}$ of the approximated optimal policy in the deterministic domain for all states

TABLE 4.3: Comparison between $J^N_{\hat{\mu}^*}$ and $J^N_{\mu^*}$ in the deterministic domain

In the deterministic domain, as well as the approximation of the optimal policy corresponds to the true one, the expected returns found on TABLE 4.3a and TABLE 4.3b are equivalent as well.

159.44	159.63	163.05	172.12	172.12
159.63	163.05	164.90	167.62	172.12
159.44	160.13	163.05	167.21	167.62
159.25	162.19	167.21	162.20	167.21
159.25	155.71	162.19	167.21	162.22

(a) Expected return $J^N_{\mu^*}$ of the true optimal policy in the stochastic domain for all states

458.22	463.29	459.39	469.48	450.64
464.25	477.88	466.55	469.89	450.64
474.53	511.70	469.89	450.64	450.64
599.97	599.97	0	428.75	438.13
450.64	450.64	0	0	0

(b) Expected return $J^N_{\hat{\mu}^*}$ of the approximated optimal policy in the stochastic domain for all states

TABLE 4.4: Comparison between $J^N_{\hat{\mu}^*}$ and $J^N_{\mu^*}$ in the stochastic domain

In the stochastic domain, it is interesting to notice in TABLE 4.4 that the expected return of the approximated optimal policy is nil at some states of the domain. This means that during the exploration phase, the agent did not reach these states and thus is unable to estimate the return from those states. Moreover, there is a non negligible margin between the returns of the optimal policy and the approximation. To conclude, this approximation method does not fit stochastic domains.

5 Q-Learning in a batch setting

The corresponding Python codes can be found in the file `section5.py` and `section5_bis.py`.

5.1 Offline Q-Learning

In order to determine how large the time horizon T should be, one first implements a routine which iteratively updates $\hat{Q}(x_k, u_k)$ with Q-learning from a given trajectory, considering a constant learning rate $\alpha = 0.05$. The discount factor is again equal to 0.99, and the update equation is given by

$$\hat{Q}\left(x_k, u_k\right) \leftarrow \left(1 - \alpha_k\right) \hat{Q}\left(x_k, u_k\right) + \alpha_k \left(r_k + \gamma \max_{u \in U} \hat{Q}\left(x_{k+1}, u\right) \right). \tag{5.1}$$

Then, for different sizes of the latter, one runs this routine and evaluates the difference in infinite norm between the corresponding approximated expected return $J^N_{\hat{\mu}^*}$[4] and the one computed before, *i.e.* one computes

$$\| J^N_{\hat{\mu}^*} - J^N_{\mu^*} \|_\infty.$$

These results can be seen in FIGURE 5.1 for the deterministic domain, in which one can see that the larger the time horizon T, the more accurate the approximation $J^N_{\hat{\mu}^*}$.

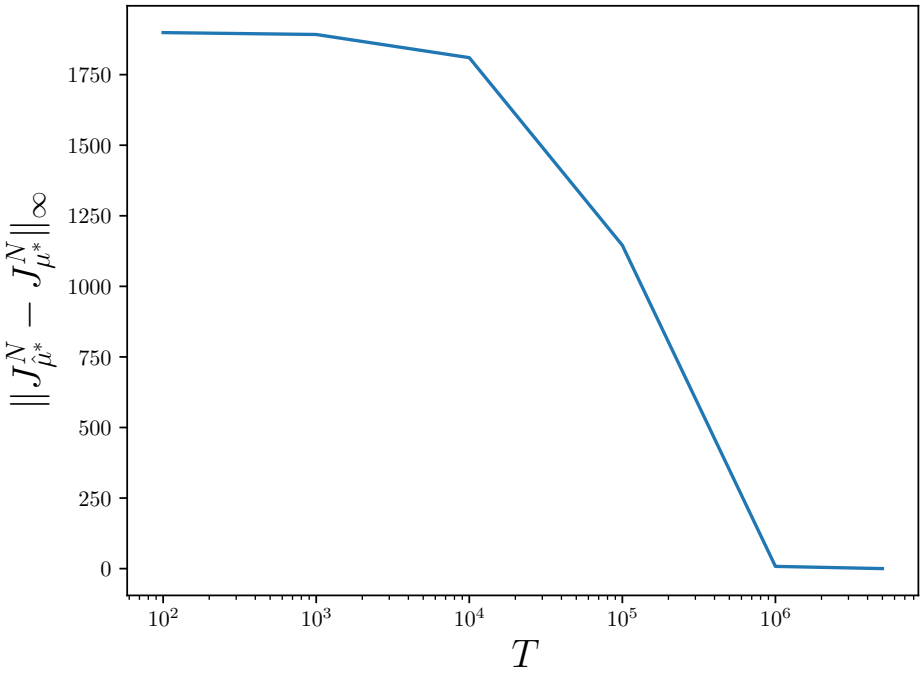


FIGURE 5.1: Convergence speed of $J^N_{\hat{\mu}^*}$ towards $J^N_{\mu^*}$, for a deterministic domain

One can thus observe that, for $T \geq 10^6$, the approximation converges to the actual value $J^N_{\mu^*}$. For $T = 5 \cdot 10^6$, one indeed obtains the results presented in TABLE 5.1, which are equal to those of 5.2.

[4]Computed using equation 3.2.

1842.03	1857.19	1881	1900	1900
1854.58	1870.28	1881.09	1891	1900
1842.03	1855.58	1870.28	1881.09	1891
1828.61	1849.01	1863.65	1863.28	1864.09
1816.32	1826.52	1849.01	1863.65	1842.01

TABLE 5.1: Expected return of the approximated optimal policy $J^N_{\hat{\mu}^*}$ in the deterministic domain for all states

1842.03	1857.19	1881	1900	1900
1854.58	1870.28	1881.09	1891	1900
1842.03	1855.58	1870.28	1881.09	1891
1828.61	1849.01	1863.65	1863.28	1864.09
1816.32	1826.52	1849.01	1863.65	1842.01

TABLE 5.2: Expected return of the optimal policy $J^N_{\mu^*}$ in the deterministic domain for all states

Similarly, the approximated optimal policy $\hat{\mu}^*$ can be again extracted from $\hat{Q}$ and is presented in TABLE 5.3.

$\downarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$
$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\uparrow$
$\uparrow$	$\rightarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$\uparrow$	$\rightarrow$	$\rightarrow$	$\uparrow$	$\uparrow$
$\uparrow$	$\rightarrow$	$\uparrow$	$\uparrow$	$\leftarrow$

TABLE 5.3: Approximated optimal policy in the deterministic domain for all states

One can thus observe that, for the deterministic domain, the optimal policy $\hat{\mu}^*$ derived from $\hat{Q}$ is the same as the previously derived one, *i.e.* μ^* derived from Q.

For a stochastic domain, one performs the same process as the one depicted before to determine a large enough time horizon value T, and one obtains the results presented in FIGURE 5.2.

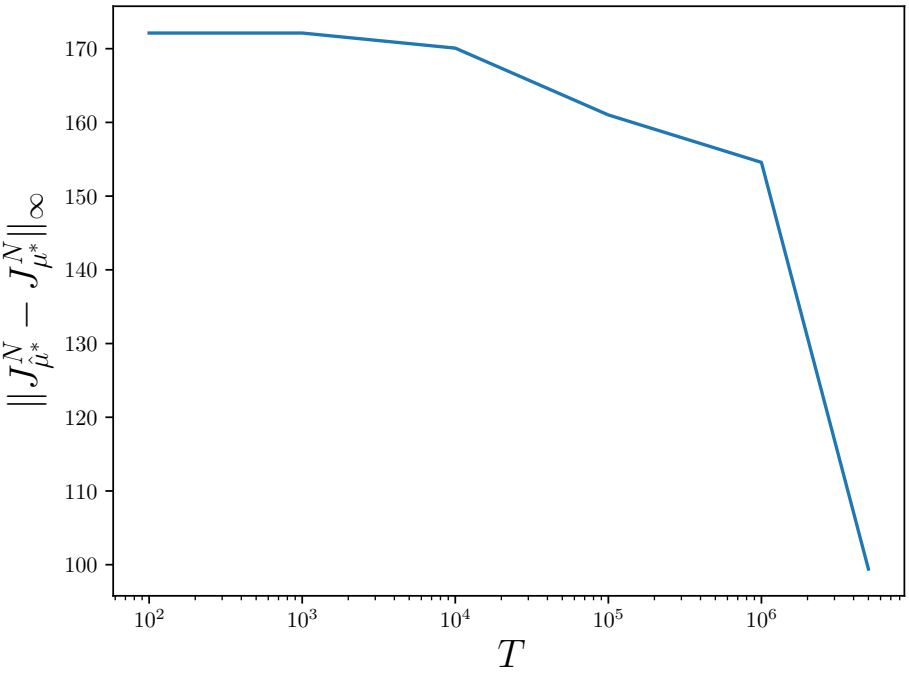


FIGURE 5.2: Convergence speed of $J^N_{\hat{\mu}^*}$ towards $J^N_{\mu^*}$, for a stochastic domain

As for the results presented in a deterministic domain, one can observe from this figure that the accuracy of the approximation increases with the horizon T. However, one can see that, even for $T = 5 \cdot 10^6$, one does not reach convergence, as the error is still quite large for this value of T. To reach convergence, one would need in this case a very large value of T to visit all states a sufficiently large amount of times to provide an approximation of good quality; however, this is not achievable in a reasonable amount of time on our machine. Therefore, for a value $T = 5 \cdot 10^6$, one obtains the results presented in TABLE 5.4, which are obviously not equal to those of TABLE 5.5, which corroborates what has just been explained regarding the convergence.

178.14	177.05	181.5	190.55	192.37
177.53	180.26	182.4	185.4	190.14
179.8	180.02	182.8	186.9	185.64
179.64	182.5	188.3	179.94	148.07
178.21	173.76	179.1	146.42	62.8

TABLE 5.4: Expected return of the approximated optimal policy $J^N_{\hat{\mu}^*}$ in the stochastic domain for all states

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.9	167.63	172.13
159.45	160.14	163.05	167.21	167.63
159.26	162.2	167.21	162.2	167.21
159.26	155.71	162.2	167.21	162.23

TABLE 5.5: Expected return of the optimal policy $J^N_{\mu^*}$ in the stochastic domain for all states

Again, one can extract the approximated optimal policy $\hat{\mu}^*$ from $\hat{Q}$; it is presented in TABLE 5.6 and is similar but not equivalent to μ^* presented in TABLE 3.5.

↓	↓	→	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	←

TABLE 5.6: Optimal policy in the stochastic domain for all states

5.2 Online Q-Learning

First of all, it is worth noting that two different algorithms have been implemented for this section[5], which both could correspond to the one described in the statement. These implementations have brought significantly different results and it is interesting to present them both.

Basically, the difference between Q-learning in offline and online mode is that, in the former, the agent tries to learn an optimal policy without actually following this policy. In other words, Q-learning is applied on a single trajectory, drawn at random without following any specified policy. In online mode, the reinforcement learning algorithm sequentially estimates the Q-function, and uses this estimate to derive an optimal policy to be followed along a trajectory upon which the Q-function will be updated. This policy is followed in a ε-greedy fashion, *i.e.* an action is chosen according to the "optimal" policy with a probability $P(exploitation) = 1 - \varepsilon$, or chosen at random with a probability $P(exploration) = \varepsilon$; this ε-greedy policy aims at providing a good balance between exploitation and exploration.
Let us pass through the three different online policies that have been implemented and let us explain what they consist in.

Protocol 1

The first experimental protocol is the following. The agent trains over 100 episodes having 1000 transitions in the domain, each episode starting from the state $(3, 0)$. The learning rate α is set to 0.05 while the exploration rate $\varepsilon = 0.25$. These values are constant over time and $\hat{Q}$ is updated after each transition.

The first strategy which has been followed to implement this protocol is to initialize $\hat{Q}$ to 0 all over the state-action space and then to continuously update its value over the 100 episodes of 1000 transition steps with the recurrence equation 5.1, taking a new step in the environment at each iteration following an ε-greedy policy according to the current value of $\hat{Q}$. In other words, the value of $\hat{Q}$ is updated at each transition and the following transition (for a same episode) is chosen following an ε-greedy policy according to this updated $\hat{Q}$.

The second strategy is to generate a new 1000 steps trajectory beforehand at each episode, following an ε-greedy policy according to the value of $\hat{Q}$ updated at the end of the previous episode. For the first episode, the considered $\hat{Q}$ is the one initialized in the same way as for the first strategy. Now, the agent follows a pre-defined trajectory at each episode and accordingly updates $\hat{Q}$ at each transition.

For these two strategies, one can estimate the convergence of $\hat{Q}$ towards $J^N_{\mu^*}$ as the number of episodes increases by plotting their difference in infinite norm on a diagram. This evolution can be

[5]One in the file `section5.py` and one in the file `section5_bis.py`

observed for the two strategies in FIGURE 5.4 for the deterministic case and in FIGURE 5.5 for the stochastic one, along with the results obtained for the two other followed protocols.

Protocol 2

The second protocol is exactly the same as the first one, except that the learning rate α is updated after each transition of each episode. Starting from $\alpha_0 = 0.05$, it is updated following $\alpha_t = 0.8 * \alpha_{t-1}$, $\forall t > 0$. This implies that the learning rate decreases along the time. The learning rate determines to what extent newly acquired information overrides old information. A learning rate of 0 makes the agent not learn anything while a learning rate being equal to 1 would make the agent consider only the most recent information. That being said, one would expect the results to not be as good as for the first protocol, since the update rule for α causes it to become negligible after a bit more than 20 transitions. For a negligible learning rate, $\hat{Q}$ stops evolving according to equation 5.1. This can be observed in FIGURE 5.3.

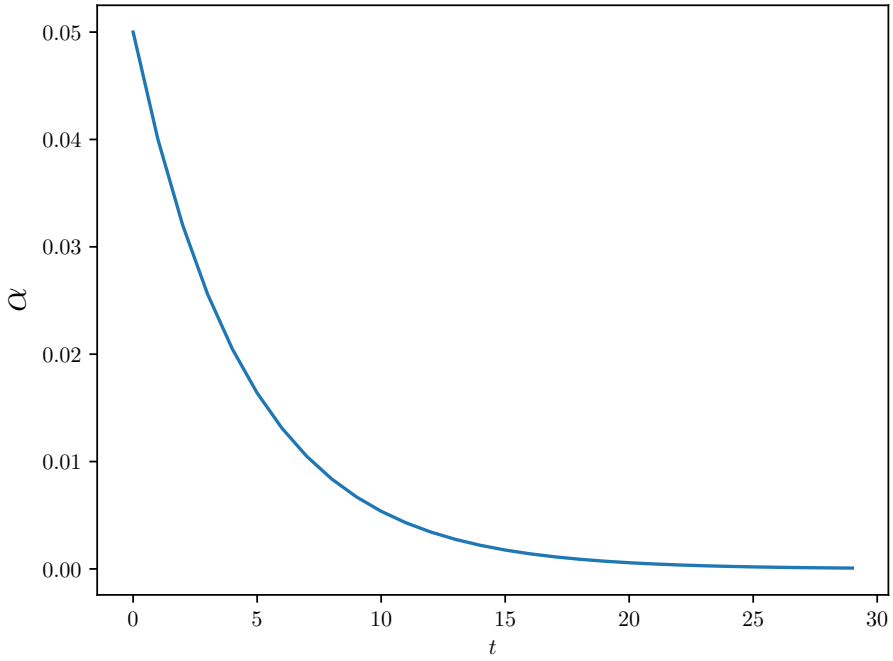


FIGURE 5.3: Evolution of alpha for $\alpha_t = 0.8 * \alpha_{t-1}$, $\forall t > 0$ and $\alpha_0 = 0.05$

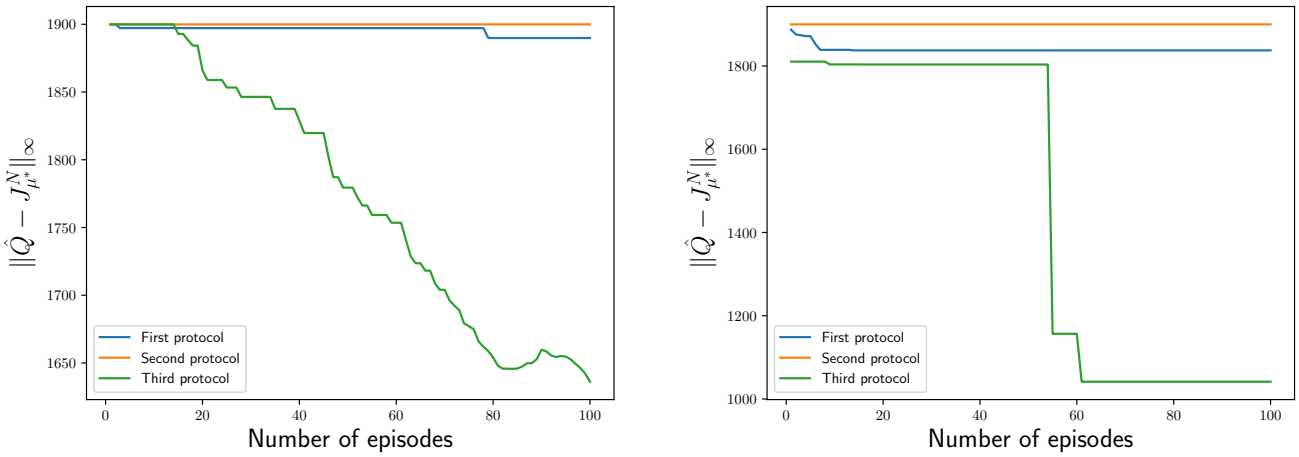
The two versions of the algorithm described in the first protocol have been implemented. Again, the results can be seen in FIGURE 5.4 and FIGURE 5.5.

Protocol 3

In the third protocol, one will consider the parameters $\alpha = 0.05$ and $\varepsilon = 0.25$ constant over the time as in the first protocol. What differs is that in this case, the tuples (x_t, a_t, r_t, x_{t+1}) are stored in a buffer such that at each time-step, $\hat{Q}$ is updated ten times by drawing ten transitions at random in the replay buffer. For the first strategy, each new transition is thus stored in the buffer, and can be drawn in the 10-transitions replay loop of the following transitions. For the second strategy, each transition of the trajectory generated prior to each episode is stored in the buffer and can be drawn for any transition of the corresponding episode.
Again, the results can be seen in FIGURE 5.4 and FIGURE 5.5.

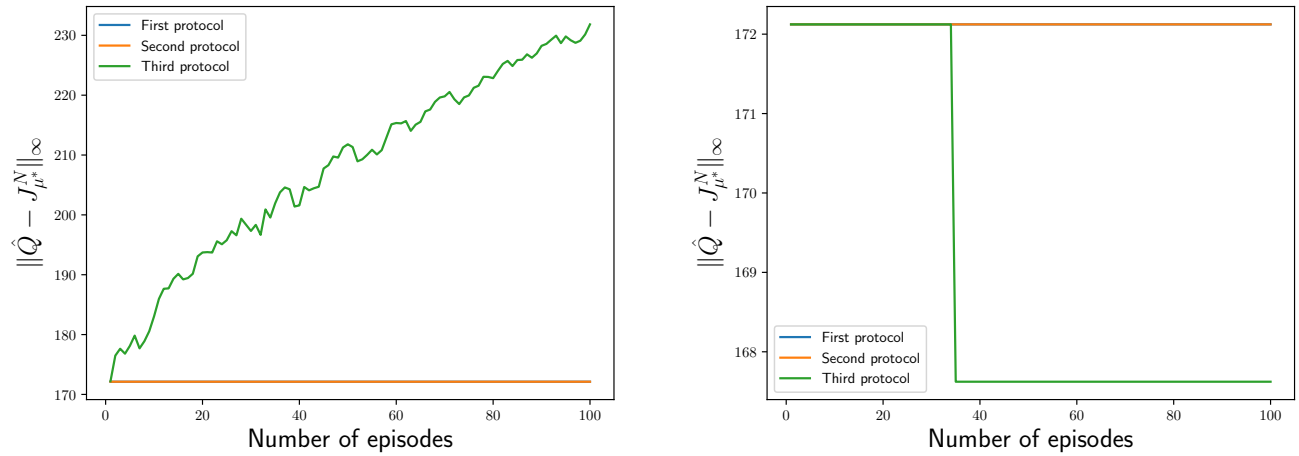
Results

The above-mentioned results can be seen in FIGURE 5.4 for the deterministic case and in FIGURE 5.5 for the stochastic one. Both strategies are presented.



(a) Convergence for the first strategy (b) Convergence for the second strategy

FIGURE 5.4: Convergence of $\hat{Q}$ in the deterministic domain



(a) Convergence for the first strategy (b) Convergence for the second strategy

FIGURE 5.5: Convergence of $\hat{Q}$ in the stochastic domain

In the deterministic domain, using the first strategy (FIGURE 5.4a), the general observation that can be done is that the first protocol sees its error decreasing a bit as the number of episodes increases, but this "convergence" comes after about 80 episodes and is not very pronounced. The second protocol does not converge, as expected, due to the too large decrease of the learning rate factor over the time. The third protocol, meanwhile, is way better efficient, as the corresponding infinite-norm error decreases much more than for the first one. The minimum value is obtained for a number of episodes equal to 100, *i.e.* the maximum number permitted in these experiments. This enhanced efficiency is related to the experience replay introduced in this third protocol.

The second strategy, illustrated on FIGURE 5.4b, provides the same tendency with some exceptions. First, the infinite norm difference $\|\hat{Q} - J^N_{\mu^*}\|_\infty$ starts at a smaller value from the first episode for the third protocol. Second, regarding the first protocol, the decrease in the convergence curve comes after a smaller number of episodes, and its asymptotic value is a bit more than 1800 while in the first strategy, it was a bit under 1900 which is in fact, the maximal expected return of the deterministic domain. Similarly to the first strategy, the second protocol does not converge at all as the convergence curve is a straight line. The major difference comes considering the third protocol. In that case, the convergence curve drops to nearly 1000 after 60 episodes while the first strategy shows that for the same number of episodes, the infinite norm difference lies between 1700 and 1750. Moreover, after the sequence of 100 episodes, this same quantity is barely about 1650 for the first strategy.

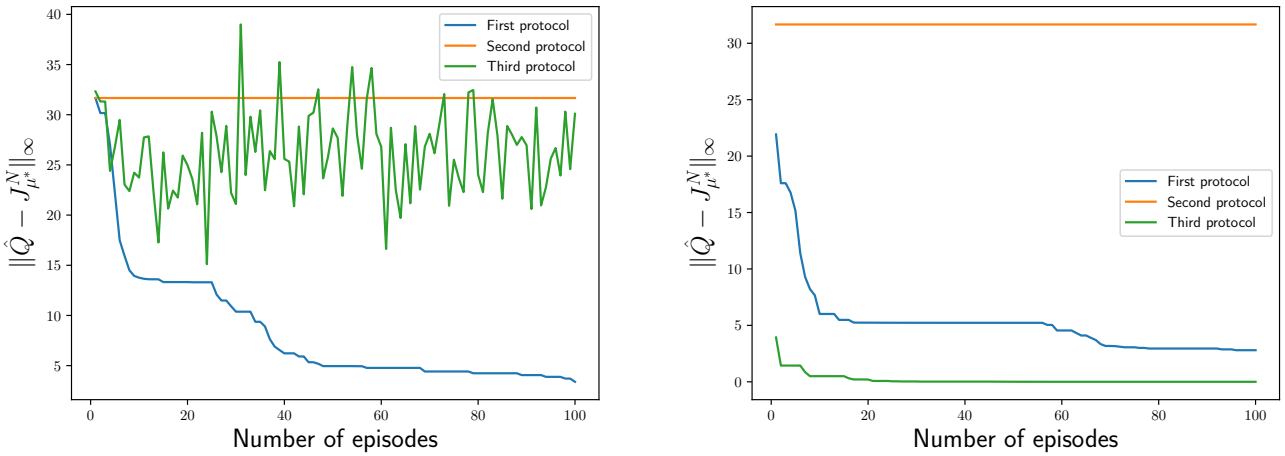
In the stochastic domain, the first strategy, depicted on FIGURE 5.5a shows that the two first protocols do not converge at all as the infinite norm difference is quite constant and its value is about 172, which is in fact the maximal expected return of the optimal policy $J^N_{\mu^*}$. Surprisingly, the third protocol diverges as the number of episodes increases. Please note that, since no clear and coherent explanation has been found, this could come from an implementation error in the code.

In the second strategy, the plot on FIGURE 5.5b shows that none of the three protocols converge as the number of episodes increases. Indeed, despite a clear decrease of the error for the third protocol, the reached plateau is at about 168.

To conclude, in a stochastic setting, the poor quality of the obtained results could be explained by the fact that the considered time horizon value $T = 1000$ is not large enough to explore all the states of the domain, when starting from (3,0).

5.3 Discount factor

One can rerun the same three experimental protocols as in the previous section, but rather than using a discount factor $\gamma = 0.99$, its value is now set to 0.4. The discount factor determines the relative importance of distant-future rewards with respect to near-future ones for the agent. A factor 0 makes the agent's behavior "opportunistic" by only considering immediate rewards while a factor that approaches 1, as in the previous case, makes the agent strive for a long-term high reward.
For the deterministic domain, the corresponding results can be observed in FIGURE 5.6, again for both above-mentioned strategies.



(a) Convergence for the first strategy

(b) Convergence for the second strategy

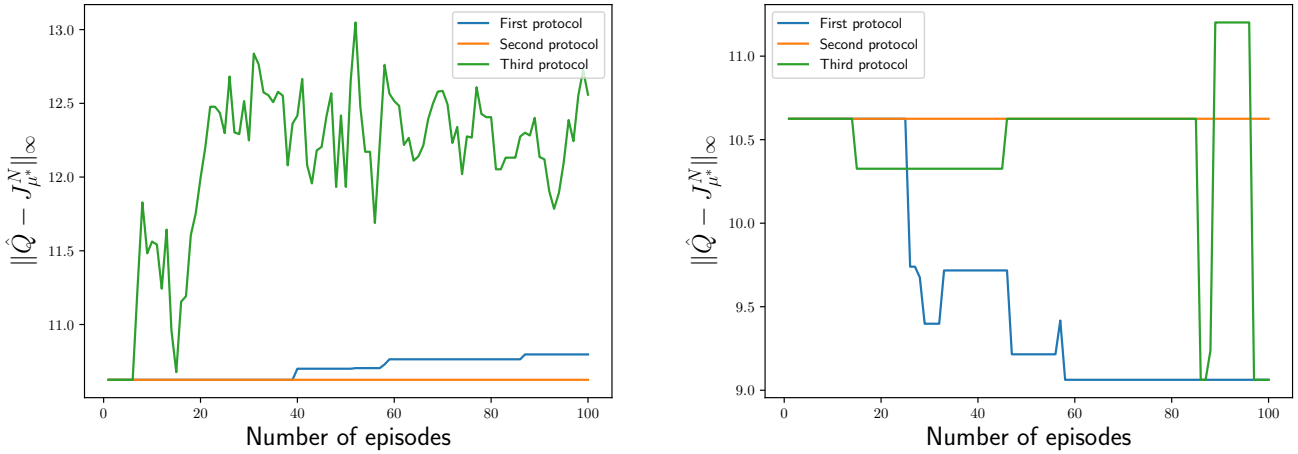
FIGURE 5.6: Convergence of $\hat{Q}$ in the deterministic domain, for $\gamma = 0.4$

Modifying the discount factor γ modifies the optimal policy as well. In particular, a smaller discount factor promotes short-term reward and thus decreases the expected return of the optimal policy $J^N_{\mu^*}$. As a result, when $\hat{Q}$ is initialized to 0, $\|\hat{Q} - J^N_{\mu^*}\|_\infty$ starts at a smaller value.

With the first strategy illustrated in FIGURE 5.6a for the deterministic domain, the first protocol converges toward a value smaller than 5, which is way better than in the preceding section. Once again, the second protocol does not converge, due to the decrease in the learning rate. The third protocol provides a random behavior as the curve is very noisy and oscillates around a value of about 25. Again, this is possibly due to an implementation error regarding the second strategy that provides better-looking results.

The second strategy (FIGURE 5.7b) provides more satisfying results for the simple reason that the third protocol now converges and the value on the vertical axis starts smaller than in the preceding case for both the first and third protocol.

For the stochastic domain, the corresponding results can be observed in FIGURE 5.7, again for both above-mentioned strategies.



(a) Convergence for the first strategy

(b) Convergence for the second strategy

FIGURE 5.7: Convergence of $\hat{Q}$ in the stochastic domain, for $\gamma = 0.4$

As usual, the protocols implemented in the stochastic domain provide very different results. Using the first strategy, FIGURE 5.7a shows that this time, the first and third protocols diverge, while the second protocol is still constant. What is interesting to notice in that case, is that the convergence curve of the infinite norm begins at a smaller value than in the deterministic domain. This means that initially, $\hat{Q}$ is better approximated. This is true due to the randomness of the dynamics in this domain, but not useful as none of the protocol will eventually converge.

Finally, from the second strategy depicted on FIGURE 5.7b, one can deduce the same general observation, that the initial value on the vertical axis is smaller than in the deterministic case. Moreover, the second protocol still generates a straight line curve. However, none of the protocol actually converges, as the amount of decrease of protocols 1 and 3 is not much more than 1.5, in addition to a quite chaotic behaviour of the third protocol.

5.4 Q-Learning with another exploration policy

For this bonus section, a "decay greedy" policy has been implemented[6]. The principle is that the exploration rate ε is multiplied by a decay factor $k \in]0, 1[$ at each transition of an episode. For the previously mentioned first protocol, one can thus tune the parameter k in order to maximize the expected return over a single infinite time horizon. Using the first strategy only, one generates the results of FIGURE 5.8 for the deterministic domain.

[6]Gimelfarb, Sanner, & Lee, 2019; Maroti, 2019

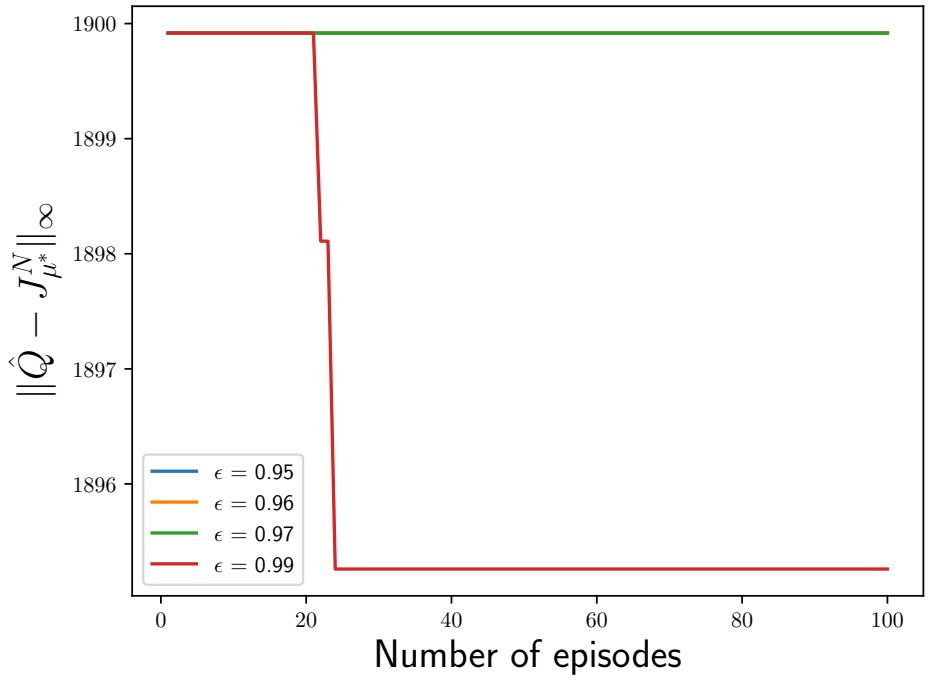


FIGURE 5.8: Evolution of the convergence for different values of k

This figure shows results quite unexpected regarding what can be observed in the literature. As previously mentioned, an implementation error is certainly present in the code, which leads to this bad-looking results. From these, one could however notice that the convergence is way better for $k = 0.99$ than for other values. At the light of the exploration/exploitation trade-off, this could be explained by the fact that, for k smaller than a certain threshold, the decay of the exploration rate ε is too fast, leading to too much exploitation to the detriment of exploration. Hence, for too small values of k, the reinforcement algorithm does not benefit from exploration enough and the exploitation of the corresponding derived policy leads to poor results.
Please note that the stochastic domain has not been investigated. Indeed, as one could not tell much from previous stochastic results, it has been decided that it was not worth it. Again, this could be partially due to an implementation error.