Faculty of Applied Sciences

Optimal Decision Making for complex problems

INFO8003-1

Assignment 1 : Reinforcement Learning in a Discrete Domain

Authors :
Delcour Florian - s181063
Makedonsky Aliocha - s171197

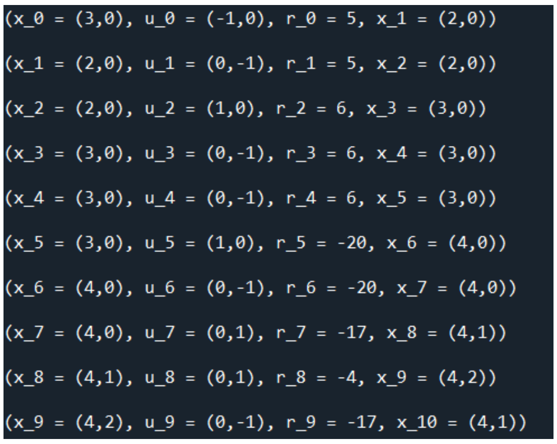
Instructors :
Ernst Damien

Teaching assistants :
Aittahar Samy
Miftari Bardhyl

February 2022

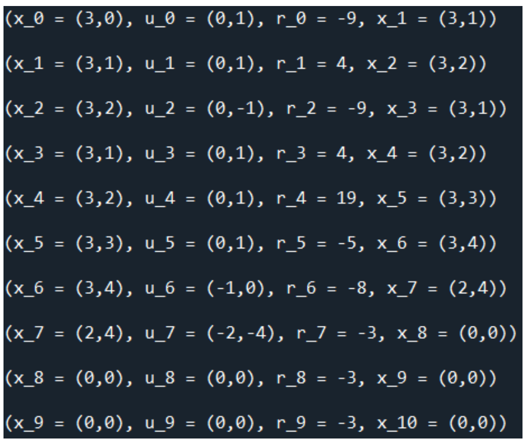
1 Implementation of the domain

The domain is represented by a grid/matrix where each cell contains a reward that is given to the agent if it reaches this cell. We created an array of four tuples containing the four possible actions : go up, go down, go left, go right. The actual state of the agent is described by its position `(x,y)`. The policy we implemented is a random policy, so the agent picks a random direction at each step. If it goes against a bound of the grid, it remains in the current cell and collects again the reward. Figure 1 and 2 are examples of trajectories in deterministic and stochastic domains. The initial state corresponds to the cell x_0`=(3,0)`. In stochastic domain, you can see that the agent moves to state x_8`=(0,0)` performing action u_7`=(-2,-4)` from state x_7`=(2,4)`. It corresponds to the case where $w > 1/2$. In fact, in the stochastic domain, there is a 50/50 chance that the agent moves to state `(0,0)` because w follows a uniform distribution between 0 and 1 and if $w > 1/2$, the agent jumps to state `(0,0)`, otherwise it follows the deterministic dynamics.



```
(x_0 = (3,0), u_0 = (-1,0), r_0 = 5, x_1 = (2,0))
(x_1 = (2,0), u_1 = (0,-1), r_1 = 5, x_2 = (2,0))
(x_2 = (2,0), u_2 = (1,0), r_2 = 6, x_3 = (3,0))
(x_3 = (3,0), u_3 = (0,-1), r_3 = 6, x_4 = (3,0))
(x_4 = (3,0), u_4 = (0,-1), r_4 = 6, x_5 = (3,0))
(x_5 = (3,0), u_5 = (1,0), r_5 = -20, x_6 = (4,0))
(x_6 = (4,0), u_6 = (0,-1), r_6 = -20, x_7 = (4,0))
(x_7 = (4,0), u_7 = (0,1), r_7 = -17, x_8 = (4,1))
(x_8 = (4,1), u_8 = (0,1), r_8 = -4, x_9 = (4,2))
(x_9 = (4,2), u_9 = (0,-1), r_9 = -17, x_10 = (4,1))
```

Figure 1: Deterministic trajectory of 10 steps, random policy



```
(x_0 = (3,0), u_0 = (0,1), r_0 = -9, x_1 = (3,1))
(x_1 = (3,1), u_1 = (0,1), r_1 = 4, x_2 = (3,2))
(x_2 = (3,2), u_2 = (0,-1), r_2 = -9, x_3 = (3,1))
(x_3 = (3,1), u_3 = (0,1), r_3 = 4, x_4 = (3,2))
(x_4 = (3,2), u_4 = (0,1), r_4 = 19, x_5 = (3,3))
(x_5 = (3,3), u_5 = (0,1), r_5 = -5, x_6 = (3,4))
(x_6 = (3,4), u_6 = (-1,0), r_6 = -8, x_7 = (2,4))
(x_7 = (2,4), u_7 = (-2,-4), r_7 = -3, x_8 = (0,0))
(x_8 = (0,0), u_8 = (0,0), r_8 = -3, x_9 = (0,0))
(x_9 = (0,0), u_9 = (0,0), r_9 = -3, x_10 = (0,0))
```

Figure 2: Stochastic trajectory of 10 steps, random policy

2 Expected return of a policy

We can use dynamic programming as we know the reward signal and the dynamics in the two different domains. Thus, to estimate the expected return of a stationary policy J^μ, we can define J^μ_N using the following recurrence equation :

$$J^\mu_N(x) = \underset{w \sim P_w(\cdot|x,u)}{E}\left[r(x,\mu(x),w) + \gamma J^\mu_{N-1}(f(x,\mu(x),w))\right], \quad \forall N \geq 1$$

The latter is for a stochastic domain but is simplified in a deterministic domain :

$$J^\mu_N(x) = r(x,\mu(x)) + \gamma J^\mu_{N-1}(f(x,\mu(x))), \quad \forall N \geq 1$$

with $J_0^\mu(x) \equiv 0$ for both cases. The value of J_N^μ is computed for every possible initial state x of the domain.

In the stochastic case, the expected reward signal can be decomposed in two terms of equal probability 0.5 because the agent has one chance out of two to follow the deterministic policy μ (first term) or to jump to state `(0,0)` (second term) :

$$\underset{w \sim P_w(\cdot|x,u)}{E} \left[r\left(x_t, \mu\left(x_t\right), w\right) \right] = 0.5 * r(x_{t+1}) + 0.5 * r((0,0))$$

where x_{t+1} is the deterministic resulting state of taking an action given by policy μ from state x_t. In a similar way, we have :

$$\underset{w \sim P_w(\cdot|x,u)}{E} \left[\gamma J_{N-1}^\mu(f(x, \mu(x), w)) \right] = 0.5 * \gamma J_{N-1}^\mu(x_{t+1}) + 0.5 * \gamma J_{N-1}^\mu((0,0))$$

The last thing to do is to choose a relevant value of N because if N is too small, the estimation J_N^μ will be far from the real value J^μ while a too large value of N will imply a too long computation time. The choice of N can be made according to :

$$\left\| J_N^\mu - J^\mu \right\|_\infty \leq \frac{\gamma^N}{1-\gamma} B_r$$

where $B_r = 19$ is the largest reward in the domain. We see that the difference between J_N^μ and J^μ is bounded by a factor that is proportional to γ^N which is a really strong property because $\gamma \leq 1$, than the bound will rapidly tends to 0 when N increases. Table 1 provides pairs of N and its corresponding bound such that we see that choosing a value of $N = 5000$ is large enough to have a good estimate of J^μ. The corresponding computation time is also small.

N	Bound
10	1718.33
100	695.46
1000	0.082
5000	$2.85 \cdot 10^{-19}$
10 000	$4.27 \cdot 10^{-41}$

Table 1: Bound value for different values of N

Table 2 and 3 shows the results of $J_N^\mu(x)$ after $N = 5000$ iterations with the random policy.

103.3	110.7	208.9	104.7	83.7
134.2	123.3	199	126.3	83.7
141.7	134.2	-63.9	135.2	74.7
-89.8	-63.9	-1.9	122.3	4.7
142.7	91.3	92.3	4.7	4.7

Table 2: Expected return for deterministic domain, random policy

-133.4	-138	-133.4	-136.8	-118.7
-131.2	-133.4	-135	-128.9	-118.7
-130.9	-135	-137.9	-132.9	-141
-138.3	-130.9	-139.2	-137.7	-137.7
-130.9	-138.3	-129.2	-130.2	130.2

Table 3: Expected return for stochastic domain, random policy

We can see that the results are quite close in Table 3 because the agent has one chance out of two to jump in state `(0,0)`. So whatever the initial state, the expected return is similar.

3 Optimal policy

To be able to compute the Q_N-functions, we first need to determine the equivalent MDP of the domain. To do so, we use the transition probabilities and the reward function :

$$r(x,u) = \underset{w \sim P_w(\cdot|x,u)}{E}[r(x,u,w)] \quad \forall x \in X, \forall u \in U$$
$$p\left(x' \mid x,u\right) = \underset{w \sim P_w(\cdot|x,u)}{E}\left[I_{x'=f(x,u,w)}\right] \quad \forall x, x' \in X, \forall u \in U$$

where :

$$I_{\{logical\_expression\}} = \begin{cases} 1 & \text{if } logical\_expression = true \\ 0 & \text{otherwise} \end{cases}$$

These equations are rewritten for the deterministic domain :

$$r(x,u) = r(x,u) \quad \forall x \in X, \forall u \in U$$
$$p\left(x' \mid x,u\right) = I_{x'=F(x,u)} \quad \forall x, x' \in X, \forall u \in U$$

and for the stochastic domain :

$$r(x,u) = 0.5 * r(x,u) + 0.5 * r((0,0),(0,0)) \quad \forall x \in X, \forall u \in U$$
$$p\left(x' \mid x,u\right) = 0.5 * I_{x'=F(x,u)} + 0.5 * I_{x'=(0,0)} \quad \forall x, x' \in X, \forall u \in U$$

Indeed, in the deterministic case, the agent always collects the reward associated to the state that it has reached. Also the probability of reaching state x' when you are in state x and doing action u will always be equal to either 1, either 0 because there is only 1 possible output state x' resulting from performing action u in state x.

For the stochastic case, the agent collects the reward as in the deterministic case with a probability of only 50% because it has also a probability of 50% to jump to state `(0,0)` and to collect the associated reward. For the transition probabilities, they are the same as for the deterministic case but we divide them all by a factor 2, except the transition $p\left((0,0)\ |(0,0), u = \{up, left\}\right)$ which remains equal to 1.

The Q_N functions are computed recursively using the following recurrence :

$$Q_N(x,u) = r(x,u) + \gamma \sum_{x' \in X} p\left(x' \mid x, u\right) \max_{u' \in U} Q_{N-1}\left(x', u'\right) \quad \forall N \geq 1$$

with $Q_0(x,u) = 0$. After having computed the Q_N-functions, we can determine the policy μ^*, which is the policy that indicates the move to make to get the immediate best Q_N (which is computed over N moves). We determined that for the minimal N such that the policy doesn't change by incrementing N is 7 for both deterministic and stochastic domains.
With those values for N, we obtain the following policy grids (Table 4 for the deterministic domain, and Table 5 for the stochastic domain), which is a grid indicating which move to make in each state.

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

Table 4: Optimal policy in the deterministic domain

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

Table 5: Optimal policy in the stochastic domain

For the deterministic domain, we see that the policy forces to reach the $(0,4)$ cell which has the highest reward (19), and is the only one with a reward of 19 that allows to stay on this same cell even when making a move (going up or right won't change the state since the agent hits a bound and so stays in the same cell). The path followed to reach it is not always the quickest (the agent goes left when in the cell $(4,4)$ instead of going up), because it favors the path allowing to accumulate the highest rewards possible.
For the stochastic domain it's much harder to interpret, the agent seems to favor the immediate reward instead of the long term reward, which is logic since the probability of reaching and staying in the stable state as in the deterministic domain is pretty low.

Table 6 and Table 7 show the expected returns $J_{\mu*}^N$ obtained with those optimal policies.

1842	1857.2	1881	1900	1900
1854.6	1870.3	1881.1	1891	1900
1842	1855.6	1870.3	1881.1	1891
1828.6	1849	1863.6	1863.3	1864.1
1816.3	1826.5	1849	1863.6	1842

Table 6: Expected return for deterministic domain, optimal policy

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.90	167.63	172.13
159.45	160.14	163.05	167.21	167.63
159.26	162.20	167.21	162.20	167.21
159.26	155.71	162.20	167.21	162.23

Table 7: Expected return for stochastic domain, optimal policy

We took the same N as in section 2 to be able to make a proper comparison.
We see that for both domains, we don't get anymore negative expected return, they are all positive and way bigger than in the section 2, since the agent will focus on the best states and so it earns a lot of rewards. The rewards near the stable state we talked about before are globally higher since the agent will reach this state and will stay in it following the optimal policy. It was not the case with our initial random policy.
For the stochastic domain, all the expected returns increase greatly but remain significantly lower than deterministic domain, which may be explained by the fact that the agent frequently chooses the highest possible immediate reward, which grows the expected return, but has a 0.5 probability to go in $(0, 0)$ which is a state with a negative reward.

4 System Identification

We generated trajectories of length of powers of 10, starting by 10 and ending in 10^7 because we tried 10^8 but it couldn't run, the trajectory is way too long to enable computation on it.
In Figure 3 we observe the speed of convergence of the estimated reward to the true reward for the deterministic domain for different length of trajectory, in Figure 4 we observe the same but for the probabilities of transition.
For the deterministic domain we see that the convergences for the probabilities of transition and the rewards are really quick, we only need a trajectory containing 10^3 moves to explore every state and so we obtain an estimated reward and an estimated probability of transition that are equal to their true values, resulting in a zero infinite norm.
For the stochastic domain (Figure 5 and Figure 6), the infinite norm doesn't reach 0 even when increasing the trajectory length, because we may never reach a state which results in having no information on it resulting in a non zero infinite norm. Maybe if we increase

even more the trajectory length we could, with a lucky run, have the expected reward and probability equal to the real ones, but it takes to much computational resources to run so we were unable to do so.

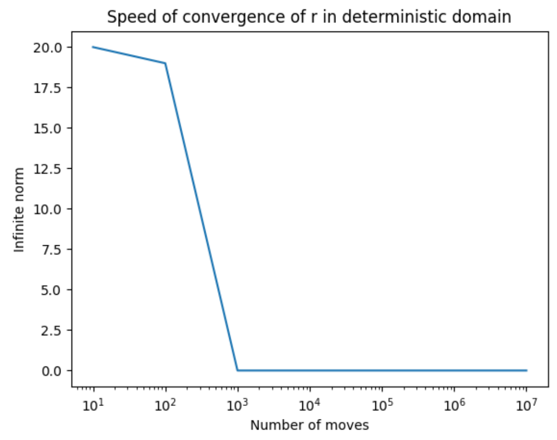


Figure 3: Speed of convergence of r in deterministic domain

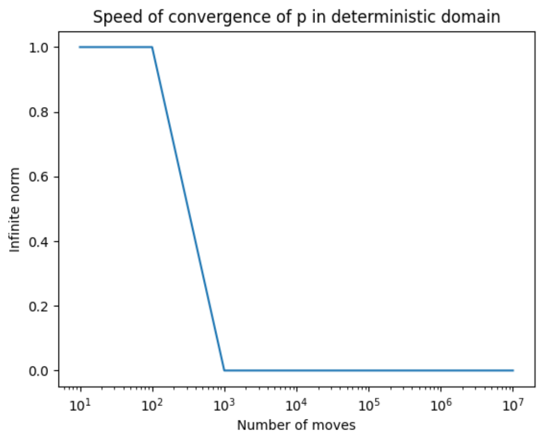


Figure 4: Speed of convergence of p in deterministic domain

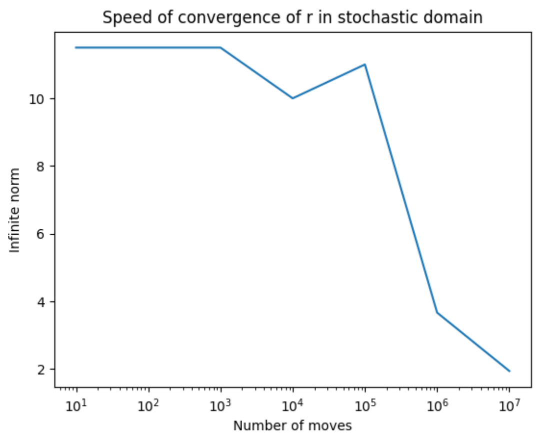


Figure 5: Speed of convergence of r in stochastic domain

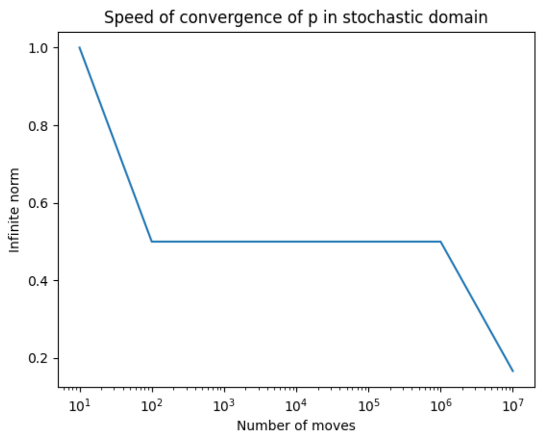


Figure 6: Speed of convergence of p in stochastic domain

Be L the length of the trajectory (the number of moves computed), we have the Table 8 for the infinite norm of the difference between the true Q_N functions and the estimated $\hat{Q_N}$, for the deterministic domain, and Table 9 for the stochastic one.

L=10	L=10^2	L=10^3	L=10^4	L=10^5	L=10^6	L=10^7
129.1	129.1	52.6	52.6	52.6	52.6	52.6

Table 8: $||Q_N$ - $\hat{Q}_N||_\infty$ in the deterministic domain

L=10	L=10^2	L=10^3	L=10^4	L=10^5	L=10^6	L=10^7
38.97	47.18	67.36	40.44	57.79	10.09	9.54

Table 9: $||Q_N$ - $\hat{Q}_N||_\infty$ in the stochastic domain

In Table 10 and Table 11, you can see the estimated optimal policy for deterministic and stochastic domains, these tables are exactly the same as the true ones in Table 4 and 5.

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

Table 10: Estimated optimal policy in the deterministic domain

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

Table 11: Estimated optimal policy in the stochastic domain

The same comment can be made for Table 12 and Table 13 which show the expected returns of the estimated optimal policy. Because the true optimal policy is the same as the estimated one, the expected returns are also the same !

1842	1857.2	1881	1900	1900
1854.6	1870.3	1881.1	1891	1900
1842	1855.6	1870.3	1881.1	1891
1828.6	1849	1863.6	1863.3	1864.1
1816.3	1826.5	1849	1863.6	1842

Table 12: Expected return for deterministic domain, estimated optimal policy

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.90	167.63	172.13
159.45	160.14	163.05	167.21	167.63
159.26	162.20	167.21	162.20	167.21
159.26	155.71	162.20	167.21	162.23

Table 13: Expected return for stochastic domain, estimated optimal policy

5 Q-learning in a batch setting

5.1 Offline Q-learning

For the offline Q-learning we must follow those steps :

1. Initialisation of $\hat{Q}(x,u)$ to 0 everywhere, set k to 0.
2. $\hat{Q}(x_k,u_k) \leftarrow (1-\alpha_k)\hat{Q}(x_k,u_k) + \alpha_k \left(r_k + \gamma \max_{u\in U} \hat{Q}(x_{k+1},u_k)\right)$
3. $k \rightarrow k+1$. If $k = t$, return $\hat{Q}$ and stop. Otherwise go back to step 2.

with t the length of the trajectory.
This algorithm eventually converges if two conditions are met :

1. The learning ratio needs to decrease but not to quickly :

$$\lim_{t\to\infty} \sum_{k=0}^{t-1} \alpha_k \to \infty \qquad \text{and} \qquad \lim_{t\to\infty} \sum_{k=0}^{t-1} \alpha_k^2 < \infty$$

2. When the length of the trajectory tends to infinity, every state-action pair needs to be visited an infinite number of times.

In such conditions, we have $\hat{Q} \to Q$ when $t \to \infty$.

Following this algorithm, with a trajectory of time horizon $T = 10^7$ (as we saw in the previous section, this length allows to have a convergence in both the stochastic and the deterministic domains, and the computations are not too long with this length, so it seems to be an acceptable choice), for the deterministic domain, we obtain the optimal policy shown in Table 14, and this optimal policy is exactly the same than the one we found in the section 3.

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

Table 14: Estimated optimal policy from Q-learning, deterministic domain

And in Table 15 we observe, for the deterministic domain, the expected return with our policy which is obviously the same one than in section 3 since the optimal policy is the same.

1842	1857.2	1881	1900	1900
1854.6	1870.3	1881.1	1891	1900
1842	1855.6	1870.3	1881.1	1891
1828.6	1849	1863.6	1863.3	1864.1
1816.3	1826.5	1849	1863.6	1842

Table 15: Expected return for deterministic domain, estimated optimal policy

For the stochastic domain, we obtain the optimal policy shown in Table 16, and this optimal policy is nearly the same than the one we found in the section 3. There are only two different moves, however since there is some randomness involved, the results may change when running the code again.

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

Table 16: Estimated optimal policy from Q-learning, stochastic domain

In Table 17 we observe, for the stochastic domain, that the expected return with our policy is nearly the same but with small differences since the optimal policy is a bit different too.

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.90	167.63	172.13
159.45	159.71	162.20	167.21	167.63
159.26	162.20	167.21	162.20	167.21
159.26	155.71	162.20	167.21	158.70

Table 17: Expected return for stochastic domain, estimated optimal policy

Those differences for the stochastic domain are explained by the fact that even with $T = 10^7$ the approximation of $\hat{r}$, $\hat{p}$, $\hat{Q}$ is not perfect as we observed in the section 4, but it's still really close from the true ones.

The offline Q-learning is thus pretty good since we get results close to the true values.

5.2 Online Q-learning

In Figure 7 we observe the evolution of $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ after each of the 100 episodes, with a constant learning rate, and no replay for the deterministic domain, and in Figure 8 for the stochastic domain. In the deterministic domain we observe that after approximately 23 episodes, the infinite norm reaches 0, which means that we have a total convergence of $J^N_{\mu_{\hat{Q}}}$ to $J^N_{\mu^*}$.
For the stochastic domain even though for the first episodes the infinite norm is quite small, it increases after. This may be caused by the randomness of the procedure, which doesn't allow to compute a policy that would be equal to the policy of the stochastic case because we doesn't visit enough the different states so we can compute correctly $\hat{Q}$ and thus neither the optimal policy.

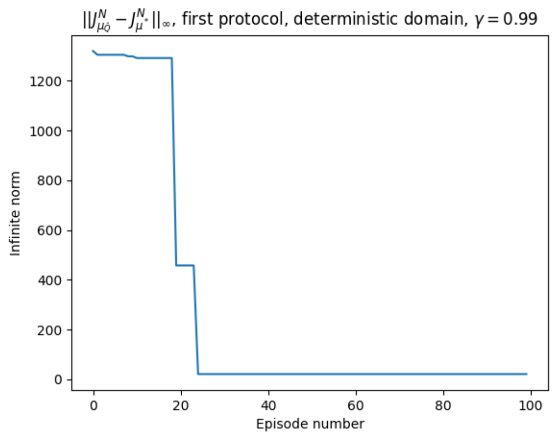


Figure 7: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the deterministic domain, first protocol

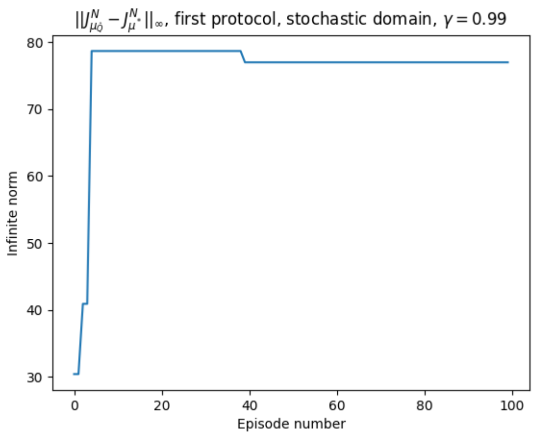


Figure 8: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the stochastic domain, first protocol

Considering the second protocol without replay, we observe the evolution of $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ after each of the 100 episodes, with a varying learning rate $(\alpha_t = 0.8 \cdot \alpha_{t-1},\ \forall t > 0)$. The results are shown in Figure 9 for deterministic domain and in Figure 10 for the stochastic domain. For both domains we are far from convergence, indeed the graphs look like there is a convergence but a plateau is reached far from 0, which means that the $J^N_{\mu_{\hat{Q}}}$ is still not close to $J^N_{\mu^*}$. This difference is due to the the small learning rate that blocks the agent from learning the true value of Q and thus the true optimal policy.

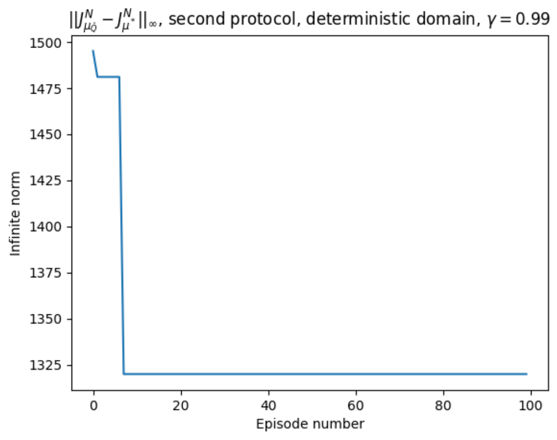


Figure 9: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the deterministic domain, second protocol

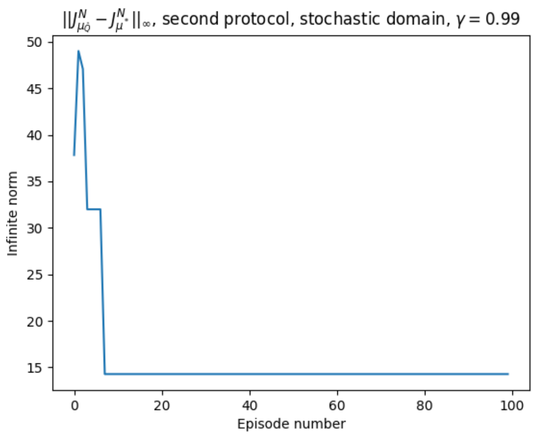


Figure 10: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the stochastic domain, second protocol

Considering the third protocol, we observe the evolution of $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ after each of the 100 episodes, with a constant learning rate, and with replay (randomly take 10 actions among the list containing one more new action at each transition, and learn Q with those 10 actions at each transition), the results are shown in Figure 11 for the deterministic domain and in Figure 12 for the stochastic domain. We observe that the convergence of $J^N_{\mu_{\hat{Q}}}$ to $J^N_{\mu^*}$ is way quicker with this protocol, which is logic since the Q is learned over 10 times more transitions, so the infinite norm between those 2 terms falls to 0 after 1 episode, then increases a little bit before decreasing again until the end. For the stochastic domain, we have varying results, which may be caused by some really good episodes that would allow convergence, and after bad episodes that would spoil the good results we obtained, which explain those variations. But after some episodes (approximately 41), this randomness is countered by the high number of transitions and we reach a total convergence.

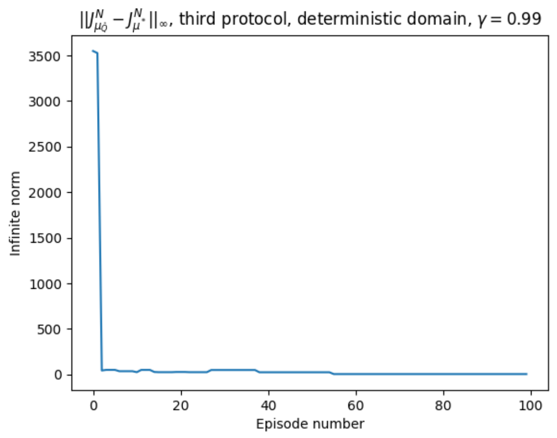


Figure 11: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the deterministic domain, third protocol

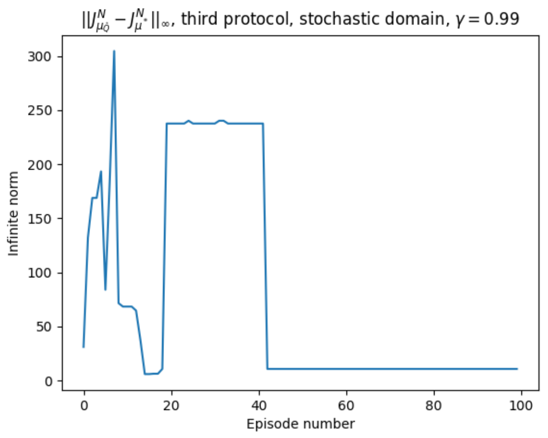


Figure 12: $||J^N_{\mu_{\hat{Q}}} - J^N_{\mu^*}||_\infty$ in the stochastic domain, third protocol

5.3 Discount factor

The effect of taking a lower discount factor is that the influence of the next move that the agent could make is reduced, the immediate reward is more important here, which will obviously change the policy in a more short-term policy.
Figure 13 shows the results for the deterministic domain and Figure 14 for the stochastic domain, using the first protocol. For the deterministic domain we observe a convergence, not total but really close too. The convergence takes more time than for previous subsection, which may be explained by the fact that, because of the lower discount factor, the closer rewards get more importance than before. And since here we have a part of randomness, it brings us bad rewards and it takes more transitions to counter this randomness.
In the case of stochastic domain, there is no convergence at all.

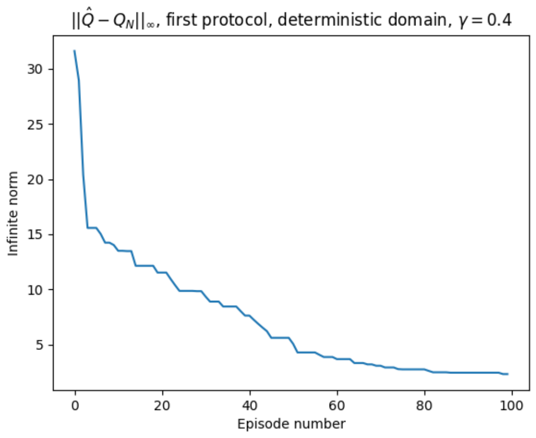


Figure 13: $||\hat{Q} - Q_N||_\infty$ in the deterministic domain, first protocol

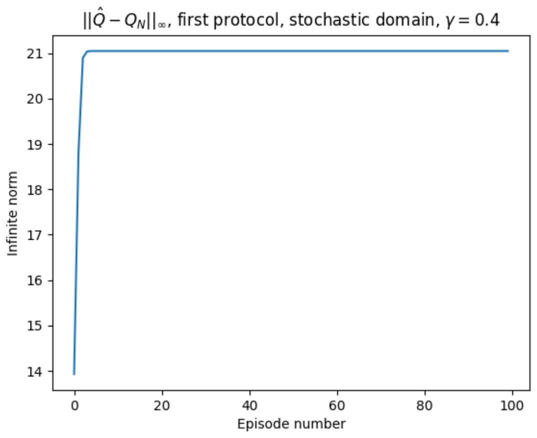


Figure 14: $||\hat{Q} - Q_N||_\infty$ in the stochastic domain, first protocol

Figure 15 displays the results for the deterministic domain and Figure 16 for the stochastic domain, using the second protocol. For both domains, we could say that we have a convergence seeing the shape of the curves but it's wrong ! Indeed looking on the y-axis we quickly see that the infinite norm is near from not moving at all, which is explained by the fact that the small learning rate blocks the convergence since the agent quickly stops learning, the value of $\hat{Q}$ will only depend on its actual value and there won't be improvement.

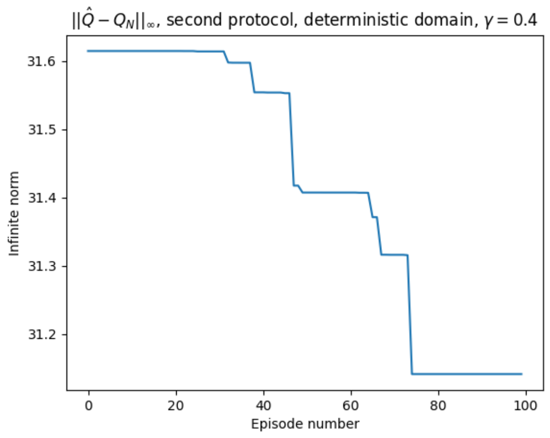


Figure 15: $||\hat{Q} - Q_N||_\infty$ in the deterministic domain, second protocol

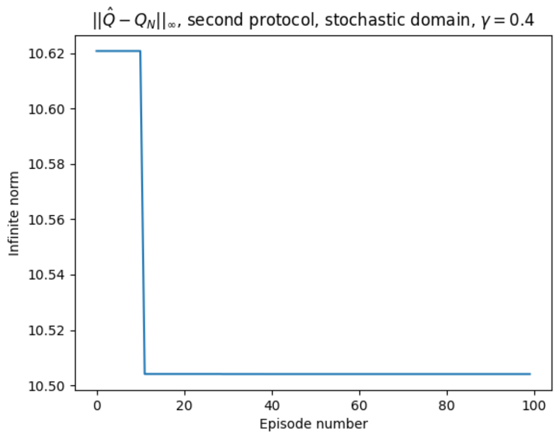
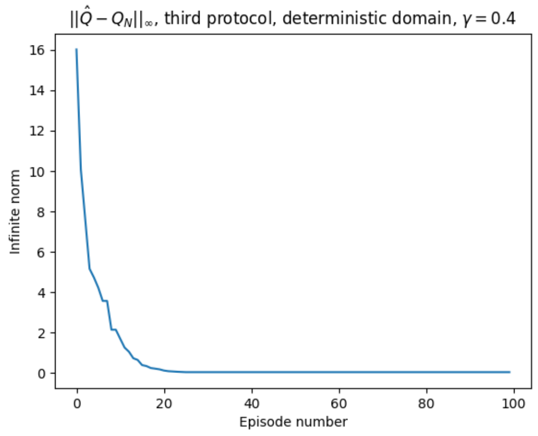


Figure 16: $||\hat{Q} - Q_N||_\infty$ in the stochastic domain, second protocol

For the third protocol, the convergence is really quick, and total since the infinite norm nearly instantly falls close to 0. It's the same justification than for the third protocol in section 5.2, the fact of having a lot more moves makes the convergence quicker. For the stochastic domain there is no convergence from $\hat{Q}$ to the true Q, unlike the convergence observed for the third protocol in the previous section. The short term policy caused by the lower discount factor, may cause the non-convergence here, even if there is a lot of transitions.



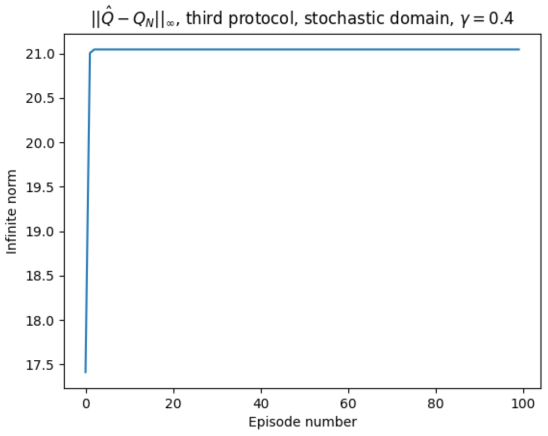


Figure 17: $||\hat{Q} - Q_N||_\infty$ in the deterministic domain, third protocol

Figure 18: $||\hat{Q} - Q_N||_\infty$ in the stochastic domain, third protocol

5.4 Q-learning with another exploration policy

The other exploration policy we chose is the softmax policy, that chooses an action at random among the possible actions, but with an higher probability for the actions with a higher $Q(x, u)$. The probability of taking the action u is given by :

$$P(u|x) = \frac{exp\left(\dfrac{Q(x,u)}{T}\right)}{\sum_{u' \in U} exp\left(\dfrac{Q(x,u')}{T}\right)}$$

T is a parameter influencing the exploration/exploitation tradeoff, higher is T, more the exploration is important since the $\dfrac{Q(x,u)}{T}$ and $\dfrac{Q(x,u')}{T}$ will tend to 0. Thus the exponentials will tend to 1 and we will have an equal probability to take any of the actions so it's exploration. Smaller is the T, more the agent exploits the $\hat{Q}$ it has already computed.

In Figure 19 we can observe the sum of all the expected cumulative rewards for different values of T for the deterministic domain and in Figure 20 for the stochastic domain. The curves are all really similar and reaches the same level at the end. The differences we can make is that for the biggest T, the agent takes more episodes to reach the same expected cumulative return than for the other T, since the agent explore more it has less time to exploit the results it found. We also observe that for the smallest T, the expected cumulative reward reached is smaller than for the other T, which may be caused by the fact that the agent didn't explore enough.
For the stochastic domain, we can't infer anything from the other curves. Indeed, those

are totally unstable because a lot of randomness is involved since there is randomness for exploring/exploiting the rewards, and randomness for the agent to move in the space, so the results are really random. In addition, the exploitation is hardly possible since there is a 0.5 probability of going to (0;0) so the agent cannot keep exploiting the best states. Only the results for T = 100 become stable after a short number of episodes, but they are really bad since the agent probably doesn't reach the cells with high rewards.

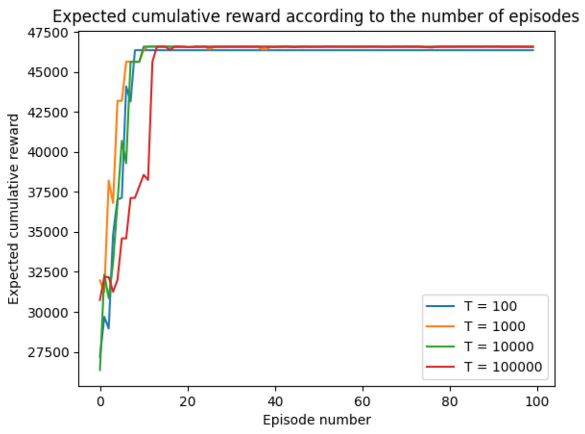


Figure 19: Expected cumulative reward, deterministic domain, softmax

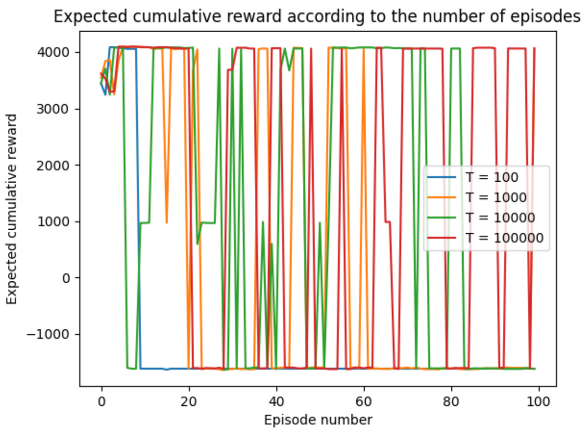


Figure 20: Expected cumulative reward, stochastic domain, softmax