Reinforcement Learning - Homework 1

Mehdi Boubnan

November 12, 2018

1 Dynamic Programming

1.1 Q1 : MDP Model

Implementing the MDP model (see code)

We notice that taking action a_2 at state s_2 has the biggest reward, and always yield to stay at the state s_2. So the best strategy is to attain that state and loop over taking action a_2 to have the biggest reward. The optimal policy here is therefore simple to guess :

$$\pi^* = [a_1, a_1, a_2]$$

1.2 Q2 : Value iteration

The stopping criterion for the value iteration is to stop when

$$\|v^{k+1} - v^k\|_\infty < \alpha \quad \text{which implies} \quad \|v^{\pi^{k+1}} - v^*\|_\infty < \frac{2\alpha\gamma}{1-\gamma}$$

We want a 0.01 optimal policy, in other words we want

$$\|v^{\pi^{k+1}} - v^*\|_\infty < 0.01$$

So we'll take the stoping criterion :

$$\alpha = \frac{0.01(1-\gamma)}{2\gamma}$$

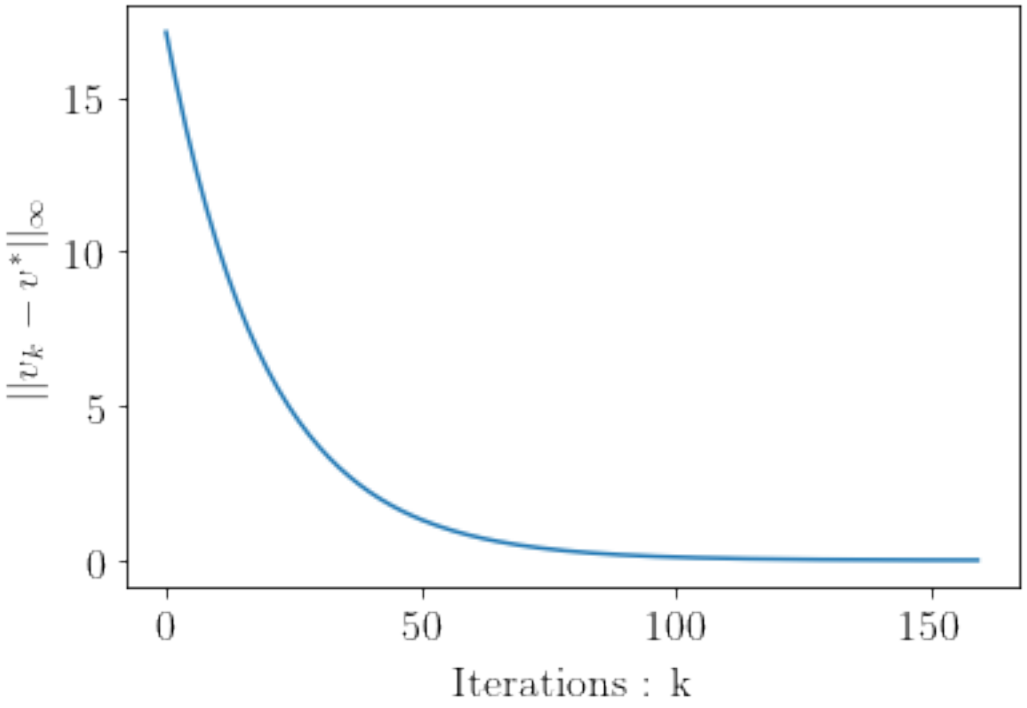
Running Value Iteration :

```
Number of iterations : 159
Greedy Policy found : [1 1 2]
```

Compute the optimal policy using the greedy policy :

```
V* : [15.39115723 16.5483871  18.        ]
```

Plotting the inifinity norm evolution along iterations



1.3 Q3 : Policy iteration

Running Policy Iteration :

```
Number of iterations : 4
Greedy Policy found : [1 1 2]
```

1.4 Comparison between VI and PI

```
Value iteration algorithm time execution on average :
2.55 ms ± 371 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
Policy iteration algorithm time execution on average :
177 µs ± 2.05 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

We notice that the **value iteration algorithm** converges to the optimal policy in **2.32ms for a total of 160 iterations.**
The **policy iteration algorithm** converges to the same optimal policy in **187μs for a total of 4 iterations.**

The policy iteration algorithm is much faster than the value iteration algorithm, despite the fact that its time complexity is much greater than the VI time complexity. In fact, since the MDP has a small number of possible actions, it admits a small finite number of policies which explains that PI is faster than VI despite the gap in computation complexity.

2 Reinforcement Learning

2.1 Q4 : Monte-Carlo Estimator

2.1.1 Setting Parameters :

We'll choose the discount factor $\gamma = 0.95$, and T_{max} the length of one episode as :

$$T_{max} = -\frac{log(\delta / R_{max})}{1-\gamma}$$

with $R_{max} = 1$ the maximum reward, and $\delta = 1e-3$.

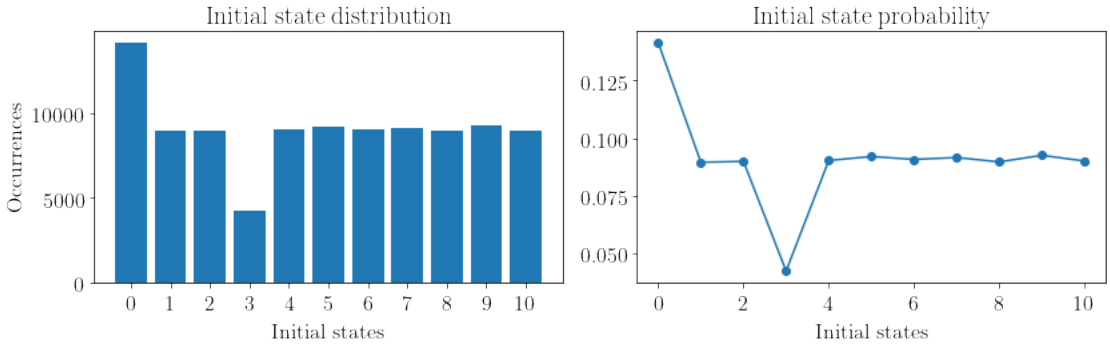
We then have $T_{max} \approx 139$

2.1.2 Building the policy :

The policy to consider here is to take right if available and up otherwise :

2.1.3 Estimating the initial state distribution :

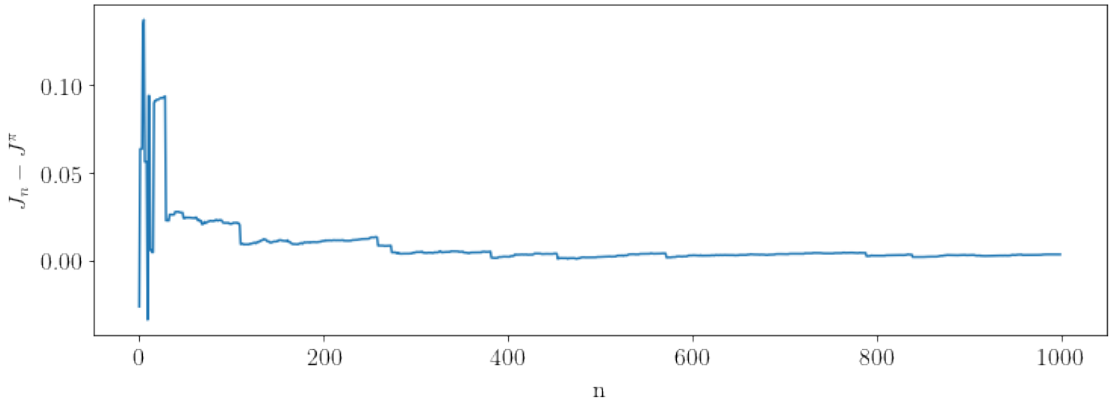
Running 100000 resets to estimate the distribution :



2.1.4 Monte Carlo Estimator :

Running the Monte Carlo algorithm for 1000 trajectories :
    See code
Computing J_n for every V_n for n in $\{1, 2, ..., 1000\}$:
    See code
Computing J^{π} :
    See code

Ploting $J_n - J^\pi$ for n in $\{1, 2, ..., 1000\}$:



2.2 Q5 : Q-Learning

2.2.1 Describing the Q-Learning algorithm parameters

We'll choose a learning rate defined as follows :

$$\alpha_i(x_t, a_t) = \frac{1}{\text{visits}(x_t, a_t)}$$

The policy exploration is chosen as follows :

For a state x_t, we'll define the following actions :

- **Action 1** : $a_t = argmax_a Q(x_t, a)$ with$a \in \mathcal{A}(x_t)$ (possible actions at state x_t)
- **Action 2** : a_t = Random action from $\mathcal{A}(x_t)$

The policy will take the action 1 with a probability $1 - \epsilon$, and action 2 with a probability ϵ. Therefore :

$$\pi(x_t) = \theta * action_1 + (1 - \theta) * action_2$$
$$\text{with} \quad \theta \sim \mathcal{B}(1 - \epsilon)$$

We will choose **epsilon = 0.4**

These parameters satisfy the **Robbins-Monro conditions**.

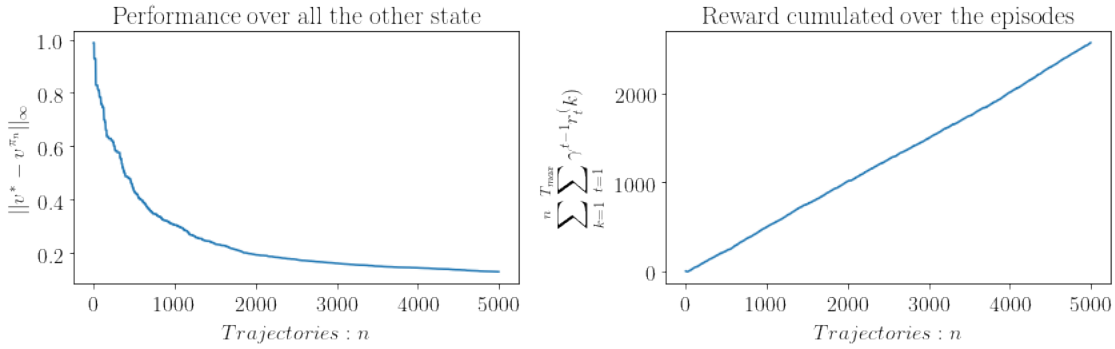
2.2.2 Q-Learning Algorithm

Running the algorithm for 5000 trajectories :

Computing the optimal policy found by the Q-Learning :

```
Q-Learning Optimal Policy :
[0.8452655903460421, 0.9134766054655945, 0.9848977267138465, 0.0, 0.7498047525540733, 0.91722268
---
Real Optimal Policy :
[0.87691855, 0.92820033, 0.98817903, 0.0, 0.82369294, 0.92820033, 0.0, 0.77818504, 0.82369294, 0
```

We notice that the optimal policy found is near to the true optimal policy.
Plotting the evolution :



2.3 Q6 : Initial state distribution effect

The **true optimal policy** is not affected by the change of mu_0, but the **converged optimal policy** found by the algorithm **is affected by the initial state distribution**.

The Robbins-Monro conditions impose that all states have to be visited **infinitely often** in order to converge towards the optimal Q and therefore the optimal value function. The **exploration** of states in then **crucial**.

- If the MDP **doesn't explore** well enough all the states, the algorithm will converge to the optimal policy only if the initial state distribution is "large" in order to start from all the states.
- On the other hand, if the MDP **explores well** all the states, if we take an initial distribution that only starts from an absorbing state for example, the algorithm won't converge to the optimal policy.