Homework 2: MDP & DP & MC

In this problem set you will study Markov decision processes (MDP). The exercises are taken from or inspired by the book of Sutton and Barto.

Instructions. You are to submit a PDF for your answers and a Python notebook for your code. For the PDF, you can type your answers for example using latex or use handwritten notes as long as they are **clearly and easily readable**. Do not forget to **fully justify** your answers. You notebook's name should follow the format: `HW2_LastName_FirstName`.

Exercise 1 (Programming)**.** Consider a 4×4 gridworld. The state space is $S = \{1, 2, \dots, 16\}$ (it can be interpreted using 2D coordinates), and the action space is $A = \{\text{up}, \text{down}, \text{right}, \text{left}\}$ (there is no option to stay still). The moves are deterministic: $S_{t+1} = S_t + A_t$, except that:

- The moves that would take the agent off the grid do not modify the state (the agent stays at the same state); for instance $p(4|4, \text{right}) = 1$, $p(4|4, \text{up}) = 1$, $p(5|5, \text{left}) = 1$ and $p(2|2, \text{up}) = 1$. Remember that $p(s'|s, a)$ denotes the probability to go, in one step, from state s to state s' when using action a.

- The states 1 or state 10 are terminal in the sense that the episode stops upon reaching one of these states (the agent cannot move anymore and cannot accumulate rewards).

For each transition (even those for which the agent takes an action without changing the state because of the gridworld's border), the agent gets a reward of -1.

We consider the undiscounted, episodic task, in which the goal is to find a policy π which maximizes the expected sum of future rewards until reaching a terminal state:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\tau-1} r(S_t, A_t)\right]$$

where τ is the first time that a terminal state is reached, $S_\tau \in \{1, 10\}$. Note that this is an episodic task in which the terminal time is not fixed a priori using a terminal horizon T. Instead, the episode stops when a terminal state is reached (which could take more or less time depending on each trajectory). In particular, some policies may lead to short episodes while some policies may lead to longer (or even infinite) episodes.

1. Propose a policy π and a state s for which $V^\pi(s) = -\infty$.

2. Assume π is the uniform policy (or "equiprobable random policy"), i.e., for every s, for each $a \in A$, $\pi(a|s) = \frac{1}{|A|}$. What is $Q^\pi(9, \text{right})$? (Bonus: What is $Q^\pi(12, \text{left})$?)

3. Write a version of the (pseudo-code for) policy iteration algorithm for this undiscounted, episodic task. The policy evaluation phase must be designed to compute the V^π function described above. Notice that the rewards are not time-dependent and there is no fixed terminal time so we still use stationary policies (i.e., policies of the form $\pi(a|s)$).

4. What is the optimal value function? You can represent it by drawing a 4×4 grid and writing the value of each non-terminal state in the corresponding cell. Give an optimal policy by saying which action to choose in each state.

Note: You may find Example 4.1 and Figure 4.1 in Sutton and Barto's book useful. However, notice that the problem is not exactly the same.

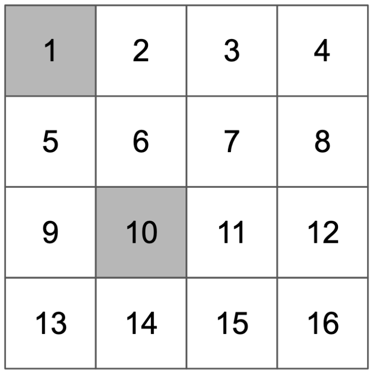

Figure 1: Inspired from example 4.1 from Sutton and Barto's book .

Exercise 2 (Programming)**.** Consider Jack's rental problem in Sutton and Barto's book (Example 4.2). In a Python notebook, implement policy iteration and value iteration to solve this problem. Plot the value function computed by each algorithm and check that they match. It should look like the one described in Figure 4.2 of Sutton and Barto's book. How many iteration does each algorithm take to converge? How about the computational time?

Exercise 3. In class, we have discussed policy iteration and value iteration for the (state) value functions, V^π and V^* respectively. Provide the pseudo-code for these algorithms but using the state-action value functions, Q^π and Q^* respectively (without using V^π and V^* at all).

Exercise 4 (Programming)**.** For the gridworld problem with 2 terminal states as described previously. Let π be the uniform policy, i.e., the policy which, in each state, gives equal probability over all possible actions. The goal is to evaluate this policy by computing V^π.

(1) First, implement policy evaluation and compute V^π in this way (with initialization 0 for all the values and at each iteration, update the values for all states using exact updates). This policy evaluation algorithm is the same as the one used for Question 3 in Exercise 1, which was one component of the policy iteration algorithm. How many iterations do you need to get convergence?

(2) Implement each of the following algorithms:

1. Every-visit Monte Carlo prediction to estimate V^π where every episode starts with $S_0 = 16$

2. Every-visit Monte Carlo prediction to estimate V^π where every episode starts with S_0 distributed uniformly over non-terminal states (exploring starts)

3. First-visit Monte Carlo prediction to estimate V^π where every episode starts with $S_0 = 16$
4. First-visit Monte Carlo prediction to estimate V^π where every episode starts with S_0 distributed uniformly over non-terminal states (exploring starts)

Let K be a number of episodes (you can take $K = 1000$ for instance but if you think that more episodes are required, you can try larger values). In each case, plot the (state) value function learnt by the algorithm after the K episodes. We want to determine a number of iterations so that the learnt value function is close to the true (optimal) value function. To this end, use V^π computed in the first step of this exercise using policy evaluation. For each algorithm: Let V^k be the value function computed as a result of episode k. Measure $\epsilon^k = \|V^\pi - V^k\|_\infty = \max_{s\in S} |V^\pi(s) - V^k(s)|$. After K episodes, plot $(\epsilon^k)_{k=0,\ldots,K}$ as a function of k. Plot the 4 curves on the same graph. Which algorithm converges faster?

Note: In practice, in RL we do not know the transition probabilities so we cannot compute V^π using exact updates of policy evaluation. So this question is for the sake of illustration.

Exercise 5 (Bonus; Programming)**.** Redo the previous exercise but with estimation of the Q^π function instead of the V^π function. You are still asked to plot the V^π function at the end (you need to modify the algorithm so that it learns the Q^π function, and at the end of the K episodes, you compute the V^π function from the Q^π function).

Exercise 6 (Bonus; Programming)**.** Use the MC algorithms estimating Q^π in Exercise 5 to develop an MC algorithm for policy iteration: the policy improvement step is the same, but the policy evaluation is done using one of the MC methods of Exercise 5.