Assignment 1

Author: **Leonardo Colosi 1799057**

Contributors: **Bruno Francesco Nocera 1863075, Silverio Manganaro 1817504, Paolo Renzi 1887793, Jacopo Tedeschi 1882789, Amine Ahardane 2050689.**

MARR, RL
November 7, 2023

Contents

1 Theory

1.1 Exercise 1

The goal is to derive the formula to get the minimum number of iterations (i) of *Value Iteration* that are needed if we want an error on the quality of the policy that is at most ϵ. To do so we can exploit the inequality:

$$V^{\pi_i}(s) \;\geq\; V^*(s) \;-\; \frac{2\gamma^i}{1-\gamma}||Q_0 - Q^*||_\infty \tag{1}$$

This is a consequence of the facts that the Bellman operator, in the *Value Iteration* algorithm, is a contraction mapping and that the expected value of the cumulative discounted reward is upper-bounded by the factor $\frac{1}{1-\gamma}$ due to the geometric nature of the summation. We can rewrite (1) in terms of the policy value error, defined as the difference between the value of the optimal policy and the value of the approximated one:

$$V^*(s) \;-\; V^{\pi_i}(s) \;\leq\; \frac{2\gamma^i}{1-\gamma}||Q_0 - Q^*||_\infty \tag{2}$$

Now imposing that the expression on the right of (2) is smaller than a given ϵ is equivalent to minimizing the policy value error.

$$\frac{2\gamma^i}{1-\gamma}||Q_0 - Q^*||_\infty \;\leq\; \epsilon \tag{3}$$

Since in the inequality (3) the term i (number of iteration) appears, we can solve for it in order to find the minimum number of algorithmic steps needed to minimize the error under ϵ. The mathematical derivation is the following:

$\frac{2\gamma^i}{1-\gamma}\|\|Q^*\|\|_\infty \leq \epsilon$	As first step we set Q_0 to 0	1
$\frac{2\gamma^i}{1-\gamma} \cdot \frac{1}{1-\gamma} \leq \epsilon$	Property: $\|\|Q^*\|\|_\infty$ is bounded to $\frac{1}{1-\gamma}$	2
$\frac{2(1-(1-\gamma))^i}{(1-\gamma)^2} \leq \epsilon$	Adding and subtracting one	3

The first to steps are a consequence of the fact that the infinity norm $||Q_0 - Q^*||_\infty$ reach it's maximum value when the difference between Q_0 and Q^* is maximum, meaning that one of them is zero and the other the highest possible value: $\frac{1}{1-\gamma}$. In this "worst" case we are still looking for an i that solve the inequality for a fixed ϵ.

The last step just consist in applying a mathematical "trick" that will allow us to define an exponentially decreasing function to set as upper limit to our original function. In particular we will be using the property that $1 + x \leq e^x \; \forall x \in \mathbb{R}$ to define such function.

Following:

$$\frac{2(1-(1-\gamma))^i}{(1-\gamma)^2} \leq \frac{2e^{-(1-\gamma)i}}{(1-\gamma)^2} \leq \epsilon \qquad \text{Property: } 1-(1-\gamma) \leq e^{-(1-\gamma)} \qquad 4$$

$$e^{-(1-\gamma)i} \leq \frac{\epsilon(1-\gamma)^2}{2} \qquad 5$$

$$-i(1-\gamma) \leq -\log(\frac{2}{\epsilon(1-\gamma)^2}) \qquad \text{Property: } log(x) = -log\left(\frac{1}{x}\right) \qquad 6$$

Finally getting an explicit expression to find the minimum number of iterations necessary to bring the error on the quality of the policy to ϵ:

$$i \geq \frac{\log \frac{2}{\epsilon(1-\gamma)^2}}{1-\gamma} \tag{4}$$

The property used at step (4) is a consequence of a first order Taylor expansion of the exponential function. For this reason, while convergence is guaranteed, we can also observe that the new function will decrease more slowly compared to our original function, meaning that we are making some "pessimistic" approximation on the actual number of step needed for some given ϵ and γ. This is shown graphically here.

1.2 Exercise 2

We have a set of seven states: $\{S_1, ..., S_7\}$ and a policy: $\pi(s) = a \;\; \forall s$.
The reward function is defined as:

$$r(s,a) = \begin{cases} 1/2 & \text{if } s = s_1 \\ 5 & \text{if } s = s_7 \\ 0 & \text{otherwise} \end{cases}$$

The transition probability (dynamics of the system) is given for the state s_6 as:

- $P(s_6 \mid s_6,\, a_1) \; = 0.3$
- $P(s_7 \mid s_6,\, a_1) \; = 0.7$

We also have an initialization for the Value Function at $k = 1$:

$$V_k \; = \; [0.5,\, 0,\, 0,\, 0,\, 0,\, 0,\, 5]$$

Setting a fixed value $\gamma = 0.9$, for the discount factor, we can compute $V_{k+1}(s_6)$ following the *Value Iteration* algorithm:

$$V(s_6) = r(s_6) + \gamma \mathbb{E}_{s' \sim P(\bullet | s_6, \pi\{s_6\})} V(s') \qquad 1$$

$$V(s_6) = 0 + 0.9 \cdot [\mathbb{E}_{s' \sim P(\bullet | s_6, \pi\{s_6\})} V(s')] \qquad 2$$

$$V(s_6) = 0 + 0.9 \cdot [P(s_6|s_6, a_1)V(s_6) + P(s_7|s_6, a_1)V(s_7))] \qquad 3$$

$$V(s_6) = 0.9 \cdot [0.3 \cdot V(s_6) + 0.7 \cdot V(s_7))] \qquad 4$$

$$V(s_6) = 0.9 \cdot 0.35 = 3.15 \qquad 5$$

The steps of the computation the computation are:

1. Expansion of the value function as sum of discounted rewards;
2. Substitution of γ and $r(s_6)$ with numerical values;
3. Expansion of the Expected value as a weighted sum, where the weights are transition probabilities;
4. Substitution of probabilities and Value function output with the given data.

2 Code Implementation

2.1 Policy Iteration

In this first part of the practical assignment our goal was to implement the transition probabilities and the reward function of the *FrozenLake Gymnasium* environment according to the guidelines:

- The reward function must return 0 everywhere and 1 for the goal cell;
- The transition probabilities must be such that the agent moves in the right direction with probability 1/3, and instead moves in one of the perpendicular directions with probabilities 1/3 and 1/3.

In addiction to this we could established that if one or more of the three possible direction of movement, the right one and the two perpendicular to that, is unfeasible then the probability of make that movement should be added to the probability of remaining in the same spot.

For example if the chosen action is move to the LEFT then we would have a probability of 1/3 to move on the left, of 1/3 to move up and of 1/3 to move down. But if the block on the left and the one up are both occupied by obstacles then we would have a probability of 1/6 to stay on the spot and a probability of 1/3 to make a move down.

Here is it the code that does that:

```
def transition_probabilities(env, s, a, env_size, directions, obstacles):
    prob_next_state = np.zeros((env_size, env_size))

    for i in [0, 1, -1]:
        s_prime = s + directions[(a+i)%4]
        s_prime = check_feasibility(s_prime, s, env_size, obstacles)
        prob_next_state[s_prime[0], s_prime[1]] += 1/3


    return prob_next_state
```

Because of the "slippery" probability the agent reaches the goal one over three times on average.

2.2 iLQR

This second task was about implementing an *iLQR* controller to keep a mechanical pendulum stable in an horizontal position (a point of non stable equilibrium). Knowing that the pendulum dynamics is given by:

$$\dot{\theta}_{t+1} \; = \; \dot{\theta} \; + \; \left(\frac{3g}{2l} \sin\theta \; + \; \frac{3.0}{ml^2} u \right) \tag{5}$$

$$\theta_{t+1} \; = \; \theta \; + \; \dot{\theta}_{t+1} dt \tag{6}$$

the formulas to compute the P and K matrices could be derived from the *LQR-LTV* algorithm. In particular in the backward function we have:

$$k_t \; = \; -(R_t \; + \; B_t^T P_{t+1} B_t)^{-1} (r_t + B_t^T p_{t+1}) \tag{7}$$

$$K_t = -(R_t \; + \; B_t^T P_{t+1} B_t)^{-1} (B_t^T P_{t+1} A_t) \tag{8}$$

$$p_t \; = \; q_t + K_t^T (R_t k_t + r_t) + (A_t + B_t K_t)^T p_{t+1} + (A_t + B_t K_t)^T P_{t+1} B_t k_t \tag{9}$$

$$P_t = Q_t \; + \; K_t^T R_t K_t \; + \; (A_t \; + \; B_t K_t)^T P_{t+1} (A_t + B_t K_t) \tag{10}$$

In the forward function the controller has been implemented as follows:

$$\mathit{control} = k_t + K_t (x_t^i - x_t^{i-1}) \tag{11}$$

where x_t^{i-1} is the reference trajectory while x_t^i is the new state of the system after the application of the control law. It is important to notice that the final value for the cost function as well as the time needed for the pendulum stabilization are both influenced by the random initial configuration of the system. This observation is also evident in the graphical rendering provided by Gymnasium.