Reinforcement Learning (2022/2023)

Assignment 1

Student Name: Gabriele G. Di Marzo
Student ID: 2012633

---

1. **Theory**

   (a) Demonstrate the convergence of the Policy Iteration Algorithm.
   SOLUTION:
   for prooving the convergence of the algorithm, i started form the difference in module of the Q - function:

$$\left|Q^{i-1\pi}(s,a) - Q^{(}s,a^{)}\right| \tag{1}$$

   considering the Q function at the next iteration, i can write the equation (1) as:

$$\left|max_a(Q^{i\pi}(s,a)) - Q^{(}s,a^{)}\right| \tag{2}$$

   considering the property of the norm: (norm of the maximum <= maximum of the norm), we can consider that

$$\left|max_a(Q^{i\pi}(s,a)) - Q^{(}s,a)\right| <= max_a\left|Q^{i\pi}(s,a) - Q^{(}s,a)\right| \tag{3}$$

$$\left|max_a(Q^{i\pi}(s,a)) - Q^{*}(s,a)\right| <= max_a\left|[r(s) + \gamma \mathop{\mathbb{E}}_{s'\sim p(.|\pi(a),s)} V^{i}(s')] - [r(s) + \gamma \mathop{\mathbb{E}}_{s'\sim p(.|\pi(a),s)} V^{*}(s')]\right| \tag{4}$$

   Removing the equals term and isolating the expectation, we can see that we obtain a common γ. This value must be smaller that 1, and repeating the procedure we can get a $\gamma^i = 0$ factor in front of the module.
   finally, we can consider the first equation and the final one, it's possible that the distance between the value of the two Q-function ($iteration Q^i$, $optimal value Q^*$), are smaller or equal to 0.
   As it turned out:

$$\left|Q^{i-1\pi}(s,a) - Q^{*}(s,a^{*})\right| <= 0 \tag{5}$$

   (b) Considering the environment settings:
      i. States : $\{s_i : i \in \{1...7\}$
      ii. Reward : $\begin{cases} 1 & \text{if } s = s_1 \\ 10 & \text{if } s = s_7 \\ 0 & \text{otherwise} \end{cases}$
      iii. Dynamics : $p(s_6|s_6,a_1) = 0.5$, $p(s_7|s_6,a_1) = 0.5$
      iv. Policy : $\pi(s) = a_1 \forall s \in S$
      v. Value function at iteration $k = 1$: $V_k = [1,0,0,0,0,0,10]$
      vi. $\gamma = 0.5$

   Compute V_{k+1}:

   SOLUTION:
   At each iteration the value function is calculated as :
   $V_{k+1} = r(s) + \gamma\, \mathbb{E}_{s'\sim p(.|\pi(a),s)}[V_k(s')]$

The Value of state s_{k+1} is equal to:

$V_{k+1}(s_6) = r(s) + \gamma \sum_{s'} P(s'|s, \pi(a)) V_k(s')$
$V_{k+1}(s_6) = 0 + \gamma(Pr(s_6|s_6, a_1)V_k(s_6) + Pr(s_7|s_6, a_1)V_k(s_7)$
$V_{k+1}(s_6) = 0 + 0.5 * (0.5 * 0 + 0.5 * 10) = 2.5$

2. **Practical**

(a) **Policy iteration:**
The policy iteration algorithm consist of two, well-defined and separate, parts. The first part performs policy evaluation by estimatin the $V - function$ for each state, given a fixed policy. The second part performs policy improvement by iterating and calculation the $Q - function$ for each possible action. Then if the new Q-value is greater that the previous value, the corresponding action became part of the policy.
The procedure is stopped when there is no difference between older and new policy.
This behavior is implemented through the boolean variable "changed". In the code there is a github link to an implementation that helped me understand the correct implementation of the code.

Contributor:
Andrea Giuseppe Di Francesco

(b) **Ilqr**:

The goal of the LQR is find a policy that is suitable for the Optimal control. To make this, we can consider the MarkovDecisionProcess [MDP] in the finite-state setting. Defining a quadratic cost function and linearizing the system, it's possible compute the $V_h^*(x_t) = x_t^T P_h x_h$.
The P matrix is extracted from the Riccardi Equation.
The state in this setting is given by $x_{t+1} = Ax_t + Bu_t$
Matrices A and B are needed to bring the problem into the canonical control form. I chose for convenience to compute them with Matlab, and copy the results into the python script.
The matlab script simply creates the symbolic expressions representing the state, inserts them into the eponymous vector and differentiates it with respect to the state and input. The result is the 4X4 matrix A and the vector B. In the code there are a number of simplify functions to try to reduce the complexity of the expressions. I have noticed a marked improvement in readability, at the expense of the powers of the various terms, which come to be raised to the 5th power.

The Matlab code:

```
clear all;
syms x x_dot  theta theta_dot ;
syms dt u L_p L g M_p M;


theta_2dot = ( (g*sin(theta)) - (cos(theta)) * ...
        ( (u + ( L_p*(theta_dot^2)*(sin(theta)) ) )  ...
        / (M) )...
    ) ...
    / ( L* ( (4/3) - ((M_p*(cos(theta)^2)) / M) ) );
x_2dot =  ((u + (L_p*(theta_dot^2)*sin(theta) ) - (L_p*theta_2dot*cos(theta)
 ) ) /(M) );

x_2dot = simplify(x_2dot);
```

```
    theta_2dot = simplify(theta_2dot);

    x_1 = x + x_dot*dt;
    x_dot_1 = x_dot + dt* x_2dot;
    theta_1 = theta + dt*theta_dot;
    theta_dot_1 =  theta_dot + dt*theta_2dot;

    x_dot_1 = simplify(x_dot_1);
    theta_dot_1 = simplify(theta_dot_1);

    state = [x_1, x_dot_1, theta_1, theta_dot_1];
    A = jacobian(state, [x, x_dot, theta, theta_dot]);
    A = simplify(A);

    B = jacobian(state, u);
    B = simplify(B);

```

K and P matrices are calculated as in the exercise carried out in class, i.e. by iterative procedure derived from the Riccardi equation

Contributor: Syrine enneifer