Assignment 1: Value Iteration, Policy Iteration and iLQR

Reinforcement Learning - A.Y. 2023/2024

November 8th, 2023

Rules

The assignment is due on November 8th, 2023. Students may discuss assignments, but each student must code up and write up their solutions independently. Students must also indicate on each homework the names of the colleagues they collaborated with and what online resources they used.

The theory solutions must be submitted in a pdf file named "XXXXXXX.pdf", where XXXXXXX is your matricula. We encourage you to type the equations on an editor rather than uploading a scanned written solution.

The practice exercises must be uploaded in a zip file named "XXXXXXX.zip", where XXXXXXX is your matricula. The zip file must have the same structure of the assignment.zip that you find in the attachments, but with the correct solution. You are only allowed to type your code in the files named "student.py". Any modification to the other files will result in penalization. You are not allowed to use any other python library that is not present in python or in the "requirements.txt" file. You can use as many functions you need inside the "student.py" file.

Theory

1. Derive the formula to get the minimum number of iterations of Value Iteration that are needed if we want an error on the quality of the policy that is at most ϵ.
2. Given the following environment settings
   - States: $\{s_i : \ i \in \{1...7\}\}$
   - Reward:
   $$r(s,a) = \begin{cases} 0.5 & \text{if } s = s_1 \\ 5 & \text{if } s = s_7 \\ 0 & \text{otherwise} \end{cases}$$
   - Dynamics: $p(s_6|s_6, a_1) = 0.3$, $p(s_7|s_6, a_1) = 0.7$
   - Policy: $\pi(s) = a_1 \ \ \forall s \in S$

   and the value function at the iteration $k = 1$:

   $$v_k = [0.5,\ 0,\ 0,\ 0,\ 0,\ 0,\ 5],$$

   with $\gamma = 0.9$.

   Compute $V_{k+1}(s_6)$ following the *Value Iteration* algorithm.

Practice

1. Implement the transition probabilities and the reward function of the FrozenLake Gymnasium environment. See below for more details on their specification.

   In folder "policy_iteration" you find three files:

   - "policy_iteration.py" implements the policy iteration algorithm exactly as we have seen during practice lessons.
   - "main.py" contains the definition of the environment and its attributes (end_state, obstacles, etc.) and the code to run the tests
   - "student.py" contains the function "reward_function", "check_feasibility" and "transition_probabilities" that you have to fill in.

   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. For example, if the action is "LEFT", the agent can either move left, up or down, each with probability 1/3.

2. Implement the iLQR algorithm for solving the Pendulum environment. Specifics of the environment are available at https://gymnasium.farama.org/environments/classic_control/pendulum/. In folder "ilqr" you find two files:

   - "main.py", that contains the python script to run the tests.
   - "student.py" contains the class implementing iLQR with the functions that you will need to fill in.

   In the *pendulum_dyn* function, you need to implement the equations for the pendulum dynamics:

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

   $$\theta_{t+1} = \theta + \dot{\theta}_{t+1} \, dt$$

   In the *backward* function, you will need to implement the formulas to compute the P and K matrices from the LQR-LTV algorithm. In this problem, there are also additional terms coming from the cost's linearization, performed using the 2nd order Taylor expansion. These terms can be computed using the following formulas:

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

   $$p_t = q_t + K^T (R_t k_t + r_t) + (A_t + B_t K)^T p_t + (A_t + B_t K)^T P_{t+1} B_t k_t$$

   Finally, in the *forward* function, you will need to use the K and k matrices to update the control in the following way:

   $$control = k_t + K_t (x_t^i - x_t^{i-1})$$