Lab1 - MDP - Radhika Nayar - 805-226-085
0(a): Github repo link: https://github.com/radforlyfe/Robotics/blob/master/MDP.ipynb
0(b,c): 100% individual work

Problem Statement:
The objective of this lab was to explore Markov Decision Processes (MDP) by developing and implementing a model of simple discretized robot behaviour in 2D grid world and using it to accomplish some prescribed tasks (goal state). The goal was to find the optimal policy (set of best actions for each state) that maximizes the expected sum of rewards.

Tasks:
1. Set up the MDP system: The robot lives in 2D grid world of length L and W and can face in any 12 headings and choose from several actions (movement followed by rotation) with a pre-rotation error. Create objects to represent state space and action space and functions that returns the state transition probability, the next state and the reward. Formulate this problem as MDP by assuming that every state is fully observable by the robot and that the Markov property holds for every state.
Goal: Given MDP{S, A, P, R, H, γ} $\rightarrow$ find the optimal policy π^*.
2. Policy Iteration: Solve the policy iteration by repeatedly performing two steps: Policy Evaluation and Policy Improvement. Policy Evaluation: Start with a random initial policy π^0 and evaluate this initial policy by identifying the optimal actions with respect to the current value function until convergence.
Find the values with simplified Bellman updates and iterate until the values converge:

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

Policy Improvement: using one-step look-ahead with resulting converged values, update the policy (find the best action - doing a greedy policy update):

$$\pi_{k+1}(s) = \arg\max_a \sum T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

At every step the policy improves, and the policy iteration is guaranteed to converge and at convergence, the initial policy and its value function are the optimal policy and optimal value function.
3. Value Iteration: an alternative iterative approach to first finding the optimal value function using the Bellman optimality conditions and then extracting the optimal policy. Start with an initial value of 0 for all states. By iterating over the horizon H, for all states, assume the next state's current value is optimal and use that to accurately estimate V*.

$$V_{i+1}^*(s) \leftarrow \max_a \sum T(s, a, s') \left[R(s, a, s') + V_i^*(s')\right]$$

The optimal value function V* is the expected sum of rewards accumulated when starting from state s and acting most optimally for the horizon of i steps.

Results:
- For a robot starting at the same state x=1, y=6, h=6 (given in 3c) -- in (3h) plot of the trajectory of robot using policy iteration and in (4c) the plot of the trajectory of robot using value iteration -- was the same. This means that optimal policy and optimal value of the robot starting at initial state converged to the same solution using policy iteration and value iteration.
- The computation time observed for value iteration (4c) is much faster (2.57s) compared to the computation time observed for policy iteration (3i) that is (4.841s). This is because in value iteration, we can solve the value function directly for optimal policy; however, in policy iteration we have to solve value function for the initial policy and for each policy in each iterative step making it computationally longer. This is the reason why the policy iteration is slower since every single policy improvement in each iterated is evaluated making it computationally longer.
- In Q5 we are asked to change the error probability to pre-rotate to 25% (changing our environment to more stochastic). The trajectory of the robot becomes much harder to predict and keeps fluctuating a lot more due to the randomness added by environment. The computation time using value iteration and pe=25% increases the time to (3.14s). The time to reach the goal becomes slower as the trajectory is changing over every state a lot more due to the randomness added by the environment. The algorithm needs to iterate through a lot more combinations of headings making the computation time longer. This result is similar to changing the reward function to more strict rule of only giving reward at heading direction of 6 (5b). Again the computation time increases since the algorithm has to take more steps in the state space (13 steps versus 11 steps in 4b).