Markov Decision Processes: Exercises

Exercise 1: Implementing MDP and Agent Classes

In this exercise, you will implement two Python classes `MDP` and `Agent`.

- `MDP` class:
  - Attributes: number of states, number of actions, transition function, reward function, discount factor γ, current state and start state.
  - Methods:
    - (necessary) getters and setters for the attributes.
    - `play(action) -> state, reward` : receives an action and returns the next state and reward.
    - `get_possible_actions(state) -> list`: receives a state and returns the list of possible actions.
    - `simulate(agent, int, boolean) -> list`: simulates the agent's interaction with the MDP for a given number of steps, returning either the total reward if the boolean parameter is set to `False`, or the detailed history of states, actions, and rewards if the boolean parameter is set to `True`.
- `Agent` class:
  - Attributes: number of states, number of actions, and its policy.
  - Methods:
    - (necessary) getters and setters for the attributes.
    - `choose_action(state) -> action`: returns the action based on the current policy for state s.

The goal is to create a modular framework where the `Agent` interacts with the `MDP`, choosing actions and receiving rewards and next states.

Exercise 2: Simple Deterministic MDP

Consider a simple MDP with two states s_1 and s_2, and two possible actions a_1 and a_2.

- States: $S = \{s_1, s_2\}$
- Actions: $A = \{a_1, a_2\}$
- Transition function (deterministic):
  - From s_1, if a_1 is chosen, go to s_2; if a_2 is chosen, stay in s_1.
  - From s_2, if a_1 is chosen, stay in s_2; if a_2 is chosen, go to s_1.
- Reward function:
  - $R(s_1, a_1) = 1$, $R(s_1, a_2) = 0$
  - $R(s_2, a_1) = 2$, $R(s_2, a_2) = 1$
- Discount factor: $\gamma = 0.9$

Task:

- Give a graphic representation of this MDP.
- Implement this MDP and simulate an agent interacting with it.
- Calculate the value function $V(s_1)$ and $V(s_2)$ given a simple deterministic policy (e.g., always choose a_1).

Exercise 3: Stochastic MDP with More States

Now, consider an MDP with three states s_1, s_2, and s_3, and two actions a_1 and a_2. Transitions are stochastic:

- States: $S = \{s_1, s_2, s_3\}$
- Actions: $A = \{a_1, a_2\}$
- Transition function (stochastic):
  - From s_1, with a_1, there is a 0.8 probability to go to s_2 and 0.2 probability to go to s_3.
  - From s_1, with a_2, there is a 0.5 probability to stay in s_1 and a 0.5 probability to go to s_3.
  - From s_2, a_1 and a_2 both lead to s_1 with probability 1.
  - From s_3, a_1 and a_2 both lead to s_2 with probability 1.
- Reward function:
  - $R(s_1, a_1) = 1$, $R(s_1, a_2) = 0$
  - $R(s_2, a_1) = 2$, $R(s_2, a_2) = 1$
  - $R(s_3, a_1) = 3$, $R(s_3, a_2) = 2$
- Discount factor: $\gamma = 0.95$

  Task:
- Draw a graphic representation of this MDP.
- Implement this MDP and simulate an agent with a stochastic policy (random action choice).
- Compute the expected value functions $V(s_1)$, $V(s_2)$, and $V(s_3)$ using the Bellman equation.

Exercise 4: Recycling Robot[1]

A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made y a reinforcement learning agent based on the current charge level of the battery.

To make a simple example, we assume that only two charge levels can be distinguished, comprising a small state set $S = \{high, low\}$. In each state, the agent can decide whether to (1) actively search for a can for a certain period of time, (2) remain stationary and wait for someone to bring it a can, or (3) head back to its home base to recharge its battery. When the energy level is high, recharging would always be foolish, so we do not include it in the action set for this state. The action sets are then $A(high) = \{search, wait\}$ and $A(low) = \{search, wait, recharge\}$.

The rewards are zero most of the time, but become positive when the robot secures an empty can, or large and negative if the battery runs all the way down. The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward). If the energy level is *high*, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a *high* energy level leaves the energy level *high* with probability α and reduces it to *low* with probability $1 - \alpha$.

On the other hand, a period of searching undertaken when the energy level is *low* leaves it *low* with probability β and depletes the battery with probability $1 - \beta$. In the latter case, the robot must be rescued, and the battery is then recharged back to *high*. Each can collected by the robot counts as a *unit* reward, whereas a reward of -3 results whenever the robot has to be rescued. Let *rsearch* and *rwait*, with $rsearch > rwait$, respectively denote the expected number of cans the robot will collect (and hence the expected reward) while searching and while waiting.

Finally, suppose that no cans can be collected during a run home for recharging, and that no cans can be collected on a step in which the battery is depleted.

Task:

- Model the MDP for this problem.
- Implement this MDP and compute the value function for the states `search` and `recycling`.
- Simulate the robot's behavior over time for different policies.

[1] Adapted from Sutton's book