1. $\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a\in\mathcal{A}}\{\mathcal{R}(s,a)+\gamma\sum_{s'\in\mathcal{N}}\mathcal{P}(s,a,s')\cdot\boldsymbol{V}(s')\}$ $\text{limit}\,(B^*)^i(V_0)\to V^*$

$V_1(s_1)$: $a=a_1$: $R(s_1,a_1)+\sum_{s'\in N}P(s_1,a_1,s')V_0(s')$ $\boxed{s'=s_3?}$

$q_1(s_1,a_1) = 8+ \quad 0.2V_0(s_1)+0.6V_0(s_2) = 8+2+0.6=10.6$

$\pi_1(s_1)=a_2$ ✓ $a=a_2$: $R(s_1,a_2)+P(s_1,a_2,s_1)V_0(s_1)+P(s_1,a_2,s_2)V_0(s_2)$

$q_1(s_1,a_2) = 10+ \quad 0.1\times10+0.2\times1 = 11.2$

$V_1(s_1) = 11.2$

$V_1(s_2)$: $a=a_1$: $R(s_2,a_1)+P(s_2,a_1,s_1)V_0(s_1)+P(s_2,a_1,s_2)V_0(s_2)$

$\pi_1(s_2)=a_1$ or a_2 $q_1(s_2,a_1) = 1+ \quad 0.3\times10+0.3\times1 = 4.3$

$a=a_2$: $R(s_2,a_2)+P(s_2,a_2,s_1)V_0(s_1)+P(s_2,a_2,s_2)V_0(s_2)$

$q_2(s_2,a_2) = -1+ \quad 0.5\times10+0.3\times1 = 4.3$

$V_1(s_2) = 4.3$

$V_2(s_1)$: $a=a_1$: $R(s_1,a_1)+P(s_1,a_1,s_1)V_1(s_1)+P(s_1,a_1,s_2)V_1(s_2)$

$\pi_2(s_1)=a_1$ $q_2(s_1a_1) = 8+0.2\times11.2+0.6\times4.3 = 12.82$ ✓

$a=a_2$: $R(s_1,a_2)+P(s_1,a_2,s_1)V_1(s_1)+P(s_1,a_2,s_2)V_1(s_2)$

$q_2(s_1,a_2) = 10+ \quad 0.1\times11.2+0.2\times4.3 = 11.98$

$V_2(s_1) = 12.82$

$V_2(s_2)$: $a=a_1$: $R(s_2,a_1)+P(s_2,a_1,s_1)V_1(s_1)+P(s_2,a_1,s_2)V_1(s_2)$

$q_2(s_2,a_1) = 1+ \quad 0.3\times11.2+0.3\times4.3 = 5.65$

$\pi_2(s_2)=a_2$

✓ $a=a_2$: $R(s_2,a_2)+P(s_2,a_2,s_1)V_1(s_1)+P(s_2,a_2,s_2)V_1(s_2)$

$q_2(s_2,a_2) = -1+ \quad 0.5\times11.2+0.3\times4.3 = 5.89$

$V_2(s_2) = 5.89$

$V_k(S_1)$: $q_k(S_1,a_1) = R(S_1,a_1) + 0.2V_{k-1}(S_1) + 0.6V_{k-1}(S_2)$

$q_k(S_1,a_2) = R(S_1,a_2) + 0.1V_{k-1}(S_1) + 0.2V_{k-1}(S_2)$

$q_k(S_1,a_2) - q_k(S_1,a_1) = 2 - 0.1V_{k-1}(S_1) - 0.4V_{k-1}(S_2) < 0$

0.1×12.82 $\qquad$ $0.4 \times 5.89)$

$\uparrow$ $\qquad$ $\uparrow$

$\pi_k(S_1) = a_1$

$V_k(S_2)$: $q_k(S_2,a_1) = R(S_2,a_1) + 0.3V_{k-1}(S_1) + 0.3V_{k-1}(S_2)$

$q_k(S_2,a_2) = R(S_2,a_2) + 0.5V_{k-1}(S_1) + 0.3V_{k-1}(S_2)$

$q_k(S_2,a_2) - q_k(S_2,a_1) = -2 + 0.2V_{k-1}(S_1) > 0$

0.2×12.82

$\uparrow$

$\pi_k(S_2) = a_2$

Stanford CME 241 (Winter 2021) - Assignment 4

Assignments:

1. **Manual Value Iteration:** Consider a simple MDP with $\mathcal{S} = \{s_1, s_2, s_3\}, \mathcal{T} = \{s_3\}, \mathcal{A} = \{a_1, a_2\}$. The State Transition Probability function

   $$\mathcal{P} : \mathcal{N} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$$

   is defined as:

   $$\mathcal{P}(s_1, a_1, s_1) = 0.2, \mathcal{P}(s_1, a_1, s_2) = 0.6, \mathcal{P}(s_1, a_1, s_3) = 0.2$$
   $$\mathcal{P}(s_1, a_2, s_1) = 0.1, \mathcal{P}(s_1, a_2, s_2) = 0.2, \mathcal{P}(s_1, a_2, s_3) = 0.7$$
   $$\mathcal{P}(s_2, a_1, s_1) = 0.3, \mathcal{P}(s_2, a_1, s_2) = 0.3, \mathcal{P}(s_2, a_1, s_3) = 0.4$$
   $$\mathcal{P}(s_2, a_2, s_1) = 0.5, \mathcal{P}(s_2, a_2, s_2) = 0.3, \mathcal{P}(s_2, a_2, s_3) = 0.2$$

   The Reward Function

   $$\mathcal{R} : \mathcal{N} \times \mathcal{A} \rightarrow \mathbb{R}$$

   is defined as:

   $$\mathcal{R}(s_1, a_1) = 8.0, \mathcal{R}(s_1, a_2) = 10.0$$
   $$\mathcal{R}(s_2, a_1) = 1.0, \mathcal{R}(s_2, a_2) = -1.0$$

   Assume discount factor $\gamma = 1$.

   Your task is to determine an Optimal Deterministic Policy *by manually working out* (not with code) simply the first two iterations of Value Iteration algorithm.

   - Initialize the Value Function for each state to be it's max (over actions) reward, i.e., we initialize the Value Function to be $v_0(s_1) = 10.0, v_0(s_2) = 1.0, v_0(s_3) = 0.0$. Then manually calculate $q_k(\cdot, \cdot)$ and $v_k(\cdot)$ from $v_{k-1}(\cdot)$ using the Value Iteration update, and then calculate the greedy policy $\pi_k(\cdot)$ from $q_k(\cdot, \cdot)$ for $k = 1$ and $k = 2$ (hence, 2 iterations).
   - Now argue that $\pi_k(\cdot)$ for $k > 2$ will be the same as $\pi_2(\cdot)$. Hint: You can make the argument by examining the structure of how you get $q_k(\cdot, \cdot)$ from $v_{k-1}(\cdot)$. With this argument, there is no need to go beyond the two iterations you performed above, and so you can establish $\pi_2(\cdot)$ as an Optimal Deterministic Policy for this MDP.

2. **Frog Croaking revisited:** Let's revisit the frog-croaking Markov Decision Process (Question 3 on Assignment 3). Compute the Optimal Value Function and Optimal Policy using Policy Iteration algorithm as well as Value Iteration algorithm (use the functions policy_iteration and value_iteration from file rl/dynamic_programming.py). Analyze the computational efficiency of Policy Iteration versus Value Iteration in terms of speed of convergence to the Optimal Value Function. How do Policy Iteration and Value Iteration compare against the brute-force method of evaluating the MDP for all 2^n deterministic policies (as was suggested in Assignment 3)? Plot some graphs of convergence speed for different values of n (the number of lilypads).

3. **Job-Hopping and Wages-Utility-Maximization:** You are a worker who starts every day either employed or unemployed. If you start your day employed, you work on your job for the day (one of n jobs, as elaborated later) and you get to earn the wage of the job for the day. However, at the end of the day, you could lose your job with probability $\alpha \in [0, 1]$, in which case you start the

next day unemployed. If at the end of the day, you do not lose your job (with probability $1 - \alpha$), then you will start the next day with the same job (and hence, the same daily wage). On the other hand, if you start your day unemployed, then you will be randomly offered one of n jobs with daily wages $w_1, w_2, \ldots w_n \in \mathbb{R}^+$ with respective job-offer probabilities $p_1, p_2, \ldots p_n \in [0, 1]$ (with $\sum_{i=1}^n p_i = 1$). You can choose to either accept or decline the offered job. If you accept the job-offer, your day progresses exactly like the *employed-day* described above (earning the day's job wage and possibly (with probability α) losing the job at the end of the day). However, if you decline the job-offer, you spend the day unemployed, receive the unemployment wage $w_0 \in \mathbb{R}^+$ for the day, and start the next day unemployed. The problem is to identify the optimal choice of accepting or rejecting any of the job-offers the worker receives, in a manner that maximizes the infinite-horizon *Expected Discounted-Sum of Wages Utility*. Assume the daily discount factor for wages (employed or unemployed) is $\gamma \in [0, 1)$. Assume Wages Utility function to be $U(w) = \log(w)$ for any wage amount $w \in \mathbb{R}^+$. So you are looking to maximize

$$\mathbb{E}[\sum_{u=t}^{\infty} \gamma^{u-t} \cdot \log(w_{i_u})]$$

at the start of a given day t (w_{i_u} is the wage earned on day u, $0 \le i_u \le n$ for all $u \ge t$).

- Express with clear mathematical notation the state space, action space, transition function, reward function, and write the Bellman Optimality Equation customized for this MDP.
- You can solve this Bellman Optimality Equation (hence, solve for the Optimal Value Function and the Optimal Policy) with a numerical iterative algorithm (essentially a Dynamic Programming algorithm customized to this problem). Write Python code for this numerical algorithm. Clearly define the inputs and outputs of your algorithm with their types (int, float, List, Mapping etc.). For this problem, don't use any of the MDP/DP code from the git repo, write this customized algorithm from scratch.

4. **Two-Stores Inventory Control:** We extend the capacity-constrained inventory example implemented in rl/chapter3/simple_inventory_mdp_cap.py as a FiniteMarkovDecisionProcess (the Finite MDP model for the capacity-constrained inventory example is described in detail in Chapters 1 and 2 of the RLForFinanceBook). Here we assume that we have two different stores, each with their own separate capacities C_1 and C_2, their own separate Poisson probability distributions of demand (with means λ_1 and λ_2), their own separate holding costs h_1 and h_2, and their own separate stockout costs p_1 and p_2. At 6pm upon stores closing each evening, each store can choose to order inventory from a common supplier (as usual, ordered inventory will arrive at the store 36 hours later). We are also allowed to transfer inventory from one store to another, and any such transfer happens overnight, i.e., will arrive by 6am next morning (since the stores are fairly close to each other). Note that the orders are constrained such that following the orders on each evening, each store's inventory position (sum of on-hand inventory and on-order inventory) cannot exceed the store's capacity (this means the action space is constrained to be finite). Each order made to the supplier incurs a fixed transportation cost of K_1 (fixed-cost means the cost is the same no matter how many units of non-zero inventory a particular store orders). Moving any non-zero inventory between the two stores incurs a fixed transportation cost of K_2.

   Model this as a derived class of FiniteMarkovDecisionProcess much like we did for SimpleInventoryMDPCap in the code repo. Set up instances of this derived class for different choices of the problem parameters (capacities, costs etc.), and determine the Optimal Value Function and Optimal Policy by invoking the function value_iteration (or policy_iteration) from file rl/dynamic_programming.py.

Analyze the obtained Optimal Policy and verify that it makes intuitive sense as a function of the problem parameters.