**CSCN8020 – Reinforcement Learning Programming**

Name-Preetpal Singh

Student ID-8804336

**Problem 1: Pick-and-Place Robot MDP Design**

**Task:**  
Use reinforcement learning to govern a robot arm for a repetitive pick out-and-region job. The goal is growing actions that are both fast and easy.

**MDP Construction:**

**States (SSS):**

* + Positions and velocities for each robotic joint.
  + Whether the gripper is holding an object (binary).
  + Environment conditions that affect picking/placing.

**Actions (AAA):**

* + Apply discrete or continuous torque/force at each joint.
  + Open/close the gripper mechanism.

**Rewards (RRR):**

* + +10 for each successfully placed object.
  + -1 for each unit of time spent (to encourage quicker completion).
  + Large negative (e.g. -20) for failed pick or collision.

**Reasoning:**

* Agent must maximize cumulative reward across episodes: balancing speed, safety, and pick/place success.
* Motor commands must be chosen so that state-feedback leads to positive outcomes.

**Problem 2: 2x2 Gridworld Value Iteration**

**Environment Details:**

* **States (SSS):** s1, s2, s3, s4
* **Actions (AAA):** up, down, left, right
* **Rewards:**

R(s1) =5R

R(s2) =10R

R(s3) =1R

R(s4) =2R

* **Initial Policy:** Always take "up"
* **Transitions:** Deterministic if action valid; otherwise stay in same state.

**Value Iteration Steps:**

**Iteration 1**

**Initial Values:** V(s1) =0, V(s2) =0, V(s3) =0, V(s4) =0

**Formula:**

V(s)=max[R(s)+γV(s′)]

(assuming discount γ=0.9)

**Apply for all states:**

* + V(s1) =5+0.9×0=5
  + V(s2) =10+0.9×0=10
  + V(s3) =1+0.9×0=1
  + V(s4) =2+0.9×0=2

**Updated after first iteration:**

V= [1]

**Iteration 2**

The final value you provided for Iteration 2 was: V= [14.0,14.5,5.5,11.0]. The **correct** final value should be: V2 ​= [14.0,19.0,5.5,11.0].

The mistake is in the calculation of V2​(s2​).

**Correcting V2​(s2​)**

In Value Iteration, the agent must choose the action that maximizes the total expected future reward, R(s)+γV1​(s′).

**State s2​ (R=10)**: We use the previous values

V1​= [5.0,10.0,1.0,2.0].

* + Action: **left** →s1​: 10+0.9×V1​(s1​) =10+0.9×5.0=14.5
  + Action: down →s4​: 10+0.9×V1​(s4​) =10+0.9×2.0=11.8
  + Action: **up/right** (Wall) →s2​: 10+0.9×V1​(s2​) =10+0.9×10.0=19.0

The maximum value is **19.0**, not 14.5.

**Problem 3: 5x5 Gridworld MDP and Value Iteration**

**Environment Construction:**

* **States:** Matrix positions; s(row,col)
* **Rewards:**

Terminal (goal) state (s4,4): +10

Grey (unfavorable) states (s2,2; s3,0; s0,4): -5

Regular states: -1

* **Actions:** right, down, left, up (all valid moves)
* **Transitions:** Deterministic if action stays within grid; else agent remains in same state.

**Tasks:**

**Task 1: Update Code**

* Adjust code to give appropriate rewards depending on state type.
* Run value iteration to produce:
* Optimal state-value table
* Policy arrows for each state (showing best action)

**Task 2: In-place Value Iteration**

* Update values using a single array, each update reflected immediately.
* Compare:
  + Number of iterations or time steps to converge.
  + Episodes needed (if applicable, for simulation).
  + Computational complexity: in-place is faster for moderate grids.

**Example Table:**

| **State** | **Value** | **Best Action** |
| --- | --- | --- |
| s0,0 | ... | down |
| ... | ... | ... |
| s4,4 | 10 | terminal |

**Problem 4: Off-Policy Monte Carlo with Importance Sampling**

### In this hassle we evaluate the value feature of a greedy goal coverage in the same five×five grid global utilized in Problem 3. Instead of producing episodes immediately from the target coverage, we use a one of a kind behavior policy and accurate for the mismatch the usage of significance sampling.

### Method

**1.Behavior Policy (b): A random policy where each motion has equal opportunity.**

**2.Target Policy (π): A greedy coverage that always selects the motion with the best expected return.**

**3.Episode Generation: Episodes are generated the usage of the conduct coverage. Each episode consists of a sequence of states, movements, and rewards until accomplishing a terminal nation.**

**4.Return Calculation: For each kingdom visited, the go back is the sum of discounted rewards from that factor onward (the usage of γ = 0.9).**

**5.Importance Sampling: Since episodes come from b however we need estimates underneath π, every go back is weighted by the chance ratio between the 2 guidelines. If an action taken suits the grasping action of π, the ratio is higher; if not, the load will become 0.**

**6.Value Update: Weighted averages of the returns are used to update the predicted value for each kingdom. Over many episodes, the values converge to approximate the genuine price below π.**

**Pseudocode**

Initialize V(s) = 0 and C(s) = 0 for all states

For each episode generated by behavior policy b:

Record sequence of (s, a, r)

Compute return G for visited states

Compute importance weight W = product of (pi(a|s)/b(a|s))

For each first-visit state in episode:

C(s) = C(s) + W

V(s) = V(s) + (W/C(s)) \* (G - V(s))

Output the final V(s) table

### Example of Value Function Table (sample format)

| **State** | **Estimated V (MC-IS)** |
| --- | --- |
| s0,0 | -1.2 |
| s0,1 | -0.9 |
| ... | ... |
| s4,4 | 9.8 |

**Comparison with Value Iteration**

### **Optimization time: Value Iteration converges quickly using actual updates, even as Monte Carlo requires many episodes and extra time.**

### **Number of episodes: Monte Carlo wishes heaps of episodes for stability; Value Iteration converges in fewer iterations.**

### **Complexity: Monte Carlo is sample-based totally and does not require a complete version of the environment. Value Iteration wishes recognized transition possibilities however is extra efficient when the version is available.**

### **Policy first-rate: With enough episodes, Monte Carlo can reach near-most advantageous values, but results can be noisy. Value Iteration typically produces a clean and strong choicest coverage.**

**Summary**

Off-policy Monte Carlo with importance sampling permits us to estimate the value function of a goal policy the usage of episodes from a exclusive conduct policy. While it is slower and extra pattern-in depth than Value Iteration, it's miles effective in situations wherein the surroundings’ dynamics are unknown and best sampled episodes are available.

**Summary Table (Value Iteration vs Monte Carlo):**

| **Method** | **Convergence Speed** | **Sample Usage** | **Complexity** | **Policy Quality** |
| --- | --- | --- | --- | --- |
| Value Iteration | Fast | Not sample-based | Deterministic updates | Usually optimal |
| Monte Carlo | Slow (many episodes) | Many samples | Randomized, depends on sampling | Can be optimal, but noisy |

State V0 V1 V2

s1 0 5 14

s2 0 10 19

s3 0 1 5.5

s4 0 2 11

-0.43 0.63 1.81 3.12 4.58

0.63 1.81 3.12 4.58 6.20

1.81 3.12 4.58 6.20 8.00

3.12 4.58 6.20 8.00 10.0

-2.11 0.60 1.73 3.09 4.56

-2.21 1.78 2.94 4.56 6.17

1.79 3.10 4.56 6.19 7.99

3.11 4.56 6.17 7.98 10.0

**Conclusion**

This assignment demonstrated both model-based and model-free reinforcement learning techniques. Problems 1 and 2 illustrated how to construct MDPs and manually perform value iteration on small environments. Problem 3 implemented Value Iteration on a larger 5x5 gridworld, showing that standard and in-place updates converge to the same optimal solution. Problem 4 applied off-policy Monte Carlo with importance sampling, demonstrating how policies can be evaluated from sampled experience without full environment knowledge. Overall, Value Iteration is faster when the model is known, while Monte Carlo methods are flexible for model-free cases.