Experiment:-8

Aim:- Write a program for policy iteration and value iteration in reinforcement learning.

Reinforcement Learning (RL) is a learning paradigm where an agent interacts with an environment to maximize cumulative rewards. Two important algorithms used for solving Markov Decision Processes (MDPs) are:

Policy Iteration
Value Iteration

Both methods aim to find the optimal policy (decision-making strategy) for the agent.

Markov Decision Process (MDP)

An MDP is defined by:

- S → Set of states
- A → Set of actions
- P(s'|s, a) → Transition probability
- R(s, a) → Reward
- γ → Discount factor

The goal is to find an optimal policy π* that maximizes long-term rewards.

1. Policy Iteration

Policy Iteration is a method that alternates between:

a) Policy Evaluation

Calculate the **value function Vπ(s)** for the current policy π.
It determines the expected long-term reward of following the policy from each state.

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_\pi(s')]$$

b) Policy Improvement

Improve the policy by choosing better actions based on the updated values:

$$\pi_{\text{new}}(s) = \arg\max_a \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_\pi(s')]$$

Process continues until:

Policy stops changing → **Optimal Policy Found**.

2. Value Iteration

Value Iteration combines policy evaluation and improvement into **one step** using Bellman Optimality Equation.

It updates the value of each state using:

$$V(s) = \max_a \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V(s')]$$

Key Idea:

Instead of evaluating a full policy, we repeatedly update values until they converge,then extract the optimal policy.

Program Implementation

```
# Reinforcement Learning: Policy Iteration and Value Iteration (Pure Python)

import numpy as np

# MDP Setup

states = [0, 1, 2, 3]  # 4 states

actions = [0, 1]       # 2 actions

gamma = 0.9            # Discount factor
```

```
# Transition Probabilities and Rewards
# T[state][action] = (next_state, reward)
T = {
    0: {0: (1, 0), 1: (2, 0)},
    1: {0: (3, 1), 1: (0, 0)},
    2: {0: (1, 0), 1: (3, 1)},
    3: {0: (3, 0), 1: (3, 0)},  # Terminal state
}
# POLICY ITERATION
def policy_evaluation(policy, V):
    theta = 0.0001
    while True:
        delta = 0
        for s in states:
            a = policy[s]
            next_s, reward = T[s][a]
            v = V[s]
            V[s] = reward + gamma * V[next_s]
            delta = max(delta, abs(v - V[s]))
        if delta < theta:
            break
    return V
def policy_improvement(V, policy):
    stable = True
    for s in states:
        old_action = policy[s]
```

```
        action_values = []
 # Evaluate each action
        for a in actions:
            next_s, reward = T[s][a]
            action_values.append(reward + gamma * V[next_s])

        # Choose best action
        best_action = np.argmax(action_values)
        policy[s] = best_action

        if best_action != old_action:
            stable = False
    return policy, stable

def policy_iteration():
    V = [0, 0, 0, 0]      # Initialize state values
    policy = [0, 0, 0, 0] # Random initial policy

    while True:
        V = policy_evaluation(policy, V)
        policy, stable = policy_improvement(V, policy)
        if stable:
            break
    return V, policy
# VALUE ITERATION
def value_iteration():
```

```
    V = [0, 0, 0, 0]
    theta = 0.0001
while True:
        delta = 0
        for s in states:
            v = V[s]
            action_values = []
 for a in actions:
                next_s, reward = T[s][a]
                action_values.append(reward + gamma * V[next_s])

            V[s] = max(action_values)
            delta = max(delta, abs(v - V[s]))

        if delta < theta:
            break
 # Extract policy
    policy = [0, 0, 0, 0]
    for s in states:
        action_values = []
        for a in actions:
            next_s, reward = T[s][a]
            action_values.append(reward + gamma * V[next_s])
        policy[s] = np.argmax(action_values)

    return V, policy
```

```
# MAIN PROGRAM
print("\n==== POLICY ITERATION ====")
V_pi, P_pi = policy_iteration()
print("Optimal Values:", V_pi)
print("Optimal Policy:", P_pi)

print("\n==== VALUE ITERATION ====")
V_vi, P_vi = value_iteration()
print("Optimal Values:", V_vi)
```

Output :

```
==== POLICY ITERATION ====
Optimal Values: [0.9, 1.0, 1.0, 0.0]
Optimal Policy: [np.int64(0), np.int64(0), np.int64(1), np.int64(0)]

==== VALUE ITERATION ====
Optimal Values: [0.9, 1.0, 1.0, 0.0]
Optimal Policy: [np.int64(0), np.int64(0), np.int64(1), np.int64(0)]
PS C:\Users\Achin\OneDrive\Desktop\college work>
```