**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Department of Aritificial Intelligence**

Program: B.Tech/MBA Tech AI Semester: VII

**Experiment No.03**

PART A

(PART A: TO BE REFFERED BY STUDENTS)

**A.1 Aim: Understand and apply the concepts of Markov Decision Processes (MDPs) by modelling a real-world decision-making scenario**

1. Model decision making process
2. Implement Bellman Optimality Equation for value iteration algorithm
3. Identify the optimal policy by analysing the results

**A.2 Prerequisite:**

Concept of Markov property, Markov chain, Markov Decision Process

**A.3 Learning Outcome:**

After completing this experiment you will be able to-

* Model real world problems in terms of MDP
* Identify the optimal policy for real world scenario

**A.4 Theory:**

**A.4.1 Markov Decision Process**

A **Markov Decision Process (MDP)** is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of a decision-maker. MDPs provide a formalized way of dealing with situations where decisions need to be made sequentially over time, under uncertainty.

**A.4.2 Key Components of MDPs**

An MDP is defined by the following components:

1. **States (S)**:
   * The set of all possible situations or configurations in which the decision-maker can find themselves. Each state s∈S provides a complete description of the system at a particular time.
2. **Actions (A)**:
   * The set of all possible actions that the decision-maker can take. For each state s∈S, there is a set of actions A(s) available. An action a∈A(s) chosen in a given state influences the next state of the system.
3. **Transition Probabilities** 
   * The transition probability P(s′∣s,a) represents the likelihood of moving from state s to state s′ when action a is taken. This property embodies the Markov property, which asserts that the future state s′ depends only on the current state s and action a, not on the history of past states or actions.
4. **Rewards (R(s,a,s′):**
   * The reward function R(s,a,s′) specifies the immediate gain or cost received after transitioning from state s to state s′ via action a. This reward guides the decision-making process, as the goal is usually to maximize the cumulative reward over time.
5. **Policy (π)**:
   * A policy π is a strategy or rule that specifies the action π(s) to be taken in each state s. The policy can be deterministic (mapping each state to a specific action) or stochastic (assigning probabilities to each action in a state).
6. **Discount Factor (γ)**:
   * The discount factor γ, where 0≤γ<1, determines the importance of future rewards. A smaller γ means the decision-maker values immediate rewards more highly, while a γ closer to 1 places more emphasis on future rewards.

**A4.1.2 Problem overview**

We want to model a decision-making process where an individual must decide whether to take an umbrella or not based on the weather conditions. The weather can be in one of three states: **Rainy**, **Sunny**, or **Cloudy**. The actions available are either **Take Umbrella** or **Don't Take Umbrella**. The objective is to determine the optimal policy that minimizes discomfort (e.g., getting wet) and maximizes convenience (e.g., not carrying an umbrella unnecessarily).

**States: Rainy, Sunny, Cloudy**

**Actions: Umbrella, No Umbrella**

**Transition Probabilities:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Next** | **Rainy** | **Sunny** | **Cloudy** |
| **Current** |  |  |  |
| **Rainy** | **0.6** | **0.2** | **0.2** |
| **Sunny** | **0.1** | **0.7** | **0.2** |
| **Cloudy** | **0.3** | **0.5** | **0.2** |

**Rewards :**

The reward function reflects the immediate gain or loss from taking an action in a given state. The rewards are designed to encourage carrying an umbrella when it rains and discourage carrying it when it is unnecessary.

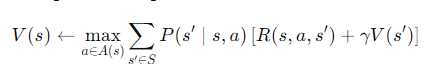
Example of Rewards

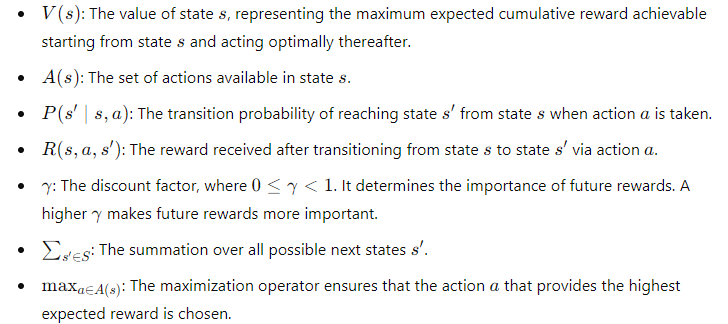
R(Rainy, Umbrella, Rainiy)=+1 (to stay dry)

R(Rainy, No Umbrella, Rainiy)= -10 (you will get wet)

**Bellman Equation:**

**For each state the value is updated as follows-**

****

****

**A.5 Task to be completed:**

1. Model the above scenario
2. Implement Bellman Equation for Policy Iteration
3. Identify the Optimal Policy

**References**

**https://youtu.be/UuTkioxL9bQ?feature=shared**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal/MS Teams assignment link at the end of the practical)**

|  |  |
| --- | --- |
| Roll No.C050 | Name: Nisha Kini |
| Program : BTI | Division: B |
| Batch: B2 | Date of Experiment: 15/1/25 |
| Date of Submission: 15/1/25 | Grade : |

**B.1 Tasks given in PART A to be completed here**

import numpy as np

states = ['Rainy', 'Sunny', 'Cloudy']

actions = ['Umbrella', 'No Umbrella']

transition\_probabilities = np.array([

    [0.6, 0.2, 0.2],

    [0.1, 0.7, 0.2],

    [0.3, 0.5, 0.2]

])

reward\_values = {

    'Rainy': {'Umbrella': 1, 'No Umbrella': -10},

    'Sunny': {'Umbrella': -1, 'No Umbrella': 1},

    'Cloudy': {'Umbrella': 0, 'No Umbrella': 0}

}

value\_function = np.zeros(len(states))

current\_policy = ['No Umbrella'] \* len(states)

discount\_factor = 0.9

convergence\_tolerance = 0.1

max\_iterations = 100

def update\_value\_function(value\_function, transition\_probabilities, reward\_values, discount\_factor, states, actions):

    updated\_value\_function = np.copy(value\_function)

    for i, state in enumerate(states):

        state\_values = []

        for action in actions:

            expected\_value = 0

            for j, next\_state in enumerate(states):

                expected\_value += transition\_probabilities[i][j] \* (reward\_values[state][action] + discount\_factor \* value\_function[j])

            state\_values.append(expected\_value)

        updated\_value\_function[i] = max(state\_values)

    return updated\_value\_function

def improve\_policy(value\_function, transition\_probabilities, reward\_values, discount\_factor, states, actions):

    new\_policy = []

    for i, state in enumerate(states):

        action\_values = []

        for action in actions:

            expected\_value = 0

            for j, next\_state in enumerate(states):

                expected\_value += transition\_probabilities[i][j] \* (reward\_values[state][action] + discount\_factor \* value\_function[j])

            action\_values.append(expected\_value)

        best\_action = actions[np.argmax(action\_values)]

        new\_policy.append(best\_action)

    return new\_policy

converged = False

iterations = 0

while not converged and iterations < max\_iterations:

    updated\_value\_function = update\_value\_function(value\_function, transition\_probabilities, reward\_values, discount\_factor, states, actions)

    delta = np.max(np.abs(updated\_value\_function - value\_function))

    value\_function = updated\_value\_function

    new\_policy = improve\_policy(value\_function, transition\_probabilities, reward\_values, discount\_factor, states, actions)

    if new\_policy == current\_policy and delta < convergence\_tolerance:

        converged = True

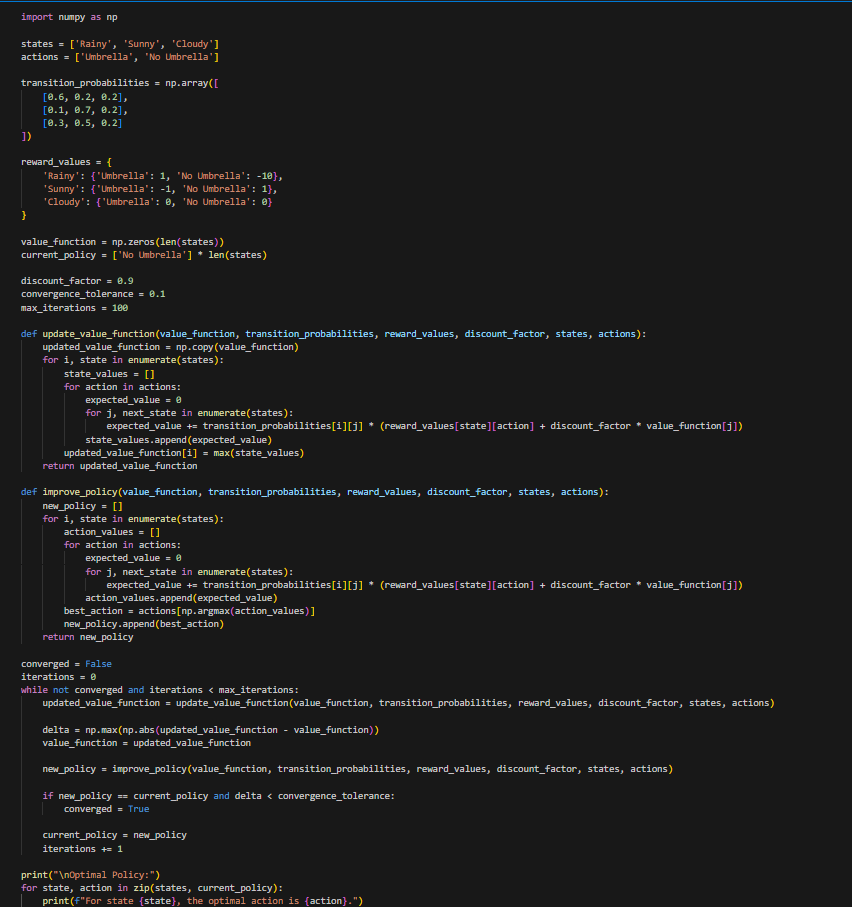
    current\_policy = new\_policy

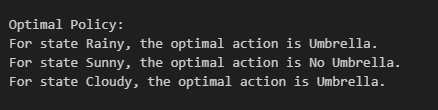
    iterations += 1

print("\nOptimal Policy:")

for state, action in zip(states, current\_policy):

    print(f"For state {state}, the optimal action is {action}.")

****

****

**B.2 Observations and Learning:**

 **Convergence of Value Function**: The value function converges after several iterations, as expected in a typical policy iteration process. In each iteration, the value function is updated based on the Bellman equation, reflecting the expected cumulative rewards from each state. The convergence tolerance determines how precise the final value function should be.

 **Policy Improvement**: After each value function update, the policy is improved by selecting the action that maximizes the expected future reward for each state. This iterative process of policy evaluation and improvement ensures that the policy gradually approaches the optimal solution.

**B.3 Conclusion:**

## In conclusion, this experiment demonstrated the application of a Markov Decision Process (MDP) to model a real-world decision-making problem, specifically the decision of whether to carry an umbrella based on weather conditions. Through the policy iteration algorithm, we were able to identify the optimal policy that minimizes discomfort (e.g., getting wet) while maximizing convenience (e.g., not carrying an umbrella unnecessarily).