Faculty of Sciences

Faculty of Mathematics, Statistics, and Computer Science

Stochastic Processes Mini Project 1

Due Date: Sunday, 20th AprilM

Probability theory is nothing but common sense reduced to calculation.

Pierre-Simon Laplace (1749–1827)

Introduction to Decision-Theoretic Planning

This mini-project introduces the basic idea of Markov Decision Processes (MDPs), which are the foundation of reinforcement learning. Through a simple grid-world environment, we explore how an agent makes decisions to maximize long-term rewards under different terminal state assumptions.

Markov Reward Process (MRP)

You encounter the *Markov Chain* in the course as a model for sequences of states with stochastic transitions. A *Markov Reward Process* (MRP) is a generalization of this model that incorporates a reward structure. It allows for evaluating not only how states evolve over time, but also the expected returns associated with those transitions a key idea in reinforcement learning and decision-theoretic planning.

Definition

A **Markov Reward Process** is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{P}$ is the state transition probability matrix:
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s].$$
- $\mathcal{R}$ is a reward function:
$$\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s].$$
- γ is a discount factor, where $\gamma \in [0, 1]$.

Definition

The **return** G_t is the total discounted reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards.
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This formulation values immediate reward over delayed reward:
  - γ close to 0 leads to "myopic" evaluation.
  - γ close to 1 leads to "far-sighted" evaluation.

We now define the *state value function* $v(s)$, which serves as a measure of the long-term value of a given state s in a Markov Reward Process. Intuitively, it captures the expected return when starting from state s and following the dynamics of the MRP thereafter.

The **state value function** $v(s)$ of an MRP is the expected return starting from state s:
$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

The value function $v(s) = \mathbb{E}[G_t \mid S_t = s]$ captures the expected return starting from state s.

Task 1. Derive a recursive expression for the value function $v(s)$ by breaking down the return G_t into two parts:

- the immediate reward received after being in state s
- the discounted value of the successor state

Use the definition of return, the Markov property, and properties of expectation to express $v(s)$ in terms of itself.

This leads to a recursive formulation known as the *Bellman Equation for MRP*.

Task 2. Express the recursive formulation from Task 1 in matrix form.

- Define a vector containing the value of each state.
- Represent transitions between states using a matrix.
- Write the equation that relates these components linearly.

Solve the resulting linear system to compute the value function for all states.
In other words, solving the Bellman equation in matrix form yields an explicit expression for the state value vector $\mathbf{v}$ in terms of the reward vector and the transition dynamics of the process.

Markov Decision Process (MDP)

A *Markov Decision Process* (MDP) extends a Markov Reward Process by introducing *actions*. In an MDP, the agent makes decisions that influence both the next state and the received reward. It provides a mathematical framework for modeling sequential decision-making in stochastic environments where all states satisfy the Markov property.

Definition

A **Markov Decision Process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function:
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a].$$
- $\mathcal{R}$ is a reward function:
$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a].$$
- $\gamma \in [0, 1]$ is a discount factor.

Important Note. In a grid-based environment, each action (e.g., *up*, *down*, *left*, *right*) defines its own transition probability distribution over the next states. That is, the probability of reaching a specific successor state depends not only on the current state but also on the action taken.

This means that for each action $a \in \mathcal{A}$, there is a separate transition matrix $\mathbf{P}^a \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$, where the entry $\mathbf{P}_{ss'}^{a}$ represents the probability of transitioning from state s to state s' when action a is taken.

In contrast to Markov Reward Processes (MRPs), where the transition dynamics are governed by a single matrix $\mathbf{P}$, MDPs maintain one transition matrix per action. This structure is essential for defining how an agent's behavior (via a policy) influences state evolution and expected rewards.

Policy: In a MDP, a *policy* defines the agent's behavior by specifying how actions are selected in each state. A policy maps each state to a probability distribution over actions, describing the likelihood of taking each action in that state.

Definition

A **policy** π is a distribution over actions given states:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Important properties of policies:

- A policy fully defines the behavior of an agent.
- In MDPs, policies depend only on the current state not on the full history.
- Policies can be:
  - *Deterministic*, selecting one specific action per state.
  - *Stochastic*, assigning a probability distribution over possible actions.

Once a policy π is fixed, the MDP behaves like a Markov Reward Process (MRP). That is:

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π,
- The resulting state sequence $S_1, S_2, \ldots$ forms a Markov process defined by $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$,
- And the statereward sequence $S_1, R_1, S_2, R_2, \ldots$ forms a Markov Reward Process:

$$\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$$

where:

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, \mathcal{P}^{a}_{ss'} \quad \text{and} \quad \mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, \mathcal{R}^{a}_{s}$$

These define the transition and reward dynamics under policy π, allowing the MDP to be evaluated as an MRP a critical step in policy evaluation and value function analysis.

Once an agent follows a fixed policy π, we can evaluate how good each state is under that policy. This is captured by the *state-value function*.

Definition
The **state-value function** $v_{\pi}(s)$ of an MDP is the expected return starting from state s, and then following policy π:

$$v_{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right]$$

The **action-value function** $q_{\pi}(s, a)$ is the expected return starting from state s, taking action a, and then following policy π:

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right]$$

There exists a fundamental relationship between the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$, expressed by the following two equations:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a\, v_\pi(s')$$

Substituting the second equation into the first yields the *Bellman Expectation Equation* for $v_\pi(s)$, which recursively expresses the value function as:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a\, v_\pi(s') \right)$$

This equation plays a central role in the evaluation of policies and forms the basis of many reinforcement learning algorithms.

For a detailed explanation of why this decomposition holds, see here.

Ultimately, our goal in solving a Markov Decision Process is to find the *optimal state-value function* $v_*(s)$, which gives the maximum expected return achievable from each state under any policy.

Definition

The **optimal state-value function** $v_*(s)$ is the maximum value function over all policies:

$$v_*(s) = \max_\pi v_\pi(s)$$

The **optimal action-value function** $q_*(s, a)$ is the maximum action-value function over all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Once the optimal state-value function $v_*(s)$ or the optimal action-value function $q_*(s, a)$ is known, it can be used to extract an optimal policy

Solving for the optimal value function $v_*(s)$ analytically is generally intractable, especially in environments with large state spaces. Instead, we use iterative methods to approximate $v_*(s)$. One of the most fundamental algorithms for this purpose is *Value Iteration*.

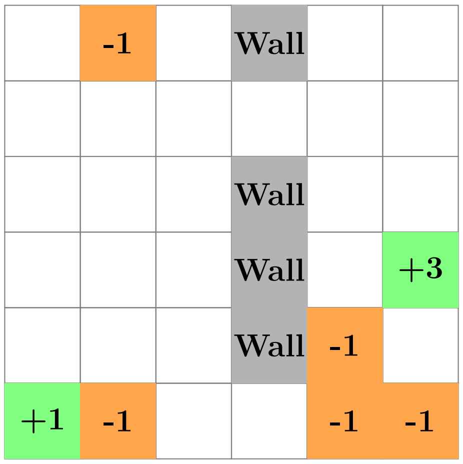
Task 3. Investigate how the **Value Iteration** algorithm works to compute the optimal state-value function $v_*(s)$. Formally describe how it iteratively updates estimates of $v(s)$ based on the Bellman optimality principle, and explain its convergence properties.

Practical Part: Grid World Environment

Now that we have studied the theoretical foundations of Markov Decision Processes, our goal is to implement these ideas in a simplified environment.

In this part, we work with a *Grid World*, where each cell corresponds to a unique state. Some of these states provide fixed rewards (positive or negative), while others represent impassable *walls*. The agent can move in four directions: up, down, left, or right unless blocked by a wall or the edge of the grid.

Below is the grid-world environment we will use throughout this practical section:



The environment is defined over a 6×6 grid, where each cell corresponds to a state:

- **White cells** represent normal states with a default reward of -0.04 per step.
- **Green cell** provides an immediate reward of $+3$ or $+1$ upon entry.
- **Orange cells** are states with an immediate reward of -1 upon entry.
- **Gray cells** are walls and cannot be entered; if the agent attempts to move into a wall, it remains in the same state.

The agent can take one of four actions in each state: `up`, `down`, `left`, or `right`. The environment follows a **uniform stochastic transition model**: at each time step, the agent transitions uniformly at random to one of the neighboring directions, regardless of the intended action. That is, each of the four directions is executed with probability $\frac{1}{4}$. If a selected transition would lead to an invalid state (e.g., into a wall or outside the grid), the agent remains in its current state with no movement and receives **-0.04 reward** for that step.

Task 4. Implement the Grid World environment by defining the following components:

- The state space $\mathcal{S} \subset \mathbb{Z}^2$
- The action space $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$
- The reward function $\mathcal{R}(s)$, including special cases for green and orange cells
- The discount factor $\gamma \in [0, 1]$
- The transition probability function $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$ for each action

Task 5. Using the **Value Iteration** algorithm, compute the optimal state-value function $v_*(s)$ for all states in the grid. Iterate until the value function converges within a small tolerance (e.g., 10^{-4}).

Task 6 (Bonus). Using the **action-value function** concept introduced earlier, derive the **optimal policy** $\pi_*(s)$. For each state, determine the action that maximizes the expected return by evaluating:

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}} q_*(s, a)$$

where $q_*(s, a)$ is computed using the optimal value function $v_*(s)$.

Implementation Notes.

A code template in the form of a Jupyter notebook is available on the eLearn platform. You may use this template to structure your implementation, but its use is not mandatory. It includes function headers and structural guidance aligned with the tasks defined in this project.

You should submit your code and any additional required files by uploading a single compressed `.zip` file via eLearn. The file name must follow the format:

```
MiniProject1_SP_lastname_studentid.zip
```