

Mini-2

Preliminary

Augment your abstract classes you worked on in Mini Project 1 in the following way:

1. Create a class for Markov Decision Processes (MDPs) that takes advantage of how you defined Markov Reward Processes.
2. For each problem, formulate the situation as an MDP. Clearly define:
 - the initial measure, μ ; that is, the starting positions of your agent.
 - The policy $\pi: \mathcal{S} \times \Sigma_{\mathcal{A}} \rightarrow [0, 1]$. Or equivalently, the action dynamics $\pi(a | s)$
 - The kernel $\kappa: \mathcal{S} \times \mathcal{A} \times \Sigma_{\mathcal{S}} \rightarrow [0, 1]$. Or equivalently, the transition dynamics $P(s' | s, a)$.
 - The reward kernel $R: \mathcal{S} \times \mathcal{B}(\mathbb{R}) \rightarrow [0, 1]$. Or equivalently, The reward function $\mathcal{R}(s) = \mathbb{E}(R_{t+1} | S_t = s)$.¹
 - γ , returns, values and q -values.
 - Policy iteration and value iteration.

Problem 1: Navigating A Windy Chasm

A drone must navigate through a windy chasm from a start position to a goal position. The environment is a 20×7 grid where the drone starts at position $(0, 3)$ and must reach the exit of the chasm defined by $(19, n)$ for any of the 7 possible choices for n .

Environment Dynamics

The drone has three actions available: **forward** (increase i by 1), **left** (decrease j by 1), and **right** (increase j by 1). However, wind currents create stochastic transitions:

- When the drone is at position (i, j) and takes an action, it first moves deterministically according to that action.
- After the action, wind effects apply based on the distance from the center line ($j = 3$):
 - If the drone ends at $(i, 3)$: with probability p , the wind pushes it to either $(i, 4)$ or $(i, 2)$ (50/50); otherwise, with probability $(1 - p)p^2$, it pushes to $(i, 5)$ or $(i, 1)$; otherwise, with probability $(1 - p)(1 - p^2)$, it stays at $(i, 3)$.
 - If the drone ends at (i, j) where $j \neq 3$: the wind probabilities scale with distance from center. Specifically, p is a function of j by the formula

$$p(j) = B^{E(j)} \quad \text{where} \quad E(j) = \frac{1}{1 + (j - 3)^2} \quad \text{and} \quad B = p(3)$$

- If the drone reaches $j \geq 6$ or $j \leq 0$, the episode terminates (crash).
- Reaching position $(19, n)$ terminates the episode successfully.

¹Note that R_{t+1} by convention refers to the reward RV for the current state due to when time steps are updated. In particular, if your rewards have zero variance, given a state, then the rewards are what you expect from the grid worlds.

Rewards

- Reaching the goal $(19, n)$: $+R$
- Crashing ($j \geq 6$ or $j \leq 0$): $-r \leq -1$
- Each step: -1

Questions

1. Extract the optimal policy $\pi^*(s)$ from your value function. Describe the qualitative behavior of the optimal policy: does the drone prefer to stay near the center, or does it take risks? How does the policy change as the wind parameter p increases? What about setting the crashing reward to -1 ?
2. **CHALLENGE:** As you all grow more comfortable with finite grids, you will find they are analogous to finite sums when learning about integration. They are approximations of the real thing. The following are adaptations of increasing complexity and increasing value to real word implementation.
 - (a) Take as your state space a 60×21 grid by cutting up every tile into 9 pieces. Is there a way to adapt p and rewards so that you get a fine-grained version of the 20×7 case?
 - (b) Take a limit so that you have a well defined state space on $[0, 6] \times [0, 19]$. This is impossible to implement without a closed form, but you can still use a good approximation. The dynamics exist in the horizontal direction so focus on interpolating here. Use $dx = .05$ and $dy = 1$ and increase p if the drone is winning too easily.
 - (c) Instead of time steps, define a fixed velocity v and continuous time. It follows that distance covered is scaled by v .
 - (d) Define the probability that the wind will blow by:

$$\int_j^{j+tv} p(x)dx + p(j) \int_i^{i+v(1-t)} dy = \frac{p(j) + p(j+vt)}{2}t + p(j)(1-vt).$$

Where vt is the amount of distance one time step in the 20×7 world would take, and tv is the portion dedicated to horizontal movement. This formula is a linear interpolation of the discrete case. In real life you would integrate over trajectories much like line integrals.

- (e) Instead of pushing discrete amounts, interpolate the consequences of wind effects instead. Start with a drift in the form of a forced velocity of v away from $j = 3$ for an interval of time $[0, 1]$. Consider what you should do if another environment effect triggers during the consequences of a prior one. If a p^2 event happens, instead change the velocity to $2v$ for the same time interval.
- (f) Consider adding angles to this procedure. How would the wind dynamics change? The tricky part is to determine how to implement it given the choices in subdivisions of dx and dy .

Problem 2: Robot Motion Control via Reward Shaping

Consider an autonomous ground vehicle robot equipped with wheel encoders and a 360-degree lidar sensor. Due to hardware imperfections, motor noise, slip, lag, and control loop oscillations, commanding the robot to move forward rarely results in perfectly straight motion. Luckily, lidar can be far more reliable than encoders. Your goal is to use reinforcement learning to finetune movement.

Setup

The robot uses a Kalman filter to estimate its 2D pose (r, θ) from lidar scans and odometry. The state space consists of discretized lidar readings.

A clever environment design is to use a cone-shaped corridor in front of the robot: think of it like a “funnel” where the robot’s forward direction is aligned with the axis of symmetry of the cone.

- If the robot moves straight, lidar readings in the left and right sectors of the forward view are identical in expectation, radially and angularly.
- If the robot veers, one sector’s displacement and angles shrink while the other sector’s displacements and angles grow, giving a direct geometric signal that the robot is off-course.

The action space consists of discrete adjustments to the left and right wheel motor encoder thresholds, representing small increments or decrements to threshold values.

Questions

- (a) Propose a specific state representation that captures data from the lidar and the robot’s deviation from straight-line motion. Justify why this representation is Markovian (or approximately so) for the motion control task. If you have no idea what to do, you can start off with the grid:

$$\begin{bmatrix} W \\ 1 & W \\ 1 & 1 & W \\ 1 & 1 & 1 & W \\ 1 & 1 & 1 & 1 & W \\ 1 & 1 & 1 & 1 & 1 & W \\ A & 1 & 1 & 1 & 1 & 1 & W \\ 1 & 1 & 1 & 1 & 1 & W \\ 1 & 1 & 1 & 1 & W \\ 1 & 1 & 1 & W \\ 1 & 1 & W \\ 1 & W \\ W \end{bmatrix}$$

and consider how to turn asymmetry into a problematic state.

- (b) Design a reward function $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defined by $\mathcal{R}(s, a) = \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a)$ that encourages forward motion and rotation corrections regardless of your encoder hardware.

- (c) Apply policy iteration to this problem. Discuss potential challenges with convergence: What aspects of the problem (continuous underlying state, sensor noise, hardware variability) might make the empirical transition probabilities $P(s'|s, a)$ difficult to estimate accurately? How might you address these challenges?

Problem 3: Circuit Design via Sequential Decisions

An agent must design a digital logic circuit on a 3×3 grid to implement a target Boolean function. The agent navigates the grid and places components to build the circuit from left to right.

Environment

The agent's position (i, j) on the grid is part of the state, along with the current circuit configuration. To avoid needless complexity, we impose the requirement that gates have fixed orientation: inputs come from the left, top, and bottom, and outputs go to the right. We require, for simplicity, that $(0, 0)$ and $(2, 0)$ are special source tiles that can inject input signals A and B into whatever component is placed on them. Moreover, that $(1, 2)$ is a special sink, C , that absorbs signals output from components. Agent actions include:

- **Move:** up, down, left, right (within grid bounds)
- **Place:** wire, NAND gate (at current position)
- **Remove:** remove component (at current position)
- **Done:** finish and evaluate the circuit

Goal

Have a framework for an agent that designs a circuit that implements the 2-bit XOR function: In particular, the signals A and B give the output a signal $C = A \text{ XOR } B$

Questions

- (a) Define a suitable state representation for this problem. What information must be included in the state to make optimal decisions? Discuss whether your state representation makes the problem Markovian.
- (b) Propose a way of evaluating the test time action. How would you go about judging the propagation of the A and B signals fairly? Would you treat this as a deterministic process or use some other way?
- (c) A 3×3 grid is very small and leads to unintended wire interactions. However, it is very necessary due to the state space explosion that happens when you introduce so many actions. Impose “insulation” from $(1, 1)$ to $(2, 1)$ that blocks all signals between them.
- (d) Analyze the complexity of applying value iteration to this type of problem. Estimate the size of the state space just for going to the 4×4 case. and discuss whether standard tabular value iteration is computationally feasible. What approximations or problem modifications might make dynamic programming more tractable?