assignment1_latex_safe

January 22, 2026

1 Stanford CME 241 (Winter 2026) - Assignment 1

Due: Friday, January 23 @ 11:59 PM PST on Gradescope.

Assignment instructions: - Make sure each of the subquestions have answers - Ensure that group members indicate which problems they're in charge of - Show work and walk through your thought process where applicable - Empty code blocks are for your use, so feel free to create more under each section as needed - Document code with light comments (i.e. 'this function handles visualization')

Submission instructions: - When complete, fill out your publicly available GitHub repo file URL and group members below, then export or print this .ipynb file to PDF and upload the PDF to Gradescope.

Link to this ipynb file in your public GitHub repo (replace below URL with yours):

https://github.com/onat-dalmaz/RL-book/blob/main/assignment1.ipynb

Group members (replace below names with people in your group): - Onat Dalmaz

1.1 Imports

[1]:
```
from __future__ import annotations

from typing import Dict, Mapping, List, Tuple
import numpy as np
import matplotlib.pyplot as plt

# RL-book imports (installed via Assignment 0). If these fail, ensure you
  activated the same venv.
from rl.distribution import Categorical, Constant
from rl.markov_process import FiniteMarkovProcess, NonTerminal, Terminal
```

1.2 Question 1: Snakes and Ladders

In the classic childhood game of Snakes and Ladders, all players start to the left of square 1 (call this position 0) and roll a 6-sided die to represent the number of squares they can move forward. The goal is to reach square 100 as quickly as possible. Landing on the bottom rung of a ladder allows for an automatic free-pass to climb, e.g. square 4 sends you directly to 14; whereas landing on a snake's head forces one to slide all the way to the tail, e.g. square 34 sends you to 6. Note, this game can be viewed as a Markov Process, where the outcome is only depedent on the current state and not the prior trajectory. In this question, we will ask you to both formally describe the Markov

Process that describes this game, followed by coding up a version of the game to get familiar with the RL-book libraries.

1.2.1 Problem Statement

How can we model this problem with a Markov Process?

1.2.2 Subquestions

Part (A): MDP Modeling Formalize the state space of the Snakes and Ladders game. Don't forget to specify the terminal state!

Part (B): Transition Probabilities Write out the structure of the transition probabilities. Feel free to abbreviate all squares that do not have a snake or ladder.

Part (C): Modeling the Game Code up a `transition_map: Transition`

$$S$$

data structure to represent the transition probabilities of the Snakes and Ladders Markov Process so you can model the game as an instance of `FiniteMarkovProcess`. Use the `traces` method to create sampling traces, and plot the graph of the distribution of time steps to finish the game. Use the image provided for the locations of the snakes and ladders.

https://drive.google.com/file/d/1yhP242sG092Ico_WOPKrUp8jVJHbuGHH/view?usp=sharing

---

1.2.3 Part (A) Answer

A convenient Markov Process model uses one state per board position.

- **State space**: (S = {0,1,2,…,100}), where state (s) means "the token is on square (s)".
- **Terminal state**: (100) is terminal (absorbing): once you reach square 100, the game ends.

To incorporate snakes/ladders, define a deterministic "jump" function (J:{0,…,100}→{0,…,100}) that maps a square to its post-jump destination (identity on ordinary squares, head->tail for snakes, bottom->top for ladders). Then the one-step evolution is: roll a die $D \in \{1, \ldots, 6\}$, move to $\min(100, s + D)$, then apply J.

1.2.4 Part (B) Answer

Let $D \sim \text{Unif}\{1, 2, 3, 4, 5, 6\}$. For any non-terminal square $s \in \{0, \ldots, 99\}$, define

$$s' = J(\min(100, s + D)).$$

Then the transition probabilities are

$$\mathbb{P}[X_{t+1} = x \mid X_t = s] = \frac{1}{6}|\{d \in \{1, \dots, 6\} : J(\min(100, s + d)) = x\}|.$$

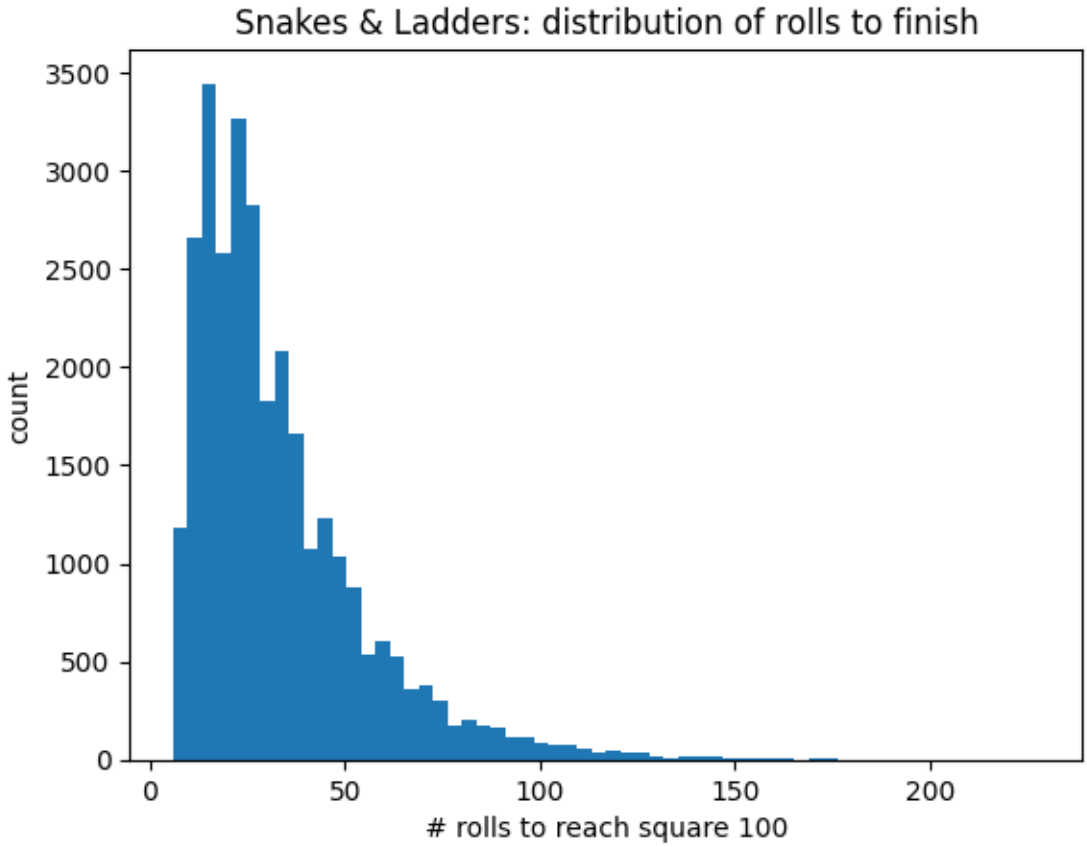
For the terminal state:

$$\mathbb{P}[X_{t+1} = 100 \mid X_t = 100] = 1.$$

Equivalently: **each die face contributes probability (1/6)** to the single next square produced by "move then jump". Squares without snakes/ladders simply have (J(u)=u).

1.2.5 Part (C) Answer

```
samples=30000  mean=33.24  median=27  p90=62
```



1.3 Question 2: Markov Decision Processes

Consider an MDP with an infinite set of states $S = \{1, 2, 3, ...\}$. The start state is $s = 1$. Each state s allows a continuous set of actions $a \in [0, 1]$. The transition probabilities are given by:

$$\mathbb{P}[s + 1 \mid s, a] = a, \mathbb{P}[s \mid s, a] = 1 - a \text{ for all } s \in S \text{ for all } a \in [0, 1]$$

For all states $s \in S$ and actions $a \in [0, 1]$, transitioning from s to $s + 1$ results in a reward of $1 - a$ and transitioning from s to s results in a reward of $1 + a$. The discount factor $\gamma = 0.5$.

1.3.1 Problem Statement

How can we derive a mathematical formulation for the value function and the optimal policy? And how do those functions change when we modify the action space?

1.3.2 Subquestions

Part (A): Optimal Value Function Using the MDP Bellman Optimality Equation, calculate the Optimal Value Function $V^*(s)$ for all $s \in S$. Given $V^*(s)$, what is the optimal action, a^*, that maximizes the optimal value function?

Part (B): Optimal Policy Calculate an Optimal Deterministic Policy $\pi^*(s)$ for all $s \in S$.

Part (C): Changing the Action Space Let's assume that we modify the action space such that instead of $a \in [0, 1]$ for all states, we restrict the action space to $a \in [0, \frac{1}{s}]$ for state s. This means that higher states have more restricted action spaces. How does this constraint affect:

- The form of the Bellman optimality equation?
- The optimal value function, $V^*(s)$?
- The structure of the optimal policy, $\pi^*(s)$?

1.3.3 Part (A) Answer

Bellman optimality ((=0.5)) for state (s) is

$$V\widehat{} * (s) = \max\_\{a \in [0, 1]\}\Big(a\big[(1 - a) + \gamma V^*(s + 1)\big] + (1 - a)\big[(1 + a) + \gamma V^*(s)\big]\Big).$$

Expanding:

$$V\widehat{} * (s) = \max\_\{a \in [0, 1]\}\Big(1 + a - 2a\widehat{}2 + \gamma\big[V^*(s) + a(V^*(s + 1) - V^*(s))\big]\Big).$$

Because the dynamics/rewards are the same at every state (no boundary/terminal state), the optimal value is translation-invariant, so we take (V$^{(s)=V)\ for\ all\ (s).\ Then\ (V\widehat{}}$(s+1)-V$*$(s)=0) and

$V = \max\{a\ [0,1]\}\ (1+a\text{-}2a\widehat{}2 +\ \ V)$ $\quad$ $(1\text{-}\ )V = \max\{a\ [0,1]\}\ (1+a\text{-}2a\widehat{}2 +\ V)$ $\quad$ (1-)V = max $\{a\ [0,1]\}\ (1+a$

Maximizing the concave quadratic (f(a)=1+a-2a$\widehat{}$2):

$$f'(a) = 1 - 4a = 0 \Rightarrow a\widehat{}=\tfrac{1}{4}, \quad f(a\widehat{}=\tfrac{1}{4}, \quad f(a\widehat{}=\tfrac{1}{4}, \quad f(a\widehat{}=\tfrac{1}{4},\ \ f(a\widehat{}) = 1 + \tfrac{1}{4} - 2 \cdot \tfrac{1}{16} = \tfrac{9}{8} = 1.125.$$

So

$$V = \frac{f(a^*)}{1-\gamma} = \frac{9/8}{1/2} = \frac{9}{4} = 2.25,$$

i.e. ($\boxed{V^*(s) = 2.25 \ \forall s}$), and the maximizing action is ($\boxed{a^* = 1/4}$).

1.3.4 Part (B) Answer

An optimal deterministic policy takes the same action in every state:

$$\boxed{\pi^*(s) = \tfrac{1}{4} \quad \forall s \in \{1, 2, 3, ...\}.}$$

1.3.5 Part (C) Answer

Bellman Optimality Equation Change: Only the **feasible action set** changes:

$$V\widehat{\ } * (s) = \max\_\{a \in [0,\ 1/s]\}\Big(a\big[(1-a) + \gamma V^*(s+1)\big] + (1-a)\big[(1+a) + \gamma V^*(s)\big]\Big).$$

Optimal Value Function Change: Because the feasible set shrinks with (s), the translation-invariance is broken and (V$\widehat{\ }$*(s)) is no longer constant. Intuitively, larger (s) is "worse" because you have less control, so (V$\widehat{\ }$(s)) decreases with (s) and approaches the baseline value obtained by taking (a 0), namely

$$\lim\_\{s \to \infty\} V\widehat{\ } * (s) = \frac{1}{1-\gamma} = 2.$$

A convenient recursion (for (=0.5)) for any chosen action (a) is:

$$V(s) = \frac{1 + a - 2a^2 + 0.5a\, V(s+1)}{0.5(1+a)}.$$

Under the optimal policy below this yields values close to 2.25 for small (s), decaying toward 2 as (s) grows.

Optimal Policy Change: Unconstrained optimum is (1/4). With the constraint (a 1/s), the policy becomes "as large as allowed" once the constraint binds:

$$\boxed{\pi^*(s) = \min\left(\tfrac{1}{4}, \tfrac{1}{s}\right).}$$

So ($\widehat{\ }$*(s)=1/4) for (s 4), and ($\widehat{\ }$(s)=1/s) for (s 5) (hence smaller and smaller actions at higher states).

1.4 Question 3: Frog in a Pond

Consider an array of $n+1$ lilypads on a pond, numbered 0 to n. A frog sits on a lilypad other than the lilypads numbered 0 or n. When on lilypad i ($1 \le i \le n-1$), the frog can croak one of two sounds: **A** or **B**.

- If it croaks **A** when on lilypad i ($1 \le i \le n-1$):
  - It is thrown to lilypad $i-1$ with probability $\frac{i}{n}$.
  - It is thrown to lilypad $i+1$ with probability $\frac{n-i}{n}$.

- If it croaks **B** when on lilypad i ($1 \leq i \leq n-1$):
  - It is thrown to one of the lilypads $0, \ldots, i-1, i+1, \ldots, n$ with uniform probability $\frac{1}{n}$.

A snake, perched on lilypad 0, will eat the frog if it lands on lilypad 0. The frog can escape the pond (and hence, escape the snake!) if it lands on lilypad n.

1.4.1 Problem Statement

What should the frog croak when on each of the lilypads $1, 2, \ldots, n-1$, in order to maximize the probability of escaping the pond (i.e., reaching lilypad n before reaching lilypad 0)?

Although there are multiple ways to solve this problem, we aim to solve it by modeling it as a **Markov Decision Process (MDP)** and identifying the **Optimal Policy**.

1.4.2 Subquestions

Part (A): MDP Modeling Express the frog-escape problem as an MDP using clear mathematical notation by defining the following components:

- **State Space**: Define the possible states of the MDP.
- **Action Space**: Specify the actions available to the frog at each state.
- **Transition Function**: Describe the probabilities of transitioning between states for each action.
- **Reward Function**: Specify the reward associated with the states and transitions.

Part (B): Python Implementation There is starter code below to solve this problem programatically. Fill in each of the 6 `TODO` areas in the code. As a reference for the transition probabilities and rewards, you can make use of the example in slide 16/31 from the following slide deck: https://github.com/coverdrive/technical-documents/blob/master/finance/cme241/Tour-MP.pdf.

Write Python code that:

- Models this MDP.
- Solves the **Optimal Value Function** and the **Optimal Policy**.

Feel free to use/adapt code from the textbook. Note, there are other libraries that are needed to actually run this code, so running it will not do anything. Just fill in the code so that it could run assuming that the other libraries are present.

Part (C): Visualization and Analysis What patterns do you observe for the **Optimal Policy** as you vary n from 3 to 25? When the frog is on lilypad 13 (with 25 total), what action should the frog take? Is this action different than the action the frog should take if it is on lilypad 1?
