Question 0, Homework 1, CME241

Link to code: https://github.com/nabilah13/RL-book/tree/master/assignment1

State Space $S = \{0, 1, 2, 3, \ldots, 100\}$

In other words, the state space is the set of integers from 0 to 100, inclusive.

This is a lot to write out, so I will just explain the logic of my code.

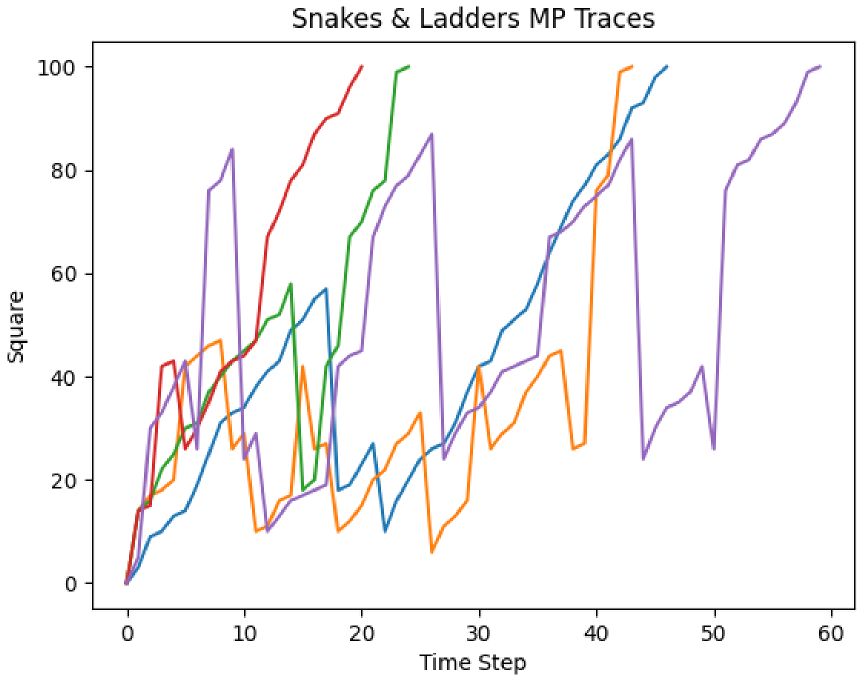
I start by initializing a 101 by 101 matrix of zeros. The final row/column represents the terminal state of winning the game. We start by saying that starting at i=0 and given a state i, states i+1, i+2,...,i+6 receive 1/6 probability corresponding to the sides of a fair die. However, we also don't give probability for state i>100, since that would be out-of-bounds. We adjust the terminal state transition probability upwards for states 95 through 99 since multiple die roll outcomes become associated with winning. Finally, we account for the snakes and ladders by giving all the columnar probability from the start of the snake/ladder to the end of the snake/ladder and then zeroing out the starting square probability. This represents how you can't really have a state equal to the starting square of a snake/ladder since you will immediately teleport to ending square.

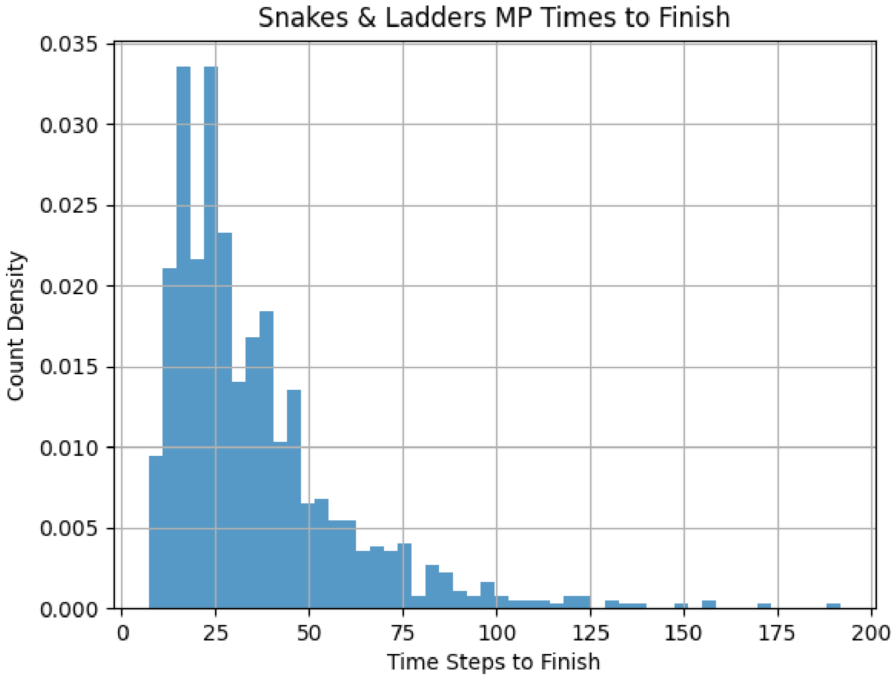
```
# Create transition matrix
# Start with uniform dist of moving 1 to 6 spaces forward
transition_matrix = np.zeros((101,101))
for i in range(100):
    for j in range(i+1, min(i+7, 101)):
        transition_matrix[i, j] = 1/6
# Prob of winning (term state) is 1 minus sum of probs on non-term states
transition_matrix[:,-1] = 1 - np.sum(transition_matrix[:, :-1], axis=1)
# Create mapping for the ladders and snakes
ls_dict = {1:38, 4:14, 8:30, 21:42,
    28:76, 50:67, 71:92, 80:99,
    97:78, 95:56, 88:24, 62:18, 48:26,
    36:6, 32:10}
# Edit transition matrix according to the mapping
for start_sq, end_sq in ls_dict.items():
    transition_matrix[:, end_sq] += transition_matrix[:, start_sq]
    transition_matrix[:, start_sq] = 0
```

Question 1(c), Homework 1, CME241

Done in code.

https://github.com/nabilah13/RL-book/tree/master/assignment1


Snakes & Ladders MP Traces
Square
Time Step
0
20
40
60
80
100
0
10
20
30
40
50
60


Snakes & Ladders MP Times to Finish
Count Density
Time Steps to Finish
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0
25
50
75
100
125
150
175
200

The expected number of rolls to finish the game from the beginning is 34.02594349895675. To calculate this number, we set up a reward function which just deterministically returns 1 for every non-terminal state. We also set gamma to 1.0. This allows us to count the time-steps.

I think of the lilypads as representing integers, so:

State Space $S = \{0, 1, 2, 3, \dots, 10\}$

In other words, the state space is the set of integers from 0 to 10, inclusive.

The transition probabilities are that given that you are at state s, you have uniform transition probability over states s' where s'>s. We can create a transition matrix for the non-terminal states as described by my code:

```
transition_matrix = np.zeros((10, 10))
for i in range(10):
    transition_matrix[i, i+1:] = 1/(10-i)
```

Define the transition matrix $P \in R^{n+1xn+1}$ using the code from part (a). The reward vector $\vec{r}$ is just a vector of ones and is of dimension n+1. We also have the $\vec{e}_1 = [1, 0, \ldots, 0]^T$ of dimension n+1. Then we have:

$$\text{E(hops from start)} = \vec{e}_1^T (I - P)^{-1} \vec{r}$$

So then: Expected number of hops to reach other side: 2.928968253968254

Also shown in code.

Let n be the number of lilypads and y_n be the number of hops needed to get across given n lilypads. The closed form solution is:

$$\mathrm{E}(y_n) = \sum_{j=0}^{n} \frac{1}{j+1}$$

We can prove by induction. When n=0 it is trivial that $\mathrm{E}(y_0) = 1/(0+1) = 1$. It is also easy to show that the closed form solution works for n=1 (proving the base case) by using the following recursion:

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} \sum_{i=0}^{n-1} \mathrm{E}(y_i)$$

To prove the inductive step, we assume that the closed form solution holds for all $i < n$, so then:

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} \sum_{i=0}^{n-1} \sum_{j=0}^{i} \frac{1}{j+1}$$

Reorder the sum:

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} \sum_{j=0}^{n-1} \sum_{i=j}^{n-1} \frac{1}{j+1}$$

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} \sum_{j=0}^{n-1} \frac{n-1-j+1}{j+1}$$

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} \sum_{j=0}^{n-1} \frac{n+1}{j+1} - 1$$

$$\mathrm{E}(y_n) = 1 + \frac{1}{n+1} ((\sum_{j=0}^{n-1} \frac{n+1}{j+1}) - n)$$

$$\mathrm{E}(y_n) = 1 - \frac{n}{n+1} + \sum_{j=0}^{n-1} \frac{1}{j+1}$$

$$\mathrm{E}(y_n) = \frac{1}{n+1} + \sum_{j=0}^{n-1} \frac{1}{j+1}$$

$$\mathrm{E}(y_n) = \sum_{j=0}^{n} \frac{1}{j+1}$$

Thus we prove the closed form solution by induction.

We set up the MDP Bellman Optimality Equation:

$$V^*(s) = \text{max}_{a \in A}(R(s,a) + 0.5 \sum_{s' \in S} P(s,a,s')V^*(s')$$

Now we recognize two facts. Firstly, $P(s,a,s') > 0$ only if $s' = s$ or $s' = s+1$. Next, we realize $R(s,a)$ and $P(s,a,s+1)$ are the same regardless of s. This also means that $V^*(s) = V^*(s')$ for any pairs of s and s'.

$$V^*(s) = \text{max}_{a \in A}(a(1-a) + (1-a)(1+a) + 0.5(aV^*(s) + (1-a)V^*(s)))$$

$$V^*(s) = 2\text{max}_{a \in A}(1 - a - 2a^2)$$

Taking a derivative to maximize this:

$$1 - 4a = 0$$

$$a = 0.25$$

Plugging back in for $V^*(s)$:

$$V^*(s) = 2 * (1 + 0.25 - 2(0.25)^2)$$

$$V^*(s) = 2 * (1.125)$$

$$V^*(s) = 2.25$$

So the optimal deterministic policy is to set $a = 0.25$ which yields an optimal value function of $V^*(s) = 2.25$.