Solutions for HW4

Yunhai Han
Department of Mechanical and Aerospace Engineering
University of California, San Diego
y8han@eng.ucsd.edu

May 12, 2020

Contents

1 Problem 1

1.1 problem formulation

In the file modelFreePrediction, implement the (first-visit) MC simulation (II) using T -long episodes and for M iterations. To do this, fill out the function MCfirstvisit, which takes in a policy π, the random getRandomcost function, and generates MT -long episodes or sampled trajectories. The program outputs the approximated cost-to-go function J^{π}. Using $\pi = "up"$ except when i is the terminal state, in which case we use 0, provide sample vector values that you obtain in the mediumGrid for the values (a) $T = 5, T = 10$, and $T = 100$, (b) $M = 10$ and $M = 50$, and parameter choice (c) $\gamma = 0.5$, and $\gamma = 0.8$. Use the parameter $\eta = 0.25$ in all simulations. What do you observe?

1.2 solutions

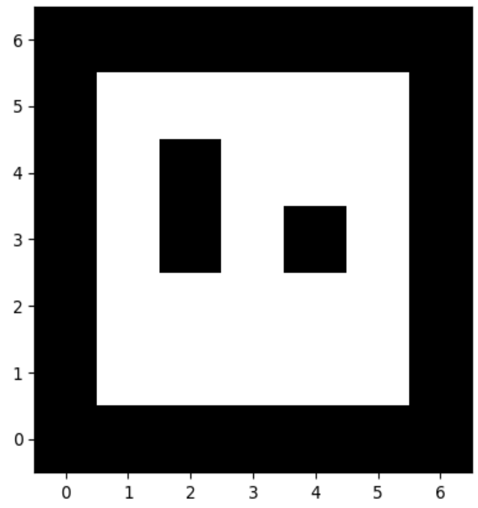


Figure 1: Medium grid map

$T = 5, M = 10, \gamma = 0.5, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & -10 & 0 & -10 \\ -10 & 0 & 0 & 0 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

In the above matrix, girds with value -10 represents the obstacles and grids with value -5 represents **LOSESTATES** and all the other grids are assigned with the initial value 0.

You could see that in this case(with π = "up"), both two **EXIT** can not be detected from the **START**(1,2). And the cost value is zero unless the next state belongs to one of the exits or losestates. That's why all the value remain zeros.

$T = 10, M = 10, \gamma = 0.5, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.013 & -0.125 & -0.5 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

In this case, with the presence of motion noise, even if the control policy is "going up", it is still possible that the agent goes right and finally it could detect the closeexit(assigned with value -1).

$T = 100, M = 10, \gamma = 0.5, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.012 & -0.25 & -1 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

This could be explained with the same reason: the control policy is "going up", so if the agent goes up at the start grid, it would never reaches the exits(In function *nextstate*, there is no probability that the agent goes down).

$T = 10, M = 50, \gamma = 0.8, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.003 & -0.168 & -0.5 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

$T = 10, M = 50, \gamma = 0.8, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.003 & -0.168 & -0.5 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

$T = 100, M = 50, \gamma = 0.8, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.025 & -0.225 & -0.91 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

All the results shown above are the results I obtain after one time of implementation. Hence, the results may be a little different due to the randomly generated trajectory.

To sum up, the given control policy and the motion uncertainty model are the key factor to these results. Using this algorithm, we are not trying to obtain the correct J values for each state space. Instead, we are evaluating the algorithm performance using the given policy. Hence, we have to try enough times to find a feasible path towards the exit grid and in this case, the motion uncertainty is helpful for us to explore more state spaces. In this case, if there is no model uncertainty(deterministic system), we will never find any feasible paths.

2 Problem 2

2.1 problem formulation

In the file modelFreePrediction, implement the TD(0) algorithm using T -long episodes and M iterations. To do this, fill out the function TDzero, which takes in a policy π, the random getRandomcost function and generates MT -long episodes. The program outputs the approximated cost-to-go function J^π. Using $\pi ="\text{up}"$ except when i is the terminal state, in which case we use 0, provide sample vector values that you obtain in the mediumGrid when you combine the values (a)$T = 5, T = 10$, and $T = 100$, (b)$M = 5, M = 10$ and $M = 50$, (c)$\gamma = 0.5$, and $\gamma = 0.8$.Use the parameter $\eta = 0.25$ in all simulations. What do you observe?

2.2 solution

The initial value $J_0^\pi(i)$ for all grids i(including the terminal states) are set as zero.

$T = 5, M = 5, \gamma = 0.3, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & -10 & 0 & -10 \\ -10 & 0 & 0 & 0 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

In this case, the terminal states can not be detected. Hence, all the values remain unchanged.

$T = 10, M = 10, \gamma = 0.3, \eta = 0.25$:

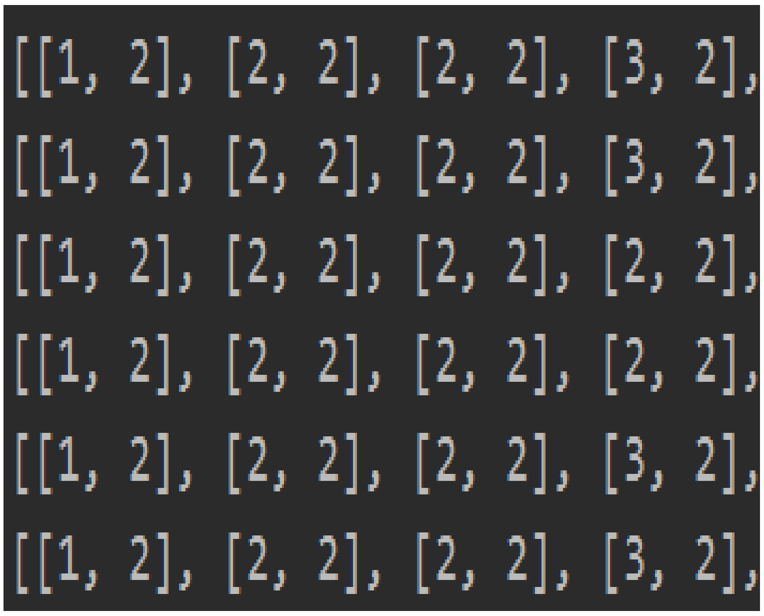
$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & -10 & 0 & -10 \\ -10 & 0 & 0 & 0 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

The same things happen in this case.

$T = 100, M = 50, \gamma = 0.3, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & -10 & 0 & -10 \\ -10 & 0 & 0 & -0.667 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

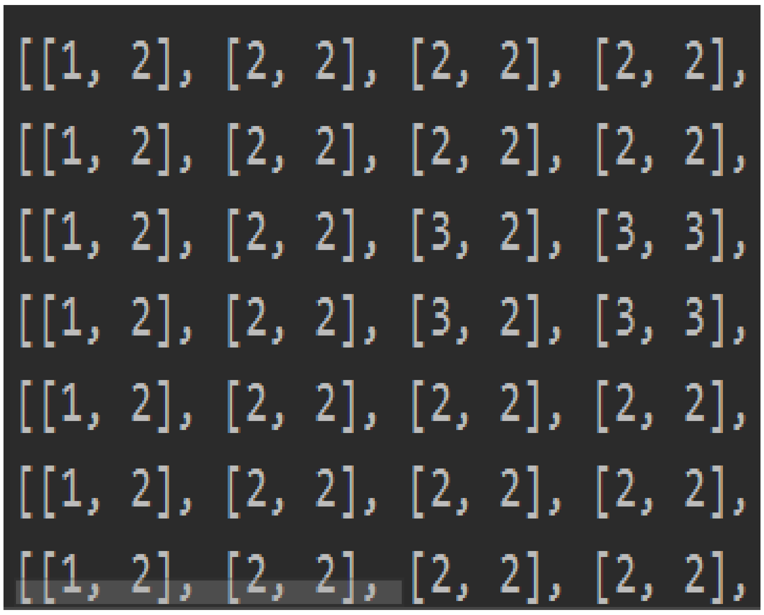
At first, I was surprised by the results here. You could see that the J value of grid (3,2) has been updated ,which means at least one feasible path is detected. However, the J values of grid (2,2) and (1,2) are still zeros. The reason is that the agent stay at the grid (2,2) more than one step(at least two steps) and the algorithm only finds the first time t that it appears in $\rho_0^(k)$. Hence, the J values of grid (2,2) can not be updated.



```
[[1, 2], [2, 2], [2, 2], [3, 2],
[[1, 2], [2, 2], [2, 2], [3, 2],
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [2, 2], [3, 2],
[[1, 2], [2, 2], [2, 2], [3, 2],
```

Figure 2: The generated paths

From the above figure, it is clear that the agent would stay at (2,2) for two serial steps and this could give us unexpected results. However, the action that enables the agent move from (2,2) to (3,2) is generated by the motion uncertainty, it is possible that the agent only stay at (2,2) for one step and moves to (3,2). After I run it multiple times, I could obtain the following results.



```
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [3, 2], [3, 3],
[[1, 2], [2, 2], [3, 2], [3, 3],
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [2, 2], [2, 2],
[[1, 2], [2, 2], [2, 2], [2, 2],
```

Figure 3: The generated paths

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.002 & -0.072 & -0.757 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

$T = 100, M = 50, \gamma = 0.8, \eta = 0.25$:

$$\begin{bmatrix} -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ -10 & 0 & 0 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & 0 & 0 & 0 & -10 \\ -10 & 0 & -10 & -1 & -10 & 0 & -10 \\ -10 & -0.039 & -0.214 & -0.781 & -5 & 0 & -10 \\ -10 & -5 & -5 & -5 & -5 & -5 & -10 \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 \end{bmatrix}$$

To sum up, if we try enough times, the results are reasonable. In theory, suppose every state is visited infinitely many times and $\{\alpha_k\}$ is Robbins Monro, then the (first-visit) TD methods guarantee $J^{\pi}_{M}(i) \rightarrow J^{\pi}(i)$ with probability one as $M \rightarrow \infty$. However, when we implement this algorithm, we can not run it infinite times, so we can just approximate the correct results. Depends on the environment layout and given policy, the question that how many iterations are enough for the good approximation has different answers. Sometimes, we may have to figure out a good control policy beforehand if we expect some better performance.

3 Problem 3

3.1 Problem formulation

Are there any tradeoffs between both approaches?

3.2 solution

The most obvious advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an online, fully incremental fashion. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one time step. Some applications have very long episodes(not in this case), so that delaying all learning until the end of the episode is too slow. Other applications are continuing tasks and have no episodes at all.

On the other sides, we have been taught that both algorithm can guarantee convergence to the correct answer. However, which one converges faster is still hard to give a mathematical proof. In practise, The initial value of J_0^π for each grid i may play an important role for the final performance because we are only able to run limited numbers of iterations, thus approximating the correct solution instead of obtaining the closed-form solution.