Reinforcement Learning

Assignment #2 Report

Student names

Pedram Abdolahi Darestani

Zahra Jabari

Student IDs

202383919

202291677

Date

July 18th , 2024

Introduction

This assignment is aimed at evaluating a given policy for a customized 5x5 gridworld problem and estimating an optimal policy for it using different methods.

- Policy evaluation: Bellman equations, policy iteration
- Optimal policy estimation: Bellman equations, policy iteration, value iteration, Monte Carlo exploring starts, Monte Carlo with epsilon-soft approach, and off-policy Monte Carlo

1) Part 1

This part of the assignment consists of two sections:

1. Given policy evaluation
    1. Bellman equations
    2. Policy iteration
2. Optimal policy estimation
    1. Bellman equations
    2. Policy iteration
    3. Value iteration

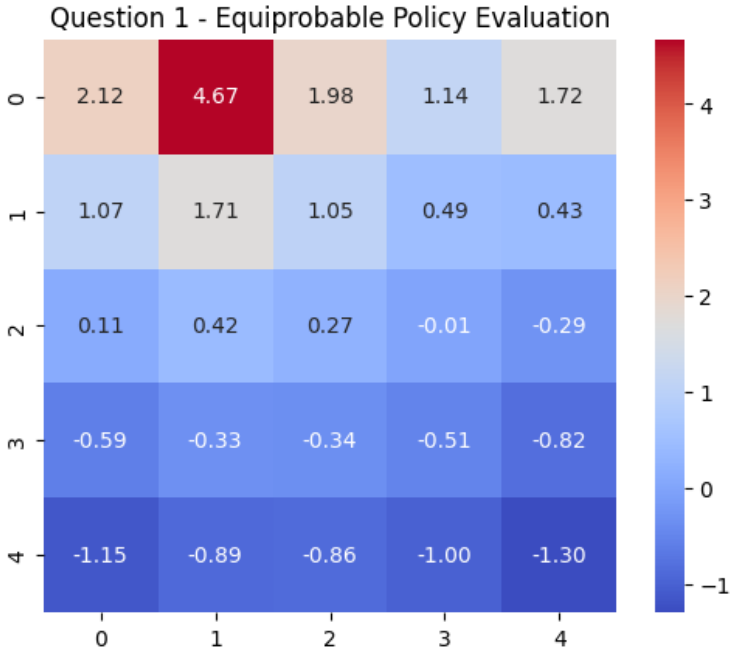
Problem specifications

- Discount rate (gamma) = 0.95
- Policy: probability of moving in any of the four directions = 0.25
- Any action at the blue square => +5 reward => jump to block 17 (numbering starts from 0 in the top left of the grid and ends at 24 in the bottom right square)
- Any action at the green square => +2.5 reward => jump to square 17 or 24 with probability 0.5
- An attempt to leave the grid yields a reward of -0.5 and does not change the state of the agent.
- All actions are deterministic and where the agent ends up after taking them is clear, except for when agent is in the green square.

1-1) Policy Evaluation

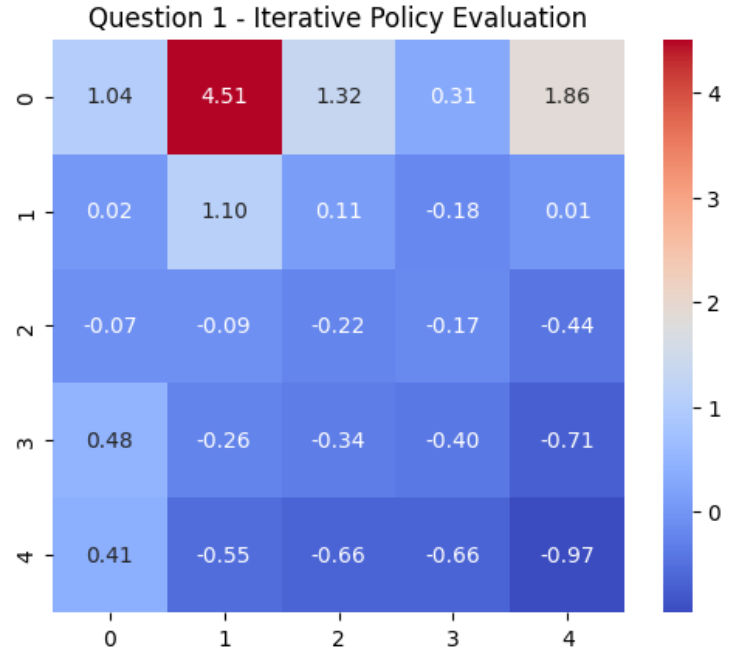
1-1-1) Solving Bellman equations

In this section, the Bellman equations were solved explicitly to evaluate the state value function, V(s). The resulting value function is given below.



1-1-2) Policy iteration

In this part, the policy iteration method was used to evaluate the given equiprobable policy and the results below were achieved.



Analysis

The policy we evaluated is a policy that takes one of the four possible actions at each state with equal probability. This could result in the decrease of the value function in the edges of the grid, with the lowest values in the lower corners. This can be observed in the (true) state value function found through solving the Bellman equations: the farther a state is from the edges, the higher its state value function. The upper edge is the exception since it has states in which large positive rewards are granted to the agent. Furthermore, we by following this policy we can expect the state value function in the +5 and +2.5 reward blocks to be higher than others. This can also be observed in the true state value function. However, the states to the sides of the block with the +5 reward show higher values as compared to the block with the +2.5 reward, which seems unusual. This could mean that being in those states could end up being more favorable in the long run.

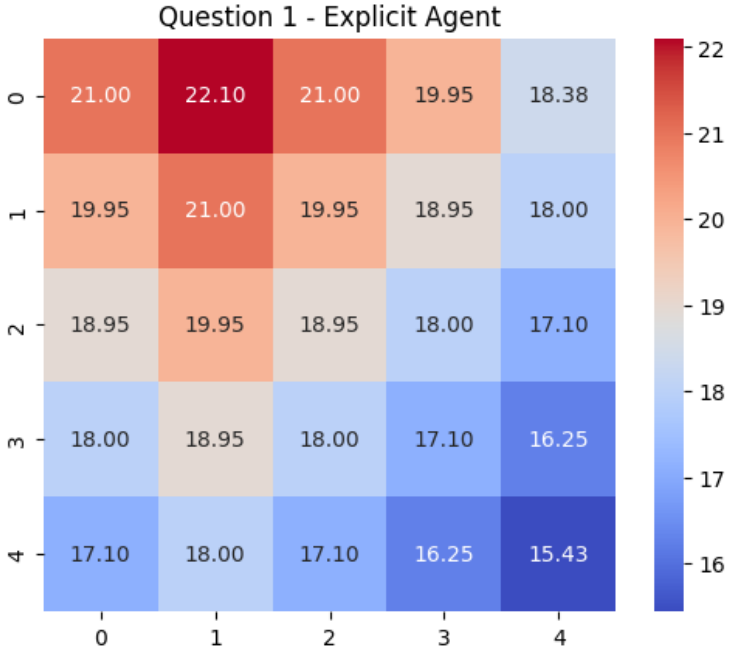
Moreover, the state value function for the +5 and +2.5 states is less than 5 and 2.5, respectively. This could mean that even if an agent ends up in these states, the rewards it gets in the long run is not higher than the instant reward of taking an action while being in that state.

Next, we look at the state values found by the policy iteration method. The values are quite similar to the results of the Bellman equation, accompanied with a few errors in the bottom left corner of the grid.

1-2) Optimal Policy Estimation

1-2-1) Solving Bellman equations

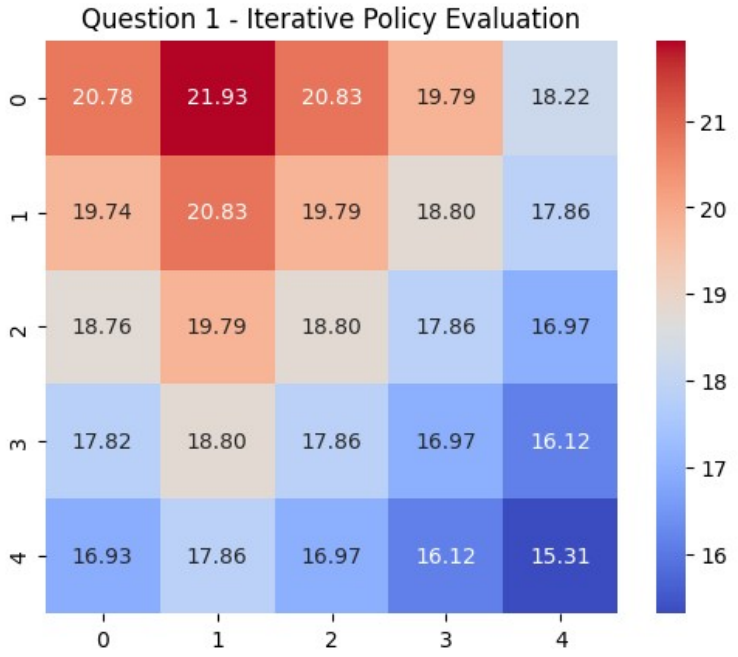
In this section, the Bellman equations were solved explicitly to evaluate the state value function, V(s), for the optimal policy. By having the optimal state value function, we can easily find the optimal policy by behaving greedily based on it. The resulting action value function is shown below.



At any state in the given grid, an optimal policy would be to take an action that leads to the state with the highest state value.

1-2-2) Policy iteration

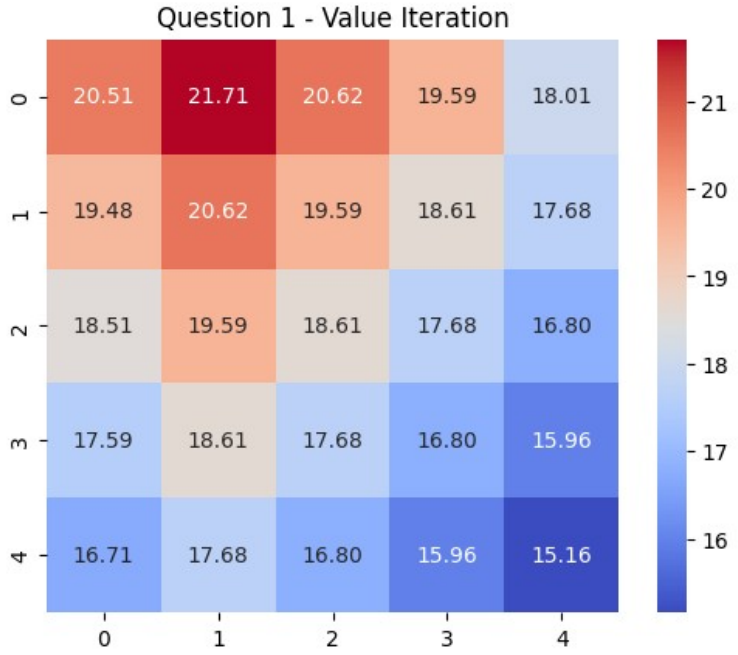
In this section, the policy iteration method was used to estimate the state value function for the optimal policy, and the results below were achieved.



As it can be seen, the estimated value functions are quite close to the actual value functions found through solving the Bellman equations. This algorithm has a huge advantage over solving the equations, and that is not requiring nearly as much computation. By having the estimated state value function, we can find an actually optimal policy in this question, but in more complex problems arriving at a “good” policy would be appreciated.

1-2-3) Value iteration

In this section, the value iteration method was used to estimate the state value function for the optimal policy, and the results below were achieved.



The value iteration method yields similar results to the previous two methods for this problem, and can provide us with the optimal policy as well. This method is also less computation intensive than solving the Bellman equations, and can provide us with a good estimation of the optimal policy.

Analysis

An optimal policy would be to take an action that leads to the state with the highest state value when we have the optimal state value function. By observing the calculated optimal value function, one can notice that the value function of each state is directly related to how many steps away it is from the +5 (blue) block. The exception to this is the +2.5 (green) block. This is because actions in this block do not lead to the nearby states, which makes the idea of “distance from the best state” irrelevant. The bottom right corner of the grid is the farthest state from the optimal state, thus, has the least state value function.

Both of the policy iteration and value iteration methods resulted in a pretty close estimation of the actual state value function, and can provide us with a good policy regardless of the small deviations from the absolute truth.

Part 2

This part of the assignment is aimed at estimation of the optimal policy using Monte Carlo methods, and consists of 3 sections.

1. On-policy Monte Carlo
    1. With exploring starts
    2. Without exploring starts, but with an e-soft approach
2. Off-policy (behavior) Monte Carlo
3. Policy iteration

Problem specifications

- Discount rate (gamma) = 0.95
- Any action at the blue square => +5 reward => jump to 22
- Any action at the green square => +2.5 reward => jump to square 22 or 24 with probability 0.5
- An attempt to leave the grid yields a reward of -0.5 and does not change the state of the agent.
- Any movement between the white blocks yields a reward of -0.2.
- There are two terminal states at blocks 14 and 20.
- All actions are deterministic and where the agent ends up after taking them is clear, except for when agent is in the green square.

2-1-1) Exploring starts on-policy Monte Carlo

In this section we make sure that all state-action pairs have equal probability of being initialized, and then we start generating episodes. In order to prevent creation of episodes of infinite length, a maximum length of 2500 was asserted and a total number of 100,000 episodes with such maximum length were generated. Greedy approach with respect to the action value function was taken. The estimation of the optimal policy found through this method is shown below.

Right		Left	Left	
Right	Up	Left	Left	Left
Up	Up	Up	Up	
Up	Up	Left	Up	Left
	Up	Left	Up	Up

As it can be seen, the actions of this policy all lead the agent to the best/blue state in the shortest number of steps, thus, we can say that this policy is optimal.

2-1-2) Epsilon-soft on-policy Monte Carlo

As compared to the previous section, here we do not have to make sure that every state-action pair has an equal probability of occurring through exploring starts. Instead, we use an epsilon-soft policy in order to carry out the exploration in finding the optimal policy. Since the epsilon-greedy policy is a subset of the epsilon-soft policy, we can use it with our approach in this section. Here, we generate 1,000,000 episodes of maximum length 10,000 to estimate the optimal policy and the results achieved are shown below.

1. Epsilon = 0.05

Right		Left	Left	
Up	Up	Up	Up	Up
Up	Up	Up	Right	
Down	Up	Up	Up	Up
	Left	Left	Up	Up

2. Epsilon = 0.1

Right		Left	Left	
Up	Up	Up	Up	Up
Up	Up	Up	Right	
Down	Up	Up	Up	Up
	Left	Left	Up	Up

3. Epsilon = 0.2

Right		Left	Left	
Up	Up	Up	Up	Up
Up	Up	Up	Up	
Up	Up	Up	Up	Up
	Left	Up	Up	Up

4. Epsilon = 0.3

Right		Left	Left	
Up	Up	Up	Left	Up
Up	Up	Up	Up	
Up	Up	Up	Up	Up
	Up	Up	Up	Up

5. Epsilon = 0.4

Right		Left	Left	
Up	Up	Up	Left	Up
Up	Up	Up	Up	
Up	Up	Up	Left	Left
	Up	Up	Up	Left

6. Epsilon = 0.5

Right		Left	Left	
Up	Up	Up	Left	Up
Up	Up	Up	Up	
Up	Up	Up	Left	Left
	Up	Up	Left	Left

It can be seen that for lower values of epsilon, when near the terminal states, the estimated optimal policy tends to jump to the terminal state out of fear of generating negative rewards. This is due to low levels of exploration and not having seen better options. As the value of epsilon increases, the closer the estimated policy becomes to the optimal policy.
The estimated optimal policies are quite similar for epsilon = 0.4 and 0.5. They both refuse to enter the terminal states and aim for the blue state which has the highest rewards. However, they both choose to enter the green state for when the agent is in the far right state of the second row (state number 9, numbering started from 0). This is a sub-optimal action since the -0.2*4 reward that the agent will sustain moving to the left is much less than the extra +2.5 it would get if it went for the blue state.
All in all, it can be said that the epsilon-soft method is effective in estimation of the optimal policy for this 5x5 gridworld problem.

2-2) Off-policy Monte Carlo

In this section we make use of a behavior policy alongside our target policy in order to guide this target policy towards the optimal one. This behavior policy is an equiprobable policy that can take any action with equal probability in any state. The estimated optimal policies are shown below.

1. Number of episodes = 1e6, maximum sequence length = 10,000
   The estimated optimal policy is devoid of value

Right		Right	Right	
Right	Right	Up	Up	Right
Left	Up	Down	Down	
Down	Right	Up	Left	Down
	Left	Down	Up	Left

2. Number of episodes = 1e7, maximum sequence length = 10,000
   The estimated optimal policy is devoid of value.

Left		Right	Left	
Left	Right	Up	Down	Right
Up	Down	Right	Up	
Left	Left	Up	Right	Down
	Down	Up	Up	Left

3. Number of episodes = 1e5, maximum sequence length = 50
   The estimated optimal policy is an alright estimate, with sub-optimal decisions made nearby the green and terminal blocks.

Right		Left	Left	
Up	Up	Up	Up	Up
Up	Up	Left	Up	
Down	Up	Up	Left	Left
	Up	Down	Left	Up

4. Number of episodes = 1e6, maximum sequence length = 50
   The estimated optimal policy is a good estimate, with sub-optimal decisions made in positions = 14, 17, and 22. These decisions do not result in termination of the episode and only yield an extra -0.4 reward before they make it to the blue square.

Right		Left	Left	
Right	Up	Left	Up	Left
Up	Up	Up	Down	
Up	Left	Up	Left	Left
	Right	Up	Up	Up

5. Number of episodes = 1e5, maximum sequence length = 60
   The estimated optimal policy is gets stuck indefinitely between positions 8 and 9, which is sub-optimal. Other than this, the policy is close to being optimal.

Right		Left	Left	
Right	Up	Up	Right	Left
Up	Up	Left	Up	
Up	Up	Left	Right	Up
	Up	Left	Up	Up

6. Number of episodes = 1e6, maximum sequence length = 60
   The estimated optimal policy enters terminal states and takes sub-optimal paths towards the blue square, marking it not the best one found so far.

Down		Left	Right	
Right	Up	Left	Left	Up
Up	Up	Left	Left	
Right	Up	Up	Up	Left
	Left	Up	Left	Up

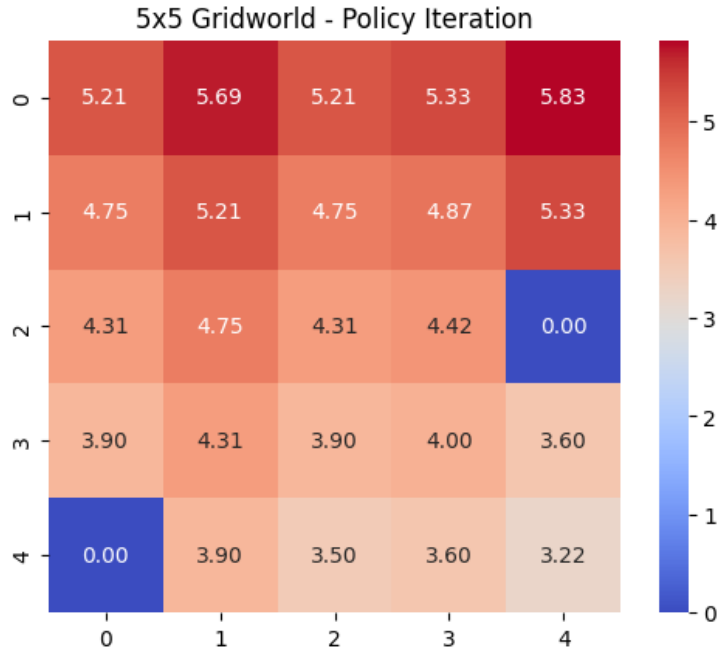
7. Number of episodes = 1e7, maximum sequence length = 60
This estimation also enters the terminal states which makes it not as good as the other policies previously found.
It is interesting that increasing the number of episodes considered has not increased the quality of the estimated optimal policy at this sequence length.

Down		Left	Up	
Right	Up	Left	Down	Up
Up	Up	Up	Left	
Right	Up	Up	Left	Left
	Up	Left	Up	Up

Maximum sequence lengths of 20 and 30 were also tested as well and they yielded similar results to the ones shown above (their results can be found in the Runs2 notebook available in the Github repository for this assignment). The best policy that the off-policy Monte Carlo control was able to find among the ones tested could be seen as the policy number 4: while it does not always take the optimal action, it always leads the agent towards the +5 (blue) state and never gets stuck in loops.
Unlike the other Monte Carlo algorithms, the off-policy Monte Carlo methods for optimal policy estimation showed to be sensitive to the maximum length of each episode.

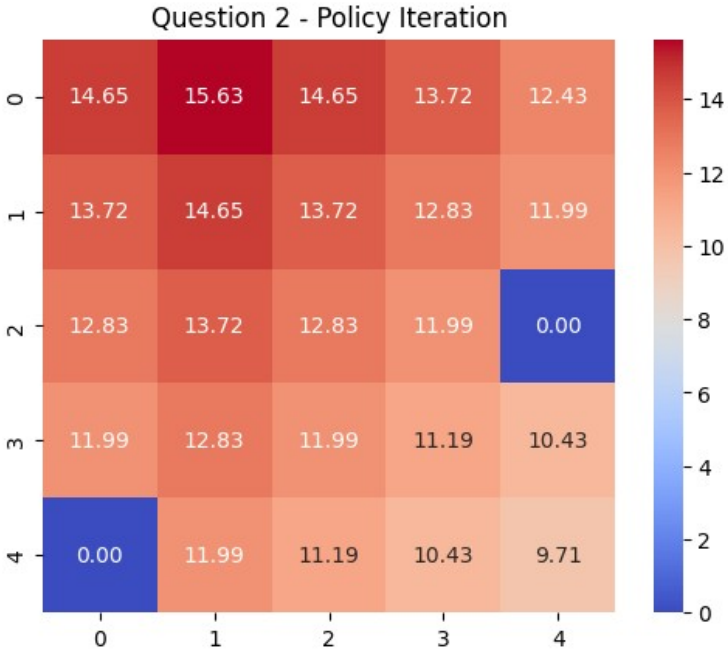
2-3) Policy iteration

In this section, the location of the blue and green blocks is permuted with a probability of 0.1 in every step. The policy iteration algorithm demands the stop of the algorithm when the policy has converged. This can result in the convergence of the policy before the permutation of the positions of the blocks. To counteract this effect, we let the algorithm continue even if the policy has converged. The difference this permutation would make is that the value function of the blue and green states will not be stationary, and will change over time/episodes. This complicates evaluation of a singular stationary value function for the states in the problem. Regardless, the algorithm was run and the resulting state value function is shown below.



The optimal policy under such circumstances is quite tricky to arrive at. Let's assume we allow the algorithm and the permutation run indefinitely. One could say that the probability of the green block being in position 2 is equal to that of the blue block. As a result, the resulting policy from the yielded state value function can be interpreted as this for an agent: "Get to the closest positive reward block to you, be it the green or the blue".

Assuming the blue and green do not change positions, running the policy iteration algorithm yields the results below.



The results are quite similar to the results of section 1; the state value function depends on how far each state is from the +5 state. However, the state value functions are quite

higher than those when the positions of the blue and green block change. It is clear that the uncertainty in getting the best reward reduces the expected reward estimation, but the level to which this estimation is affected deserves attention. Should the state value function in the blue and green positions be reduced to about 1/3 of the optimal value when the uncertainty is injected? Perhaps the simulation of the “running the algorithm indefinitely” is insufficient. To check this, the algorithm was run for 100 and 1000 epochs, with each epoch having 0.1 chance of permutation, and the results they yielded were identical. Runtime for 100 epochs was about 1 hour and 10 minutes, and increasing it would be exceptionally time consuming.