UCLA

232E - Large-Scale Social and Complex Networks: Design and Algorithms

Report on

Project 3: Reinforcement Learning and Inverse Reinforcement Learning

Authors

Jayanth Shreekumar (805486993)
Leo Ly (805726182)
Yuheng He (505686149)

Spring 2022

Introduction

Reinforcement learning (RL) is a training method applied in machine learning that rewards behaviours that are required to achieve a certain goal and punishes others. It is a set of algorithm where the task is to learn from interaction to achieve a goal. There are mainly two parts to a RL framework:

- Agent: The agent takes decisions based on rewards and punishments provided by the environment.
- Environment: The environment is where the agent lives and interacts. However, while agent can utilize the environment to make its decisions, it cannot influence the dynamics of the environment in any way.

Environment

In this project, we assume that the environment of the agent is modeled by a Markov Decision Process (MDP). The agent can perform certain actions to change their state, and receive a reward based on their action, and advance to a new state. An MDP is a tuple containing:

- S: The state of states. In this project, it is a 2-D grid of 10x10, so we have 100 states.
- A: The set of actions. In this project, the agent can perform 1 of four actions when in any state: up, down, left, or right.
- $P^a_{ss'}$: The set of transition probabilities that determines the probability of an agent transitioning from one state to another based on the current state, the next state, and the action taken. In this project, the probability of transitioning in a required direction was always given to be (1 - x), where x is the probability of transitioning to any other state due to wind w. The edges and corners were special states and were governed by the fact that falling off the state space after taking an action meant staying in the same state as before.
- $R^a_{ss'}$: The reward awarded that is determined by an agent transitioning from one state to another based on the current state.
- γ the discount factor, and it is used to compute the present value of future reward.

Policy

procedure VALUE ITERATION($\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$, $\mathcal{S}$, $\mathcal{A}$, γ):
  for all $s \in \mathcal{S}$ **do** ▷ Initialization
    $V(s) \leftarrow 0$
  end for
  $\Delta \leftarrow \infty$
  while $\Delta > \epsilon$ **do** ▷ Estimation
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$;
      $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
    end for
  end while
  for all $s \in \mathcal{S}$ **do** ▷ Computation
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
  end for
end procedure return π

Figure 1: Value Iteration Algorithm

A policy $\pi(s)$ comprises the suggested actions that the agent should take for every possible state s∈ S. The **optimal policy** is one which results in optimal value function, which was calculated using the algorithm that is shown in Figure 1. Optimal policy is used to obtain the best value $V*$ by recursively applying the Bellman Optimality equation for state-action value function:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P^a_{ss'}[R^a_{ss'} + \gamma V(s)']$$

We solve for V*(s) iteratively and after we have all V*(s), we compute the optimal actions given by the optimal policy for every state in the computation step as :

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P^a_{ss'}[R^a_{ss'} + \gamma V(s)']$$

Question 1

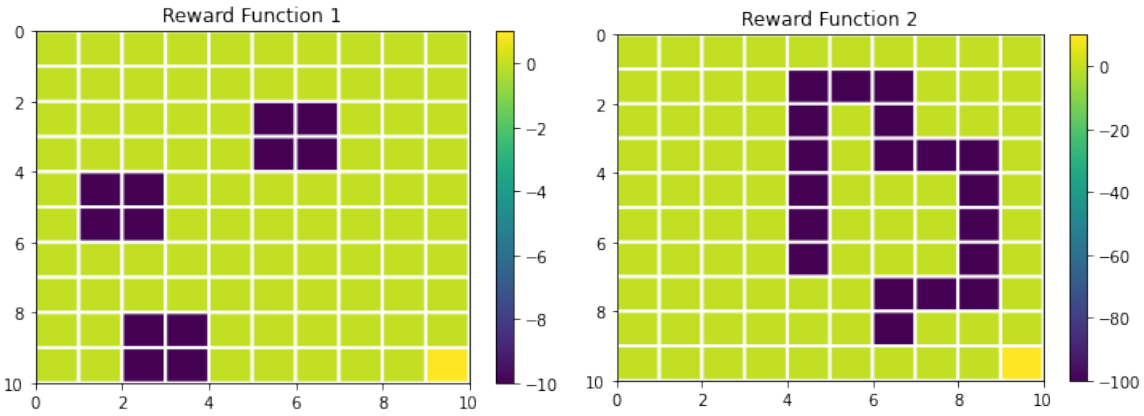


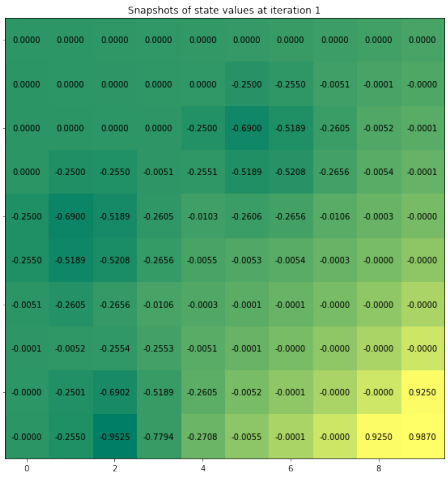
Figure 2: Heatmaps of the given reward functions

As suggested, we used pcolor() from the matplotlib library for our plots. We see that the rewards given for the states in purple is much less, meaning that we can visualize those as a barrier of sorts. Also, the reward for the final end state is the highest, which makes sense intuitively, as reaching the end means the agent has done its job.

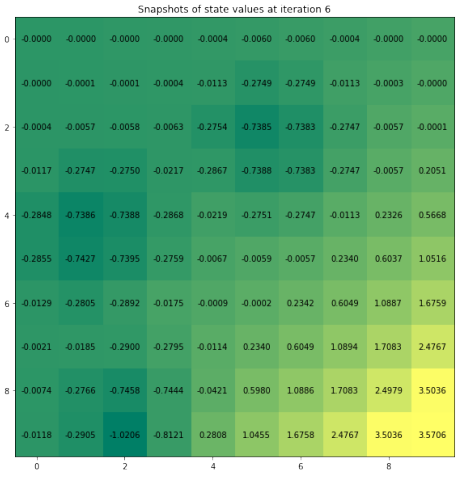
Question 2

We implemented the RL framework as described in the project specifications sheet. We set w to 0.1 and γ to 0.8 as suggested. For the estimation step, we set ϵ to 0.01. **We see that the value iteration algorithm converges in 21 steps**. Our optimal value function plot along with the 5 snapshots is shown below:

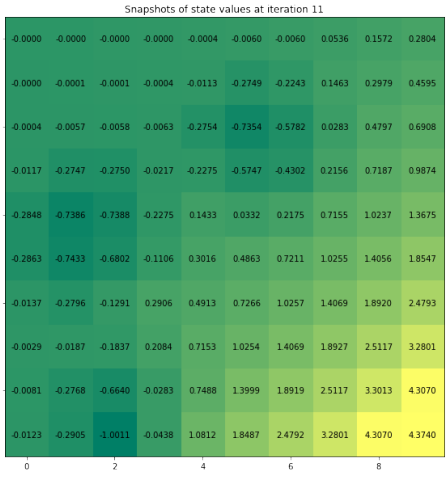
(a) **Optimal values** for each state using reward function 1



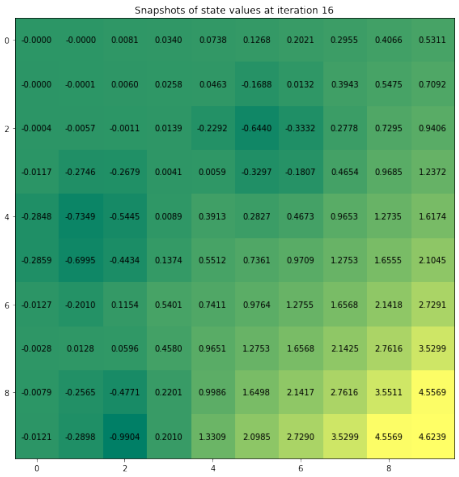
(b) Snapshot of value function at iteration 1 using reward function 1



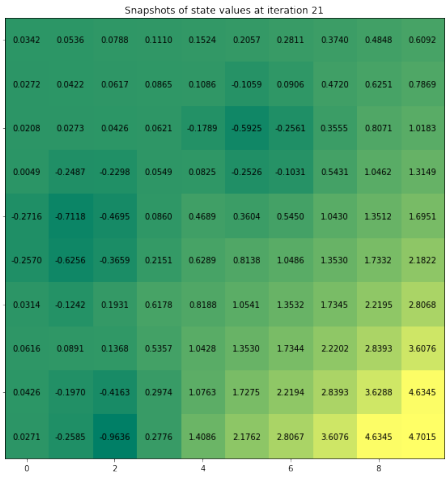
(c) Snapshot of value function at iteration 6 using reward function 1



(d) Snapshot of value function at iteration 11 using reward function 1



(e) Snapshot of value function at iteration 16 using reward function 1



(f) Snapshot of value function at iteration 21 using reward function 1

Figure 3: Optimal values and snapshots at iteration 1, 6, 11, 16, and 21

- We see that the optimal values for iteration 1 are much lesser in magnitude than those of iteration 21, and the values progressively increase in an exponentially decreasing manner as the iteration number increases. This is to be expected: as the algorithm converges, we are able to navigate the whole state-space optimally.
- All plots have negative optimal values for states that have negative rewards.
- The plots generated at iteration 1 and 6 are sparse, and have 0s as optimal state values. As iteration number increases, the sparsity decreases, which means that the algorithm is progressing.

Question 3

The heatmap of the optimal state values using reward function 1 is shown below:

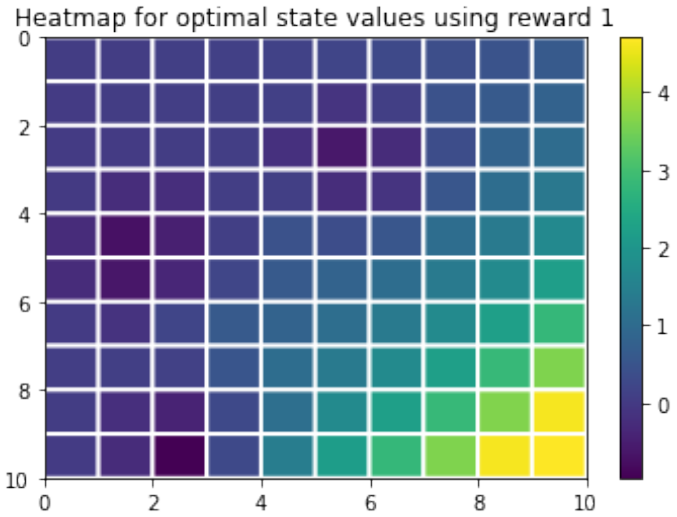


Figure 4: Heatmap of the optimal state values using reward function 1

Question 4

We make several observations based on the optimal state values:

- The final state at the bottom right has the highest optimal value. This makes sense intuitively as it also has the highest reward associated with it.
- The states that have a negative reward also have a negative value, thus discouraging the agent from ever occupying these states.
- States close to the states with negative rewards have low optimal values as occupying these states means that the agent has to take a round-about path to the end state.
- We see that the "optimal paths" that the agent should take have the highest optimal values associated with it. This makes sense intuitively as well.

From these observations, we conclude that the idea of finding the reward function using the environment and its optimal values is possible, and we do this in the inverse RL section.

Question 5

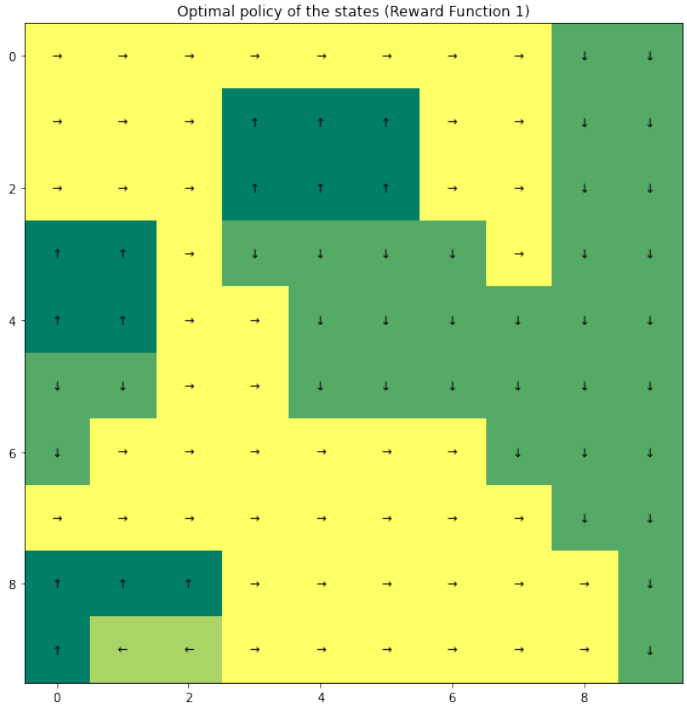


Figure 5: Optimal policy plot for reward function 1

Yes, the optimal policy that we obtained makes sense:

- All arrows eventually lead in the direction of the end state.
- policy for states with negative rewards are pointed in such a way that the agent will immediately be made to exit those corresponding states. In general, the arrows point so that the agent moves away from these states. This makes intuitive sense as we need to maximize total reward.

Yes, it is possible for the agent to compute the optimal action to take at each state by observing the optimal values of its neighboring states. This is because, **given the current state, the optimal next state would be the one that has the highest value**. This is strengthened by the fact that the optimal policy is always such that it points in the direction of the next state having the largest value among all its neighbours. As we computed the value function in a recursive manner to obtain the optimal values, greedily choosing the state with the highest value works and makes intuitive sense.

Question 6

Here is the optimal value plot using reward function 2 with wind w = 0.1 and discount factor $\gamma = 0.8$

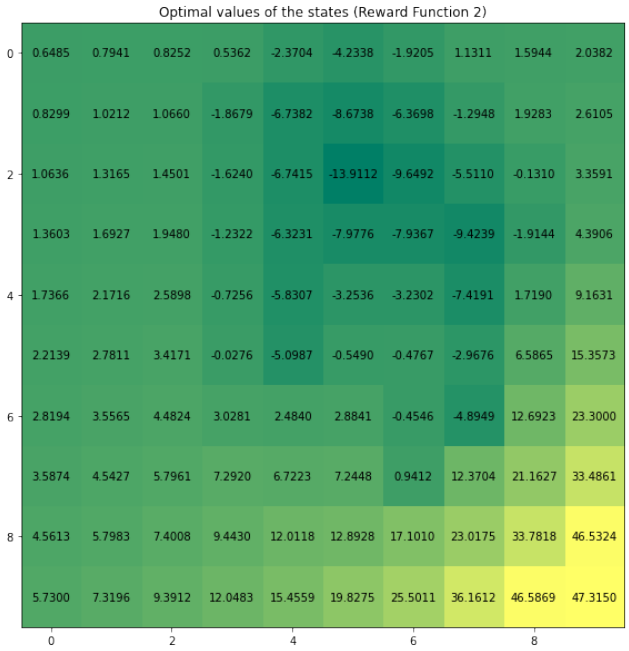


Figure 6: Optimal value for each state using reward function 2

Question 7

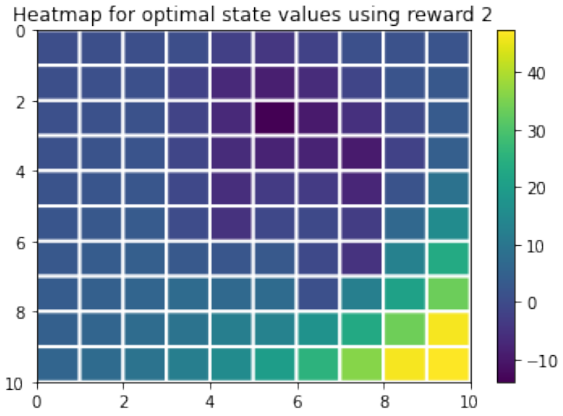


Figure 7: Heatmap of the optimal state values using reward function 2

Again, we see that the optimal values are highest near the end state. Similar to when using reward function 1, we see that the optimal values for states that had negative reward are also negative, thus discouraging the agent from ever visiting that state. Also, the interesting point to note there is that **states that are within the snake-like negative rewards have negative optimal values.** This makes intuitive sense as, if the agent is in one of those states, it has to take a round-about path to the end state, which is undesirable.

Question 8

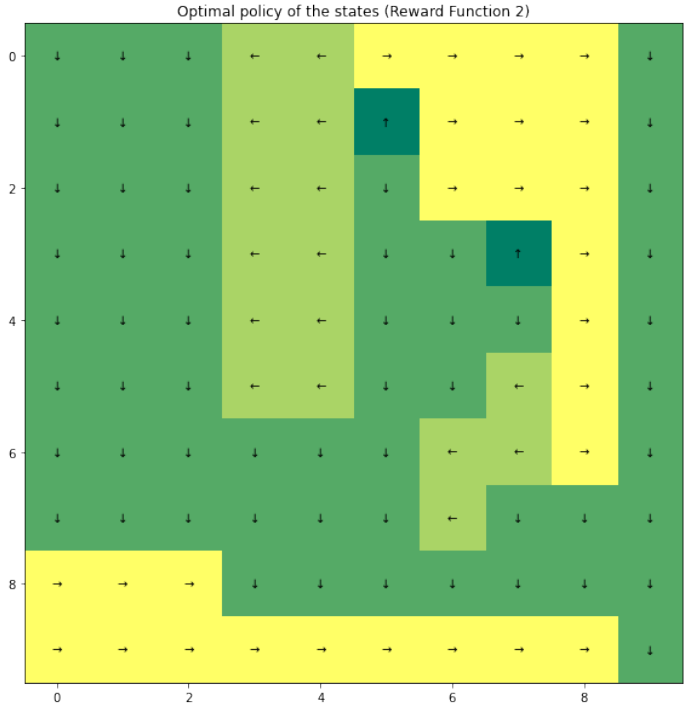


Figure 8: Optimal policy plot for reward function 2

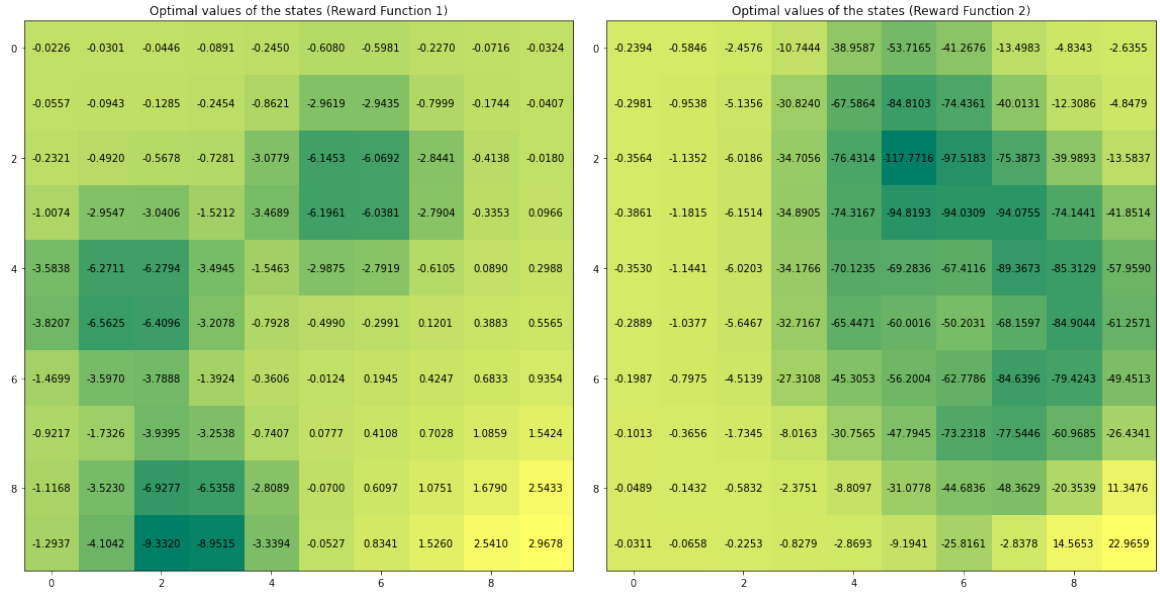
Yes, the optimal policy that we obtained makes sense:

- All arrows eventually lead in the direction of the end state.
- Policy for states with negative rewards are pointed in such a way that the agent will immediately be made to exit those corresponding states. In general, the arrows point so that the agent moves away from these states. This makes intuitive sense as we need to maximize total reward.
- All arrows point in such a way as to avoid the snake-like negative reward. This makes sense as we want to maximize the reward we accumulate.
- Also, we noticed that the large block of light green arrows point to the left, away from the negative rewards. We think this is because the task of the agent is to get the maximum cumulative reward, but there is no constraint on the length of the path it takes. Therefore, the algorithm performs in such a way as to always avoid going to the states with negative rewards.

Question 9

As required, we updated the wind value w to 0.6 and ran our experiments again on both reward functions.

Here are the optimal values:



(a) Optimal values for reward function 1 with w = 0.6 (b) Optimal values for reward function 2 with w = 0.6

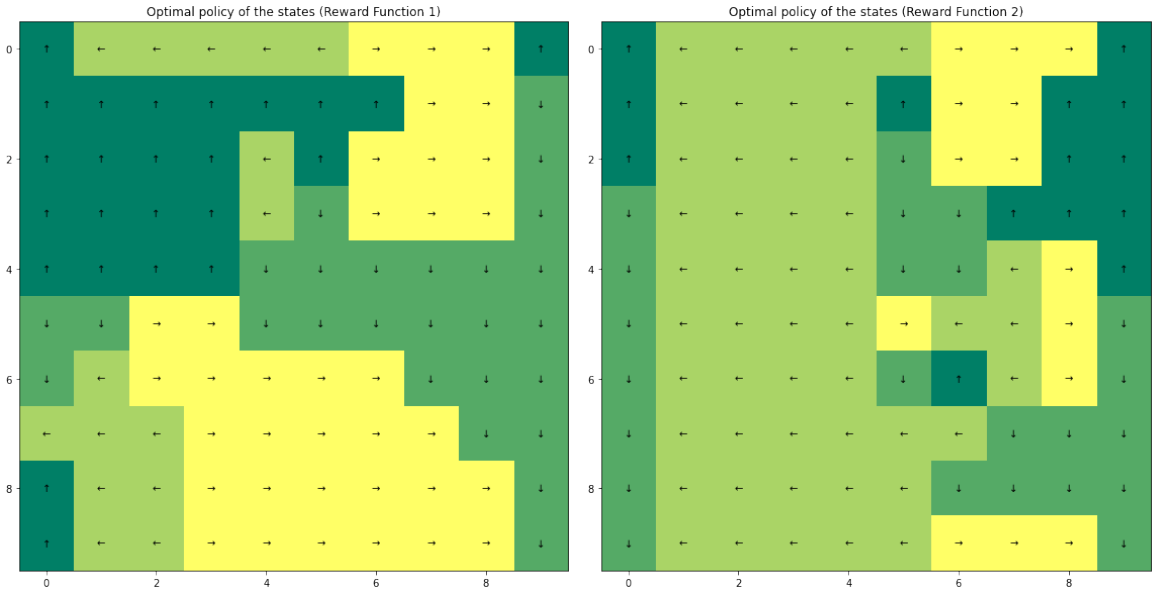
Figure 9: Optimal values of the given reward functions with wind w = 0.6

Here are the heatmaps of optimal values:

(a) Heatmap of optimal state values for reward function 1 with w = 0.6 (b) Heatmap of optimal state values for reward function 2 with w = 0.6

Figure 10: Heatmap of optimal values for the given reward functions with wind w = 0.6

Here are the optimal policies:



(a) Optimal policy for reward function 1 with w = 0.6 (b) Optimal policy for reward function 2 with w = 0.6

Figure 11: Optimal policies for the given reward functions with wind w = 0.6

We make several observations using these plots:

- Optimal value of end state has decreased, so the agent has less incentive to actually approach the end state.
- The negative values of states with negative rewards has increased. Also, due to this, neighbouring states also have increased negative values. This means there are now lesser paths that lead to the end state.
- The policy plots have arrows point towards the start state. This suggest staying at the start state is optimal which is obviously a mistake.
- In policy 2, the arrow marks to the right of the plot are very contradicting. Right arrows point to the right, but these states have arrows pointing back to the left. This is clearly a problem, and leads to local optima where the agent just oscillates between the two states.

All these problems are caused due to the exploitation vs exploration conundrum in reinforcement learning. We can think of the wind factor w as the probability of exploration. Consider two extreme cases: w = 0, and w = 1. When w = 0, there is no exploration, and the algorithm uses exploitation to the fullest, greedily choosing the next state that has the best reward. However, this is not a good idea, as the algorithm does not take into account the future paths, and the best current choice may not be the overall best choice to achieve the goal. In the case of w = 1, the algorithm simply uses exploration completely. While it is better to have a bit of exploration so that the agent can learn new paths, relying completely on exploration means that the agent is moving randomly without regard for the rewards it accrues. We need to strike a fine balance between exploration and exploitation for optimal policy.

We conclude stating that w = 1 is a better value as the agent has a higher chance of reaching the end state while also having the option of exploration.

Inverse Reinforcement Learning

Introduction

Inverse Reinforcement Learning (IRL) is the task of learning an expert's reward function by observing the optimal behavior of the expert. IRL is motivated by apprenticeship learning, whose goal is for the agent to learn a policy by observing the behavior of an expert. The task of apprenticeship can be accomplished in the following two ways.

1. Learn the policy directly from expert behavior, and
2. Learn the expert's reward function and use it to generate the optimal policy.

The second way is preferred in that the reward function provides a much more parsimonious description of behavior. Reward function is considered as the most succinct, robust, and transferable definition of the task, and therefore, extracting the reward function of an expert would help design more robust agents.

IRL Algorithm

The Linear Programming formulation of the IRL is given as follows.

$$\begin{aligned}
&\max_{\boldsymbol{R},t_i,u_i} \sum_{i=1}^{|S|}(t_i - \lambda u_i) \\
&\quad s.t.[(\boldsymbol{P}_{a1}(i) - \boldsymbol{P}_a(i))(\boldsymbol{I} - \gamma \boldsymbol{P}_{a1})^{-1}\boldsymbol{R}] \geq t_i, \forall a \in \boldsymbol{A}i, \forall i \\
&\qquad (\boldsymbol{P}_{a1}(i) - \boldsymbol{P}_a(i))(\boldsymbol{I} - \gamma \boldsymbol{P}_{a1})^{-1}\boldsymbol{R} \geq 0, \forall a \in \boldsymbol{A}i \\
&\qquad -\boldsymbol{u} \leq \boldsymbol{R} \leq \boldsymbol{u} \\
&\qquad |\boldsymbol{R}_i| \leq R_{max}, i = 1, 2, \ldots, |S|
\end{aligned}$$

$\boldsymbol{R}$ is the reward vector, $\boldsymbol{P}_a$ is the transition probability matrix corresponding to the action a, λ is the adjustable coefficient, and t_i's and u_i's are the extra optimization variables.

Question 10

The expressions are given as follows.

$$c = \begin{bmatrix} \mathbf{1}_{|S|\times 1} \\ -\boldsymbol{\lambda} \cdot \mathbf{1}_{|S|\times 1} \\ \mathbf{0}_{|S|\times 1} \end{bmatrix} \; x = \begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{u} \\ \boldsymbol{R} \end{bmatrix} \; b = \begin{bmatrix} \mathbf{0}_{(2\cdot(|A|-1)\cdot|S|\times 1} \\ \boldsymbol{R}_{max} \\ \boldsymbol{R}_{max} \end{bmatrix} \; D = \begin{bmatrix} \boldsymbol{I}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & -(\boldsymbol{P_{a1}} - \boldsymbol{P_a})(\boldsymbol{I} - \gamma \boldsymbol{P_{a1}})^{-1} \\ \mathbf{0}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & -(\boldsymbol{P_{a1}} - \boldsymbol{P_a})(\boldsymbol{I} - \gamma \boldsymbol{P_{a1}})^{-1} \\ \mathbf{0}_{|S|\times|S|} & -\boldsymbol{I}_{|S|\times|S|} & \boldsymbol{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & -\boldsymbol{I}_{|S|\times|S|} & -\boldsymbol{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & \boldsymbol{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & \boldsymbol{I}_{|S|\times|S|} \end{bmatrix}$$

where $\boldsymbol{P_a}$ is in $\boldsymbol{P_{a2}}$, $\boldsymbol{P_{a3}}$, and $\boldsymbol{P_{a4}}$.

0.1 Performance measure

Question 11

We swept λ from 0 to 5 to get 500 evenly spaced values of λ. The accuracy rates we obtained for different λ is shown in Fig. 12.

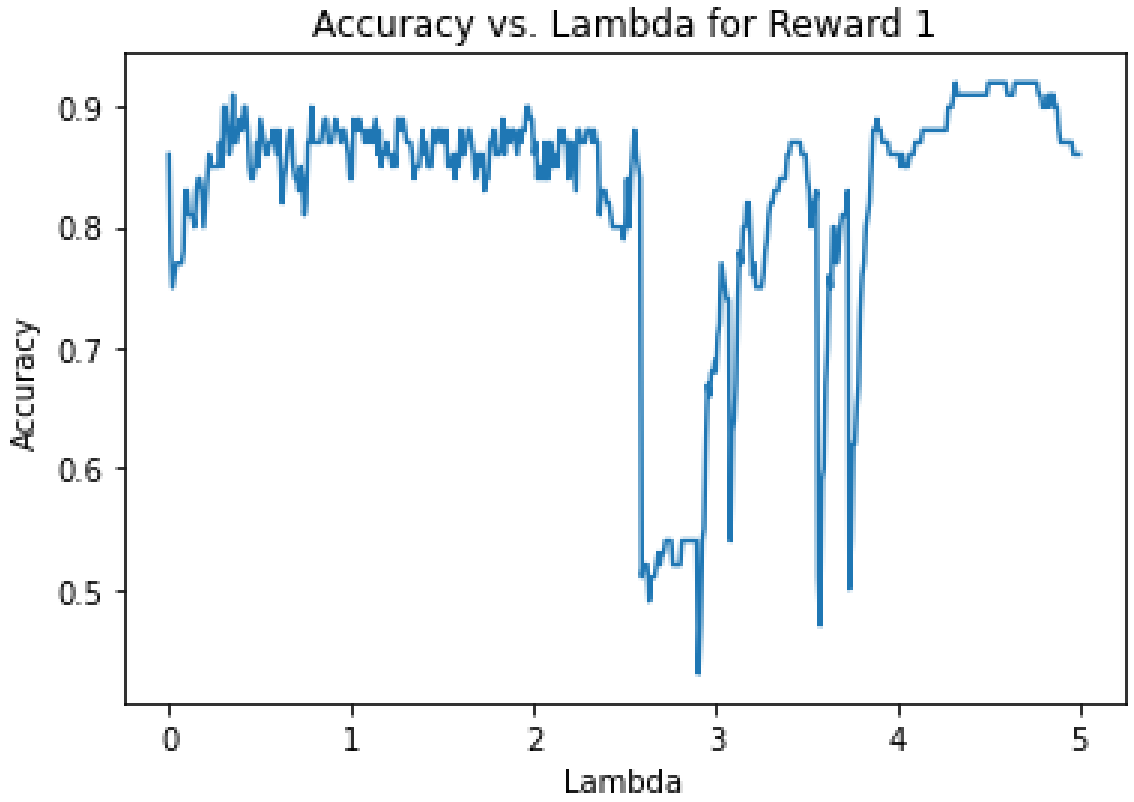


Figure 12: Accuracy Rates of Different λ Values

As we can see from Fig. 12, the accuracy rate oscillates between 0.8 and 0.95 but drops drastically for some λ values, and then stabilizes at the level of 0.9.

Question 12

We obtained the max accuracy rate 0.92 for this part, and the corresponding λ value $\lambda_{max}^{(1)}$ is 4.7695.

Question 13

In this part, we generated the heat map for the obtained reward and reward 1 using $\lambda_{max}^{(1)}$. The heat maps are given in Fig. 13 and Fig. 14, respectively.

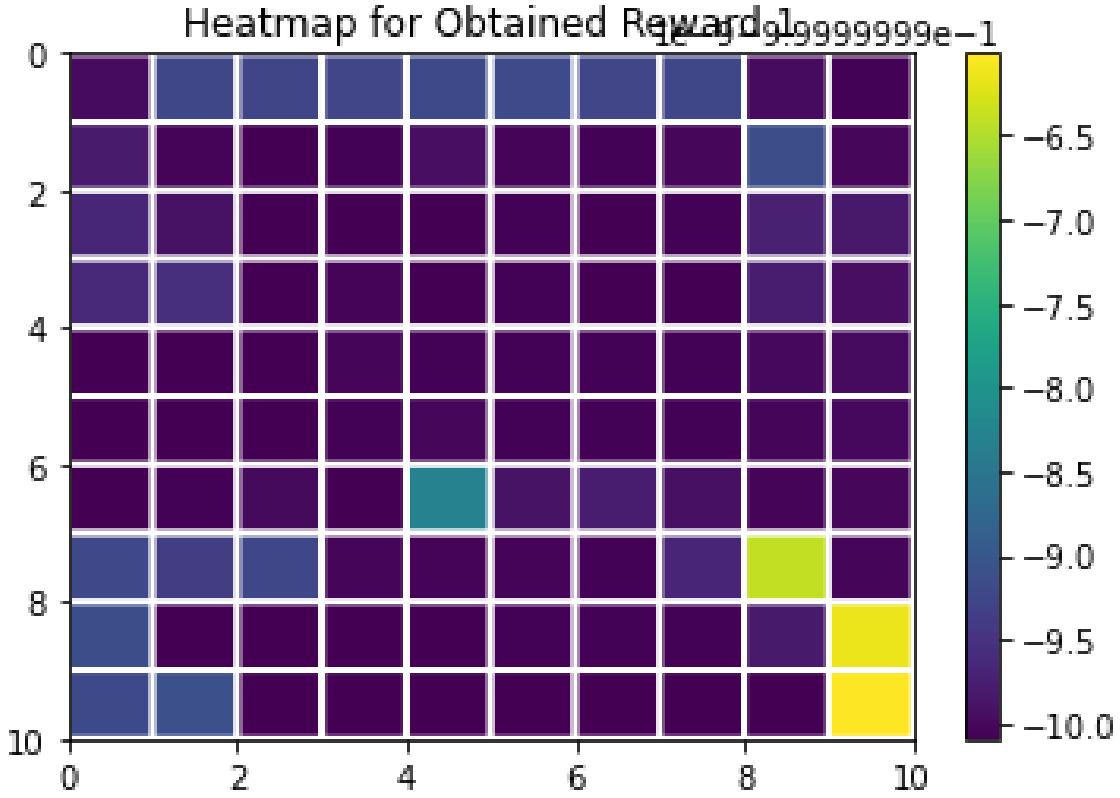


Figure 13: Heat Map for the Obtained Reward for $\lambda_{max}^{(1)}$

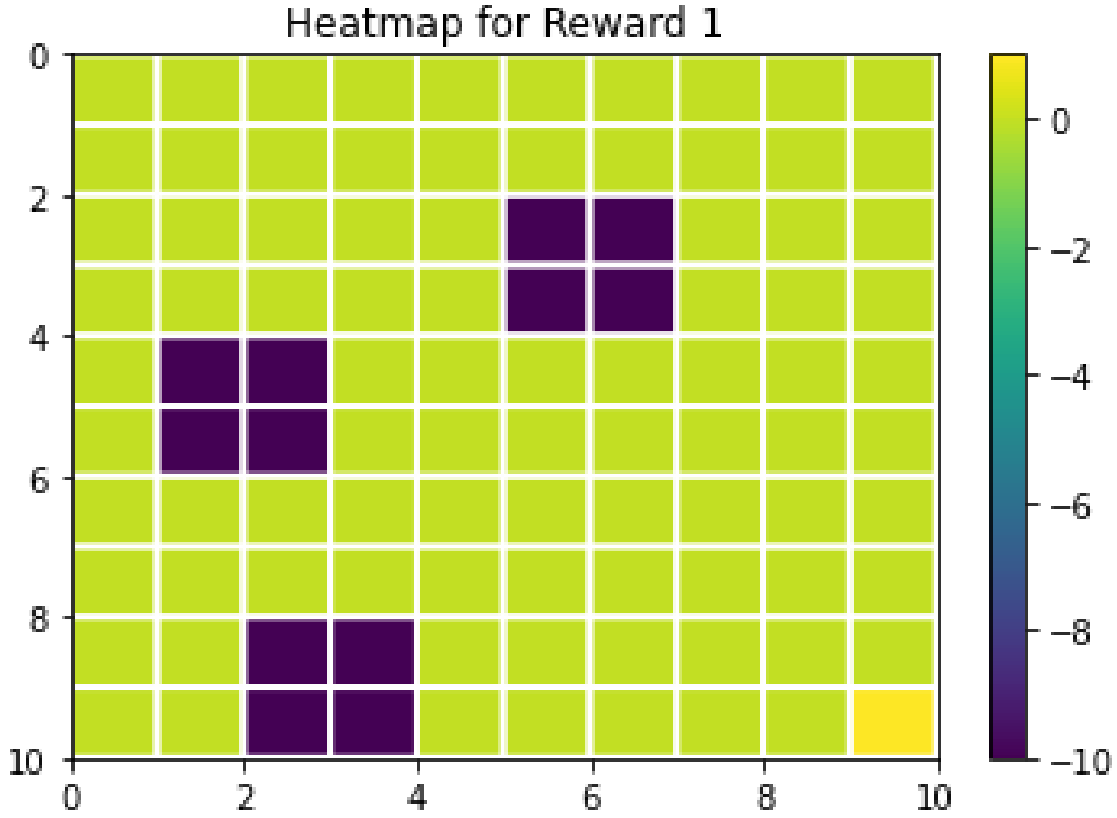


Figure 14: Heat Map for Reward 1 for $\lambda_{max}^{(1)}$

Question 14

In this part, we obtained the optimal values of the states using the extracted reward function with $\lambda_{max}^{(1)}$. The result is plotted in Fig. 15

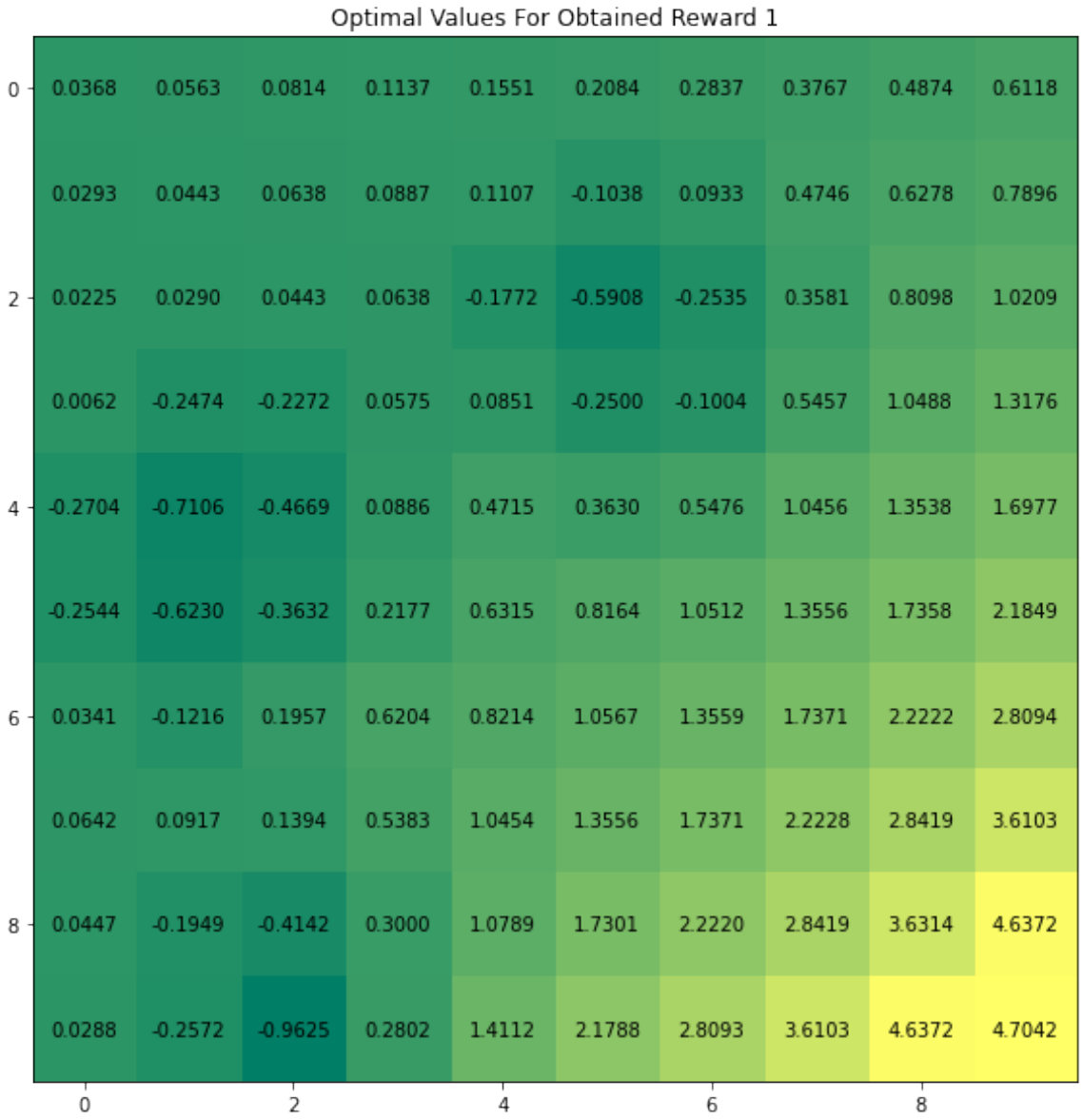


Figure 15: Optimal Values For Extracted Reward

Question 15

Similarities: We observed that in both cases, the regions around 99 have relatively higher values. This is explained by the fact that 99 has the highest reward score out of all other states. We also observed that surrounding the negative reward values, the optimal values are also low. We can clearly see the three low values blocks that are corresponding to the negative reward blocks.

It is also obtained that the optimal values change in correspond to the reward function. As the agent moves towards state 99, the values increase and when the agent moves towards the negative reward blocks, the values decrease.

Differences: We discerned that the scaling for the optimal value of the extracted reward function is different from that of the heat map of reward function 1. This makes sense in that the optimal value is computed from the extracted reward function

We can see that in Fig. 15, the values change gradually, while the change in the heat map is more abrupt. This is explained by the way these matrices are defined. The reward function 1 is the

expert ground truth and it only has 3 values, while the extracted reward function is computed from policy and thus the change is more smooth.

Question 16

In this part, we obtained the optimal policy of using the extracted reward function with $\lambda_{max}^{(1)}$. The result is plotted in Fig. 16

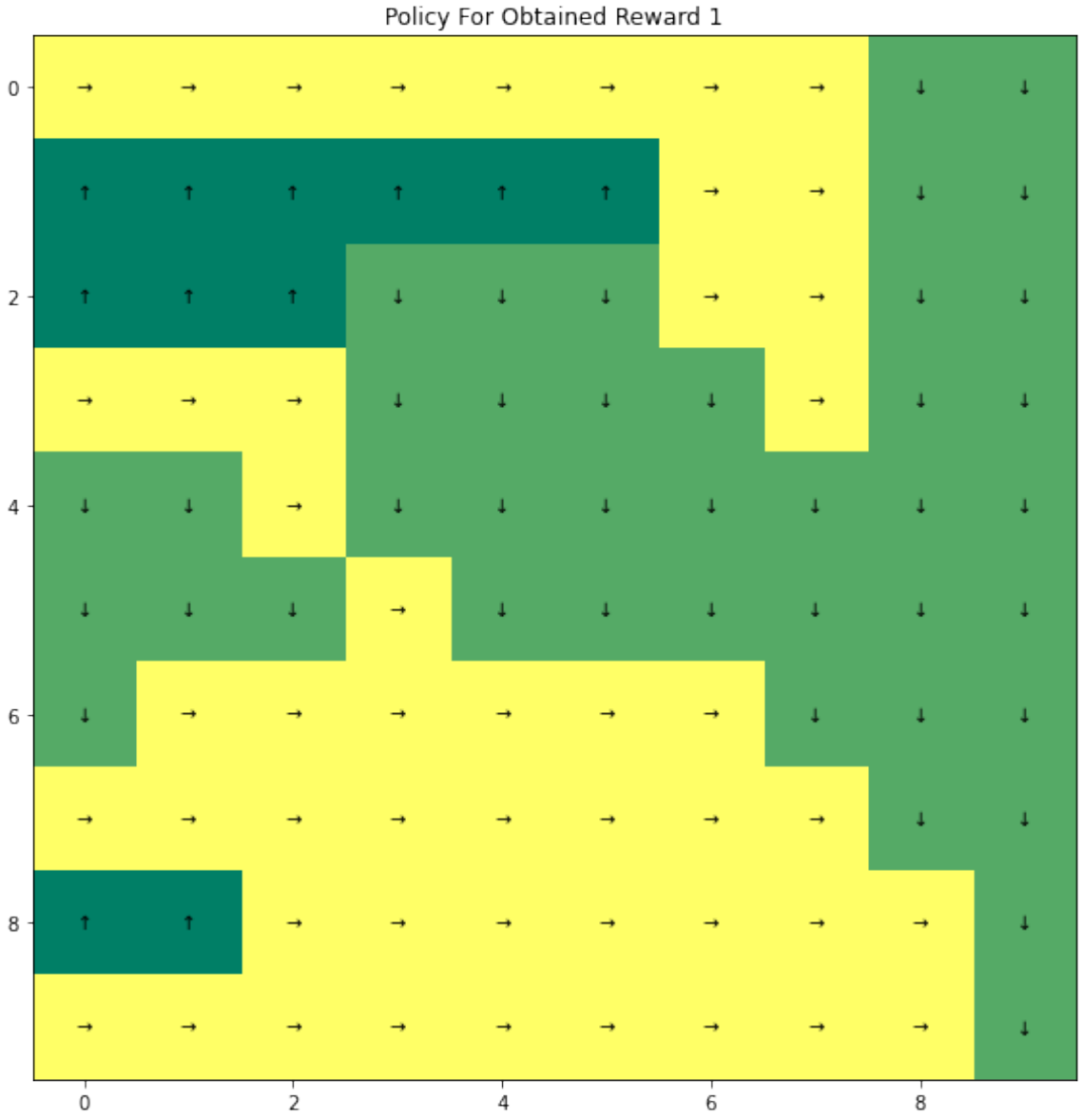


Figure 16: Optimal Policy For Extracted Reward

Question 17

Similarities: We observed that in both Fig. 5 and Fig. 16 the majority of arrows are right and down. This is explained by the fact that the bottom right state (state 99) has the highest reward, and therefore the decision will be mostly directing to the bottom right part, making the majority action right and down.

Differences: We could see in Fig. 16, some actions are different from the one in Fig. 5. For example, in Fig. 16, state 91, 92 are arrows pointing left while in Fig. 5, the arrows are pointing left. If we start from state 90, using the extracted reward function 1, we go directly to state 99, where the highest reward exists, and if we are using reward function 1, we will go up two rows and then start moving towards state 99, making it less optimal. These discrepancies are explained by the ways the reward functions are defined. For reward function 1, most of the values are set to 0 while for the extracted function, the values surrounding the negative blocks are small positive numbers, resulting in different actions in the policy map.

Question 18

Similar to Question 11, we swept λ from 0 to 5 to get 500 evenly spaced values for λ. We used the optimal policy of the agent found in Question 9 to fill in the $O_E(s)$ values. Fig. 17.

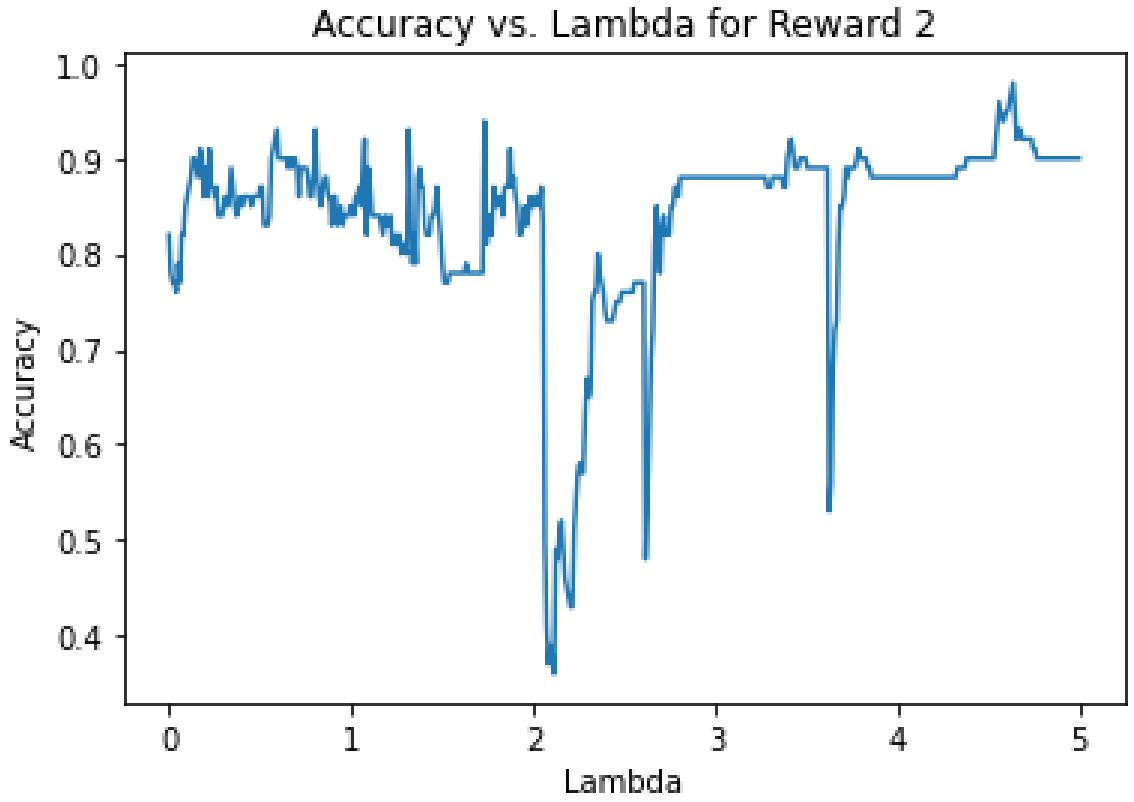


Figure 17: Accuracy vs. λ for Reward 2

Question 19

Based on the plot, $\lambda^{(2)}_{max} = 4.6393$ with an accuracy of 0.98.

Question 20

Using the IRL algorithm, we were able to extract the reward from the given optimal policy in Question 9. Fig. 18 and Fig. 19 show the derived heatmap and the ground truth heatmap, respectively. We can see that the heatmaps are substantially different between the ground truth version and the derived version. The most similar point between the 2 heatmaps are the reward at state 100 – both have the highest rewards. Regarding differences, Obtained reward 2 has a completely different range comparing to the true heatmap for reward 2.

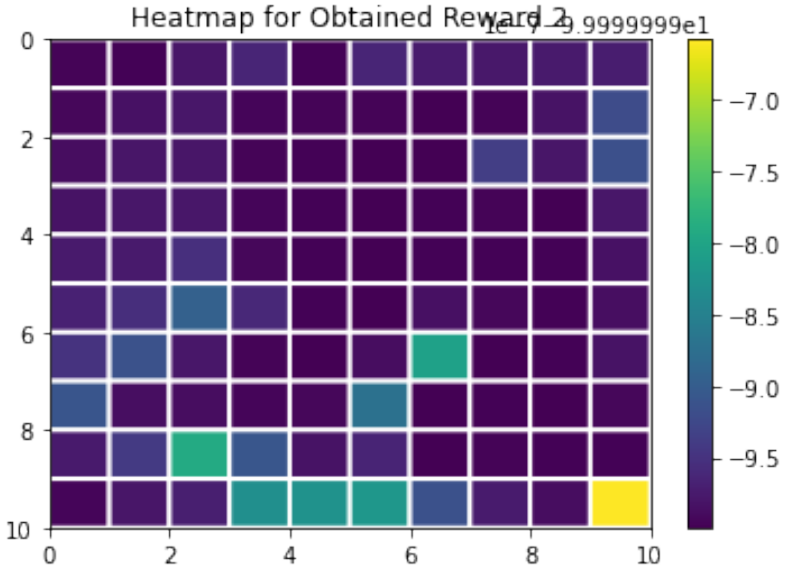


Figure 18: Reward 2 Heatmap Derived from IRL

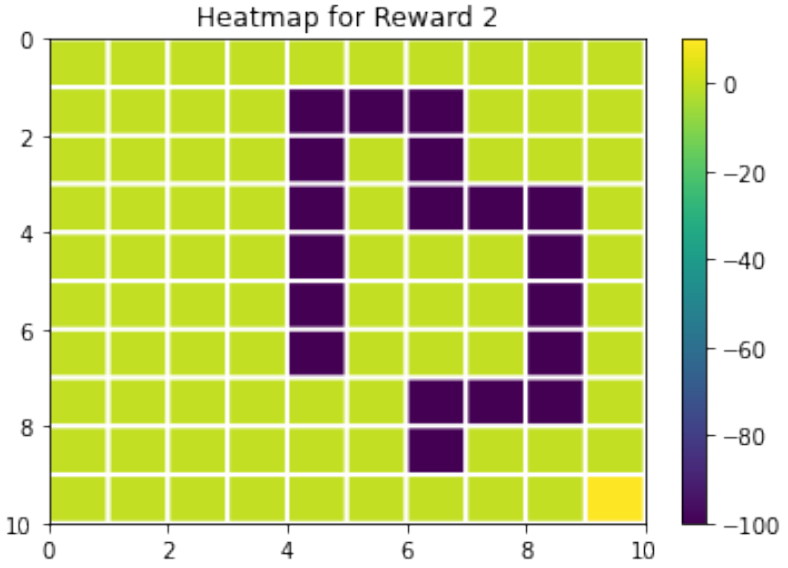


Figure 19: True Reward 2 Heatmap

Question 21
Optimal values of the states were computed using the extracted reward function found in Question 20. Fig. 20 shows the heatmap of derived optimal values for Reward 2. We can see that the highest score is at the bottom right as expected. The snake-like negative reward gets negative values.

Question 22

Similarities: Both have the same gradient, with highest scores are at the bottom right and the lowest scores are within the region with negative rewards. It shows that the IRL algorithm is able to discern the negative reward region well even if the region is more complex than negative reward blocks in Reward function 1.

Differences: The optimal values are different significantly in terms of scale and range. In the ground truth data, the optimal values range is from -117.77 to 22.96, while the range derived by the IRL algorithm is from -13.91 to 47.32. It is due to the fact that the most negative value in true Reward function 2 is -100. The optimal function needs to calculate accordingly. In the IRL algorithm, inputs are the optimal policy only. Without having very negative values in reward, we see a smaller values range in the derived reward.

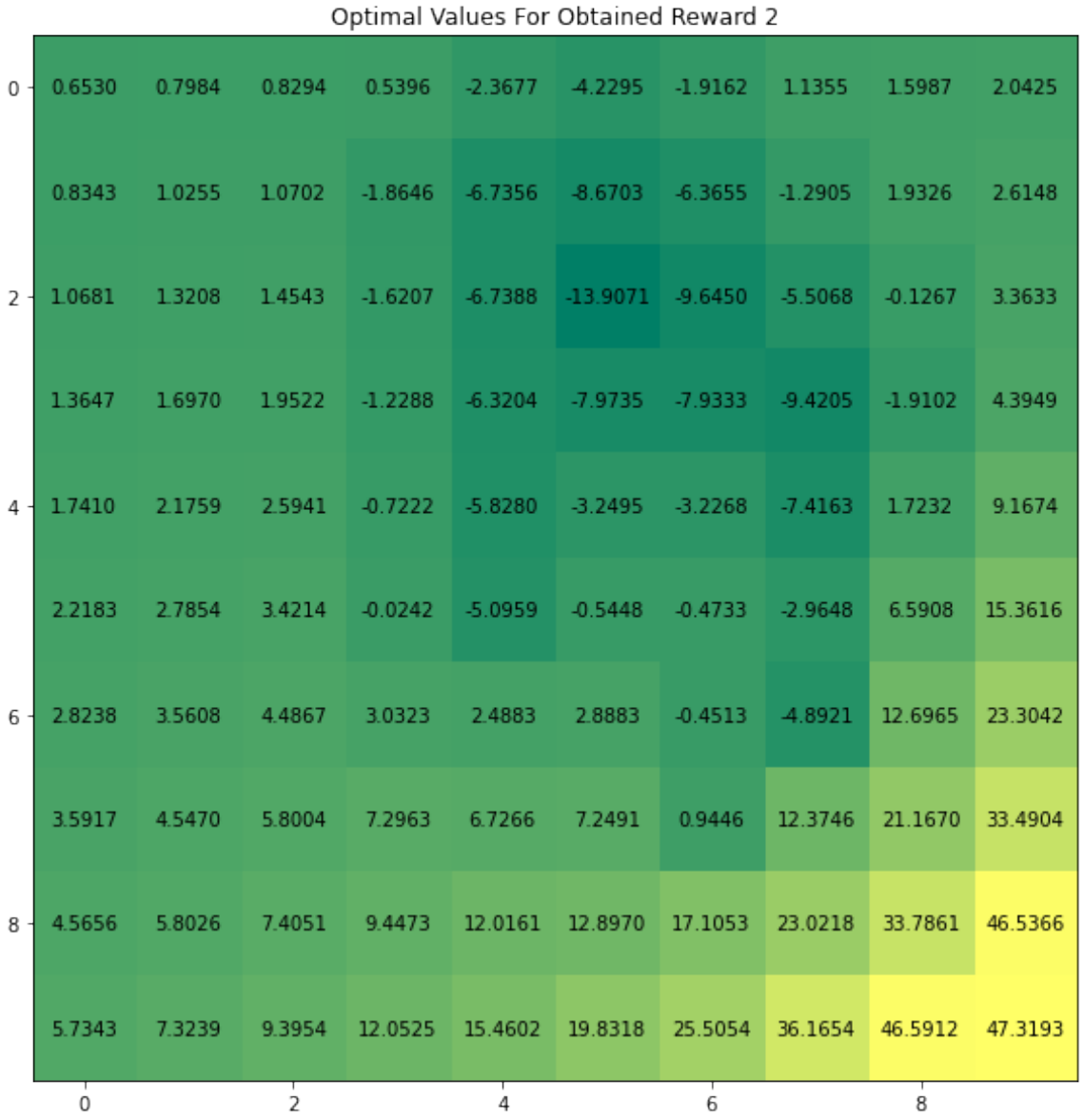


Figure 20: Optimal Values For Extracted Reward

Question 23
After finding the optimal values, we were able to compute the optimal policy of the agent. Fig. 21 shows the derived policy, while Fig. 22 is the ground truth policy for Reward 2.

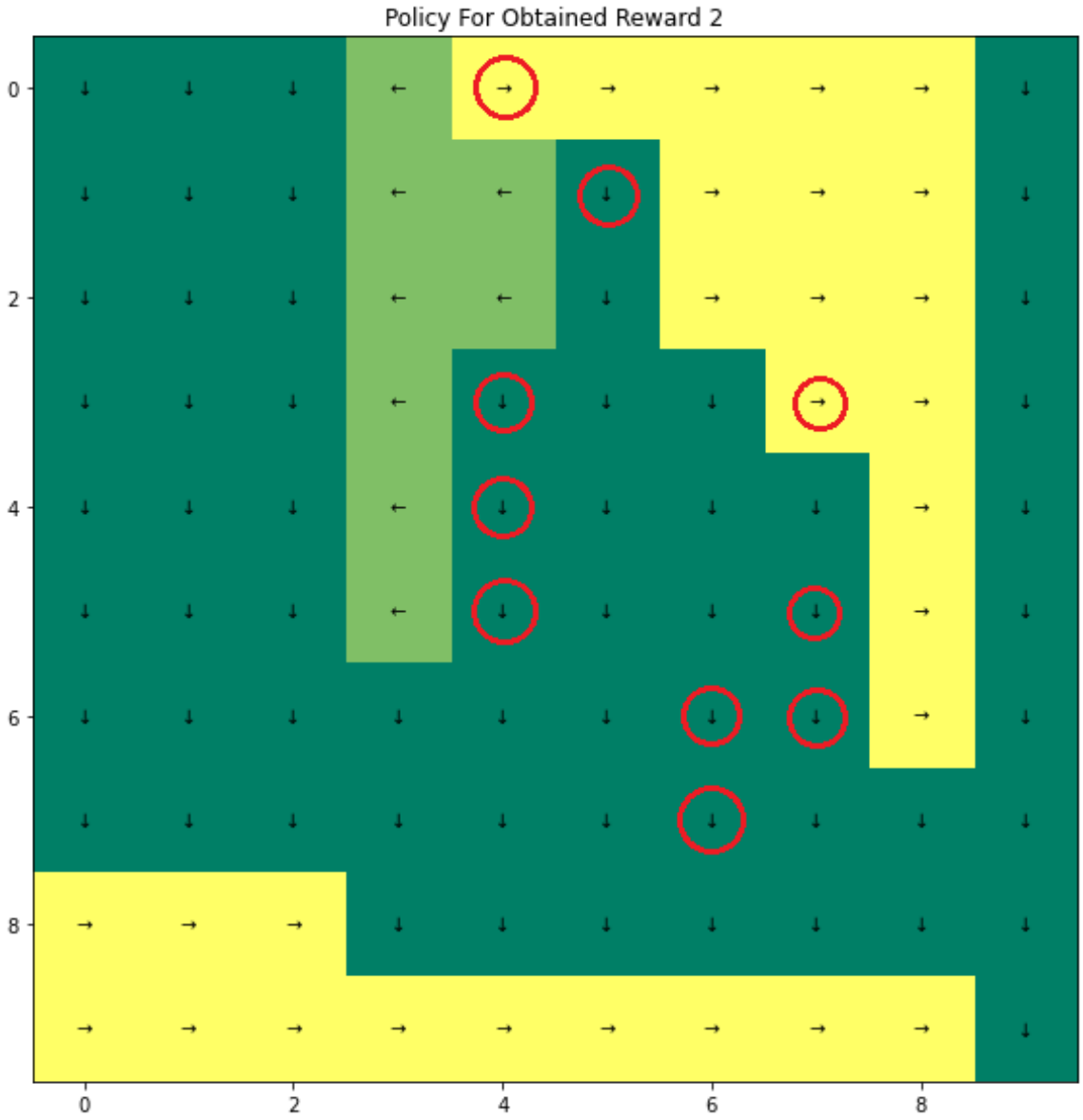


Figure 21: Optimal Policy for Extracted Reward 2

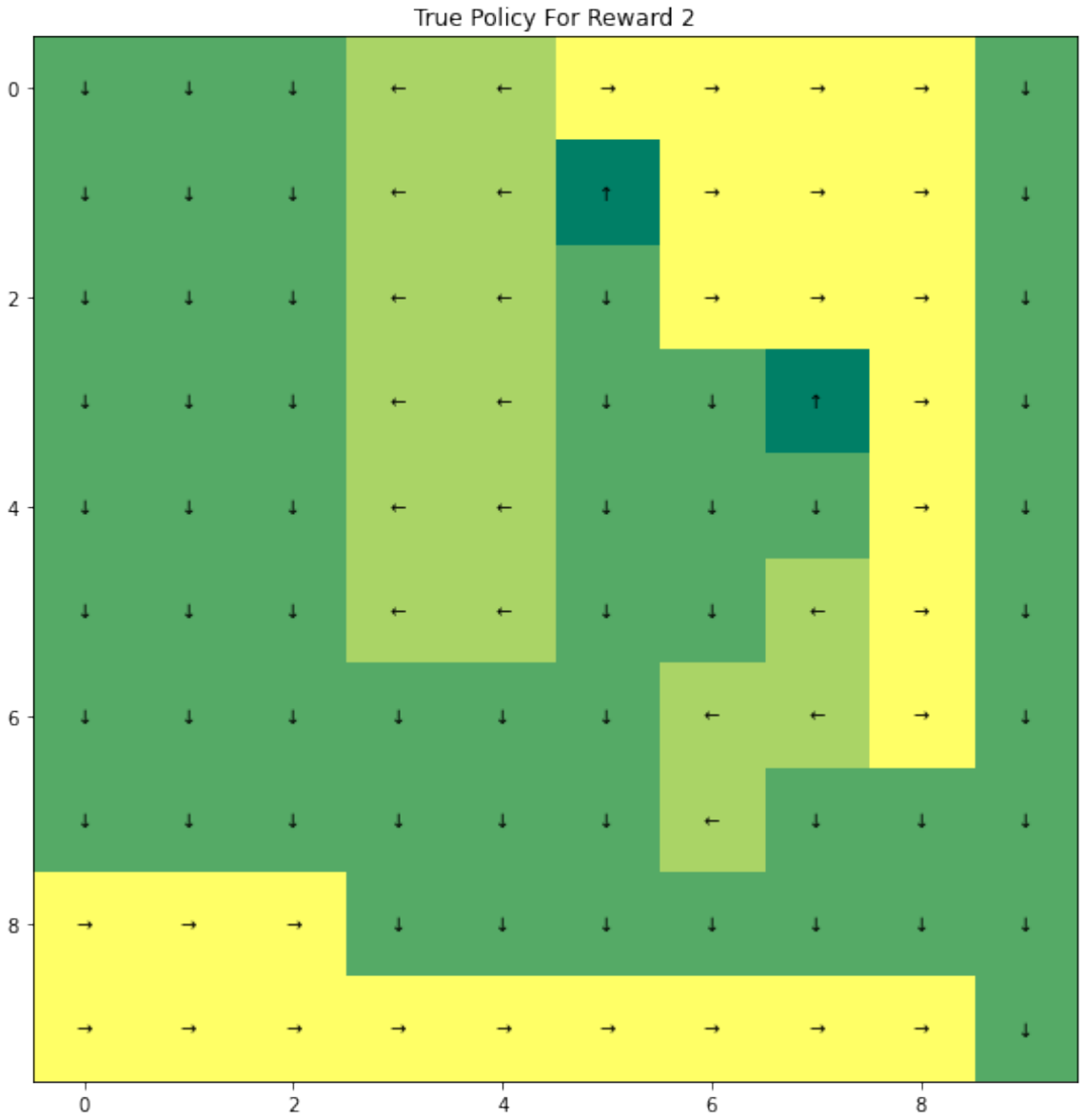


Figure 22: Ground Truth Policy for Reward 2

Question 24

Similarities: The accuracy is 90%, which means most of derived policy matches with the ground truth's. If we start from any edge, the policy guides us to the bottom right.
Differences:

- As accuracy is 90%, there are 10 policies that are different from the ground truth.
- States 5 and 16, while being different from the ground truth, are still reasonable and are able to go to the highest reward (state 99) without stepping into the penalty zone.
- The rest of the state in red are either in the penalty zone or close to the penalty zone. The agent keeps going into the penalty areas instead of moving away. One possible reason is because of lenient convergence criteria.

Question 25

We believed that the critical discrepancy can be solved by increasing ϵ to make the convergence criteria smaller. Minimizing ϵ reduces the noise caused by fluctuations in values generated by the IRL algorithm. One disadvantage we perceive is that the algorithm takes much longer to solve due to the stricter requirement.

By lowering ϵ from 0.01 to 10^{-8}, we were able to achieve 93% accuracy (Fig. 23). We see that at states 35, 45, and 58, the agent is able to steer to the left, avoiding the negative zone.

At $\epsilon = 10^{-10}$, the accuracy is 100%, which means the derived policy totally matches the ground truth (Fig. 24).

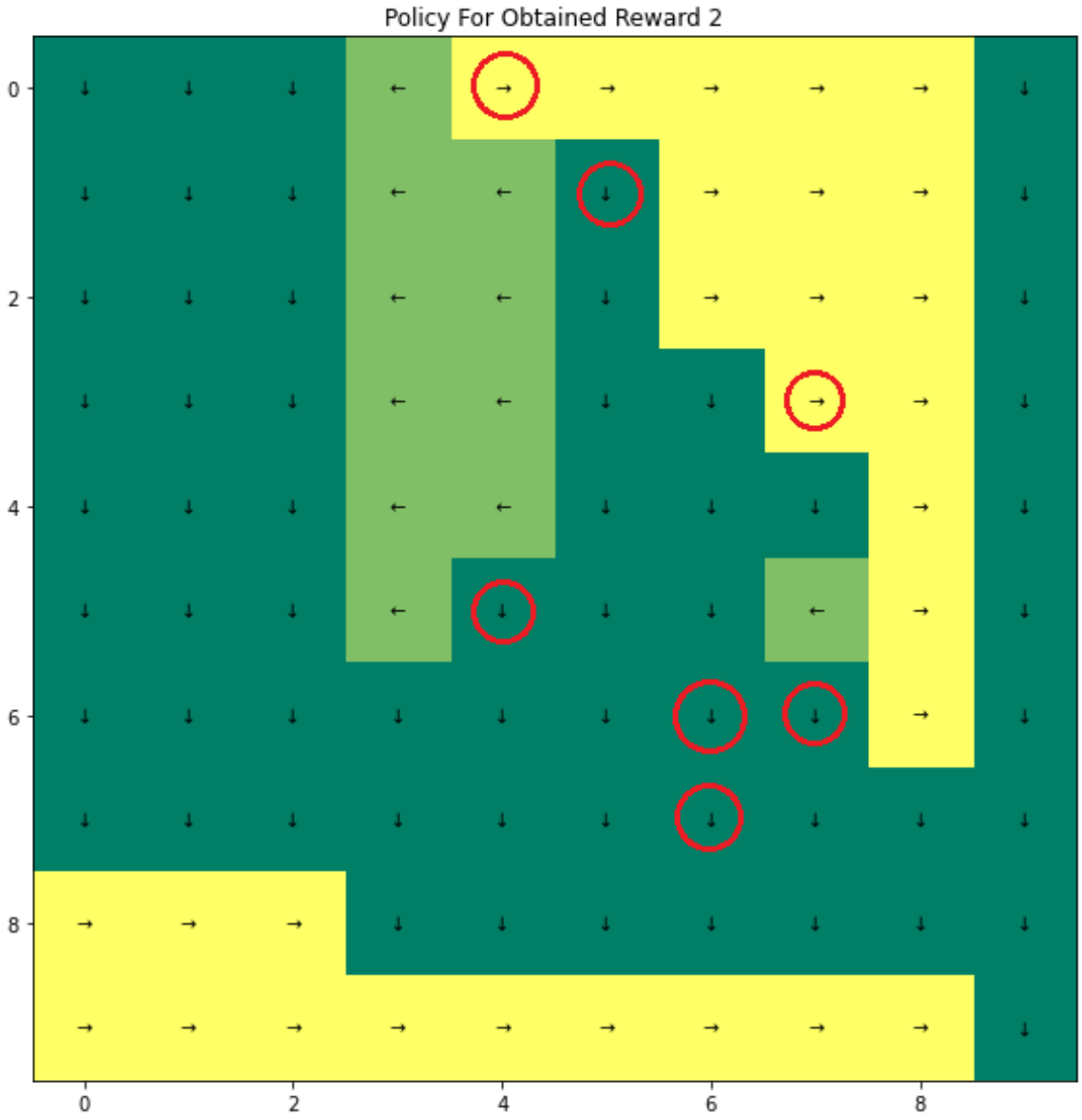


Figure 23: Optimal Policy for Extracted Reward 2, $\epsilon = 10^{-7}$

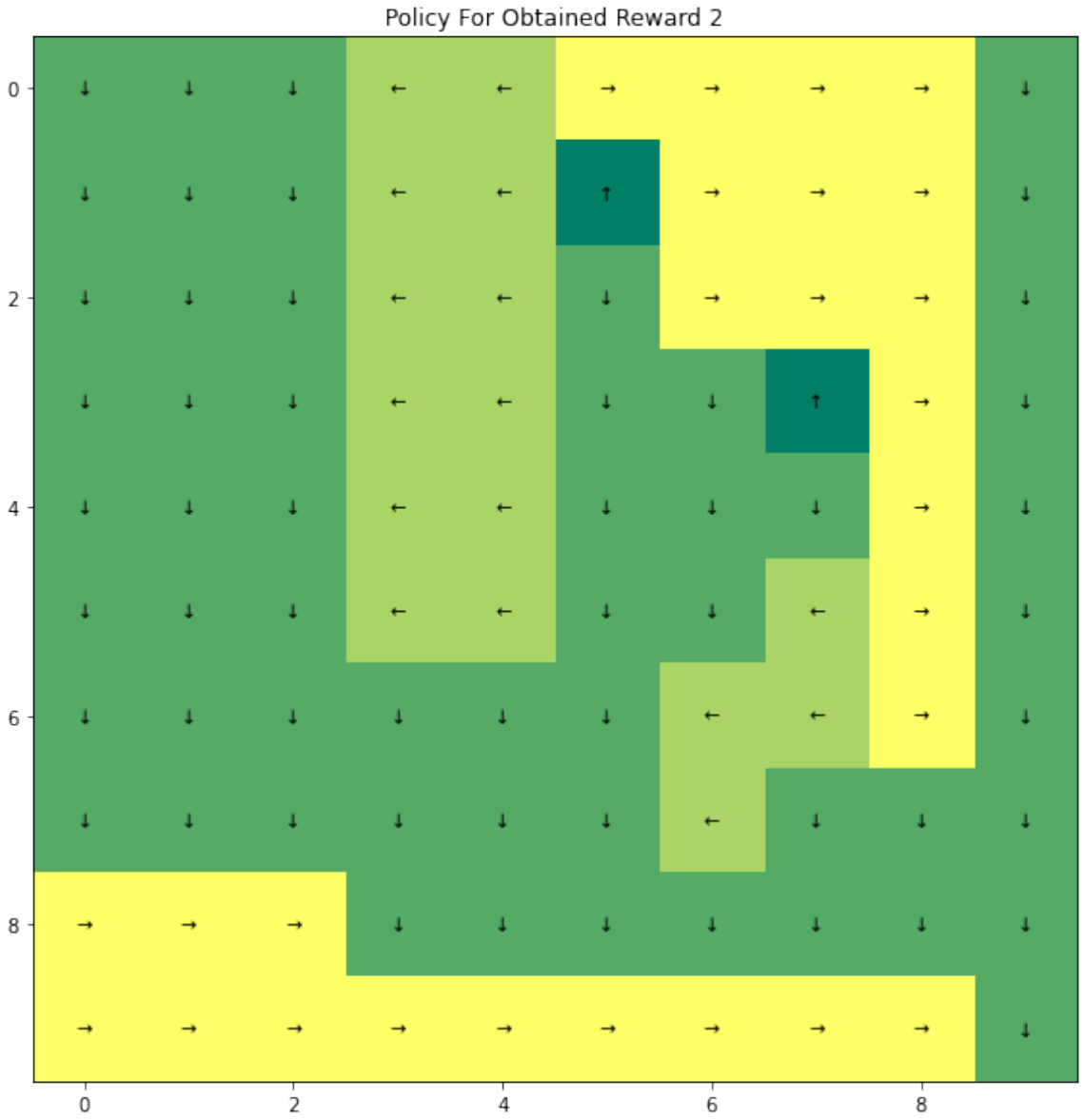


Figure 24: Optimal Policy for Extracted Reward 2, $\epsilon = 10^{-10}$