UCLA

ECE 232 - Large Scale Social and Complex Networks: Design and Algorithms
Spring 2021

Project 3: Reinforcement Learning and Inverse Reinforcement Learning

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

Question 1:

In this question, we are asked to plot the heatmaps of two reward functions for an agent navigating in a 2-D environment (grid-world). The plots are shown in Figure 1. To generate the plots, we used the `pcolor()` function from `matplotlib.pyplot` package.

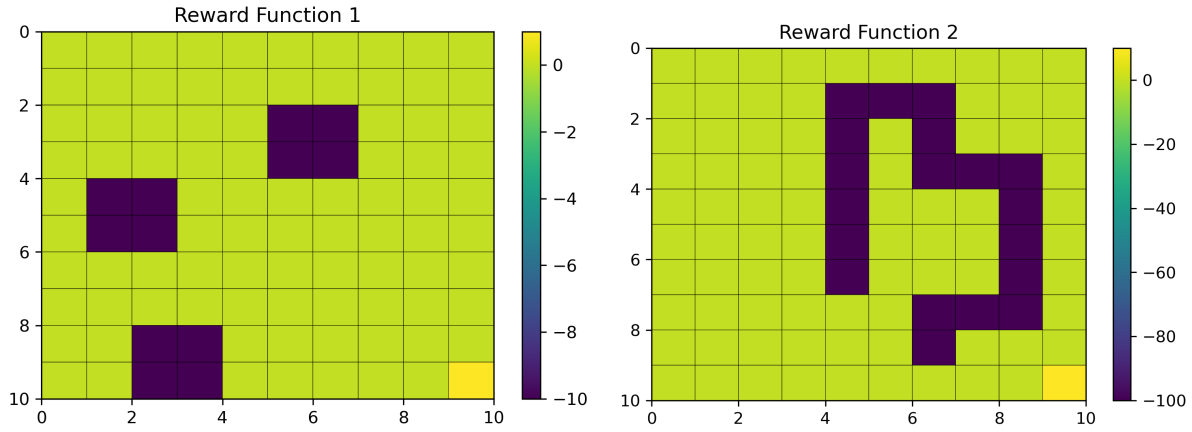


Figure 1: Heatmap visualization for ground-truth reward functions.

From the generated heatmaps, it is possible to identify the states where rewards are maximum and also the states to avoid within a single glimpse. In the case of our plots, the darker regions correspond to states with minimal reward, while brighter regions correspond to states where reward is maximum. In particular, state 99 (lower right corner) provides the highest reward for both reward functions.

Question 2:

In this question, we are asked to create an environment for the agent by setting up the state-space, action set, transition probabilities, discount factor, and reward function, implement the initialization and estimation steps in value iteration algorithm, generate a plot of the optimal values of each state and snapshots of state values at linearly distributed steps until convergence.

Reinforcement learning (RL) involves finding suitable actions in certain scenarios to maximize a given reward and achieve a goal, discovering policies π through interaction with the environment. It involves an agent, which interacts with the environment through some action A_t after observing the environment O_t at time t. The reinforcement learning agent interacts with the environment and it learns from the consequences of its actions. The environment accepts the interaction and gives the reward R_t at time t. The agent utilizes its own past experiences to select future actions. The sequence of actions, observations and rewards until time t forms the history h_t at time t. Assuming that the environment is fully observable, with $S_t = O_t$:

$$h_t = \{O_1, R_1, A_1, O_2, R_2, A_2, ..., A_{t-1}, O_t, R_t\}$$

The goal of the agent is to find A_t as a function of $h_t : f(h_t)$, thereby approximating $f(h_t)$ to maximize the cumulative reward for horizon T.

In this project, we model the environment using Markov Decision Process (MDP). The Markov process is a stochastic process whose future probabilities are determined by the most recent values seen. A Markov state is an information state in which the future is independent of the past given present, i.e. the current state can be thought of as containing the history in a latent space:

$$\mathcal{P}(S_{t+1}|S_t, A_t, ..., S_1, A_1) = \mathcal{P}(S_{t+1}|S_t, A_t)$$

In MDP, the agent has control over the actions it can take at particular states. The agent not only receives a reward from the environment, but is able to transit to the next state governed by the state transition matrix. The domain of an MDP is $\{S, A, P^a_{ss'}, R^a_{ss'}, \gamma\}$, where:

- S: State, representation/configuration of the environment. For most Markov processes, we assume that the environment state S^e_t is the same as the observation state O_t (fully observable environment). In the case of this project, the state space is a 10×10 square grid with 100 states. The state space is discrete.
- A: Action, interaction with the environment. In the case of this project, an agent can choose one of four actions within the grid world: move up, move down, move left and move right. The action space is also discrete like the state space.
- $P^a_{ss'}$: State transition matrix, describes a Markov chain / transition probability from one state S to another state S' for action a. It can be thought of as a model of the system when coupled with R. It follows the Markov process. For this project, each action corresponds to a movement in the intended

direction with probability $1 - w$, but has a probability of w of moving in a random direction instead due to wind. Several other heuristics are as follows:

- If two states S and S' are not neighboring, then $\mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a) = 0$.
- For inner grid states (non-boundary states), an agent moves in the desired direction (via desired action) with probability $1 - w + 0.25w$, while the probability of moving in the other three directions due to wind is $0.25w$. The agent cannot be in the same state at the next time step.
- At corner states (states with only two neighboring states), an agent can be blown off the grid in two directions. For moving off the grid consciously, the probability of being off the grid is $1 - w + 0.25w + 0.25w$, while the probability of being blown off the grid unconsciously (when the agent actually decides to stay on the grid) is $0.25w + 0.25w$.
- At edge states (states with three neighboring states), an agent can be blown off the grid in one direction. For moving off the grid consciously, the probability of being off the grid is $1 - w + 0.25w$, while the probability of being blown off the grid unconsciously (when the agent actually decides to stay on the grid) is $0.25w$.

- $R^a_{ss'}$: Immediate scalar feedback from the environment indicating how the agent is performing when transitioning from state S to another state S' for action a. The ground truth rewards from the environment are provided by the two reward functions from Question 1. In other words, for this project, we assume that the reward function only depends on the state the agent transitions to (S').
- $\gamma \in [0, 1)$ is the discount factor, which is used to compute the present value of future reward. If γ is close to 0, future rewards are discounted more and vice versa.

To find the optimal policy π^*, we can use the value iteration algorithm. Policy is defined as a set of rules/probability distribution that maps states to actions (describes agent's behavior). The state-value function V is the total discounted reward an agent is expected to receive in its lifetime starting from state S and following policy π. V is used for policy evaluation and value iteration (find optimal value for fixed policy) through dynamic programming. Optimal state-value function for all policies, V^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

An optimal policy π^* is one whose return/reward is greater than all other policies π' for all states, i.e.

$$\pi^* \geq \pi' \iff V^{\pi^*}(s) \geq V^{\pi'}(s), \ \ \forall s \in S$$

In other words, optimal policies achieve V^*, given by the recursive Bellman optimality equation:

$$V^*(s) = \max_{a \in A} \mathbb{E}[r_t + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] = \max_{a \in A} P^a_{s',s}[R^a_{s',s} + \gamma V^*(s')]$$

To solve for V^* iteratively, the value iteration algorithm is used. It is composed of three steps:

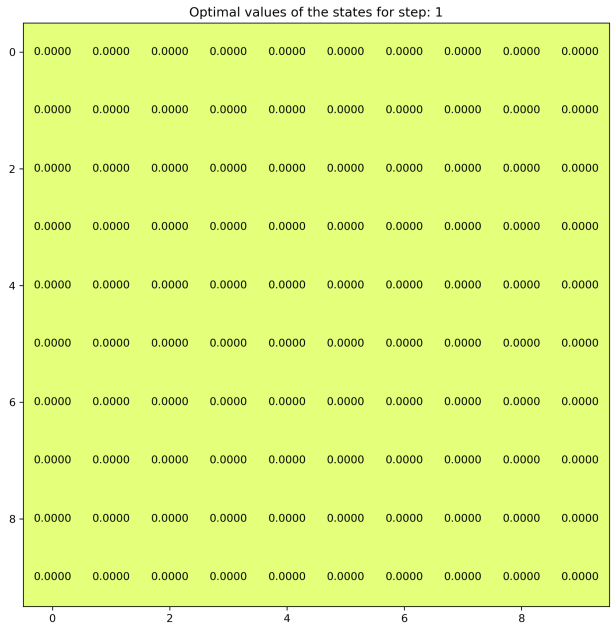
- Initialization: Set the value for all states to 0 and initialize variable Δ to ∞. Select a value of ϵ, which dictates when to stop the value iteration algorithm (convergence criterion).
- Estimation: While $\Delta > \epsilon$, for all states s in the state space S:
    - Set Δ to 0.
    - Copy old value of V(s) to variable v.
    - Compute V(s) using Bellman equation.
    - Find the maximum of Δ and $|v - V(s)|$ and set the value of Δ to the maximum value.
- Computation: Find the actions corresponding to the optimal policy for each state:

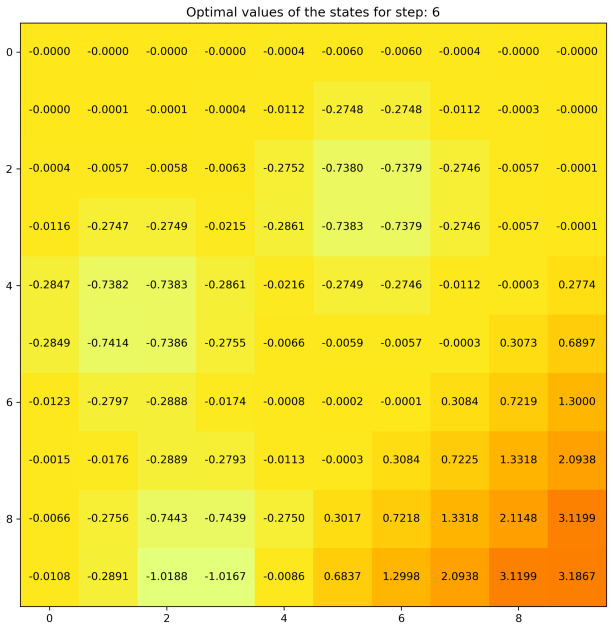
$$\pi^*(s) \leftarrow \arg\max_{a \in A} P^a_{s',s}[R^a_{s',s} + \gamma V^*(s')]$$

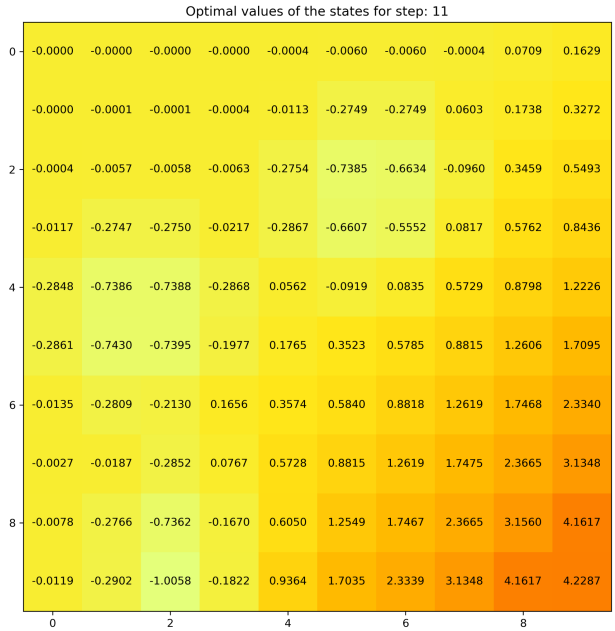
For this question, we set w to 0.1, γ to 0.8, ϵ to 0.01 and use reward function 1. The number of states is 100 and the number of actions is 4. Figure 2 shows the optimal values for all the states found using value iteration.

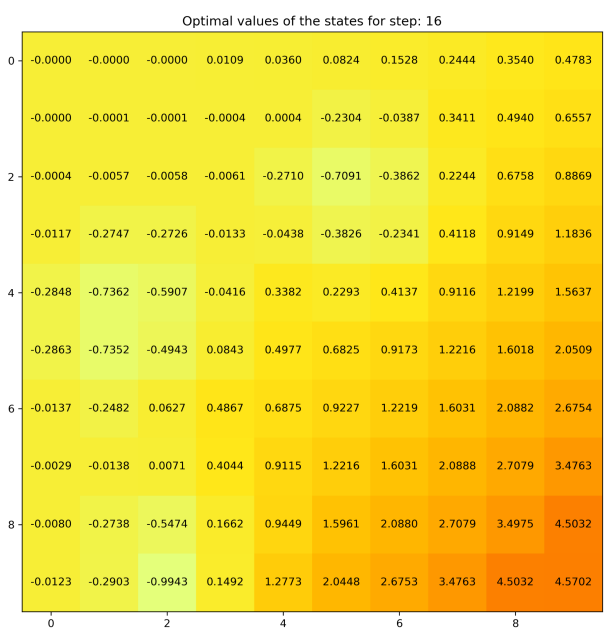
Figure 2: Optimal values of the states for reward function 1.

We observed that the algorithm converges in 22 steps (**N = 22).** Figure 3 shows the snapshots of the state values for 5 different steps linearly distributed from 1 to N. **We show the snapshots for step 1, step 6, step 11, step 16 and step 21**.









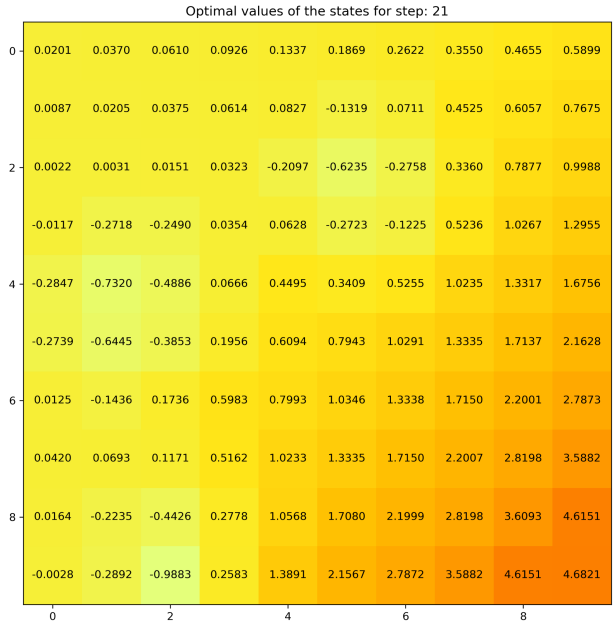


Figure 3: Snapshots of state values in value iteration for steps 1, 6, 11, 16 and 21.

From Figure 2 and 3, we can see that the state 99 (the state in the lower right corner) has the highest optimal value, which is evident because state 99 yields the highest reward for reward function 1. As one moves further from state 99, the optimal value of each state decreases slowly. In addition, the blocks of states which yield negative reward show negative optimal values. The neighboring states near these blocks also have a lower optimal value compared to the neighboring states near the state yielding the highest reward. In other words, as an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase.

In addition, from the snapshots in Figure 3, most of the states have a value of 0 during the earlier steps, yielding a sparse $V(s)$ matrix. However, as steps increase, the state values near state 99 (state with highest reward) starts to increase gradually, while the state values near the states with negative reward start to decrease progressively. Since state 99 has the highest reward, the values increase much faster for state 99 and its neighboring states. As the algorithm nears convergence, the sparsity in $V(s)$ disappears, with most of the states assigned a certain non-zero optimal value, with the states near state 99 being assigned very high and positive values and states near the blocks with negative reward being assigned negative values. We can also observe that the rate of change of state values is faster during initial steps, following a logarithmic rate of convergence until $\Delta < \epsilon$.

Question 3:

Figure 4 shows the heatmap of optimal state values across the 2D grid for reward function 1 obtained in Question 2.

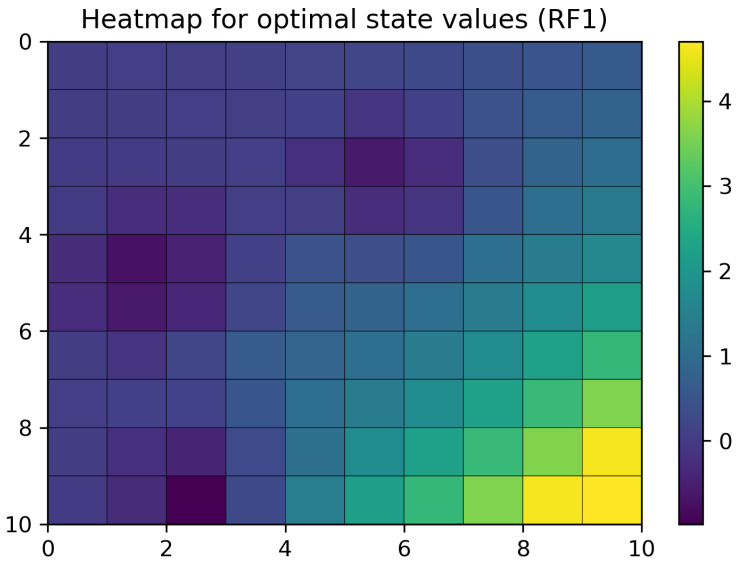


Figure 4: Heatmap visualization for optimal state values (reward function 1).

Question 4:

We can make several inferences from the heatmap in Figure 4. Note that the darker regions correspond to states with low optimal values, while the brighter regions correspond to states with high optimal values. State 99 has the highest reward and highest optimal value.

- As one moves further from state 99, which is the state providing the highest reward, the optimal value of each state decreases slowly. In addition, the neighboring states near the blocks yielding negative reward also have a lower optimal value compared to the neighboring states near the state yielding the highest reward. In other words, as an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase. The further a state is from the state yielding the highest reward, the lower its value. Mathematically:

$$V^*(s) = \max_{a \in A} \mathbb{E}[r_t + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] = \max_{a \in A} P^a_{s',s}[R^a_{s',s} + \gamma V^*(s')]$$

  From the above equation, we see that $V(s) \propto R^a_{s',s}$.

- There is a visible, progressive and gradual decay of optimal values surrounding the state with the highest reward. This is due to the discount factor in the Bellman equation, which discounts future rewards.

- The patterns seen in the heatmap of reward function 1 in Figure 1 are roughly visible in the heatmap of optimal values. There are three blocks of negative rewards in Figure 1 for reward function 1. One can almost make out the same three blocks from the heatmap of optimal values within the same expected regions in the state space. In addition, the region near the state with the highest reward is also brighter. This provides us with an intuition that it is possible to extract reward function by observing how the environment or an expert behaves. This is roughly what is done in inverse reinforcement learning (IRL).

Question 5:

Figure 5 shows the optimal actions taken by the agent in the 2D grid for reward function 1, shown using arrows.

```
0 ['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
1 ['→' '→' '→' '↑' '↑' '↑' '→' '→' '↓' '↓']
2 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↓']
3 ['↑' '↑' '→' '↓' '↓' '↓' '↓' '→' '↓' '↓']
4 ['↑' '↑' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
5 ['↓' '↓' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
6 ['↓' '→' '→' '→' '→' '→' '→' '↓' '↓' '↓']
7 ['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
8 ['↑' '↑' '↑' '→' '→' '→' '→' '→' '→' '↓']
9 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 5: Optimal actions undertaken by the agent for reward function 1, shown using arrows.

We can make several observations from Figure 5:

- All the arrows eventually lead to a path towards state 99, which has the highest reward out of all the states. Furthermore, when supported by the heatmap in Figure 4, the arrow at a certain state tends to point towards the state that will eventually maximize the expected reward. In addition, the agent tends to move away from the region with negative rewards (darker regions on heatmap) and the arrows point to the brighter regions in the heatmap in Figure 4.. This matches our intuition because the goal of the agent is to maximize the total cumulative reward it can get within a finite horizon, and thus should aim to reach state 99 as the rewards from other states are either negative or zero.
- 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 evident when we check which way the arrow is pointing given the optimal values of neighboring states from Figure 2. At each state, the arrow always points towards that neighboring state whose optimal value is highest among all neighbors. As a result, for this particular case, even if the agent is unaware of future optimal values, the agent can still build an optimal policy just by observing the local optimal values within the neighborhood of each state. Mathematically, in the value iteration algorithm, the optimal policy for each state is obtained as follows:

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

From the above equation, we see that optimal policy depends on the optimal values of the neighboring states, apart from discount factor (which is fixed in our case), transition matrix and rewards. Thus, value iteration leads the agent to choose an action such that the expected reward is maximized, based on observation of local optimum values.

Question 6:

Figure 6 shows the optimal values for all the states found using value iteration for reward function 2.

Figure 6: Optimal values of the states for reward function 2.

From Figure 6, we can see that the state 99 (the state in the lower right corner) has the highest optimal value, which is evident because state 99 yields the highest reward for reward function 2. As one moves further from state 99, the optimal value of each state decreases slowly. In addition, the group of states which yield negative reward show negative optimal values. The neighboring states near these blocks also have a lower optimal value compared to the neighboring states near the state yielding the highest reward. In other words, as an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase.

Question 7:

Figure 7 shows the heatmap of optimal state values across the 2D grid for reward function 2 obtained in Question 6.

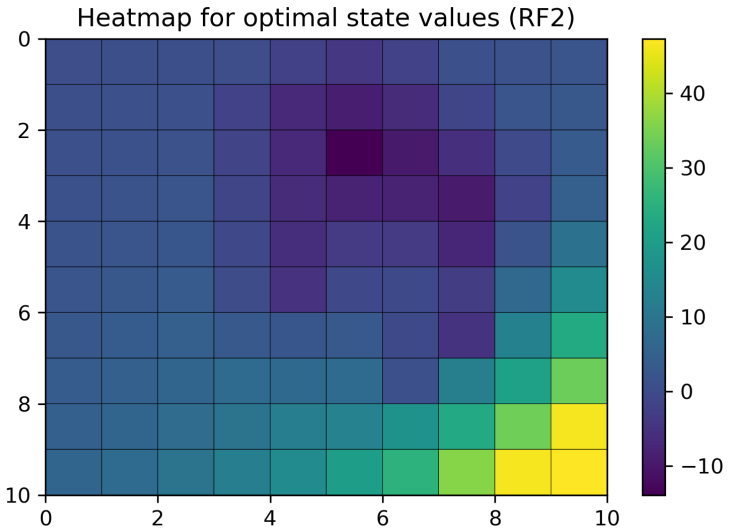


Figure 7: Heatmap visualization for optimal state values (reward function 2).

We can make several inferences from the heatmap in Figure 7. Note that the darker regions correspond to states with low optimal values, while the brighter regions correspond to states with high optimal values. State 99 again has the highest reward and highest optimal value.

- As one moves further from state 99, which is the state providing the highest reward, the optimal value of each state decreases slowly. In addition, the neighboring states near those states yielding negative reward also have a lower optimal value compared to the neighboring states near the state yielding the highest reward. In other words, as an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase. The further a state is from the state yielding the highest reward, the lower its value. Mathematically:

$$V^*(s) = \max_{a \in A} \mathbb{E}[r_t + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] = \max_{a \in A} P^a_{s',s}[R^a_{s',s} + \gamma V^*(s')]$$

  From the above equation, we see that $V(s) \propto R^a_{s',s}$.

- The patterns seen in the heatmap of reward function 2 in Figure 1 are roughly visible in the heatmap of optimal values in Figure 7. From Figure 1, we see that the negative rewards form a snake-like chain in the state-space. This chain is roughly visible in the heatmap, with the darkest regions corresponding to the states with lowest rewards. In addition, the region near the state with the highest reward is also

brighter. This provides us with an intuition that it is possible to extract reward function by observing how the environment or an expert behaves. This is roughly what is done in IRL.

- We see that state 52 has the lowest optimal value and is being shown the darkest in the heatmap. This is because state 52 is surrounded by states with negative rewards on three sides with only one path that does not penalize the agent. Thus, value iteration encourages the agent to avoid this state, as the probability of landing in a state with negative reward is the highest among all the states if the agent lands on this state.

- Even though some of the states with negative rewards are closer to state 99 than some farther states, the farther states have higher optimal values than the states near the negative-reward chain. This indicates that being close to a state with high reward does not necessarily mean a state will have a high optimal value, if most of the neighbors of that state have negative rewards. This is particularly visible for states near the upper left corner, which have a higher optimal value than the states near the negative-reward chain.

- There is a visible, progressive and gradual decay of optimal values surrounding the state with the highest reward. This is due to the discount factor in the Bellman equation, which discounts future rewards.

Question 8:

Figure 8 shows the optimal actions taken by the agent in the 2D grid for reward function 2, shown using arrows.

```
0 ['↓' '↓' '↓' '←' '←' '→' '→' '→' '→' '↓']
1 ['↓' '↓' '↓' '←' '←' '↑' '→' '→' '→' '↓']
2 ['↓' '↓' '↓' '←' '←' '↓' '→' '→' '→' '↓']
3 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↑' '→' '↓']
4 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↓' '→' '↓']
5 ['↓' '↓' '↓' '←' '←' '↓' '↓' '←' '→' '↓']
6 ['↓' '↓' '↓' '↓' '↓' '↓' '←' '←' '→' '↓']
7 ['↓' '↓' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
8 ['→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓' '↓']
9 ['→' '→' '→' '→' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 8: Optimal actions undertaken by the agent for reward function 2, shown using arrows.

We can make several observations from Figure 8:

- All the arrows eventually lead to a path towards state 99, which has the highest reward out of all the states. Furthermore, when supported by the heatmap in Figure 7, the arrow at a certain state tends to point towards the state that will eventually maximize the expected reward. In addition, the agent tends to move away from the region with negative rewards (darker regions on heatmap) and the arrows point to the brighter regions in the heatmap in Figure 7. This matches our intuition because the goal of the agent is to maximize the total cumulative reward it can get within a finite horizon, and thus should aim to reach state 99 as the rewards from other states are either negative or zero.

- We also notice how the agent attempts to move away from the snake-like negative reward chain towards regions which are geodesically further away from state 99. This is because the goal of the agent is to not reach the state with maximum reward in the shortest path, but to maximize the expected cumulative reward within the finite horizon. The agent tries to make sure it does not incur negative reward during its journey, even if it means taking a longer but optimal (in terms of total reward) path.

- 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 evident when we check which way the arrow is pointing given the optimal values of neighboring states from Figure 6. At each state, the arrow always points towards that neighboring state whose optimal value is highest among all neighbors. As a result, for this particular case, even if the agent is unaware of future optimal values, the agent can still build an optimal policy just by observing the local optimal values within the neighborhood of each state. Mathematically, in the value iteration algorithm, the optimal policy for each state is obtained as follows:

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

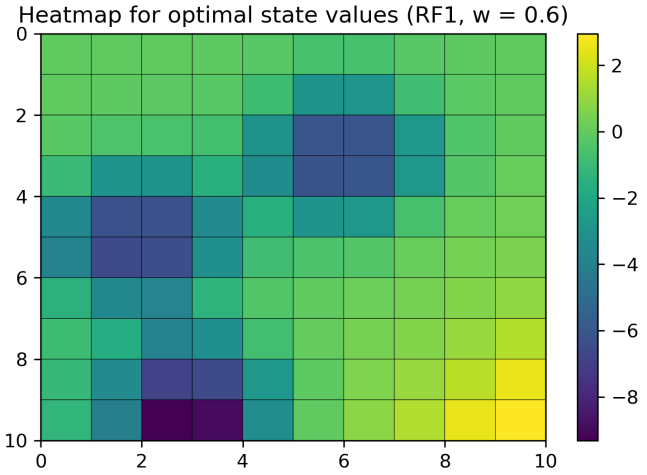
From the above equation, we see that optimal policy depends on the optimal values of the neighboring states, apart from discount factor (which is fixed in our case), transition matrix and rewards. Thus, value iteration leads the agent to choose an action such that the expected reward is maximized, based on observation of local optimum values.

Question 9:

In this question, we are asked to change the value of w from 0.1 to 0.6 and visualize its effects on the learned policy. Figure 9 shows the grid of optimal values, heatmap plot of optimal values and plot of optimal actions (using arrows) for reward function 1, while Figure 10 shows the same for reward function 2.

a.

b.

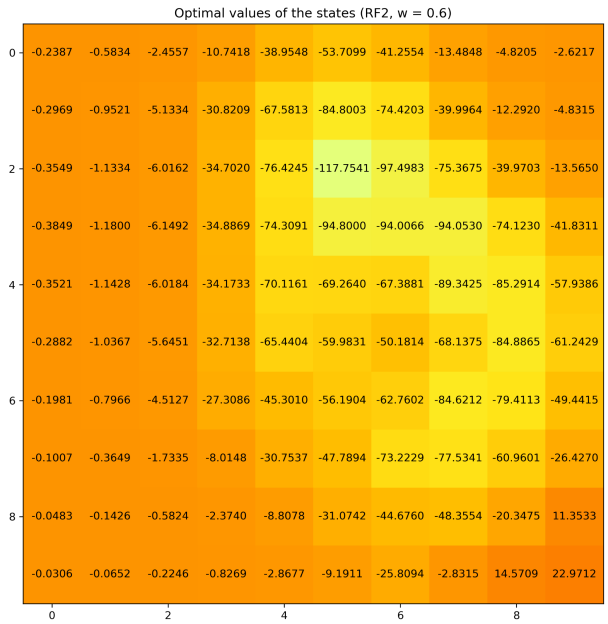


c.

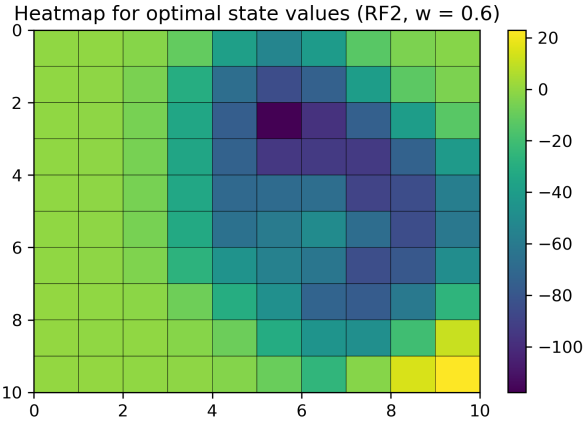
```
0 ['↑' '←' '←' '←' '←' '←' '→' '→' '→' '↑']
1 ['↑' '↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓']
2 ['↑' '↑' '↑' '↑' '←' '↑' '→' '→' '→' '↓']
3 ['↑' '↑' '↑' '↑' '←' '↓' '→' '→' '→' '↓']
4 ['↑' '↑' '↑' '↑' '↓' '↓' '↓' '↓' '↓' '↓']
5 ['↓' '↓' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
6 ['↓' '←' '→' '→' '→' '→' '→' '↓' '↓' '↓']
7 ['←' '←' '←' '→' '→' '→' '→' '→' '↓' '↓']
8 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']
9 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 9: (**a**) Optimal values of the states for reward function 1 for $w = 0.6$, (**b**) Heatmap visualization for optimal state values (reward function 1) for $w = 0.6$, (**c**) Optimal actions undertaken by the agent for reward function 1, shown using arrows for $w = 0.6$.

a.



b.



c.

```
0 ['↑' '←' '←' '←' '←' '←' '→' '→' '→' '↑']
1 ['↑' '←' '←' '←' '←' '↑' '→' '→' '↑' '↑']
2 ['↑' '←' '←' '←' '←' '↓' '→' '→' '↑' '↑']
3 ['↓' '←' '←' '←' '←' '↓' '↓' '↑' '↑' '↑']
4 ['↓' '←' '←' '←' '←' '↓' '↓' '←' '→' '↑']
5 ['↓' '←' '←' '←' '←' '→' '←' '←' '→' '↓']
6 ['↓' '←' '←' '←' '←' '↓' '↑' '←' '→' '↓']
7 ['↓' '←' '←' '←' '←' '←' '←' '↓' '↓' '↓']
8 ['↓' '←' '←' '←' '←' '←' '↓' '↓' '↓' '↓']
9 ['↓' '←' '←' '←' '←' '←' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 10: (**a**) Optimal values of the states for reward function 2 for $w = 0.6$, (**b**) Heatmap visualization for optimal state values (reward function 2) for $w = 0.6$, (**c**) Optimal actions undertaken by the agent for reward function 2, shown using arrows for $w = 0.6$.

We can make several observations from all the plots in Figure 9 and 10:

- The optimal values for the states neighboring state 99 (state with highest reward), as well as the value for state 99 itself has decreased. The negative value of those states with negative rewards have been amplified. In summary, more states take on higher or lower values instead of just the neighboring states with positive and negative rewards, giving the agent more choice on where to move, ultimately decreasing the probability of landing in the state with highest reward.

- For reward function 2, the heatmap and value plots point out that the path to the state with highest optimal value has been cut-off due to the increase in adversarial influence by the negative-reward chain over a longer state-space. This is due to amplification of negative rewards and attenuation of positive rewards.
- From the policy plots, it is evident that the increase in influence of negative rewards leads to the agent often moving out of the grid. In addition, fewer paths now exist for both reward functions that can lead to the agent to state 99. This happened because the states neighboring state 99 do not provide any positive or negative reward to the agent (0 reward), causing the values for those states to be overpowered by the states where reward is negative.
- We can observe some local optima in the policy plots, where the agent just oscillates between two states. This is called deadlock condition.
- All of the above reasons can be attributed to the exploration versus exploitation dilemma. We can think of w as the probability of exploration. Exploitation is making the best possible decision (by maximizing future reward), and exploration involves taking an immediately suboptimal action to gather information. While the suboptimal action will necessarily involve a reduced amount of reward in the immediate future, it may allow the agent to learn better strategies that enable policy improvement in the long term. If there is no balance between exploration and exploitation, it would lead to a bad sample complexity. In this case, since w is very high, the balance has not been achieved, leading to the agent exploring more suboptimal actions at each state rather than exploiting knowledge of optimal values reasonably. This leads to the extracted policy being suboptimal.

The appropriate value of w is 0.1, as the agent is more likely to reach the state with the highest reward while having the option to perform limited exploration and less likely to get stuck in local optima or get blown off the grid. In addition, the radius of adversarial influence of negative reward chains/blocks are reduced and more paths point towards state 99 when w is 0.1.

Question 10:

In this question, we are asked to recast the linear programming (LP) formulation of IRL using block matrices for ease of implementation.

IRL is the task of extracting the reward function from the optimal policy, particularly when the reward function is not known a priori. The goal of IRL is to find the set of possible reward functions $R^a_{ss'}$ such that π^* is an optimal policy, given state space, action space, optimal deterministic policy π^* and state transition matrix. In IRL, the agent observes an expert's behavior and learn's the expert's reward function to generate the optimal policy. An expert's optimal policy can be used to extract the reward function, thereby computing the optimal policy of the agent. We prefer learning the reward function rather than the policy directly because the reward function provides a robust, succinct and transferable definition of the task at hand.

The LP formulation of IRL is given by:

$$\max_{\mathbf{R},t_i,u_i} \text{s.t.} \begin{cases} \sum_{i=1}^{|S|}(t_i - \lambda u_i) \\ [(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R}] \geq t_i, \forall a \in A \setminus a_1, \forall i \\ (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R} \succeq 0, \ \ \forall a \in A \setminus a_1 \\ -\mathbf{u} \preceq \mathbf{R} \preceq \mathbf{u} \\ |\mathbf{R}_i| \leq R_{\max}, i = 1, 2, ..., |S| \end{cases}$$

$\mathbf{R}$ is the reward vector $(\mathbf{R}(i) = \mathbf{R}(s_i))$, A and S are action and state spaces respectively, $\mathbf{P}_a$ corresponds to transition matrix for action a, λ is the adjustable penalty coefficient, t_i and u_i are extra optimization variables needed to pose the problem as LP formulation, γ is the discount factor, $R_{\max}$ is the maximum value of ground truth reward. The goal is to find a feasible value of $\mathbf{R}$, also called linear feasibility problem. The policy π given by $\pi(s) \equiv a_1$ is optimal for all values of $a \in A$ except a_1 if the reward satisfies:

$$(\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R} \succeq 0$$

To tackle degeneracy (prevent optimal value of $\mathbf{R}$ be 0 and choose $\mathbf{R}$ such that deviating from the optimal policy should reduce the total reward as much as possible) and find non-trivial reward vectors $\mathbf{R}$, we introduce the 2nd and 5th constraint in the LP formulation.

For simplicity, we want to recast the LP formulation as follows:

$$\max_{\mathbf{x}} \text{s.t.} \begin{cases} \mathbf{c}^T\mathbf{x} \\ \\ \mathbf{D}\mathbf{x} \preceq \mathbf{b}, \forall a \in A \setminus a_1 \end{cases}$$

We need to express $\mathbf{c}$, $\mathbf{x}$, $\mathbf{D}$ and $\mathbf{b}$ in terms of $\mathbf{R}$, $R_{\max}$, $\mathbf{P}_{a_1}$, $\mathbf{P}_a$, t_i, $\mathbf{u}$ and λ. $\mathbf{c}$, $\mathbf{x}$, $\mathbf{D}$ and $\mathbf{b}$ are given by:

$$\mathbf{c} = \begin{bmatrix} \mathbf{1}_{|S|\times 1} \\ -\lambda \cdot \mathbf{1}_{|S|\times 1} \\ \mathbf{0}_{|S|\times 1} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} \mathbf{t} \\ \mathbf{u} \\ \mathbf{R} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \mathbf{0}_{(2\cdot(|A|-1)\cdot|S|)\times 1} \\ \mathbf{0}_{(2\cdot|S|)\times 1} \\ R_{\max} \cdot \mathbf{1}_{|S|\times 1} \\ R_{\max} \cdot \mathbf{1}_{|S|\times 1} \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{I}_{((|A|-1)\cdot|S|)\times|S|} & \mathbf{0}_{((|A|-1)\cdot|S|)\times|S|} & -[(\mathbf{P}_{a_1} - \mathbf{P}_{a_{2,\ldots,|A|}})(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}]_{((|A|-1)\cdot|S|)\times|S|} \\ \\ \mathbf{0}_{((|A|-1)\cdot|S|)\times|S|} & \mathbf{0}_{((|A|-1)\cdot|S|)\times|S|} & -[(\mathbf{P}_{a_1} - \mathbf{P}_{a_{2,\ldots,|A|}})(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}]_{((|A|-1)\cdot|S|)\times|S|} \\ \mathbf{0}_{|S|\times|S|} & -\mathbf{I}_{|S|\times|S|} & -\mathbf{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & -\mathbf{I}_{|S|\times|S|} & \mathbf{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & \mathbf{I}_{|S|\times|S|} \\ \mathbf{0}_{|S|\times|S|} & \mathbf{0}_{|S|\times|S|} & -\mathbf{I}_{|S|\times|S|} \end{bmatrix}$$

In our case, $|A| = 4$ and $|S| = 100$.

Question 11:

In this question, we are asked to find the effect of λ on the accuracy of the IRL algorithm for reward function 1. The accuracy of the IRL algorithm is given as:

$$\text{Accuracy} = \frac{\sum_{s \in S} m(s)}{|S|}, \quad m(s) = \begin{cases} 1, O_A(s) = O_E(s) \\ 0, \text{else} \end{cases}$$

where, $O_A(s)$ refers to the agent's optimal action at state s found from the extracted reward function from LP formulation of IRL, while $O_E(s)$ refers to the expert's optimal action at state s, in our case, the policies obtained in Question 5.

We sweep λ for 500 evenly spaced points within 0 and 5. We use `solvers.lp()` function from `cvxopt` package in Python to solve the recasted LP formulation after extracting $\mathbf{c}$, $\mathbf{x}$, $\mathbf{D}$ and $\mathbf{b}$ using the derived block matrix notation. We then perform value iteration on the extracted reward to obtain the agent's optimal policy. Figure 11 shows the plot of λ versus accuracy.

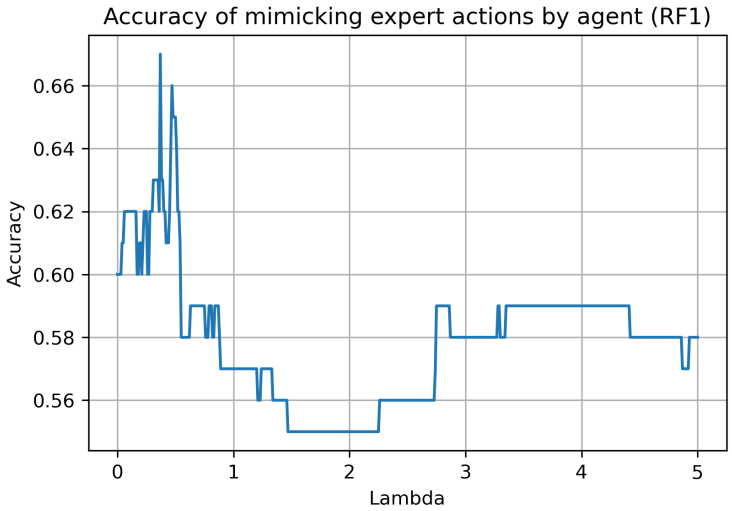


Figure 11: Effect of λ on the accuracy of IRL algorithm (reward function 1).

From Figure 11, we see that the accuracy initially increases with increase in λ, reaching a globally optimum policy and then drops. Afterwards, the accuracy increases slightly but gets stuck in a local optimum. This is expected because λ acts as a regularizer (L1 or Lasso normalization), encouraging simpler reward vectors. Finite values of λ improves robustness, succinctness and transferability of the extracted reward function such that it generalizes well when extracting the optimal policy, which explains the initial rise in accuracy. Large values of λ will yield reward vectors that underfit the tasks and are not expressive enough to derive the optimal global policy, with increased risk of being stuck in a locally optimum policy.

Question 12:

In this question, we are asked to compute the value of λ for which accuracy is maximum from Question 11. We refer to this value as $\lambda^{(1)}_{\max}$. From Figure 11, the maximum accuracy is *67%* and **corresponding $\lambda^{(1)}_{\max}$ is 0.37.**

Question 13:

In this question, we are asked to generate heat maps for extracted reward with $\lambda^{(1)}_{\max}$ as well as the ground truth rewards for reward function 1. Figure 12 shows the two plots side by side.

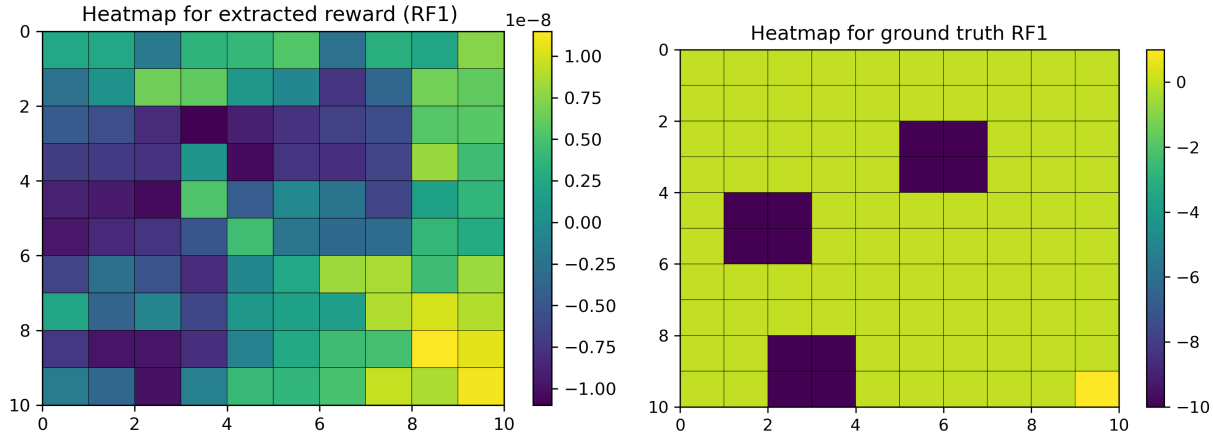


Figure 12: (Left) Heatmap for extracted reward with $\lambda^{(1)}_{\max}$, (Right) Ground truth reward (reward function 1).

From Figure 12, we can observe that the IRL algorithm has correctly identified the regions around state 99 as having high reward, while the regions surrounding the negative-reward blocks have been assigned low immediate rewards. For the extracted reward, as one moves further from state 99, which is the state providing the highest reward, the lower is the immediate reward. In addition, the neighboring states near the blocks yielding negative reward also have a lower reward in the extracted reward heatmap compared to the neighboring states near the state yielding the highest reward. In other words, the extracted rewards are more "continuous" and "spread out" across the state space compared to the actual reward function. This is because the rewards have been extracted from the expert policy and not the ground truth rewards. The expert policy, in turn, is a reflection of the optimal values, which itself is not discrete in nature but changes more gradually compared to the reward function. The use of policy for reward extraction is also the reason for the difference in scales between the extracted reward function and expert reward function.

Learning from policy provides the agent with better information about the state-space in terms of the reward. For example, the closer the agent gets to state 99, the higher will be the rewards it observes. This gradual increase or decrease in reward when approaching "key states" (states with very high or very low reward) helps the agent formulate better policies over rarer rewards, as the agent has greater foresight about the state-space from gradually changing immediate rewards at each state. Such feedback provides the agent with heuristics about which direction to travel to and is specially useful when the state-space is only partially observable and decisions must be made locally based on the immediate rewards.

Question 14:

Figure 13 shows the heatmap of optimal state-values from the extracted reward function with $\lambda^{(1)}_{\max}$ using value iteration. We provide analysis of the heatmap in Question 15.

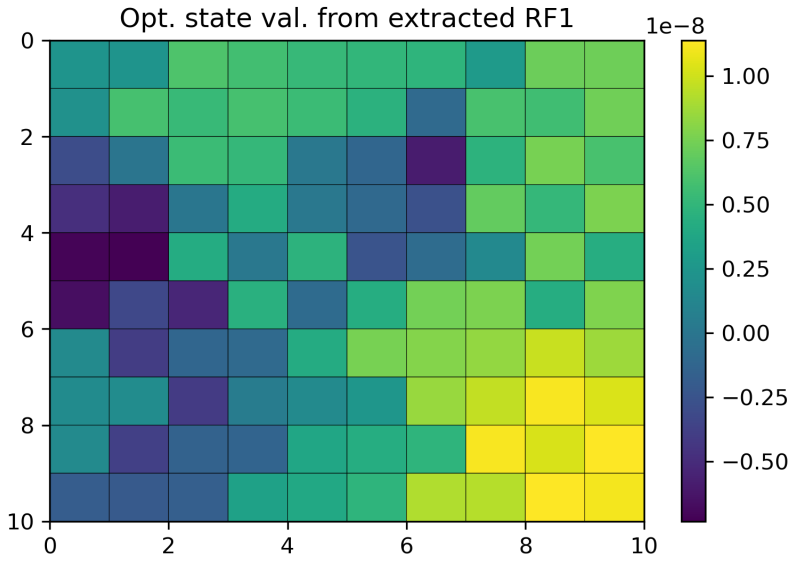


Figure 13: Heatmap visualization for optimal state values for extracted reward function with $\lambda^{(1)}_{\max}$.

Question 15:

In this question, we are asked to compare and contrast the heatmaps in Question 3 and Question 14.

Similarities:

- In both cases, the region near state 99 has the highest optimal value. This is because state 99 provides the highest reward in the state space among all states. As one moves away from state 99, the values gradually decrease in both plots. In addition, we can see low optimal values for those states (and neighboring states) where the reward was negative. The patterns seen in the heatmap of reward function 1 are roughly visible in both the heatmaps in Question 3 and Question 14. There are three blocks of negative rewards in Figure 1 for reward function 1. One can almost make out the same three blocks from the heatmap of optimal values within the same expected regions in the state space for the heatmap in Question 3, and the same regions are also roughly identifiable in Question 14. In addition, the region near the state with the highest reward is also brighter in both the plots.

- As an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase. In other words, there is a visible, progressive and gradual decay of optimal values surrounding the state with the highest reward. This is due to the discount factor in the Bellman equation, which discounts future rewards. This gradual change is visible in both plots.

Differences:

- The scaling is much smaller for the heatmap in Question 14 over Question 3. This is because the heatmap in Question 14 is evaluated from the extracted reward, which in turn is calculated from the expert policy and not expert reward. Thus, the range/scale of optimal state values would not be the same.

- The regions in the heatmap in Question 3 are much more well-defined and homogenous compared to Question 14. This is because the heatmap in Question 3 was extracted from the ground truth reward function, which had a 0 reward in most of the states except three blocks of negative rewards and 1 high-reward state. However, the heatmap in Question 14 was evaluated from the extracted reward, which was shown to be “continuous” and “spread out” across the state space compared to the actual reward function. This is because the rewards have been extracted from the expert policy and not the ground truth rewards. This was evident in the reward vector heatmap plots in Question 13, where we saw an immediate reward being assigned to all of the states. Thus, most of the elements in the extracted reward vectors are not 0 but have some value.

- Comparatively, the agent is less harshly penalized near a region with lower reward in the state-space in Question 14 compared to the ground truth case in Question 3. In addition, there is a higher probability for the agent to not move towards the optimal state (state 99) in case of the state-space in Question 14 over Question 3, as immediate and local rewards can quickly accumulate in the wrong direction and overpower the maximum achievable reward. In other words, the agent will accumulate a lower expected reward in the long run for the state-space in Question 14 over the agent in Question 3. Consequently, the agent is more likely to move off the grid for the state-space in Question 14 over Question 3.

Question 16:

Figure 14 shows the optimal actions taken by the agent in the 2D grid for extracted reward function with $\lambda^{(1)}_{\max}$, shown using arrows. We provide analysis of the actions in Question 17.

```
0 ['→' '↓' '↓' '↓' '→' '↑' '←' '→' '→' '↑']
1 ['→' '→' '→' '←' '←' '↑' '←' '→' '→' '↑']
2 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↑']
3 ['↑' '↑' '→' '↓' '←' '↓' '→' '→' '↑' '←']
4 ['↑' '→' '→' '↑' '←' '↓' '↓' '→' '↑' '↑']
5 ['↓' '↓' '→' '↑' '↓' '↓' '↓' '↓' '↓' '↓']
6 ['↓' '↓' '↓' '→' '→' '→' '→' '↓' '↓' '↓']
7 ['←' '←' '←' '→' '→' '↑' '→' '→' '↓' '↓']
8 ['↑' '↑' '↑' '→' '↓' '→' '→' '→' '→' '↓']
9 ['↓' '←' '→' '→' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 14: Optimal actions undertaken by the agent for extracted reward function with $\lambda^{(1)}_{\max}$.

Question 17:

In this question, we are asked to compare and contrast the action plots in Question 5 and Question 16.

Similarities:

- Majority of the actions in both the policy maps involve moving down and right. This is because the most optimal state (state 99, provides highest reward) is located in the lower right corner in the state-space in both cases. In other words, in both cases, the agent wants to reach the states with the highest rewards.

Differences:

- Some of the actions in Question 16 will lead the agent to be blown off the grid. This is not the case for the agent in Question 5. This is because there is a higher probability for the agent to not move towards the optimal state (state 99) in case of the state-space in Question 14 over Question 3, as immediate and local rewards can quickly accumulate in the wrong direction and overpower the maximum achievable reward. In other words, the agent will accumulate a lower expected reward in the long run for the state-space in Question 14 over the agent in Question 3, ultimately causing the agent to be more likely to move off the grid or take suboptimal actions in the long run for the case in Question 16 compared to Question 5.
- A few actions in Question 16 point to each other. These are local optima in the policy plots, where the agent just oscillates between two states forever (called deadlock condition). This is a result of the gradually changing immediate rewards among neighboring states. If the state value between two neighboring states are very similar and non-zero, then the agent will be forced to go back and forth between the two states if exploration probability is low. Deadlock condition is absent for the policy map in Question 5, as most of the rewards in the state space are zero. This improves the foresight of the agent and passes on the influence of the high reward state to neighboring states much further than possible for the extracted values in Question 16. In other words, the values in the state-space is much less noisy for Question 5 over Question 16, which allows the agent to filter out the correct direction from a further distance.
- For the policy map in Question 5, the agent is guaranteed to reach state 99 from any state. However, this is not the case for the policy plot in Question 16, where the agent has a higher risk of being blown off the grid or getting stuck in local optima / deadlock condition.
- There is more variance and stochasticity in the actions taken in Question 16 compared to Question 5, where the actions are more ordered and homogenous with the goal of projecting a path towards state 99.

Question 18:

In this question, we are asked to repeat Question 11, but for the expert concerning reward function 2. Figure 15 shows the plot of λ versus accuracy.

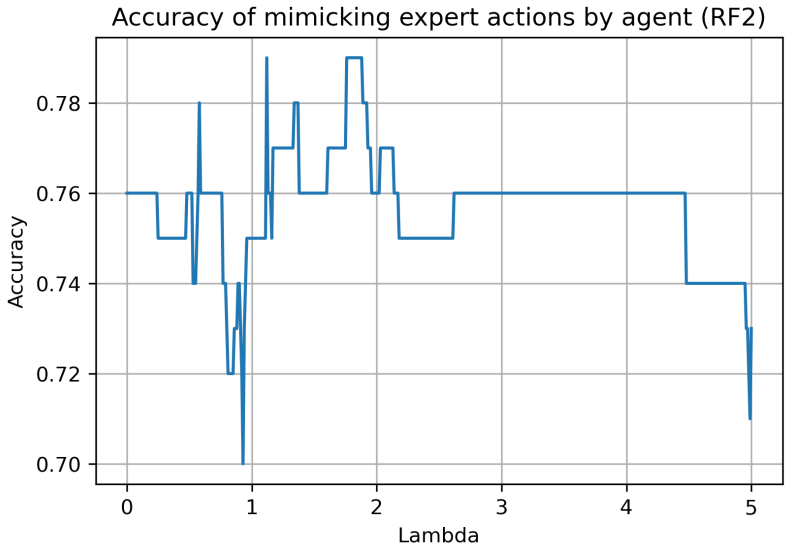


Figure 15: Effect of λ on the accuracy of IRL algorithm (reward function 2).

From Figure 15, we see that the accuracy initially drops with increase in λ until around 1, beyond which accuracy starts to rise until around 2. After λ = 2, the accuracy drops slightly and stays constant (local optimum) until around 4.5, beyond which the accuracy drops. This is expected because λ acts as a regularizer (L1 or Lasso normalization), encouraging simpler reward vectors. Finite values of λ improves robustness, succinctness and transferability of the extracted reward function such that it generalizes well when extracting the optimal policy, which explains the initial rise in accuracy. Large values of λ will yield reward vectors that underfit the tasks and are not expressive enough to derive the optimal global policy, with increased risk of being stuck in a locally optimum policy.

Question 19:

In this question, we are asked to compute the value of λ for which accuracy is maximum from Question 18. We refer to this value as $\lambda_{\max}^{(2)}$. From Figure 15, the maximum accuracy is 79% and **corresponding $\lambda_{\max}^{(2)}$ is 1.12.**

Question 20:

In this question, we are asked to generate heat maps for extracted reward with $\lambda_{\max}^{(2)}$ as well as the ground truth rewards for reward function 2. Figure 16 shows the two plots side by side.

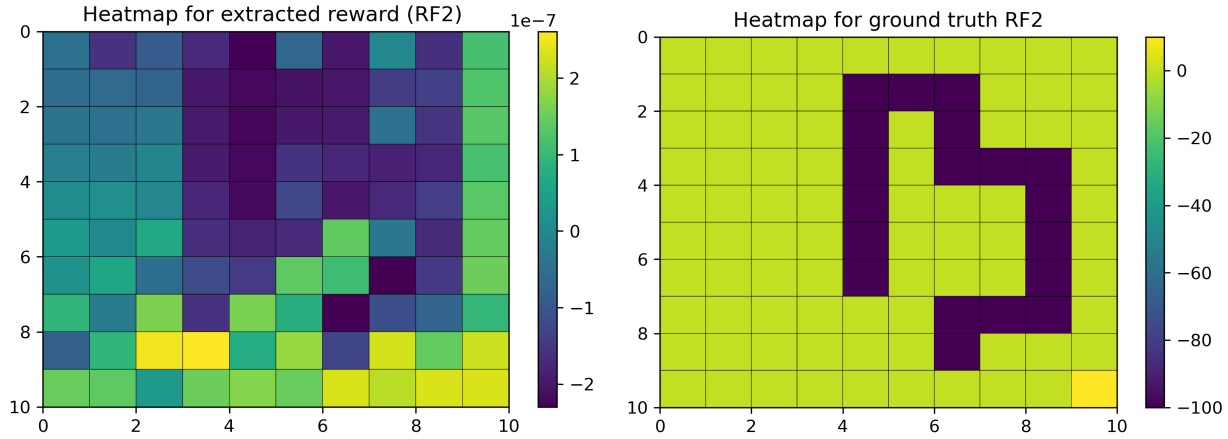


Figure 15: (Left) Heatmap for extracted reward with $\lambda_{\max}^{(2)}$, (Right) Ground truth reward (reward function 2).

From Figure 15, we observe that the extracted rewards by the IRL algorithm are more "continuous" and "spread out" across the state space compared to the actual reward function. This is because the rewards have been extracted from the expert policy and not the ground truth rewards. The expert policy, in turn, is a reflection of the optimal values, which itself is not discrete in nature but changes more gradually compared to the reward function. The use of policy for reward extraction is also the reason for the difference in scales between the extracted reward function and expert reward function.

In addition, we can observe that the IRL algorithm has correctly identified the regions around state 99 as having high reward, while the regions surrounding the negative-reward snake-like chain have been assigned low immediate rewards. For the extracted reward, as one moves further from state 99, which is the state providing the highest reward in ground truth reward function, the lower is the immediate reward. In addition, the neighboring states near the chain yielding negative reward also have a lower reward in the extracted reward heatmap compared to the neighboring states near the state yielding the highest reward. However, one key noticeable and counterintuitive difference is the fact the IRL algorithm identifies another region in the lower left corner (around state 28 and 38) as having the highest reward over state 99. This is because the reward function solely is not a good indicator of expected return in the long run, but should be used in conjunction with transition matrix and optimal state values. Since IRL learns the reward function from expert policy, which in turn is a reflection of optimal expert state values, IRL is better equipped with heuristics about the state-space over vanilla value iteration. Learning from policy provides the agent with better information about the state-space in terms of the reward. For example, the closer the agent gets to state 99, the higher will be the rewards it observes. This gradual increase or decrease in reward when approaching "key states" (states with very

high or very low reward) helps the agent formulate better policies over rarer rewards, as the agent has greater foresight about the state-space from gradually changing immediate rewards at each state. Such feedback provides the agent with heuristics about which direction to travel to and is specially useful when the state-space is only partially observable and decisions must be made locally based on the immediate rewards.

Question 21:

Figure 16 shows the heatmap of optimal state-values from the extracted reward function with $\lambda^{(2)}_{\max}$ using value iteration. We provide analysis of the heatmap in Question 22.

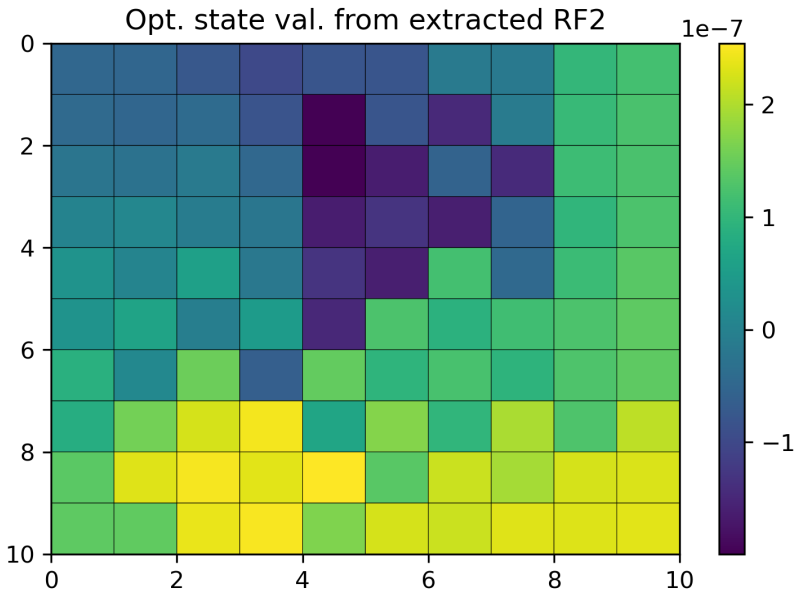


Figure 16: Heatmap visualization for optimal state values for extracted reward function with $\lambda^{(2)}_{\max}$.

Question 22:

In this question, we are asked to compare and contrast the heatmaps in Question 7 and Question 21.

Similarities:

- The region with negative rewards (lowest optimal values) are roughly similar and in the same position in both heatmaps
  .
- The region near state 99 has high (but not highest in one of the plots) optimal values in both the plots. This is because state 99 provides the highest reward among all the states. As one moves away from state 99, the values gradually decrease in both plots.

- As an agent moves towards states with low reward, the optimal values of the neighboring states will progressively decrease, whereas if an agent moves towards desirable states with higher reward, the optimal values of the states it faces will progressively increase. In other words, there is a visible, progressive and gradual decay of optimal values surrounding the state with the highest reward. This is due to the discount factor in the Bellman equation, which discounts future rewards. This gradual change is visible in both plots, albeit more pronounced in the plot in Question 21.

Differences:

- The region near state 99 has high optimal values, but is not the highest in case of the heatmap in Question 21. The highest optimal values are now obtained in the lower-left corner of the state-space. This was evident from the reward function heatmap in Question 20, which showed high reward in that region. This is because the reward function solely is not a good indicator of expected return in the long run, but should be used in conjunction with transition matrix and optimal state values. Since IRL learns the reward function from expert policy, which in turn is a reflection of optimal expert state values, IRL is better equipped with heuristics about the state-space over vanilla value iteration. Learning from policy provides the agent with better information about the state-space in terms of the reward. For example, the closer the agent gets to state 99, the higher will be the rewards it observes. This gradual increase or decrease in reward when approaching "key states" (states with very high or very low reward) helps the agent formulate better policies over rarer rewards, as the agent has greater foresight about the state-space from gradually changing immediate rewards at each state. Such feedback provides the agent with heuristics about which direction to travel to and is specially useful when the state-space is only partially observable and decisions must be made locally based on the immediate rewards.

- The scaling is much smaller for the heatmap in Question 21 over Question 7. This is because the heatmap in Question 21 is evaluated from the extracted reward, which in turn is calculated from the

expert policy and not expert reward. Thus, the range/scale of optimal state values would not be the same.

- The regions in the heatmap in Question 7 are much more well-defined and homogenous compared to Question 21. This is because the heatmap in Question 7 was extracted from the ground truth reward function, which had a 0 reward in most of the states except the snake-like chain of negative reward states and 1 high-reward state. However, the heatmap in Question 21 was evaluated from the extracted reward, which was shown to be "continuous" and "spread out" across the state space compared to the actual reward function. This is because the rewards have been extracted from the expert policy and not the ground truth rewards. This was evident in the reward vector heatmap plots in Question 20, where we saw an immediate reward being assigned to all of the states. Thus, most of the elements in the extracted reward vectors are not 0 but have some value.

- Comparatively, the agent is less harshly penalized near a region with lower reward in the state-space in Question 21 compared to the ground truth case in Question 7. There is a higher probability for the agent to not move towards the optimal regions in case of the state-space in Question 21 over Question 7, as immediate and local rewards can quickly accumulate in the wrong direction and overpower the maximum achievable reward. In other words, the agent will accumulate a lower expected reward in the long run for the state-space in Question 21 over the agent in Question 7. Consequently, the agent is more likely to move off the grid for the state-space in Question 21 over Question 7.

Question 23:

Figure 17 shows the optimal actions taken by the agent in the 2D grid for extracted reward function with $\lambda^{(2)}_{\max}$, shown using arrows. We provide analysis of the actions in Question 24.

```
0 ['↑' '←' '↓' '←' '→' '↑' '→' '↑' '→' '↓']
1 ['↓' '↓' '↓' '←' '←' '↑' '→' '↑' '→' '↓']
2 ['↓' '↓' '↓' '←' '←' '→' '→' '→' '→' '→']
3 ['↓' '↓' '↓' '←' '←' '↓' '↓' '→' '→' '↓']
4 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↓' '→' '↓']
5 ['↓' '↓' '↓' '←' '↓' '↓' '↓' '←' '→' '↓']
6 ['↓' '↓' '↓' '←' '↓' '↓' '←' '←' '→' '↓']
7 ['←' '→' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
8 ['→' '→' '→' '←' '←' '↓' '↓' '↓' '↓' '↓']
9 ['↓' '↑' '↑' '↑' '←' '→' '↓' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 17: Optimal actions undertaken by the agent for extracted reward function with $\lambda^{(2)}_{\max}$.

Question 24:

In this question, we are asked to compare and contrast the action plots in Question 8 and Question 23.

Similarities:

- In both cases, the agent tries to move away from regions of low-reward / low optimal value and move towards regions of high-reward / high optimal value.

- In both cases, we see some arrows pointing away from each other (divergence). This might be due to incompatible states.

- The actions are evenly distributed in all four directions (up, down, left and right) for both cases. In other words, the agent moves in all directions to reach optimal states.

Differences:

- There are two high-reward regions in Question 23 compared to Question 8, evident from the arrows converging to two regions compared to 1 in Question 8. This is because the IRL algorithm identifies another region in the lower left corner (around state 28 and 38) as having the highest reward over state 99, evident from Question 20 as well as Question 22. The reward function alone is not a good indicator of expected return in the long run, but should be used in conjunction with transition matrix and optimal state values. Since IRL learns the reward function from expert policy, which in turn is a reflection of optimal expert state values, IRL is better equipped with heuristics about the state-space over vanilla value iteration.

- Some of the actions in Question 23 will lead the agent to be blown off the grid. This is not the case for the agent in Question 8. This is because there is a higher probability for the agent to not move towards the optimal regions in case of the state-space in Question 21 over Question 7, as immediate and local rewards can quickly accumulate in the wrong direction and overpower the maximum achievable reward. In other words, the agent will accumulate a lower expected reward in the long run for the state-space in Question 21 over the agent in Question 7, ultimately causing the agent to be more likely to move off the grid or take suboptimal actions in the long run for the case in Question 23 compared to Question 8.

- A few actions in Question 23 point to each other. These are local optima in the policy plots, where the agent just oscillates between two states forever (called deadlock condition). This is a result of the gradually changing immediate rewards among neighboring states. If the state value between two neighboring states are very similar and non-zero, then the agent will be forced to go back and forth

between the two states if exploration probability is low. Deadlock condition is absent for the policy map in Question 8, as most of the rewards in the state space are zero. This improves the foresight of the agent and passes on the influence of the high reward state to neighboring states much further than possible for the extracted values in Question 23. In other words, the values in the state-space is much less noisy for Question 8 over Question 23, which allows the agent to filter out the correct direction from a further distance.

- For the policy map in Question 8, the agent is guaranteed to reach the optimal state from any state. However, this is not the case for the policy plot in Question 23, where the agent has a higher risk of being blown off the grid or getting stuck in local optima / deadlock condition.

- There is more variance and stochasticity in the actions taken in Question 23 compared to Question 8, where the actions are more ordered and homogenous with the goal of projecting a path towards state 99.

Question 25:

Figure 18 illustrates two major discrepancies in the action map in Question 23.

Moving off the grid
Deadlock condition / Local optima

```
0 ['↑' '←' '↓' '←' '→' '↑' '→' '↑' '→' '↓']
1 ['↓' '↓' '↓' '←' '←' '↑' '→' '↑' '→' '↓']
2 ['↓' '↓' '↓' '←' '←' '→' '→' '→' '→' '→']
3 ['↓' '↓' '↓' '←' '←' '↓' '↓' '→' '→' '↓']
4 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↓' '→' '↓']
5 ['↓' '↓' '↓' '←' '↓' '↓' '↓' '←' '→' '↓']
6 ['↓' '↓' '↓' '←' '↓' '↓' '←' '←' '→' '↓']
7 ['←' '→' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
8 ['→' '→' '→' '←' '←' '↓' '↓' '↓' '↓' '↓']
9 ['↓' '↑' '↑' '↑' '←' '→' '↓' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 17: Optimal actions undertaken by the agent for extracted reward function with $\lambda_{\max}^{(2)}$, with two major discrepancies highlighted.

The two major discrepancies are as follows:

- *Moving off the grid:* Some of the actions in Figure 17 will lead the agent to be blown off the grid. This is because there is a high probability for the agent to not move towards the optimal regions in case of the state-space in Question 21, as immediate and local rewards can quickly accumulate in the wrong direction and overpower the maximum achievable reward. In other words, the agent will accumulate a lower expected reward in the long run for the state-space in Question 21 over the agent in Question 7, ultimately causing the agent to be more likely to move off the grid or take suboptimal actions in the long run.
- *Deadlock Condition / Local Optima:* A few actions point to each other in Figure 17. These are local optima in the policy plot, where the agent just oscillates between two states forever (called deadlock condition). This is a result of the gradually changing immediate rewards among neighboring states. If the state value between two neighboring states are very similar and non-zero, then the agent will be forced to go back and forth between the two states if exploration probability is low.

To solve the issue with being blown off the grid, we hardcode the values for edge and corner states (except state 99) to $-\infty$. This forces the value iteration algorithm to select a state that falls within the grid rather than a state outside the grid, thereby restricting the possible set of actions in edge and corner states. In addition, we reduced

the value of ϵ from 0.01 to 10^{-10} to provide a stricter convergence criterion, ultimately leading the value iteration algorithm to take more epochs to converge, thereby increasing the probability of finding more optimal policies, thereby improving the accuracy of the IRL algorithm. The resulting optimal policies are shown in Figure 18 for both expert reward functions (reward functions 1 and 2).

```
0 ['→' '↓' '↓' '↓' '←' '←' '→' '→' '→' '↓']
1 ['→' '→' '→' '←' '←' '↑' '→' '→' '→' '↑']
2 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↑']
3 ['↑' '↑' '→' '↓' '↓' '↓' '↓' '→' '↓' '↓']
4 ['↑' '→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
5 ['↓' '→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
6 ['↓' '→' '→' '→' '→' '→' '→' '↓' '↓' '↓']
7 ['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
8 ['↑' '↑' '→' '→' '→' '→' '→' '→' '→' '↓']
9 ['→' '→' '→' '→' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

```
0 ['↓' '↓' '↓' '←' '→' '→' '→' '→' '→' '↓']
1 ['↓' '↓' '↓' '←' '←' '↑' '→' '→' '→' '↓']
2 ['↓' '↓' '↓' '←' '←' '↓' '→' '→' '→' '↓']
3 ['↓' '↓' '↓' '←' '←' '↓' '↓' '→' '→' '↓']
4 ['↓' '↓' '↓' '←' '→' '↓' '↓' '↓' '→' '↓']
5 ['↓' '↓' '↓' '←' '↓' '↓' '↓' '←' '→' '↓']
6 ['↓' '↓' '↓' '←' '↓' '↓' '←' '←' '→' '↓']
7 ['→' '→' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
8 ['→' '→' '→' '←' '←' '↓' '↓' '↓' '↓' '↓']
9 ['→' '↑' '↑' '↑' '→' '→' '→' '→' '→' '↓']
    0   1   2   3   4   5   6   7   8   9
```

Figure 18: (Left): Optimal actions undertaken by the agent for extracted reward function with expert concerning reward function 1 (Right) Optimal actions undertaken by the agent for extracted reward function with expert concerning reward function 2

From Figure 18, it is clear that the first modification has caused the agent to stay within the grid in both cases, thereby solving the issue of being blown off the grid. All of the agent's movements are now within the grid. In addition, more paths now point towards the optimal state (state 99) for both cases.

Figure 19 shows the plot for λ vs. accuracy for both expert reward functions.

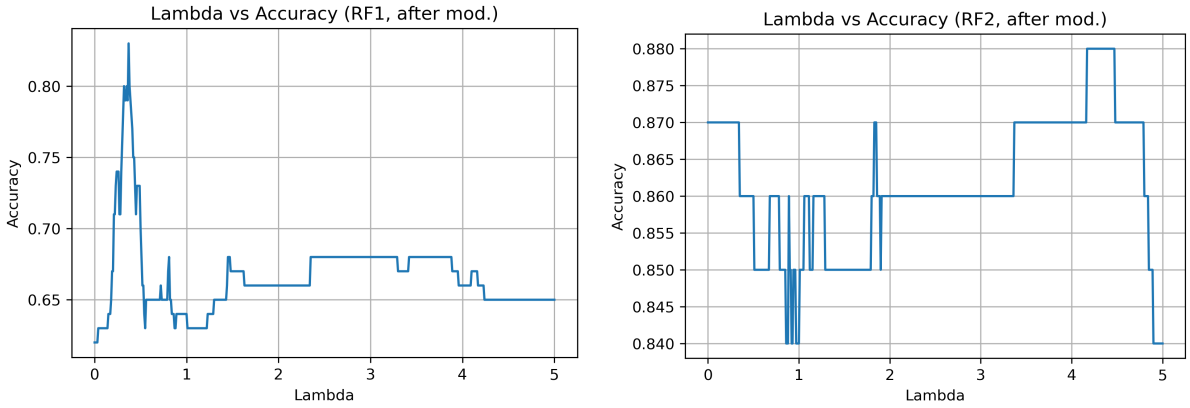


Figure 19: (Left): Effect of λ on the accuracy of IRL algorithm (reward function 1, after modifications) (Right) Effect of λ on the accuracy of IRL algorithm (reward function 2, after modifications)

For reward function 1, the new maximum accuracy was 83%, with new $\lambda_{\max}^{(1)}$ being 0.37, For reward function 2, the new maximum accuracy was 88%, with new $\lambda_{\max}^{(2)}$ being 4.17. For reward function 1, the change in accuracy

is +16% (from 67%), while for reward function 2, the change in accuracy is +9% (from 79%). It is evident that the two modifications have improved the accuracy of the IRL algorithm significantly.

Figure 20 shows the new extracted reward heatmaps for both ground truth reward functions. Figure 21 shows the new optimal values for both ground truth reward functions.

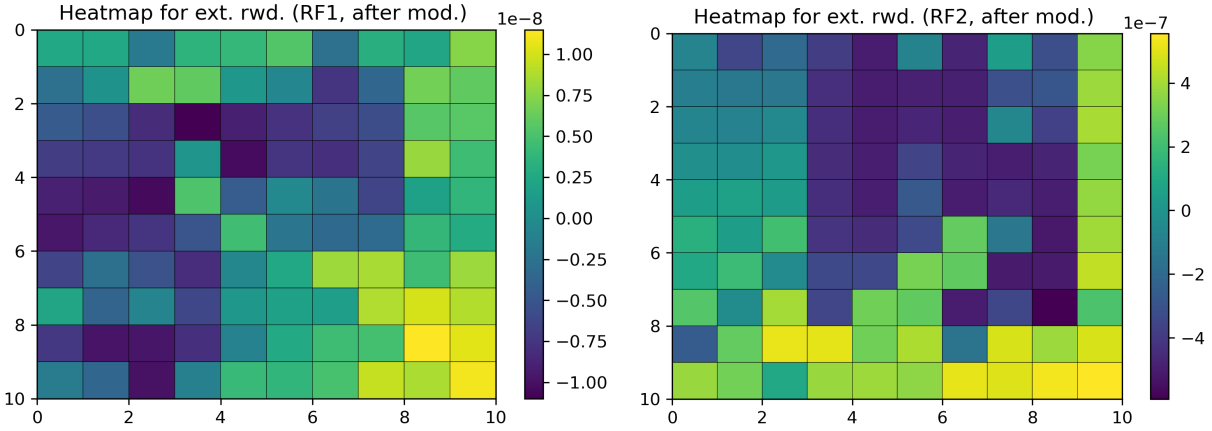


Figure 20: (Left) Heatmap for extracted reward with $\lambda_{\max}^{(1)}$ for modified policy, (Right) Heatmap for extracted reward with $\lambda_{\max}^{(2)}$ for modified policy.

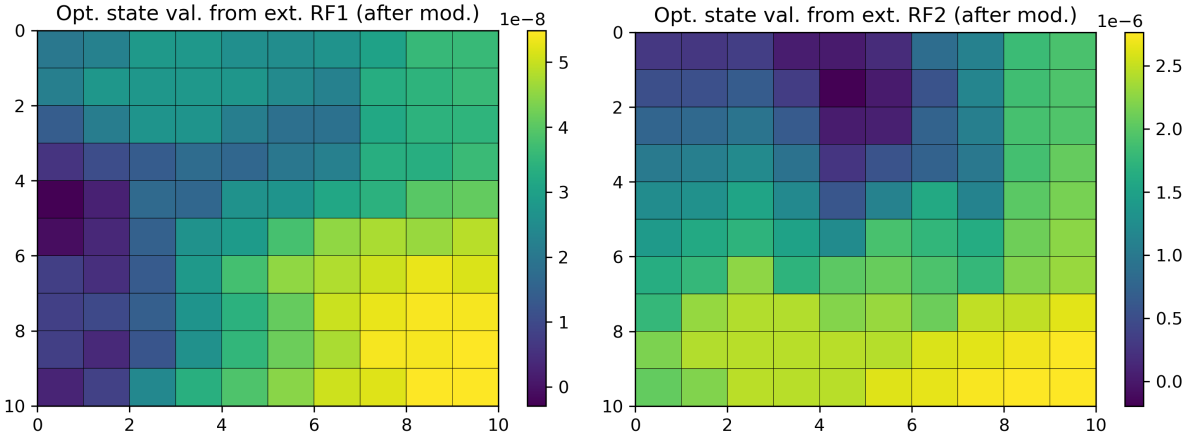


Figure 21: (Left) Heatmap visualization for optimal state values for extracted reward function with $\lambda_{\max}^{(1)}$ for modified policy, (Right) Heatmap visualization for optimal state values for extracted reward function with $\lambda_{\max}^{(2)}$ for modified policy.

From Figures 21 and 22, we can see that the regions surrounding the optimal states are much more distinguishable from the negative reward regions compared to the policy where the agent was moving off the grid and $\epsilon$was high, increasing the probability of the agent reaching optimal state (state 99). Both reward and optimal value heatmaps now more closely resemble the expert's reward function and expert's optimal value heatmaps.

Proj3_Saha_Young

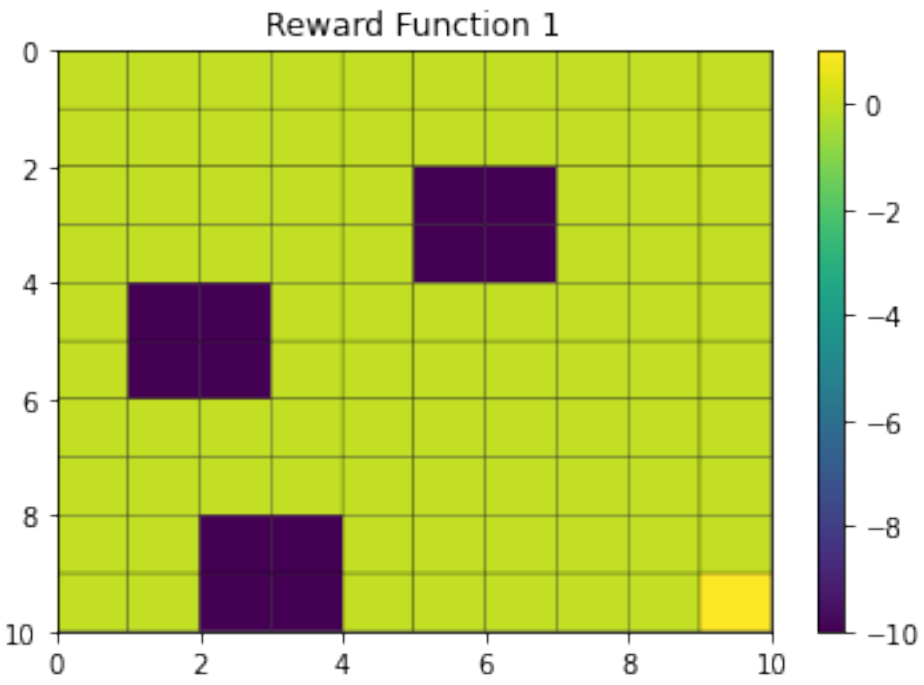
May 22, 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from cvxopt import solvers, matrix
from tqdm import tqdm
```

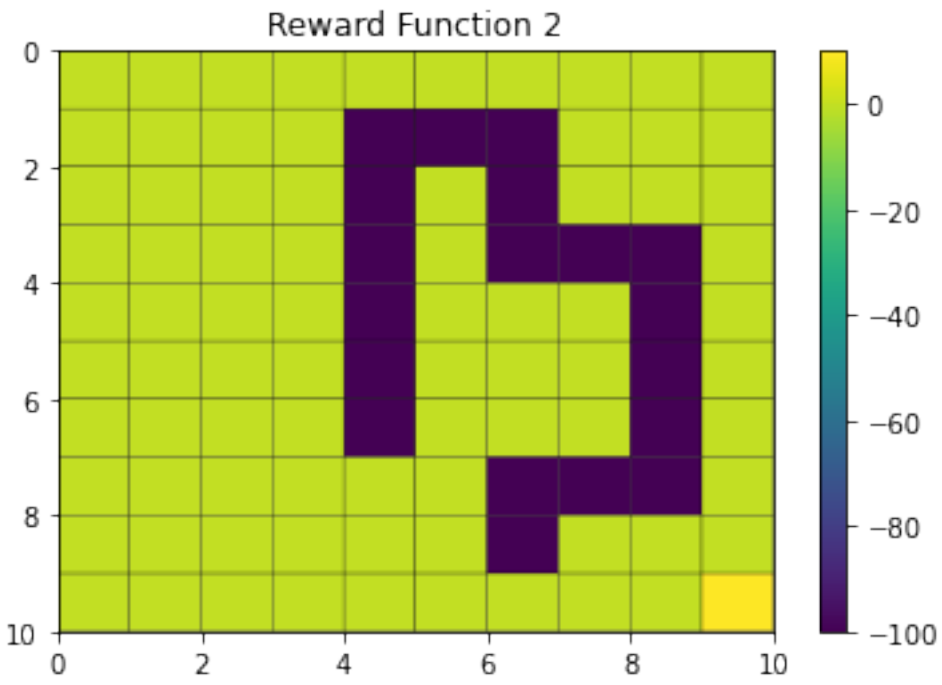
0.1 Question 1

[2]:
```
rwd_1 = np.array([[0,0,0,0,0,0,0,0,0,0],
          [0,0,0,0,0,0,0,0,0,0],
          [0,0,0,0,0,-10,-10,0,0,0],
          [0,0,0,0,0,-10,-10,0,0,0],
          [0,-10,-10,0,0,0,0,0,0,0],
          [0,-10,-10,0,0,0,0,0,0,0],
          [0,0,0,0,0,0,0,0,0,0],
          [0,0,0,0,0,0,0,0,0,0],
          [0,0,-10,-10,0,0,0,0,0,0],
          [0,0,-10,-10,0,0,0,0,0,1]])

rwd_2 = np.array([[0,0,0,0,0,0,0,0,0,0],
          [0,0,0,0,-100,-100,-100,0,0,0],
          [0,0,0,0,-100,0,-100,0,0,0],
          [0,0,0,0,-100,0,-100,-100,-100,0],
          [0,0,0,0,-100,0,0,0,-100,0],
          [0,0,0,0,-100,0,0,0,-100,0],
          [0,0,0,0,-100,0,0,0,-100,0],
          [0,0,0,0,0,0,-100,-100,-100,0],
          [0,0,0,0,0,0,-100,0,0,0],
          [0,0,0,0,0,0,0,0,0,10]])
```

[3]:
```
plt.pcolor(rwd_1,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Reward Function 1")
plt.savefig('Q1a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[4]:
```
plt.pcolor(rwd_2,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Reward Function 2")
plt.savefig('Q1b.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.2 Question 2

[5]:
```
state_num = np.zeros((10,10))
k=0
for i in range(0,10):
    for j in range(0,10):
        state_num[j][i]=k
        k+=1
```

[6]:
```
def tp_calc(w):
    tp_u= np.zeros((100,100)) #Up
    tp_d= np.zeros((100,100)) #Down
    tp_l= np.zeros((100,100)) #Left
    tp_r= np.zeros((100,100)) #Right

    #Up
    for i in range(0,100):
        if(i%10==0):
            tp_u[i][i]+=w/4
        else:
            tp_u[i][i-1]=1-w+w/4
        if(i-10<0):
            tp_u[i][i]+=w/4
        else:
```

```
            tp_u[i][i-10]=w/4
        if(i+10>99):
            tp_u[i][i]+=w/4
        else:
            tp_u[i][i+10]=w/4
        if((i+1)%10==0):
            tp_u[i][i]+=w/4
        else:
            tp_u[i][i+1]=w/4
        if(i%10==0):
            tp_u[i][i]+=1-w

    #Down
    for i in range(0,100):
        if(i%10==0):
            tp_d[i][i]+=w/4
        else:
            tp_d[i][i-1]=w/4
        if(i-10<0):
            tp_d[i][i]+=w/4
        else:
            tp_d[i][i-10]=w/4
        if(i+10>99):
            tp_d[i][i]+=w/4
        else:
            tp_d[i][i+10]=w/4
        if((i+1)%10==0):
            tp_d[i][i]+=w/4
        else:
            tp_d[i][i+1]=1-w+w/4
        if((i+1)%10==0):
            tp_d[i][i]+=1-w

    #Left
    for i in range(0,100):
        if(i%10==0):
            tp_l[i][i]+=w/4
        else:
            tp_l[i][i-1]=w/4
        if(i-10<0):
            tp_l[i][i]+=w/4
        else:
            tp_l[i][i-10]=1-w+w/4
        if(i+10>99):
            tp_l[i][i]+=w/4
        else:
            tp_l[i][i+10]=w/4
```

```
        if((i+1)%10==0):
            tp_l[i][i]+=w/4
        else:
            tp_l[i][i+1]=w/4
        if(i-10<0):
            tp_l[i][i]+=1-w

    #Right
    for i in range(0,100):
        if(i%10==0):
            tp_r[i][i]+=w/4
        else:
            tp_r[i][i-1]=w/4
        if(i-10<0):
            tp_r[i][i]+=w/4
        else:
            tp_r[i][i-10]=w/4
        if(i+10>99):
            tp_r[i][i]+=w/4
        else:
            tp_r[i][i+10]=1-w+w/4
        if((i+1)%10==0):
            tp_r[i][i]+=w/4
        else:
            tp_r[i][i+1]=w/4
        if(i+10>99):
            tp_r[i][i]+=1-w

    return tp_u, tp_d, tp_l, tp_r
```

[7]:
```
def value_iteration(state_num,w,gamma,rwd,epsilon,tp_u, tp_d, tp_l, tp_r):
    state_val=np.zeros(100)
    delta=np.inf
    r = (rwd.T).ravel()
    N = 0
    while(delta>epsilon):
        delta=0
        old_state_val=np.copy(state_val)
        for s in range(0,100):
            u_val=np.sum(tp_u[s]*(r+gamma*old_state_val))
            d_val=np.sum(tp_d[s]*(r+gamma*old_state_val))
            l_val=np.sum(tp_l[s]*(r+gamma*old_state_val))
            r_val=np.sum(tp_r[s]*(r+gamma*old_state_val))
            state_val[s]=max(u_val,d_val,r_val,l_val)
            delta=max(delta,abs(old_state_val[s]-state_val[s]))
        N = N+1
    state_val = np.transpose(state_val.reshape(10,10))
```

```
    return state_val, N
```

[8]:
```
w = 0.1
gamma = 0.8
epsilon = 0.01
tp_u, tp_d, tp_l, tp_r = tp_calc(w)
V_opt, N=value_iteration(state_num,w,gamma,rwd_1,epsilon,tp_r,tp_l,tp_u,tp_d)
print("Number of steps needed to converge:", N)
```

```
Number of steps needed to converge: 22
```

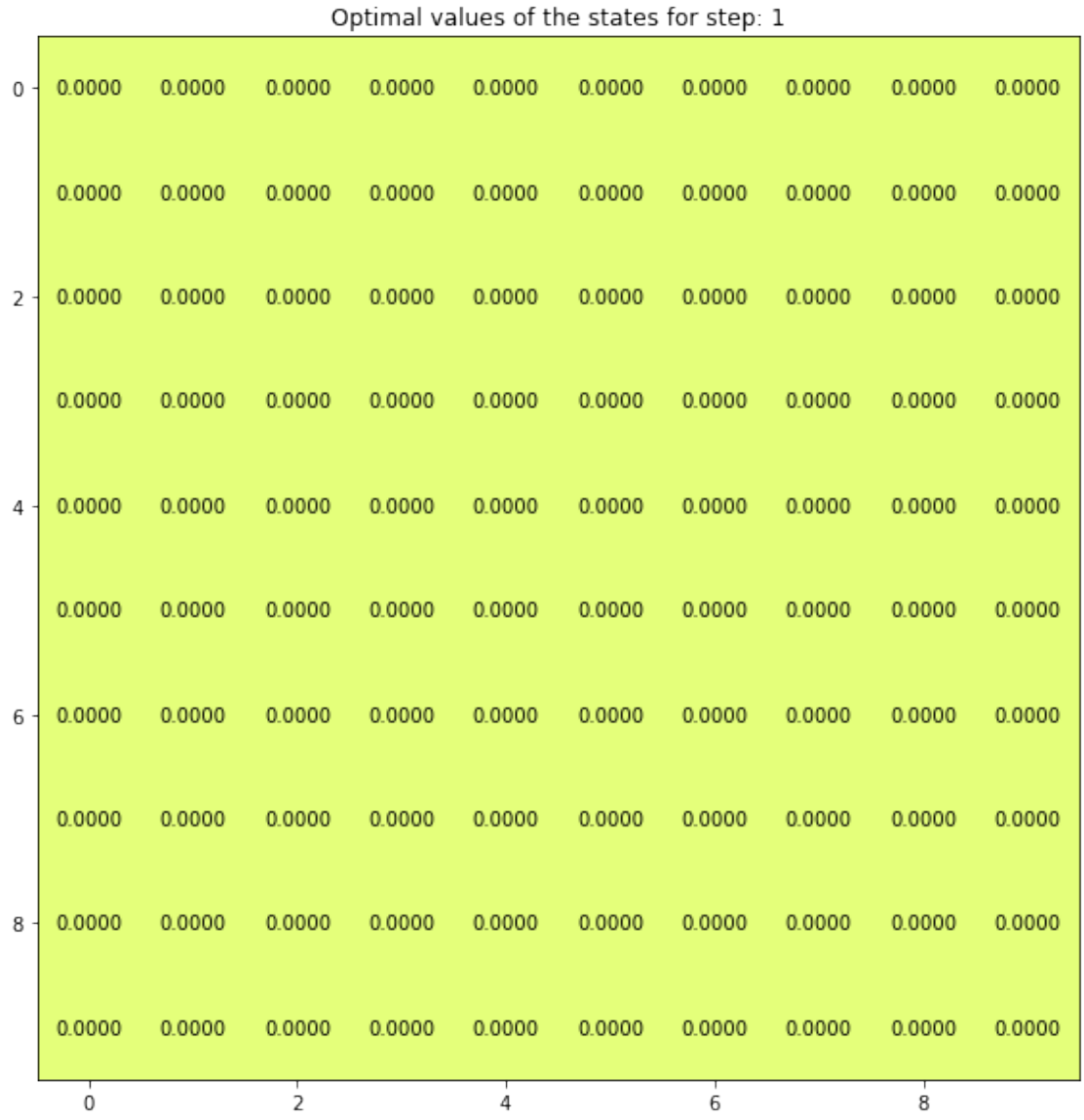
[9]:
```
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(V_opt,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(V_opt[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Optimal values of the states (Reward Function 1)")
plt.savefig('Q2a.png',dpi=300,bbox_inches='tight')
plt.show()
```

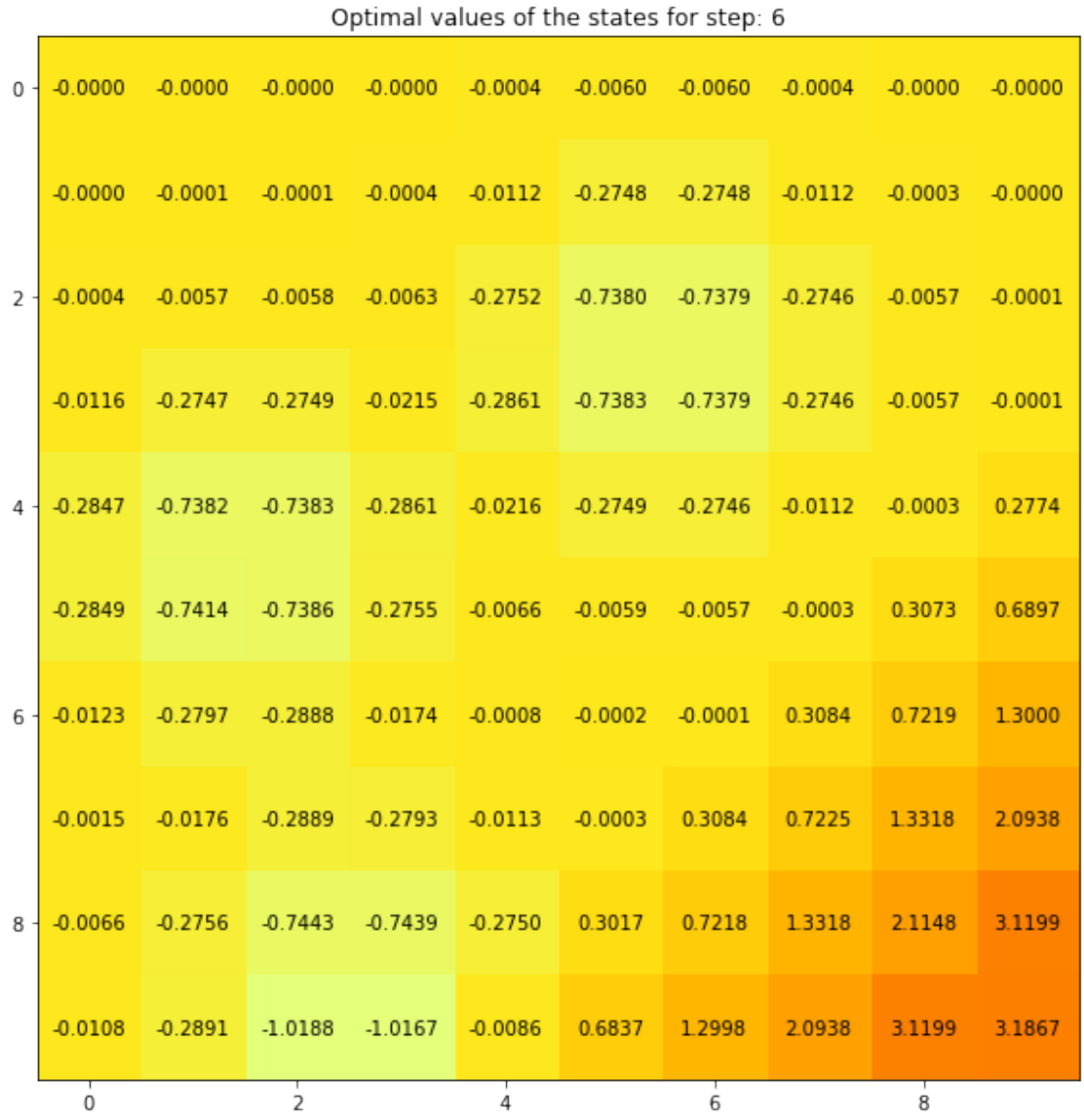
[10]:
```
def value_iteration_with_snapshots(state_num,w,gamma,rwd,epsilon,tp_u, tp_d,␣
↪tp_l, tp_r,div):
    state_val=np.zeros(100)
    snapshot = []
    delta=np.inf
    r = (rwd.T).ravel()
    N = 0
    while(delta>epsilon):
        if((N==0) or (N==5) or (N==10) or (N==15) or (N==20)):
            cur_state_val = np.copy(state_val)
            cur_state_val = np.transpose(cur_state_val.reshape(10,10))
            snapshot.append(cur_state_val)
```

```
        delta=0
        old_state_val=np.copy(state_val)
        for s in range(0,100):
            u_val=np.sum(tp_u[s]*(r+gamma*old_state_val))
            d_val=np.sum(tp_d[s]*(r+gamma*old_state_val))
            l_val=np.sum(tp_l[s]*(r+gamma*old_state_val))
            r_val=np.sum(tp_r[s]*(r+gamma*old_state_val))
            state_val[s]=max(u_val,d_val,r_val,l_val)
            delta=max(delta,abs(old_state_val[s]-state_val[s]))
        N = N+1
    return snapshot
```

[11]:
```
snapshots =␣
↪value_iteration_with_snapshots(state_num,w,gamma,rwd_1,epsilon,tp_r,tp_l,tp_u,tp_d,4)
```

[12]:
```
N = 1
for k in range(len(snapshots)):
    V = snapshots[k]
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.imshow(V,cmap="Wistia")
    for i in range(10):
        for j in range(10):
            c = "{:.4f}".format(V[j,i])
            ax.text(i, j, str(c), va='center', ha='center')
    plt.title("Optimal values of the states for step: "+str(N))
    plt.savefig('Q2b'+str(N)+'.png',dpi=300,bbox_inches='tight')
    N = N+5
    plt.show()
```


Optimal values of the states for step: 1
0
2
4
6
8
0.0000
0
2
4
6
8

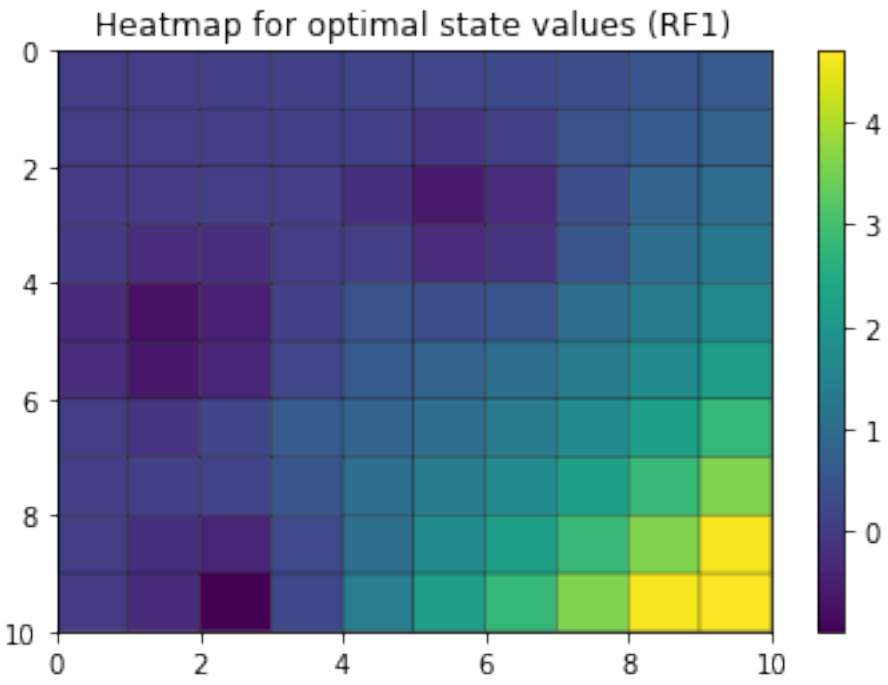

Optimal values of the states for step: 6
-0.0000 -0.0000 -0.0000 -0.0000 -0.0004 -0.0060 -0.0060 -0.0004 -0.0000 -0.0000
-0.0000 -0.0001 -0.0001 -0.0004 -0.0112 -0.2748 -0.2748 -0.0112 -0.0003 -0.0000
-0.0004 -0.0057 -0.0058 -0.0063 -0.2752 -0.7380 -0.7379 -0.2746 -0.0057 -0.0001
-0.0116 -0.2747 -0.2749 -0.0215 -0.2861 -0.7383 -0.7379 -0.2746 -0.0057 -0.0001
-0.2847 -0.7382 -0.7383 -0.2861 -0.0216 -0.2749 -0.2746 -0.0112 -0.0003 0.2774
-0.2849 -0.7414 -0.7386 -0.2755 -0.0066 -0.0059 -0.0057 -0.0003 0.3073 0.6897
-0.0123 -0.2797 -0.2888 -0.0174 -0.0008 -0.0002 -0.0001 0.3084 0.7219 1.3000
-0.0015 -0.0176 -0.2889 -0.2793 -0.0113 -0.0003 0.3084 0.7225 1.3318 2.0938
-0.0066 -0.2756 -0.7443 -0.7439 -0.2750 0.3017 0.7218 1.3318 2.1148 3.1199
-0.0108 -0.2891 -1.0188 -1.0167 -0.0086 0.6837 1.2998 2.0938 3.1199 3.1867
0
2
4
6
8


Optimal values of the states for step: 11
0
2
4
6
8
-0.0000 -0.0000 -0.0000 -0.0000 -0.0004 -0.0060 -0.0060 -0.0004 0.0709 0.1629
-0.0000 -0.0001 -0.0001 -0.0004 -0.0113 -0.2749 -0.2749 0.0603 0.1738 0.3272
-0.0004 -0.0057 -0.0058 -0.0063 -0.2754 -0.7385 -0.6634 -0.0960 0.3459 0.5493
-0.0117 -0.2747 -0.2750 -0.0217 -0.2867 -0.6607 -0.5552 0.0817 0.5762 0.8436
-0.2848 -0.7386 -0.7388 -0.2868 0.0562 -0.0919 0.0835 0.5729 0.8798 1.2226
-0.2861 -0.7430 -0.7395 -0.1977 0.1765 0.3523 0.5785 0.8815 1.2606 1.7095
-0.0135 -0.2809 -0.2130 0.1656 0.3574 0.5840 0.8818 1.2619 1.7468 2.3340
-0.0027 -0.0187 -0.2852 0.0767 0.5728 0.8815 1.2619 1.7475 2.3665 3.1348
-0.0078 -0.2766 -0.7362 -0.1670 0.6050 1.2549 1.7467 2.3665 3.1560 4.1617
-0.0119 -0.2902 -1.0058 -0.1822 0.9364 1.7035 2.3339 3.1348 4.1617 4.2287
0
2
4
6
8


Optimal values of the states for step: 16
0
2
4
6
8
-0.0000 -0.0000 -0.0000 0.0109 0.0360 0.0824 0.1528 0.2444 0.3540 0.4783
-0.0000 -0.0001 -0.0001 -0.0004 0.0004 -0.2304 -0.0387 0.3411 0.4940 0.6557
-0.0004 -0.0057 -0.0058 -0.0061 -0.2710 -0.7091 -0.3862 0.2244 0.6758 0.8869
-0.0117 -0.2747 -0.2726 -0.0133 -0.0438 -0.3826 -0.2341 0.4118 0.9149 1.1836
-0.2848 -0.7362 -0.5907 -0.0416 0.3382 0.2293 0.4137 0.9116 1.2199 1.5637
-0.2863 -0.7352 -0.4943 0.0843 0.4977 0.6825 0.9173 1.2216 1.6018 2.0509
-0.0137 -0.2482 0.0627 0.4867 0.6875 0.9227 1.2219 1.6031 2.0882 2.6754
-0.0029 -0.0138 0.0071 0.4044 0.9115 1.2216 1.6031 2.0888 2.7079 3.4763
-0.0080 -0.2738 -0.5474 0.1662 0.9449 1.5961 2.0880 2.7079 3.4975 4.5032
-0.0123 -0.2903 -0.9943 0.1492 1.2773 2.0448 2.6753 3.4763 4.5032 4.5702
0
2
4
6
8

0.3 Question 3

[13]:
```
plt.pcolor(V_opt,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for optimal state values (RF1)")
plt.savefig('Q3a.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.4 Question 5

[14]:
```
def policy_iteration(state_num,w,gamma,rwd,epsilon,tp_u, tp_d, tp_l, tp_r):
    state_val=np.zeros(100)
    delta=np.inf
    policy=np.zeros(100)
    arrows=np.zeros(100)
    r = (rwd.T).ravel()
    while(delta>epsilon):
        delta=0
        old_state_val=np.copy(state_val)
        for s in range(0,100):
            u_val=np.sum(tp_u[s]*(r+gamma*old_state_val))
            d_val=np.sum(tp_d[s]*(r+gamma*old_state_val))
            l_val=np.sum(tp_l[s]*(r+gamma*old_state_val))
            r_val=np.sum(tp_r[s]*(r+gamma*old_state_val))
            state_val[s]=max(u_val,d_val,r_val,l_val)
            delta=max(delta,abs(old_state_val[s]-state_val[s]))
    for s in range(0,100):
        u_val=np.sum(tp_u[s]*(r+gamma*state_val))
        d_val=np.sum(tp_d[s]*(r+gamma*state_val))
        l_val=np.sum(tp_l[s]*(r+gamma*state_val))
        r_val=np.sum(tp_r[s]*(r+gamma*state_val))
        arr=[u_val,d_val,l_val,r_val] #Up: 0, Down: 1, Left: 2, Right: 3
```

```
        policy[s]=np.amax(arr)
        arrows[s]=arr.index(np.amax(arr))
    arrows = np.transpose(arrows.reshape(10,10))
    policy = np.transpose(policy.reshape(10,10))
    pic_arrow=np.chararray((10,10),unicode=True)
    for i in range(10):
        for j in range(10):
            if(arrows[j][i]==0.):
                pic_arrow[j][i] = u'\u2191'
            elif(arrows[j][i]==1.):
                pic_arrow[j][i] = u'\u2193'
            elif(arrows[j][i]==2.):
                pic_arrow[j][i]=u'\u2190'
            elif(arrows[j][i]==3.):
                pic_arrow[j][i] = u'\u2192'
    return policy, pic_arrow, arrows
```

[15]:
```
policy, pic_arrow, arrows =␣
↪policy_iteration(state_num,w,gamma,rwd_1,epsilon,tp_u, tp_d, tp_l, tp_r)
```

[16]:
```
print(pic_arrow)
```

```
[['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
 ['→' '→' '→' '↑' '↑' '↑' '→' '→' '↓' '↓']
 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↓']
 ['↑' '↑' '→' '↓' '↓' '↓' '↓' '→' '↓' '↓']
 ['↑' '↑' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '↓' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '→' '→' '→' '→' '→' '→' '↓' '↓' '↓']
 ['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
 ['↑' '↑' '↑' '→' '→' '→' '→' '→' '→' '↓']
 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']]
```

[17]:
```
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(policy,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(policy[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Policy values (Reward Function 1)")
plt.savefig('Q5.png',dpi=300,bbox_inches='tight')
plt.show()
```

0.5 Question 6

[18]:
```
V_opt, N=value_iteration(state_num,w,gamma,rwd_2,epsilon,tp_r,tp_l,tp_u,tp_d)
print("Number of steps needed to converge:", N)
```

```
Number of steps needed to converge: 32
```

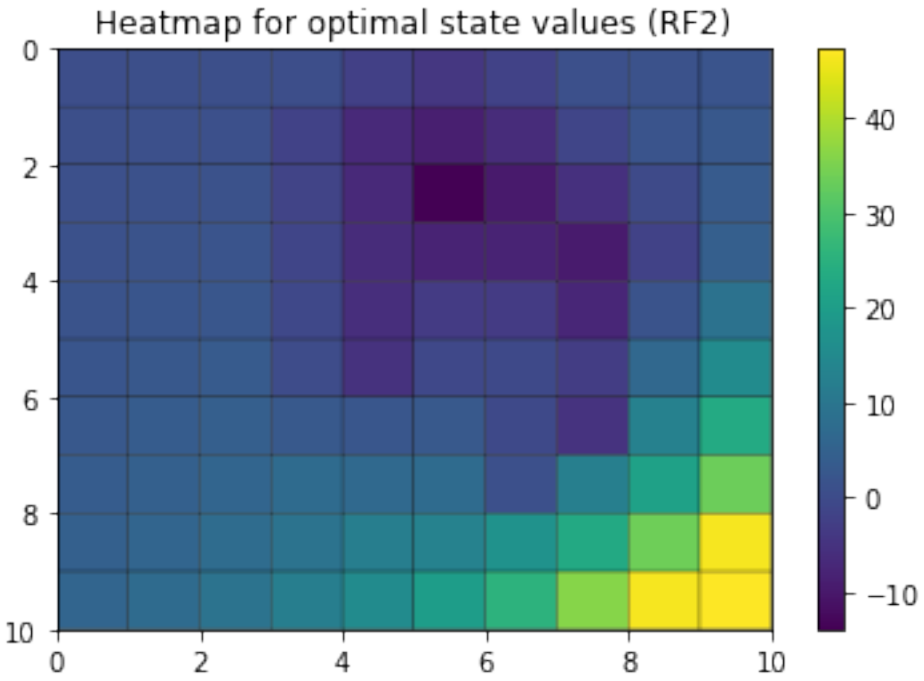
[19]:
```
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(V_opt,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(V_opt[j,i])
```

```
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Optimal values of the states (Reward Function 2)")
plt.savefig('Q6.png',dpi=300,bbox_inches='tight')
plt.show()
```

0.6 Question 7

[20]:
```
plt.pcolor(V_opt,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for optimal state values (RF2)")
```

```
plt.savefig('Q7.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.7 Question 8

[21]:
```
policy, pic_arrow, arrows =␣
↪policy_iteration(state_num,w,gamma,rwd_2,epsilon,tp_u, tp_d, tp_l, tp_r)
```

[22]:
```
print(pic_arrow)
```

```
[['↓' '↓' '↓' '←' '←' '→' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↑' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↑' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↓' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '←' '→' '↓']
 ['↓' '↓' '↓' '↓' '↓' '↓' '←' '←' '→' '↓']
 ['↓' '↓' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
 ['→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓' '↓']
 ['→' '→' '→' '→' '→' '→' '→' '→' '→' '↓']]
```

[23]:
```
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(policy,cmap="Wistia")
for i in range(10):
```

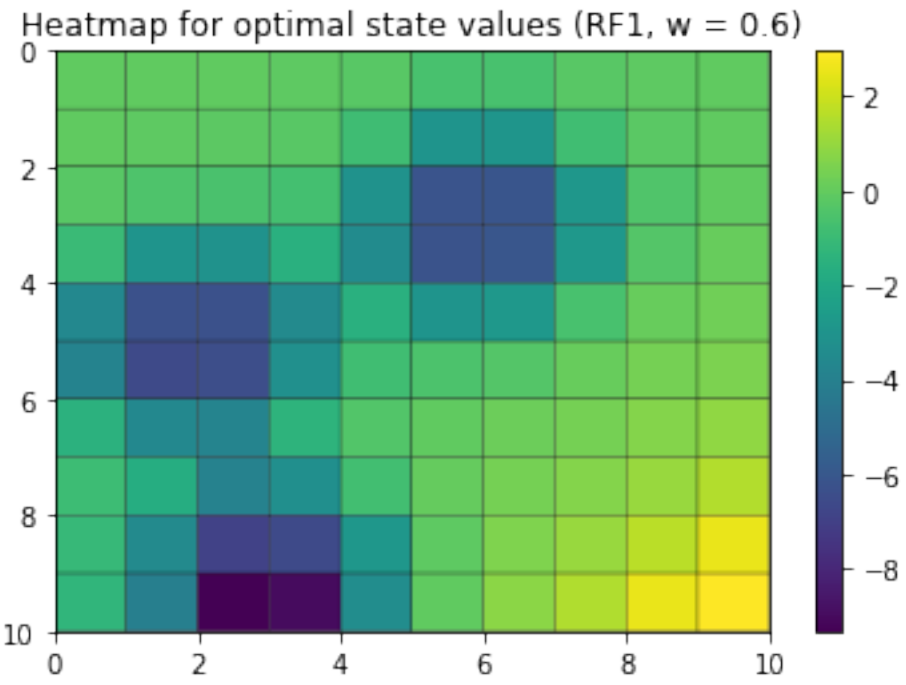
```
    for j in range(10):
        c = "{:.4f}".format(policy[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Policy values (Reward Function 2)")
plt.savefig('Q8.png',dpi=300,bbox_inches='tight')
plt.show()
```

0.8 Question 9

[24]:
```
w = 0.6
gamma = 0.8
epsilon = 0.01
tp_u, tp_d, tp_l, tp_r = tp_calc(w)
V_opt, N=value_iteration(state_num,w,gamma,rwd_1,epsilon,tp_r,tp_l,tp_u,tp_d)
print("Number of steps needed to converge:", N)

fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(V_opt,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(V_opt[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Optimal values of the states (RF1, w = 0.6)")
plt.savefig('Q9a.png',dpi=300,bbox_inches='tight')
plt.show()

plt.pcolor(V_opt,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for optimal state values (RF1, w = 0.6)")
plt.savefig('Q9b.png',dpi=300,bbox_inches='tight')
plt.show()

policy, pic_arrow,arrows =␣
 ↪policy_iteration(state_num,w,gamma,rwd_1,epsilon,tp_u, tp_d, tp_l, tp_r)
print(pic_arrow)

fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(policy,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(policy[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Policy values (RF1, w = 0.6)")
plt.savefig('Q9c.png',dpi=300,bbox_inches='tight')
plt.show()
```

```
Number of steps needed to converge: 19
```


Optimal values of the states (RF1, w = 0.6)
0
2
4
6
8
-0.0222 -0.0294 -0.0436 -0.0877 -0.2433 -0.6060 -0.5962 -0.2253 -0.0701 -0.0311
-0.0549 -0.0933 -0.1271 -0.2437 -0.8600 -2.9596 -2.9416 -0.7986 -0.1740 -0.0418
-0.2310 -0.4905 -0.5660 -0.7259 -3.0754 -6.1430 -6.0681 -2.8446 -0.4160 -0.0225
-1.0056 -2.9525 -3.0381 -1.5186 -3.4663 -6.1956 -6.0396 -2.7942 -0.3410 0.0888
-3.5801 -6.2670 -6.2759 -3.4916 -1.5457 -2.9898 -2.7972 -0.6180 0.0797 0.2884
-3.8109 -6.5523 -6.4046 -3.2066 -0.7947 -0.5044 -0.3069 0.1101 0.3769 0.5444
-1.4575 -3.5842 -3.7835 -1.3929 -0.3655 -0.0201 0.1844 0.4131 0.6706 0.9222
-0.9081 -1.7176 -3.9276 -3.2549 -0.7473 0.0680 0.3993 0.6900 1.0724 1.5286
-1.1009 -3.5061 -6.9138 -6.5373 -2.8169 -0.0810 0.5972 1.0617 1.6651 2.5292
-1.2763 -4.0863 -9.3172 -8.9536 -3.3480 -0.0643 0.8210 1.5122 2.5270 2.9536


Heatmap for optimal state values (RF1, w = 0.6)
0
2
4
6
8
10
0
2
4
6
8
10
2
0
−2
−4
−6
−8

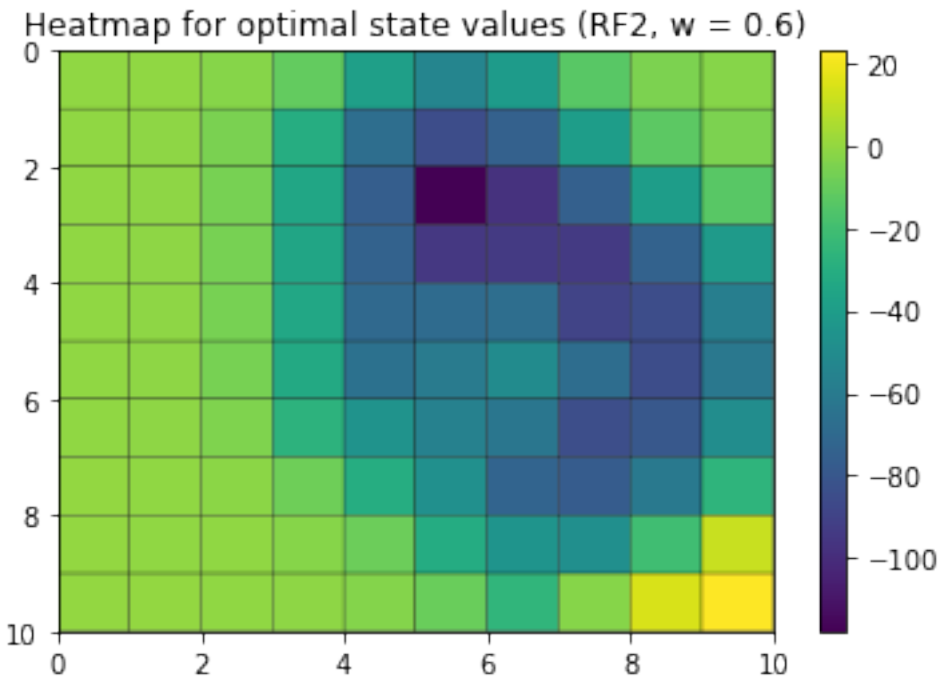
```
[['↑' '←' '←' '←' '←' '←' '→' '→' '→' '↑']
 ['↑' '↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓']
 ['↑' '↑' '↑' '↑' '←' '↑' '→' '→' '→' '↓']
 ['↑' '↑' '↑' '↑' '←' '↓' '→' '→' '→' '↓']
 ['↑' '↑' '↑' '↑' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '↓' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '←' '→' '→' '→' '→' '→' '↓' '↓' '↓']
 ['←' '←' '←' '→' '→' '→' '→' '→' '↓' '↓']
 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']
 ['↑' '←' '←' '→' '→' '→' '→' '→' '→' '↓']]
```

[25]:
```
w = 0.6
gamma = 0.8
epsilon = 0.01
tp_u, tp_d, tp_l, tp_r = tp_calc(w)
V_opt, N=value_iteration(state_num,w,gamma,rwd_2,epsilon,tp_r,tp_l,tp_u,tp_d)
print("Number of steps needed to converge:", N)

fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(V_opt,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(V_opt[j,i])
```

```
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Optimal values of the states (RF2, w = 0.6)")
plt.savefig('Q9d.png',dpi=300,bbox_inches='tight')
plt.show()

plt.pcolor(V_opt,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for optimal state values (RF2, w = 0.6)")
plt.savefig('Q9e.png',dpi=300,bbox_inches='tight')
plt.show()

policy, pic_arrow,arrows =␣
 ↪policy_iteration(state_num,w,gamma,rwd_2,epsilon,tp_u, tp_d, tp_l, tp_r)
print(pic_arrow)

fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(policy,cmap="Wistia")
for i in range(10):
    for j in range(10):
        c = "{:.4f}".format(policy[j,i])
        ax.text(i, j, str(c), va='center', ha='center')
plt.title("Policy values (RF2, w = 0.6)")
plt.savefig('Q9f.png',dpi=300,bbox_inches='tight')
plt.show()
```

```
Number of steps needed to converge: 27
```


Optimal values of the states (RF2, w = 0.6)
0
2
4
6
8
-0.2387 -0.5834 -2.4557 -10.7418 -38.9548 -53.7099 -41.2554 -13.4848 -4.8205 -2.6217
-0.2969 -0.9521 -5.1334 -30.8209 -67.5813 -84.8003 -74.4203 -39.9964 -12.2920 -4.8315
-0.3549 -1.1334 -6.0162 -34.7020 -76.4245 -117.7541 -97.4983 -75.3675 -39.9703 -13.5650
-0.3849 -1.1800 -6.1492 -34.8869 -74.3091 -94.8000 -94.0066 -94.0530 -74.1230 -41.8311
-0.3521 -1.1428 -6.0184 -34.1733 -70.1161 -69.2640 -67.3881 -89.3425 -85.2914 -57.9386
-0.2882 -1.0367 -5.6451 -32.7138 -65.4404 -59.9831 -50.1814 -68.1375 -84.8865 -61.2429
-0.1981 -0.7966 -4.5127 -27.3086 -45.3010 -56.1904 -62.7602 -84.6212 -79.4113 -49.4415
-0.1007 -0.3649 -1.7335 -8.0148 -30.7537 -47.7894 -73.2229 -77.5341 -60.9601 -26.4270
-0.0483 -0.1426 -0.5824 -2.3740 -8.8078 -31.0742 -44.6760 -48.3554 -20.3475 11.3533
-0.0306 -0.0652 -0.2246 -0.8269 -2.8677 -9.1911 -25.8094 -2.8315 14.5709 22.9712
0
2
4
6
8


Heatmap for optimal state values (RF2, w = 0.6)
0
2
4
6
8
10
0
2
4
6
8
10
20
0
−20
−40
−60
−80
−100

```
[['↑' '←' '←' '←' '←' '←' '→' '→' '→' '↑']
 ['↑' '←' '←' '←' '←' '↑' '→' '→' '↑' '↑']
 ['↑' '←' '←' '←' '←' '↓' '→' '→' '↑' '↑']
 ['↓' '←' '←' '←' '←' '↓' '↓' '↑' '↑' '↑']
 ['↓' '←' '←' '←' '←' '↓' '↓' '←' '→' '↑']
 ['↓' '←' '←' '←' '←' '→' '←' '←' '→' '↓']
 ['↓' '←' '←' '←' '←' '↓' '↑' '←' '→' '↓']
 ['↓' '←' '←' '←' '←' '←' '←' '↓' '↓' '↓']
 ['↓' '←' '←' '←' '←' '←' '↓' '↓' '↓' '↓']
 ['↓' '←' '←' '←' '←' '←' '→' '→' '→' '↓']]
```

0.9 Question 11

[26]:
```
def getcDbMatrices(arrows_expert,P_ss,ind,gamma,lambda_val,maximum):
    I = np.identity(100)
    mat1 = np.zeros((300,100))
    iden_mat = np.zeros((300,100))
    i=0
    for s in range(100):
        opt = int(arrows_expert[ind][s])
        for action in range(len(P_ss)):
            if(opt==action):
```

```
                continue
            pa1 = P_ss[opt]
            pa = P_ss[action]
            mat1[i,:] = np.matmul((pa1[s]-pa[s]).reshape(1,100), np.linalg.
 ↪inv(I-gamma*pa1))
            iden_mat[i,s] = 1
            i=i+1
    mat1 = -mat1
    R = np.vstack((mat1, mat1,-I, I, I, -I))
    t = np.vstack((iden_mat, np.zeros((700,100))))
    u = np.vstack((np.zeros((600,100)), -I, -I, np.zeros((200,100))))
    D = np.hstack((R,t,u))
    ones = np.zeros((100,1))+1
    c = np.vstack((np.zeros((100,1)), ones, -lambda_val*ones))
    b = np.zeros((800,1))
    Rmax = np.zeros((100,1))+maximum[ind]
    b = np.vstack((b, Rmax, Rmax))
    return c,D,b
```

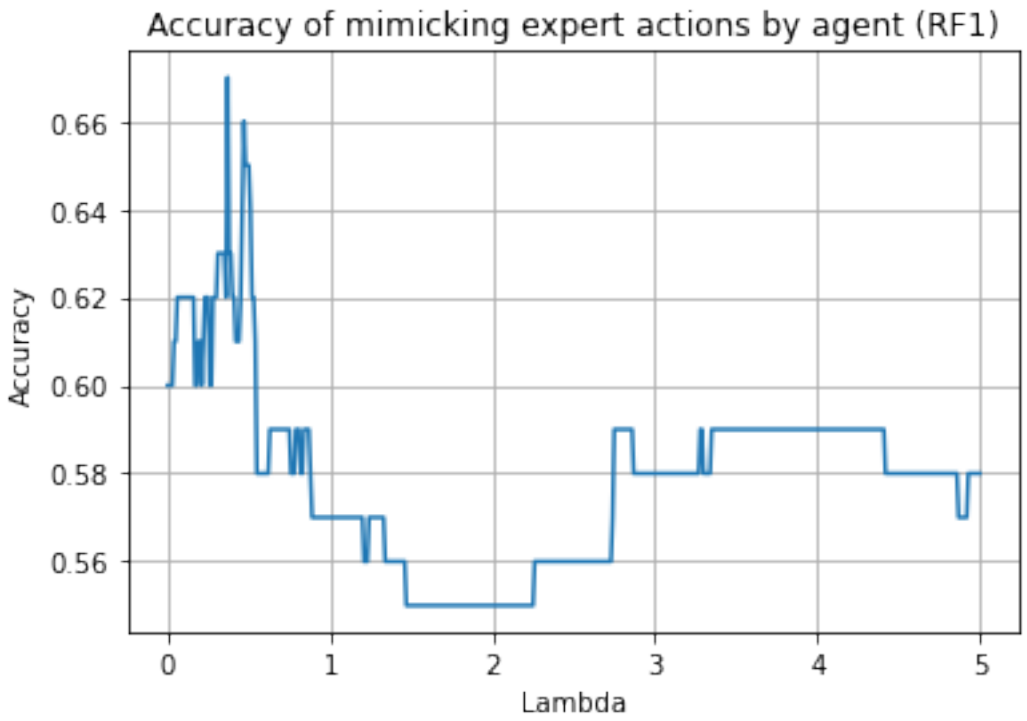
[27]:
```
w = 0.1
gamma = 0.8
epsilon = 0.01
tp_u, tp_d, tp_l, tp_r = tp_calc(w)

P_ss = [tp_u, tp_d, tp_l, tp_r]

#Policies for Reward function 1 (expert)
V_opt_1,␣
 ↪N_1=value_iteration(state_num,w,gamma,rwd_1,epsilon,tp_r,tp_l,tp_u,tp_d)
policy_1, pic_arrow_1,arrows_1 =␣
 ↪policy_iteration(state_num,w,gamma,rwd_1,epsilon,tp_u, tp_d, tp_l, tp_r)

#Policies for Reward function 2 (expert)
V_opt_2,␣
 ↪N_2=value_iteration(state_num,w,gamma,rwd_2,epsilon,tp_r,tp_l,tp_u,tp_d)
policy_2, pic_arrow_2,arrows_2 =␣
 ↪policy_iteration(state_num,w,gamma,rwd_2,epsilon,tp_u, tp_d, tp_l, tp_r)

arrows_expert = [np.transpose(arrows_1).flatten(), np.transpose(arrows_2).
 ↪flatten()]
```

[28]:
```
lambdas = np.arange(0,5.01,0.01)
maximum = [1,10]
acc_list_list = []
I = np.identity(100)
ind = 0
```

```
while(ind<2):
    acc_list = []
    for i,lambda_val in enumerate(tqdm(lambdas)):

        #Extract reward function
        c,D,b = getcDbMatrices(arrows_expert,P_ss,ind,gamma,lambda_val,maximum)
        solvers.options['show_progress']=False
        sol = solvers.lp(matrix(c),matrix(D),matrix(b))
        R = np.array(sol['x'][:100])

        #Extract agent's policy from extracted reward function
        policy_agent, pic_arrow_agent,arrows_agent =␣
↪policy_iteration(state_num,w,gamma,

                                                                    ␣
↪R,epsilon,tp_u, tp_d, tp_l, tp_r)

        #Performance measure
        acc=0
        for j in range(len(np.transpose(arrows_agent).flatten())):
            if(np.transpose(arrows_agent).flatten()[j]==arrows_expert[ind][j]):
                acc = acc+1
        acc = acc/100.0

        acc_list.append(acc)
    acc_list_list.append(acc_list)
    ind = ind+1
```

```
100%|      | 501/501 [00:35<00:00, 13.93it/s]
100%|      | 501/501 [00:38<00:00, 12.89it/s]
```

[29]:
```
plt.plot(lambdas,acc_list_list[0])
plt.xlabel("Lambda")
plt.ylabel("Accuracy")
plt.title("Accuracy of mimicking expert actions by agent (RF1)")
plt.grid()
plt.savefig('Q11.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.10 Question 12

[30]:
```
print("Max value of accuracy:",acc_list_list[0][np.argmax(acc_list_list[0])])
print("Corresponding value of lambda:",lambdas[np.argmax(acc_list_list[0])])
```

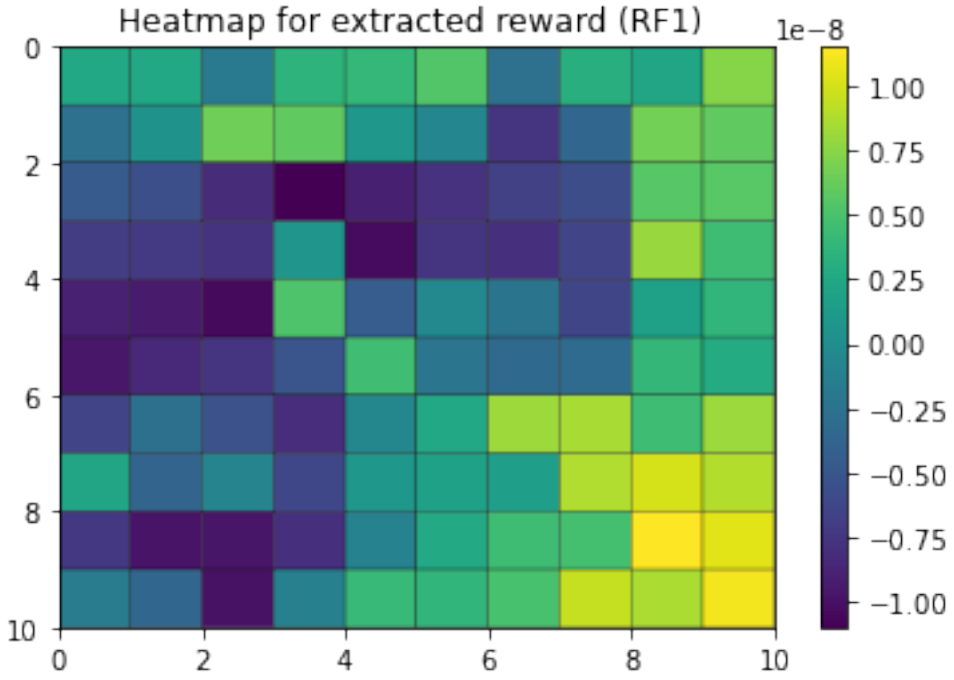
```
Max value of accuracy: 0.67
Corresponding value of lambda: 0.37
```

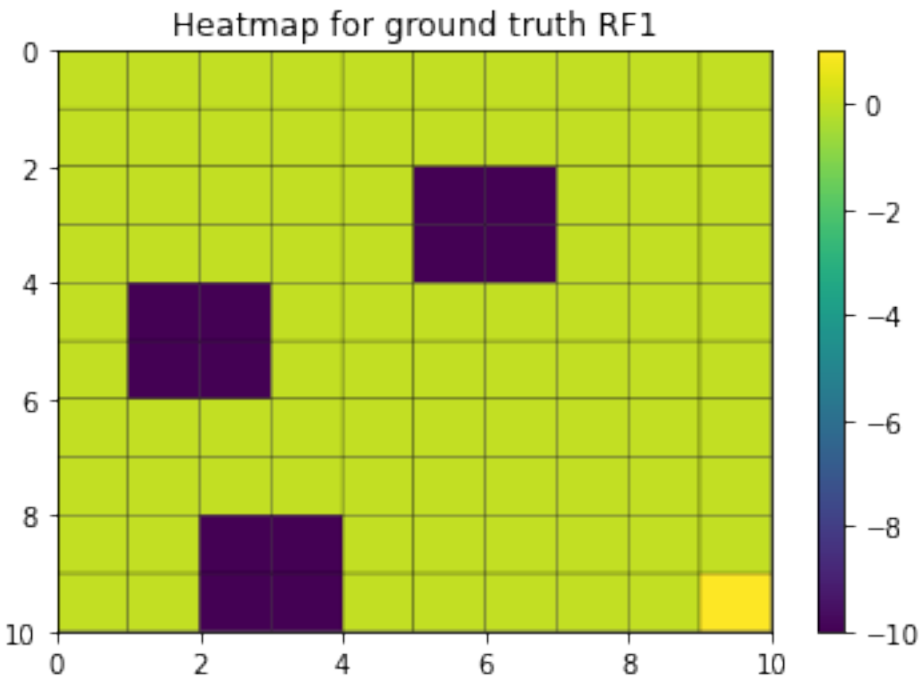
0.11 Question 13

[31]:
```
c,D,b = getcDbMatrices(arrows_expert,P_ss,0,gamma,0.37,maximum)
solvers.options['show_progress']=False
sol = solvers.lp(matrix(c),matrix(D),matrix(b))
R = np.array(sol['x'][:100])
R = np.transpose(R.reshape(10,10))
```

[32]:
```
plt.pcolor(R,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for extracted reward (RF1) ")
plt.savefig('Q13a.png',dpi=300,bbox_inches='tight')
plt.show()
```

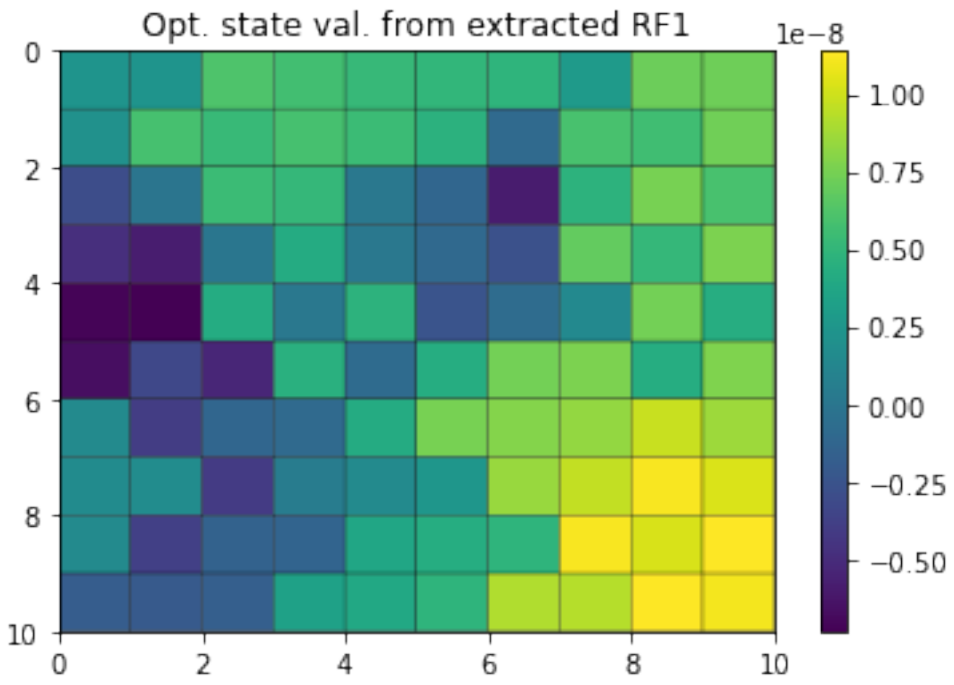
```
plt.pcolor(rwd_1,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for ground truth RF1")
plt.savefig('Q13b.png',dpi=300,bbox_inches='tight')
plt.show()
```





0.12 Question 14

[33]:
```
V_extracted, N=value_iteration(state_num,0.
↪1,gamma,R,epsilon,tp_r,tp_l,tp_u,tp_d)
plt.pcolor(V_extracted,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Opt. state val. from extracted RF1")
plt.savefig('Q14.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.13 Question 16

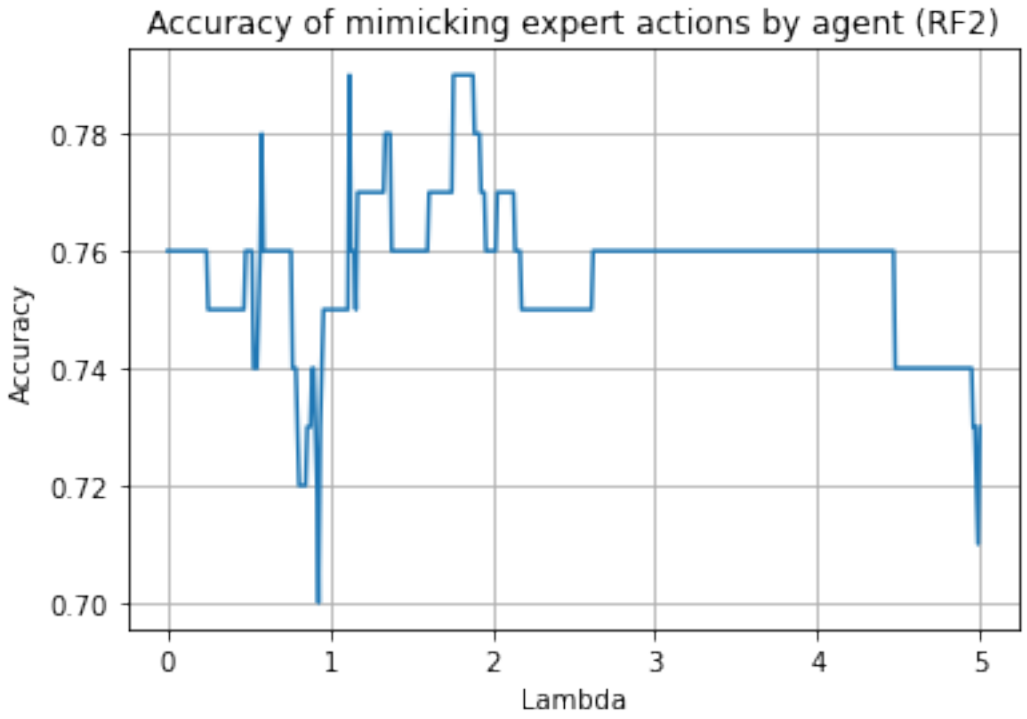
[34]:
```
policy_agent, pic_arrow_agent,arrows_agent = policy_iteration(state_num,w,gamma,
                                                              R,epsilon,tp_u,␣
 ↪tp_d, tp_l, tp_r)
print(pic_arrow_agent)
```

```
[['→' '↓' '↓' '↓' '→' '↑' '←' '→' '→' '↑']
 ['→' '→' '→' '←' '←' '↑' '←' '→' '→' '↑']
 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↑']
 ['↑' '↑' '→' '↓' '←' '↓' '→' '→' '↑' '←']
 ['↑' '→' '→' '↑' '←' '↓' '↓' '→' '↑' '↑']
 ['↓' '↓' '→' '↑' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '↓' '↓' '→' '→' '→' '→' '↓' '↓' '↓']
 ['←' '←' '←' '→' '→' '↑' '→' '→' '↓' '↓']
 ['↑' '↑' '↑' '→' '↓' '→' '→' '→' '→' '↓']
 ['↓' '←' '→' '→' '→' '→' '→' '→' '→' '↓']]
```

0.14 Question 18

[35]:
```
plt.plot(lambdas,acc_list_list[1])
plt.xlabel("Lambda")
plt.ylabel("Accuracy")
plt.title("Accuracy of mimicking expert actions by agent (RF2)")
```

```
plt.grid()
plt.savefig('Q18.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.15 Question 19

[36]:
```
print("Max value of accuracy:",acc_list_list[1][np.argmax(acc_list_list[1])])
print("Corresponding value of lambda:",lambdas[np.argmax(acc_list_list[1])])
```

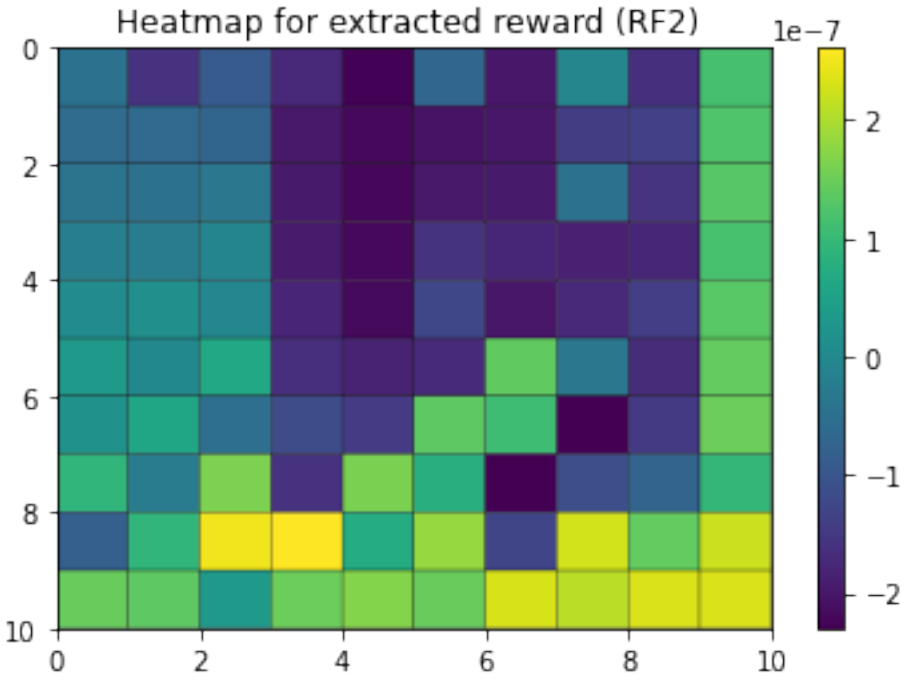
```
Max value of accuracy: 0.79
Corresponding value of lambda: 1.12
```

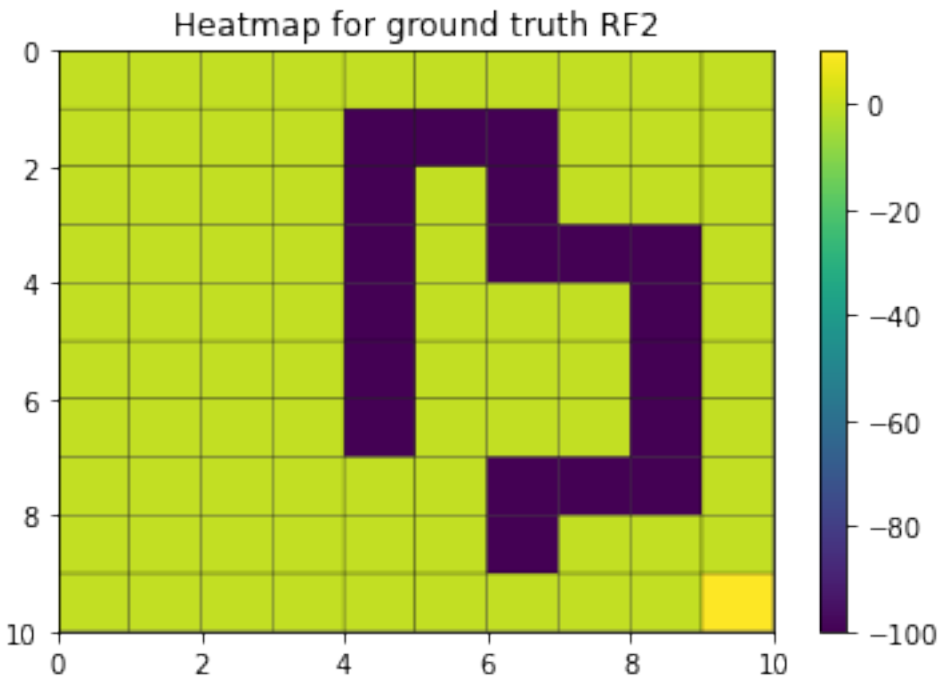
0.16 Question 20

[37]:
```
c,D,b = getcDbMatrices(arrows_expert,P_ss,1,gamma,1.12,maximum)
solvers.options['show_progress']=False
sol = solvers.lp(matrix(c),matrix(D),matrix(b))
R = np.array(sol['x'][:100])
R = np.transpose(R.reshape(10,10))
```

[38]:
```
plt.pcolor(R,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
```

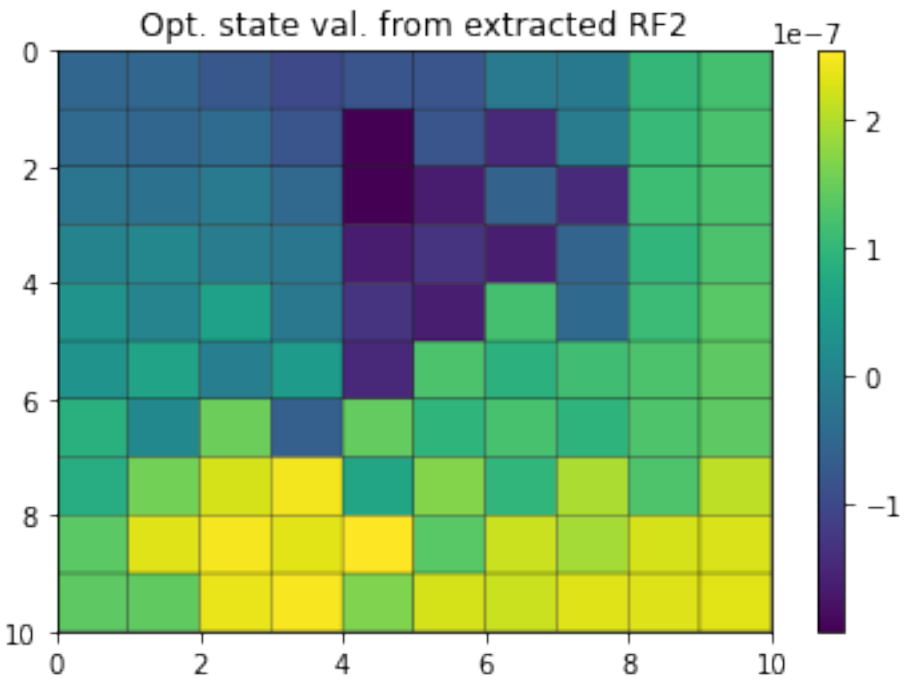
```
plt.title("Heatmap for extracted reward (RF2) ")
plt.savefig('Q20a.png',dpi=300,bbox_inches='tight')
plt.show()

plt.pcolor(rwd_2,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for ground truth RF2")
plt.savefig('Q20b.png',dpi=300,bbox_inches='tight')
plt.show()
```





0.17 Question 21

[39]:
```
V_extracted, N=value_iteration(state_num,0.
 ↪1,gamma,R,epsilon,tp_r,tp_l,tp_u,tp_d)
plt.pcolor(V_extracted,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Opt. state val. from extracted RF2")
plt.savefig('Q21.png',dpi=300,bbox_inches='tight')
plt.show()
```



0.18 Question 23

[40]:
```
policy_agent, pic_arrow_agent,arrows_agent = policy_iteration(state_num,w,gamma,
                                                              R,epsilon,tp_u,␣
 ↪tp_d, tp_l, tp_r)
print(pic_arrow_agent)
```

```
[['↑' '←' '↓' '←' '→' '↑' '→' '↑' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↑' '→' '↑' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '→' '→' '→' '→' '→']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '↓' '→' '↓']
 ['↓' '↓' '↓' '←' '↓' '↓' '↓' '←' '→' '↓']
 ['↓' '↓' '↓' '←' '↓' '↓' '←' '←' '→' '↓']
 ['←' '→' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
 ['→' '→' '→' '←' '←' '↓' '↓' '↓' '↓' '↓']
 ['↓' '↑' '↑' '↑' '←' '→' '↓' '→' '→' '↓']]
```

0.19 Question 25

[65]:
```
def policy_iteration_Q25(state_num,w,gamma,rwd,epsilon,tp_u, tp_d, tp_l, tp_r):
    state_val=np.zeros(100)
    delta=np.inf
    policy=np.zeros(100)
```

```
    arrows=np.zeros(100)
    r = (rwd.T).ravel()
    while(delta>epsilon):
        delta=0
        old_state_val=np.copy(state_val)
        for s in range(0,100):
            u_val=np.sum(tp_u[s]*(r+gamma*old_state_val))
            d_val=np.sum(tp_d[s]*(r+gamma*old_state_val))
            l_val=np.sum(tp_l[s]*(r+gamma*old_state_val))
            r_val=np.sum(tp_r[s]*(r+gamma*old_state_val))
            state_val[s]=max(u_val,d_val,r_val,l_val)
            delta=max(delta,abs(old_state_val[s]-state_val[s]))
    for s in range(0,100):
        u_val=np.sum(tp_u[s]*(r+gamma*state_val))
        d_val=np.sum(tp_d[s]*(r+gamma*state_val))
        l_val=np.sum(tp_l[s]*(r+gamma*state_val))
        r_val=np.sum(tp_r[s]*(r+gamma*state_val))

        #Set state values of edge/corners states to -infinity
        if s!=99:
            if s%10==0:
                u_val=-np.inf
            if s%10==9:
                d_val = -np.inf
            if s<=9:
                l_val=-np.inf
            if s>=90:
                r_val=-np.inf

        arr=[u_val,d_val,l_val,r_val] #Up: 0, Down: 1, Left: 2, Right: 3
        policy[s]=np.amax(arr)
        arrows[s]=arr.index(np.amax(arr))
    arrows = np.transpose(arrows.reshape(10,10))
    policy = np.transpose(policy.reshape(10,10))
    pic_arrow=np.chararray((10,10),unicode=True)
    for i in range(10):
        for j in range(10):
            if(arrows[j][i]==0.):
                pic_arrow[j][i] = u'\u2191'
            elif(arrows[j][i]==1.):
                pic_arrow[j][i] = u'\u2193'
            elif(arrows[j][i]==2.):
                pic_arrow[j][i]=u'\u2190'
            elif(arrows[j][i]==3.):
                pic_arrow[j][i] = u'\u2192'
    return policy, pic_arrow, arrows
```

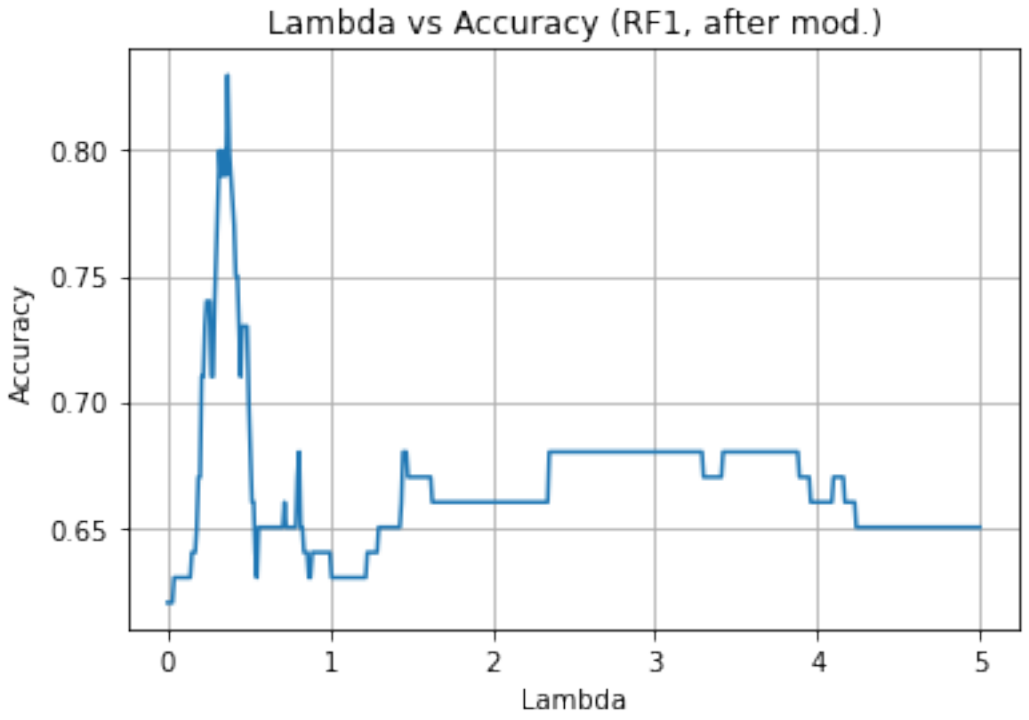
[66]:
```
w = 0.1
gamma = 0.8
epsilon = pow(10,-10)
tp_u, tp_d, tp_l, tp_r = tp_calc(w)

P_ss = [tp_u, tp_d, tp_l, tp_r]

#Policies for Reward function 1 (expert)
V_opt_1,␣
↪N_1=value_iteration(state_num,w,gamma,rwd_1,epsilon,tp_r,tp_l,tp_u,tp_d)
policy_1, pic_arrow_1,arrows_1 =␣
↪policy_iteration(state_num,w,gamma,rwd_1,epsilon,tp_u, tp_d, tp_l, tp_r)

#Policies for Reward function 2 (expert)
V_opt_2,␣
↪N_2=value_iteration(state_num,w,gamma,rwd_2,epsilon,tp_r,tp_l,tp_u,tp_d)
policy_2, pic_arrow_2,arrows_2 =␣
↪policy_iteration(state_num,w,gamma,rwd_2,epsilon,tp_u, tp_d, tp_l, tp_r)

arrows_expert = [np.transpose(arrows_1).flatten(), np.transpose(arrows_2).
↪flatten()]
```

[67]:
```
lambdas = np.arange(0,5.01,0.01)
maximum = [1,10]
acc_list_list = []
I = np.identity(100)
ind = 0
while(ind<2):
    acc_list = []
    for i,lambda_val in enumerate(tqdm(lambdas)):

        #Extract reward function
        c,D,b = getcDbMatrices(arrows_expert,P_ss,ind,gamma,lambda_val,maximum)
        solvers.options['show_progress']=False
        sol = solvers.lp(matrix(c),matrix(D),matrix(b))
        R = np.array(sol['x'][:100])

        #Extract agent's policy from extracted reward function
        policy_agent, pic_arrow_agent,arrows_agent =␣
↪policy_iteration_Q25(state_num,w,gamma,
                                                                    ␣
↪R,epsilon,tp_u, tp_d, tp_l, tp_r)

        #Performance measure
        acc=0
        for j in range(len(np.transpose(arrows_agent).flatten())):
            if(np.transpose(arrows_agent).flatten()[j]==arrows_expert[ind][j]):
```

```
                acc = acc+1
        acc = acc/100.0

        acc_list.append(acc)
    acc_list_list.append(acc_list)
    ind = ind+1
```

```
100%|      | 501/501 [00:58<00:00,  8.62it/s]
100%|      | 501/501 [01:20<00:00,  6.19it/s]
```

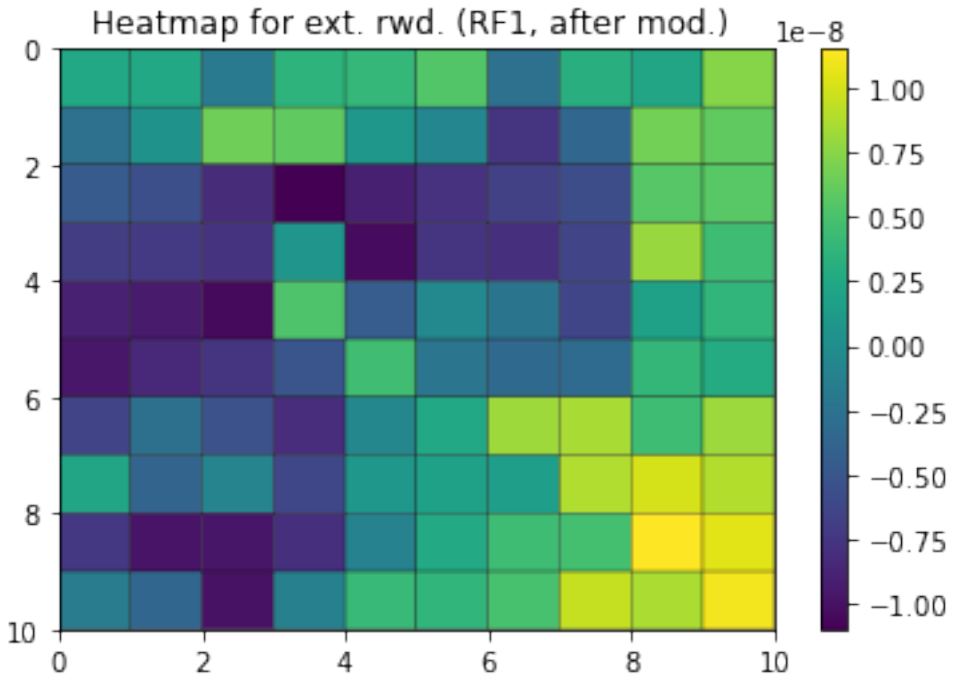
[68]:
```
plt.plot(lambdas,acc_list_list[0])
plt.xlabel("Lambda")
plt.ylabel("Accuracy")
plt.title("Lambda vs Accuracy (RF1, after mod.)")
plt.grid()
plt.savefig('Q25a.png',dpi=300,bbox_inches='tight')
plt.show()
```



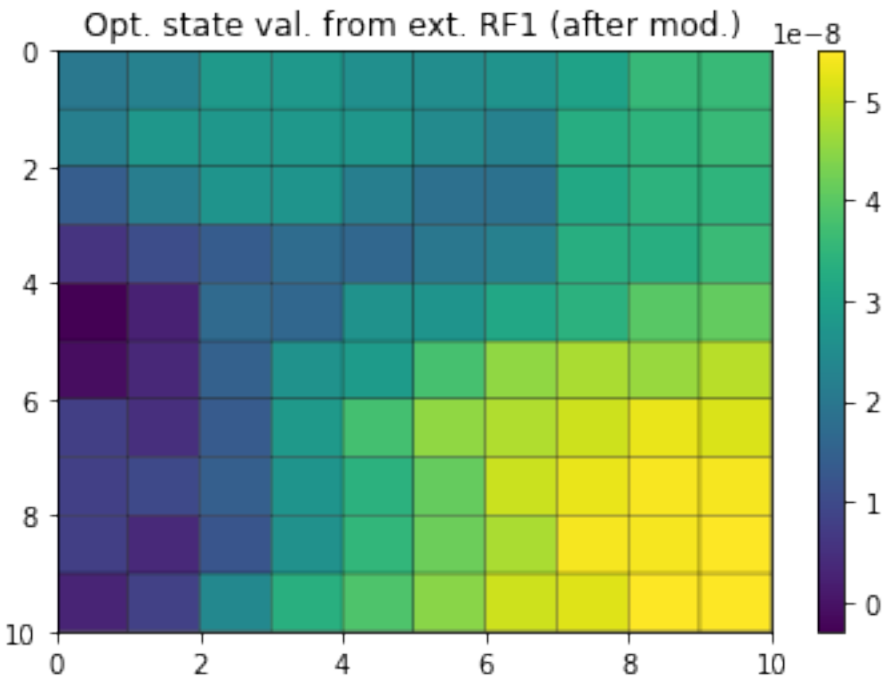
[69]:
```
print("Max value of accuracy:",acc_list_list[0][np.argmax(acc_list_list[0])])
print("Corresponding value of lambda:",lambdas[np.argmax(acc_list_list[0])])
```

```
Max value of accuracy: 0.83
Corresponding value of lambda: 0.37
```

[74]:
```
c,D,b = getcDbMatrices(arrows_expert,P_ss,0,gamma,0.37,maximum)
solvers.options['show_progress']=False
sol = solvers.lp(matrix(c),matrix(D),matrix(b))
R = np.array(sol['x'][:100])
R = np.transpose(R.reshape(10,10))

plt.pcolor(R,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for ext. rwd. (RF1, after mod.) ")
plt.savefig('Q25b.png',dpi=300,bbox_inches='tight')
plt.show()
```



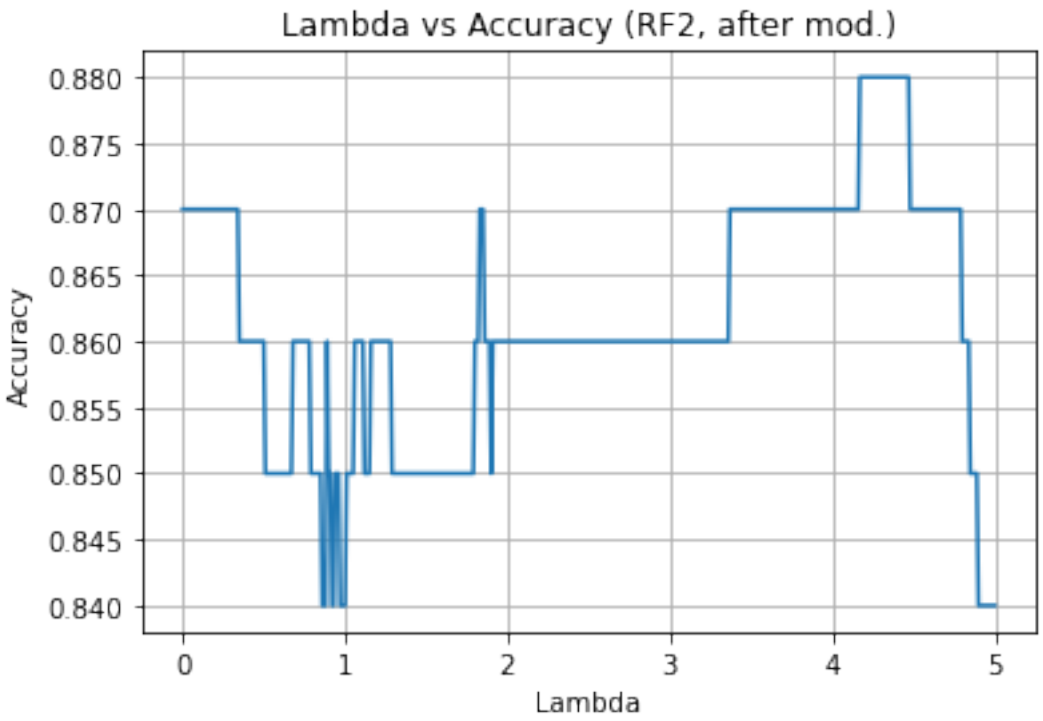
[77]:
```
V_extracted, N=value_iteration(state_num,0.
↪1,gamma,R,epsilon,tp_r,tp_l,tp_u,tp_d)
plt.pcolor(V_extracted,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Opt. state val. from ext. RF1 (after mod.)")
plt.savefig('Q25c.png',dpi=300,bbox_inches='tight')
plt.show()
```



[75]:
```
policy_agent, pic_arrow_agent,arrows_agent =␣
↪policy_iteration_Q25(state_num,w,gamma,
                                                    R,epsilon,tp_u,␣
↪tp_d, tp_l, tp_r)
print(pic_arrow_agent)
```

```
[['→' '↓' '↓' '↓' '←' '←' '→' '→' '→' '↓']
 ['→' '→' '→' '←' '←' '↑' '→' '→' '→' '↑']
 ['↑' '↑' '↑' '↑' '↑' '↑' '→' '→' '↓' '↑']
 ['↑' '↑' '→' '↓' '↓' '↓' '↓' '→' '↓' '↓']
 ['↑' '→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '→' '→' '→' '↓' '↓' '↓' '↓' '↓' '↓']
 ['↓' '→' '→' '→' '→' '→' '→' '↓' '↓' '↓']
 ['→' '→' '→' '→' '→' '→' '→' '→' '↓' '↓']
 ['↑' '↑' '→' '→' '→' '→' '→' '→' '→' '↓']
 ['→' '→' '→' '→' '→' '→' '→' '→' '→' '↓']]
```

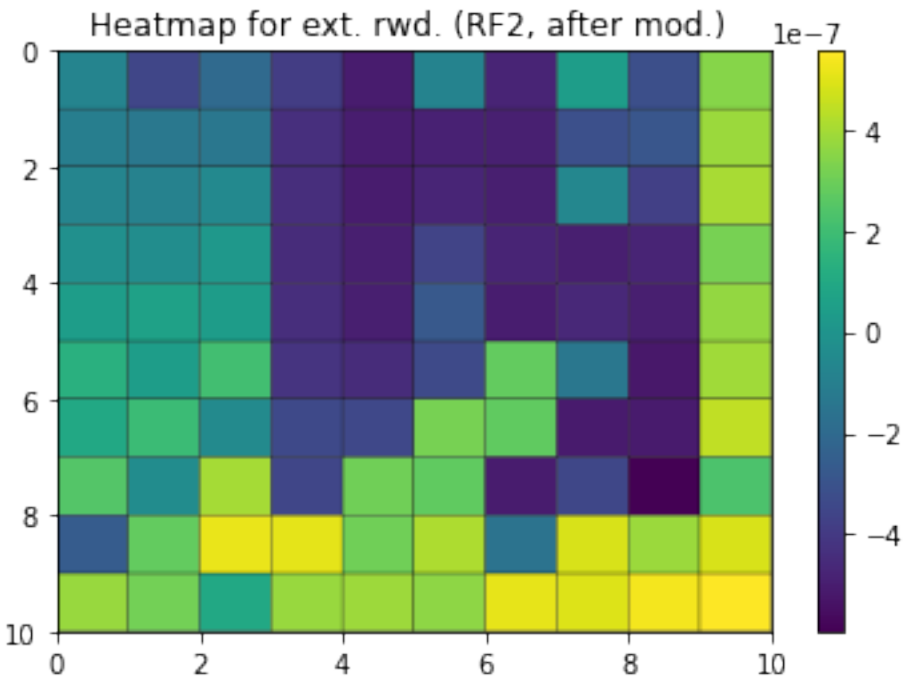
[78]:
```
plt.plot(lambdas,acc_list_list[1])
plt.xlabel("Lambda")
plt.ylabel("Accuracy")
plt.title("Lambda vs Accuracy (RF2, after mod.)")
plt.grid()
plt.savefig('Q25d.png',dpi=300,bbox_inches='tight')
plt.show()
```



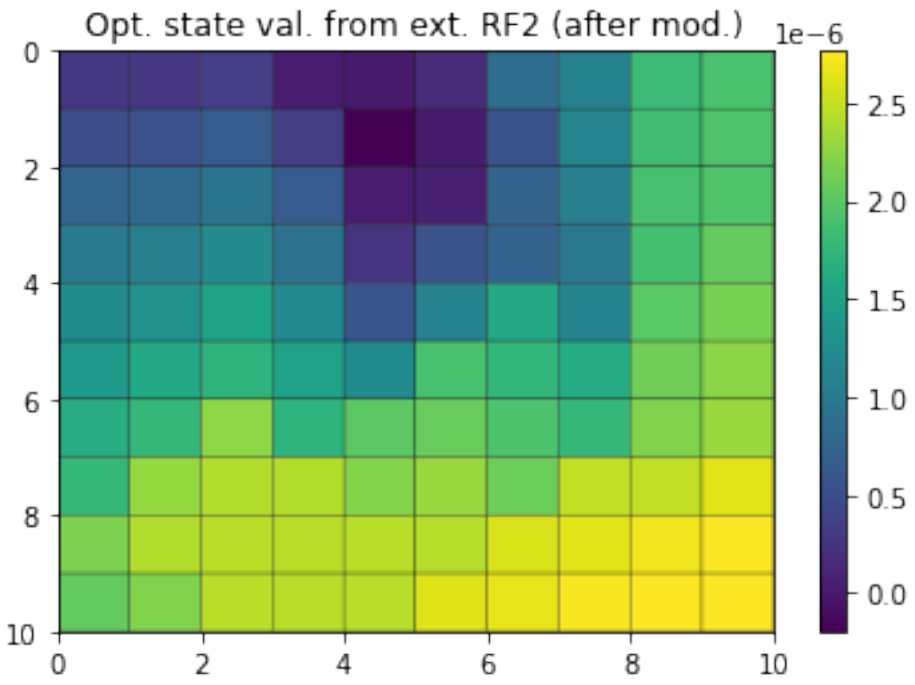
[79]:
```
print("Max value of accuracy:",acc_list_list[1][np.argmax(acc_list_list[1])])
print("Corresponding value of lambda:",lambdas[np.argmax(acc_list_list[1])])
```

```
Max value of accuracy: 0.88
Corresponding value of lambda: 4.17
```

[80]:
```
c,D,b = getcDbMatrices(arrows_expert,P_ss,1,gamma,4.17,maximum)
solvers.options['show_progress']=False
sol = solvers.lp(matrix(c),matrix(D),matrix(b))
R = np.array(sol['x'][:100])
R = np.transpose(R.reshape(10,10))

plt.pcolor(R,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Heatmap for ext. rwd. (RF2, after mod.) ")
plt.savefig('Q25e.png',dpi=300,bbox_inches='tight')
plt.show()
```



[81]:
```
V_extracted, N=value_iteration(state_num,0.
 ↪1,gamma,R,epsilon,tp_r,tp_l,tp_u,tp_d)
plt.pcolor(V_extracted,edgecolors='black')
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("Opt. state val. from ext. RF2 (after mod.)")
plt.savefig('Q25f.png',dpi=300,bbox_inches='tight')
plt.show()
```



[82]:
```
policy_agent, pic_arrow_agent,arrows_agent =␣
↪policy_iteration_Q25(state_num,w,gamma,
                                                    R,epsilon,tp_u,␣
↪tp_d, tp_l, tp_r)
print(pic_arrow_agent)
```

```
[['↓' '↓' '↓' '←' '→' '→' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↑' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '→' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '←' '↓' '↓' '→' '→' '↓']
 ['↓' '↓' '↓' '←' '→' '↓' '↓' '↓' '→' '↓']
 ['↓' '↓' '↓' '←' '↓' '↓' '↓' '←' '→' '↓']
 ['↓' '↓' '↓' '←' '↓' '↓' '←' '←' '→' '↓']
 ['→' '→' '↓' '↓' '↓' '↓' '←' '↓' '↓' '↓']
 ['→' '→' '→' '←' '←' '↓' '↓' '↓' '↓' '↓']
 ['→' '↑' '↑' '↑' '→' '→' '→' '→' '→' '↓']]
```