University of California, Los Angeles
Department of Electrical and Computer Engineering
ECE 232E Large Scale Social and Complex Networks: Design and Algorithms

Project 3: Reinforcement learning and Inverse Reinforcement learning

505430686 Viacheslav Inderiakin

904627828 Mia Levy

805626088 Connor Roberts

804737257 Tameez Latib

May 22, 2021

1. Reinforcement Learning

Question 1: For visualization purpose, generate heat maps of Reward function 1 and Reward function 2. For the heat maps, make sure you display the coloring scale. You will have 2 plots for this question.

Answer: Heatmaps for reward functions R_1 and R_2 are shown in figure 1.

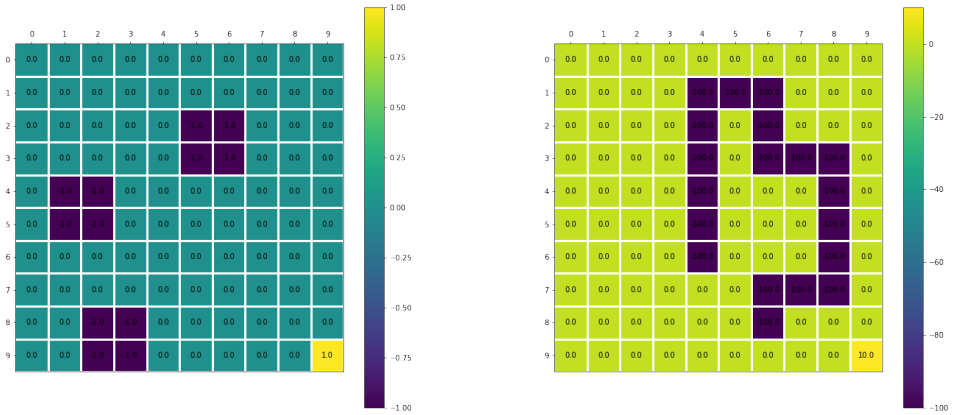


Figure 1: Heatmaps for R_1 (left) and R_2 (right).

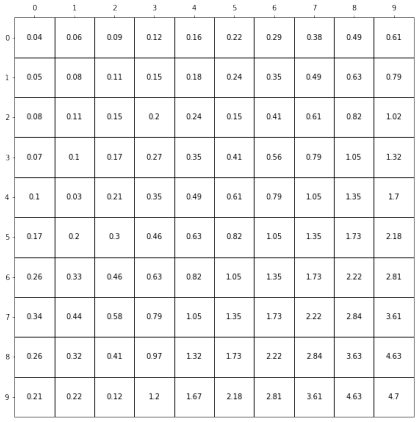
Question 2: Create the environment of the agent using the information provided in section 2. To be specific, create the MDP by setting up the state-space, action set, transition probabilities, discount factor, and reward function. For creating the environment, use the following set of parameters:

- Number of states = 100
- Number of actions = 4
- $w = 0.1$
- Discount factor $\gamma = 0.8$
- Reward function R_1.

After you have created the environment, then write an optimal state-value function that takes as input the environment of the agent and outputs the optimal value of each state in the grid. For the optimal state-value function, you have to implement the Initialization (lines 2-4) and Estimation (lines 5-13) steps of the Value Iteration algorithm. For the estimation step, use $\varepsilon = 0.01$. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal value of that state. In this part of question, you should have 1 plot. Let's assume that your value iteration algorithm converges in N steps. Plot snapshots of state values in 5 different steps linearly distributed from 1 to N. Report N and your step numbers. What observations do you have from the plots?

Answer: In this problem, we observed that the value iteration algorithm converges in $N = 20$ steps to the values shown in figure 2. As expected, we can see that the values

closest to the state s_{99}, which has positive reward, have the highest value. The higher is the distance between a state s_i and s_{99}, the the more discounted the expected reward is, and therefore, the lower is the value of $V(s_i)$, which is also reflected in the figure 2. We can also observe that the values of the states close to the regions with negative reward are lower. This is because wind creates a probability to randomly move to these regions.



	0	1	2	3	4	5	6	7	8	9
0	0.04	0.06	0.09	0.12	0.16	0.22	0.29	0.38	0.49	0.61
1	0.05	0.08	0.11	0.15	0.18	0.24	0.35	0.49	0.63	0.79
2	0.08	0.11	0.15	0.2	0.24	0.15	0.41	0.61	0.82	1.02
3	0.07	0.1	0.17	0.27	0.35	0.41	0.56	0.79	1.05	1.32
4	0.1	0.03	0.21	0.35	0.49	0.61	0.79	1.05	1.35	1.7
5	0.17	0.2	0.3	0.46	0.63	0.82	1.05	1.35	1.73	2.18
6	0.26	0.33	0.46	0.63	0.82	1.05	1.35	1.73	2.22	2.81
7	0.34	0.44	0.58	0.79	1.05	1.35	1.73	2.22	2.84	3.61
8	0.26	0.32	0.41	0.97	1.32	1.73	2.22	2.84	3.63	4.63
9	0.21	0.22	0.12	1.2	1.67	2.18	2.81	3.61	4.63	4.7

Figure 2: State-value function obtained for R_1, $w = 0.1$ and $\varepsilon = 0.01$.

The history of state values for $n = 3$, 7, 10, 13, 16 is provided in figure 3. From that figure, we can observe that states closest to the state s_{99} are updated first and with the highest amplitude of update. As the number of iteration increases, updates reach the states more and more distant states, until all states are reached and convergence happens.

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
1	0.0	-0.0	-0.0	-0.0	-0.0	-0.03	-0.03	-0.0	-0.0	-0.0
2	-0.0	-0.0	-0.0	-0.0	-0.03	-0.07	-0.07	-0.03	-0.0	-0.0
3	-0.0	-0.03	-0.03	-0.0	-0.03	-0.07	-0.07	-0.03	-0.0	-0.0
4	-0.03	-0.07	-0.07	-0.03	-0.0	-0.03	-0.03	-0.0	-0.0	-0.0
5	-0.03	-0.07	-0.07	-0.03	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
6	-0.0	-0.03	-0.03	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.51
7	-0.0	-0.0	-0.03	-0.03	-0.0	-0.0	-0.0	-0.0	0.53	1.29
8	-0.0	-0.03	-0.07	-0.07	-0.03	-0.0	-0.0	0.53	1.31	2.32
9	-0.0	-0.03	-0.1	-0.1	-0.03	-0.0	0.51	1.29	2.32	2.39

(a) $n = 3$

	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
1	-0.0	-0.0	-0.0	-0.0	-0.0	-0.03	-0.03	-0.0	-0.0	-0.0
2	-0.0	-0.0	-0.0	-0.0	-0.03	-0.07	-0.07	-0.03	-0.0	0.15
3	-0.0	-0.03	-0.03	-0.0	-0.03	-0.07	-0.07	-0.03	0.18	0.43
4	-0.03	-0.07	-0.07	-0.03	-0.0	-0.03	-0.03	0.18	0.47	0.81
5	-0.03	-0.07	-0.07	-0.03	-0.0	-0.0	0.18	0.47	0.85	1.29
6	-0.0	-0.03	-0.03	-0.0	-0.0	0.18	0.47	0.85	1.33	1.92
7	-0.0	-0.0	-0.03	-0.03	0.18	0.47	0.85	1.33	1.95	2.72
8	-0.0	-0.03	-0.07	0.1	0.44	0.84	1.33	1.95	2.74	3.75
9	-0.0	-0.03	-0.1	0.33	0.78	1.29	1.92	2.72	3.75	3.81

(b) $n = 7$

	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.08	0.19
1	-0.0	-0.0	-0.0	-0.0	-0.0	-0.03	-0.03	0.08	0.21	0.36
2	-0.0	-0.0	-0.0	-0.0	-0.03	-0.07	0.0	0.18	0.39	0.6
3	-0.0	-0.03	-0.03	-0.0	-0.03	0.0	0.14	0.37	0.63	0.89
4	-0.03	-0.07	-0.07	-0.03	0.08	0.19	0.37	0.63	0.93	1.27
5	-0.03	-0.07	-0.07	0.05	0.21	0.4	0.63	0.93	1.31	1.76
6	-0.0	-0.03	0.05	0.21	0.4	0.63	0.93	1.31	1.8	2.38
7	-0.0	0.04	0.16	0.37	0.63	0.93	1.31	1.8	2.42	3.18
8	-0.0	0.0	0.08	0.55	0.9	1.31	1.8	2.42	3.21	4.21
9	-0.0	-0.03	-0.08	0.78	1.24	1.76	2.38	3.18	4.21	4.28

(c) $n = 10$

	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	-0.0	-0.0	-0.0	0.03	0.1	0.18	0.3	0.42
1	-0.0	-0.0	-0.0	-0.0	-0.0	0.05	0.16	0.3	0.44	0.6
2	-0.0	-0.0	-0.0	0.02	0.05	-0.01	0.22	0.41	0.62	0.83
3	-0.0	-0.03	-0.01	0.07	0.16	0.22	0.37	0.6	0.86	1.12
4	-0.03	-0.07	0.02	0.16	0.3	0.42	0.6	0.86	1.16	1.5
5	-0.01	0.01	0.11	0.27	0.44	0.63	0.86	1.16	1.54	1.99
6	0.07	0.14	0.27	0.44	0.63	0.86	1.16	1.54	2.03	2.61
7	0.15	0.25	0.39	0.6	0.86	1.16	1.54	2.03	2.65	3.42
8	0.11	0.17	0.26	0.78	1.13	1.54	2.03	2.65	3.44	4.44
9	0.09	0.1	0.03	1.01	1.47	1.99	2.61	3.42	4.44	4.51

(d) $n = 13$

	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	0.01	0.04	0.09	0.14	0.21	0.3	0.41	0.53
1	-0.0	0.01	0.03	0.07	0.1	0.16	0.27	0.41	0.55	0.71
2	0.01	0.03	0.07	0.12	0.16	0.08	0.34	0.53	0.74	0.94
3	0.0	0.02	0.1	0.19	0.27	0.34	0.48	0.71	0.97	1.24
4	0.02	-0.03	0.13	0.27	0.41	0.53	0.71	0.98	1.27	1.62
5	0.09	0.12	0.23	0.39	0.56	0.74	0.98	1.28	1.66	2.1
6	0.18	0.25	0.38	0.56	0.74	0.98	1.28	1.66	2.14	2.73
7	0.26	0.36	0.5	0.71	0.98	1.28	1.66	2.14	2.76	3.53
8	0.2	0.26	0.35	0.89	1.25	1.65	2.14	2.76	3.55	4.56
9	0.16	0.17	0.08	1.12	1.59	2.1	2.73	3.53	4.56	4.62

(e) $n = 16$

Figure 3: Snapshots of state-value functions after n iterations.

Question 3: Generate a heat map of the optimal state values across the 2-D grid. For generating the heat map, you can use the same function provided in the hint earlier (see the hint after question 1).

Answer: A heatmap for the optimal state-value function is provided in figure 4. For ease of understanding, each cell is subscripted with its value. For the trends shown by $V(s)$, refer to question 2.

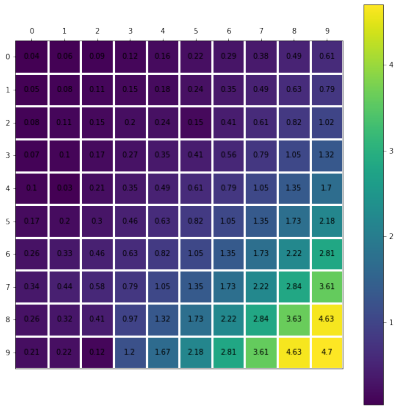


Figure 4: A heatmap of the optimal state-value function for R_1.

Question 4: Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in question 3 to explain)

Answer: As we get closer to the goal, we have higher value. Therefore we see a triangular pattern, or a gradient with the goal being the highest value and as you move farther from the goal, the value decreases. The value is especially low for states close to the regions with negative reward, as the wind creates a probability to randomly slip off to them.

Question 5: Implement the computation step of the value iteration algorithm (lines 14-17) to compute the optimal policy of the agent navigating the 2-D state-space. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The optimal actions should be displayed using arrows. Does the optimal policy of the agent match your intuition? Please provide a brief explanation. Is it possible for the agent to compute the optimal action to take at each state by observing the optimal values of it's neighboring states? In this question, you should have 1 plot.

Answer: The optimal policy for reward function R_1 and wind probability $w = 0.1$ is provided in figure 5. From that figure, we can see that the agent behavior is very intuitive, and the arrows point in the direction of the "goal" while trying to avoid "obstacles" which have negative reward.

To compute optimal action, the agent needs both the reward R and value V of the neighbouring states, as the optimal policy needs to maximize the expected reward

$\pi^*(s) = argmax_{a \in A}\Big(\mathbb{E}[G(s)]\Big) = argmax_{a \in A}\Big(\sum_{s' \in S} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]\Big)$, which depends on R.

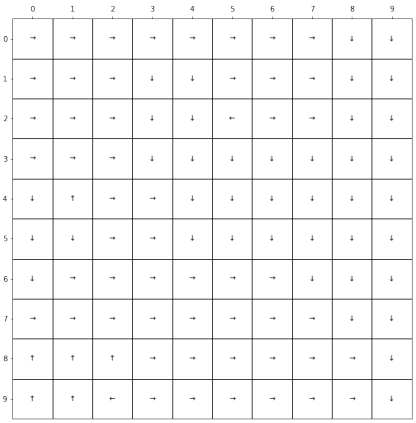


Figure 5: Optimal policy for R_1 and $w = 0.1$.

Question 6: Modify the environment of the agent by replacing Reward function 1 with Reward function 2. Use the optimal state-value function implemented in question 2 to compute the optimal value of each state in the grid. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal value of that state. In this question, you should have 1 plot.

Answer: Optimal state-value function for reward R_2 is provided in figure 6. For observed trends, see question 7.

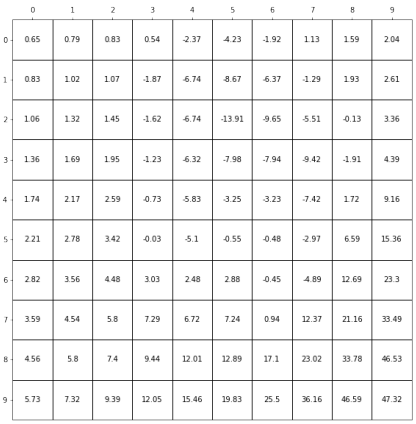


Figure 6: State-value function obtained for R_2, $w = 0.1$ and $\varepsilon = 0.01$.

Question 7: Generate a heat map of the optimal state values (found in question 6) across the 2-D grid. For generating the heat map, you can use the same function provided in the hint earlier. Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in this question to explain)

Answer: A heatmap for the optimal state-value function is provided in figure 7. For ease of understanding, each cell is subscripted with its value. This figure shows the same patterns as figure 4: the closer the state is to "obstacles" which have negative reward, the lower is state-value function. The value is the lowest for state s_{52}, which has the highest probability of being blown away to a state with negative R. Also similarly to figure 4, the states close to the goal have high reward.

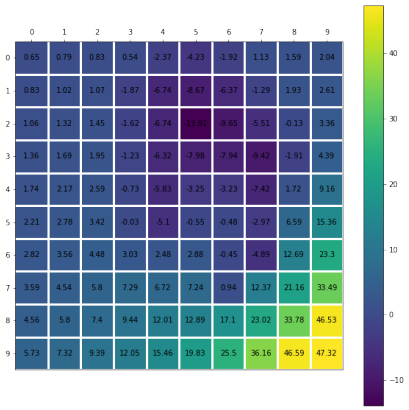


Figure 7: A heatmap of the optimal state-value function for R_2.

Question 8: Implement the computation step of the value iteration algorithm (lines 14-17) to compute the optimal policy of the agent navigating the 2-D state-space. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The optimal actions should be displayed using arrows. Does the optimal policy of the agent match your intuition? Please provide a brief explanation. In this question, you should have 1 plot.

Answer: The optimal policy for reward function R_1 and wind probability $w = 0.1$ is provided in figure 8. Similarly to the optimal policy for reward function R_1, optimal policy for reward function R_2 is intuitive. The arrows mostly point to the goal state. However, for states that are close to "obstacles" with negative reward, the agent prioritizes to move away from the obstacle. This is because the penalty for being randomly blown to these states is very high, and the agent tries to minimize that risk.

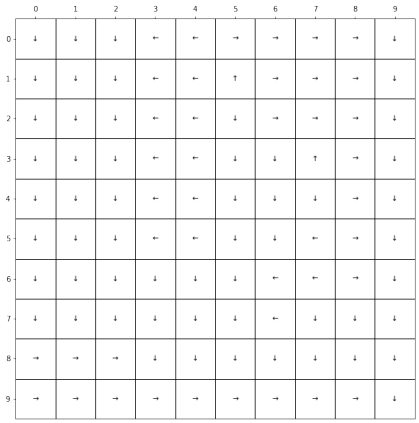


Figure 8: Optimal policy for R_2 and $w = 0.1$.

Question 9: Change the hyper parameter w to 0.6 and find the optimal policy map similar to previous question for reward functions. Explain the differences you observe. What do you think about value of new w compared to previous value? Choose the w that you think give rise to better optimal policy and use that w for the next stages of the project.

Answer: Optimal policies for reward functions R_1 and R_2 and wind probability $w = 0.6$ are provided in figure 9. From this figure, we can see that high wind probability makes the agent more cautious of the risk of being blown to a wall. This is why it prioritizes getting as far away from the states with negative reward as possible, and moves to the goal state only if the path is safe and short.

As we observed from figure 9, the increased value of w makes agent behavior more stochastic and prevents it from achieving higher expected rewards. This is why for Inverse Reinforcement Learning part of the project, we use $w = 0.1$.

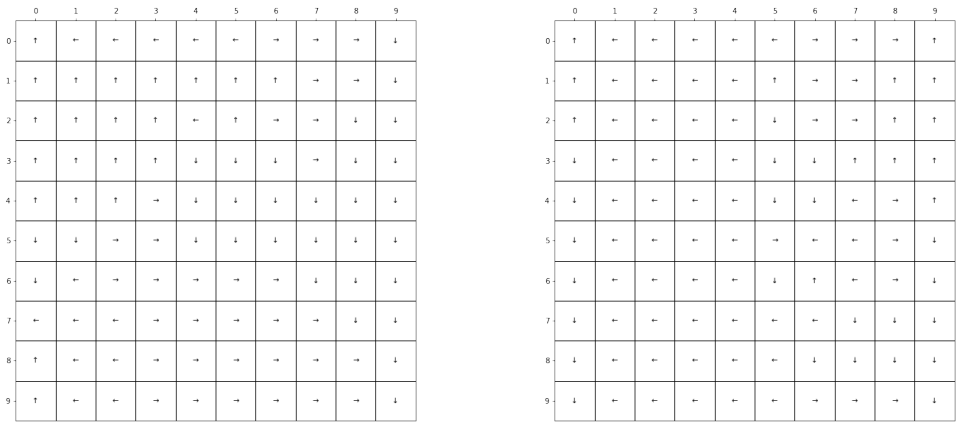


Figure 9: Optimal policy computed with $w = 0.6$ for R_1 (left) and R_2 (right).

2. Inverse Reinforcement learning (IRL)

Question 10: Express $\mathbf{c}$, $\mathbf{x}$, $\mathbf{D}$, $\mathbf{b}$ in terms of $\mathbf{R}$, $\mathbf{P}_a$, $\mathbf{P}_{a_1}$, t_i, $\mathbf{u}$, λ and R_{max}

Answer: To reformulate the problem

$$max_{\mathbf{R},\, t_i,\, u_i} \sum_{i=1}^{|S|} (t_i - \lambda u_i)$$

$$subject\ to \left[\Big(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i)\Big) \Big(\mathbf{I} - \gamma \mathbf{P}_{a_1}\Big)^{-1} \mathbf{R} \right] \geq t_i, \ \ \forall\, a \in A \setminus a_1, \ \forall\, i \in S$$

$$\Big(\mathbf{P}_{a_1} - \mathbf{P}_a\Big) \Big(\mathbf{I} - \gamma \mathbf{P}_{a_1}\Big)^{-1} \mathbf{R} \geq 0, \ \ \forall\, a \in A \setminus a_1$$

$$-\mathbf{u} \leq \mathbf{R} \leq \mathbf{u}$$

$$|\mathbf{R}_i| \leq R_{max}, \ \ \forall\, i \in S$$

As the problem

$$max_{\mathbf{x}}\ \mathbf{c}^T \mathbf{x}$$

$$subject\ to\ \mathbf{D}\mathbf{x} \leq \mathbf{b} \ \ \forall\, a \in A \setminus a_1$$

We need to set

$$c = \begin{bmatrix} \mathbf{1} \\ -\lambda \mathbf{1} \\ \mathbf{0} \end{bmatrix}, \ \ x = \begin{bmatrix} \mathbf{t} \\ \mathbf{u} \\ \mathbf{R} \end{bmatrix}$$

$$D = \begin{bmatrix} \mathbf{I}_a & \mathbf{0} & -(\mathbf{P_a} - \mathbf{P_{a1}})(\mathbf{I} - \gamma \mathbf{P_{a1}})^{-1} \\ \mathbf{0} & \mathbf{0} & -(\mathbf{P_a} - \mathbf{P_{a1}})(\mathbf{I} - \gamma \mathbf{P_{a1}})^{-1} \\ \mathbf{0} & -\mathbf{I} & \mathbf{I} \\ \mathbf{0} & -\mathbf{I} & -\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{bmatrix}, \ \ b = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{R_{max}} \\ \mathbf{R_{max}} \end{bmatrix}$$

where the rows of matrix $\mathbf{I}_a$ are structured in the way that $\Big[\mathbf{I}_a\Big]_{row,\, i} = 1$ corresponds to row $-(\mathbf{P_a}(i) - \mathbf{P_{a1}}(i))(\mathbf{I} - \gamma \mathbf{P_{a1}})^{-1}$ and zero otherwise.

Question 11: Sweep λ from 0 to 5 to get 500 evenly spaced values for λ. For each value of λ compute $O_A(s)$ by following the process described above. For this problem, use the optimal policy of the agent found in question 5 to fill in the $O_E(s)$ values. Then use equation 3 to compute the accuracy of the IRL algorithm for this value of λ. You need to repeat the above process for all 500 values of λ to get 500 data points. Plot λ (x-axis) against Accuracy (y-axis). In this question, you should have 1 plot.

Answer: The accuracy of IRL algorithm on optimal policy found in question 5 for different values of λ is provided in figure 10.

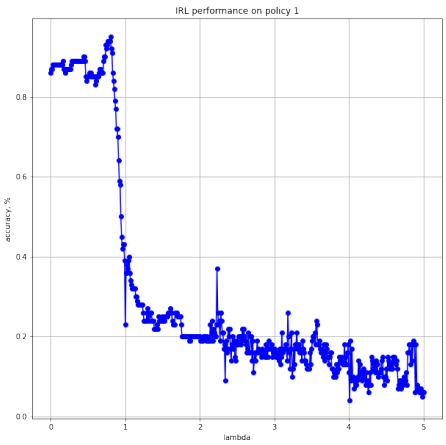


Figure 10: Inverse reinforcement learning performance on optimal policy of reward function R_1.

Question 12: Use the plot in question 11 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as $\lambda^{(1)}_{max}$. Please report $\lambda^{(1)}_{max}$

Answer: $\lambda^{(1)}_{max} = 0.8$

Question 13: For $\lambda^{(1)}_{max}$, generate heat maps of the ground truth reward and the extracted reward. Please note that the ground truth reward is the Reward function 1 and the extracted reward is computed by solving the linear program given by equation 2 with the parameter set to $\lambda^{(1)}_{max}$. In this question, you should have 2 plots.

Answer: Heatmaps of the ground truth reward and the reward reconstructed using IRL algorithm are provided in figure 11. From that figure we can conclude that while IRL algorithm was unable to perfectly reconstruct R_1, it successfully identified regions close to the states with non-zero rewards.

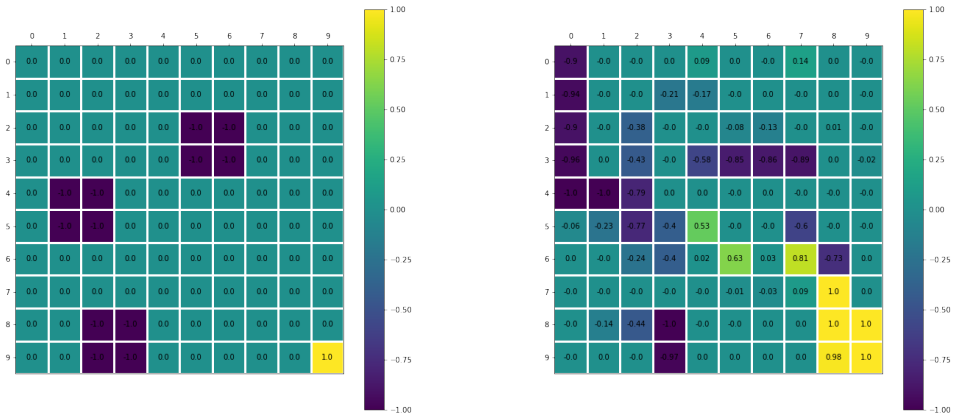


Figure 11: Heatmaps for ground truth R_1 vs. extracted reward

Question 14: Use the extracted reward function computed in question 13, to compute the optimal values of the states in the 2-D grid. For computing the optimal values you need to use the optimal state-value function that you wrote in question 2. For visualization purpose, generate a heat map of the optimal state values across the 2-D grid (similar to the figure generated in question 3). In this question, you should have 1 plot.

Answer: A heatmap of the optimal state-value function is provided in figure 12.

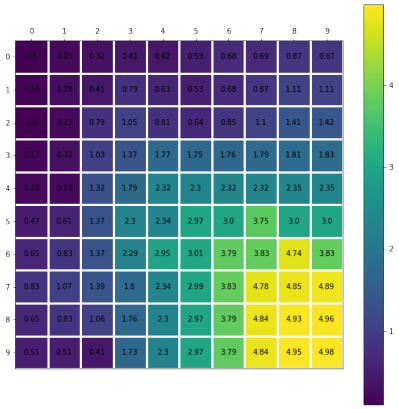


Figure 12: A heatmap of the optimal state-value function for the extracted reward.

Question 15: Compare the heat maps of Question 3 and Question 14 and provide a brief explanation on their similarities and differences.

Answer: In figure 13, we see the value function heatmaps for the ground truth, as well as IRL-learned reward. Both heatmaps appear to have have a similar gradient towards the bottom right corner. However, our learned value function appears to have higher values as you get further from the bottom right corner than the expert value function. In general however, the learned value function has a very similar shape to the true value function we derived in question 3.

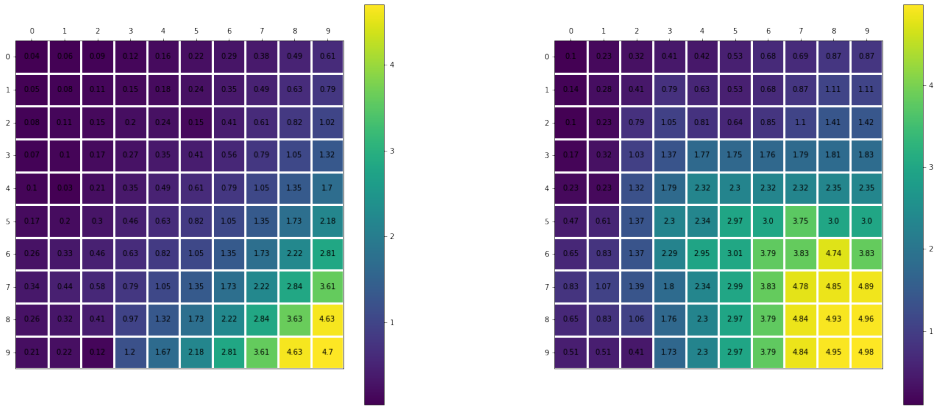


Figure 13: Heatmap for ground truth value function of R_1 vs the value function learned using IRL algorithm.

Question 16: Use the extracted reward function found in question 13 to compute the optimal policy of the agent. For computing the optimal policy of the agent you need to use the function that you wrote in question 5. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The actions should be displayed using arrows. In this question, you should have 1 plot.

Answer: The optimal policy for extracted value function is provided in figure 14.

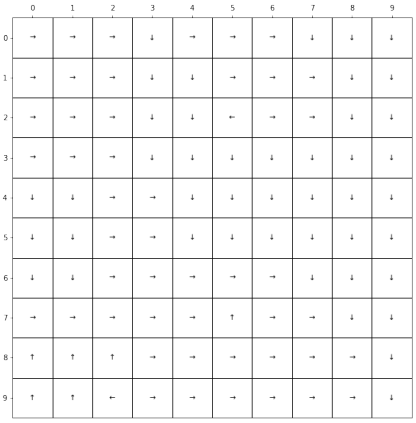


Figure 14: Optimal policy for the extracted reward.

Question 17: Compare the figures of Question 5 and Question 16 and provide a brief explanation on their similarities and differences.

Answer: In figure 15 we see the expert policy on the right vs irl-learned policy. For ease of comparison, state with non-zero reward are marked with grey color. Given our 0.95 accuracy we know there are 5 states where our policies differ. Along the top row

we see there are two arrows from the learned policy that point down when they should be pointing sideways. Row 6, col 1 points down instead of right, row 7, col 5 point up instead of right, and row 4, col 1 points down instead of up. These slight changes can be attributed to the very small variations of the learned value function. In 13 if we look at the instances where our policy varies from the expert policy, we see there are minute differences in the value function of those states compared to their surrounding states that cause our learned policy to have these variations. For the discussion on why value function did not allowed perfectly reconstruct ground truth policy refer to question 25.

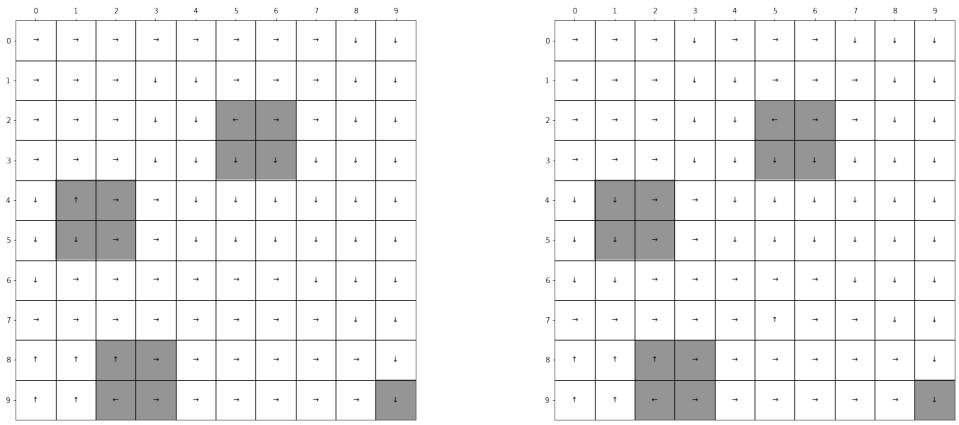


Figure 15: Optimal policies for ground truth reward R_1 (left) vs the reward learned using IRL algorithm (right).

Question 18: Sweep λ from 0 to 5 to get 500 evenly spaced values for λ. For each value of λ compute $O_A(s)$ by following the process described above. For this problem, use the optimal policy of the agent found in question 9 to fill in the $O_E(s)$ values. Then use equation 3 to compute the accuracy of the IRL algorithm for this value of λ. You need to repeat the above process for all 500 values of λ to get 500 data points. Plot λ (x-axis) against Accuracy (y-axis). In this question, you should have 1 plot.

Answer: The accuracy of IRL algorithm on optimal policy found in question 9 for different values of λ is provided in figure 16.

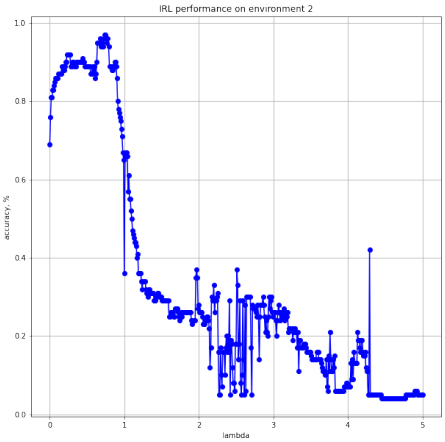


Figure 16: Inverse reinforcement learning performance on R_1

Question 19: Use the plot in question 18 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as $\lambda^{(2)}_{max}$. Please report $\lambda^{(2)}_{max}$

Answer: $\lambda^{(2)}_{max} = 0.74$

Question 20: For $\lambda^{(2)}_{max}$, generate heat maps of the ground truth reward and the extracted reward. Please note that the ground truth reward is the Reward function 2 and the extracted reward is computed by solving the linear program given by equation 2 with the λ parameter set to $\lambda^{(2)}_{max}$. In this question, you should have 2 plots.

Answer: Heatmaps of the ground truth reward and the reward reconstructed sing IRL algorithm are provided in figure 17. From that figure we can see that for some states, the reconstructed distribution of the reward can differ substantially from the ground truth. However, IRL successfully assigns high reward to the paths considered optimal by the original RL agent, which is sufficient for reconstructing behavior. Regions with negative R are also found with decent accuracy.

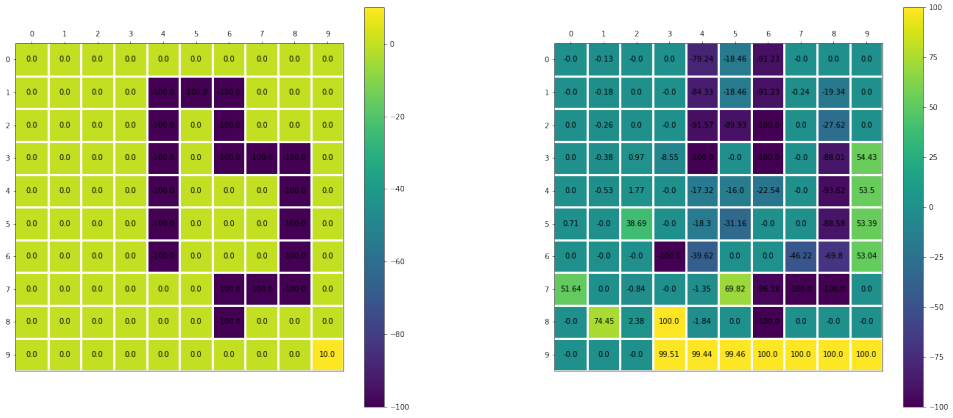


Figure 17: Heatmap for ground truth R_2 vs. extracted reward

Question 21: Use the extracted reward function computed in question 20, to compute the optimal values of the states in the 2-D grid. For computing the optimal values you need to use the optimal state-value function that you wrote in question 2. For visualization purpose, generate a heat map of the optimal state values across the 2-D grid (similar to the figure generated in question 7). In this question, you should have 1 plot.

Answer: A heatmap of the optimal state-value function is provided in figure 18.

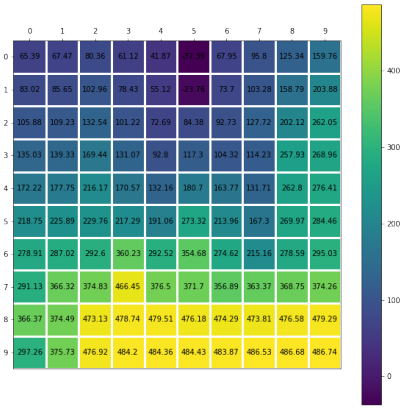


Figure 18: A heatmap of the optimal state-value function for the extracted reward.

Question 22: Compare the heat maps of Question 7 and Question 21 and provide a brief explanation on their similarities and differences.

Answer: In figure 19, we see the value function heatmaps for the ground truth, as well as IRL-learned reward. While both state-value functions increase towards bottom-left corner, in the reconstructed value function increase over x-axis is much steeper. The absolute value of states for IRL case is also much higher, which is explained by the fact that constraints on the absolute value of the positive and negative rewards are the same, so the maximal allowed reward is +100 opposed to ground truth +10. In general however, the increase in reconstructed value function matches the path of the optimal agent found in question 8.

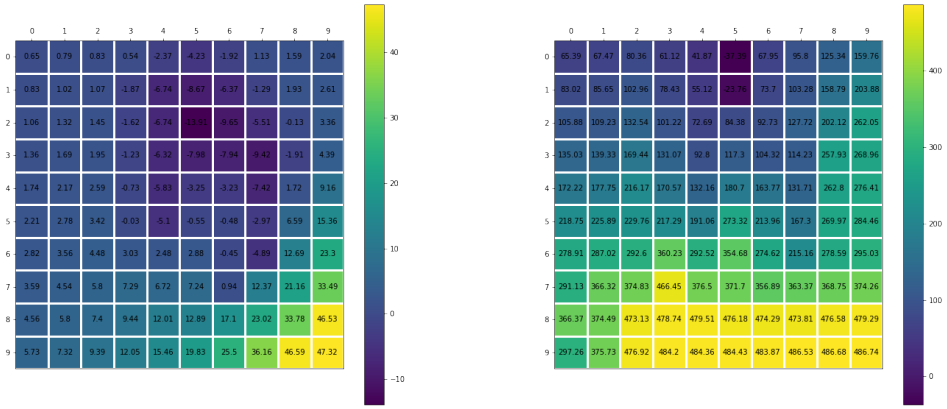


Figure 19: Heatmap for ground truth value of R_2 vs the value function learned using IRL algorithm.

Question 23: Use the extracted reward function found in question 20 to compute the optimal policy of the agent. For computing the optimal policy of the agent you need to use the function that you wrote in question 9. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The actions should be displayed using arrows. In this question, you should have 1 plot.

Answer: The optimal policy for the extracted value function is provided in figure 20.

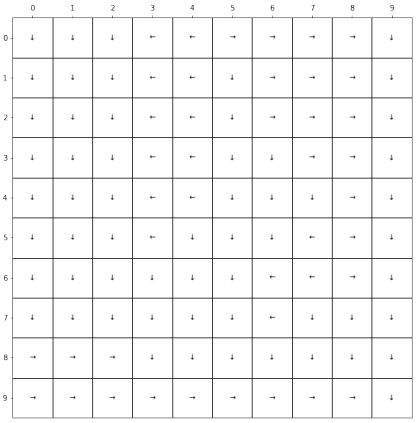


Figure 20: Optimal policy for the extracted reward.

Question 24: Compare the figures of Question 9 and Question 23 and provide a brief explanation on their similarities and differences.

Answer: In figure 21 we see the expert policy on the right vs irl-learned policy. For ease of comparison, the states with non-zero reward are marked with grey color. Given our 0.97 accuracy, there are only 3 states for which expert and ground truth policies

differ. These states are s_{45}, s_{51}, s_{73}. As in question 17, these mismatches are due to variations in reconstructed value function. For the discussion on why value function did not allowed perfectly reconstruct ground truth policy refer to question 25.

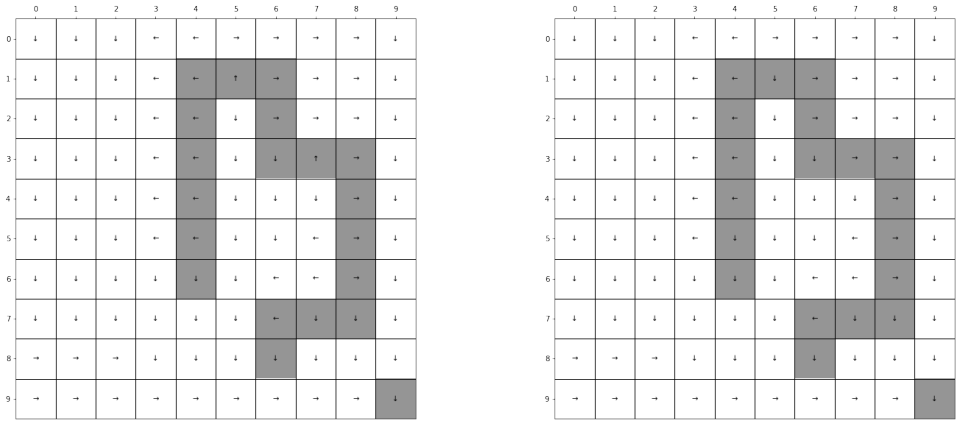


Figure 21: Optimal policies for ground truth reward R_2 (left) vs the reward learned using IRL algorithm (right).

Question 25: From the figure in question 23, you should observe that the optimal policy of the agent has two major discrepancies. Please identify and provide the causes for these two discrepancies. One of the discrepancy can be fixed easily by a slight modification to the value iteration algorithm. Perform this modification and re-run the modified value iteration algorithm to compute the optimal policy of the agent. Also, recompute the maximum accuracy after this modification. Is there a change in maximum accuracy? The second discrepancy is harder to fix and is a limitation of the simple IRL algorithm.

Answer: The discrepancies we observe can be divided into harmless (s_{51} in reconstructed environment R_2 and s_{16}, s_{30}, s_{57}, s_{70} in reconstructed environment R_1) and those that lead to major loss of expected reward (s_{45}, s_{73} in reconstructed environment R_2 and s_{14} in reconstructed environment R_2, which point agent to the "obstacles").

There are 2 reasons for existence of these problems. The first reason is that we do not always use IRL-extracted reward to its full potential and stop value iteration before complete convergence. The other reason is that by its construction Inverse Reinforcement Learning suffers from "degenerate solutions" problem. Since a single policy may be simultaneously optimal for many reward functions, unique reconstruction of the original reward is impossible, which might affect reconstructed policy in future.

To improve the IRL performance, we set $\varepsilon = 0$ and run the algorithm for different values of $\lambda \in [0,\ 500]$. We found that for both ground truth rewards R_1 and R_2 waiting until full convergence of learned state-value function helps perfectly (with 100% accuracy) reconstruct original policy (see figure 22). Optimal hyperparameters λ that provide the best performance are listed below:

$$\lambda^{(1)}_{max,\ adjusted} = 0.8$$

$$\lambda^{(2)}_{max,\, adjusted} = 0.8$$

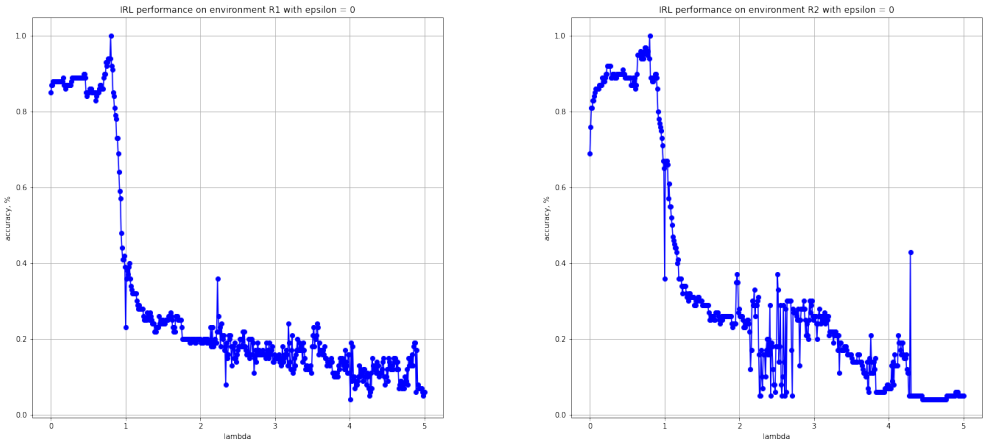


Figure 22: Inverse Reinforcement Learning performance recomputed for $\varepsilon = 0$ for π_1^* (left) and π_2^* (right).

Project 3-final

May 22, 2021

1 Import libs

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

2 Create MDP

[2]:
```
# set environment and actions
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
```

[3]:
```
def transition_probabilities(w):
    # creates a dictionary with key (action, state_x, state_y), which returns␣
↪matrix with probabilities
    # init probabilities
    probs = {}
    for x in range(10):
        for y in range(10):
            probs[('top', x, y)], probs[('bottom', x, y)], probs[('right', x,␣
↪y)], probs[('left', x, y)] = \
                np.zeros((10, 10)), np.zeros((10, 10)), np.zeros((10, 10)), np.
↪zeros((10, 10))
            # prcoess top-left angle
            if x == 0 and y == 0:
                # action == top
                probs[('top', x, y)][y, x], probs[('top', x, y)][y, x+1],␣
↪probs[('top', x, y)][y+1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y, x], probs[('bottom', x, y)][y, x+1],␣
↪probs[('bottom', x, y)][y+1, x] = \
                    w * 2/4, w * 1/4, 1 - w * 3/4
                # action == left
```

```
                probs[('left', x, y)][y, x], probs[('left', x, y)][y, x+1],␣
↪probs[('left', x, y)][y+1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x], probs[('right', x, y)][y, x+1],␣
↪probs[('right', x, y)][y+1, x] = \
                    w * 2/4, 1 - w * 3/4, w * 1/4
            # process top-right angle
            elif x == 9 and y == 0:
                # action == top
                probs[('top', x, y)][y, x], probs[('top', x, y)][y, x-1],␣
↪probs[('top', x, y)][y+1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y, x], probs[('bottom', x, y)][y, x-1],␣
↪probs[('bottom', x, y)][y+1, x] = \
                    w * 2/4, w * 1/4, 1 - w * 3/4
                # action == left
                probs[('left', x, y)][y, x], probs[('left', x, y)][y, x-1],␣
↪probs[('left', x, y)][y+1, x] = \
                    w * 2/4, 1 - w * 3/4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x], probs[('right', x, y)][y, x-1],␣
↪probs[('right', x, y)][y+1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
            # process bottom-right angle
            elif x == 9 and y == 9:
                # action == top
                probs[('top', x, y)][y, x], probs[('top', x, y)][y, x-1],␣
↪probs[('top', x, y)][y-1, x] = \
                    w * 2/4, w * 1/4, 1 - w * 3/4
                # action == bottom
                probs[('bottom', x, y)][y, x], probs[('bottom', x, y)][y, x-1],␣
↪probs[('bottom', x, y)][y-1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == left
                probs[('left', x, y)][y, x], probs[('left', x, y)][y, x-1],␣
↪probs[('left', x, y)][y-1, x] = \
                    w * 2/4, 1 - w * 3/4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x], probs[('right', x, y)][y, x-1],␣
↪probs[('right', x, y)][y-1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
            # process bottom-left angle
            elif x == 0 and y == 9:
                # action == top
```

```
                probs[('top', x, y)][y, x], probs[('top', x, y)][y, x+1],␣
↪probs[('top', x, y)][y-1, x] = \
                    w * 2/4, w * 1/4, 1 - w * 3/4
                # action == bottom
                probs[('bottom', x, y)][y, x], probs[('bottom', x, y)][y, x+1],␣
↪probs[('bottom', x, y)][y-1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == left
                probs[('left', x, y)][y, x], probs[('left', x, y)][y, x+1],␣
↪probs[('left', x, y)][y-1, x] = \
                    1 - w * 2/4, w * 1/4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x], probs[('right', x, y)][y, x+1],␣
↪probs[('right', x, y)][y-1, x] = \
                    w * 2/4, 1 - w * 3/4, w * 1/4
            # process top row
            elif y == 0:
                # action == top
                probs[('top', x, y)][y, x-1], probs[('top', x, y)][y, x],␣
↪probs[('top', x, y)][y, x+1], \
                    probs[('top', x, y)][y+1, x] = w * 1/4, 1 - w * 3/4, w * 1/
↪4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y, x-1], probs[('bottom', x, y)][y, x],␣
↪probs[('bottom', x, y)][y, x+1], \
                    probs[('bottom', x, y)][y+1, x] = w * 1/4, w * 1/4, w * 1/
↪4, 1 - w * 3/4
                # action == left
                probs[('left', x, y)][y, x-1], probs[('left', x, y)][y, x],␣
↪probs[('left', x, y)][y, x+1], \
                    probs[('left', x, y)][y+1, x] = 1 - w * 3/4, w * 1/4, w * 1/
↪4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x-1], probs[('right', x, y)][y, x],␣
↪probs[('right', x, y)][y, x+1], \
                    probs[('right', x, y)][y+1, x] = w * 1/4, w * 1/4, 1 - w *␣
↪3/4, w * 1/4
            # process bottom row
            elif y == 9:
                # action == top
                probs[('top', x, y)][y, x-1], probs[('top', x, y)][y, x],␣
↪probs[('top', x, y)][y, x+1], \
                    probs[('top', x, y)][y-1, x] = w * 1/4, w * 1/4, w * 1/4, 1␣
↪- w * 3/4
                # action == bottom
```

```
                probs[('bottom', x, y)][y, x-1], probs[('bottom', x, y)][y, x],␣
↪probs[('bottom', x, y)][y, x+1], \
                    probs[('bottom', x, y)][y-1, x] = w * 1/4, 1 - w * 3/4, w *␣
↪1/4, w * 1/4
                # action == left
                probs[('left', x, y)][y, x-1], probs[('left', x, y)][y, x],␣
↪probs[('left', x, y)][y, x+1], \
                    probs[('left', x, y)][y-1, x] = 1 - w * 3/4, w * 1/4, w * 1/
↪4, w * 1/4
                # action == right
                probs[('right', x, y)][y, x-1], probs[('right', x, y)][y, x],␣
↪probs[('right', x, y)][y, x+1], \
                    probs[('right', x, y)][y-1, x] = w * 1/4, w * 1/4, 1 - w *␣
↪3/4, w * 1/4
            # process left row
            elif x == 0:
                # action == top
                probs[('top', x, y)][y-1, x], probs[('top', x, y)][y, x],␣
↪probs[('top', x, y)][y+1, x], \
                    probs[('top', x, y)][y, x+1] = 1 - w * 3/4, w * 1/4, w * 1/
↪4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y-1, x], probs[('bottom', x, y)][y, x],␣
↪probs[('bottom', x, y)][y+1, x], \
                    probs[('bottom', x, y)][y, x+1] = w * 1/4, w * 1/4, 1 - w *␣
↪3/4, w * 1/4
                # action == left
                probs[('left', x, y)][y-1, x], probs[('left', x, y)][y, x],␣
↪probs[('left', x, y)][y+1, x], \
                    probs[('left', x, y)][y, x+1] = w * 1/4, 1 - w * 3/4, w * 1/
↪4, w * 1/4
                # action == right
                probs[('right', x, y)][y-1, x], probs[('right', x, y)][y, x],␣
↪probs[('right', x, y)][y+1, x], \
                    probs[('right', x, y)][y, x+1] = w * 1/4, w * 1/4, w * 1/4,␣
↪1 - w * 3/4
            # process right row
            elif x == 9:
                # action == top
                probs[('top', x, y)][y-1, x], probs[('top', x, y)][y, x],␣
↪probs[('top', x, y)][y+1, x], \
                    probs[('top', x, y)][y, x-1] = 1 - w * 3/4, w * 1/4, w * 1/
↪4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y-1, x], probs[('bottom', x, y)][y, x],␣
↪probs[('bottom', x, y)][y+1, x], \
```

```
                    probs[('bottom', x, y)][y, x-1] = w * 1/4, w * 1/4, 1 - w * 3/4, w * 1/4
                # action == left
                probs[('left', x, y)][y-1, x], probs[('left', x, y)][y, x], probs[('left', x, y)][y+1, x], \
                    probs[('left', x, y)][y, x-1] = w * 1/4, w * 1/4, w * 1/4, 1 - w * 3/4
                # action == right
                probs[('right', x, y)][y-1, x], probs[('right', x, y)][y, x], probs[('right', x, y)][y+1, x], \
                    probs[('right', x, y)][y, x-1] = w * 1/4, 1 - w * 3/4, w * 1/4, w * 1/4
            else:
                # action == top
                probs[('top', x, y)][y-1, x], probs[('top', x, y)][y+1, x], \
                probs[('top', x, y)][y, x-1], probs[('top', x, y)][y, x+1] = 1 - w * 3/4, w * 1/4, w * 1/4, w * 1/4
                # action == bottom
                probs[('bottom', x, y)][y-1, x], probs[('bottom', x, y)][y+1, x], \
                probs[('bottom', x, y)][y, x-1], probs[('bottom', x, y)][y, x+1] = w * 1/4, 1 - w * 3/4, w * 1/4, w * 1/4
                # action == left
                probs[('left', x, y)][y-1, x], probs[('left', x, y)][y+1, x], \
                probs[('left', x, y)][y, x-1], probs[('left', x, y)][y, x+1] = w * 1/4, w * 1/4, 1 - w * 3/4, w * 1/4
                # action == right
                probs[('right', x, y)][y-1, x], probs[('right', x, y)][y+1, x], \
                probs[('right', x, y)][y, x-1], probs[('right', x, y)][y, x+1] = w * 1/4, w * 1/4, w * 1/4, 1 - w * 3/4

    return probs
```

[4]:
```
# set wind probability
w = 0.1
# set transition probabiities
P = transition_probabilities(w)
```

3 Run tests and visually inspect border cases

[5]:
```
def run_tests(A, transition_probs):
    # this function checks if transition probabilities sum to 1
    check = True
    for action in A:
```

```
        for x in range(10):
            for y in range(10):
                if np.sum(transition_probs[(action, x, y)]) != 1:
                    print("error in matrix (", action, x, y, ")")
                    check = False
    if check:
        print("it's all good!")

def test_border_case(x, y, transition_probs):
    # allows to visually inspect border cases
    print("top")
    print(transition_probs[("top", x, y)])
    print("bottom")
    print(transition_probs[("bottom", x, y)])
    print("left")
    print(transition_probs[("left", x, y)])
    print("right")
    print(transition_probs[("right", x, y)])
```

[6]:
```
# run tests to see if transition probabilities pass sanity check
run_tests(A, P)
```

```
it's all good!
```

[7]:
```
# before proceeding, check all cases:
test_border_case(0, 8, P)
```

```
top
[[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.    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.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.925 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.025 0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]]
bottom
[[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.    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.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
```

```
 [0.025 0.025 0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.925 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]]
left
[[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.    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.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.925 0.025 0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]]
right
[[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.    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.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.925 0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.025 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]]
```

4 Reward functions

[8]:
```
def create_reward(scores):
    # creates reward matrix from an array of type (state_x, state_y, reward)
    rewards = np.zeros((10, 10))
    for (x, y, reward) in scores:
        rewards[y, x] = reward

    return rewards
```

[98]:
```
R1 = create_reward([(1, 4, -1), (1, 5, -1), (2, 4, -1), (2, 5, -1),
                    (2, 8, -1), (2, 9, -1), (3, 8, -1), (3, 9, -1),
                    (5, 2, -1), (5, 3, -1), (6, 2, -1), (6, 3, -1),
                    (9, 9, 1)])

R2 = create_reward([(4, 1, -100), (4, 2, -100), (4, 3, -100), (4, 4, -100), (4,␣
 ↪5, -100), (4, 6, -100),
                    (5, 1, -100),
                    (6, 1, -100), (6, 2, -100), (6, 3, -100), (6, 7, -100), (6,␣
 ↪8, -100),
                    (7, 3, -100), (7, 7, -100),
```

```
                    (8, 3, -100), (8, 4, -100), (8, 5, -100), (8, 6, -100), (8,␣
↪7, -100),
                    (9, 9, 10)])
```

5 Create environments

[10]:
```
class EnvironmentRL:
    # an MDP is defined by S, A, P, R, gamma
    def __init__(self, S, A, P, R, gamma):
        self.S = S
        self.A = A
        self.P = P
        self.R = R
        self.gamma = gamma
```

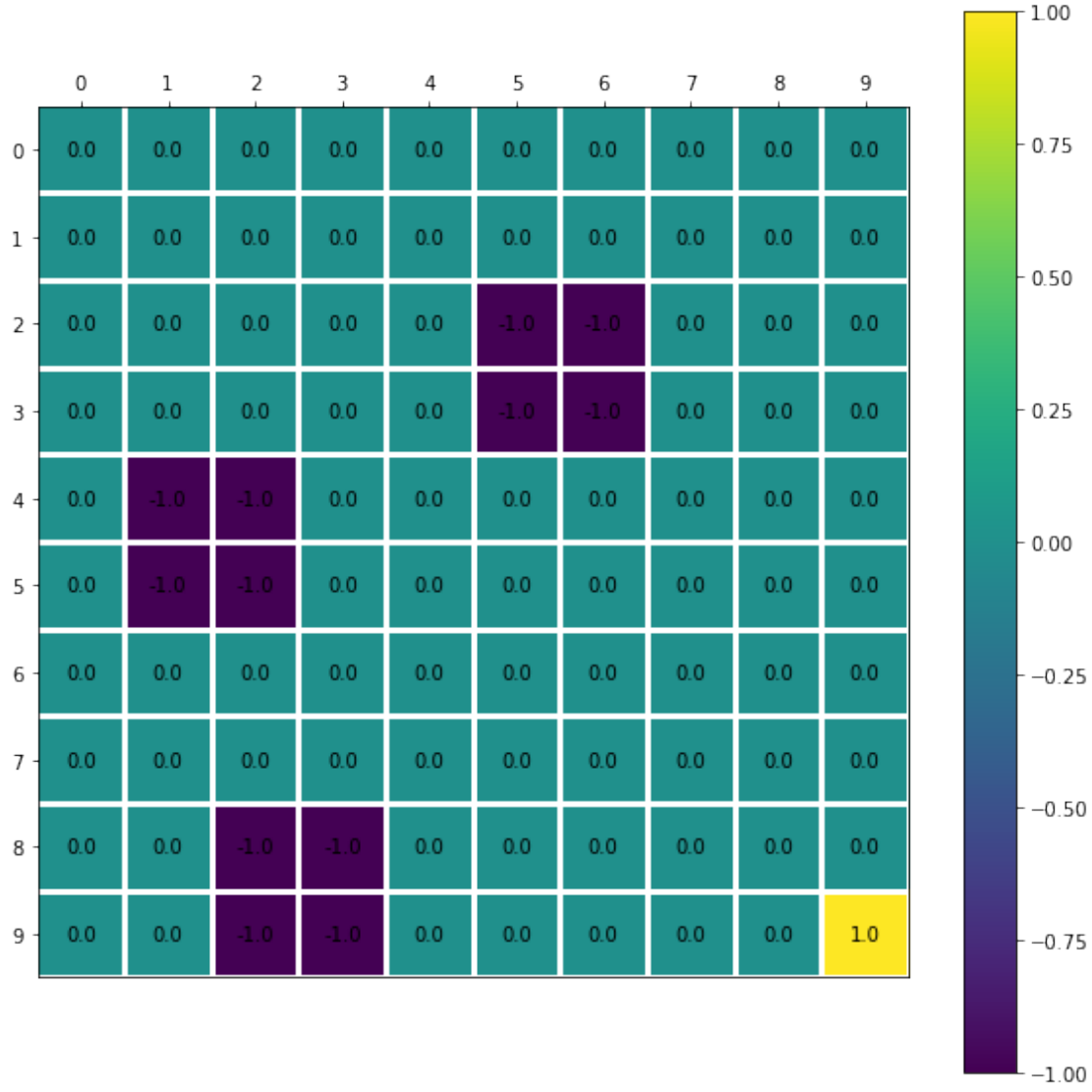
[11]:
```
# set up discount factor
gamma = 0.8
# Create environments
env1 = EnvironmentRL(S, A, P, R1, gamma)
# Create environments
env2 = EnvironmentRL(S, A, P, R2, gamma)
```

6 Question 1

For visualization purpose, generate heat maps of Reward function 1 and Reward function 2. For the heat maps, make sure you display the coloring scale. You will have 2 plots for this question

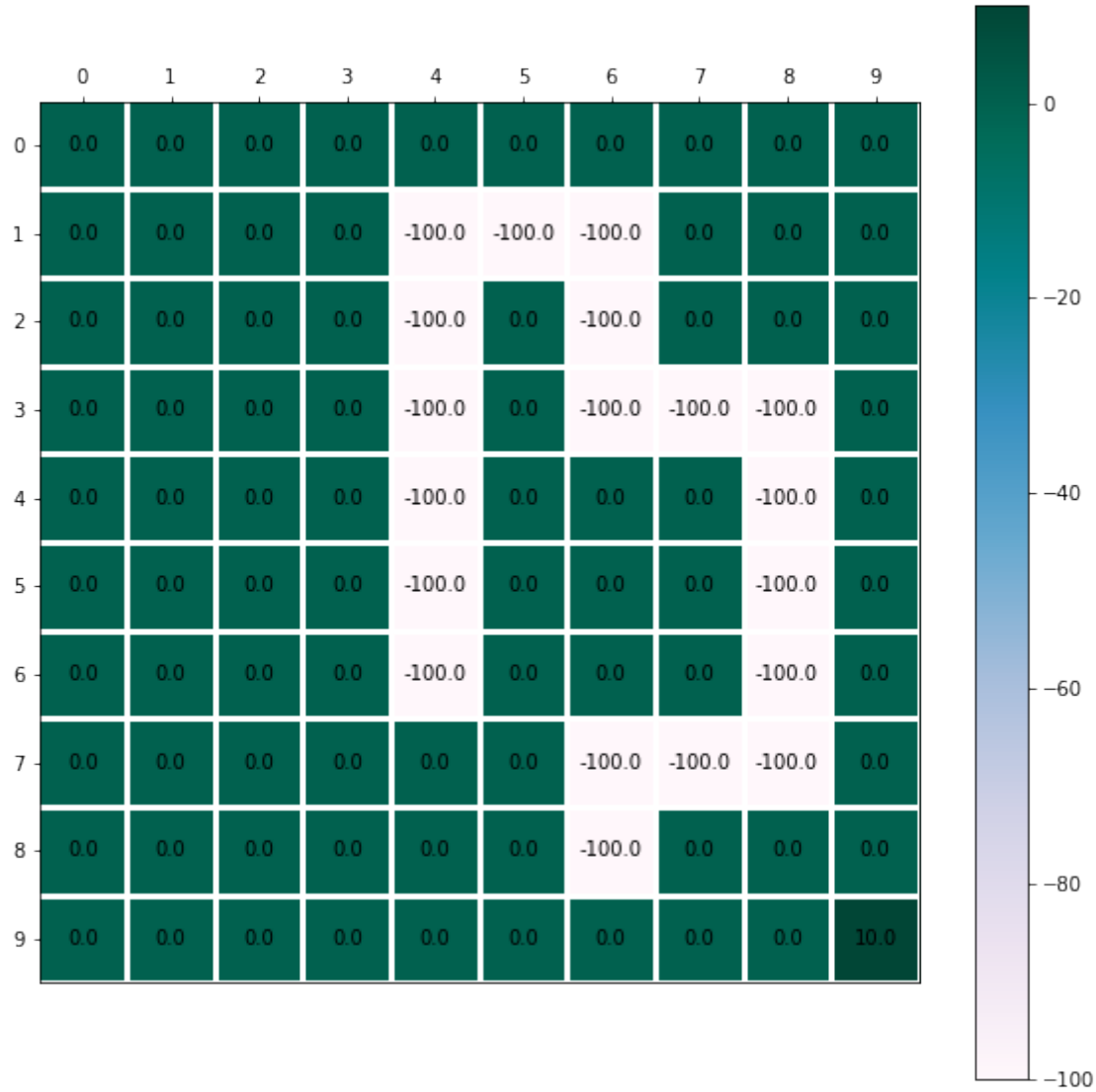
[73]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(R1)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, R1[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)
```

```
ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q1-r1.png')
```



[74]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(R2, cmap = 'PuBuGn')

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)
```

```
# # cmaps['Sequential'] = [
#             'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',
#             'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',
#             'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']
# PuBuGn

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, R2[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q1-r2.png')
```



7 Question 2

Create the environment of the agent using the information provided in section 2. To be specific, create the MDP by setting up the state-space, action set, transition probabilities, discount factor, and reward function. For creating the environment, use the following set of parameters:

- `Number of states = 100 (state space is a 10 by 10 square grid as displayed in figure 1)`
- `Number of actions = 4 (set of possible actions is displayed in figure 2)`
- `w = 0.1`
- `Discount factor = 0.8`
- `Reward function 1`

After you have created the environment, then write an optimal state-value function that takes as

input the environment of the agent and outputs the optimal value of each state in the grid. For the optimal state-value function, you have to implement the Initialization (lines 2-4) and Estimation (lines 5-13) steps of the Value Iteration algorithm. For the estimation step, use $\varepsilon = 0.01$. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal value of that state. In this part of question, you should have 1 plot. Let's assume that your value iteration algorithm converges in N steps. Plot snapshots of state values in 5 different steps linearly distributed from 1 to N. Report N and your step numbers. What observations do you have from the plots?

[15]:
```
def value_iteration(environment, epsilon, to_record = [], print_steps = False):
    # initialization
    V = np.zeros((environment.S.shape[0], environment.S.shape[1]))
    # set up delta in a way that will allow to start estimation cycle
    Delta = np.inf
    # record iteration
    current_step = 0
    # init an array of records
    recorded_v = []
    # estimation step
    while Delta > epsilon:
        # print current step
        if print_steps == True:
            print("Step", current_step)
        # reset epsilon
        Delta = 0
        # if we need to record state, do it
        if current_step in to_record:
            recorded_v.append(V.copy())
        # for each state
        for x in range(environment.S.shape[1]):
            for y in range(environment.S.shape[0]):
                # record previous value of V to check how much did it change
                v_small = V[y, x]
                # do a step of value iteration; calculate the expected value of␣
↪each action
                v_top = np.sum(environment.P[("top", x, y)] * (environment.R +␣
↪environment.gamma * V))
                v_bottom = np.sum(environment.P[("bottom", x, y)] *␣
↪(environment.R + environment.gamma * V))
                v_left = np.sum(environment.P[("left", x, y)] * (environment.R␣
↪+ environment.gamma * V))
                v_right = np.sum(environment.P[("right", x, y)] * (environment.
↪R + environment.gamma * V))
                # choose the maximal value
                V[y, x] = max(v_top, v_bottom, v_left, v_right)
                # recalculate value of an update
                Delta = max(Delta, abs(v_small - V[y, x]))
```

```
        # record step
        current_step += 1

    return V, recorded_v
```

[16]:
```
# set precision
eps = 0.01

# do value iteration for environment 1
V1, rec1 = value_iteration(env1, eps, to_record=[3, 7, 10, 13, 16], print_steps
 ↪= True)
```

```
Step 0
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Step 7
Step 8
Step 9
Step 10
Step 11
Step 12
Step 13
Step 14
Step 15
Step 16
Step 17
Step 18
Step 19
Step 20
```

[75]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
```

```
    for j in range(10):
        text = ax.text(j, i, round(V1[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q2-v1.png')
```

	0	1	2	3	4	5	6	7	8	9
0	0.04	0.06	0.09	0.12	0.16	0.22	0.29	0.38	0.49	0.61
1	0.05	0.08	0.11	0.15	0.18	0.24	0.35	0.49	0.63	0.79
2	0.08	0.11	0.15	0.2	0.24	0.15	0.41	0.61	0.82	1.02
3	0.07	0.1	0.17	0.27	0.35	0.41	0.56	0.79	1.05	1.32
4	0.1	0.03	0.21	0.35	0.49	0.61	0.79	1.05	1.35	1.7
5	0.17	0.2	0.3	0.46	0.63	0.82	1.05	1.35	1.73	2.18
6	0.26	0.33	0.46	0.63	0.82	1.05	1.35	1.73	2.22	2.81
7	0.34	0.44	0.58	0.79	1.05	1.35	1.73	2.22	2.84	3.61
8	0.26	0.32	0.41	0.97	1.32	1.73	2.22	2.84	3.63	4.63
9	0.21	0.22	0.12	1.2	1.67	2.18	2.81	3.61	4.63	4.7

[83]:
```
current_plot = 4

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(rec1[current_plot][i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q2-v1-' + str(current_plot) + '.png')
```

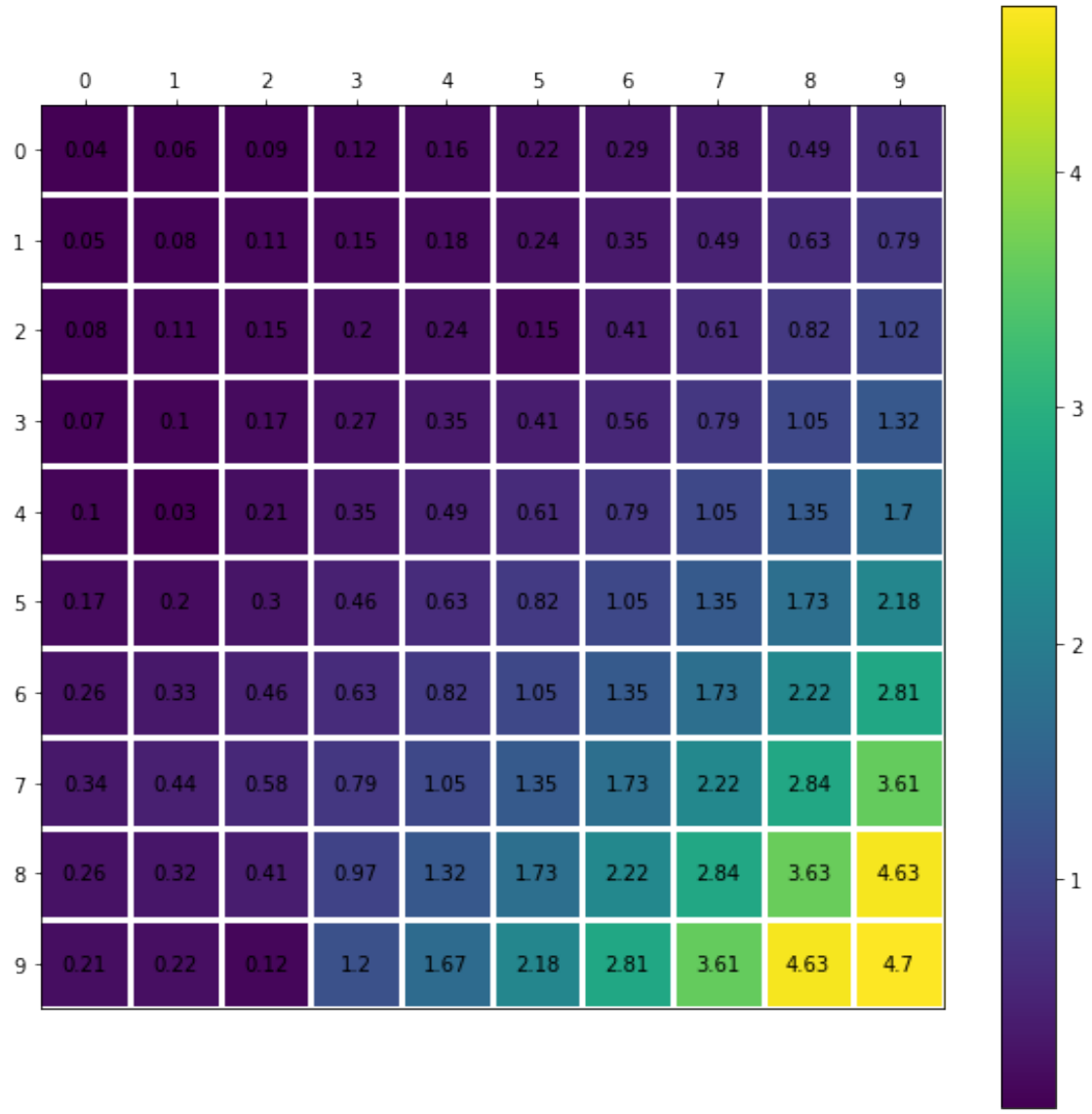
	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	0.01	0.04	0.09	0.14	0.21	0.3	0.41	0.53
1	-0.0	0.01	0.03	0.07	0.1	0.16	0.27	0.41	0.55	0.71
2	0.01	0.03	0.07	0.12	0.16	0.08	0.34	0.53	0.74	0.94
3	0.0	0.02	0.1	0.19	0.27	0.34	0.48	0.71	0.97	1.24
4	0.02	-0.03	0.13	0.27	0.41	0.53	0.71	0.98	1.27	1.62
5	0.09	0.12	0.23	0.39	0.56	0.74	0.98	1.28	1.66	2.1
6	0.18	0.25	0.38	0.56	0.74	0.98	1.28	1.66	2.14	2.73
7	0.26	0.36	0.5	0.71	0.98	1.28	1.66	2.14	2.76	3.53
8	0.2	0.26	0.35	0.89	1.25	1.65	2.14	2.76	3.55	4.56
9	0.16	0.17	0.08	1.12	1.59	2.1	2.73	3.53	4.56	4.62

8 Question 3

Generate a heat map of the optimal state values across the 2-D grid. For generating the heat map, you can use the same function provided in the hint earlier (see the hint after question 1).

[77]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(V1)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)
```

```
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(V1[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q3-v1.png')
```



9 Question 4

Question 4: (15 points) Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in question 3 to explain)

Basically, the closer to the bottom left the better. The further the lower because of gamma. Very low (or negative) in the obstacles on the map. Bottom left should be close to 5, as it is 1 / (1 - gamma)

10 Question 5

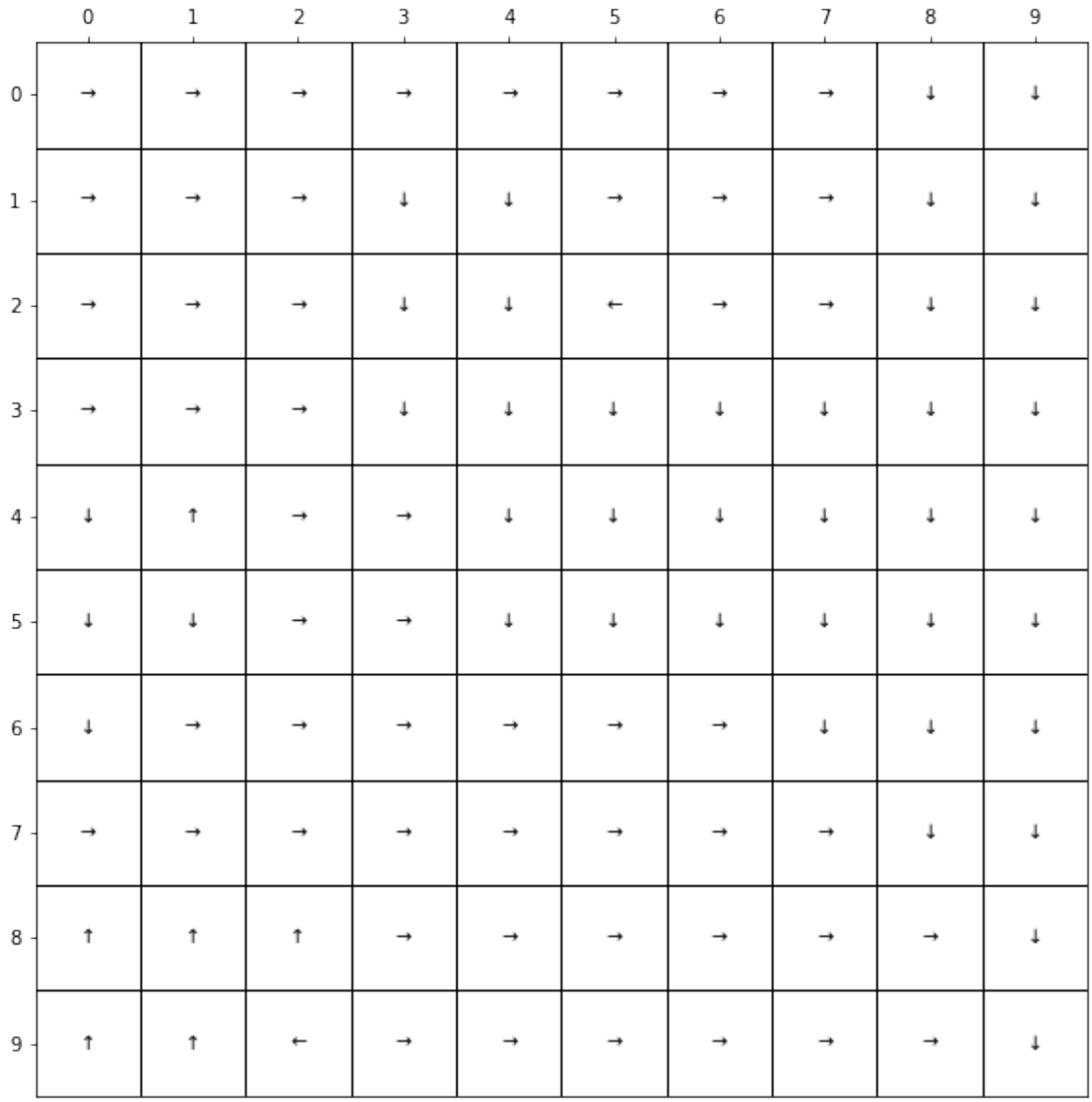
Implement the computation step of the value iteration algorithm (lines 14-17) to compute the optimal policy of the agent navigating the 2-D state-space. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The optimal actions should be displayed using arrows. Does the optimal policy of the agent match your intuition? Please provide a brief explanation. Is it possible for the agent to compute the optimal action to take at each state by observing the optimal values of it's neighboring states? In this question, you should have 1 plot.

[31]:
```
def computation(environment, V):
    # initialization
    pi = np.zeros((environment.S.shape[0], environment.S.shape[1]),␣
↪dtype=object)
    # for each state
    for x in range(environment.S.shape[1]):
        for y in range(environment.S.shape[0]):
            # do a step of value iteration; calculate the expected value of␣
↪each action
            v_top = np.sum(environment.P[("top", x, y)] * (environment.R +␣
↪environment.gamma * V))
            v_bottom = np.sum(environment.P[("bottom", x, y)] * (environment.R␣
↪+ environment.gamma * V))
            v_left = np.sum(environment.P[("left", x, y)] * (environment.R +␣
↪environment.gamma * V))
            v_right = np.sum(environment.P[("right", x, y)] * (environment.R +␣
↪environment.gamma * V))
            # init array of values
            action_values = [v_top, v_bottom, v_left, v_right]
            # select the best action
            pi[y, x] = environment.A[np.argmax(action_values)]

    return pi
```

[32]:
```
pi1 = computation(env1, V1)
```

[78]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
#  + 100 * np.abs(R1)
im = ax.imshow( -100 * np.ones((10, 10)), cmap="Greys", norm=colors.
↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)
```

```
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi1[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif pi1[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif pi1[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif pi1[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q5-p1.png')
```



We can see that the agent tries to avoid obstacles.

11 Question 6

Modify the environment of the agent by replacing Reward function 1 with Reward function 2. Use the optimal state-value function implemented in question 2 to compute the optimal value of each state in the grid. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal value of that state. In this question, you should have 1 plot.

[35]:
```
# set precision
eps = 0.01
```

```
# do value iteration for environment 1
V2, rec2 = value_iteration(env2, eps)
```

[79]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(V2[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q6-v2.png')
```

	0	1	2	3	4	5	6	7	8	9
0	0.65	0.79	0.83	0.54	-2.37	-4.23	-1.92	1.13	1.59	2.04
1	0.83	1.02	1.07	-1.87	-6.74	-8.67	-6.37	-1.29	1.93	2.61
2	1.06	1.32	1.45	-1.62	-6.74	-13.91	-9.65	-5.51	-0.13	3.36
3	1.36	1.69	1.95	-1.23	-6.32	-7.98	-7.94	-9.42	-1.91	4.39
4	1.74	2.17	2.59	-0.73	-5.83	-3.25	-3.23	-7.42	1.72	9.16
5	2.21	2.78	3.42	-0.03	-5.1	-0.55	-0.48	-2.97	6.59	15.36
6	2.82	3.56	4.48	3.03	2.48	2.88	-0.45	-4.89	12.69	23.3
7	3.59	4.54	5.8	7.29	6.72	7.24	0.94	12.37	21.16	33.49
8	4.56	5.8	7.4	9.44	12.01	12.89	17.1	23.02	33.78	46.53
9	5.73	7.32	9.39	12.05	15.46	19.83	25.5	36.16	46.59	47.32

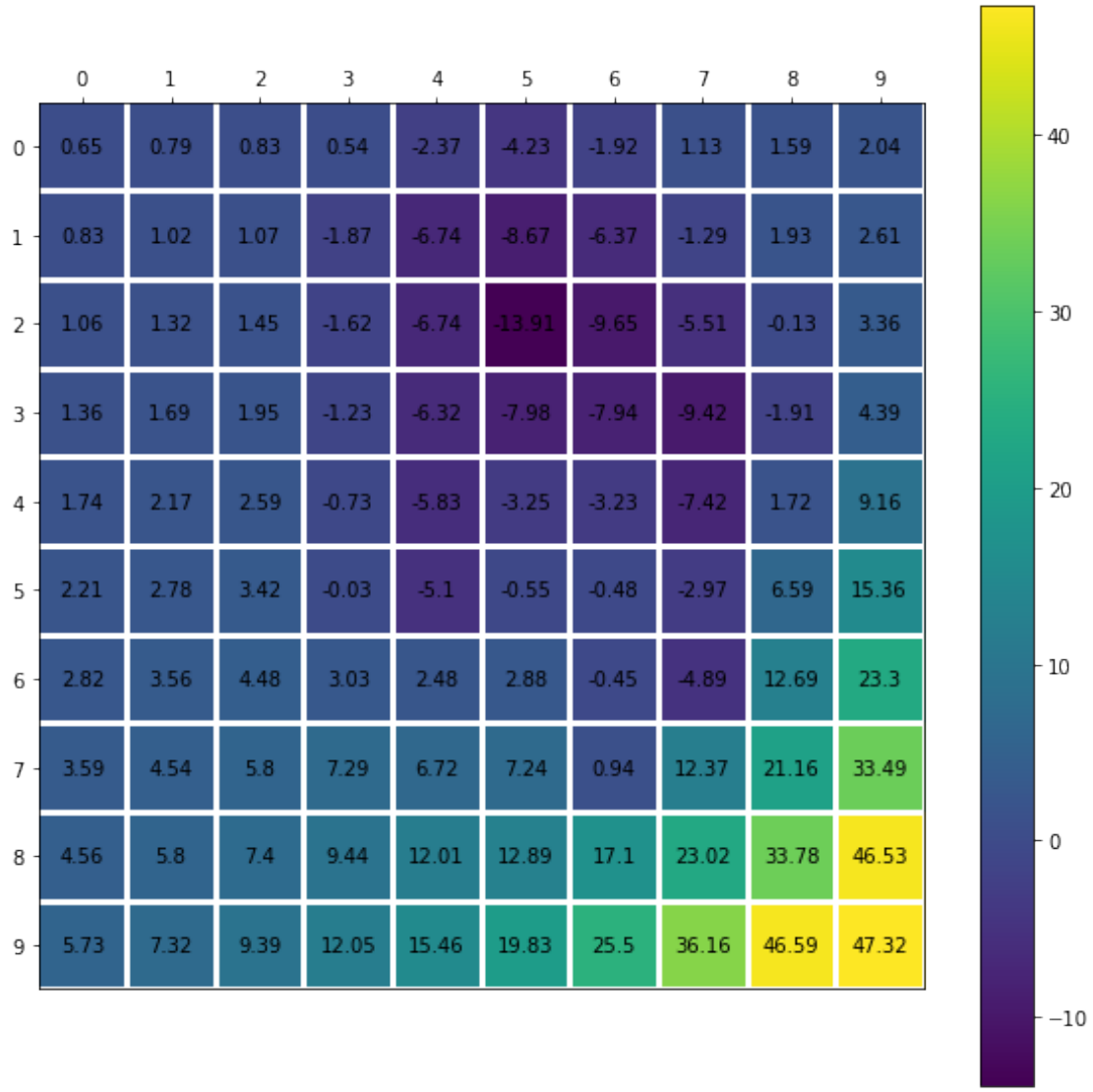
We can see that the values are poor at obstacles and good at the end.

12 Question 7

Generate a heat map of the optimal state values (found in question 6) across the 2-D grid. For generating the heat map, you can use the same function provided in the hint earlier. Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in this question to explain)

[84]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(V2)

ax.set_xticks(np.arange(10), minor=False)
```

```
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(V2[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q7-v2.png')
```

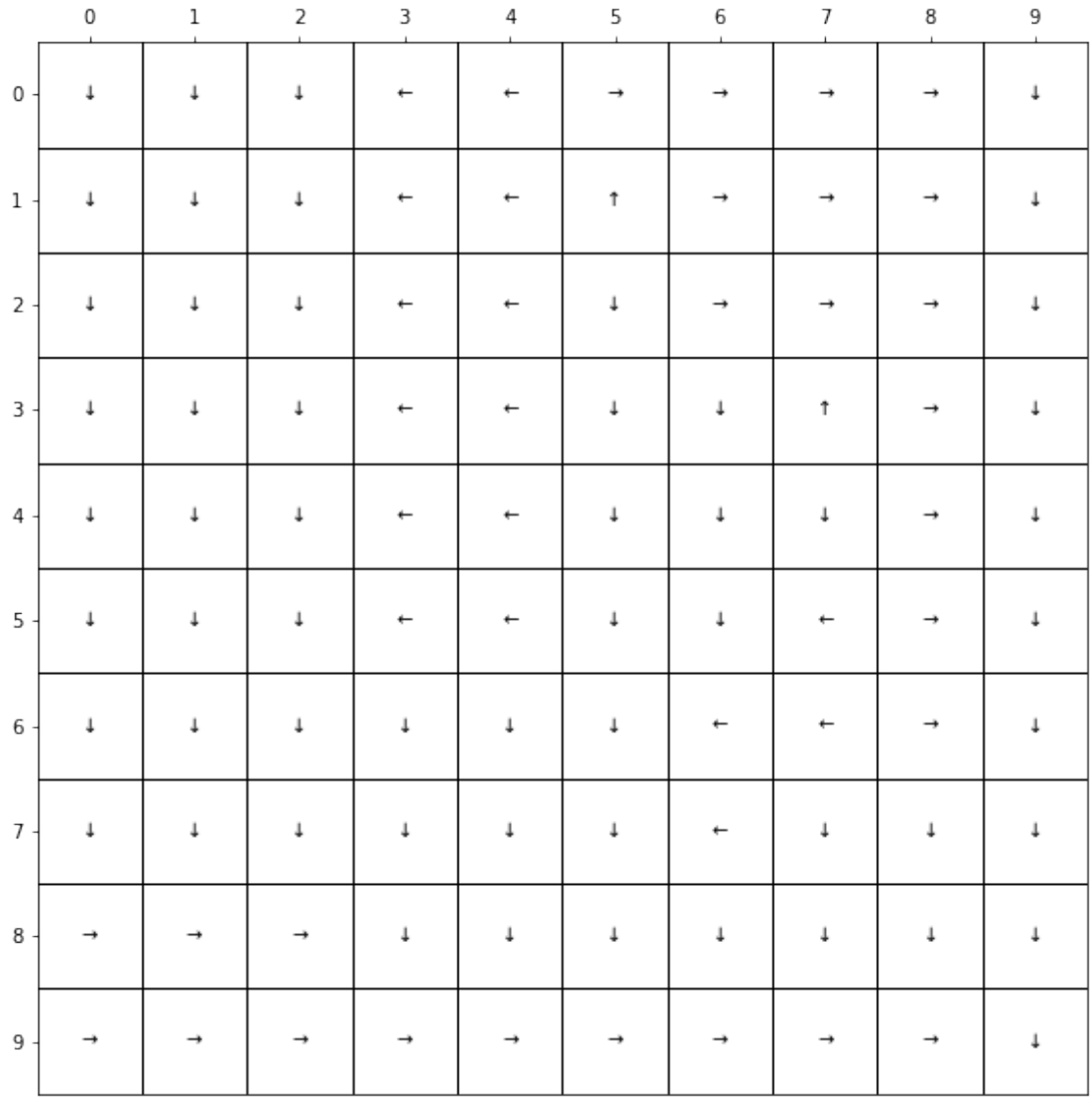


13 Question 8

Implement the computation step of the value iteration algorithm (lines 14-17) to compute the optimal policy of the agent navigating the 2-D state-space. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The optimal actions should be displayed using arrows. Does the optimal policy of the agent match your intuition? Please provide a brief explanation. In this question, you should have 1 plot.

[38]:
```
pi2 = computation(env2, V2)
```

[85]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
temp_r2 = R2.copy()
temp_r2[9, 9] = -100
#  - temp_r2
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi2[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q8-p2.png')
```



We can see that the agent tries to avoid obstacles. As a peenalty for hittinh a wall is very high and there exist a possibility to be blown away by the wind, the agent prefers to leave a safety margin between it and a wall.

14 Question 9

Change the hyper parameter w to 0.6 and and the optimal policy map similar to previous question for reward functions. Explain the differences you observe. What do you think about value of new w compared to previous value? Choose the w that you think give rise to better optimal policy and use that w for the next stages of the project.

[47]:
```
# set up discount factor
gamma = 0.8
# set wind probability
w = 0.6
# set new transition probabiities
P2 = transition_probabilities(w)
# create new environments
env3 = EnvironmentRL(S, A, P2, R1, gamma)
env4 = EnvironmentRL(S, A, P2, R2, gamma)
```

[48]:
```
# do value iteration for environments
V3, _ = value_iteration(env3, eps)
V4, _ = value_iteration(env4, eps)
# compute optimal policy for environments
pi3 = computation(env3, V3)
pi4 = computation(env4, V4)
```

[86]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(V3[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q9-v3.png')
```

	0	1	2	3	4	5	6	7	8	9
0	-0.0	-0.0	-0.0	-0.01	-0.02	-0.06	-0.06	-0.01	0.01	0.03
1	-0.01	-0.01	-0.01	-0.02	-0.09	-0.29	-0.29	-0.06	0.02	0.06
2	-0.02	-0.05	-0.06	-0.07	-0.3	-0.61	-0.58	-0.24	0.03	0.11
3	-0.1	-0.3	-0.3	-0.15	-0.33	-0.58	-0.54	-0.18	0.11	0.2
4	-0.36	-0.63	-0.62	-0.33	-0.12	-0.23	-0.17	0.11	0.26	0.34
5	-0.38	-0.65	-0.62	-0.28	-0.01	0.06	0.15	0.28	0.44	0.57
6	-0.15	-0.36	-0.35	-0.07	0.07	0.17	0.29	0.46	0.7	0.94
7	-0.09	-0.17	-0.37	-0.23	0.1	0.28	0.46	0.72	1.09	1.54
8	-0.11	-0.35	-0.67	-0.51	-0.04	0.39	0.69	1.09	1.68	2.54
9	-0.13	-0.41	-0.9	-0.72	-0.02	0.51	0.93	1.54	2.54	2.97

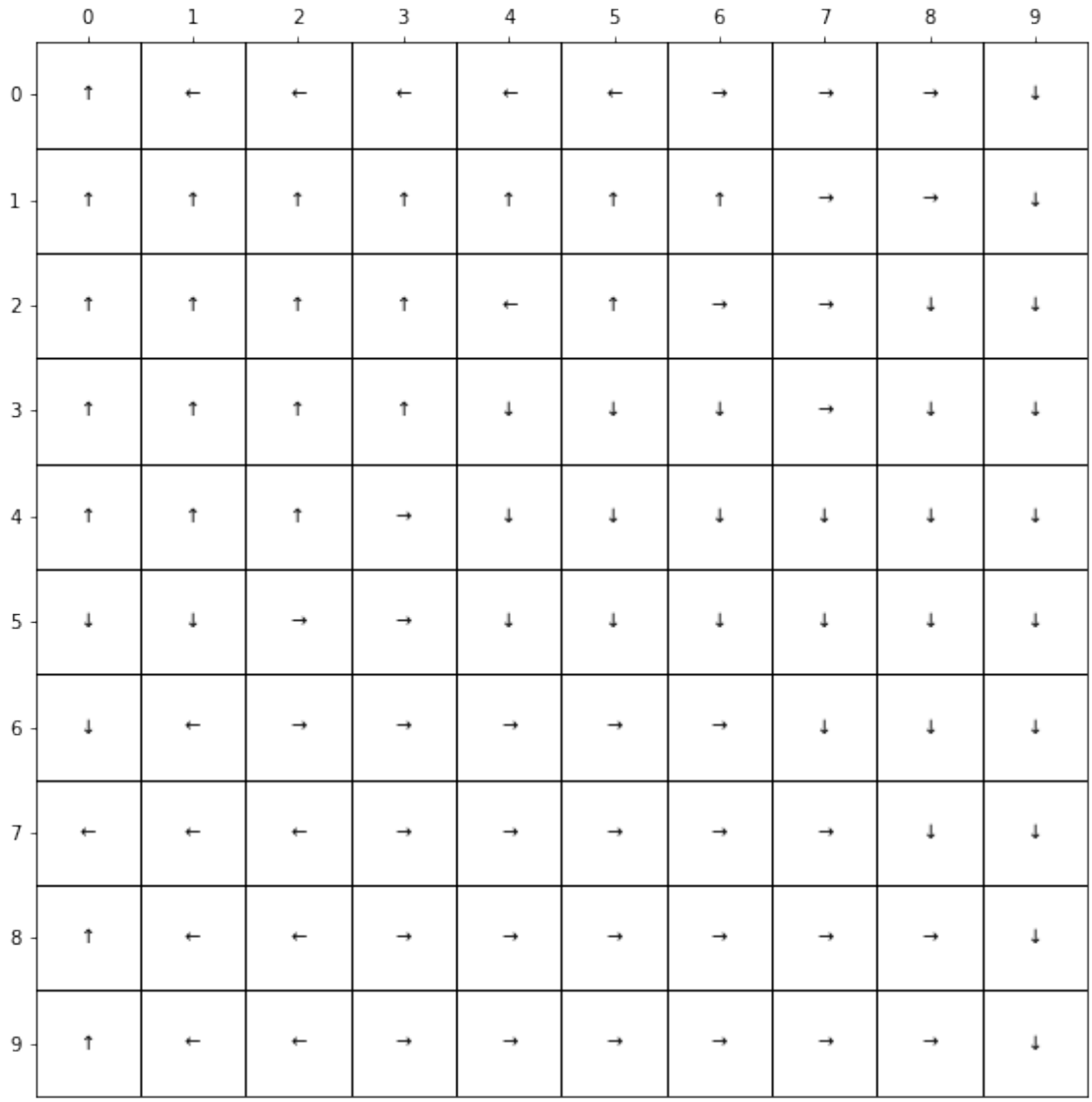
[87]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(-100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
```

```
    for j in range(10):
        text = ax.text(j, i, round(V4[i, j], 2),
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q9-v4.png')
```

	0	1	2	3	4	5	6	7	8	9
0	-0.24	-0.58	-2.46	-10.74	-38.96	-53.72	-41.27	-13.5	-4.83	-2.64
1	-0.3	-0.95	-5.14	-30.82	-67.59	-84.81	-74.44	-40.01	-12.31	-4.85
2	-0.36	-1.14	-6.02	-34.71	-76.43	-117.77	-97.52	-75.39	-39.99	-13.58
3	-0.39	-1.18	-6.15	-34.89	-74.32	-94.82	-94.03	-94.08	-74.14	-41.85
4	-0.35	-1.14	-6.02	-34.18	-70.12	-69.28	-67.41	-89.37	-85.31	-57.96
5	-0.29	-1.04	-5.65	-32.72	-65.45	-60.0	-50.2	-68.16	-84.9	-61.26
6	-0.2	-0.8	-4.51	-27.31	-45.31	-56.2	-62.78	-84.64	-79.42	-49.45
7	-0.1	-0.37	-1.73	-8.02	-30.76	-47.79	-73.23	-77.54	-60.97	-26.43
8	-0.05	-0.14	-0.58	-2.38	-8.81	-31.08	-44.68	-48.36	-20.35	11.35
9	-0.03	-0.07	-0.23	-0.83	-2.87	-9.19	-25.82	-2.84	14.57	22.97

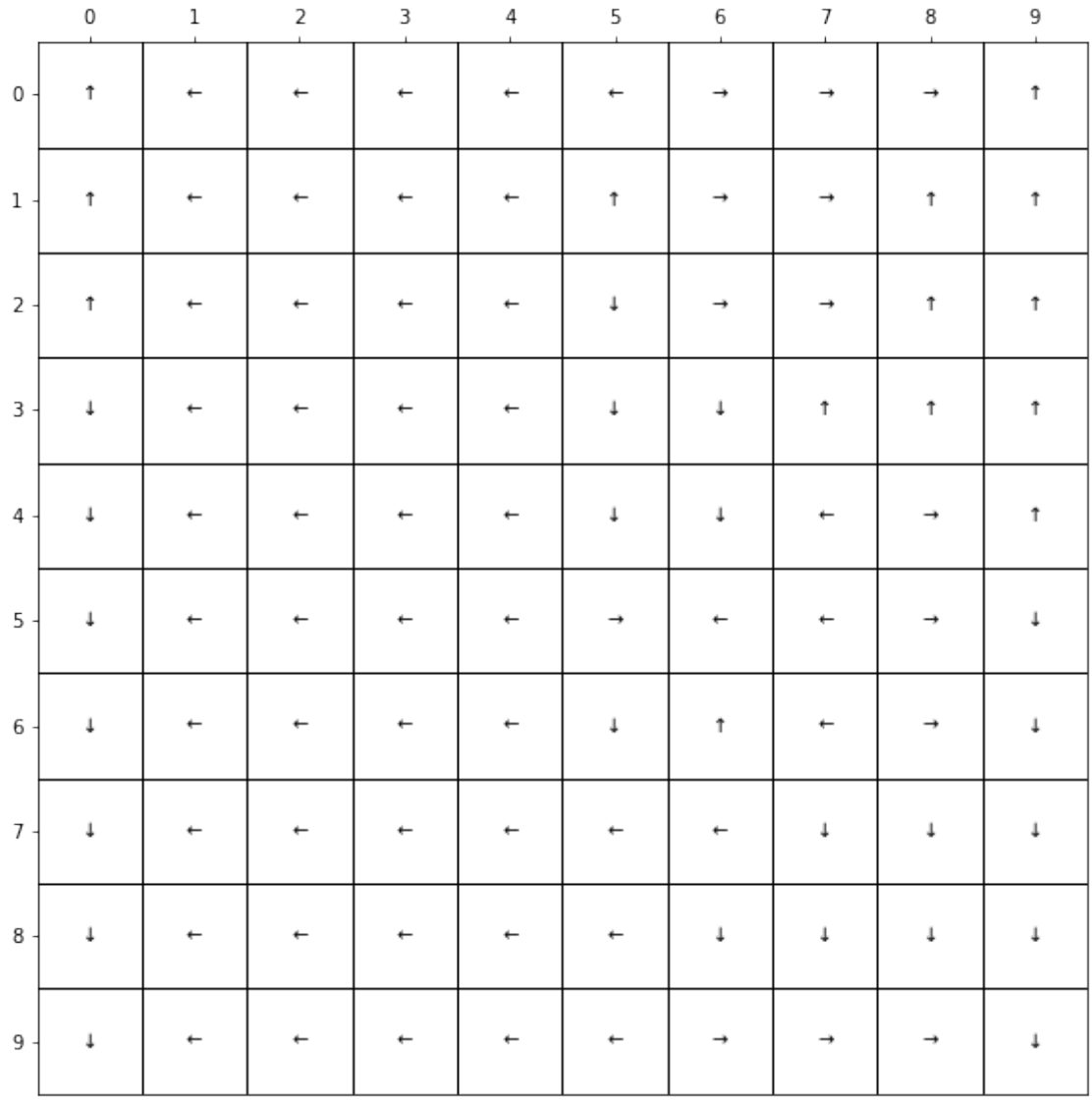
In both cases values are much lower, because control is less precise

[88]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
# + 100 * np.abs(R1)
im = ax.imshow( -100 * np.ones((10, 10)) , cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi3[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif pi3[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif pi3[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif pi3[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q9-p3.png')
```



[89]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
temp_r2 = R2.copy()
temp_r2[9, 9] = -100
# - temp_r2
im = ax.imshow(-100 * np.ones((10, 10)) , cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)
```

```
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi4[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif pi4[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif pi4[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif pi4[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q9-p4.png')
```



we see that in case 2 agent tries to avoid obstacles as much as possible because of the strong wind. From some points, it does not even try to go to the end state, because it has too high risk.

15 Question 10

Express **c**, **x**, **D**, **b** in terms of **R**, $\mathbf{P}_a$, $\mathbf{P}_{a_1}$, t_i, **u**, λ and R_{max}

To reformulate the problem

$$max_{\mathbf{R},\, t_i,\, u_i} \sum_{i=1}^{|S|} (t_i - \lambda u_i)$$

$$subject\ to \left[\Big(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i)\Big)\Big(\mathbf{I} - \gamma\mathbf{P}_{a_1}\Big)^{-1}\mathbf{R}\right] \geq t_i, \ \ \forall\, a \in A \setminus a_1,\ \forall\, i \in S$$

$$\Big(\mathbf{P}_{a_1} - \mathbf{P}_a\Big)\Big(\mathbf{I} - \gamma\mathbf{P}_{a_1}\Big)^{-1}\mathbf{R} \geq 0, \ \ \forall\, a \in A \setminus a_1$$

$$-\mathbf{u} \leq \mathbf{R} \leq \mathbf{u}$$

$$|\mathbf{R}_i| \leq R_{max}, \ \ \forall\, i \in S$$

As the problem

$$max_{\mathbf{x}}\ \mathbf{c}^T\mathbf{x}$$

$$subject\ to\ \mathbf{Dx} \leq \mathbf{b}\ \ \forall\, a \in A \setminus a_1$$

We need to set

$$c = \begin{bmatrix} \mathbf{1} \\ -\lambda\mathbf{1} \\ \mathbf{0} \end{bmatrix}, \ \ x = \begin{bmatrix} \mathbf{t} \\ \mathbf{u} \\ \mathbf{R} \end{bmatrix}$$

$$D = \begin{bmatrix} \mathbf{I} & \mathbf{0} & -(\mathbf{P_a} - \mathbf{P_{a1}})(\mathbf{I} - \gamma\mathbf{P_{a1}})^{-1} \\ \mathbf{0} & \mathbf{0} & -(\mathbf{P_a} - \mathbf{P_{a1}})(\mathbf{I} - \gamma\mathbf{P_{a1}})^{-1} \\ \mathbf{0} & -\mathbf{I} & \mathbf{I} \\ \mathbf{0} & -\mathbf{I} & -\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{bmatrix}, \ \ b = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{R_{max}} \\ \mathbf{R_{max}} \end{bmatrix}$$

16 Question 11

Sweep λ from 0 to 5 to get 500 evenly spaced values for λ. For each value of λ compute $O_A(s)$ by following the process described above. For this problem, use the optimal policy of the agent found in question 5 to fill in the $O_E(s)$ values. Then use equation 3 to compute the accuracy of the IRL algorithm for this value of λ. You need to repeat the above process for all 500 values of λ to get 500 data points. Plot λ (x-axis) against Accuracy (y-axis). In this question, you should have 1 plot.

Note: obviously, we will use $w = 0.1$ for the IRL section of this project

[54]:
```
def irl_cvxopt(environment, policy, lbd, Rmax):
    # calculate size of the state space
    S = environment.S.shape[0] * environment.S.shape[0]
    # create weight vector
    c = np.ones((S,1))
    c = np.vstack((c,(-lbd*c)))
    c = np.vstack((c,np.zeros((S,1))))
    # create a vector of constraints
    b = np.vstack((np.zeros((S,1)),np.zeros((S,1)),np.zeros((S,1)),np.
 ↪zeros((S,1)),np.zeros((S,1)),np.zeros((S,1)),np.zeros((S,1)),np.zeros((S,1)),
                   Rmax*np.ones((S,1)),Rmax*np.ones((S,1))))
```

```
    # calculate P_a matrix
    # method flatten takes the first row, then appends the second row to it,␣
↪then the third and so on
    # this is why to perserve the order of rows in P_a we need to add rows in␣
↪the same order, starting at [0, 0],
    # then [0, 1], then ..., then [0, 9], then [1, 0] and so on
    # init P_a with the first state
    P_a = environment.P[(policy[0, 0], 0, 0)].flatten()
    Pa1 = np.zeros((S,S))
    for y in range(environment.S.shape[1]):
        for x in range(environment.S.shape[0]):
            # skip the first state
            if not (x == 0 and y == 0):
                # get a new row
                new_row = environment.P[(policy[y, x], x, y)].flatten()
                # append it to P_a
                P_a = np.vstack((P_a, new_row))
    # get an identity matrix
    iden = np.identity(environment.S.flatten().shape[0])
    # calculate the inverse
    cur_inv = np.linalg.inv(iden - lbd * P_a)

    # boolean variable to check if we already have matrix D to append to
    first = 0
    for y in range(environment.S.shape[1]):
        for x in range(environment.S.shape[0]):
            # for each action the constraints are separate
            for action in environment.A:
                # constraints exist only for non-optimal action
                if not (action == policy[y, x]):
                    # we need to reproduce (P_a1[i] - P_a[i]) @ inv @ R >= t_i
                    # let's calculate (P_a1[i] - P_a[i]) @ inv
                    weights = (P_a[10 * y + x] - environment.P[(action, x, y)].
↪flatten()) @ cur_inv
                    # now let's reformulate original problem as t_i - (P_a1[i]␣
↪- P_a[i]) @ inv @ R <= 0
                    # we need to create a vector that has -1 for t_i and␣
↪weights for R
                    eq2 = np.zeros((S))
                    eq2[10 * y + x] = -1
                    output = np.hstack((eq2, np.zeros(S), weights))
                    # we double this vector with weights variable to represent␣
↪(P_a1[i] - P_a[i]) @ inv @ R >= 0
                    output1 = np.hstack((np.zeros(S), np.zeros(S), weights))
```

```
                    # we store both at temp variable and add them to eq, which␣
↪is a block matrix
                    temp = np.vstack((output,output1))
                    if first == 0:
                        eq = temp
                        first = 1
                    else:
                        eq = np.vstack((eq, temp))

    # we invert block matrix to account for negative signs (see on top)
    D = -eq
    # add -u + r <= 0
    D = np.vstack((D, np.hstack((np.zeros((S,S)),-np.eye((S)),np.eye((S))))))
    # add -u - r <= 0
    D = np.vstack((D, np.hstack((np.zeros((S,S)),-np.eye((S)),-np.eye((S))))))
    # add r <= R_max
    D = np.vstack((D, np.hstack((np.zeros((S,S)),np.zeros((S,S)),np.eye((S))))))
    # add -r <= R_max
    D = np.vstack((D, np.hstack((np.zeros((S,S)),np.zeros((S,S)),-np.
↪eye((S))))))

    # Create cvxopt objects from the produced matrices
    c = matrix(c)
    b = matrix(b)
    G = matrix(D)
    # solve the problem
    # max cx
    # Gx + s = b
    solvers.options['show_progress'] = False
    try:
        # solve the problem
        sol=solvers.lp(-c,G,b)
        solution = sol['x'][-100:]
        inc = 0
        R = np.zeros((environment.S.shape[0],environment.S.shape[0]))
        for i in range(environment.S.shape[0]):
            for j in range(environment.S.shape[0]):
                R[i,j] = solution[inc]
                inc = inc + 1
    except ValueError:
        R = np.zeros((10,10))

    return R
```

[55]:
```
def irl_cvxpy(environment, policy, lbd, Rmax):
    # calculate P_a matrix
```

```
    # method flatten takes the first row, then appends the second row to it, then the third and so on
    # this is why to perserve the order of rows in P_a we need to add rows in the same order, starting at [0, 0],
    # then [0, 1], then ..., then [0, 9], then [1, 0] and so on
    # init P_a with the first state
    P_a = environment.P[(policy[0, 0], 0, 0)].flatten()
    for y in range(environment.S.shape[1]):
        for x in range(environment.S.shape[0]):
            # skip the first state
            if not (x == 0 and y == 0):
                # get a new row
                new_row = environment.P[(policy[y, x], x, y)].flatten()
                # append it to P_a
                P_a = np.vstack((P_a, new_row))
    # get an identity matrix
    iden = np.identity(environment.S.flatten().shape[0])
    # calculate the inverse
    cur_inv = np.linalg.inv(iden - lbd * P_a)

    # initialize optimization variables
    # the variables included here are (R, t, u), each of which is from R^|S|
    R = cp.Variable(environment.S.flatten().shape[0])
    t = cp.Variable(environment.S.flatten().shape[0])
    u = cp.Variable(environment.S.flatten().shape[0])
    # declare loss; it is the reverse of the thing we want to maximize
    loss = -cp.sum(t - lbd * u)
    # declare objective
    objective = cp.Minimize(loss)
    # declare constraints
    constraints = []
    # declare constraints on the values of R, u
    constraints.append(R <= u)
    constraints.append(-u <= R)
    constraints.append(R <= Rmax * np.ones(environment.S.flatten().shape[0]))
    constraints.append(-R <= Rmax * np.ones(environment.S.flatten().shape[0]))
    # declare constraints on the states
    for y in range(environment.S.shape[1]):
        for x in range(environment.S.shape[0]):
            # for each action the constraints are separate
            for action in environment.A:
                # constraints exist only for non-optimal action
                if not (action == policy[y, x]):
                    # calculate left par of the equation
                    weights = (P_a[10 * y + x] - environment.P[(action, x, y)].flatten()) @ cur_inv
                    # add constraints
```

```
                    constraints.append(weights @ R >= t[10 * y + x])
                    constraints.append(weights @ R >= 0)

    # solve the problem
    prob = cp.Problem(objective, constraints)
    try:
        result = prob.solve(solver = cp.SCS)
    except cp.SolverError:
        print("solver failed!")
        return np.zeros((10, 10))

    # output solution if possible
    if prob.status == "infeasible":
        print("Problem is infeasible!")
        return np.zeros((10, 10))
    elif prob.status == "unbounded":
        print("Problem is unbounded!")
        return np.zeros((10, 10))
    else:
        pred_R = R.value
        pred_R = pred_R.reshape((10, 10))
        return pred_R
```

[64]:
```
def irl(environment, policy, lbd, Rmax):
    # try cvxopt solver
    pred_R = irl_cvxopt(environment, policy, lbd, Rmax)
    # if solver failed
    if np.all(pred_R == 0):
        # try cvxpy solver
        pred_R = irl_cvxpy(environment, policy, lbd, Rmax)

    return pred_R
```

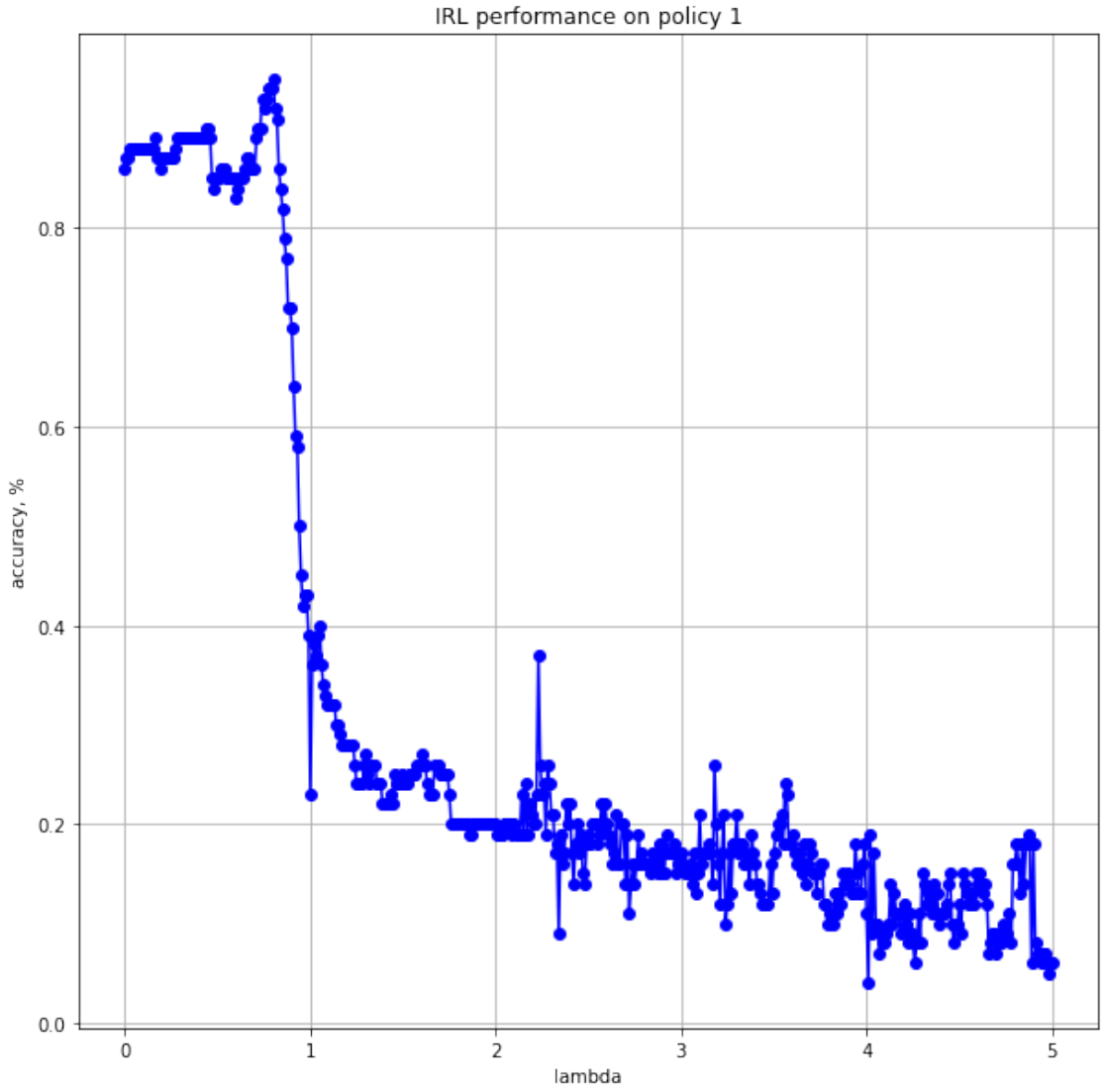
[57]:
```
def compare_policies(policy_1, policy_2):
    # this function calculates accuracy according to the equation 3 in the␣
 ↪project spec
    # init counter
    accuracy = 0
    # sweep overl all states
    for i in range(policy_1.shape[0]):
        for j in range(policy_1.shape[1]):
            # check if actions are the same
            if policy_1[i, j] == policy_2[i, j]:
                # update counter
                accuracy += 1
    # normilize counter
    accuracy /= policy_1.flatten().shape[0]
```

```
    return accuracy
```

[65]:
```
# declare number of lambda points to test over
NUM_POINTS = 500

# set all necessary constants for creating an environment
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# init the vector of accuracies
accs = np.zeros((NUM_POINTS+1))
# init lambdas to sweep through
lambdas = np.linspace(0, 5, num=NUM_POINTS+1)
# sweep through them
for i in range(lambdas.shape[0]):
    # compute reverse-engineered R
    irl_R = irl(env1, pi1, lambdas[i], np.max(np.abs(R1)))
    # create temporary RL environment
    irl_env = EnvironmentRL(S, A, P, irl_R, gamma)
    # do value iteration on it
    irl_V, _ = value_iteration(irl_env, eps)
    # fetch optimal policy
    irl_pi = computation(irl_env, irl_V)
    # compare policies and store accuracy
    accs[i] = compare_policies(pi1, irl_pi)
```

```
C:\Users\slavc\Desktop\UCLA\Courses\Spring_2021\232E\232E\venv_232E\lib\site-
packages\cvxpy\problems\problem.py:1267: UserWarning: Solution may be
inaccurate. Try another solver, adjusting the solver settings, or solve with
verbose=True for more information.
  warnings.warn(
```

[67]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(lambdas, accs, "-ob")
ax.grid()
ax.title.set_text("IRL performance on policy 1")
ax.set_xlabel('lambda')
ax.set_ylabel('accuracy, %')
plt.savefig('plots/q11-pi1.png')
```



17 Question 12

Use the plot in question 11 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as $\lambda^{(1)}_{max}$. Please report $\lambda^{(1)}_{max}$

[130]:
```
lbd_max_1 = lambdas[np.argmax(accs)]
lbd_max = 0.8
print("Lambda Max: ", lbd_max_1)
```

```
Lambda Max:  0.8
```

18 Question 13

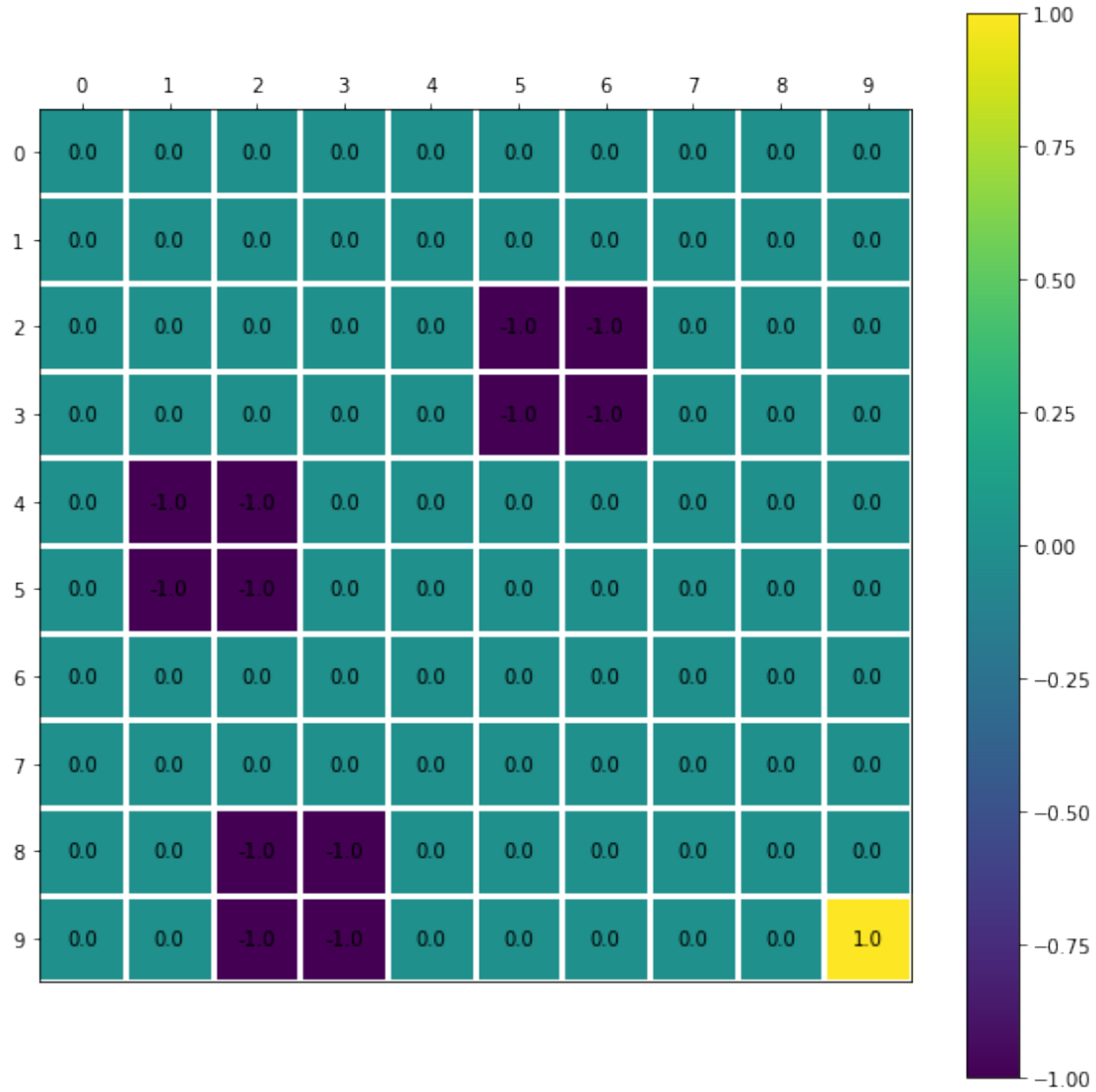
For $\lambda_{max}^{(1)}$, generate heat maps of the ground truth reward and the extracted reward. Please note that the ground truth reward is the Reward function 1 and the extracted reward is computed by solving the linear program given by equation 2 with the λ parameter set to $\lambda_{max}^{(1)}$. In this question, you should have 2 plots.

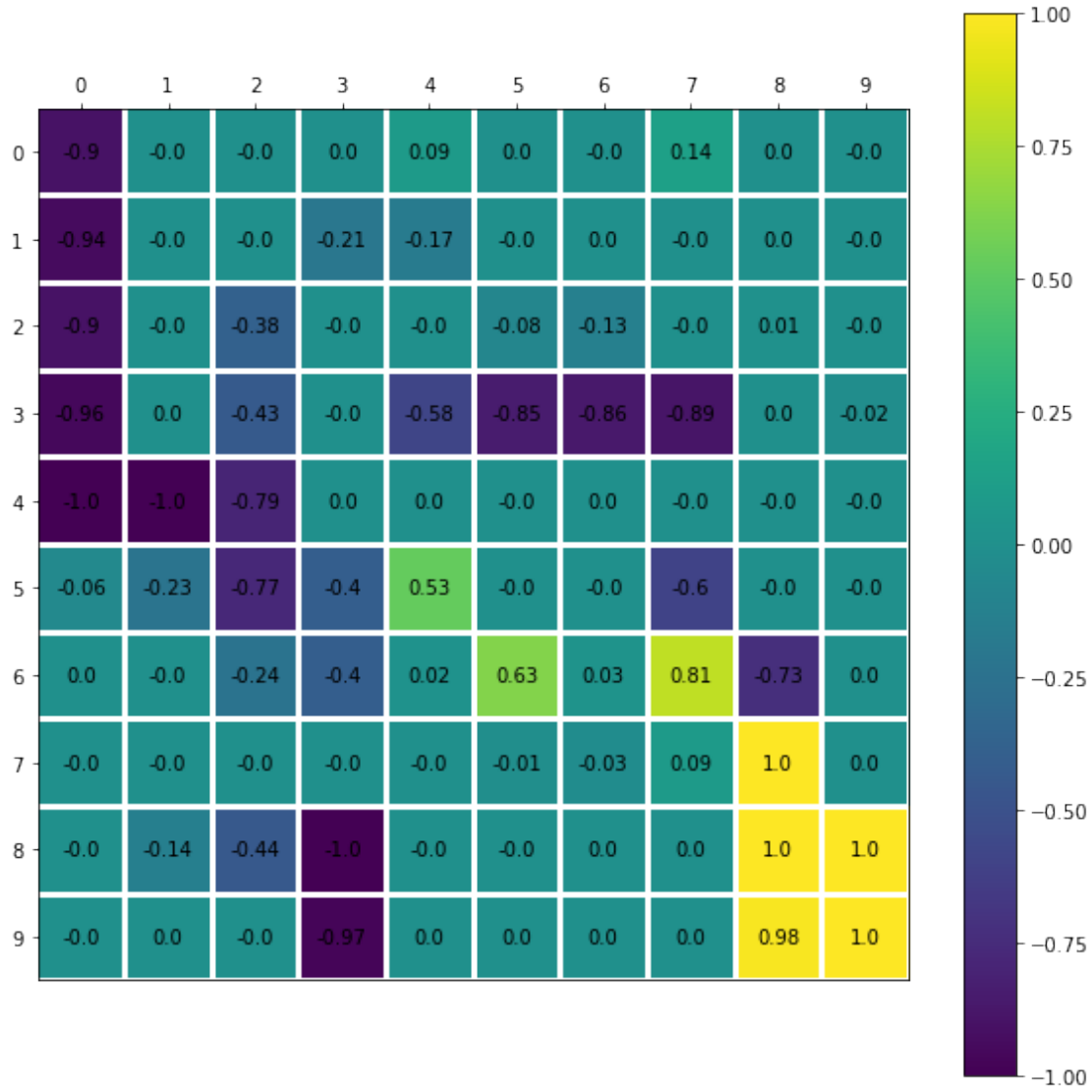
[131]:
```
# recreate all parameters of env1
R1 = create_reward([(1, 4, -1), (1, 5, -1), (2, 4, -1), (2, 5, -1),
                    (2, 8, -1), (2, 9, -1), (3, 8, -1), (3, 9, -1),
                    (5, 2, -1), (5, 3, -1), (6, 2, -1), (6, 3, -1),
                    (9, 9, 1)])
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# compute reverse-engineered R
irl_R1 = irl(env1, pi1, lbd_max_1, np.max(np.abs(R1)))
# create temporary RL environment
irl_env1 = EnvironmentRL(S, A, P, irl_R1, gamma)
# do value iteration on it
irl_V1, irl_rec1 = value_iteration(irl_env1, eps)
# fetch optimal policy
irl_pi1 = computation(irl_env1, irl_V1)
# compare policies
print(compare_policies(pi1, irl_pi1))
```

```
0.95
```

[103]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(R1)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, R1[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)
```

```
cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)


plt.savefig('plots/q13-r1.png')


fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(irl_R1)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

irl_R1 = irl_R1.round(decimals = 2)
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, irl_R1[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q13-irl-r.png')
```


0 1 2 3 4 5 6 7 8 9
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0
3 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0
4 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
6 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
8 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0
9 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 1.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

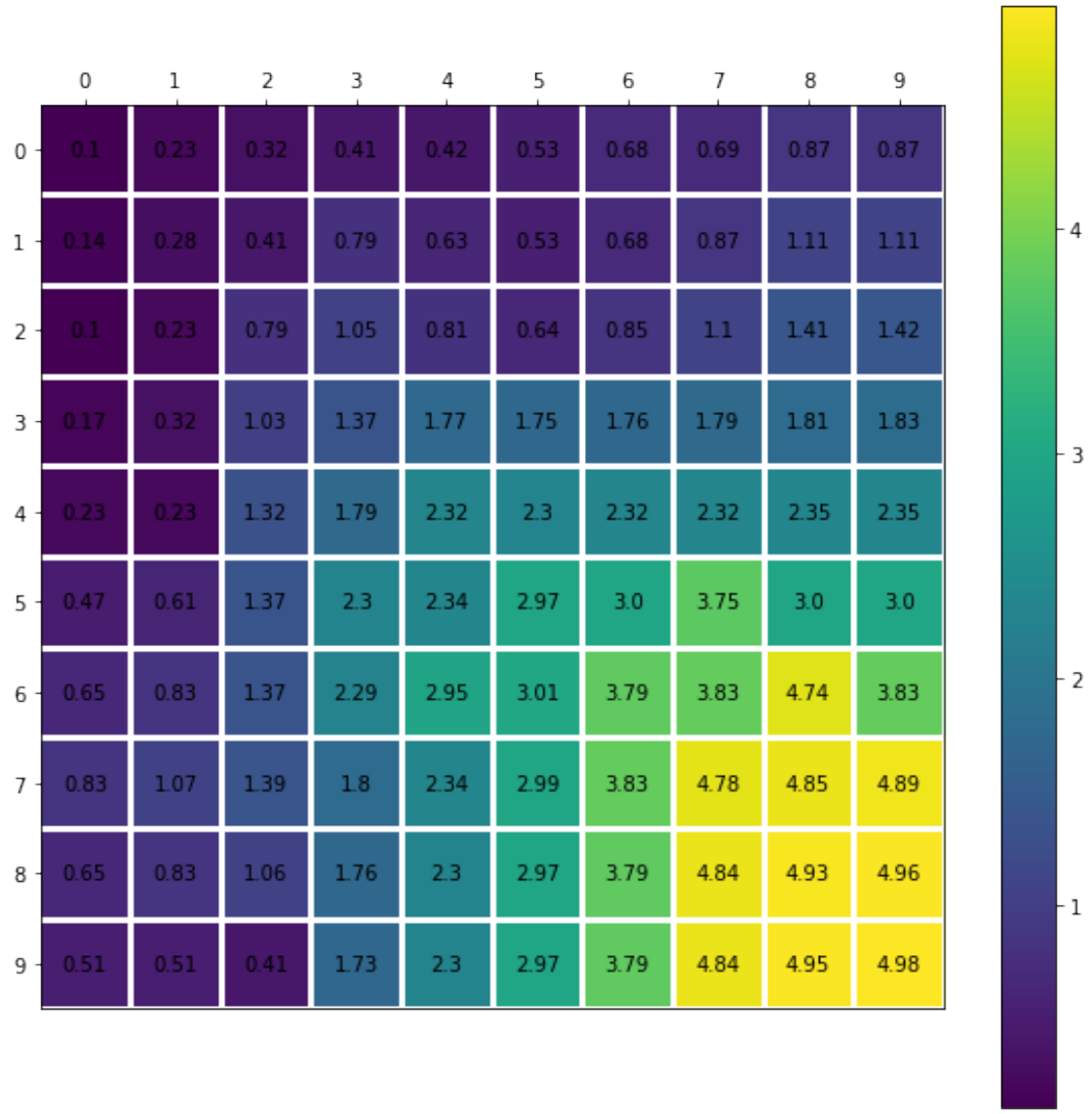


19 Question 14

Use the extracted reward function computed in question 13, to compute the optimal values of the states in the 2-D grid. For computing the optimal values you need to use the optimal state-value function that you wrote in question 2. For visualization purpose, generate a heat map of the optimal state values across the 2-D grid (similar to the figure generated in question 3). In this question, you should have 1 plot.

[105]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow((irl_V1))
```

```
ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, round(irl_V1[i, j], 2),
                       ha="center", va="center", color="black")
cbar = ax.figure.colorbar(im, ax=ax)

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q14.png')
```



20 Question 15

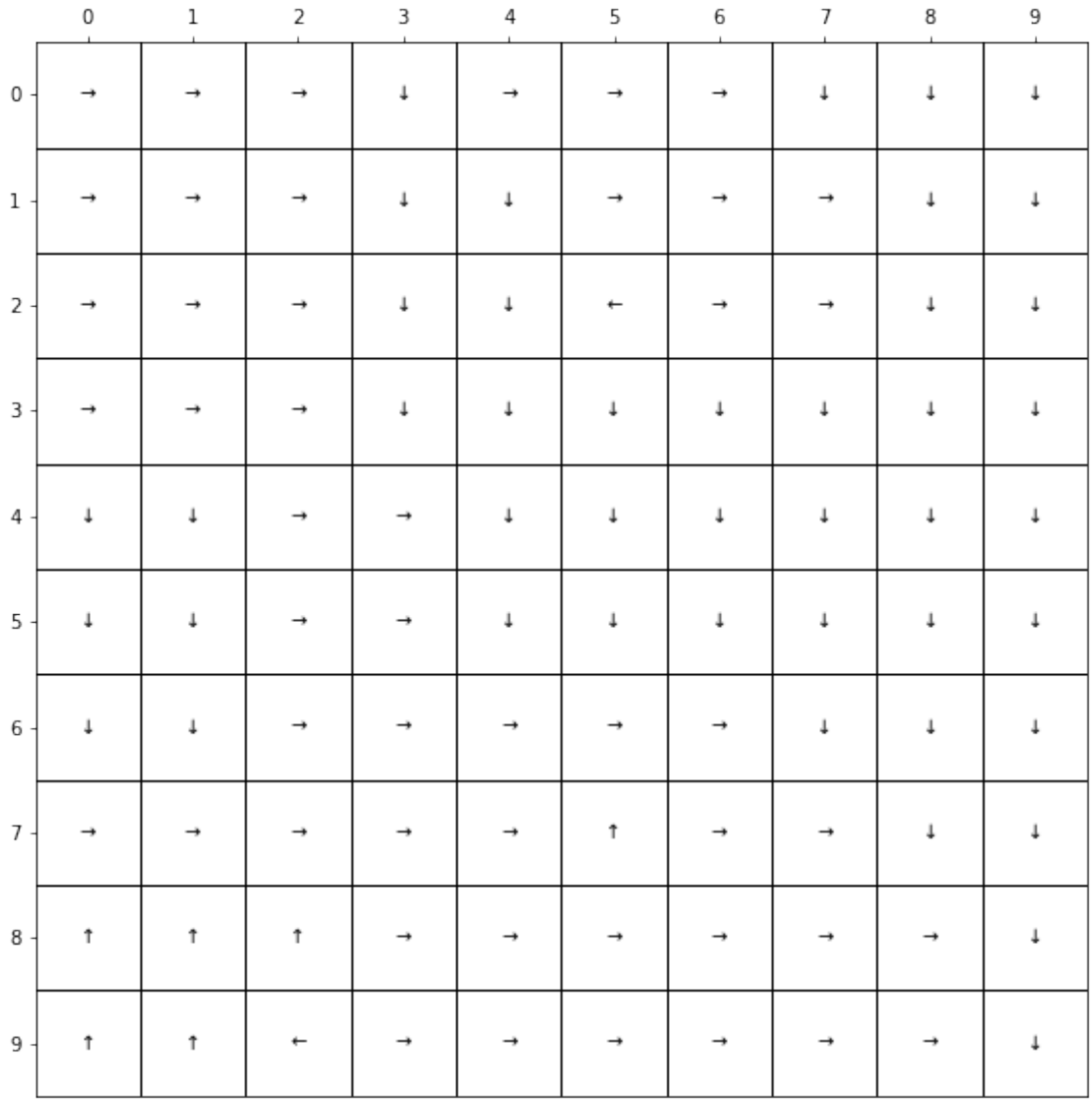
Compare the heat maps of Question 3 and Question 14 and provide a brief explanation on their similarities and differences.

21 Question 16

Use the extracted reward function found in question 13 to compute the optimal policy of the agent. For computing the optimal policy of the agent you need to use the function that you wrote in question 5. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The actions should be

displayed using arrows. In this question, you should have 1 plot.

[127]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
# + 100 * np.abs(R1)
im = ax.imshow( -100 * np.ones((10, 10)), cmap="Greys", norm=colors.
 ↪CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if irl_pi1[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q16.png')
```

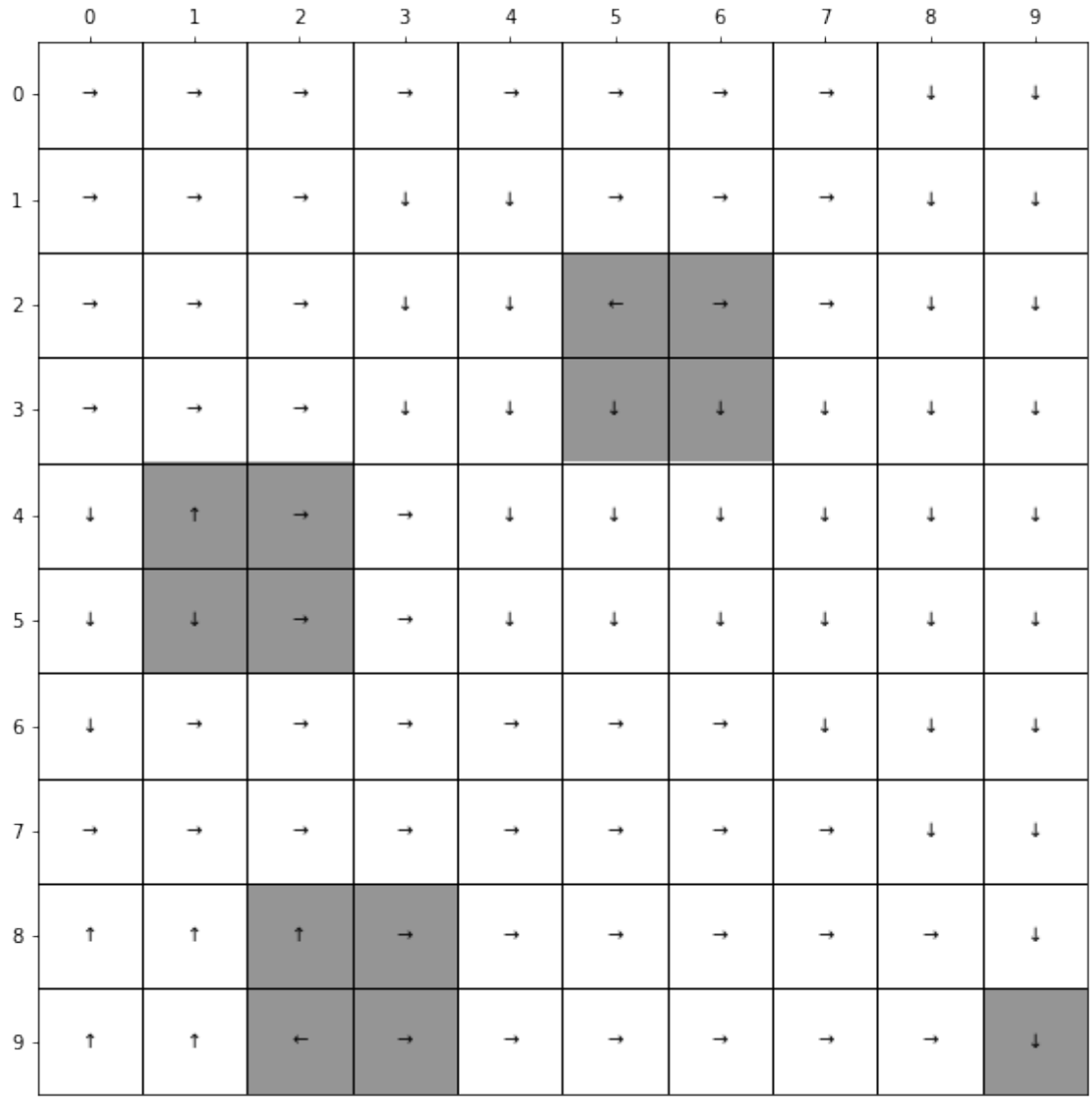


22 Question 17

Compare the figures of Question 5 and Question 16 and provide a brief explanation on their similarities and differences.

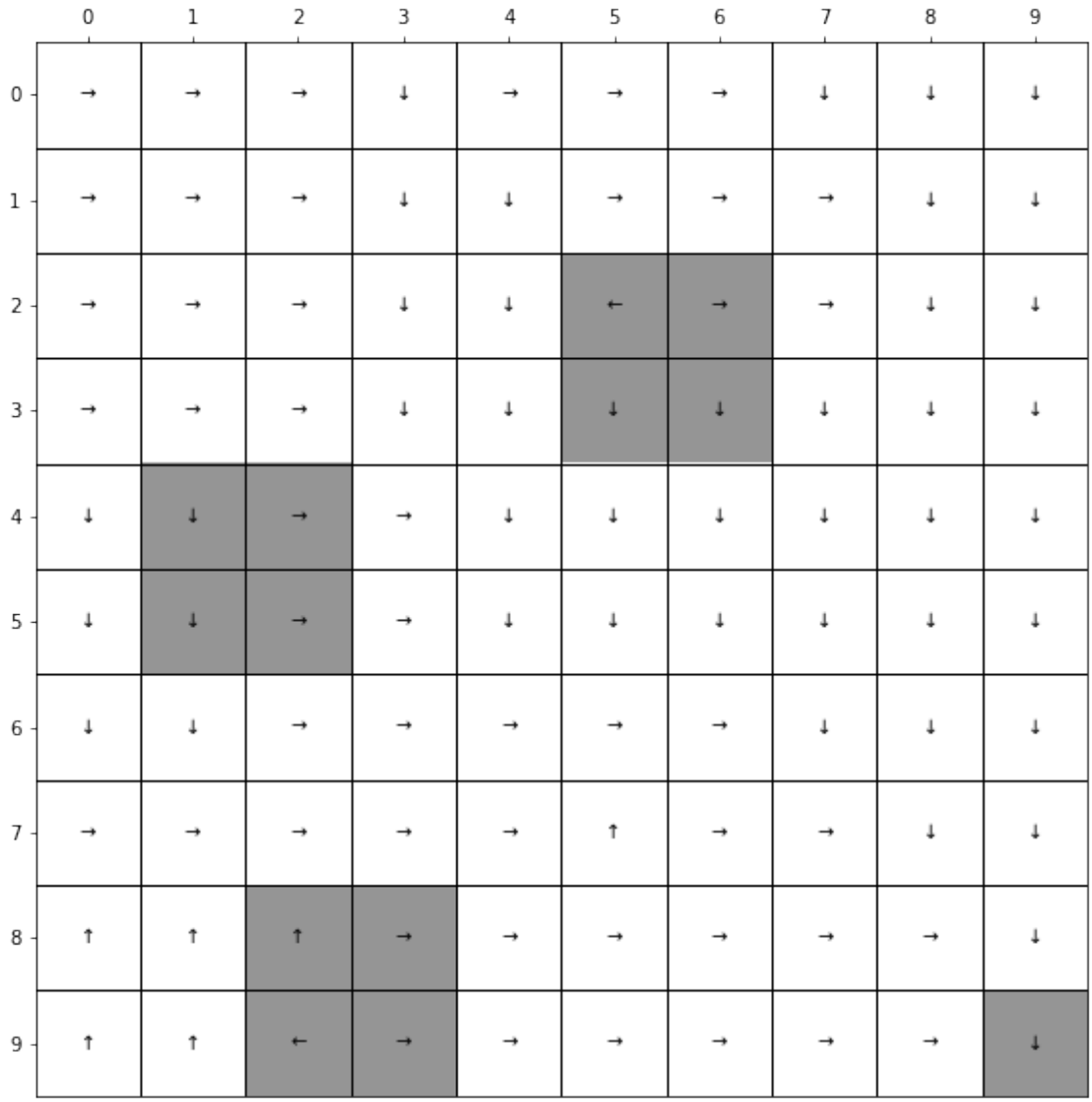
[132]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow( -100 * np.ones((10, 10)) + 100 * np.abs(R1), cmap="Greys",␣
 ↪norm=colors.CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
```

```
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi1[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
↪color="black")
        elif pi1[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
↪color="black")
        elif pi1[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
↪color="black")
        elif pi1[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q17-gt.png')
```



[133]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
#
im = ax.imshow( -100 * np.ones((10, 10)) + 100 * np.abs(R1), cmap="Greys",␣
 ↪norm=colors.CenteredNorm(0))

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
```

```
for i in range(10):
    for j in range(10):
        if irl_pi1[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi1[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q17-irl.png')
```



23 Question 18

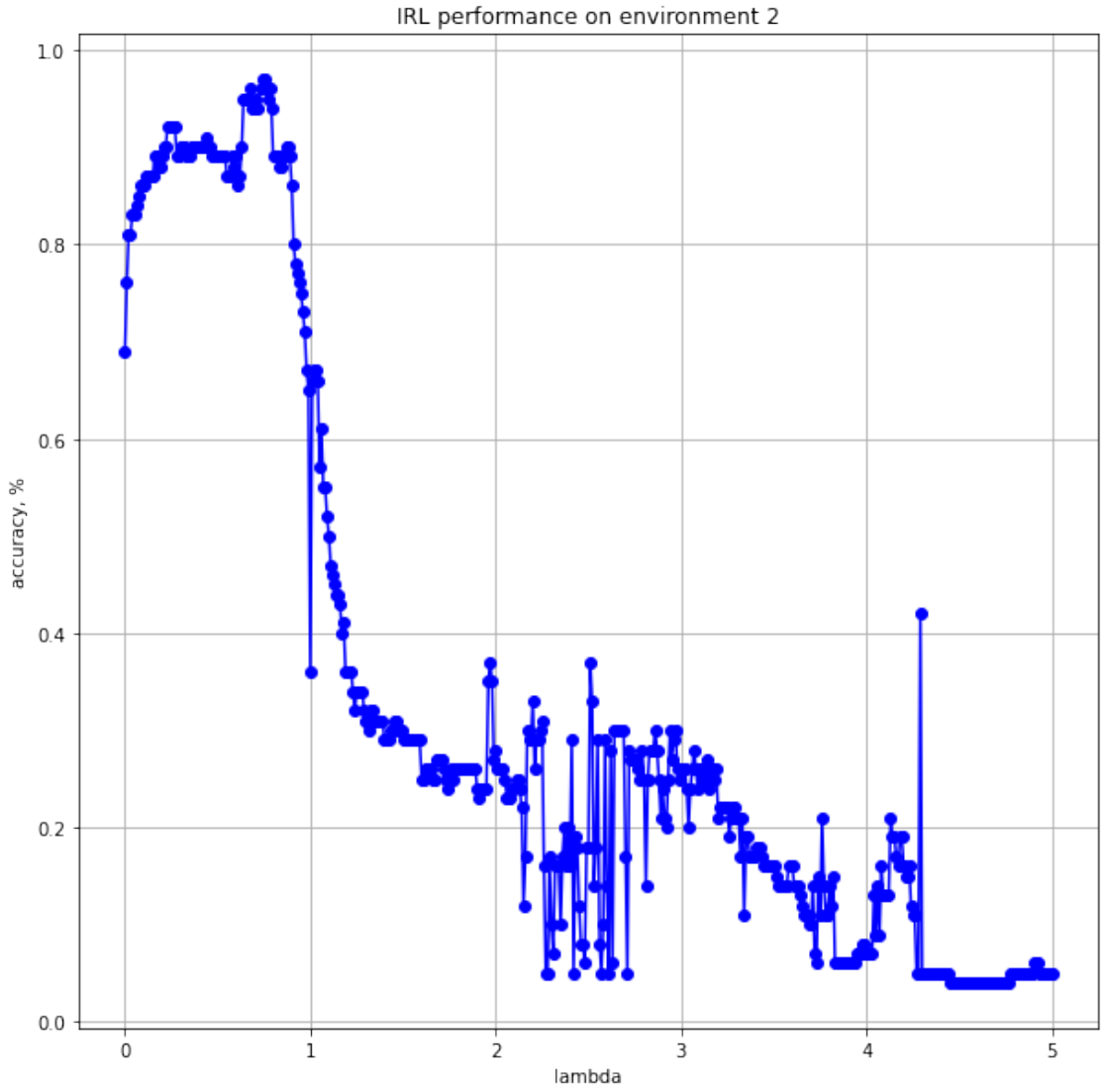
Sweep λ from 0 to 5 to get 500 evenly spaced values for λ. For each value of λ compute $O_A(s)$ by following the process described above. For this problem, use the optimal policy of the agent found in question 9 to fill in the $O_E(s)$ values. Then use equation 3 to compute the accuracy of the IRL algorithm for this value of λ. You need to repeat the above process for all 500 values of λ to get 500 data points. Plot λ (x-axis) against Accuracy (y-axis). In this question, you should have 1 plot.

[106]:
```
# declare number of lambda points to test over
NUM_POINTS = 500

# set all necessary constants for creating an environment
```

```
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# init the vector of accuracies
accs2 = np.zeros((NUM_POINTS+1))
# init lambdas to sweep through
lambdas = np.linspace(0, 5, num=NUM_POINTS+1)
# sweep through them
for i in range(lambdas.shape[0]):
    # compute reverse-engineered R
    irl_R2 = irl(env2, pi2, lambdas[i], np.max(np.abs(R2)))
    # create temporary RL environment
    irl_env2 = EnvironmentRL(S, A, P, irl_R2, gamma)
    # do value iteration on it
    irl_V2, _ = value_iteration(irl_env2, eps)
    # fetch optimal policy
    irl_pi2 = computation(irl_env2, irl_V2)
    # compare policies and store accuracy
    accs2[i] = compare_policies(pi2, irl_pi2)
```

```
C:\Users\slavc\Desktop\UCLA\Courses\Spring_2021\232E\232E\venv_232E\lib\site-
packages\cvxpy\problems\problem.py:1267: UserWarning: Solution may be
inaccurate. Try another solver, adjusting the solver settings, or solve with
verbose=True for more information.
  warnings.warn(
```

[107]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(lambdas, accs2, "-ob")
ax.grid()
ax.title.set_text("IRL performance on environment 2")
ax.set_xlabel('lambda')
ax.set_ylabel('accuracy, %')
plt.savefig('plots/q18-pi2.png')
```



24 Question 19

Use the plot in question 18 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as $\lambda^{(2)}_{max}$. Please report $\lambda^{(2)}_{max}$

[134]:
```
lbd_max2 = lambdas[np.argmax(accs2)]
# lbd_max2 = 0.74
print("Lambda Max: ", lbd_max2)
```

```
Lambda Max:  0.74
```

25 Question 20

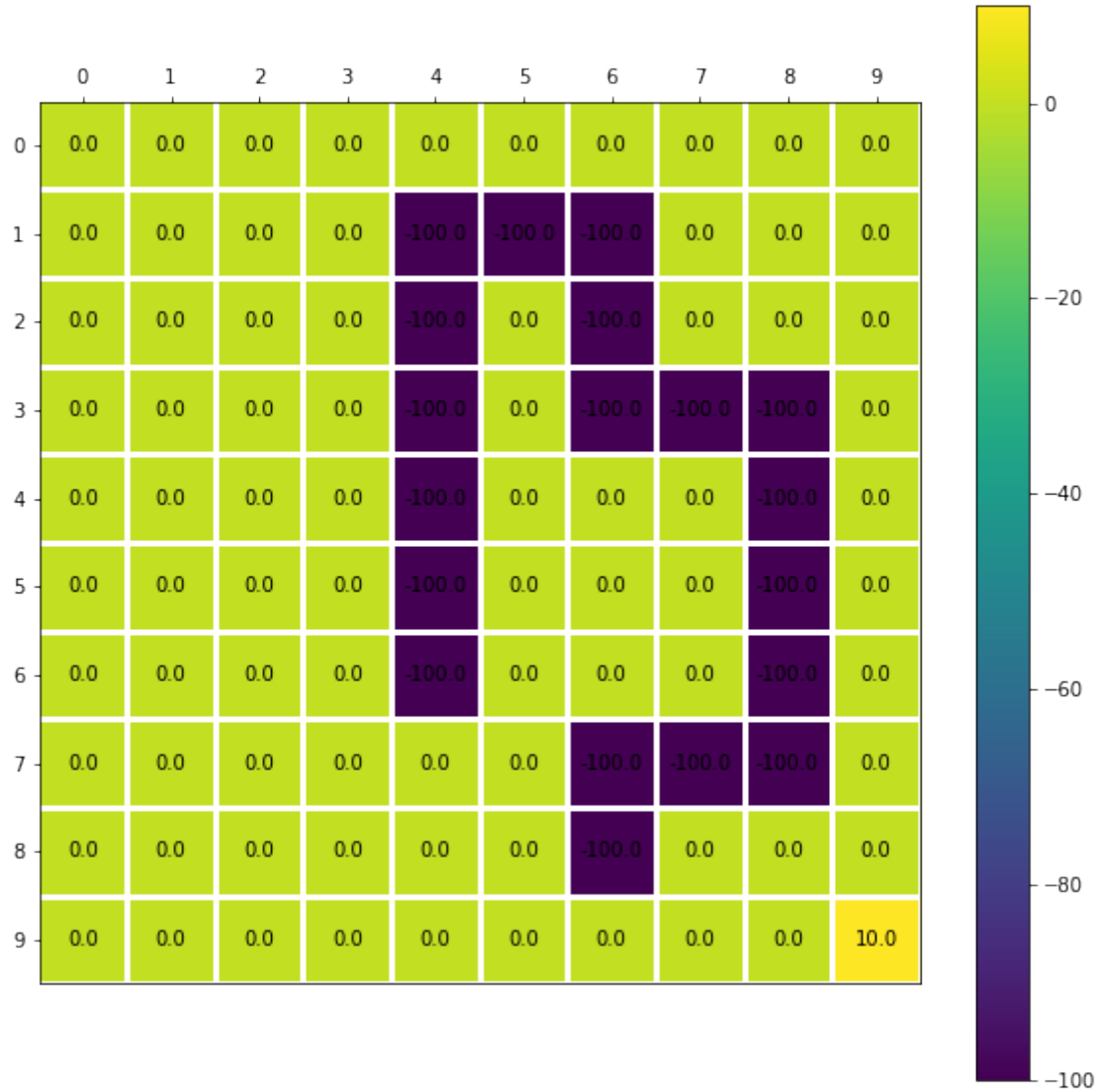
For $\lambda_{max}^{(2)}$, generate heat maps of the ground truth reward and the extracted reward. Please note that the ground truth reward is the Reward function 2 and the extracted reward is computed by solving the linear program given by equation 2 with the λ parameter set to $\lambda_{max}^{(2)}$. In this question, you should have 2 plots.

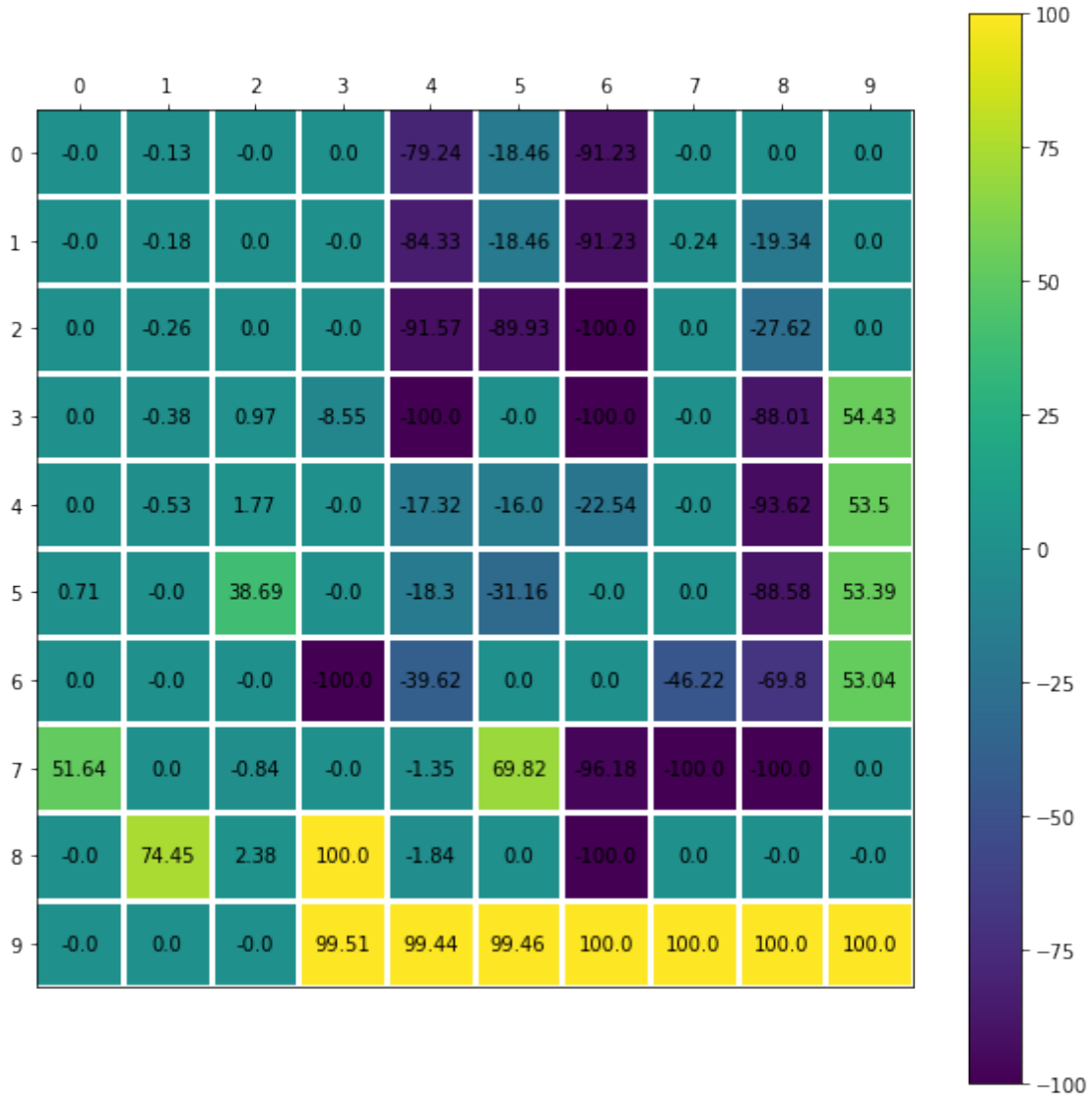
[135]:
```
# compute reverse-engineered R
R2 = create_reward([(4, 1, -100), (4, 2, -100), (4, 3, -100), (4, 4, -100), (4,␣
 ↪5, -100), (4, 6, -100),
                    (5, 1, -100),
                    (6, 1, -100), (6, 2, -100), (6, 3, -100), (6, 7, -100), (6,␣
 ↪8, -100),
                    (7, 3, -100), (7, 7, -100),
                    (8, 3, -100), (8, 4, -100), (8, 5, -100), (8, 6, -100), (8,␣
 ↪7, -100),
                    (9, 9, 10)])
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# compute reverse-engineered R
irl_R2 = irl(env2, pi2, lbd_max2, np.max(np.abs(R2)))
# create temporary RL environment
irl_env2 = EnvironmentRL(S, A, P, irl_R2, gamma)
# do value iteration on it
irl_V2, _ = value_iteration(irl_env2, eps)
# fetch optimal policy
irl_pi2 = computation(irl_env2, irl_V2)
# compare policies
print(compare_policies(pi2, irl_pi2))
```

```
0.97
```

[113]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(R2)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
```

```
    for j in range(10):
        text = ax.text(j, i, R2[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q20-r1.png')


fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(irl_R2)

ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

irl_R2 = irl_R2.round(decimals = 2)
irl_V2 = irl_V2.round(decimals = 2)
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, irl_R2[i, j],
                       ha="center", va="center", color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

cbar = ax.figure.colorbar(im, ax=ax)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q20-irl-r.png')
```


0 1 2 3 4 5 6 7 8 9
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 0.0 0.0 -100.0 -100.0 -100.0 0.0 0.0 0.0
2 0.0 0.0 0.0 0.0 -100.0 0.0 -100.0 0.0 0.0 0.0
3 0.0 0.0 0.0 0.0 -100.0 0.0 -100.0 -100.0 -100.0 0.0
4 0.0 0.0 0.0 0.0 -100.0 0.0 0.0 0.0 -100.0 0.0
5 0.0 0.0 0.0 0.0 -100.0 0.0 0.0 0.0 -100.0 0.0
6 0.0 0.0 0.0 0.0 -100.0 0.0 0.0 0.0 -100.0 0.0
7 0.0 0.0 0.0 0.0 0.0 0.0 -100.0 -100.0 -100.0 0.0
8 0.0 0.0 0.0 0.0 0.0 0.0 -100.0 0.0 0.0 0.0
9 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 10.0
0
−20
−40
−60
−80
−100

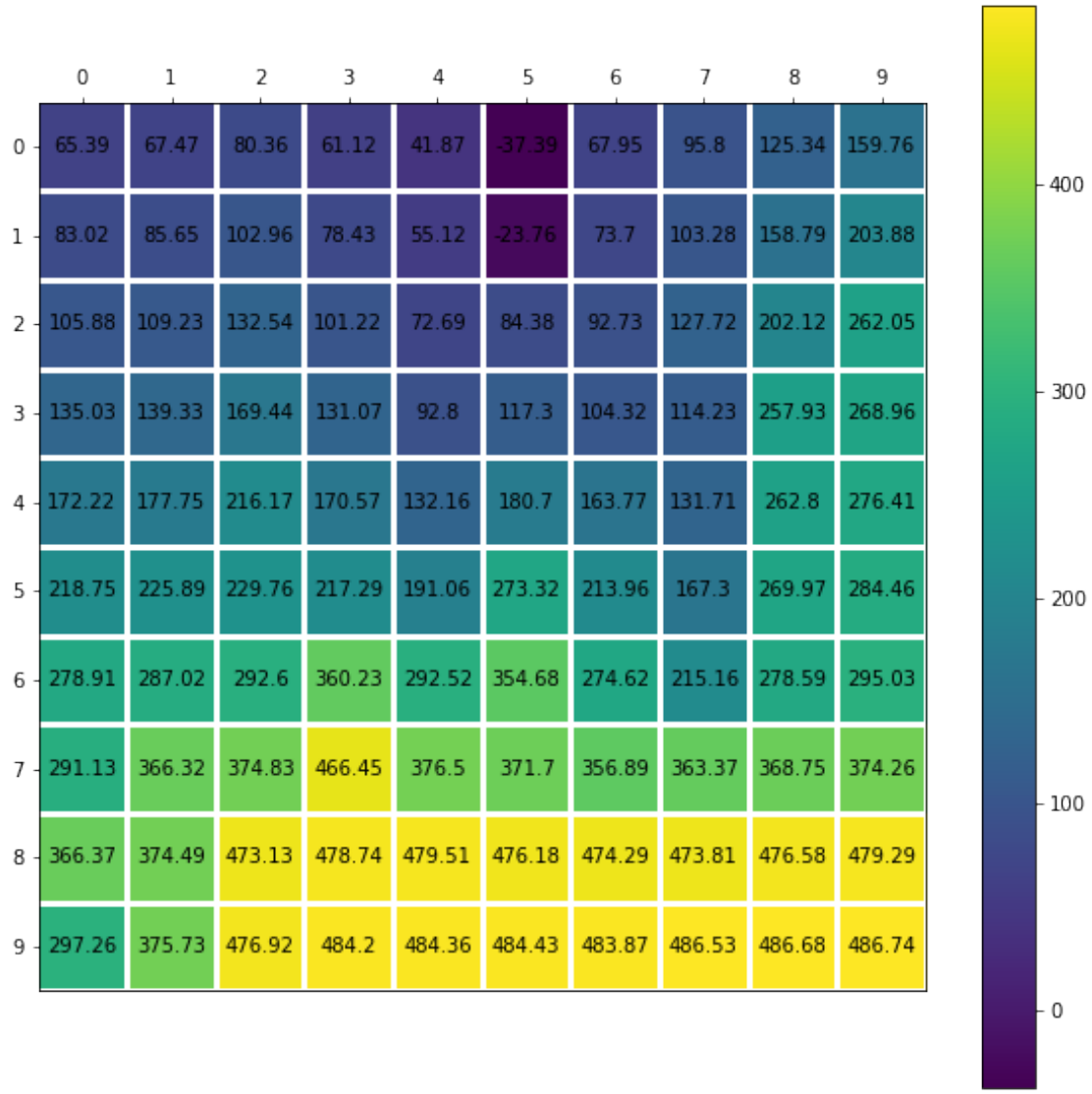


26 Question 21

Use the extracted reward function computed in question 20, to compute the optimal values of the states in the 2-D grid. For computing the optimal values you need to use the optimal state-value function that you wrote in question 2. For visualization purpose, generate a heat map of the optimal state values across the 2-D grid (similar to the figure generated in question 7). In this question, you should have 1 plot.

[112]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow((irl_V2))
```

```
ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        text = ax.text(j, i, irl_V2[i, j],
                       ha="center", va="center", color="black")
cbar = ax.figure.colorbar(im, ax=ax)

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
ax.tick_params(which="minor", bottom=False, left=False)


plt.savefig('plots/q21.png')
```



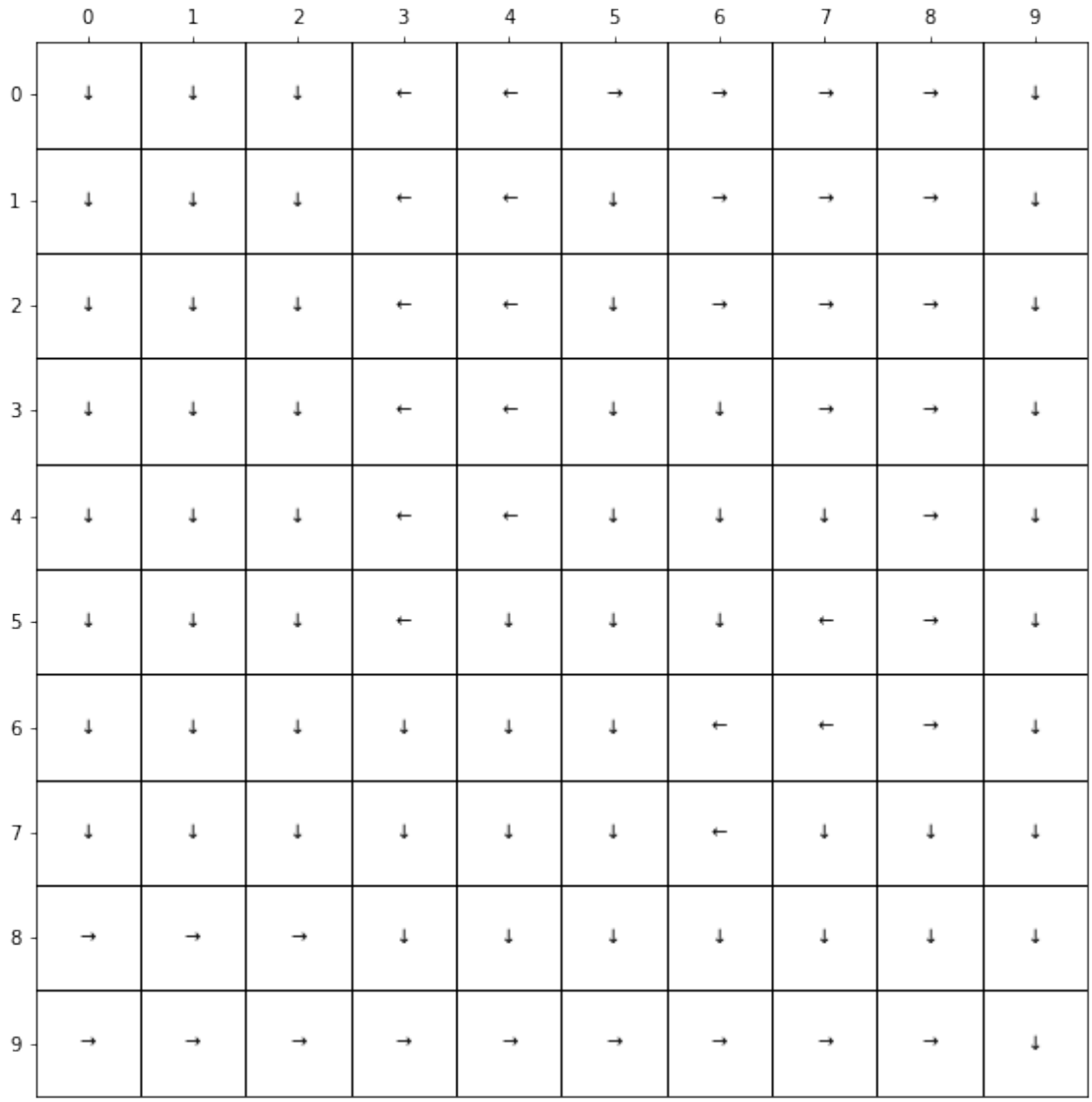
27 Question 22

Compare the heat maps of Question 7 and Question 21 and provide a brief explanation on their similarities and differences.

28 Question 23

Use the extracted reward function found in question 20 to compute the optimal policy of the agent. For computing the optimal policy of the agent you need to use the function that you wrote in question 9. For visualization purpose, you should generate a figure similar to that of figure 1 but with the number of state replaced by the optimal action at that state. The actions should be displayed using arrows. In this question, you should have 1 plot.

[114]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))

im = ax.imshow(-100 * np.ones((10, 10)),cmap="Greys" ,  norm=colors.
 ↪CenteredNorm(0))


ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if irl_pi2[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q23.png')
```

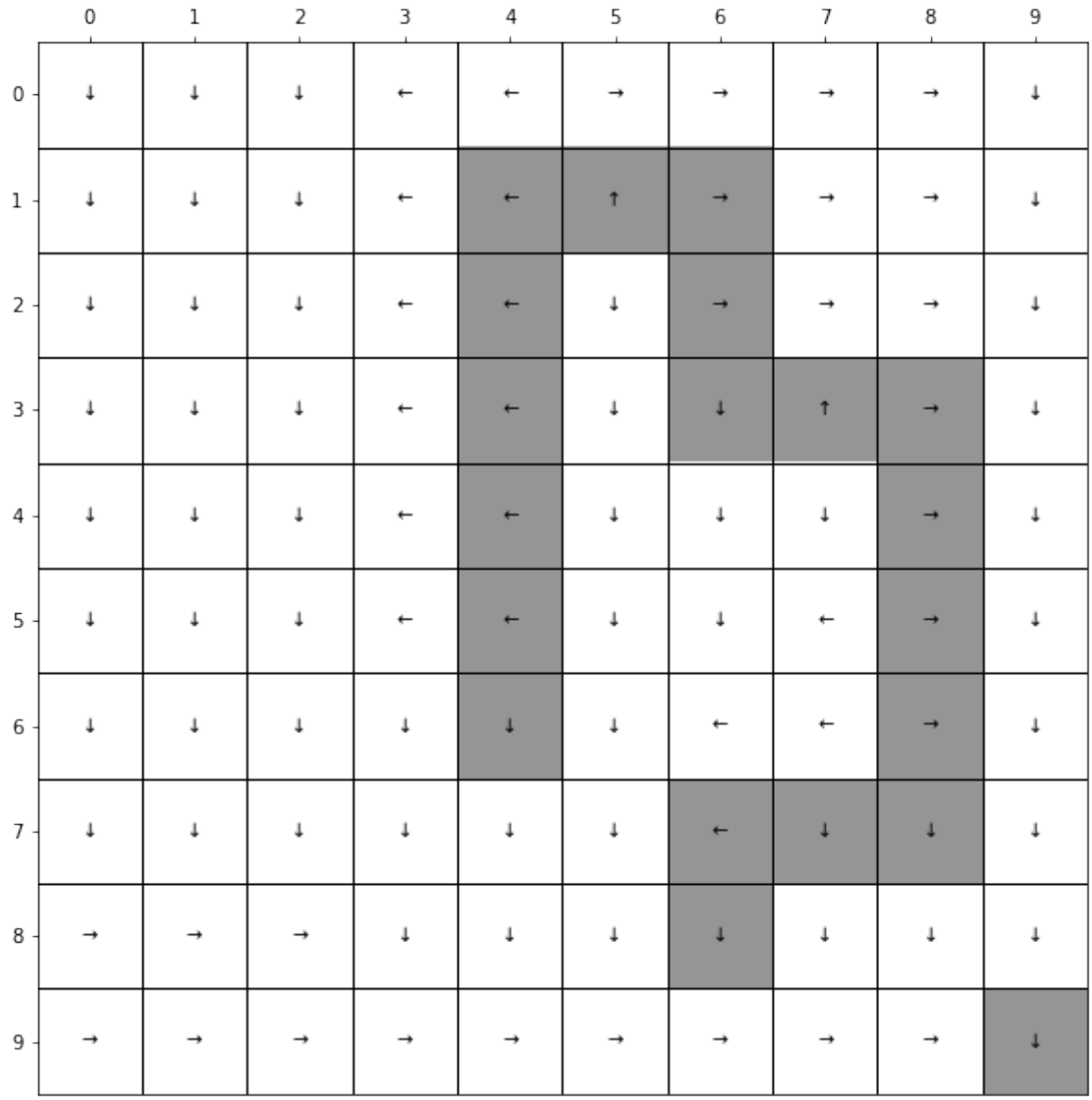


29 Question 24

Compare the figures of Question 9 and Question 23 and provide a brief explanation on their similarities and differences.

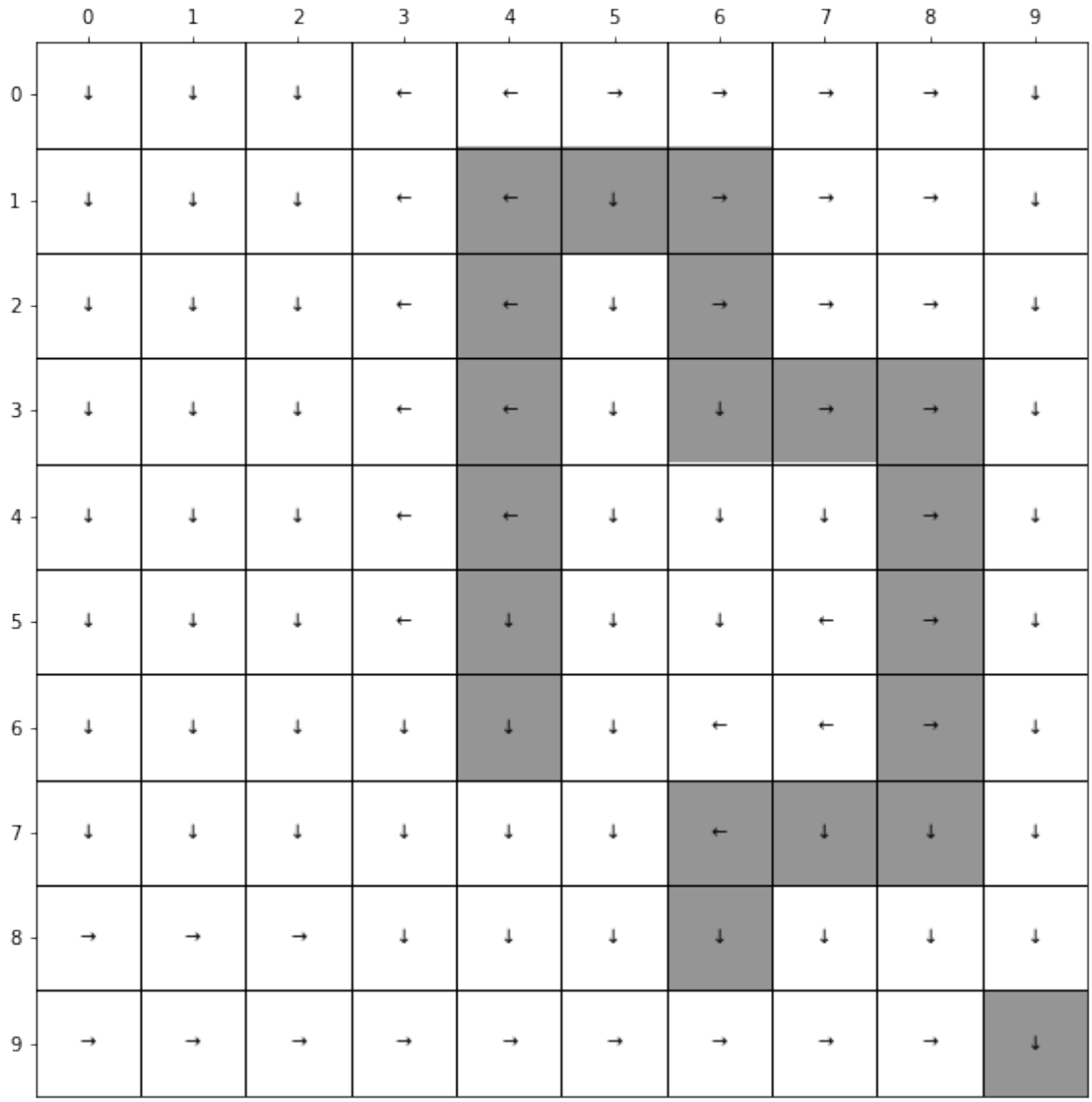
[138]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
temp_r2 = R2.copy()
temp_r2[9, 9] = -100
im = ax.imshow(-100 * np.ones((10, 10)) - temp_r2, cmap="Greys", norm=colors.
 ↪CenteredNorm(0))
```

```
ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)

# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if pi2[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif pi2[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q24-gt.png')
```



[137]:
```
from matplotlib import colors

fig, ax = plt.subplots(figsize=(10,10))
temp_r = R2.copy()
temp_r[9, 9] = -100
im = ax.imshow(-100 * np.ones((10, 10)) + np.abs(temp_r),cmap="Greys" , ␣
↪norm=colors.CenteredNorm(0))


ax.set_xticks(np.arange(10), minor=False)
ax.set_xticks(0.5 + np.arange(10), minor=True)
ax.set_yticks(np.arange(10), minor=False)
ax.set_yticks(0.5 + np.arange(10), minor=True)
```

```
# Loop over data dimensions and create text annotations.
for i in range(10):
    for j in range(10):
        if irl_pi2[i, j] == "top":
            text = ax.text(j, i, u'\u2191', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "bottom":
            text = ax.text(j, i, u'\u2193', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "left":
            text = ax.text(j, i, u'\u2190', ha="center", va="center",␣
 ↪color="black")
        elif irl_pi2[i, j] == "right":
            text = ax.text(j, i, u'\u2192', ha="center", va="center",␣
 ↪color="black")

ax.tick_params(top=True, bottom=False,
               labeltop=True, labelbottom=False)

ax.grid(which="minor", color="black", linestyle='-', linewidth=1)
ax.tick_params(which="minor", bottom=False, left=False)

plt.savefig('plots/q24-irl.png')
```



30 Question 25

From the figure in question 23, you should observe that the optimal policy of the agent has two major discrepancies. Please identify and provide the causes for these two discrepancies. One of the discrepancy can be fixed easily by a slight modification to the value iteration algorithm. Perform this modification and re-run the modified value iteration algorithm to compute the optimal policy of the agent. Also, recompute the maximum accuracy after this modification. Is there a change in maximum accuracy? The second discrepancy is harder to fix and is a limitation of the simple IRL algorithm.

[117]:
```
# recreate all parameters of env1
R1 = create_reward([(1, 4, -1), (1, 5, -1), (2, 4, -1), (2, 5, -1),
```

```
                    (2, 8, -1), (2, 9, -1), (3, 8, -1), (3, 9, -1),
                    (5, 2, -1), (5, 3, -1), (6, 2, -1), (6, 3, -1),
                    (9, 9, 1)])
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# do one more value iteration on recreated environment 1 to see if decreasing␣
 ↪epsilon helps
irl_V1_opt, _ = value_iteration(irl_env1, 0)
# fetch optimal policy
irl_pi1_opt = computation(irl_env1, irl_V1_opt)
# compare policies
print("epsilon = 0.01 -> accuracy =", compare_policies(pi1, irl_pi1))
print("epsilon = 0 -> accuracy =", compare_policies(pi1, irl_pi1_opt))
```

```
epsilon = 0.01 -> accuracy = 0.95
epsilon = 0 -> accuracy = 1.0
```

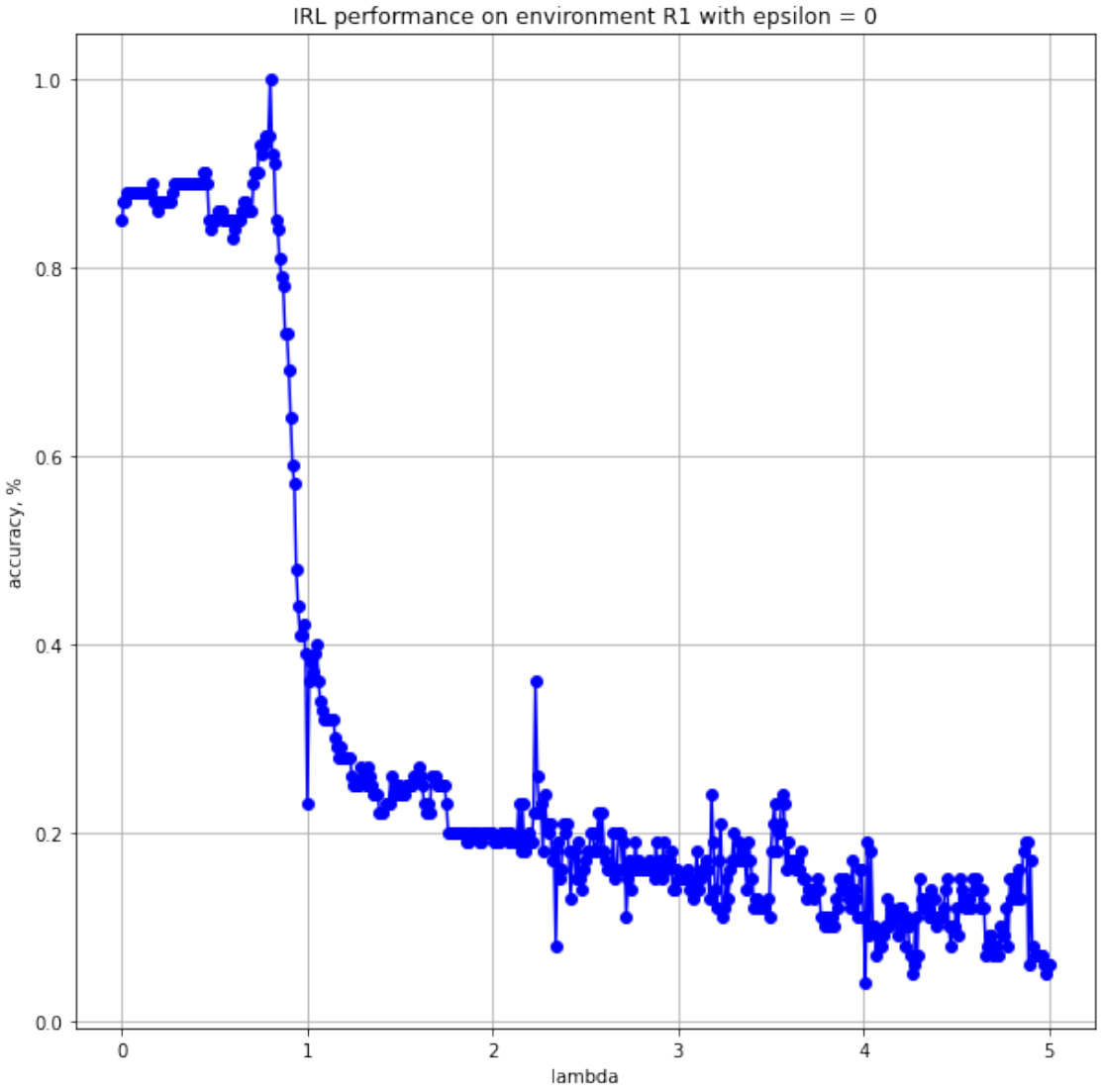
[118]:
```
# recreate all parameters of env1
R2 = create_reward([(4, 1, -100), (4, 2, -100), (4, 3, -100), (4, 4, -100), (4,␣
 ↪5, -100), (4, 6, -100),
                    (5, 1, -100),
                    (6, 1, -100), (6, 2, -100), (6, 3, -100), (6, 7, -100), (6,␣
 ↪8, -100),
                    (7, 3, -100), (7, 7, -100),
                    (8, 3, -100), (8, 4, -100), (8, 5, -100), (8, 6, -100), (8,␣
 ↪7, -100),
                    (9, 9, 10)])
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0.01

# do one more value iteration on recreated environment 1 to see if decreasing␣
 ↪epsilon helps
irl_V2_opt, _ = value_iteration(irl_env2, 0)
# fetch optimal policy
irl_pi2_opt = computation(irl_env2, irl_V2_opt)
# compare policies
print("epsilon = 0.01 -> accuracy =", compare_policies(pi2, irl_pi2))
print("epsilon = 0 -> accuracy =", compare_policies(pi2, irl_pi2_opt))
```

```
epsilon = 0.01 -> accuracy = 0.97
epsilon = 0 -> accuracy = 0.97
```

[122]:
```
# declare number of lambda points to test over
NUM_POINTS = 500

# set all necessary constants for creating an environment
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0

# init the vector of accuracies
accs = np.zeros((NUM_POINTS+1))
# init lambdas to sweep through
lambdas = np.linspace(0, 5, num=NUM_POINTS+1)
# sweep through them
for i in range(lambdas.shape[0]):
    # compute reverse-engineered R
    irl_R = irl(env1, pi1, lambdas[i], np.max(np.abs(R1)))
    # create temporary RL environment
    irl_env = EnvironmentRL(S, A, P, irl_R, gamma)
    # do value iteration on it
    irl_V, _ = value_iteration(irl_env, eps)
    # fetch optimal policy
    irl_pi = computation(irl_env, irl_V)
    # compare policies and store accuracy
    accs[i] = compare_policies(pi1, irl_pi)
```

```
C:\Users\slavc\Desktop\UCLA\Courses\Spring_2021\232E\232E\venv_232E\lib\site-
packages\cvxpy\problems\problem.py:1267: UserWarning: Solution may be
inaccurate. Try another solver, adjusting the solver settings, or solve with
verbose=True for more information.
  warnings.warn(
```

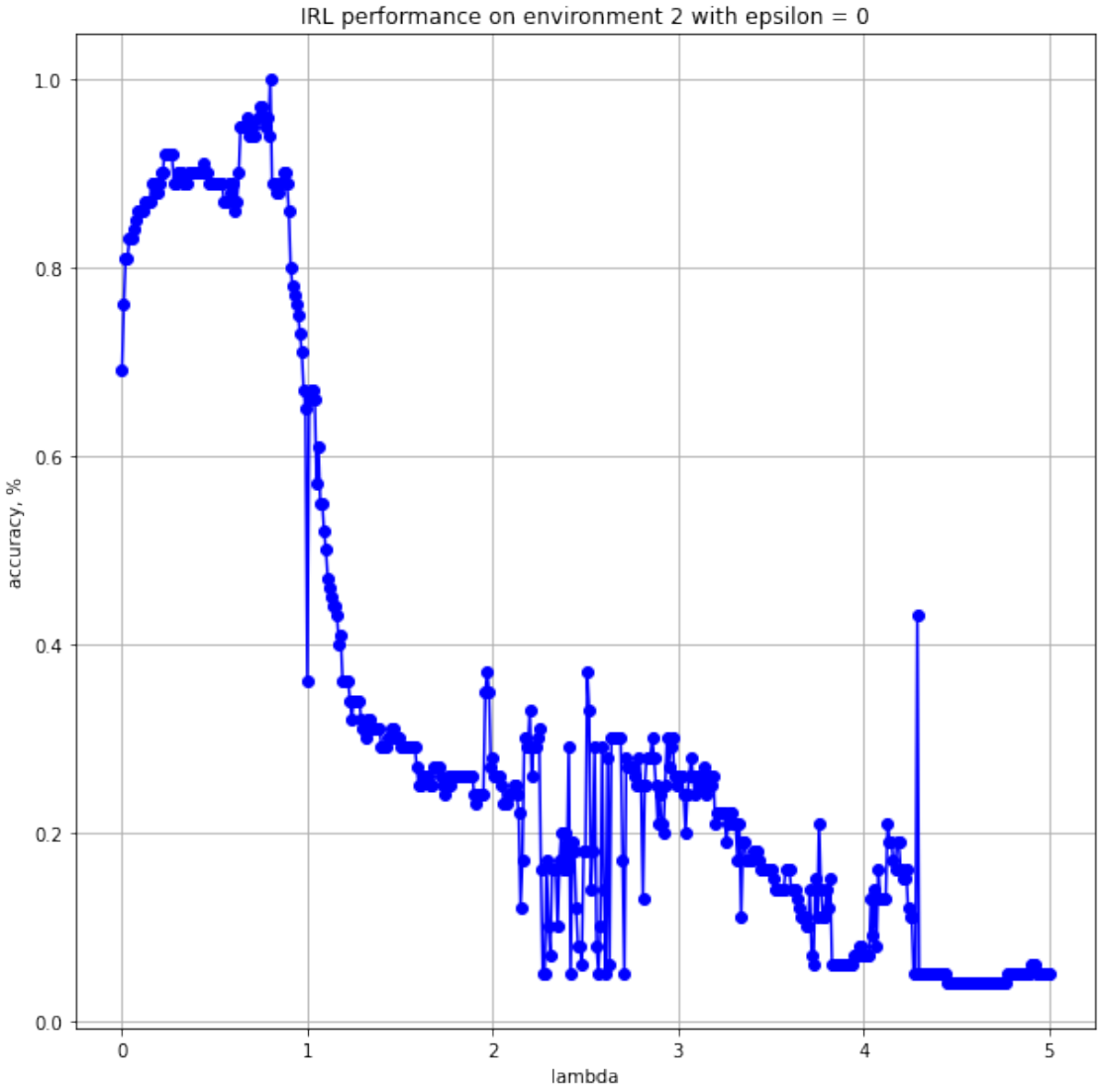
[124]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(lambdas, accs, "-ob")
ax.grid()
ax.title.set_text("IRL performance on environment R1 with epsilon = 0")
ax.set_xlabel('lambda')
ax.set_ylabel('accuracy, %')
plt.savefig('plots/q25-pi1.png')
```



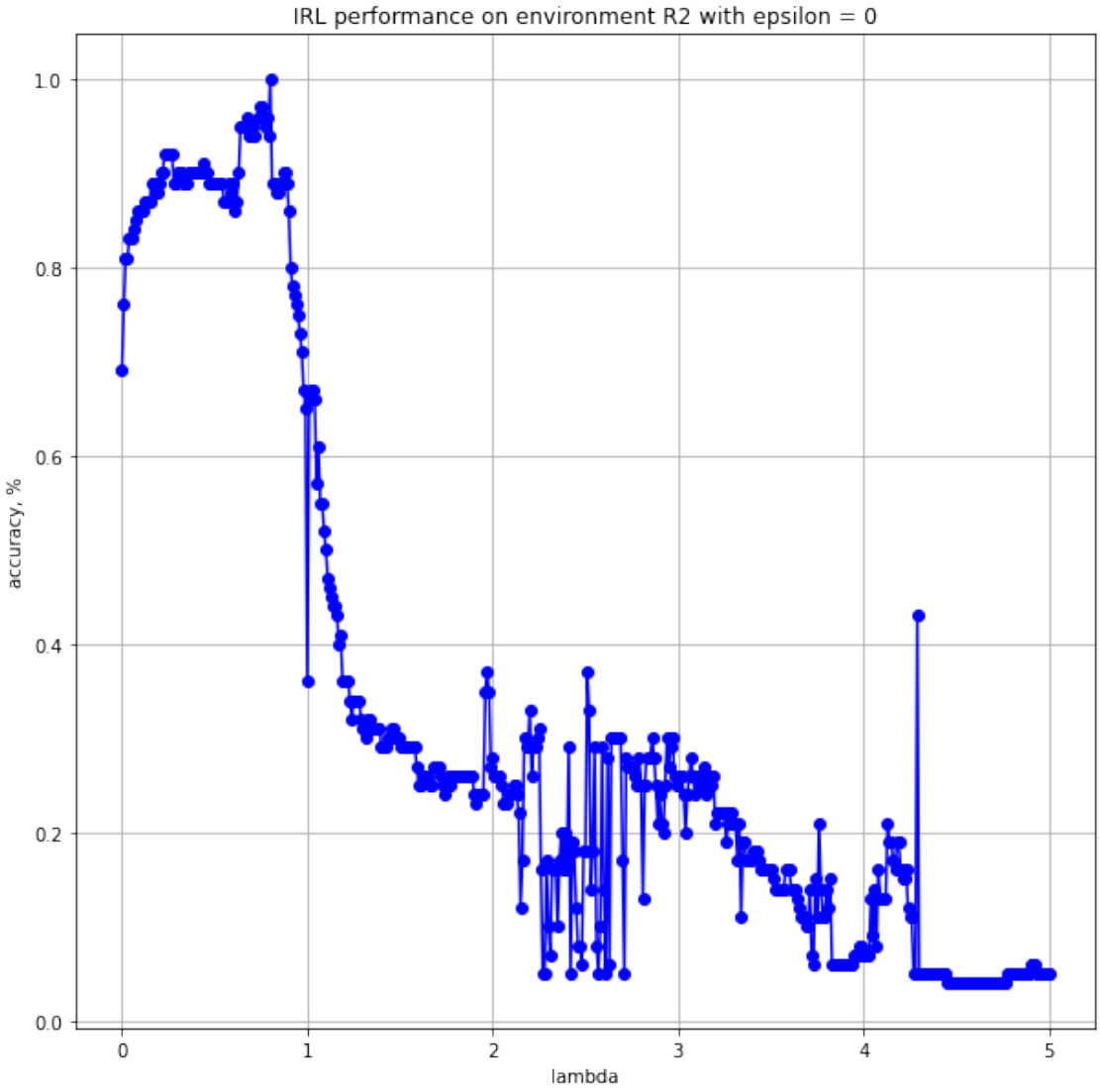
[120]:
```
# declare number of lambda points to test over
NUM_POINTS = 500

# set all necessary constants for creating an environment
S = np.arange(100).reshape((10, 10)).T
A = ["top", "bottom", "left", "right"]
w = 0.1
P = transition_probabilities(w)
gamma = 0.8
eps = 0

# init the vector of accuracies
accs2 = np.zeros((NUM_POINTS+1))
```

```
# init lambdas to sweep through
lambdas = np.linspace(0, 5, num=NUM_POINTS+1)
# sweep through them
for i in range(lambdas.shape[0]):
    # compute reverse-engineered R
    irl_R2 = irl(env2, pi2, lambdas[i], np.max(np.abs(R2)))
    # create temporary RL environment
    irl_env2 = EnvironmentRL(S, A, P, irl_R2, gamma)
    # do value iteration on it
    irl_V2, _ = value_iteration(irl_env2, eps)
    # fetch optimal policy
    irl_pi2 = computation(irl_env2, irl_V2)
    # compare policies and store accuracy
    accs2[i] = compare_policies(pi2, irl_pi2)
```

```
C:\Users\slavc\Desktop\UCLA\Courses\Spring_2021\232E\232E\venv_232E\lib\site-
packages\cvxpy\problems\problem.py:1267: UserWarning: Solution may be
inaccurate. Try another solver, adjusting the solver settings, or solve with
verbose=True for more information.
  warnings.warn(
```



[121]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(lambdas, accs2, "-ob")
ax.grid()
ax.title.set_text("IRL performance on environment R2 with epsilon = 0")
ax.set_xlabel('lambda')
ax.set_ylabel('accuracy, %')
plt.savefig('plots/q25-pi2.png')
```


IRL performance on environment R2 with epsilon = 0
accuracy, %
lambda
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5