EE 232E Project 3

Reinforcement learning
And
Inverse Reinforcement learning

Wei DU
UID: 005024944
Email: ericdw@g.ucla.edu

Xiao Yang
UID: 104946787
Email: avadayang@icloud.com

Fangyao Liu
UID:204945018
Email:fangyaoliu@g.ucla.edu

Ruchen Zhen
UID:205036408
Email:rzhen@ucla.edu

2 Reinforcement Learning(RL)

The basic idea of reinforcement learning is that given states, actions, transitional probabilities, discount factor and reward function, we need to find out the maximum expected state value.

Q1: For visualization purpose, first we need to generate heat maps of Reward function 1 and Reward function 2.

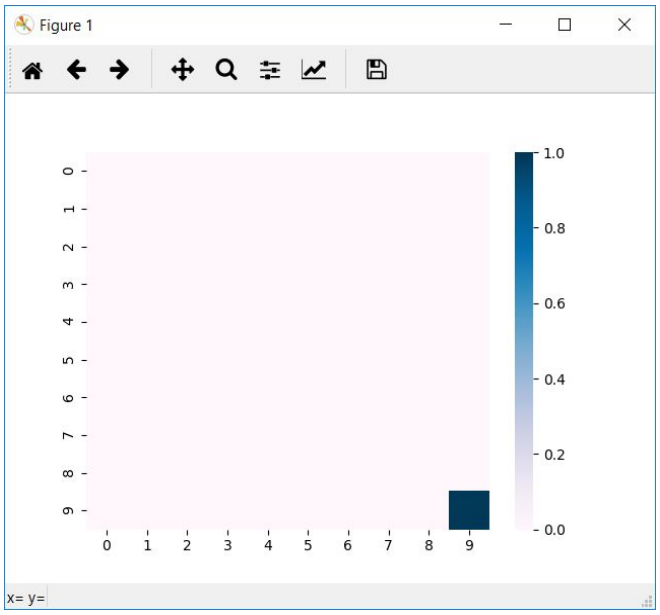

Figure Q1: reward function1

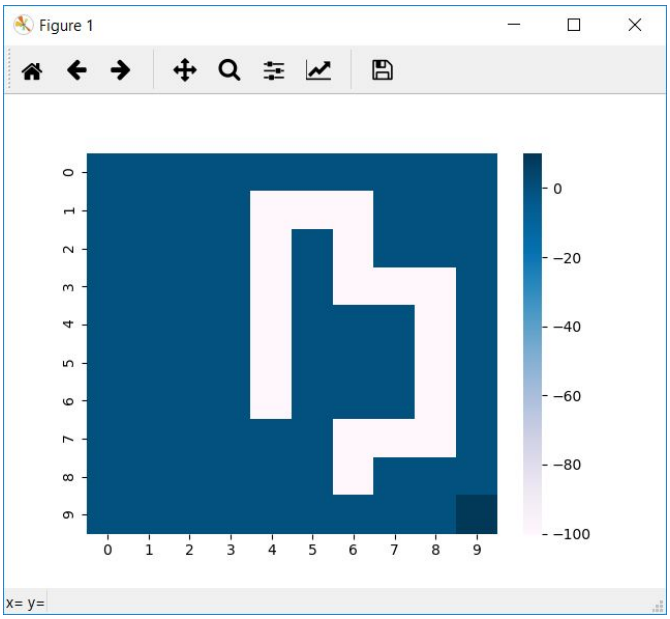

Figure Q1: reward function 2

3 Optimal policy learning using RL algorithms

In this section, we are going to apply value iteration algorithm to find out the optimal policy of different reward functions given states, actions, transitional probabilities and discount factor.

Q2: we will first implement the Initialization and Estimation steps of the Value Iteration algorithm. In estimation step, we take epsilon to be 0.01. There are usually two ways when we perform state value update. One is synchronous update and another one is asynchronous update. Since two ways are going to return us approximately same result, here, we will adopt asynchronous update to optimize space complexity. After several iterations, we reach an optimal state value for each state. Then according to the value, we generate a 2D grid containing their state value in the grid. For visualization purpose, we only save two digits after the decimal point.

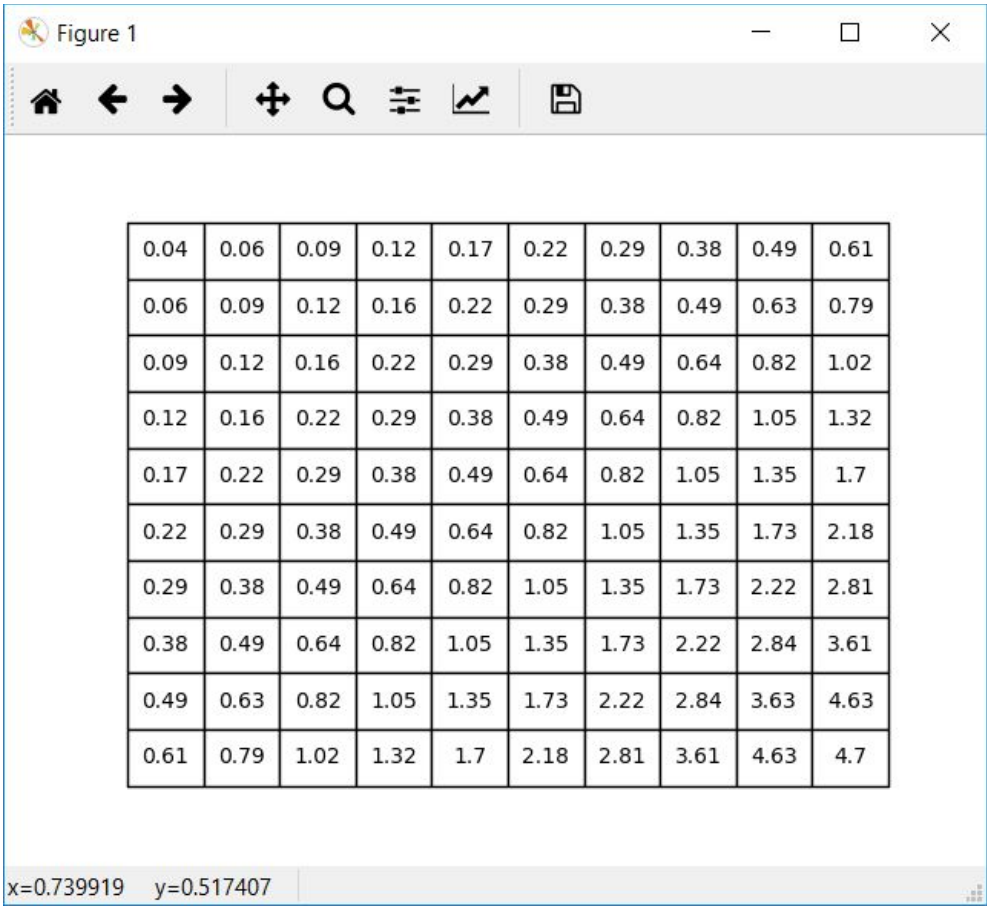

Figure Q2: optimal state value for reward function 1

Q3 Then we need to plot a heatmap given optimal state values above.

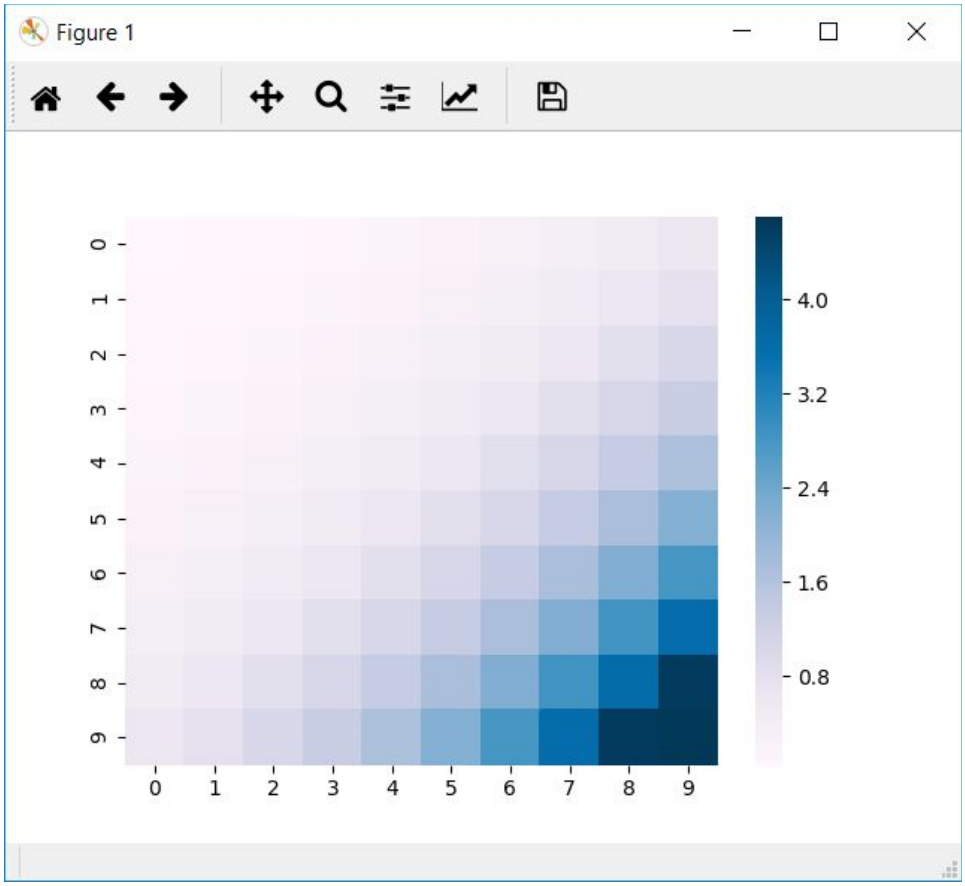

Figure Q3: heatmap of optimal state values

Q4 From heatmap above, we can find out the state values have such a distribution that largest optimal values appear at the right down corner of the plot and values gradually decrease when we move up or left from the right down corner.

This distribution does make sense because the definition of the optimal state value is total expected reward starting at this state and execute the policy. Since the reward function is positive only at the right down corner, so every state will take a policy that moves right or down to maximize its expected total value. However, there is still a discount factor and this factor will reduce the reward if this reward is achieved after many steps. So the distribution can be explained as states around right down corner have larger state values because they are near to the only state with positive reward. But when you are far away from that corner, your state value will start to decrease because of the discount factor.

Q5 As the last step of value iteration, we will compute the optimal policy of the agent navigating the state space. The plot of optimal policy is presented below:

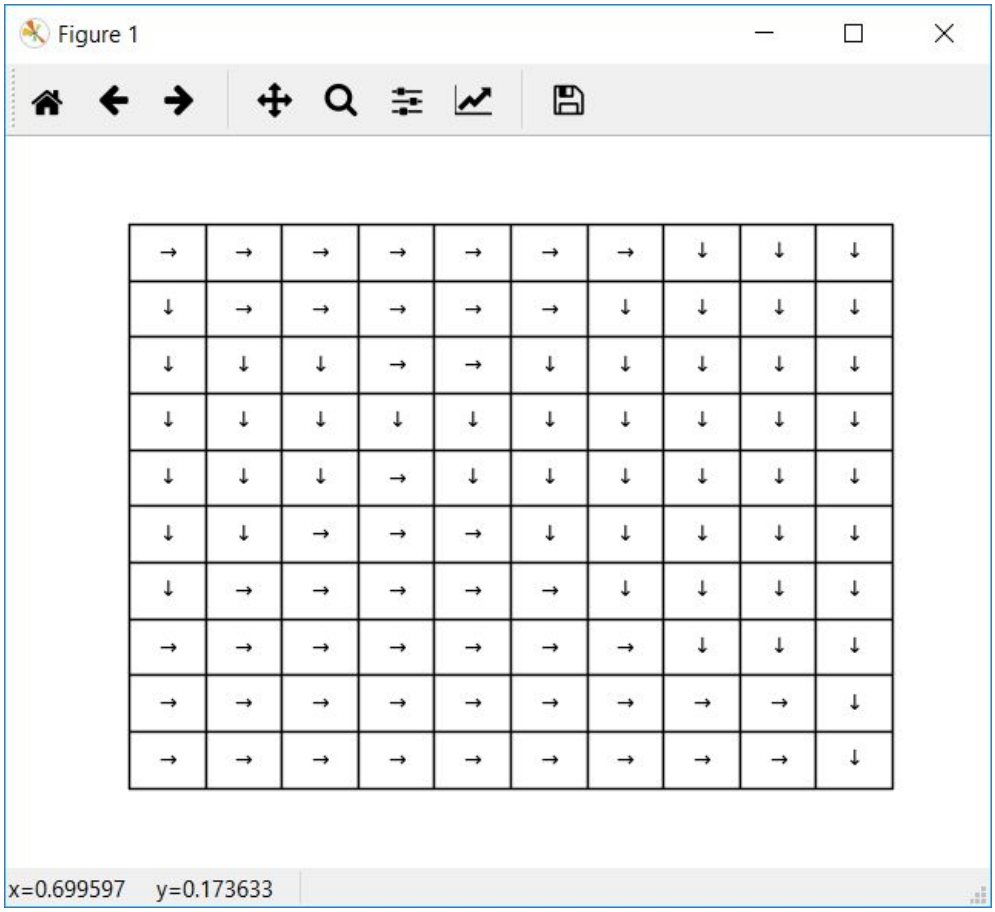

Figure Q5: optimal policy

This graph does match my intuition because the goal of our algorithm is to find the optimal policy that return us maximum expected reward. Since the state at the right down corner is the only one that has nonzero reward value, all the states will definitely move right or down.

Generally It is not enough for the agent to find out the optimal action just given optimal values of its neighbouring states. According to the computation step:

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

It can be observed that we should take action that return us the maximum sum of reward value and state value. That's why it is not enough just observing the optimal state if we want to find out the optimal action. But for this reward function, most of the reward values are equal to zero, so it might be possible for us to find out the optimal policy just by observing the optimal state value.

Q6 Replace reward function 1 with reward function 2. The optimal state value plot is presented below:

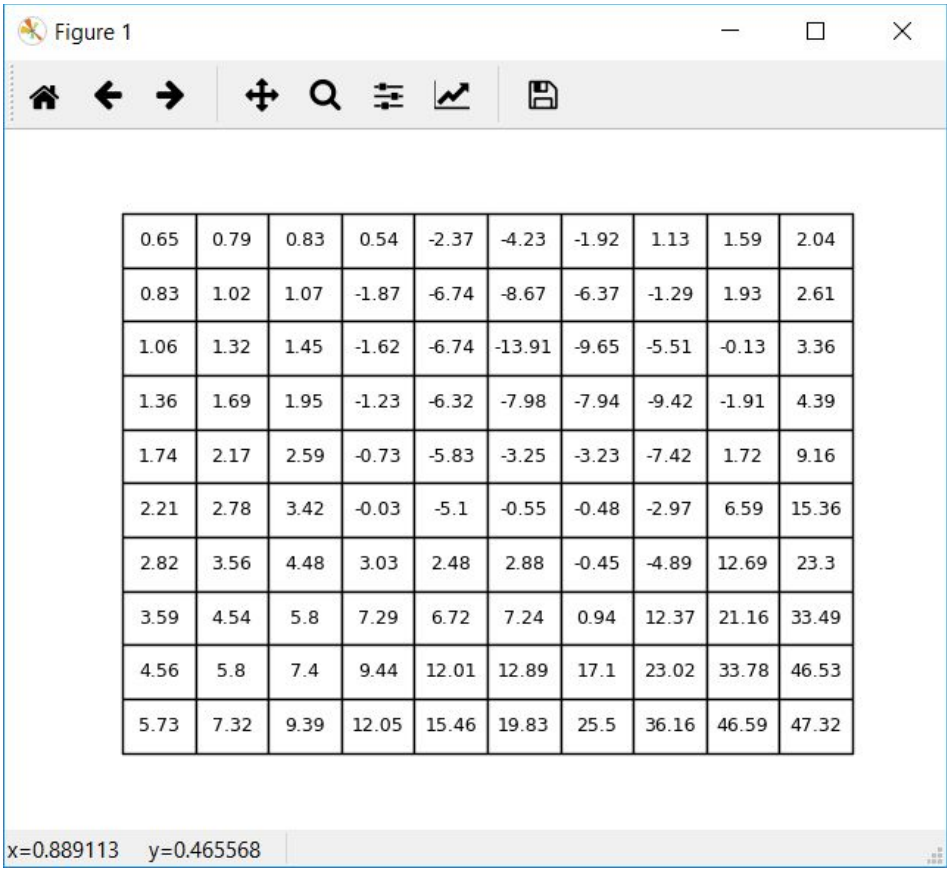

Figure Q6: optimal state value for reward function 2

For visualization purpose, we only save two digits after the decimal point

Q7: The heatmap of optimal state values is presented below:

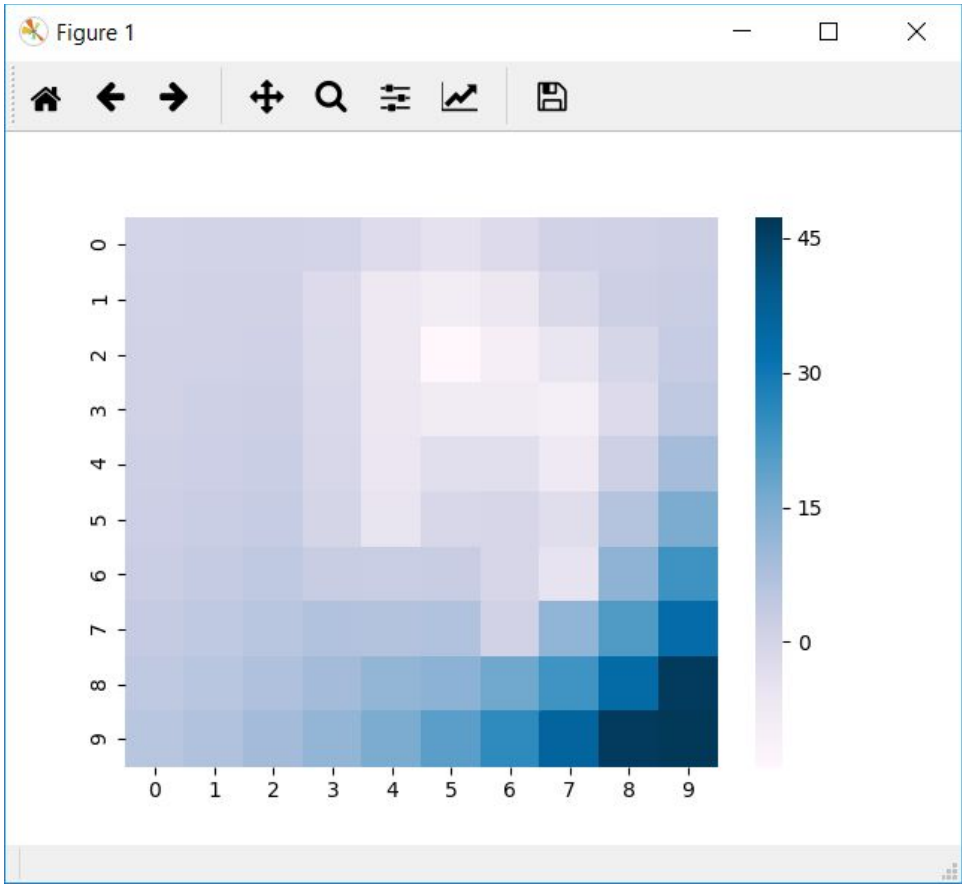

Figure Q7: heatmap of optimal state values

Q8:
Reward function 2 is completely different from reward function1. It not only has a positive value at the right down corner, but also a certain areas with negative values. As we mentioned before, the optimal state value is the maximum expected total payoff if starting at this state. Also in the iteration step, we update the state value using its neighboring values, reward functions and discount factors. Because of the two reasons above, we are going to have a distribution that:

1 The right down corner always has the maximum state value because of the location of the positive reward value. When distance from certain state to right down corner increases, the optimal state values, or we can say the effect of positive reward value, are going to decrease because of the discounting factor.

2 Because the influence of negative reward value, states that are near the negative reward values are going to have small or negative state values because there is always a possibility of transiting into states with negative values. And these negative reward values have exactly the opposite effect to other state values compared to the right down corner.

By superpositioning the following reasons, we are going to have such a distribution that states near right down corner are going to have positive large state value. States near negative reward values are going to have negative or small state values. When distance increases from certain state to the states with nonzero state values, the effect of reward function to the state value is going to decrease.

Q9: Now we perform computation step to optimal state value. We will have an optimal policy plot:

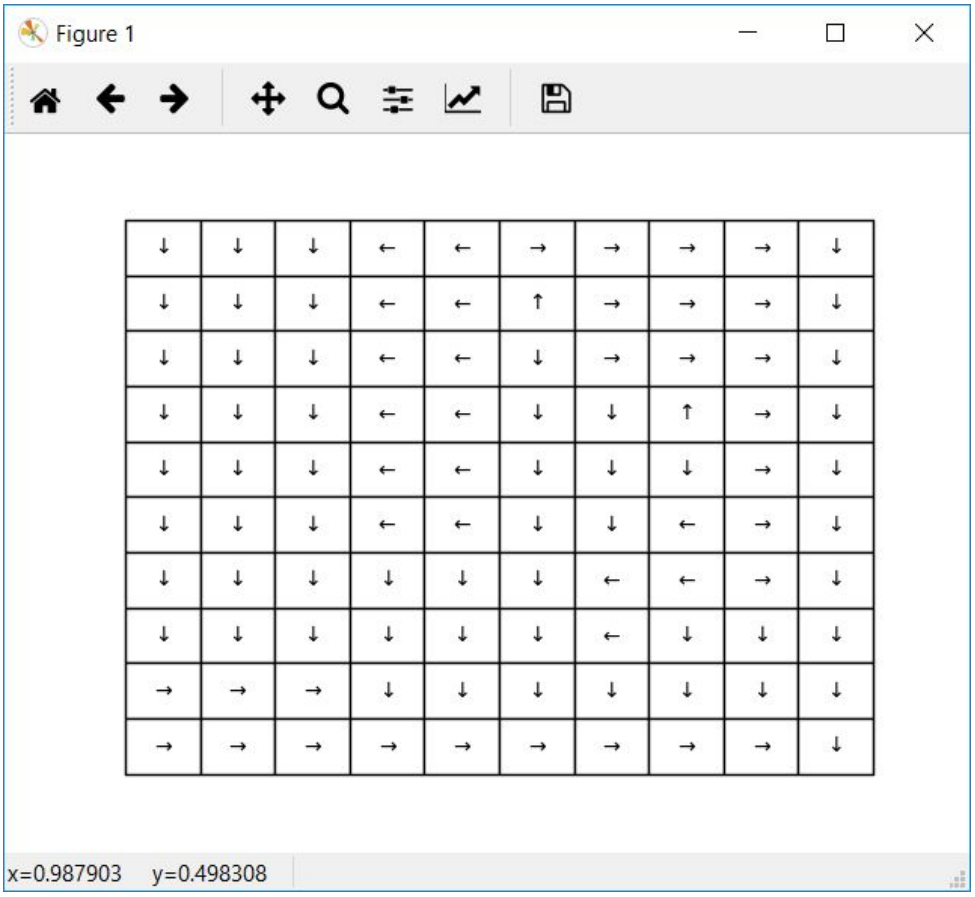

Figure Q9: optimal policy

This plot matches my intuition. Since the goal is to maximum the total expected value, then taking action towards positive reward function and avoids negative reward functions seem to be an intuitional and vague policy. As we can see here, the final policy plot exactly share the same idea with the intuitional ideas.

4 Inverse Reinforcement learning (IRL)

Inverse Reinforcement learning (IRL) is the task of learning an expert's reward function by observing the optimal behavior of the expert. This task can be accomplished in two ways:

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

In this part of the project, we will use IRL algorithm to extract the reward function and the second way is preferred because the reward function provides a much more parsimonious description of behavior. We will use the optimal policy computed in the previous section as the expert behavior and use the algorithm to extract the reward function of the expert. Then, we will use the extracted reward function to compute the optimal policy of the agent. We will compare the optimal policy of the agent to the optimal policy of the expert and use some similarity metric between the two to measure the performance of the IRL algorithm.

4.1 IRL algorithm

We will use the LP formulation in this project. The LP formulation of the IRL is given by equation 1

$$\begin{array}{ll} \underset{\mathbf{R},t_i,u_i}{\text{maximize}} & \sum_{i=1}^{|\mathcal{S}|}(t_i - \lambda u_i) \\ \text{subject to} & [(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R}] \geq t_i, \;\; \forall a \in \mathcal{A} \setminus a_1, \forall i \\ & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{R} \succeq 0, \;\; \forall a \in \mathcal{A} \setminus a_1 \\ & -\mathbf{u} \preceq \mathbf{R} \preceq \mathbf{u} \\ & |\mathbf{R}_i| \leq R_{max}, \;\; i = 1, 2, \cdots, |\mathcal{S}| \end{array} \tag{1}$$

we can recast the LP in equation 1 into an equivalent form given by equation 2 using block matrices.

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{D}\mathbf{x} \preceq 0, \;\; \forall a \in \mathcal{A} \setminus a_1 \end{array} \tag{2}$$

Q10 :

We rewrite these constraints to:

$$-(Pa1(i) - Pa(i))(I - \gamma Pa1)^{-1}\,\mathrm{R} + \mathrm{ti} \leq 0 \quad \text{.....①}$$

$$-(Pa1 - Pa)(I - \gamma Pa1)^{-1} R \leq 0 \ldots\ldots..②$$

$$-R - u \leq 0 \ldots\ldots..③$$

$$R - u \leq 0 \ldots\ldots..④$$

$$-Ri \leq Rm \ldots\ldots..⑤$$

$$Ri \leq Rm \ldots\ldots..⑥$$

In this case, we have 100 states and 4 actions.We can get that :

$$\mathbf{X} = [\ \mathbf{R}, \mathbf{t}, \mathbf{u}\ ]^T$$

Hence, we can get:

$$\mathbf{C}^T = [\mathbf{0}, \mathbf{1}, -\lambda]$$

Then we should use $D \cdot X \leq 0$ to contain all the constraints we listed above.

We define $(Pa1 - Pa)(I - \gamma Pa1)^{-1}$ as T(a,s)

$$①: \mathbf{D\_1 = [-T(A,S)\ a} \in A \backslash a1\ \mathbf{,\ 1\ ,\ 0]} \in R^{300x300}$$

300 is because in every state, we should consider all other 3 actions that is not the best one.

And we have 100 states hence we get 300 rows.

In ①, **1** is matrix that for every 3 rows, the 1 index of col will plus one.

$$②: \mathbf{D\_2 = [-T(A,S)\ a} \in A \backslash a1\ \mathbf{,\ 0\ ,\ 0]} \in R^{300x300}$$

$$③: \mathbf{D\_3 = [-I\ ,\ 0\ ,\ -I]} \in R^{100x300}$$

$$④: \mathbf{D\_4 = [I\ ,\ 0\ ,\ -I]} \in R^{100x300}$$

$$⑤: \mathbf{D\_5 = [-I\ ,\ 0\ ,\ 0]} \in R^{100x300}$$

$$⑥: \mathbf{D\_6 = [I\ ,\ 0\ ,\ 0]} \in R^{100x300}$$

So D is the matrix that vertically stack all these D_i matrix together.

We can see that in ⑤ and ⑥, we use another term Rm which is not included in matrix D, so we will put Rm in **b** to take them.

Hence ,

$$\mathbf{b} = [0\ldots\ldots\ldots..0,\ Rm\ldots\ldots..Rm] \in R^{1000x1}$$

In matrix b, we have 800 zeros and 200 Rm.

Then we can use cvxopt.solver.lp to solve this lp problem by imputing **C,D,b**

And all these matrices have been shown above.

4.2 Performance measure

In this project, we use a very simple measure to evaluate the performance of the IRL algorithm. We first get optimal policy found in previous section π. And we will solve the linear program given by equation 2 to extract the reward function; then, we will use the extracted reward function to compute the optimal policy π'. Finally, we do the entry wise comparison and get the difference of these two as D and have it divided by the number of entries, which is equivalent to the number of states, and get accuracy for this π'.

Q11 we used 500 different λ evenly spaced from 0 to 5 to generation 500 optimal policy π' and get the accuracy of each of them.

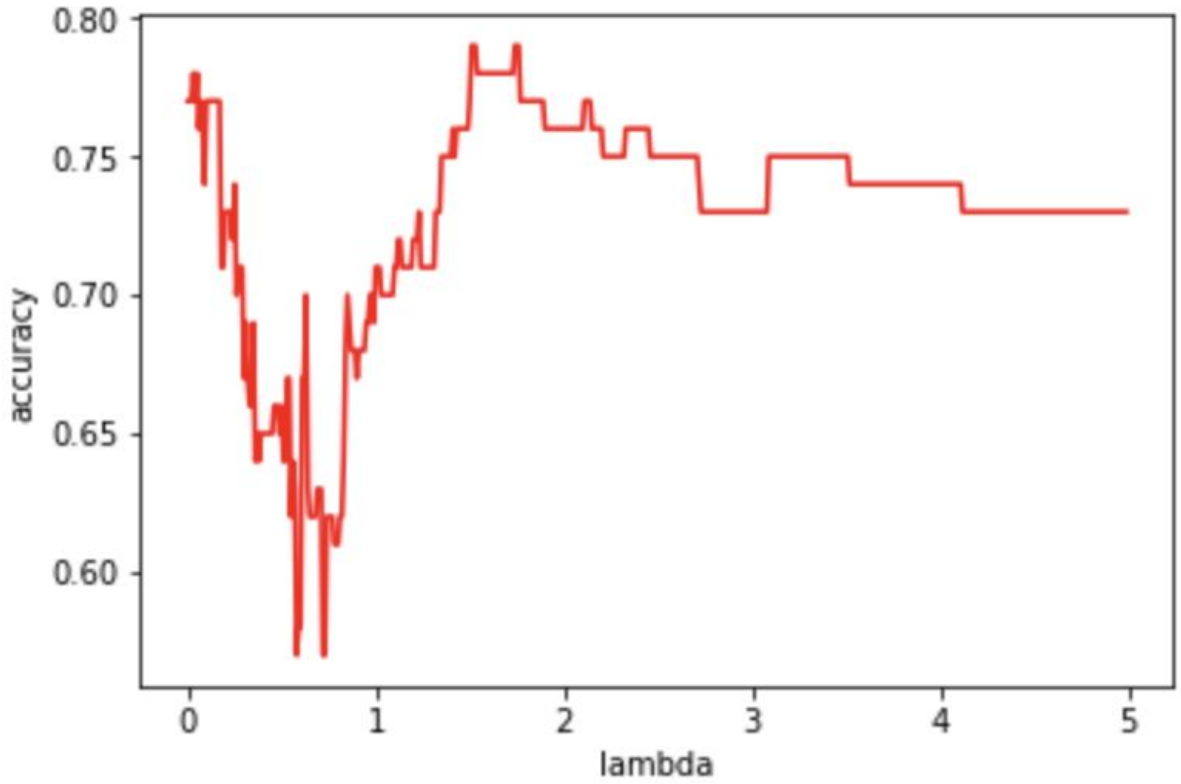

Figure q11 λ against Accuracy

From Q11 we noticed that we have six values of λ achieved max value, and here we only take the minimal one.

Q12 λmax = 1.51

We used reward function 1 as ground truth to compute optimal policy π, and then extract reward with λmax.

Q13

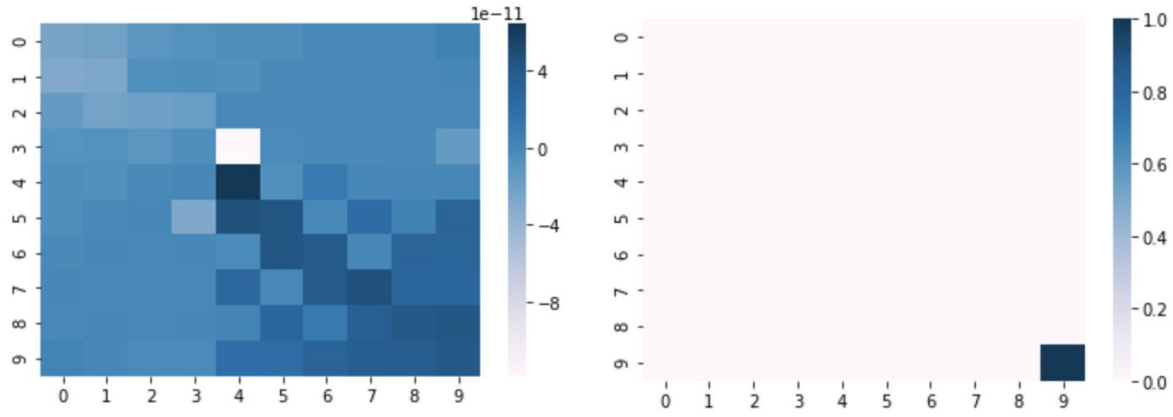

Figure q13 extracted reward (left) and ground truth (right)

With extracted reward function, we then get the optimal values of the states

Q14

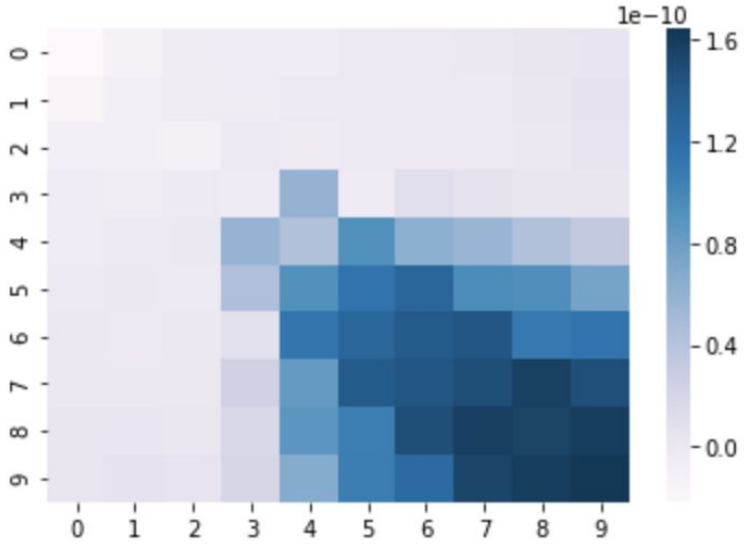

Figure q14 optimal values of states using extracted reward

Q15

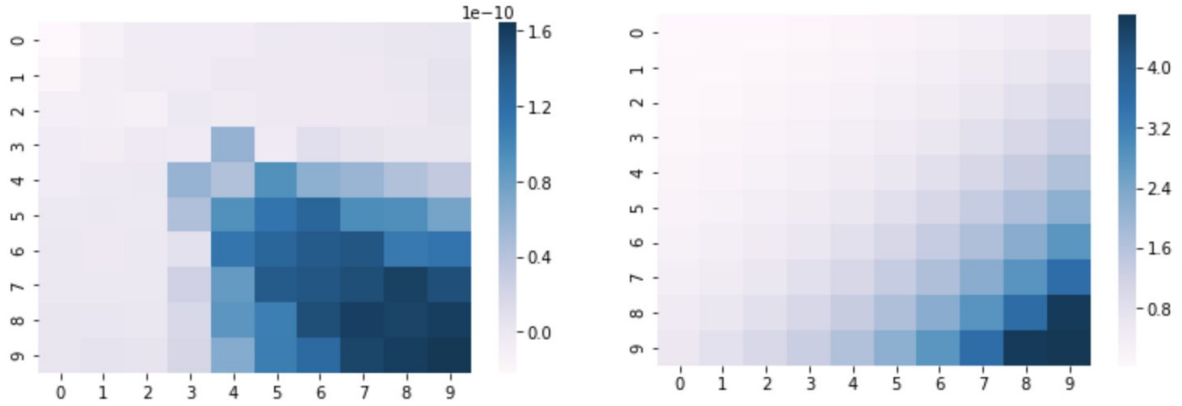

Figure q15-1 optimal state of value computed using extracted reward (left) and ground truth (right)

In comparing the above figures, an overall similarity between two can be seen: state value gradually increases from upper left corner diagonally to the the bottom right corner, which meets our expectation, since intuitively, the more closer to unit with high reward, the higher probability the walker can get to that unit and receive reward thus has high state value.

Difference is also obvious, 1) the one on the left has a bigger cluster of units with high value, which is explained by the same discrepancy noticed from reward function these two are based on. 2) one on the left doesn't have fine granularity of transition as the one on the right does and this can be explained by procedure in algorithm as shown below:

6: **while** $\Delta > \epsilon$ **do** ▷ Estimation
7: $\Delta \leftarrow 0$
8: **for all** $s \in \mathcal{S}$ **do**
9: $v \leftarrow V(s)$;
10: $V(s) \leftarrow \underset{a \in \mathcal{A}}{max} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
11: $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
12: **end for**
13: **end while**

Using extracted reward function, it meets the condition at line 6 to early before it gets truly converged. To prove that, we changed the condition of while loop to "not stop until number of iteration exceeds 1000 times" and then we get this:

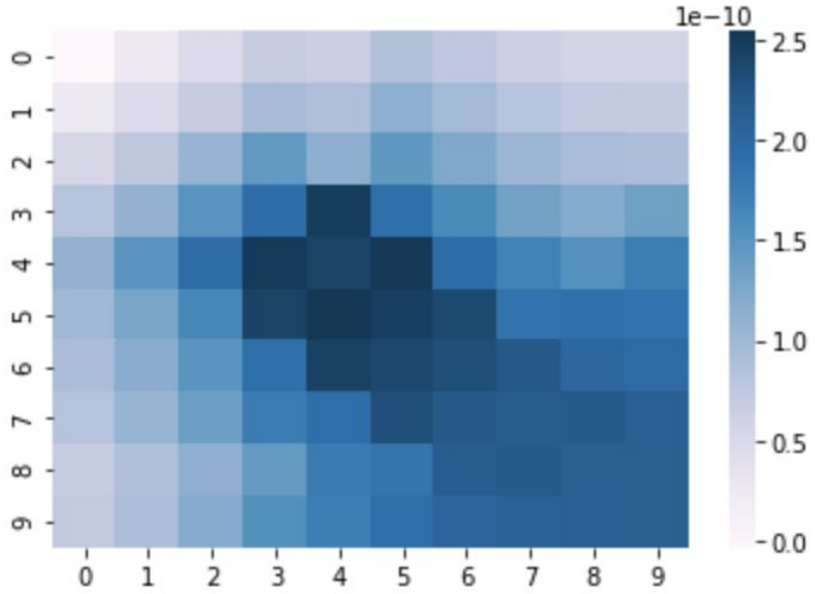

Figure q15-2 state of value computed using extracted reward with 1000 times of iteration

And now it has finer granularity of transition effect, which satisfies intuition and proves our hypothesis.

Then question becomes, why extracted reward function meets delta <= epsilon this fast? This can be explained by the fact that there are many units on retracted reward function that have negative value, and they are located close to the units have high value. So, when value updates from bottom try to affect units above, they meet this neutralization effect and thus decrease the influence to the units above.

Then we computed the optimal policy based on the extracted reward

Q16

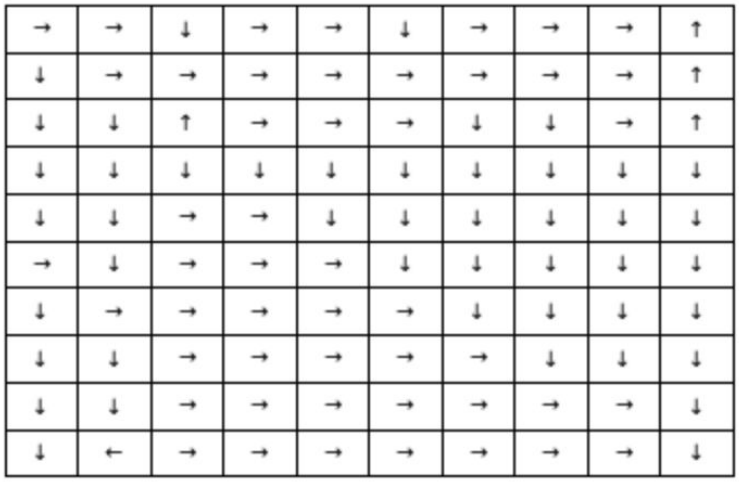

Figure q16 optimal policy based on extracted reward

Q17

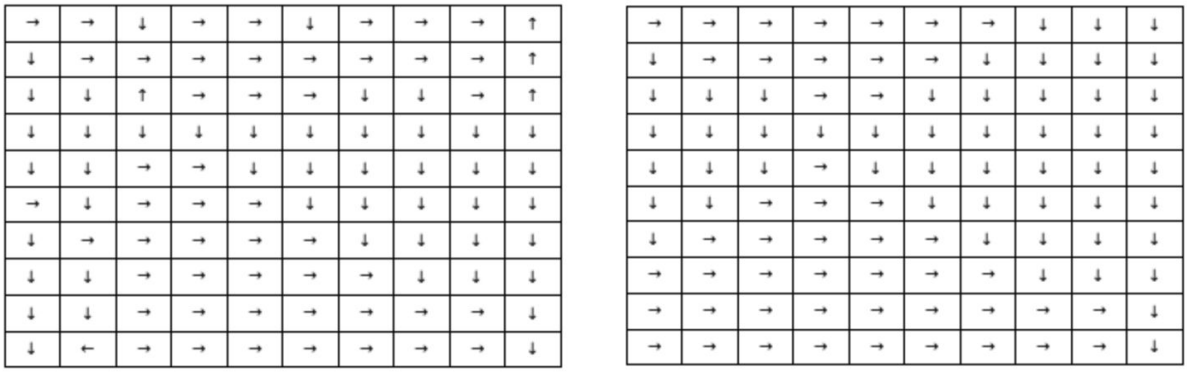

Figure q17-1 optimal policy based on extracted reward (left) and ground truth (right)

From figure above, we can see they both present that the walker tends to move to the bottom right corner. Since they run the same algorithm which prefers a walker getting more reward.

The difference, one on the left seems to show a more scattered pattern, eg there exists few arrows not following the general pattern, eg few of which points upwards which is not where high value units exist,. And this is because in computing those units, they are located in the 0-optimal-state-value zone, so they can point to any direction. And the reason why we have 0-optimal-state-value has same reason as we explained in Q15's answer.

When we fix it, we get result shown below, which proves our hypothesis. Now all arrows are guiding walker to units that have high value. Though destination shifts to the center a little, that is caused by reward function it is extracted from, which has high value at the center

→	→	↓	↓	↓	↓	↓	↓	↓	↓
↓	→	→	↓	↓	↓	↓	↓	↓	↓
↓	↓	↓	↓	→	↓	↓	↓	↓	↓
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
→	→	→	→	↓	←	←	↓	↓	↓
→	→	→	→	↑	←	←	←	↓	↓
→	→	→	→	↑	↑	←	←	↓	↓
→	→	→	→	↑	↑	↑	←	←	↓
→	→	→	→	↑	↑	↑	↑	→	→
→	→	→	→	→	→	→	↑	↑	↑

Figure q17-2 optimal policy based on extracted reward with 1000 iteration

Q18:

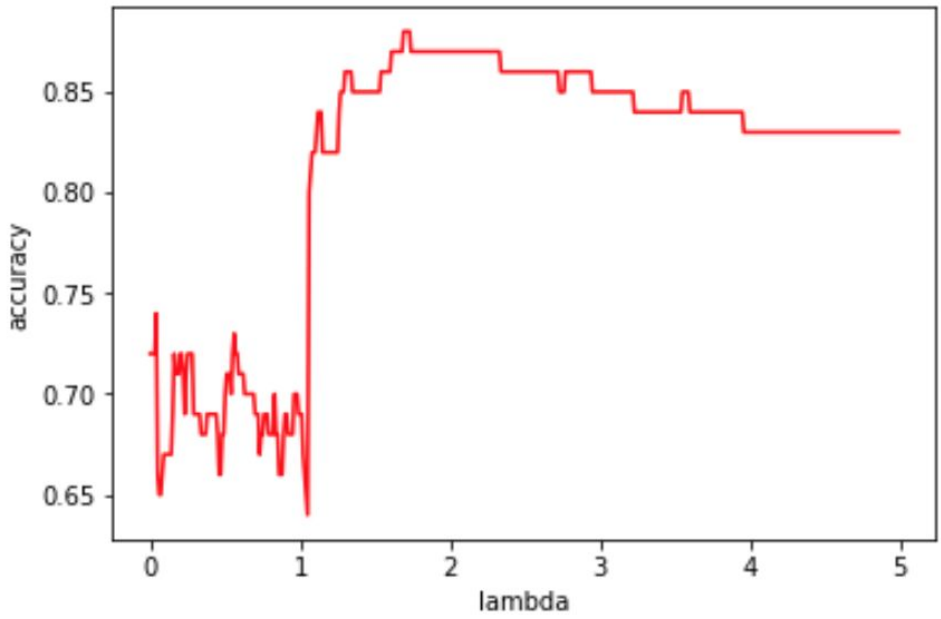

Figure Q18: λ against Accuracy Plot, IRL learning from reward function 2

Q19:

λ_max_2 = 1.69

Maximum accuracy: 0.88

Q20:

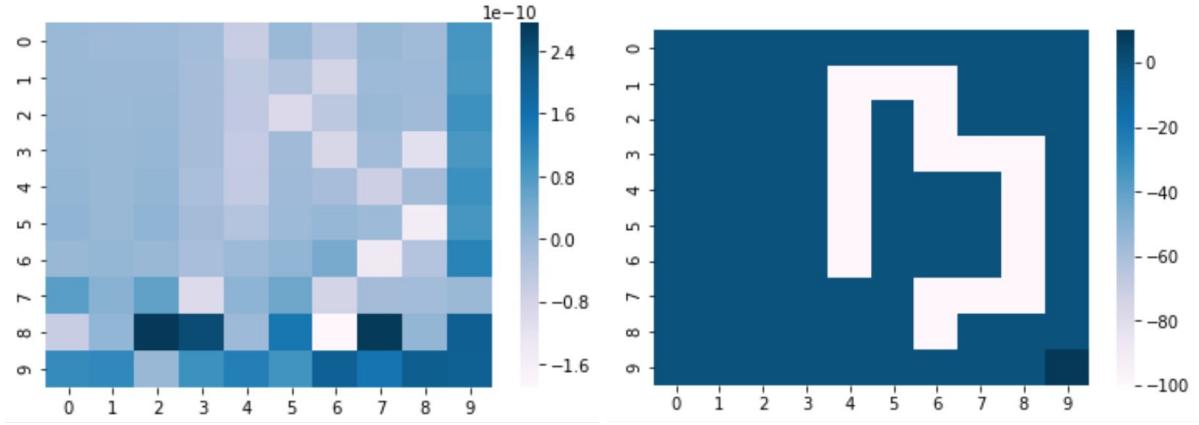

Figure Q20: Optimal state-value function computed using extracted reward function (left) and its ground truth function: reward function 2 (right).

Q21:

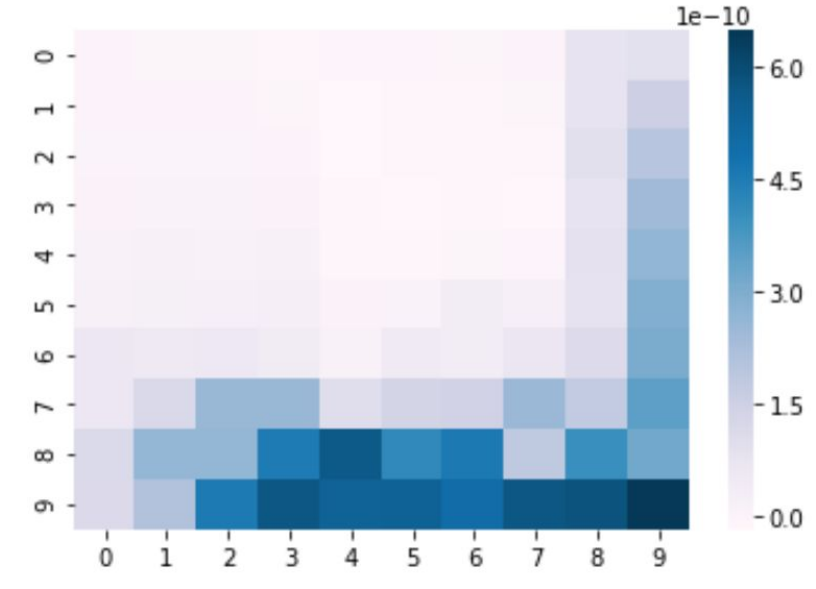

Figure Q21: Heatmap of optimal state value function, computed from the extracted reward function 2.

Q22

Comparing the optimal state value function of reward function 2 (Q7), with the optimal state value function of the extracted reward function 2 (Q21), we can see that the

overall tendency are similar, i.e. the bottom right corner is the darkest state, which means it has the largest expected reward, and the upper left part is light, which means lower expected reward. The difference is that, in the plot of Q7, the color lights gradually, while in the plot of Q22, the transition seems to be sharp.

Q23:

↓	←	↓	←	→	↑	→	→	→	↓
↓	↓	↓	←	←	↑	→	→	→	↓
↓	↓	↓	←	←	↓	→	→	→	↓
↓	↓	↓	←	←	↓	↓	↑	→	↓
↓	↓	↓	←	←	↓	↓	→	→	↓
↓	↓	↓	←	←	↓	↓	←	→	↓
↓	↓	↓	↓	↓	↓	←	↓	→	→
←	→	↓	↓	↓	↓	→	↓	↓	↓
→	→	→	↓	←	↓	↓	↓	↓	↓
→	→	→	↑	↓	→	→	→	→	↓

Figure Q23: Optimal policy calculated by optimal state value function computed from the extracted reward function 2.

Q24:

Compared with the figure in Q9, we can see that mostly the two figure are similar, i.e. in most states, the arrow points the same direction. The difference is that, in some places the agent might stay in the local optimum instead of the global optimum. For instance, in state 38 and 39, the arrows point each other which means, the agent will be very likely to stay in these two states instead of moving towards the global optimum, state 99.

Q25:

From figure from Q23 we noticed some units will not guide walker to destination that has max reward. Such as arrow on the first row pointing to upward. This can be explained by the 0-optimal-value-zone in the related optimal state value graph. And our fix is similar to what we did in Q15 and Q17, forcing procedure iterates more times in the while loop. Then we get modified policy as shown below:

↓	↓	↓	←	→	→	→	→	→	↓
↓	↓	↓	←	←	→	→	→	→	↓
↓	↓	↓	←	←	↓	→	→	→	↓
↓	↓	↓	←	←	↓	↓	→	→	↓
↓	↓	↓	←	↓	↓	↓	→	→	↓
↓	↓	↓	↓	↓	↓	↓	↓	→	↓
↓	↓	↓	↓	↓	↓	←	↓	→	↓
→	→	↓	↓	↓	↓	→	↓	↓	↓
→	→	→	←	←	←	→	↓	←	↓
→	→	↑	↑	←	→	→	↑	↓	←

And now policy only directs walker to units that have highest value.

Another issue is that, in some places the agent might stay in the local optimum instead of the global optimum. For instance, in state 38 and 39, the arrows point each other which means, the agent will be very likely to stay in these two states instead of moving towards the global optimum, state 99. This could be solved with random changing direction, similar with what the wind is doing.