Large-Scale Social and Complex Networks: Design and Algorithms
ECE 232E Summer 2018
Prof Vwani Roychowdhury
UCLA, Department of ECE

Project 3
Reinforcement learning and Inverse Reinforcement learning

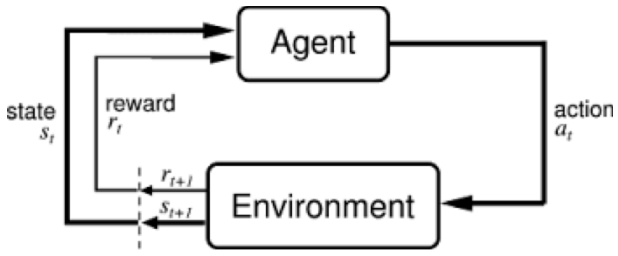
Due on Monday, August 6, 2018 by 11:59 pm

Team Members

Jennifer MacDonald, UID: 604501712
Nguyen Nguyen, UID: 004870721
Sam Yang, UID: 604034791

1 Introduction

In this project, we explored the concepts of Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL). Reinforcement Learning is an area of machine learning concerned how to obtain an objective or goal along a dimension over several steps. There are two parts to Reinforcement Learning: the agent, or the part of the algorithm that takes actions, and the environment, or the virtual space in which the agent moves. In this case, an action refers to all the moves that an agent can make at each step. The environment and the agent continue to interact with each other, with the environment changing based on the actions by presenting rewards and new states.



In the first part, we explored Reinforcement Learning in this project by calculating the optimal policy of an agent in a 2-D environment by implementing the value iteration algorithm. In the second part, we explored Inverse Reinforcement Learning (IRL), where we observed the optimal

policy of the agent and tried to derive the reward function in the context of apprenticeship learning.

2 Reinforcement learning (RL)

To understand how to answer the problems in this project, we took a closer look at the two main objects in Reinforcement Learning:

- Agent
- Environment

We accomplished this by looking at a single agent in a 2-D environment.

2.1 Environment

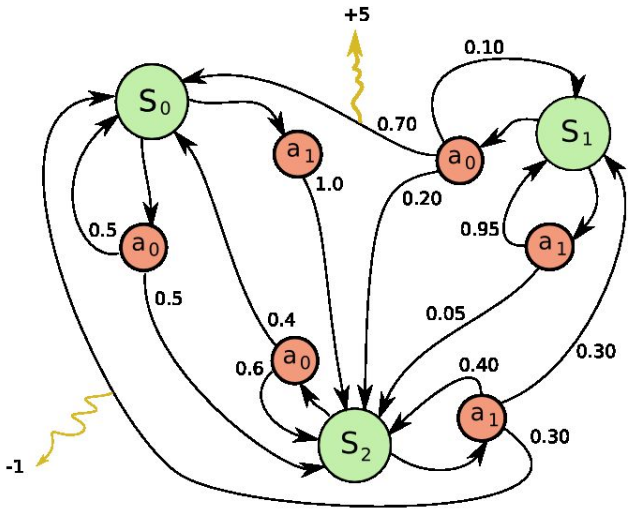
We assume that all of the environments follow the Markov Decision Process (MDP). MDP is a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. One important assumption in MDP is we have to assume that it's only looking at the present state, which means that the transition probabilities to the next state depends solely on the current state. Though, the current state should store enough information from its previous state. The second important assumption is that we assume the environment is stationary. MDP also has the concept of reward which tells how useful it is entering to that state. The goal is straightforward since its goal to get the highest reward from all possible outcomes. The core problem of MDP is to find a policy, typically a function π, that specifies the action $\pi(s)$ that the decision maker will choose when in state s.

In MDP, agents perform actions to change the state they are currently in based off of their current state. After taking an action, they are given some representation of the new state and some reward value associated with the new state. Thus, MDP can represented mathematically as a tuple $(S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$ where:

- S is a set of states
- A is a set of actions
- $P^a_{ss'}$ is a set of transition probabilities, where $P^a_{ss'}$ is the probability of transitioning from state $s \in S$ to state $s' \in S$ after taking action $a \in A$
    - $P^a_{ss'} = \mathrm{P}(s_{t+1} = s' | s_t = s, a_t = a)$
- Given any current state and action, s and a, together with any next state, s', the expected value of the next reward is $R^a_{ss'}$
    - $R^a_{ss'} = \mathrm{E}(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$

- $\gamma \in [0, 1)$ is the discount factor, and it is used to compute the present value of future reward
    - If γ is close to 1 then the future rewards are discounted less
    - If γ is close to 0 then the future rewards are discounted more

Below is an example of the MDP process, with the green nodes being specific states and the red nodes being the possible actions that can be done from each state.



We further looked more closely at the state space that the agent occupies below.

2.1.1 State space

For this project, we will consider the state space to be a 10x10 grid with 100 states, like the one shown below.

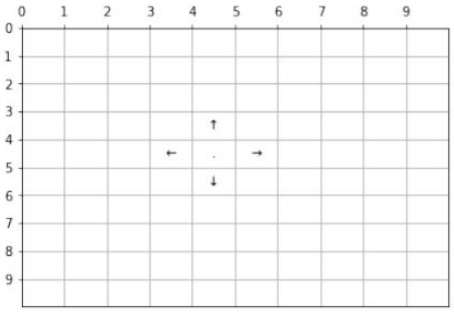
	0	1	2	3	4	5	6	7	8	9
0	0.0	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.0
1	1.0	11.0	21.0	31.0	41.0	51.0	61.0	71.0	81.0	91.0
2	2.0	12.0	22.0	32.0	42.0	52.0	62.0	72.0	82.0	92.0
3	3.0	13.0	23.0	33.0	43.0	53.0	63.0	73.0	83.0	93.0
4	4.0	14.0	24.0	34.0	44.0	54.0	64.0	74.0	84.0	94.0
5	5.0	15.0	25.0	35.0	45.0	55.0	65.0	75.0	85.0	95.0
6	6.0	16.0	26.0	36.0	46.0	56.0	66.0	76.0	86.0	96.0
7	7.0	17.0	27.0	37.0	47.0	57.0	67.0	77.0	87.0	97.0
8	8.0	18.0	28.0	38.0	48.0	58.0	68.0	78.0	88.0	98.0
9	9.0	19.0	29.0	39.0	49.0	59.0	69.0	79.0	89.0	99.0

2.1.2 Action set

The action set *A* for this project has four following actions:

- Move Right
- Move Left
- Move Up
- Move Down

Each one of these states can be seen in an example below.



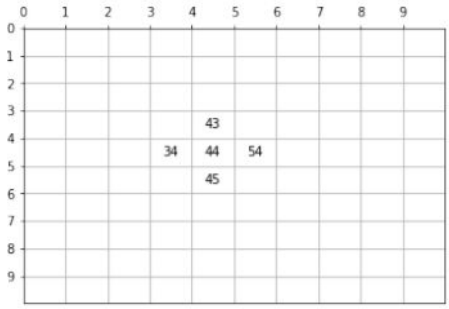
From the state in the center with the small dot, one can see the four actions that can be taken.

2.1.3 Transition probabilities

To calculate the transition probabilities from one state to another, we follow the following four rules:

1. If state *s'* and *s* are not neighboring states in the 2-D grid, then $P(s_{t+1} = s' | s_t = s, a_t = a) = 0$. If *s'* and *s* are neighbors in the 2-D grid, then you can make an action *a* from the action set *A*, where *s* has to be a neighbor of itself. Simply put, if the states are not next to each other, then the transition probability is 0, otherwise, a move can be made to neighbor *s'*.

2. Each action corresponds to a movement in the intended direction with probability 1 - *w*, but has a probability of *w* of moving in a random direction instead due to wind.



We can calculate the transition probabilities from *s*, which has a value of 44, to its neighbors.

$P(s_{t+1} = 43|s_t = 44, a_t = \uparrow) = 1 - w + w/4$

$P(s_{t+1} = 34|s_t = 44, a_t = \uparrow) = w/4$

$P(s_{t+1} = 54|s_t = 44, a_t = \uparrow) = w/4$

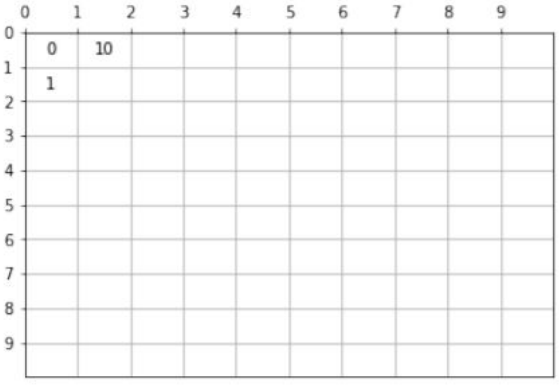
$P(s_{t+1} = 45|s_t = 44, a_t = \uparrow) = w/4$

In this case, the transitional probability is uniformly distributed for the neighbors of s. If the agent is at a non-boundary state, then the transition from its current place to itself would be 0 ($P(s_{t+1} = 44|s_t = 44, a_t = a) = 0$).

3. If the agent's state is at one of the four corners of the environment, the agent will stay at its current position if the next move would take it off of the environment. If the current state is in one of the four corners, it can perform two actions:

    - Action to move off the grid
    - Action to stay in the grid

    An example of this can be seen below.



The transition probabilities for moving off the grid are:

$P(s_{t+1} = 10|s_t = 0, a_t = \uparrow) = w/4$
$P(s_{t+1} = 1|s_t = 0, a_t = \uparrow) = w/4$
$P(s_{t+1} = 0|s_t = 0, a_t = \uparrow) = 1 - w + w/4 + w/4$

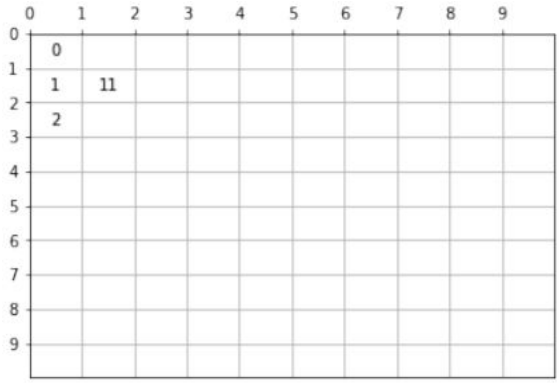
The transition probabilities for staying on the grid are:
$P(s_{t+1} = 10|s_t = 0, a_t = \rightarrow) = 1 - w + w/4$
$P(s_{t+1} = 1|s_t = 0, a_t = \rightarrow) = w/4$
$P(s_{t+1} = 0|s_t = 0, a_t = \rightarrow) = w/4 + w/4$

At a corner state, you can go off the grid in two out of four different directions. Therefore, $P(s_{t+1} = 0|s_t = 0, a_t = \uparrow) = w/4 + w/4 = w/2$.

4. If the agent's state is at one of the edges, the agent will stay at its current position of the next move will take it off of the environment. If the current state is at one of the edges, it can perform two actions:

    - Action to move off the grid
    - Action to stay in the grid

    An example of this can be seen below.



The transition probabilities for moving off the grid are:
$P(s_{t+1} = 0|s_t = 1, a_t = \leftarrow) = w/4$
$P(s_{t+1} = 11|s_t = 1, a_t = \leftarrow) = w/4$
$P(s_{t+1} = 2|s_t = 1, a_t = \leftarrow) = w/4$
$P(s_{t+1} = 1|s_t = 1, a_t = \leftarrow) = 1 - w + w/4$

The transition probabilities for staying on the grid are:
$P(s_{t+1} = 0|s_t = 1, a_t = \uparrow) = 1 - w + w/4$

$P(s_{t+1} = 11|s_t = 1, a_t = \uparrow) = w/4$
$P(s_{t+1} = 2|s_t = 1, a_t = \uparrow) = w/4$
$P(s_{t+1} = 1|s_t = 1, a_t = \uparrow) = w/4$

At a corner state, you can go off the grid in one out of four different directions. Therefore, $P(s_{t+1} = 1|s_t = 1, a_t = \uparrow) = w/4$.

The difference between a corner state and an edge state is that a corner state has two neighbors, but an edge state has three.

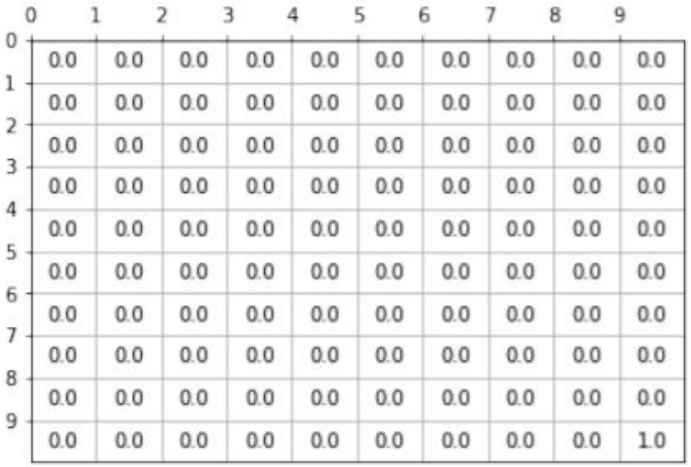
2.1.4 Reward function

For this project, we assumed that the reward function is independent of the current state *s* and the action that is taken at that state *a*, where the reward function depends only on the state that you transition to *s'*. Therefore, $R^a_{ss'} = R(s')$. We explore the optimal policies more in depth by using two reward functions:

- Reward function 1
- Reward function 2

The figures for Reward function 1 and 2 are shown below:

Plot of Reward Function 1



Plot of Reward Function 2

	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

Question 1: (10 points) 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

To create the heat maps, we manually added the values for each state and then plotted a heat map using pyplot. The heat maps Reward functions 1 and 2 are shown below in Figures 1 and 2, respectively.

Figure 1: Heat map of Reward function 1

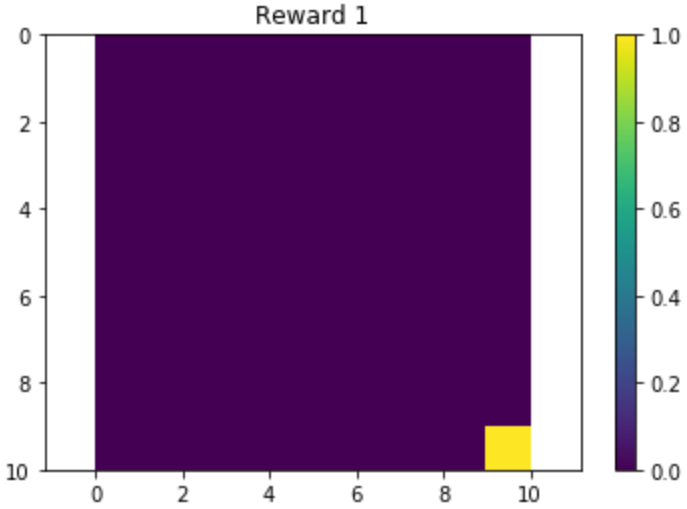
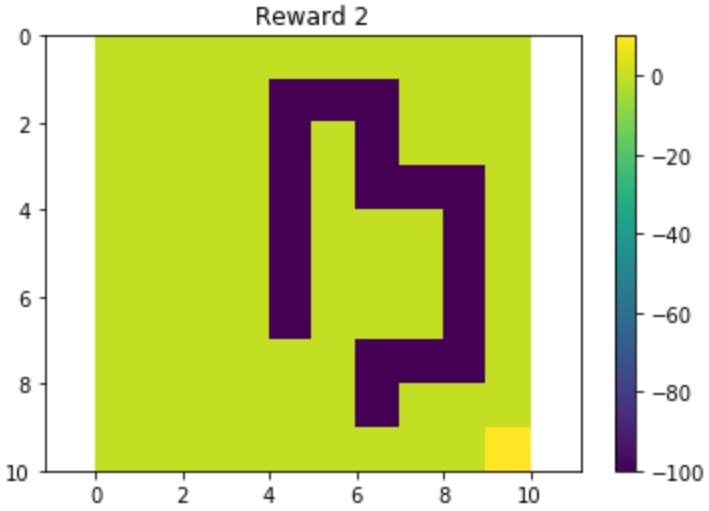


Figure 2: Heat map of reward function 2



To plot the heat maps, we used the pcolor function from pyplot:
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.pcolor.html

3 Optimal policy learning using RL algorithms

Now that we have our heat maps to reference, we can use the RL algorithm to find the optimal policy for each reward function. To do this, we had to:

- Find the optimal state-value or action-value
- Use the optimal state-value or action-value to determine the deterministic optimal policy

There are multiple algorithms to determine the optimal state-value such as value iteration, policy iteration, Q-learning, and Sarsa. However in this project, we will focus on the Value Iteration algorithm which can be seen below. Value iteration is a method of computing an optimal MDP policy and its value. Value iteration starts at the end and works backward. The policy function π is not used, but instead the value it uses the value π(s) is calculated within V(s) where it is needed. Thus, substituting the calculation π(s) into V(s) gives:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s,s') \left(R_a(s,s') + \gamma V_i(s') \right) \right\},$$

where i is the iteration. It will iterate for all states *s* until V converges with the left-hand side equal to the right-hand side. This is also called the Bellman equation. Below is the pseudocode implementation of Value Iteration algorithm .

1: **procedure** VALUE ITERATION($\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$, $\mathcal{S}$, $\mathcal{A}$, γ):
2: **for all** $s \in \mathcal{S}$ **do** ▷ Initialization
3: $V(s) \leftarrow 0$
4: **end for**
5: $\Delta \leftarrow \infty$
6: **while** $\Delta > \epsilon$ **do** ▷ Estimation
7: $\Delta \leftarrow 0$
8: **for all** $s \in \mathcal{S}$ **do**
9: $v \leftarrow V(s)$;
10: $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
11: $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
12: **end for**
13: **end while**
14: **for all** $s \in \mathcal{S}$ **do** ▷ Computation
15: $\pi(s) \leftarrow \arg \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
16: **end for**
17: **end procedure** return π

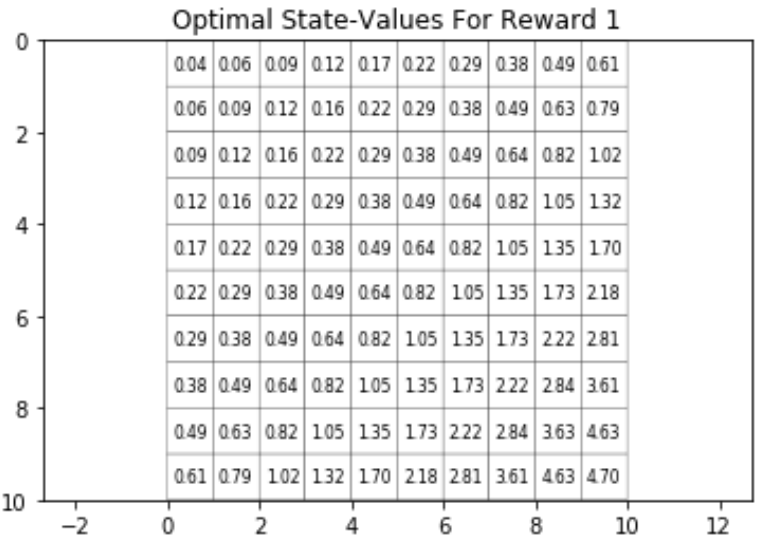
Question 2: (40 points) Create the environment of the agent using the information provided in section 2. To be specific, create the MDP by setting up the state-space, action set, transition probabilities, discount factor, and reward function. For creating the environment, use the following set of parameters:

- *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 = 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 question, you should have 1 plot.

We reproduced the above Value Iteration algorithm above to estimate the optimized values using the required values specified for Reward function 1. Then, we plotted the optimal value as seen below in Figure 3.

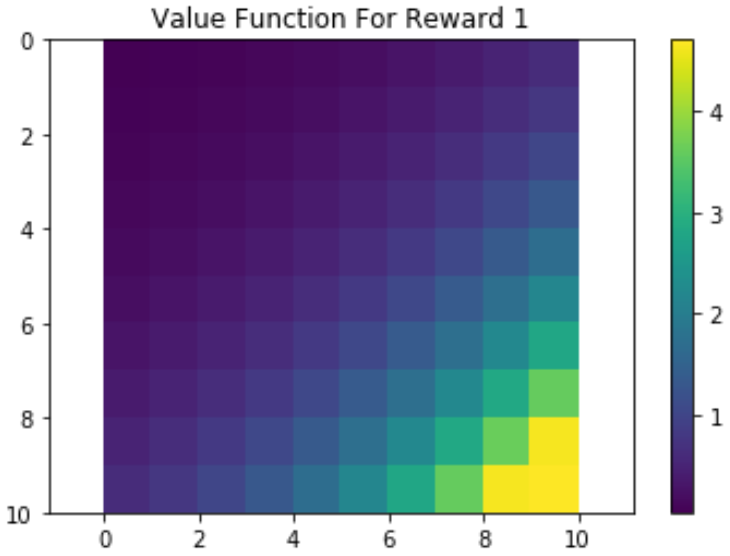
Figure 3: Optimal State-Values for Reward 1



Question 3: (5 points) 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).

We generated a heat map of the optimal state values for Reward function 1 as seen below in Figure 4.

Figure 4: Value function for Reward 1



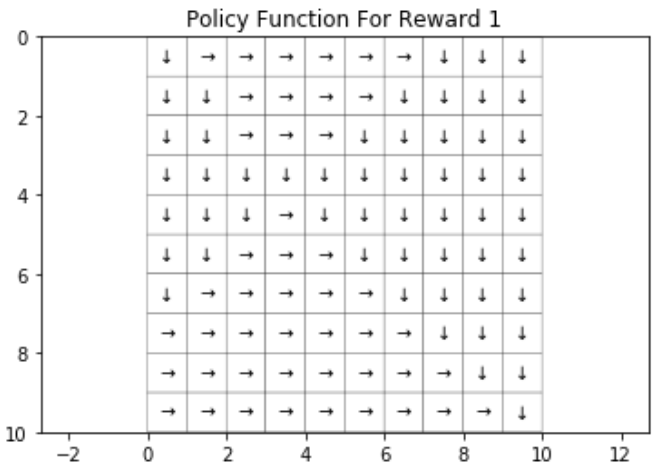
Question 4: (15 points) Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in question 3 to explain)

The optimal state values in the 2-D grid represent the expected discounted cumulative reward an agent will receive upon being in some state. Since the only reward is at the bottom right corner, it makes sense that the state at the very bottom right has the highest magnitude. Since the neighboring states can transition to this high-value state in a small number of steps, they will also have a high magnitude, only slightly diminished since the discount factor (0.8) causes the future reward to be smaller. Following similar logic, the more steps it takes for the agent to get to the reward, the smaller the value will be. Since the upper left state is the furthest away from the reward, it has the smallest magnitude. The distribution in the 2-D map indicates that the further away the state is from the reward, the smaller the value will be. This is consistent with the idea of a value representing the expected cumulative discounted reward.

Question 5: (30 points) 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 its neighboring states? In this question, you should have 1 plot.

To plot the optimal policy of the agent, we had to plot the the unicode arrows in a numpy character array according to the values obtained from the value-iteration function. We then plotted those arrow values onto a blank heatmap grid, as seen below in Figure 5.

Figure 5: Policy Function for Reward 1

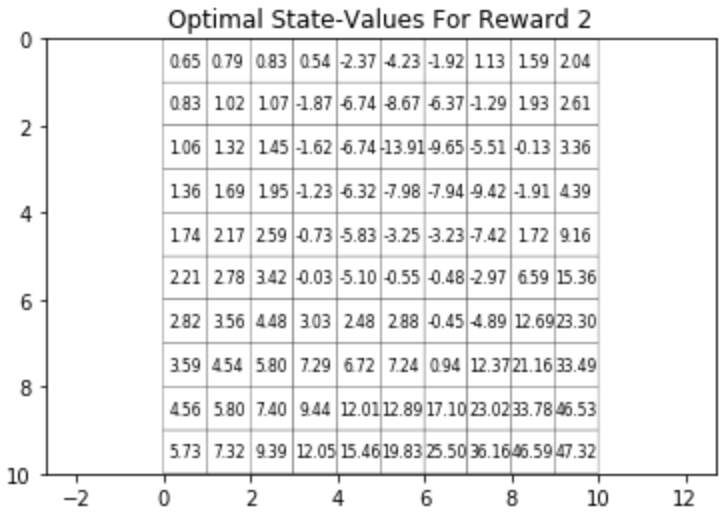


The optimal policy of the agent seems to match our intuition. This is because the arrows always follows the value function reward heatmap seen in Figure 4, where the arrows direct us to the hottest part of the map tend to follow an optimal route. This is because they want to take optimal actions to get higher rewards. Therefore, it is possible for the agent to compute the optimal action to take at each state by observing the optimal values of its neighboring states. The actions chosen are always either down or to the right, which makes sense since the only reward is at the bottom right.

Question 6: (10 points) 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.

We replaced the environment with that of Reward function 2 and calculated the optimal-state value function with the same parameters. The plot of the optimal value map is shown below in Figure 6.

Figure 6: Optimal State-Values for Reward 2



Question 7: (10 points) 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.

We generated a heat map of the optimal state values for Reward function 2 as seen in Figure 7 below.

Figure 7:Value function for Reward 2

Question 8: (20 points) Explain the distribution of the optimal state values across the 2-D grid. (Hint: Use the figure generated in question 7 to explain)

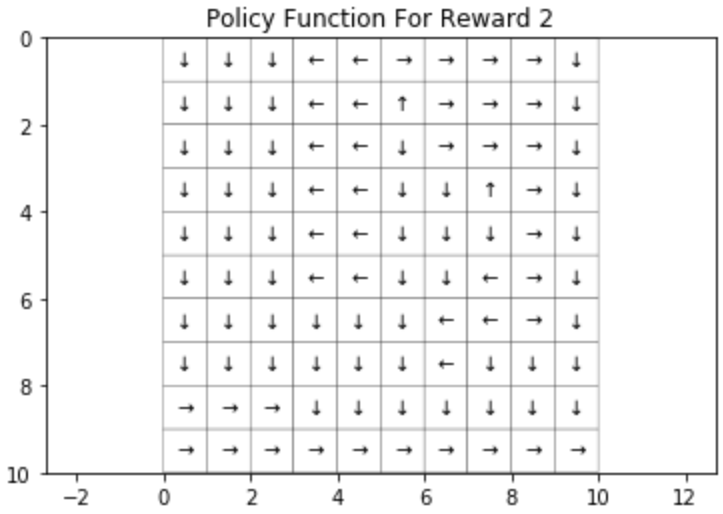
Similar to reward 1, the single positive reward is at the bottom right, so the optimal state values should increase as the states get closer to the bottom right. One difference is that there is a region

of negative optimal state values near the center of the 2-D map. This is caused by the negative rewards from reward 2. While the magnitudes of the negative rewards are -100, the lowest optimal state values are near -10. The reason for this is that the value is a function of the *next* state's reward plus the value of the greedily-chosen next state. So even if the agent is at one of the -100 states, the value is not nearly as low since the structure of the rewards allows the agent to escape the penalties in just one step. The penalties simply persist because of the probability of an incorrect action (i.e. the agent tried to escape the penalty, but the "wind" pushed him into a penalty state). This idea is reinforced by how the lowest optimal value state is the state that is surrounded by 3 penalized states, and one neutral state. This state is the one that has the highest probability of accidentally transitioning into a -100 reward state.

Question 9: (20 points) 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.

We then plotted the arrow values onto a blank heatmap grid, as seen below in Figure 8.

Figure 8: Policy Function for Reward 2



The optimal policy of the agent seems to match our intuition. The goal of the optimal policy to achieve the highest reward, thus the arrows always direct to the next state with the highest value function reward in heatmap as seen in Figure 7. Therefore, in Figure 8, we observed that the arrows direct us to the hottest part of the map which tend to follow an optimal route. The arrow

avoid the “cold” part of the map by moving away from that area before generally moving down and to the right where the hottest part of the map is. For states that have the penalty, the arrows direct the agent to get out of the penalty region as quickly as possible.

4 Inverse Reinforcement learning (IRL)

Next, we moved onto Inverse Reinforcement learning, or IRL. IRL is the observation of the agent’s reward function by its optimal behavior. This comes from apprenticeship learning, where the goal of the agent is to learn a policy by observing how the expert behaves. This is accomplished by:

1. Learning the policy directly from expert behavior
2. Learning the expert’s reward function and using it to generate the optimal policy

We prefer the second method which gives us the reward function instead of the policy because it provides a simpler description of the behavior and would help design more robust agents.

We used the optimal policy that we calculated before to get the reward function of the expert, and then use that to calculate the optimal policy of the agent and compare the two values. The comparison determines how well the IRL algorithm has performed.

4.1 IRL algorithm

In the environment we’re using for this project, we can use two IRL algorithms to get the reward function:

- Linear Programming (LP) formulation
- Maximum Entropy formulation

We used the LP formula for this part since we had previously covered the linear programming formulation in lectures. The equation can be seen below:

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

In the equation above, R is the reward vector ($R(i) = R(s_i)$), P_a is the transition probability matrix, λ is the adjustable penalty coefficient, and t_i's and u_i's are the extra optimization variables (please note that $u(i) = u_i$). We can simplify the above equation by using block matrices, as seen below.

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

Question 10: (10 points) Express c, x, D in terms of R, P_a, P_{a1}, t_i, u, λ and R_{max}

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

For a in {a2, a3, a4}

$$b = \begin{bmatrix} \mathbf{0}_{(2\cdot(|A|-1)\cdot|S|)\times 1} \\ \mathbf{R_{max}} \\ \mathbf{R_{max}} \end{bmatrix}$$

4.2 Performance measure

To evaluate the performance of the IRL algorithm, we had to first understand the notation that makes up the algorithm:

- $O_A(s)$: Optimal action of the agent at state s
- $O_E(s)$: Optimal action of the expert at state s
- $$m(s) = \begin{cases} 1, O_A(s) = O_E(s) \\ 0, \text{else} \end{cases}$$

We can use m(s) to get the equation for accuracy:

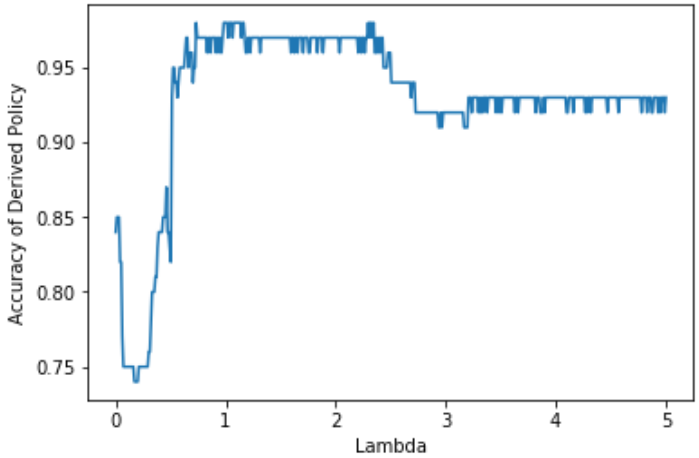
$$Accuracy = \frac{\sum_{s\in\mathcal{S}} m(s)}{|\mathcal{S}|}$$

We used the optimal policy found earlier to calculate the $O_E(s)$ values for Reward Function 1 and Reward Function 2. To calculate $O_A(s)$ we solved the linear program to get the reward function of the expert. We then use that and the value iteration algorithm to get the optimal policy of the agent, which are influenced by the adjustable penalty coefficient λ. λ can be adjusted to maximize the accuracy.

Question 11: (30 points) 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.

By solving the linear programming formulation for λ values between 0 to 5, we report the following relationship of regularization strength and derived policy accuracy, seen below in Figure 9.

Figure 9: Accuracy of IRL algorithm for Reward Function 1

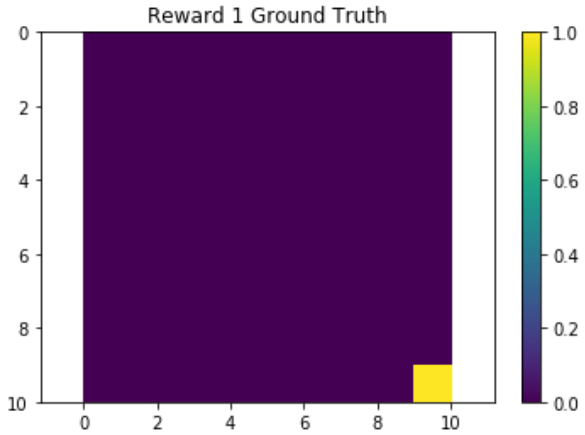
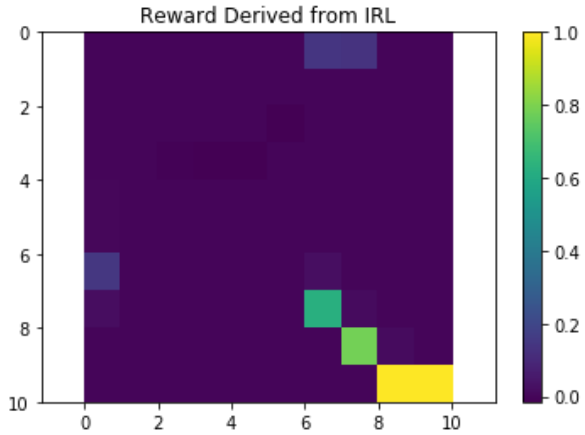


Question 12: (5 points) Use the plot in question 11 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as λ(1) max. Please report λ(1) max

We found that there were multiple lambda values that corresponded to the maximum accuracy (0.98, or just two errors out of 100 actions in the derived policy). We decided to report the largest lambda containing the maximum accuracy, since a higher regularization parameter tends to yield a simpler reward. We report **λ(1) max = 2.345 with an accuracy of 0.98.**

Question 13: (15 points) For λ(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 λ parameter set to λ(1) max. In this question, you should have 2 plots.

Table 1: Ground Truth and Derived Reward Graphs 1

Ground Truth Reward 1	Derived Reward 1
	

Question 14: (10 points) 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.

After running value iteration on the reward derived from inverse reinforcement learning, here is the optimal state values and the accompanying heat map to visualize the optimal state values:

Figure 10: Optimal state values of extracted reward function 1

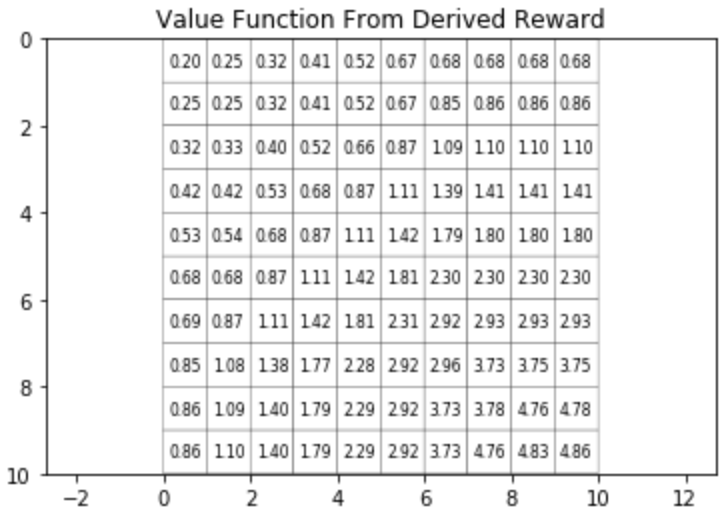
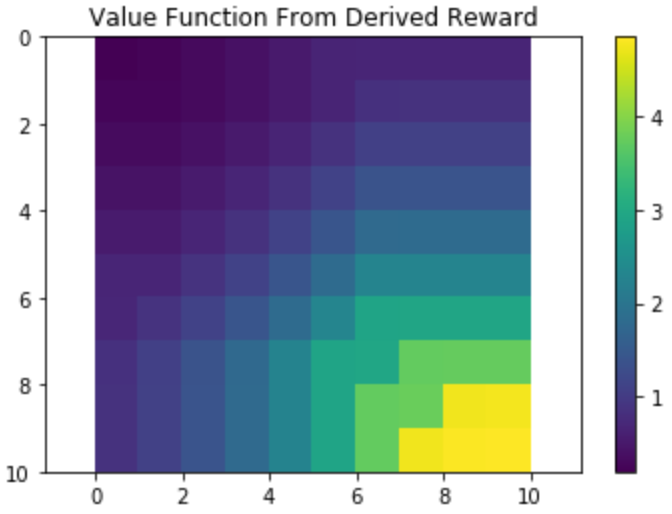


Figure 11: Optimal state value heat map of extracted reward function 1

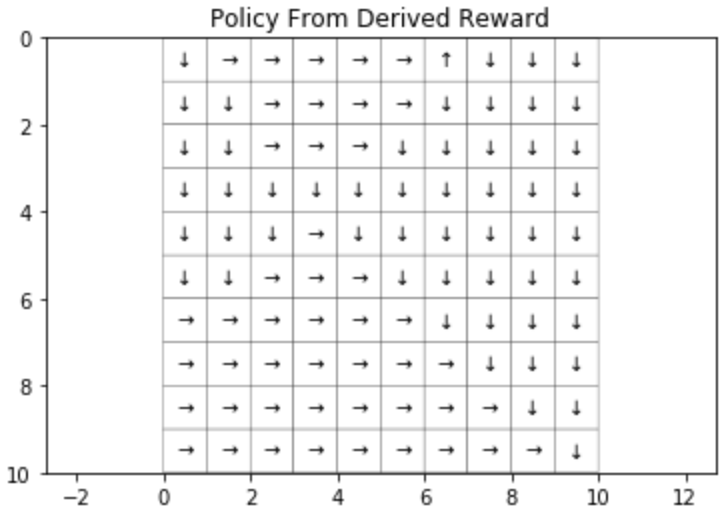


Question 15: (10 points) Compare the heat maps of Question 3 and Question 14 and provide a brief explanation on their similarities and differences.

By comparing the heat maps in Question 3 and question 14 above, we observed that the trends are similar in which both start from top left corner and end in bottom right corner. The values seem more coarsely tuned in the derived optimal state values, since the rewards aren't as sparse as the ground truth reward. This is because the derived reward has rewards in states surrounding the bottom right corner, so the values at those states will also have magnitudes similar to the max magnitude at the bottom right. There is less distinction from state to state as the values are closer together for the derived optimal state values.

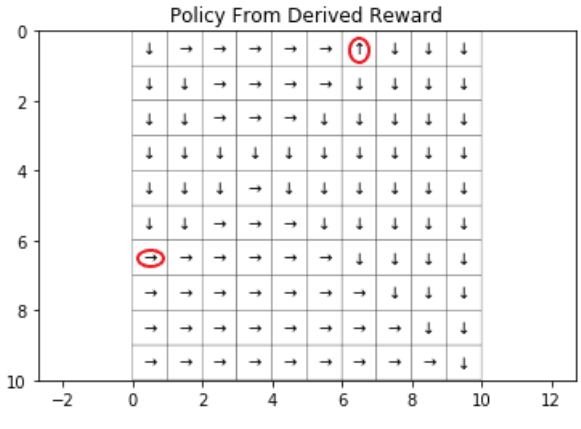
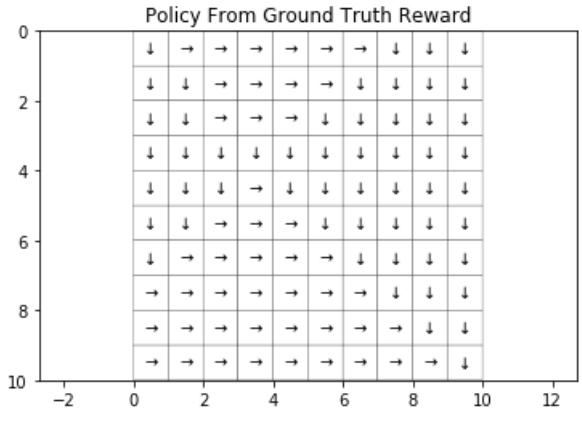
Question 16: (10 points) Use the extracted reward function found in question 13 to compute the optimal policy of the agent. For computing the optimal policy of the agent you need to use the function that you wrote in question 5. For visualization purpose, you should generate a figure similar to that 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.

Figure 12: Derived Policy Graph 1



Question 17: (10 points) Compare the figures of Question 5 and Question 16 and provide a brief explanation on their similarities and differences.

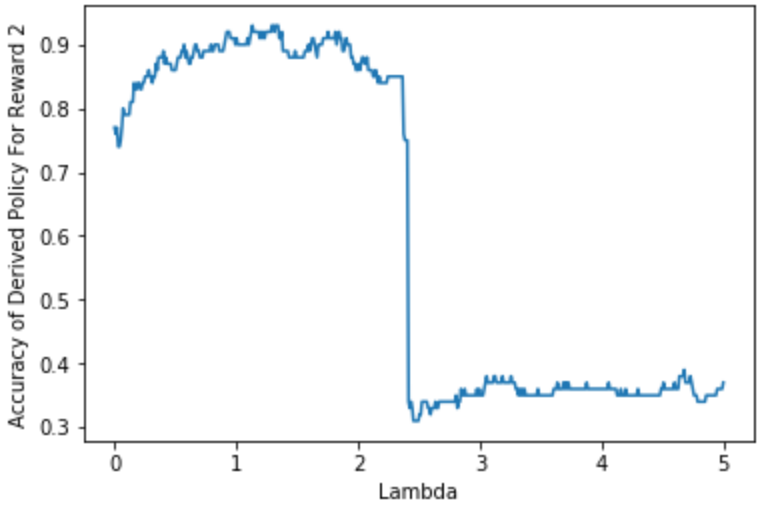
Table 2: Comparison between Figure from Question 16 and Figure from Question 5

Figure from Question 16	Figure from Question 5
	

We observed that both figures in Question 5 and Question 16 have some similarities such that both exhibit highest reward at bottom right corner and lowest reward top left. However, in Table 4 above, the reader can see the differences, circle in red, between figures in Question 5 and Question 16. These differences are caused by fluctuations in the extracted reward function. From the derived reward function shown in Question 13, there are small fluctuations in the rewards at the states where the discrepancies occur. The ground truth rewards are zero at those states, whereas the derived reward has small positive values that end up influencing the derived policy. A possible explanation for the suboptimal behavior is that the derived reward's fluctuations require stricter convergence criteria for value iteration. We will explore changing this in Question 25.

Question 18: (30 points) Sweep λ from 0 to 5 to get 500 evenly spaced values for λ. For each value of λ compute OA(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 OE(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.

Figure 13: Accuracy of IRL algorithm for Reward Function 2



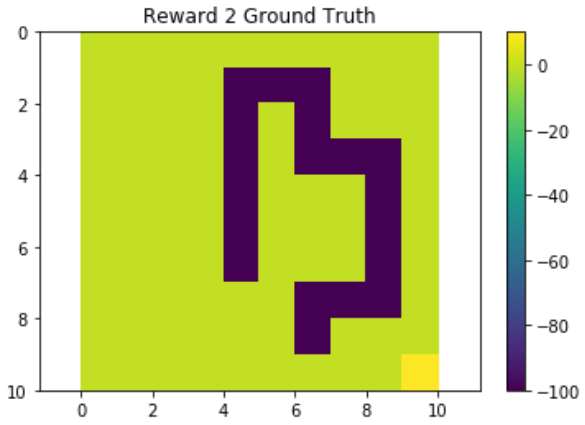
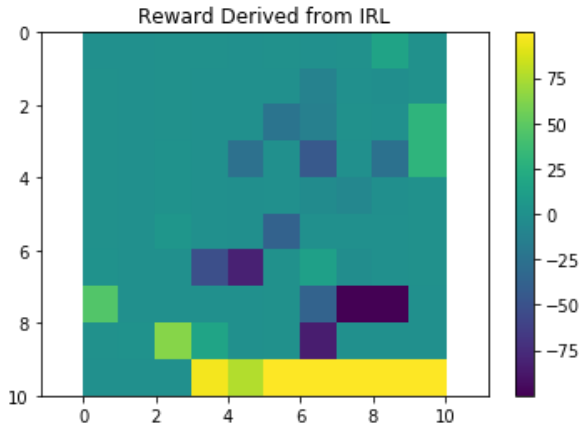
Question 19: (5 points) Use the plot in question 18 to compute the value of λ for which accuracy is maximum. For future reference we will denote this value as λ(2) max. Please report λ(2) max.

We found that there were multiple lambda values that corresponded to the maximum accuracy (0.94, or just six errors out of 100 actions in the derived policy). We decided to report the largest

lambda containing the maximum accuracy, since a higher regularization parameter tends to yield a simpler reward. We report **λ(2) max = 1.333 with an accuracy of 0.94.**

Question 20: (15 points) For λ(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 λ(2) max. In this question, you should have 2 plots.

Table 3: Ground Truth and Derived Reward Graphs 2

Ground Truth Reward 2	Derived Reward 2
	

Question 21: (10 points) 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.

Figure 10: Optimal state values of extracted reward function 2

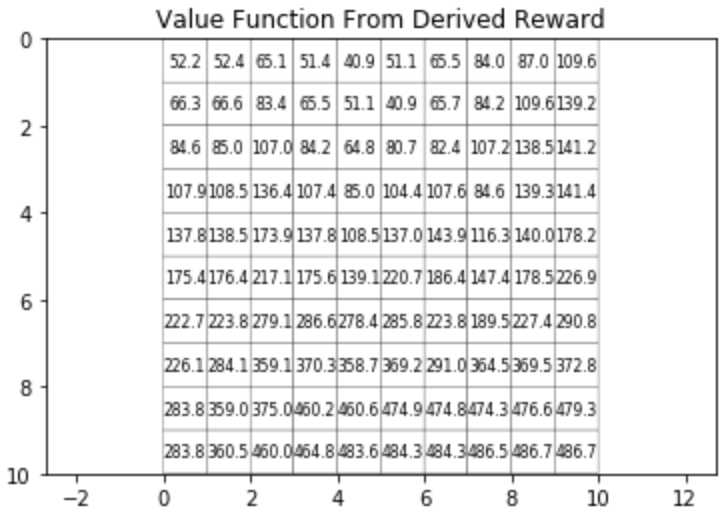
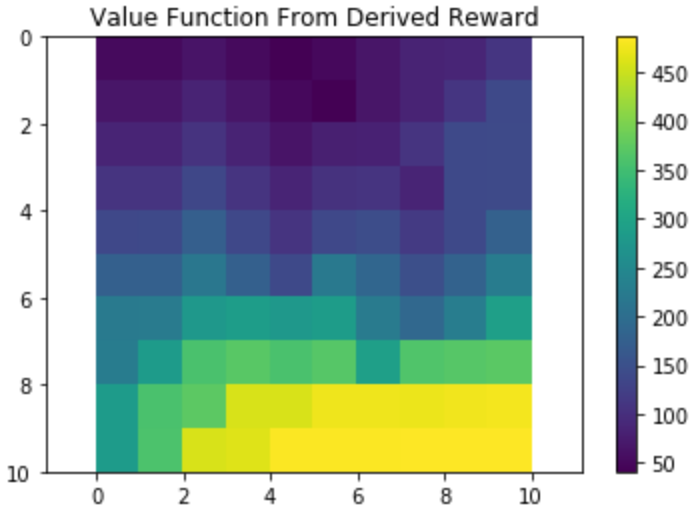


Figure 11: Optimal state value heat map of extracted reward function 2



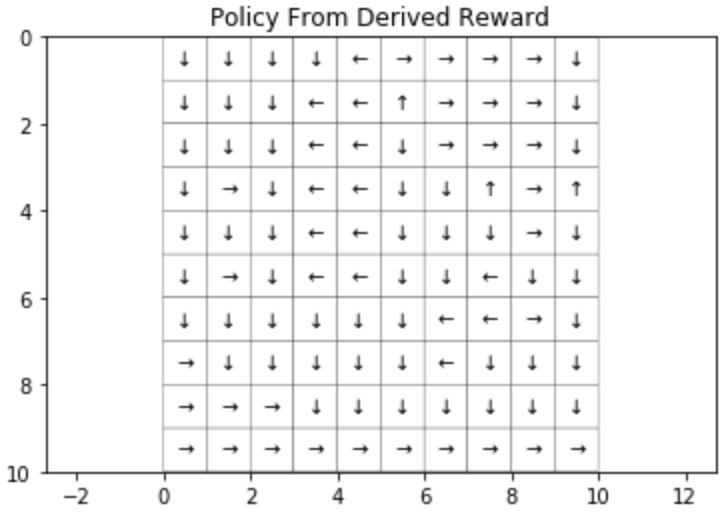
Question 22: (10 points) Compare the heat maps of Question 7 and Question 21 and provide a brief explanation on their similarities and differences.

The derived optimal state values are similar in the regard that they have high values towards the reward at the bottom right, low values towards the top left (because these states are far from the reward), and low values in the penalty region. However, they differ rather significantly when it comes to the magnitude of values. This is due to the fact that Rmax is set to be 100 to accomodate for the fact that the negative penalties are -100 in the ground truth data. Since there isn't much information from the policy to determine that the reward is smaller magnitude than

the penalty, the magnitude of the reward is set to Rmax. Another difference is that the high values from the derived reward are stretched further along the states at the bottom of the 2-D grid. This is attributed to the fact that the derived reward has positive rewards along the bottom row, whereas the corresponding ground truth rewards are zero..

Question 23: (10 points) 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.

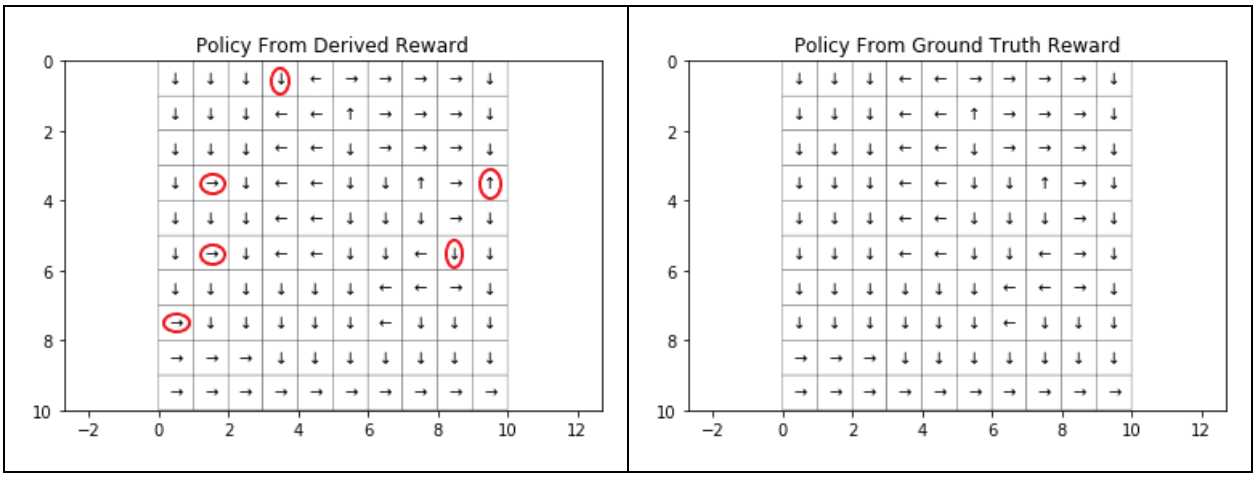
Figure 12: Derived Policy Graph 2



Question 24: (10 points) Compare the figures of Question 9 and Question 23 and provide a brief explanation on their similarities and differences.

Table 4: Comparison between Figure from Question 23 and Figure from Question 9

Figure from Question 23	Figure from Question 9

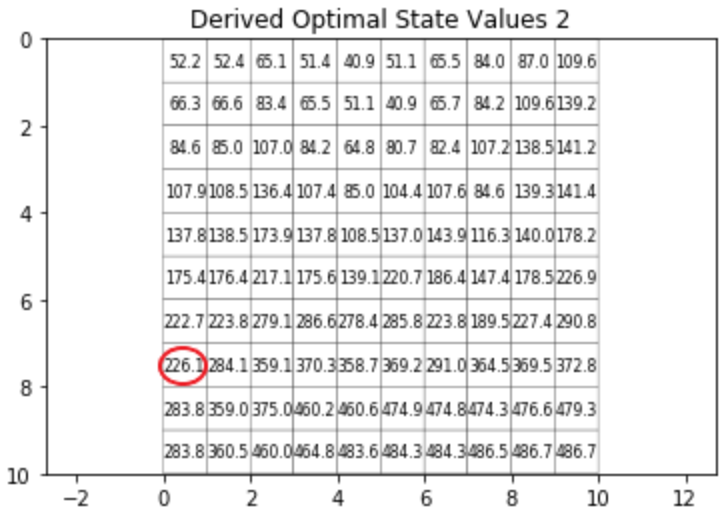


The derived policy has an accuracy of 0.94, so there are 6 states that have differing policies. Three of the discrepancies are in the first 3 columns, where "down" turns into "right." These strategies do not appear too unreasonable, since they still head towards the end reward and are not close to the negative penalty regions. The remaining 3 discrepancies aren't as intuitive, and may be caused by small fluctuations within the derived reward. Since there is more noise in the derived reward, the value iteration algorithm that produces the derived policy may need a more strict convergence condition -- an idea we will explore in Question 25.

Question 25: (60 points) 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. If you can provide a solution to the second discrepancy then we will give you a bonus of 50 points.

To get an idea of how to fix the discrepancies, we focus on one of the discrepancies at the state in column 1, where the optimal policies says to turn right and the derived policy says to turn down. Here is the optimal state policy for the derived reward:

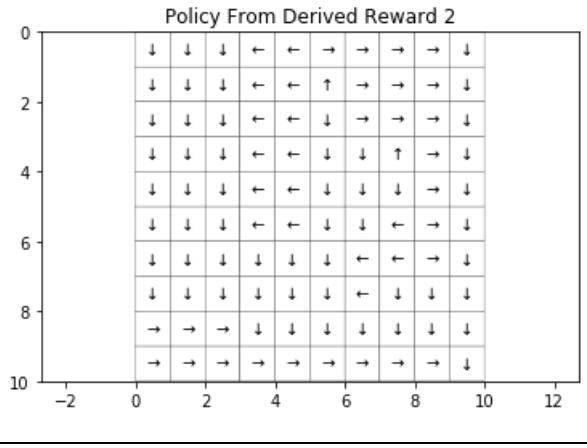
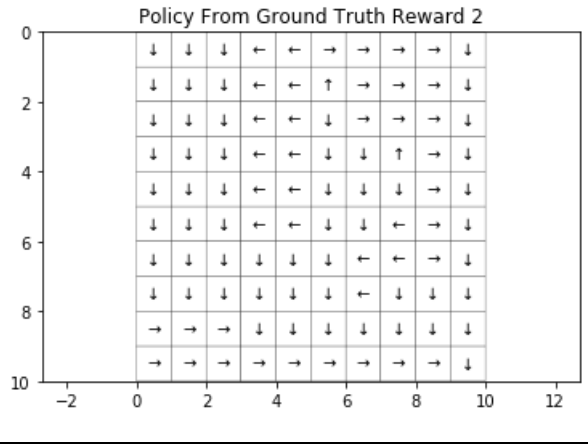
Figure 13: Derived Optimal State Values 2



At the state that we are examining (circled in red in Figure 13 above), the value of the state to the right is 284.1, whereas the state down is 283.3. Since the derived reward is more complicated than the true reward, it's likely that the increased number of fluctuations causes noise in the optimized state values. This may mean that the derived reward needs more precision in order to get a stronger level of convergence. An optimization we can make to the previous value iteration algorithm is to make the convergence criterion stricter, perhaps by decreasing ε by a few orders of magnitude.

After decreasing ε from 0.01 to 0.0000001, we found that accuracy of the derived policy actually matched the ground truth policy entirely (accuracy = 100%). Here are the two policies:

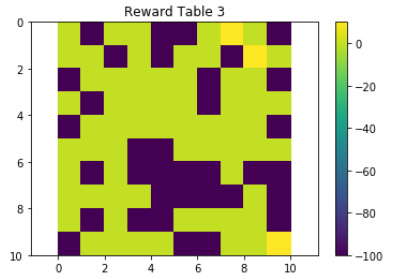
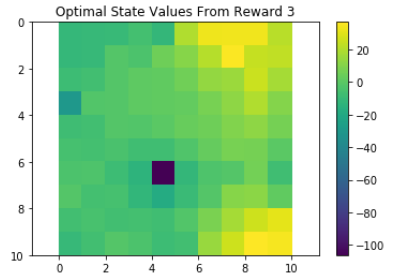
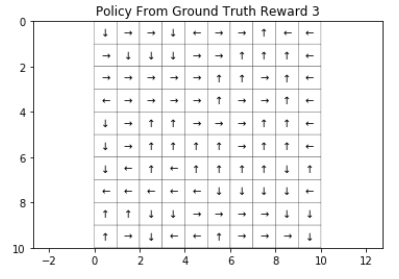
Table 5: Ground truth and derived policy graphs with smaller ε for Reward Function 2

Derived Policy 2 (smaller ε)	Ground Truth Policy 2 (smaller ε)
	

Since the accuracy here is already at 100%, it is difficult to look for other optimizations to improve on the accuracy. Instead, we re-run the same modifications on two random reward functions and see if the accuracy still reaches 100%.

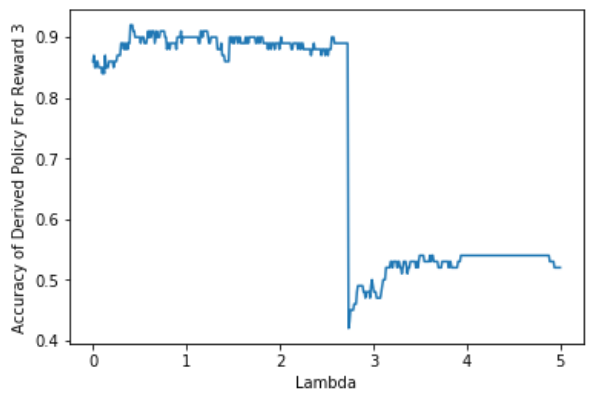
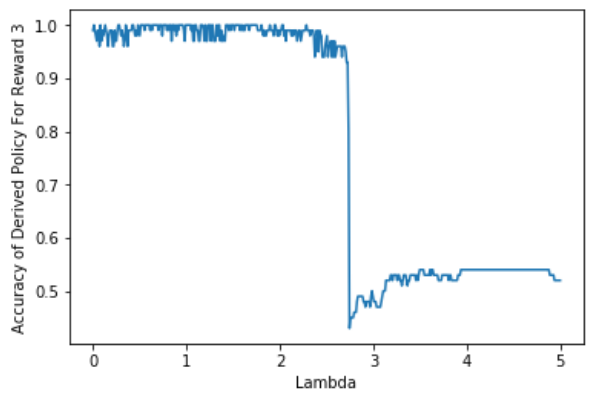
Here is the first random reward function, its ground truth optimal state values, and the optimal policy:

Table 6: Ground truth reward, optimal state values, and optimal policies for a random reward

Reward #3 (Random)	**Optimal State Values**	**Optimal Policy**
		

We show the accuracy vs. lambda plots for lambda between 0 and 5, using the previous epsilon value of 0.01 and the newer epsilon value of 0.0000001:

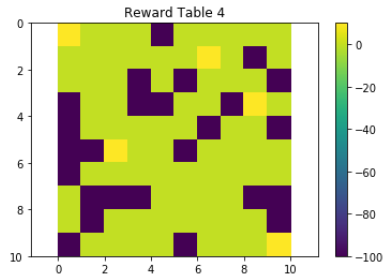
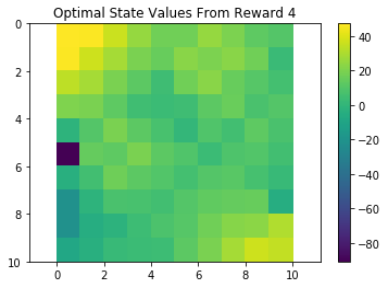
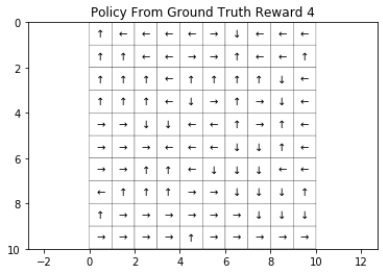
Table 7: Accuracy vs Lambda for Reward 3 With Varying Epsilon Values

Accuracy vs. Lambda for $\varepsilon = 0.01$	**Accuracy vs. Lambda for $\varepsilon = 0.0000001$**
	

The max accuracy for the an epsilon value of 0.01 was **0.92**, whereas the max accuracy with epsilon = 0.0000001 was **1.00**. This reinforces the idea that using a stricter convergence criteria is important for derived policies, as the small fluctuations within the derived reward may yield less room for error when computing the policy.

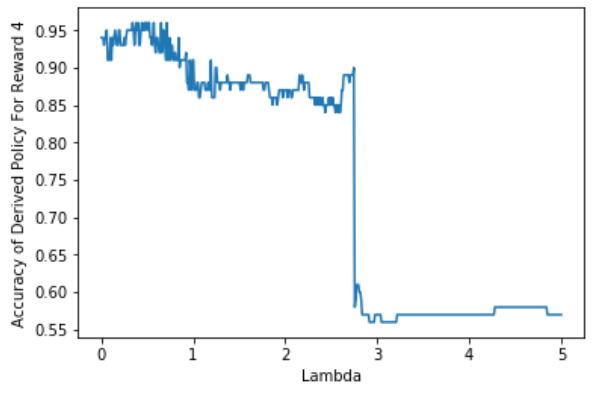
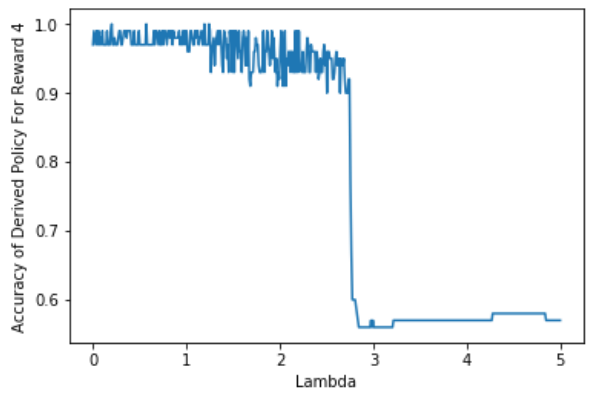
We then run the same analysis on another random reward function, Reward #4. Here are plots of the reward, optimal state values, and policy actions:

Table 8: Ground truth reward, optimal state values, and policy for another random reward

Reward #4 (Random)	**Optimal State Values**	**Optimal Policy**
		

We show the accuracy vs. lambda plots for this reward for lambda between 0 and 5, using the previous epsilon value of 0.01 and the newer epsilon value of 0.0000001:

Table 9: Accuracy vs Lambda for Reward 4 With Varying Epsilon Values

Accuracy vs. Lambda for $\varepsilon = 0.01$	**Accuracy vs. Lambda for $\varepsilon = 0.0000001$**
	

The max accuracy for the an epsilon value of 0.01 was **0.96**, whereas the max accuracy with epsilon = 0.0000001 was **1.00**. This reinforces the idea that using a stricter convergence criteria is important for derived policies, as the small fluctuations within the derived reward may yield less room for error when computing the policy.

We have now shown two random rewards where the derived policy from inverse reinforcement learning matches the original optimal policy when the estimation step (epsilon) is set to be

sufficiently small during the value iteration algorithm. It appears that during reinforcement learning, it is critical to ensure that the convergence criterion is strict enough to sufficiently allow value iteration to converge. This is particularly important for derived policies, since the random fluctuations that arise from the linear programming formulation seem to require more iterations of value iteration before converging back to the original true policy.