# A7 - Value Iteration

CS 4300
Fall 2016
Leland Stenquist - 2,4,6
Ryan Keepers - 1,3,5

## 1. Introduction

The value iteration and policy algorithms seek to create a set of utility states representing the value of being in a given location within an environment and, with that information, a policy for the best plan of action in moving from any given location to a goal state (or not, depending on the reward for making at each step).

Questions we wanted to answer: at what reward value did the policy first suggest moving toward the death as opposed to the goal for the 3x4 grid from the book? At what reward value did the policy first attempt to walk past the death location, instead of circumventing it? Do these values produce similar results on the wumpus board? Finally, do we consider the policies for the wumpus board matching the reward ranges from the textbook on page 648 figure 17.2 to represent reasonably intelligent choices for the given reward?

## 2. Method

The value iteration strategy for finding a policy relies primarily on two functions. These functions are CS4300_MDP_value_iteration and CS4300_MDP_policy. The goal of these functions working together is to find the best policy that optimizes an agent's score when it traverses a board. We run these functions on two board types. One board type is based on the textbook the other is based on the Wumpus world. In each situation the agent chooses a direction. It has an 80% chance of taking that direction and a 10% chance for each choice of going left or right of its chosen direction. Each space is worth some value of points (ex: -0.04). The end points are worth points also (ex: 1 or -1). To exit the board the agent goes to one of these end points.

The first function CS4300_MDP_value_iteration is used to create Utilities for every state on the board. This function uses a for loop to update the utilities until they converge. They are updated based on their Reward as well as their max probability of being chosen given a state times the utility based on the previous loop. Gamma is used as a learning rate or as the book calls it discount. Once the Utilities have converged with accuracy delta the function is complete and it returns the Utilities.

The second function CS4300_MDP_policy is used to create an optimal policy based on the utilities created by CS4300_MDP_value_iteration. This function does this by looking at every state. For each state it sums the probability of getting to each state based on it's current state times the utility. The action with the greatest summation is chosen to be put in the optimal policy.

** A note for running these functions: the function CS4300_preserve_static_utilities includes a hard-coded flag named WUMPUS for swapping between the book's 3x4 board and the wumpus board. Ensure this flag is appropriately toggled for any runs of the program.

# 3. Verification of Program

The program includes test case files for most helper functions (summation, get_state_probs, get_coords) for smaller scale verification of functionality. Functions without test cases were either copied and minorly tweaked (book_board_probs) or utilized no complex logic (preserve_static_utilities).

CS4300_MDP_policy_test is a test utility which verifies the policies produced by CS4300_MDP_policy against the policy outputs shown on the book on page 648, figure 17.2, sections (a) and (b). Given that these tests all match, we assert the program on a whole is working according to expectation.

Utilities from the textbook on page 651, Figure 17.3

| | | | |
|---|---|---|---|
| 0.812 | 0.868 | 0.918 | +1 |
| 0.762 | nop | 0.660 | -1 |
| 0.705 | 0.655 | 0.611 | 0.388 |

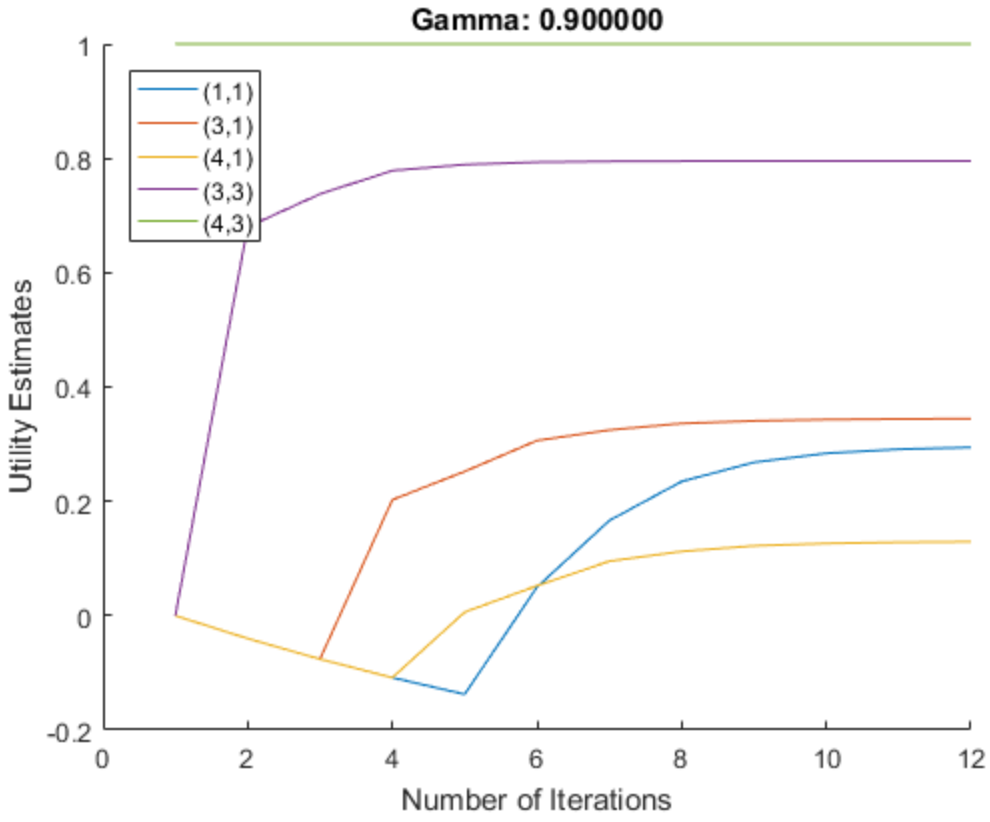
Utilities from CS4300_MDP_value_iteration with R = -0.04, gamma = 0.999999, eta = 0.10

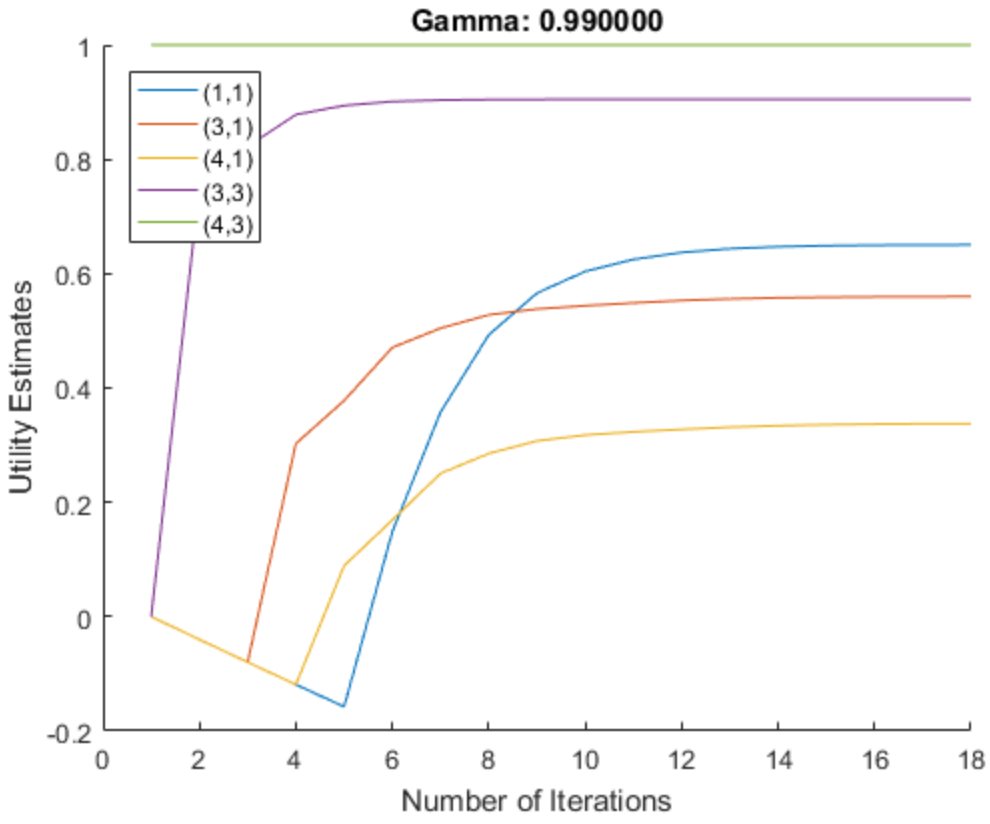
| | | | |
|---|---|---|---|
| 0.8115 | 0.8678 | 0.9178 | +1 |
| 0.7615 | nop | 0.6603 | -1 |
| 0.7052 | 0.6552 | 0.6113 | 0.3878 |

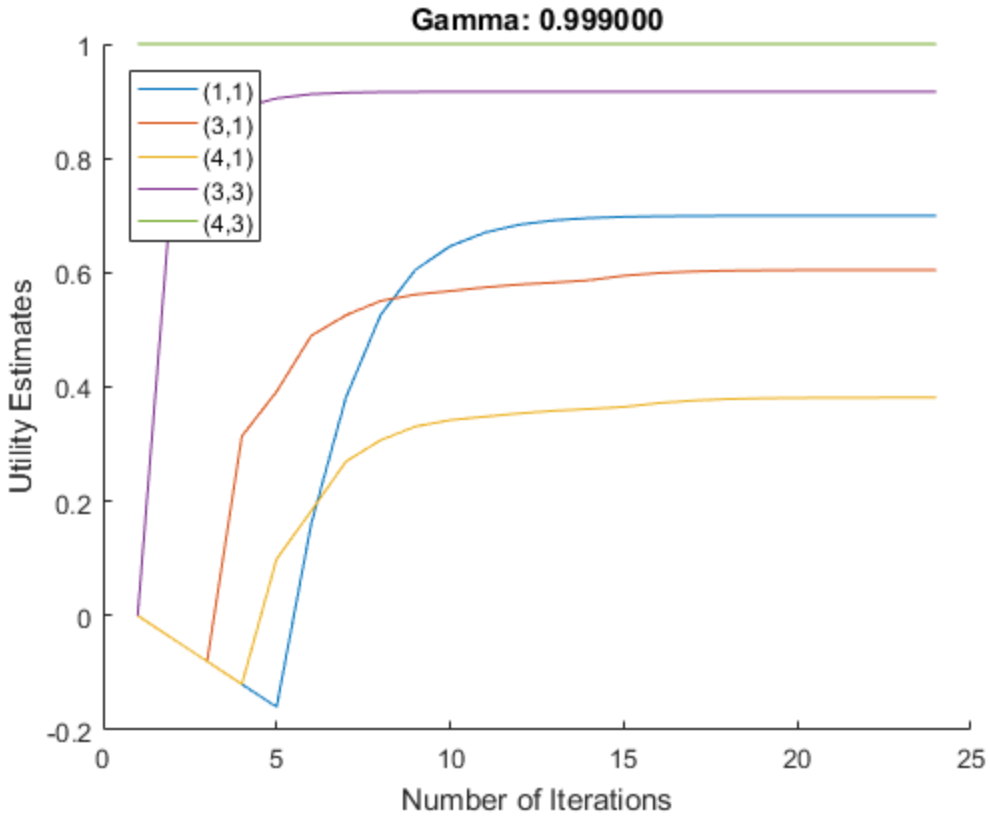
# 4. Data and Analysis

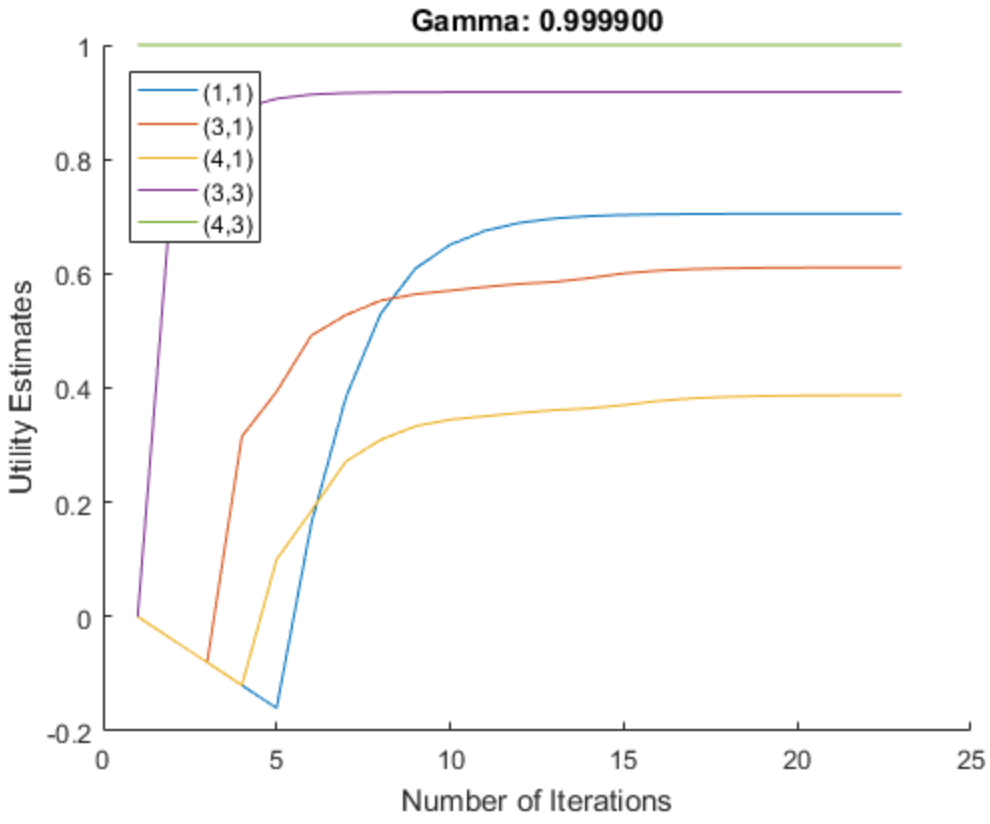
## Gamma's Effect on Utility

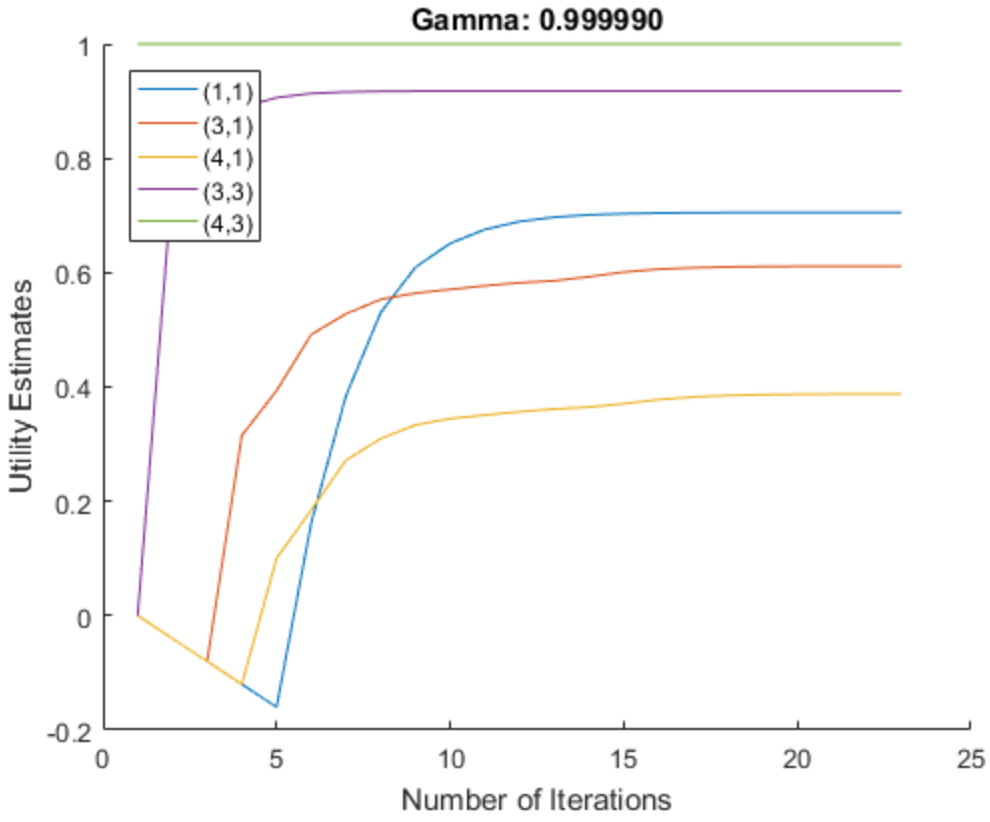
### Book Board

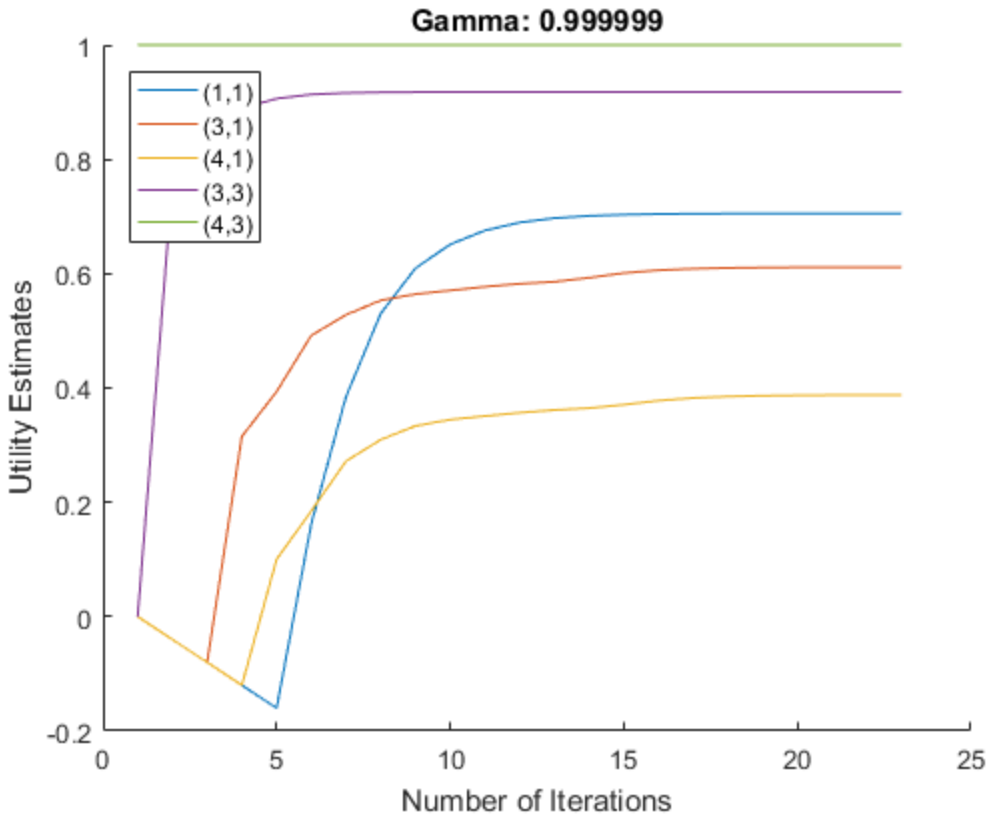



Gamma: 0.990000
(1,1)
(3,1)
(4,1)
(3,3)
(4,3)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
2
4
6
8
10
12
14
16
18


Gamma: 0.999000
(1,1)
(3,1)
(4,1)
(3,3)
(4,3)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25

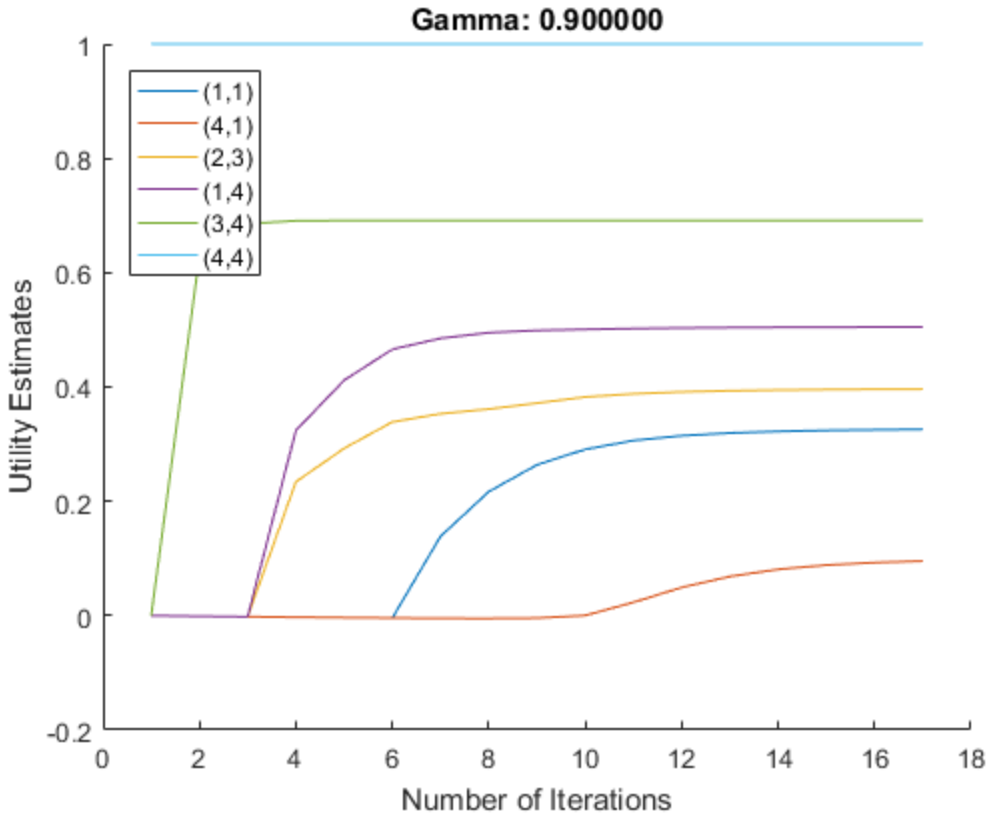

Gamma: 0.999900
(1,1)
(3,1)
(4,1)
(3,3)
(4,3)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25

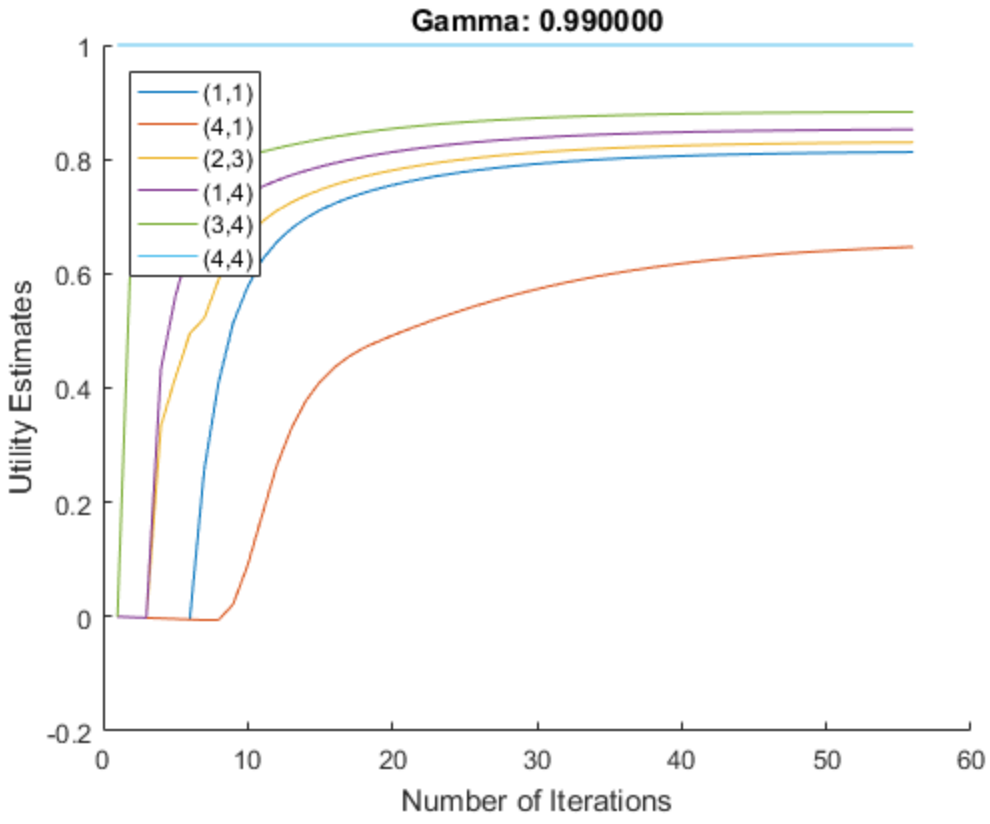

Gamma: 0.999990
(1,1)
(3,1)
(4,1)
(3,3)
(4,3)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25

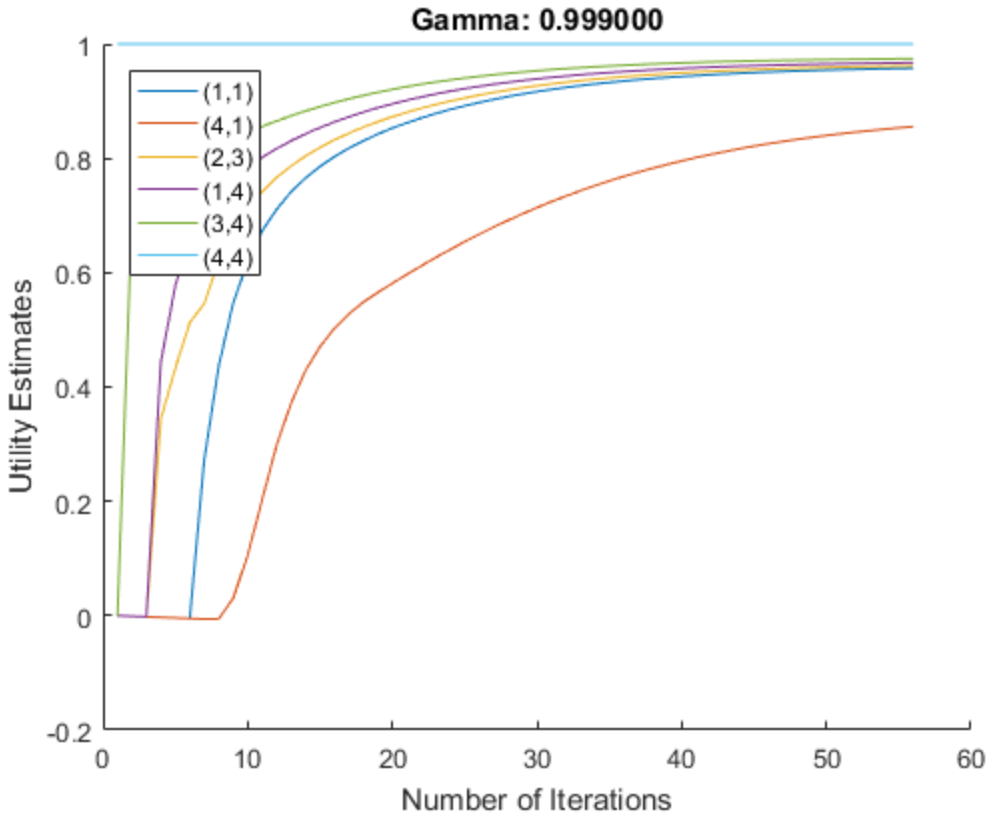

Gamma: 0.999999
(1,1)
(3,1)
(4,1)
(3,3)
(4,3)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25

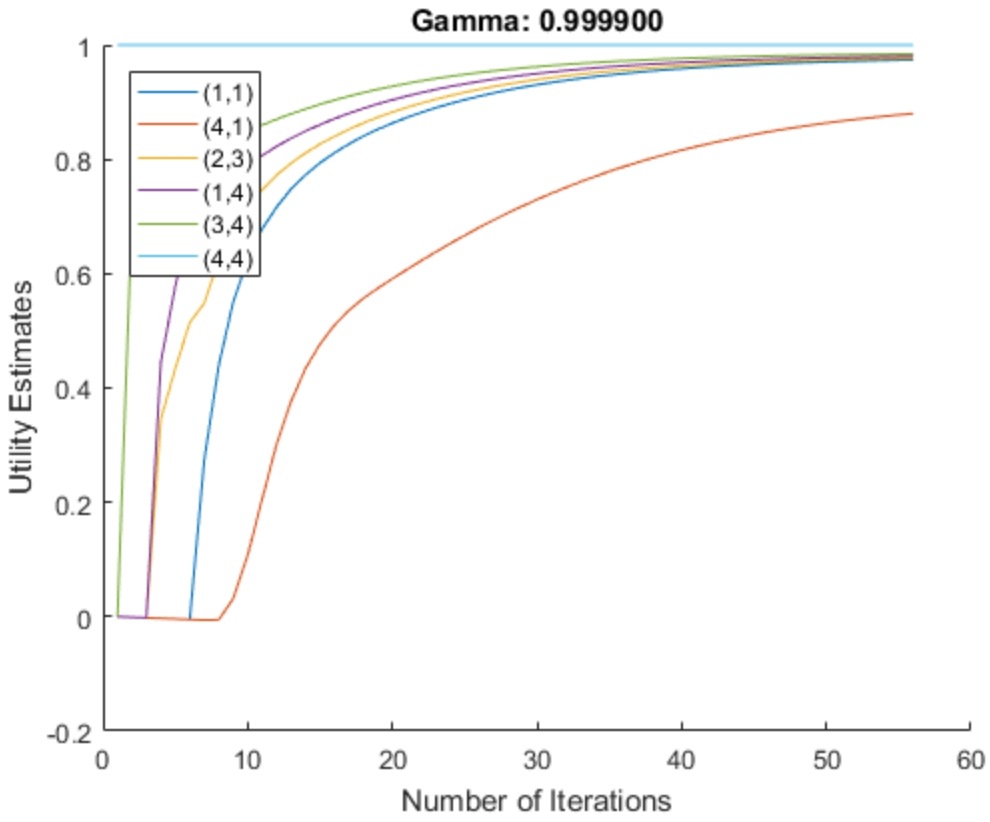
## Wumpus Board

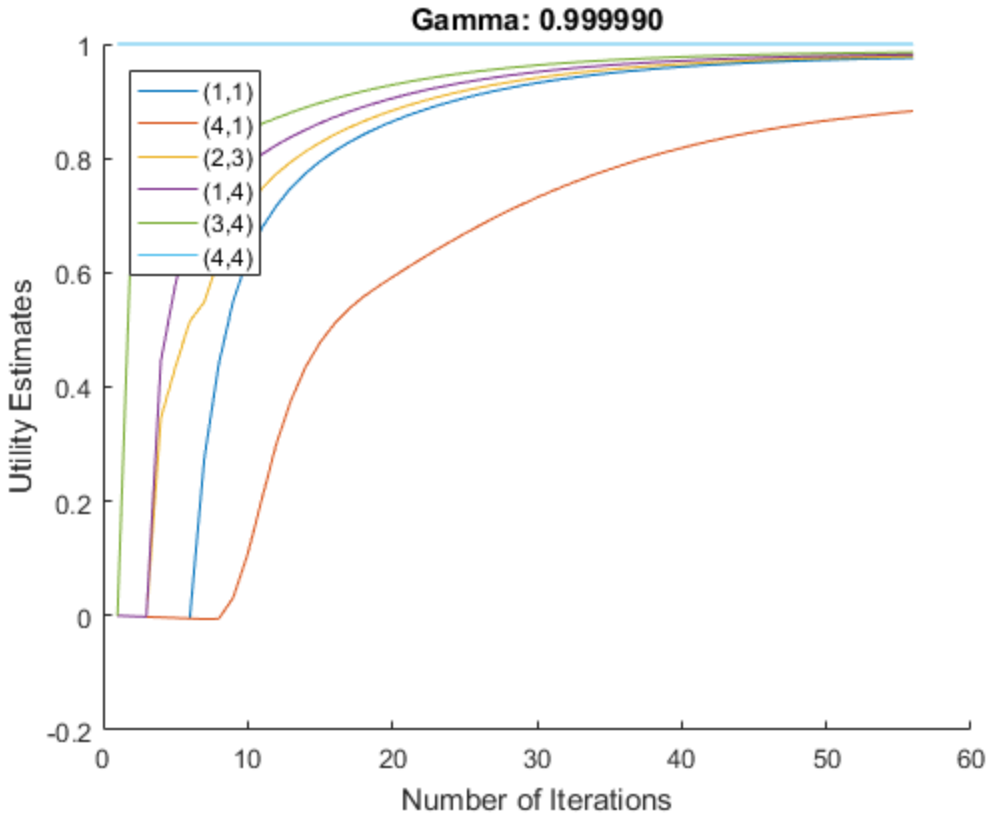
These boards were run where R(pit) = -1, R(Wumpus) = -1, R(Gold) = 1, R(Clear) = 0.001. These are just scalers of 1000 so that our code can easily be switched between the book board and the wumpus board.

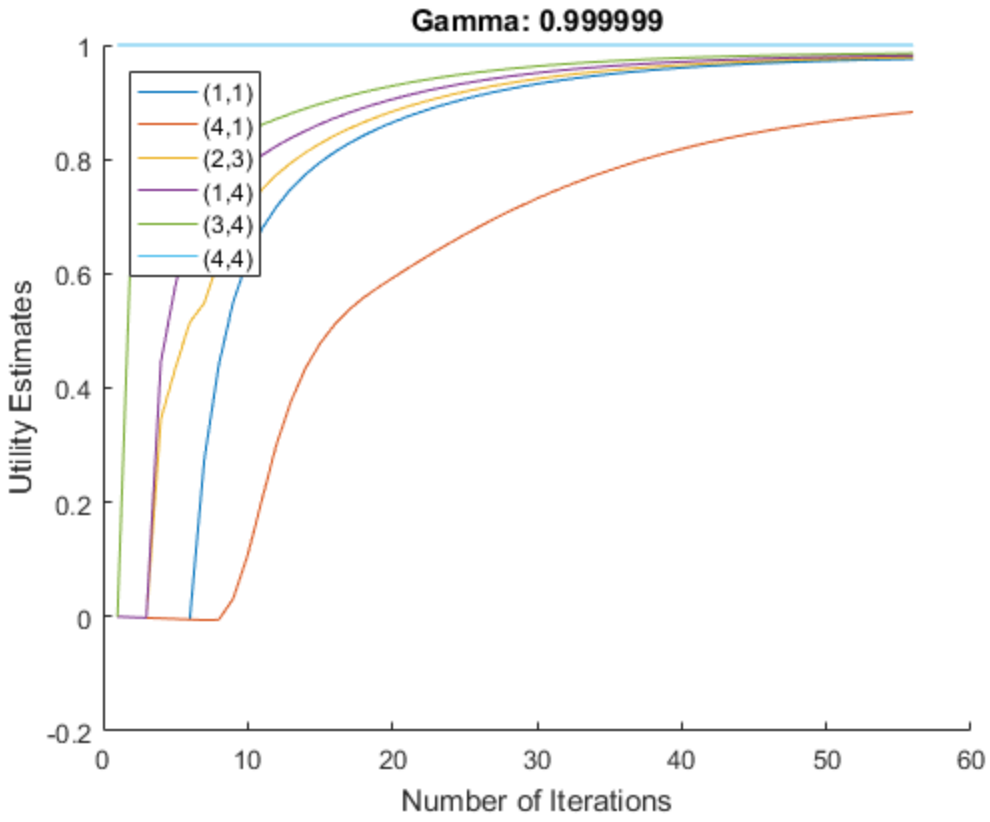



Gamma: 0.990000
(1,1)
(4,1)
(2,3)
(1,4)
(3,4)
(4,4)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60


Gamma: 0.999000
(1,1)
(4,1)
(2,3)
(1,4)
(3,4)
(4,4)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60


Gamma: 0.999900
(1,1)
(4,1)
(2,3)
(1,4)
(3,4)
(4,4)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60


Gamma: 0.999990
(1,1)
(4,1)
(2,3)
(1,4)
(3,4)
(4,4)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60


Gamma: 0.999999
(1,1)
(4,1)
(2,3)
(1,4)
(3,4)
(4,4)
Utility Estimates
Number of Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60

# Reward Level at which Agent Risks Death

## Book Board

| Reward | Explanation of Action Taken |
|---|---|
| -0.05 | takes the route up (dangerous, but not deadly) at 3,1 for the first time |
| -0.46 | takes route up into death at 4,1 for the first time |
| -1.65 | takes the route right into death at 3,2 for the first time |

## Wumpus Board

For values: death = -1, gold = 1

| Reward | Explanation of Action Taken |
|---|---|
| -0.07 | at 2,3 dangerously goes up instead of left |
| -0.17 | at 4,2 goes up into death |
| -0.26 | at 3,1 goes up into death |
| -0.46 | at 2,2 goes right into death |
| -0.68 | at 2,3 goes right into death |

**Note:** At 3,4 never goes down into death instead of right into gold.

# Wumpus Board Policies

These boards were run where R(pit) = -1, R(Wumpus) = -1, R(Gold) = 1. These are just scalers of 1000 so that our code can easily be switched between the book board and the wumpus board.

| R(clear) = -0.001 | | | |
|---|---|---|---|
| 4 | 4 | 1 | x |
| 1 | 2 | x | x |
| 1 | 2 | x | 3 |
| 1 | 2 | 3 | 3 |

| R(clear) = -0.011 | | | |
|---|---|---|---|
| 4 | 4 | 1 | x |
| 1 | 2 | x | x |
| 1 | 2 | x | 3 |
| 1 | 2 | 3 | 2 |

| R(clear) = -0.040 | | | |
|---|---|---|---|
| 4 | 4 | 4 | x |
| 1 | 2 | x | x |
| 1 | 2 | x | 3 |
| 1 | 2 | 2 | 2 |

| R(clear) = -0.200 | | | |
|---|---|---|---|
| 4 | 4 | 4 | x |
| 1 | 1 | x | x |
| 1 | 1 | x | 1 |
| 1 | 1 | 2 | 2 |

| R(clear) = -2.000 | | | |
|---|---|---|---|
| 4 | 4 | 4 | x |
| 4 | 4 | x | x |
| 4 | 4 | x | 1 |
| 4 | 4 | 1 | 1 |

# 5. Interpretation

Assuming that a “normal” value for R is roughly -0.04 (as seems to be the canon value from the book) the 3x4 board from the book tends to avoid outright danger in the ordinary case, and avoids choosing to walk immediately into death until extremely detrimental reward losses. A reward of -0.05 introduces the first definitely dangerous move (walking upwards past the death location), but it isn’t until the reward drops to almost ten times that amount (-0.46) that it makes the first choice to head straight into death. And that only occurs from the bottom right corner, it isn’t until a reward of -1.65 that it chooses to head into the nearest death at every possible opportunity (watching only locations adjacent to the death tile).

The wumpus board, similar to the board from the book, tends to not make risky decisions until a reward of about -0.07, where it decides to go upward when adjacent to the wumpus as opposed to left. However, it decides to opt for death much more quickly than the board from the book. At a reward of -0.17 it makes the first decision to go directly into a death tile. This is at location 4,2, where the agent would be surrounded by two death tiles and a wall. There’s a low likelihood that it will make it out of that situation without either death or a huge amount of standing still while avoiding death as much as possible. So it makes sense that this location is the first to give up and at such a high reward.

However, the rest of the dangerous locations on the wumpus board quickly follow the first. While it took until a reward of -1.65 for every adjacent location on the textbook board to opt for death, the wumpus board fully opts for death in adjacent locations at a reward of only -0.68. Taking the board into consideration on a whole, at -0.50 the policy already guarantees death for any agent not in the upper left quadrant.

So the wumpus board seems to give up much more easily. That said, most policies <= -0.1 seem reasonably constructed. It’s only at -0.1 that the policy becomes significantly risky when adjacent to death tiles. Prior to that, the policy looks reasonably safe and generally attempts to minimize risky maneuvers (that bottom right corner aside).

# 6. Critique

We feel it is had to really know exactly how accurate these algorithms are. To do so we would have to create an agent that ran through the boards n times and then create statistics to see the mean score for different policies. We then could compare these means to the the optimal policy and see how it compares.

Using visual analysis though we can definitely see trends in the optimal policies created that indicate the algorithm is working successfully. This is especially apparent as the reward becomes increasingly negative. If we have a negative reward very close to zero the agent tends to be as careful as possible as it tries to get to the gold. As that reward is made increasingly negative the agent begins to be more risky in trying to get out faster until eventually the reward is so negative that the agent just gives up and goes straight for an exist. These trends suggest that these policy generation strategies are correct.

As for computation time and resources required, it seems that as gamma is increased the algorithm definitely takes longer but not drastically so. This algorithm seems to converge fairly quickly at least on the board given. Still as the board increases the algorithm will run through all the potential states whether it is even possible to visit them or not. This takes a lot of time especially when it has to be done a times for each action. An optimization for this would be to only check states where p(s`|s,a)U(s) is not guaranteed to be zero. In other words only check the states in the direction of the action and to either side. I would say that for larger problems these algorithms should definitely be modified with optimizations.

# 7. Log

Ryan : 8 hours
Sections 1, 3, 5
Leland :  8 hours
Sections 2, 4, 6