Problem 2 (Policy and Value Iteration, 15 pts)

This question asks you to implement policy and value iteration in a simple environment called Gridworld. The "states" in Gridworld are represented by locations in a two-dimensional space. Here we show each state and its reward:

R=−1	R=−1	R=−50	R=−1	R=−1
R=−1	R=−1	R=−50	R=−1	R=−1
R=−1 START	R=−1	R=−50	R=−1	R=−1
R=−1	R=5	R=−50	R=−1	R=50

The set of actions is {N, S, E, W}, which corresponds to moving north (up), south (down), east (right), and west (left) on the grid. Taking an action in Gridworld does not always succeed with probability 1; instead the agent has probability 0.1 of "slipping" into a state on either side. For example, if the agent tries to go up, the agent may end up going to the left with probability 0.1 or to the right with probability 0.1 , but never down. Moving into a wall (i.e., off the edge of the grid) will keep the agent in the same state with probability 0.8, but the agent may end up slipping to a state on either side (defined as before) with probability 0.1. Assume that rewards are received when exiting a state. Let discount factor $\gamma = 0.75$.

The code used to represent the grid is in `gridworld.py`. Your job is to implement the following methods in file `T6_P2.py`. **You do not need to modify or call any function in the `gridworld.py` file to complete this question. Please use the helper functions `get_transition_prob` and `get_reward` in `T6_P2.py` to implement your solution.** Assume that rewards are received when exiting a state. For example, `get_reward(s, a)` when state s is the bottom left corner incurs a reward of -1 for all actions a.

Do not use any outside code. (You may still collaborate with others according to the standard collaboration policy in the syllabus.)

Embed all plots in your writeup.

Problem 2 (cont.)

Important: The state space is represented using flattened indices (ints) rather than unflattened indices (tuples). Therefore value function `V` is a 1-dimensional array of length `state_count`. If you get stuck, you can use function `unflatten_index` to print unflattened indices (so you can easily visualize positions on the board) to help you debug your code. You can see examples of how to use these helper functions in function `use_helper_functions`.

You can change the number of iterations that the policy or value iteration is run for by changing the `max_iter` and `print_every` parameters of the `learn_strategy` function calls at the end of the code.

1a. Implement function `policy_evaluation`. Your solution should iteratively learn value function V using convergence tolerance `theta = 0.01`. (i.e., if $V^{(t)}$ represents V on the tth iteration of your policy evaluation procedure, then if $|V^{(t+1)}[s] - V^{(t)}[s]| \leq \theta$ for all s, then terminate and return $V^{(t+1)}$.)

1b. Implement function `update_policy_iteration`. Perform 10 iterations of policy iteration, and for every 2nd iteration include a plot of the learned value function and the associated policy (`max_iter = 10`).

These plots of the learned value function and implied policy are automatically created and saved to your homework directory when you run `T6_P2.py`. Do not modify the plotting code. Include all of your plots in your homework submission writeup. For each part of this problem, please fit all the the plots for that part onto 1 page of your writeup.

1c. How many iterations does it take for the policy to converge? (Hint: change `print_every = 1` to see the policy and value plots for every iteration!) Include a plot in your writeup of the learned value function and policy.

2a. Implement function `update_value_iteration`. Perform 10 iterations of value iteration and for every 2nd iteration include a plot of the learned value function and the associated policy (`max_iter = 10`). Include all plots in your writeup.

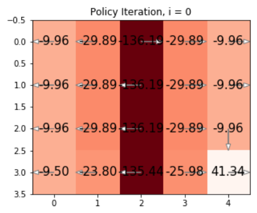
2b. Set the convergence tolerance for value iteration to 0.1 by setting parameter `ct = 0.1` in the `learn_strategy` function call. How many iterations does it take for the values to converge? Include a plot in your writeup of the final converged learned value function and policy.

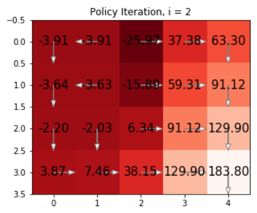
3. Compare the convergence and runtime of both policy iteration and value iteration. What did you find?

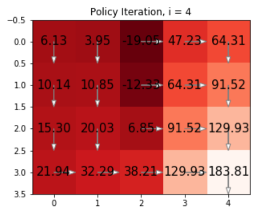
Solution

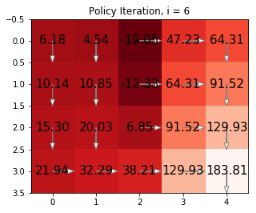
1a.

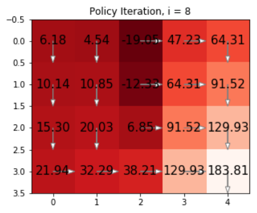
1b. The plots of the 0th, 2nd, 4th, 6th, and 8th iteration for `policy_evaluation` are shown below.



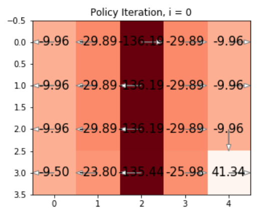


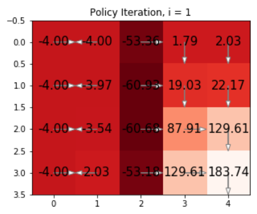


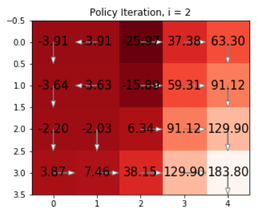


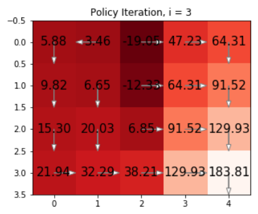


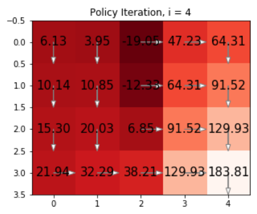
1c. Running for 10 iterations and setting `print_every` = 1 gives the following 10 plots

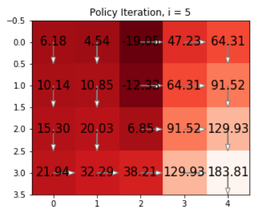


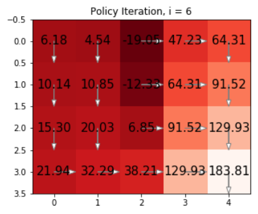


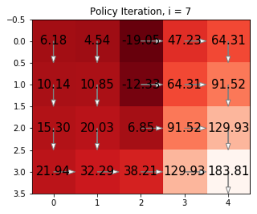


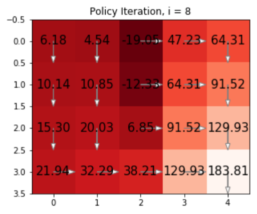


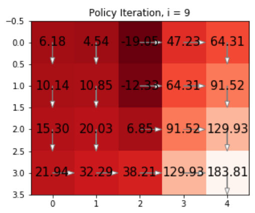






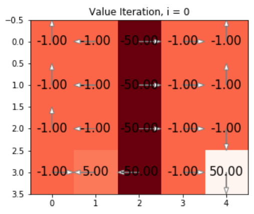


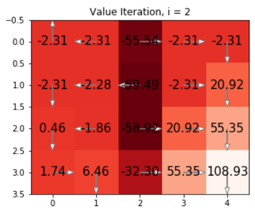


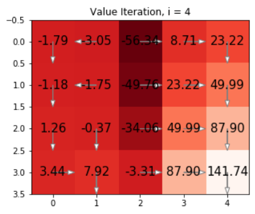


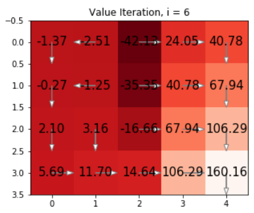
The policy converges on the fifth iteration ($i = 4$), since the policy do not change for any of the subsequent iterations ($i = 5$ through $i = 9$), and the fifth iteration has a different policy (observe row 1 column 2) than the fourth iteration ($i = 3$). The value function converges on the sixth iteration ($i = 5$) for the same reason.

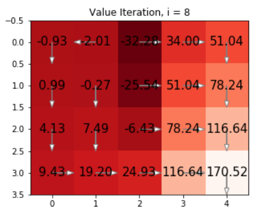
2a.





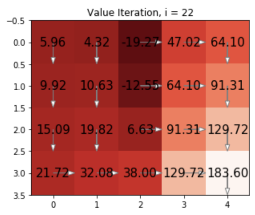






2b. It takes 23 iterations ($i = 22$) for the values to converge in value iteration.

```
Converged at iteration 22
```



```
Value iteration complete.
```

3. From my results, policy iteration converged to the optimal value function and policy in fewer iterations than value iteration. In part 1(c), the policy iteration was able to converge to the optimal policy in just $i = 4$ iterations and the optimal value function in just $i = 5$ iterations. Whereas the value iteration converged to the optimal value function in $i = 22$ iterations. Moreover, the value iteration was run with a looser convergence threshold of 0.1, whereas the policy iteration was run with a stricter convergence threshold of 0.01, which adds further evidence that the policy iteration converges much faster than the value iteration, since had the value iteration been run with the stricter convergence threshold of 0.01 it would have taken even more than $i = 22$ iterations to converge.

   While the runtimes of the policy and value iterations were not measured explicitly, I noticed that for each iteration the runtime of the policy iteration was slower than the value iteration, which makes intuitive sense since policy iteration utilizes policy evaluation, which is an extra costly step that value iteration doesn't use. This allows policy iteration to converge in fewer iterations than value iterations, but it causes each iteration to become more computationally expensive in policy iteration. This is consistent with the complexity analysis of policy and value iteration that was done in lecture 21, so we can make a theoretical comparison of the difference in runtimes. Let $|S|$ be the number of states (20 for this problem), $|A|$ be the number of actions (4 for this problem), and L be the maximum number of reachable states. For value iteration, the work require per iteration is $O(|S||A|L)$, whereas the work required per iteration for policy iteration is $O(|S||A|L + |S|^3)$. The addition of the $|S|^3$ term demonstrates the extra work of policy evaluation needed for every iteration of the policy iteration algorithm.