# Question 3

*Note: documentations and unimportant comments are omitted in this word document as including them takes up too much space, but they are included in the notebooks provided*

## Part a)

Text

Description automatically generated with medium confidence

Policy iteration consists of two steps, an evaluation step, and an improvement step. In the evaluation step, we iteratively compute the state values for every state until the values converge. In the improvement step, we update the policy for every state such that the state values improves. We iteratively repeat these two steps until the policies stops updating and converges.

Policy iteration works because is an sufficient condition to state that Thus, as long as a new policy increase in the policy improvement step, it is guaranteed that this new policy is improved from the old policy.

In the deterministic setting, all probabilities are assumed to be unity, and both reward and the next state are deterministically a function of the current state and action taken.

Our pseudo code can be simplified by removing the probability terms:

Text

Description automatically generated with medium confidence

We can verify if the optimum conditions have been met by testing if the state value derived from the new policy does indeed converge to the same state value derived from the policy evaluation loop. If so, this implies that an optimum state is reached since state values have unique optimum states.

## Part b) Policy Iteration algorithm

**Implementation**



The policy evaluation corresponds to the first and second block in the pseudo code, and the implementation follows exactly line-by-line from the pseudo code. Note that to iterate through every state, we have to iterate through the entire grid which is row by columns by headings large. Where the first channel is the row, the second channel is the column, and the last channel is the heading (directions). Hence, we use a nested for-loop to get access to all the states. The policy evaluation function will return a value grid that approximates the state value of every state given the current policy.

The implementation of the helper function get\_state\_value is as follows:



Where the environment object’s internal functions are used to calculate the next state reward and whether or not a terminal state has reached. Note that there’s a small bug in the file simple\_grid\_env.py, this is described in detail in question3.ipynb. The presented implementation assumes that the mistake is fixed.

The policy improvement step corresponds to the third block in the pseudo code, and is implemented as follows: 

Similarly, a nested loop is used to access each state, and the code follows line-by-line from the pseudo code. This function returns an improved policy and a Boolean that describes whether or not the policy grid has converted to a stable state.

Lastly, a wrapper iteration function to include both the policy evaluation function and the policy improvement function is as follows:



Chart, treemap chart

Description automatically generated

Figure 1 Final policy evaluation for simple grid

Treemap chart

Description automatically generated

Figure 2 Final policy for simple grid

**Part c)** The final policy evaluation and policy are as follows:

Chart, treemap chart

Description automatically generated

Figure 3 Final policy evaluation on full grid

Qr code

Description automatically generated

Figure 4 Final policy on full grid

The simplest way to verify that the policy is indeed correct is to simply implement the policy and see if 1) the path that the robot takes does make sense, in this case it should reach one of the win states from all states 2) if the state values derived by following the policy are equal to the state values converged by policy evaluation.

One way to do this is examining the grid by eye and tracking where the agent moves to for every state, in the figure below we illustrate three paths undertaken by the agent starting from the red circle labelled 1, the blue circle labelled 1, and the green circle labelled 1. We can also see that the state values for each of the starting states and the states that the agent walked past are also correct (E.g. for the first red state, the state value would be 10 + 4(-1) = 6). Note that we can repeat the same step for all other cells and show that they all reach a win state with the shortest path possible.

Treemap chart

Description automatically generated

We have also written a custom function that validates that the state values from the converged policy does equal to the converged state values from the policy evaluation step.



The code helped us validate that the converged policy is indeed optimal.

## Part d)

Text

Description automatically generated

Similar to the policy iteration, we remove the probability terms in the deterministic setting as follows.

Text, letter

Description automatically generated

Unlike policy iteration, we do not calculate the state value and improve the policy in two separate steps, but rather both are ‘combined’ into one loop. We simply find the action which maximizes the action state value and assign the value to the state value. We repeat until convergence, and we will obtain the optimal state value for all states. Note that this is valid because optimum state value has a unique solution.

In the next step, we find the action which maximizes the state value for all states, and this effectively returns the optimum policy (which is not necessary unique).



A picture containing text, cabinet

Description automatically generated

Figure 5 State values from value iteration of the simple grid

Background pattern

Description automatically generated

Figure 6 Policy from value iteration of the simple grid

We observe that the state value grid and the policy grid derived using the value iteration is identical compared to when policy iteration is used. Note that the same random seeds are used for fair comparison.

However, the two algorithms differ in computational time (taken from average of three runs with random initial seeds):

Policy iteration: 6.78426

Value iteration: 3.38841

We observe that policy iteration takes almost twice as long as value iteration whilst the results converged by both algorithms are identical. This indicates that value iteration is a better algorithm for small grid sizes as it is significantly faster.

**Part e)** The final policy and state values on the large grid are shown:

Chart, treemap chart

Description automatically generated

Figure 7 Final state value from value iteration

Qr code

Description automatically generated

Figure 8 Final policy from value iteration

Similar to part c), we can verify the results by initializing the robot at a every state and see if 1) reaches a terminal state with +10 and 2) if all the state value function are identical. However, since both the state value function and policy are identical to the grids derived in part c), we will not repeat the same step. We can conclude, as we have in part c), that the converged policy is indeed the same.

## Part f)

We note that in both our policy iteration implementation and value iteration implementation’s pseudo code, the value update contains two probabilistic terms:

Which is a joint probability distribution of the reward and state transition. In our deterministic implementation, this term is assumed to be 1. This is because in the deterministic setting, both the reward and the next state are a function of the previous state and previous action (rather than a probability distribution of the s and a), such that:

In the probabilistic setting where the state transition is probabilistic, can no longer be assumed to be unity. However, since the reward is still deterministic, we can simplify the term to be:

To implement this, we simply need a model of that captures , and we can substitute this term into the both the policy evaluation and policy improvement steps.

Text

Description automatically generated with medium confidence