ECEN 743: Reinforcement Learning
Assignment 3

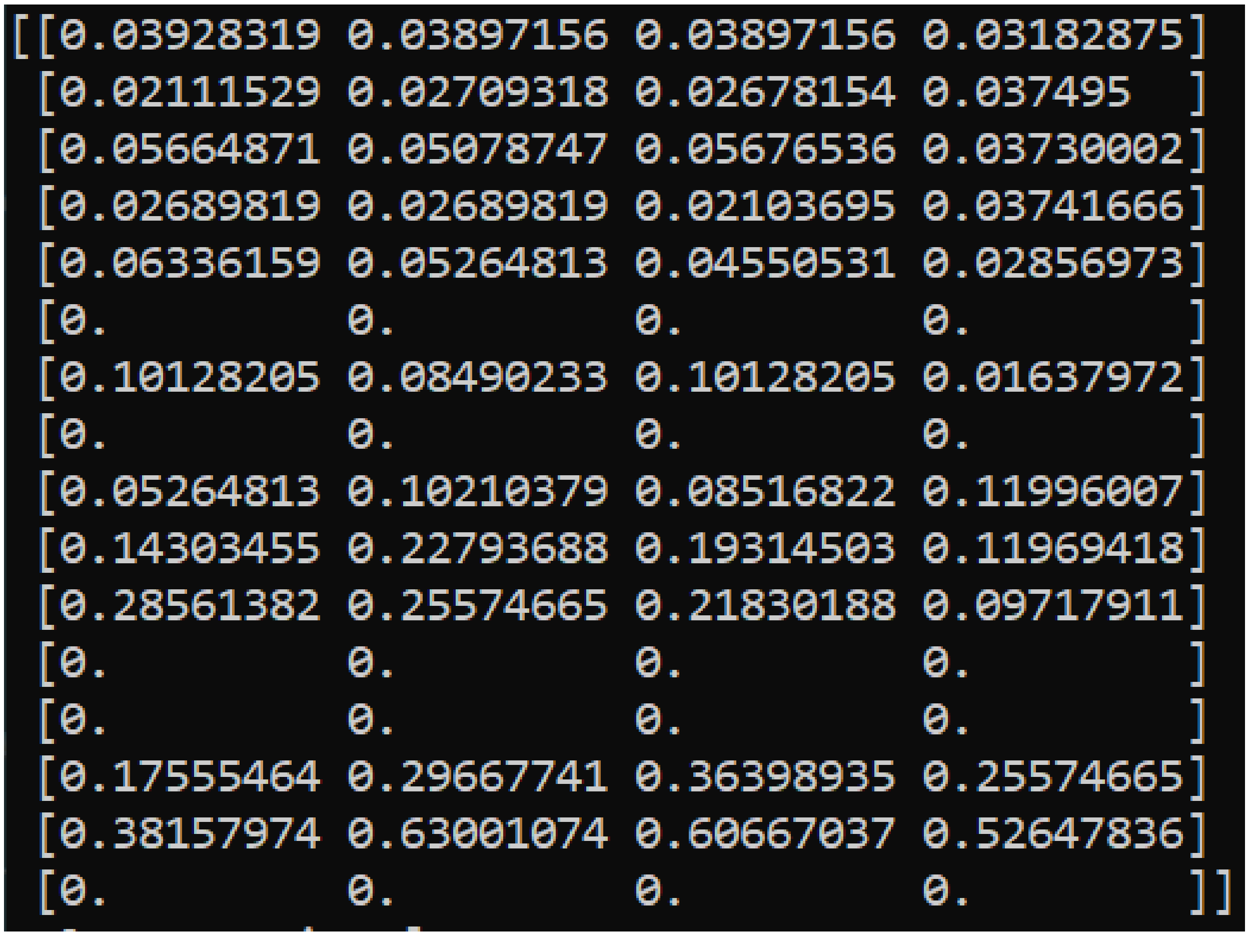
Planning:

1. **Q-Value Iteration (QVI)**
   a. What is the optimal policy and value function?

From Q-Value Iteration, we get this resulting optimal policy and value function from 14 iterations (it was close to convergence at this point, a fact we will go into later):
Resulting Value Function:

0.03928319	0.037495	0.05676536	0.03741666
0.06336159	0.	0.10128205	0.
0.11996007	0.22793688	0.28561382	0.
0.	0.36398935	0.63001074	0.

Resulting Q-Value Function (screenshot from code):



```
[[0.03928319 0.03897156 0.03897156 0.03182875]
 [0.02111529 0.02709318 0.02678154 0.037495  ]
 [0.05664871 0.05078747 0.05676536 0.03730002]
 [0.02689819 0.02689819 0.02103695 0.03741666]
 [0.06336159 0.05264813 0.04550531 0.02856973]
 [0.         0.         0.         0.        ]
 [0.10128205 0.08490233 0.10128205 0.01637972]
 [0.         0.         0.         0.        ]
 [0.05264813 0.10210379 0.08516822 0.11996007]
 [0.14303455 0.22793688 0.19314503 0.11969418]
 [0.28561382 0.25574665 0.21830188 0.09717911]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.17555464 0.29667741 0.36398935 0.25574665]
 [0.38157974 0.63001074 0.60667037 0.52647836]
 [0.         0.         0.         0.        ]]
```

Resulting Policy:

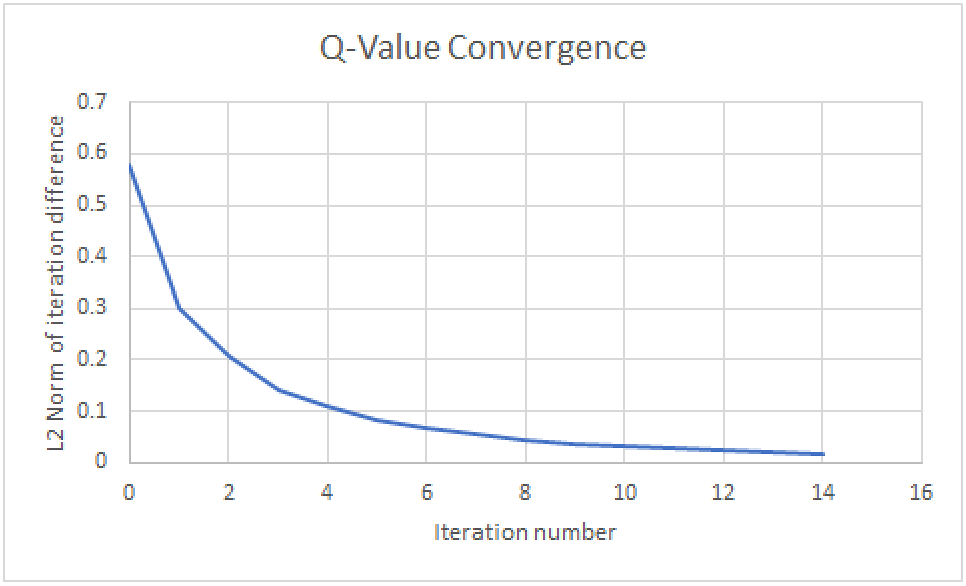
0	3	2	3	0	0	0	0	3	1	0	0	0	2	1	0

While seemingly strange actions are taken here, it is important to note that the environment is slippery, and thus the outcome of an action is stochastic. This leads to choosing actions that are more likely to lead to the solution, rather than the most direct path.

The convergence criteria for this implementation is a boundary that the mean squared error (MSE) must be lesser than a tolerance value. In this instance, the MSE needed to be less than `.000005 (default).`

b. Plot $U_k = ||Q_k - Q_{k-1}||$, where Q_k is the Q-value during the k^{th} iteration.

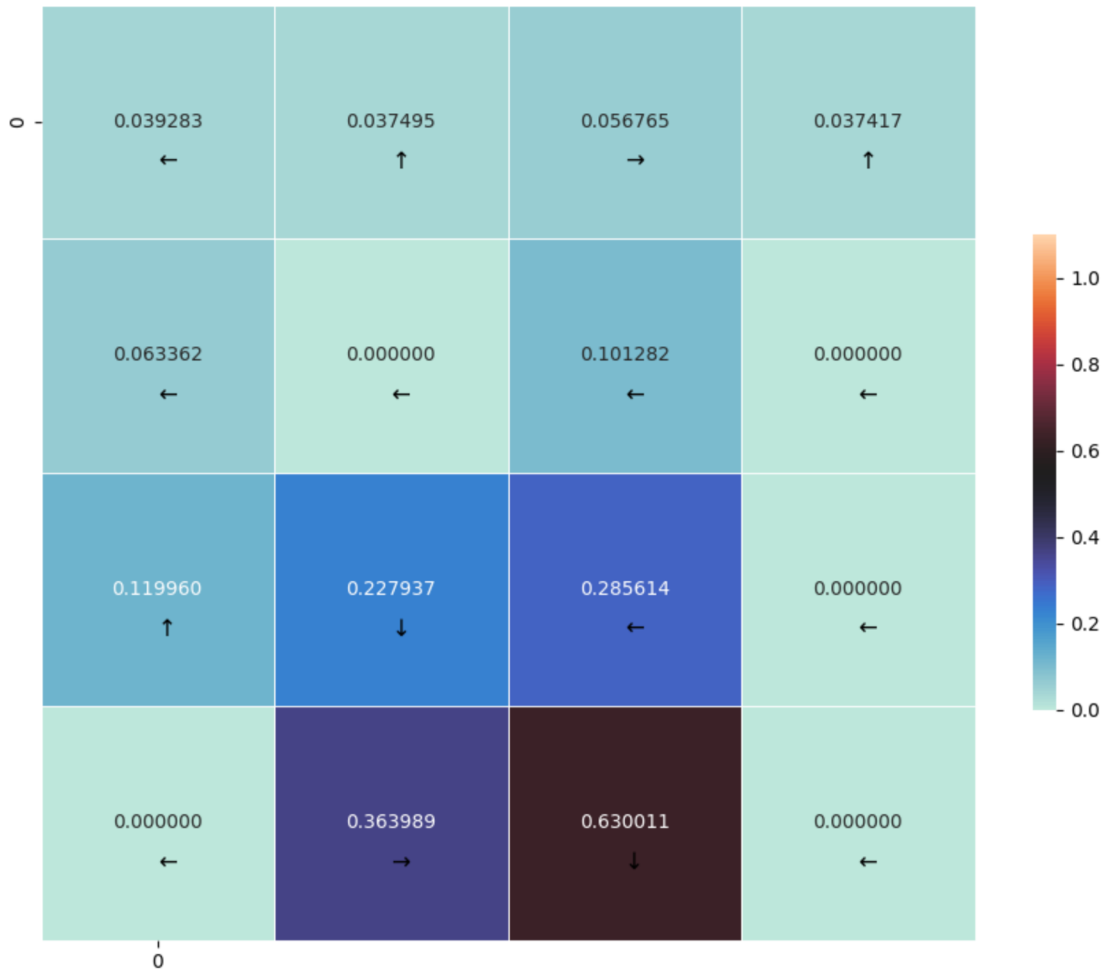
For this question, I interpreted this norm to be the L2 norm, but regardless it should show some semblance of convergence.



c. Plot the heat maps of the optimal policy and value function.

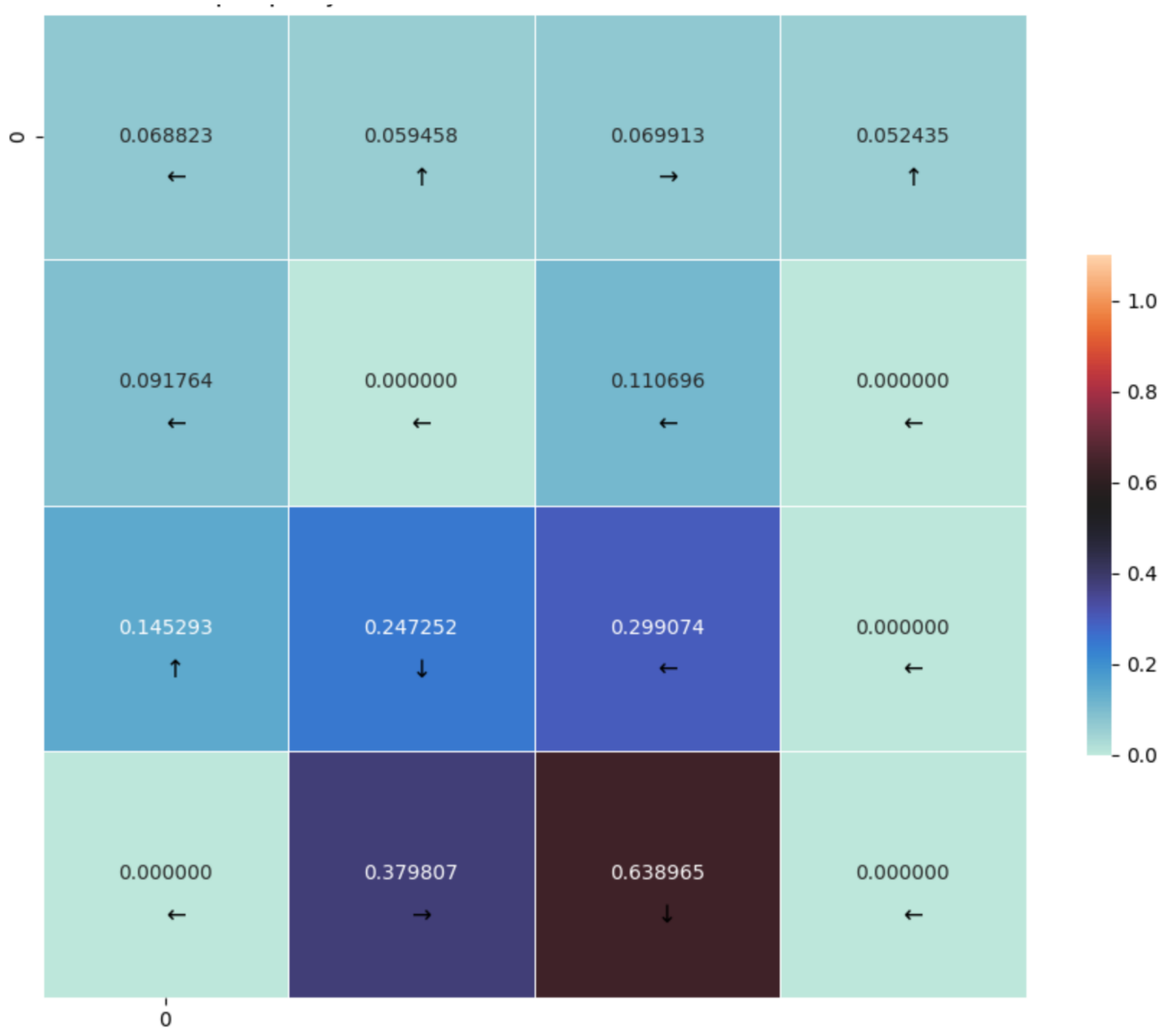
This was done using the provided function.

Heat Maps of optimal policy and value function

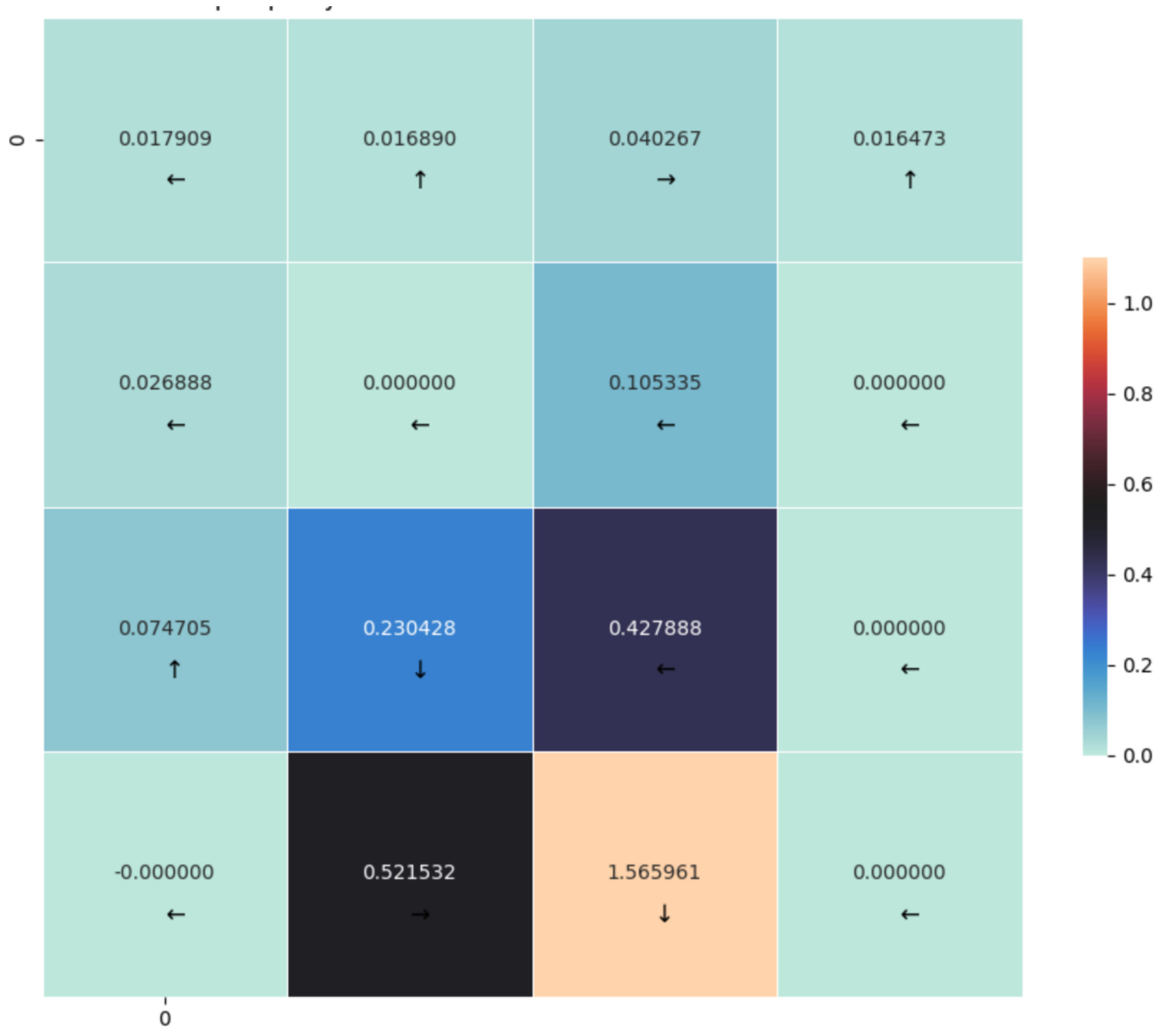


2. **Policy Evaluation:** compute the value of (i) the optimal policy obtained using QVI, and (ii) a uniformly random policy where actions have equivalent probaility.
   a. By solving a linear system of equations

Optimal Policy Solution through linear system of equations:



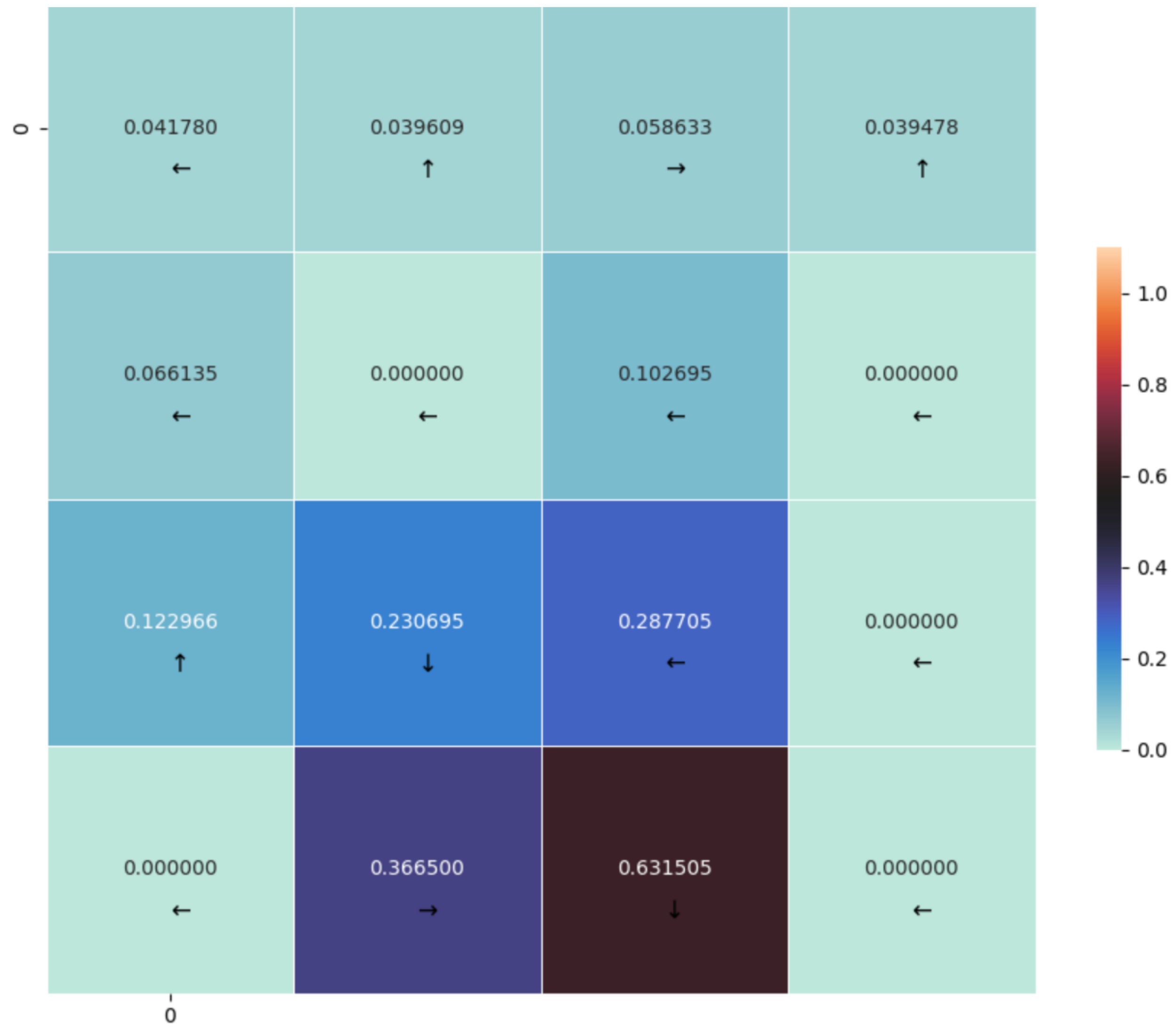
Random Policy Solution



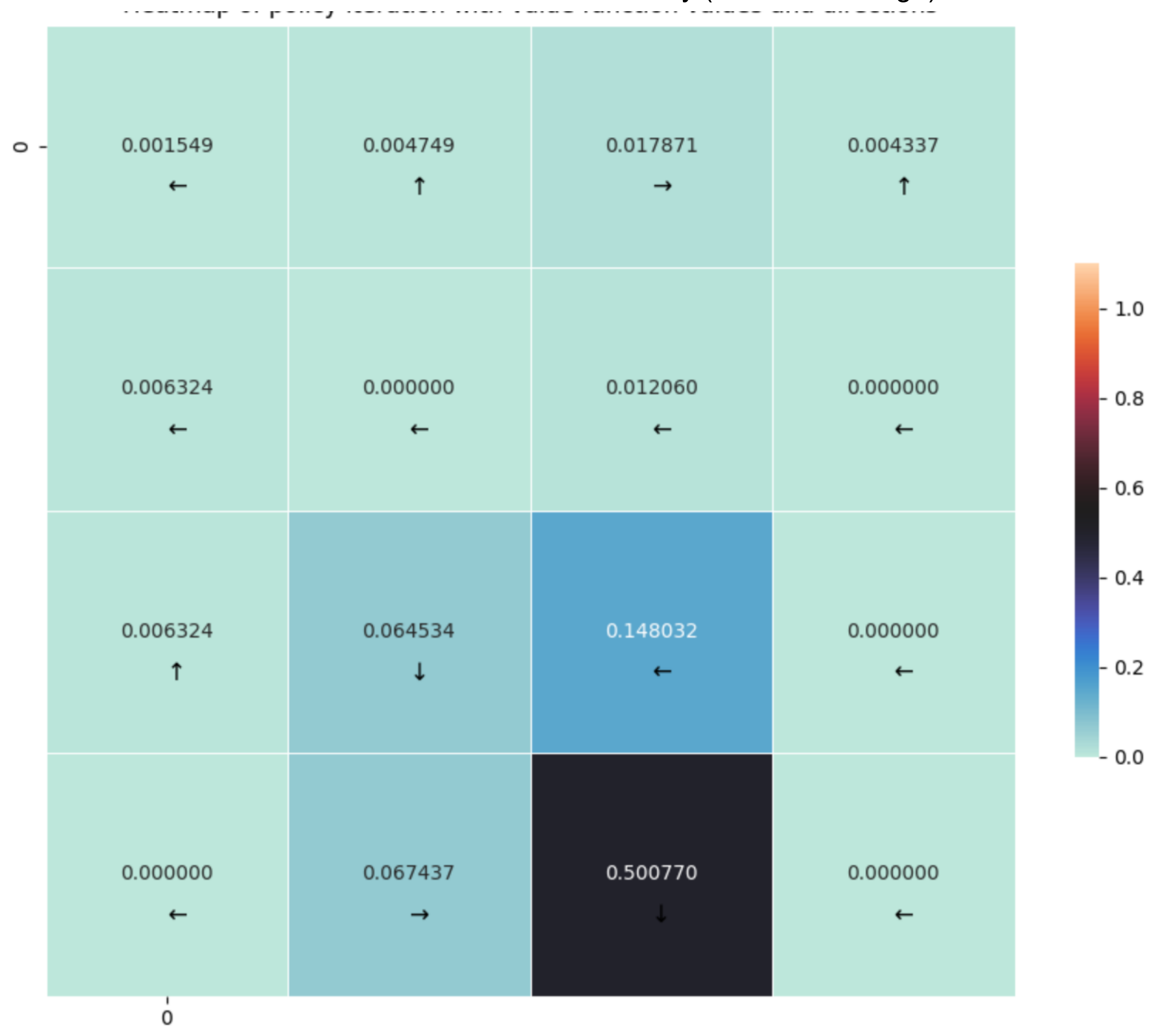
b. By the iterative approach (programmatically, policy evaluation code)

The iterative approach is implemented in the code under the PolicyEvaluation function in utils. The resulting value functions are shown using the provided function.

Value Function of the optimal policy (converged in 15 iterations):



Value Function of the Random Policy (Did not converge)



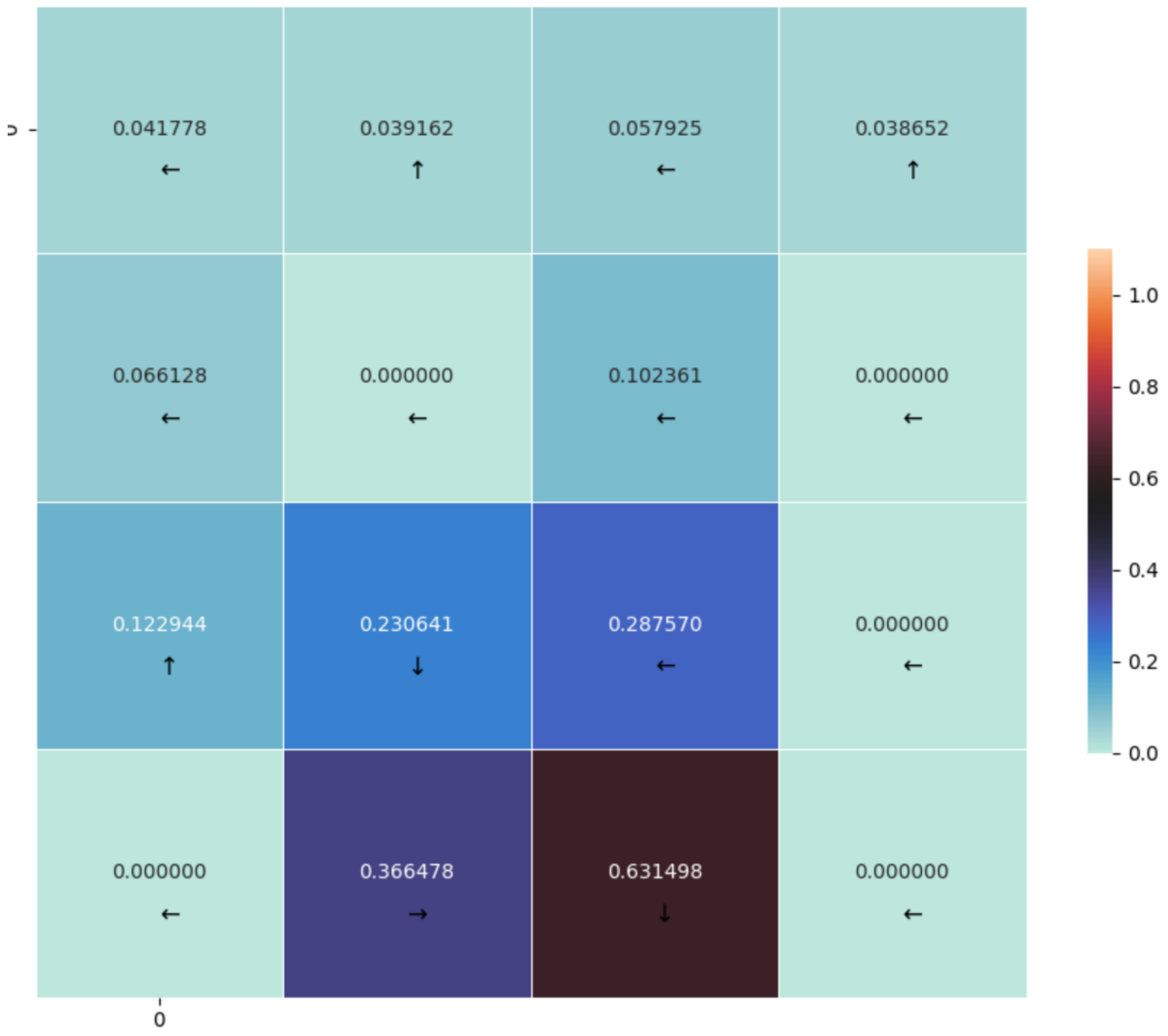
It should be noted that the policy shown in the image is not the random policy, but the value function was generated in accordance with it.

c. Which method is better and why?

The iterative approach was far easier to setup and run, however it will take longer to compute from a computer's standpoint. Additionally, the iterative approach converges within a tolerance, so is not quite as precise, especially for the random case. That being said, the system of equations was far harder to setup. Additionally, the system of equations method requires knowledge of the full transition matrix before operations can be done, which if it is not known could add to preprocessing. Iterative policy evaluation still needs to use elements of the transition function, though it does not need to know the whole thing at once. Overall, the preferred method would be the system of equations for its speed and accuracy.

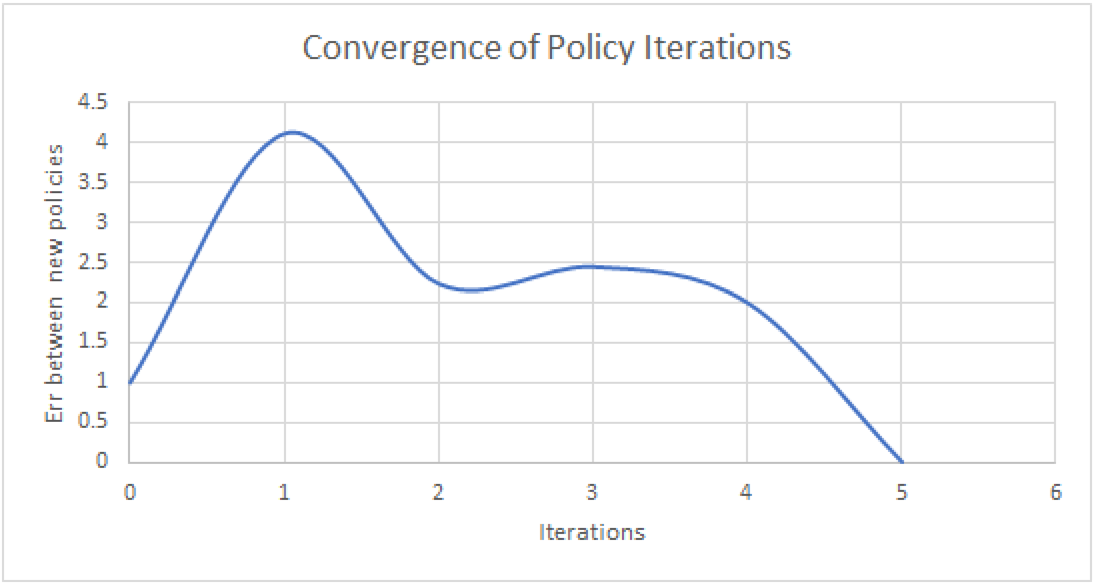
3. **Policy Iteration (PI):**
    a. What is the optimal policy and value function?

The optimal policy and value function gathered through policy iteration is:



This was done through the Policy Iteration Program in the utils file. It converged in 5 iterations of the PolicyIterations function, which in turn required a total of 57 iterations of the PolicyEvaluation function.

b. Compare the convergence of QVI and PI

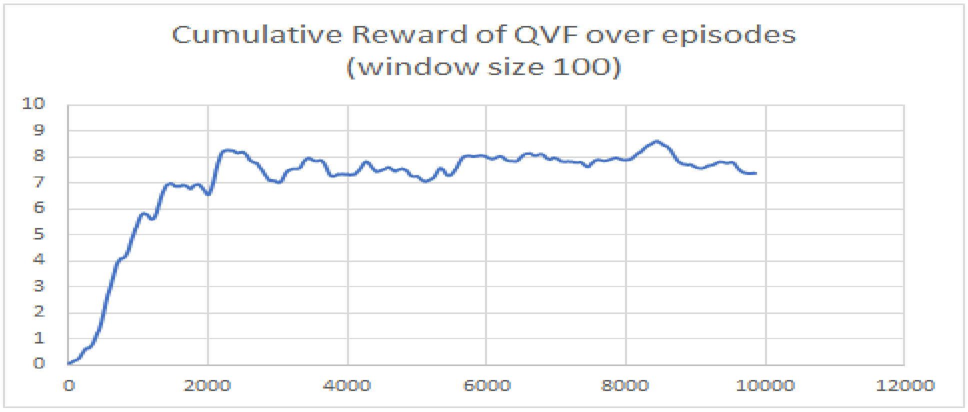


The above graph is not a great indicator of the convergence of PI, as the error metric I am using is not checking on the changes to the value function, but rather to the policy.
PI converged more quickly than QVI technically. However, PI took more total iterations to converge if we consider the necessity of the Policy Evaluation calls to converge.
If Policy Evaluation were a quick process, then it could be said that PI converges quicker, but in my implementation it seems that QVI converges more quickly due to requiring less total calls.

Learning:

4. **Tabular Q-Learning:**
   a. Plot G_k, where G_k is the cumulative reward obtained in episode k. Use a sliding average to obtain smooth plots.

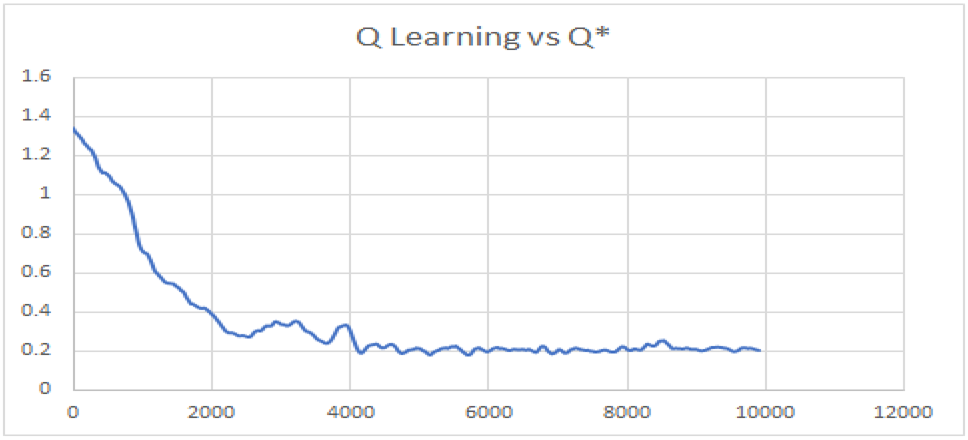
Resultant Value Function from Tabular Q-Learning. The alpha value used for this was .3. The window size for the graph was 100



Resultant Q-Value Function:

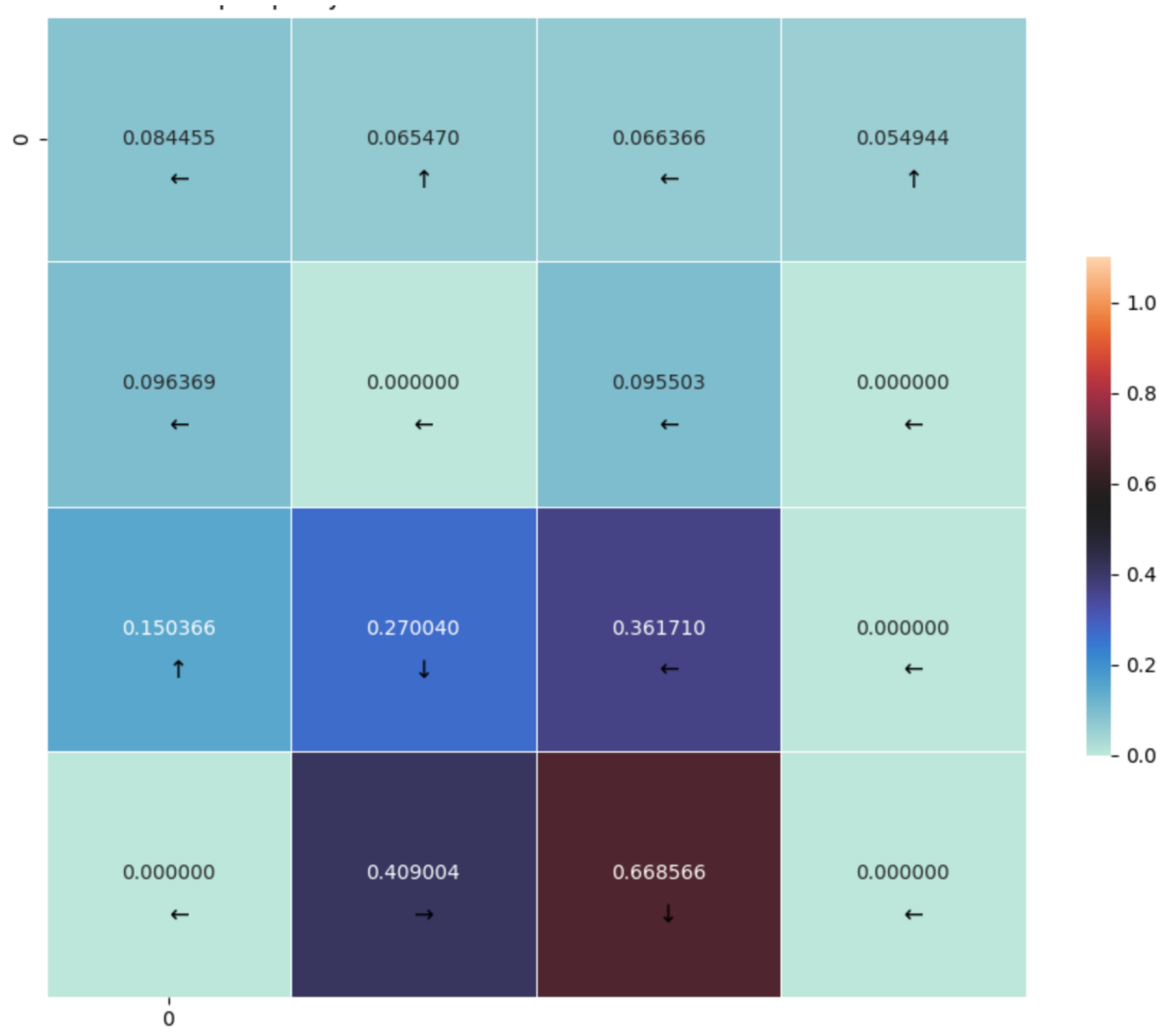
   b. Plot $||Q_k - Q^*||$, where k is the episode number and Q^* is the optimal Q function from QVI.

I took this norm to be the L2 norm, similar to above. As you can see, it does begin to approach Q* but due to some of the variables used in the calculation, it is converging to a constant rather than 0. That being said, this still demonstrates that it converges to Q*. The window size was 100.

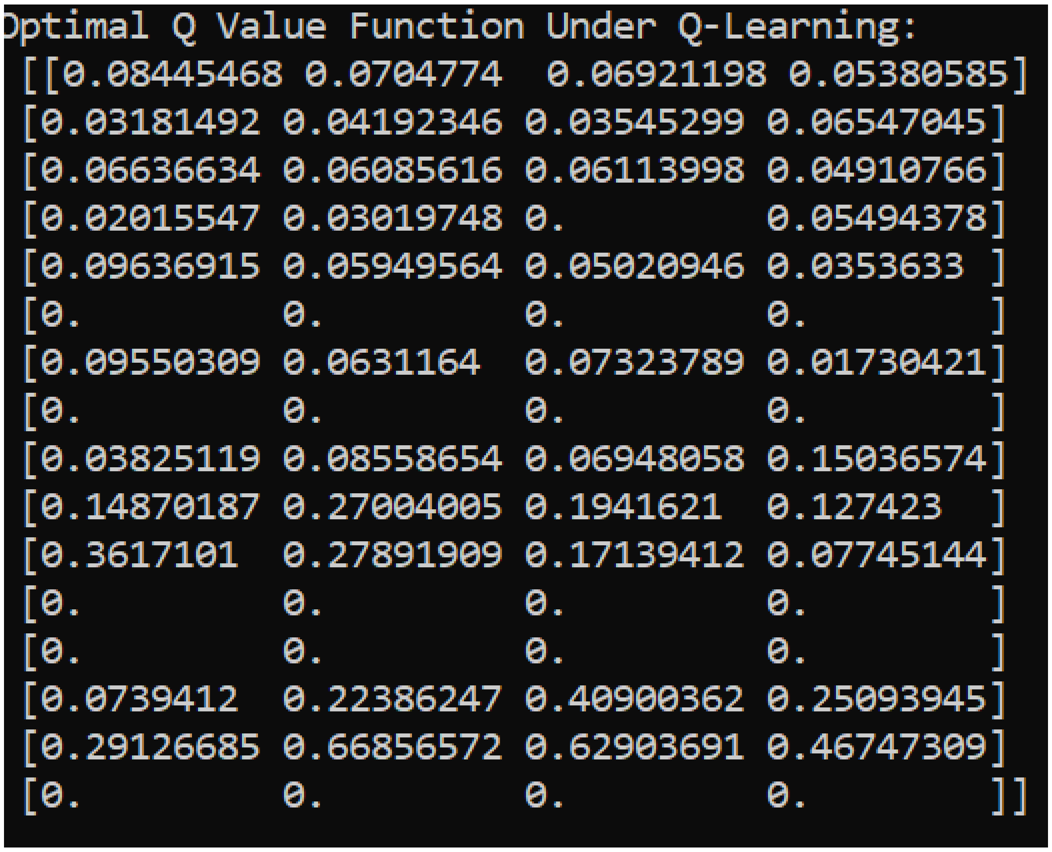


c. What is the policy and Q-value function obtained at the end of the learning? Are you able to learn the optimal policy?

These are found using an e-greedy implementation, with an epsilon value of .9, an alpha of .1, and a gamma of .85:

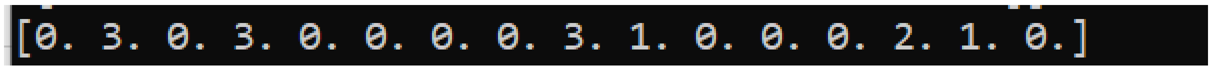


Q-Value Function:



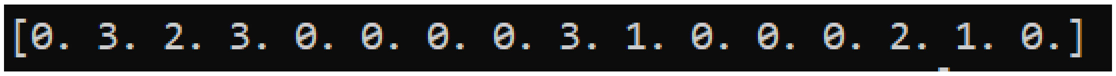
```
Optimal Q Value Function Under Q-Learning:
 [[0.08445468 0.0704774  0.06921198 0.05380585]
 [0.03181492 0.04192346 0.03545299 0.06547045]
 [0.06636634 0.06085616 0.06113998 0.04910766]
 [0.02015547 0.03019748 0.         0.05494378]
 [0.09636915 0.05949564 0.05020946 0.0353633 ]
 [0.         0.         0.         0.        ]
 [0.09550309 0.0631164  0.07323789 0.01730421]
 [0.         0.         0.         0.        ]
 [0.03825119 0.08558654 0.06948058 0.15036574]
 [0.14870187 0.27004005 0.1941621  0.127423  ]
 [0.3617101  0.27891909 0.17139412 0.07745144]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.0739412  0.22386247 0.40900362 0.25093945]
 [0.29126685 0.66856572 0.62903691 0.46747309]
 [0.         0.         0.         0.        ]]
```

Policy:



```
[0. 3. 0. 3. 0. 0. 0. 0. 3. 1. 0. 0. 0. 2. 1. 0.]
```

Optimal Policy:



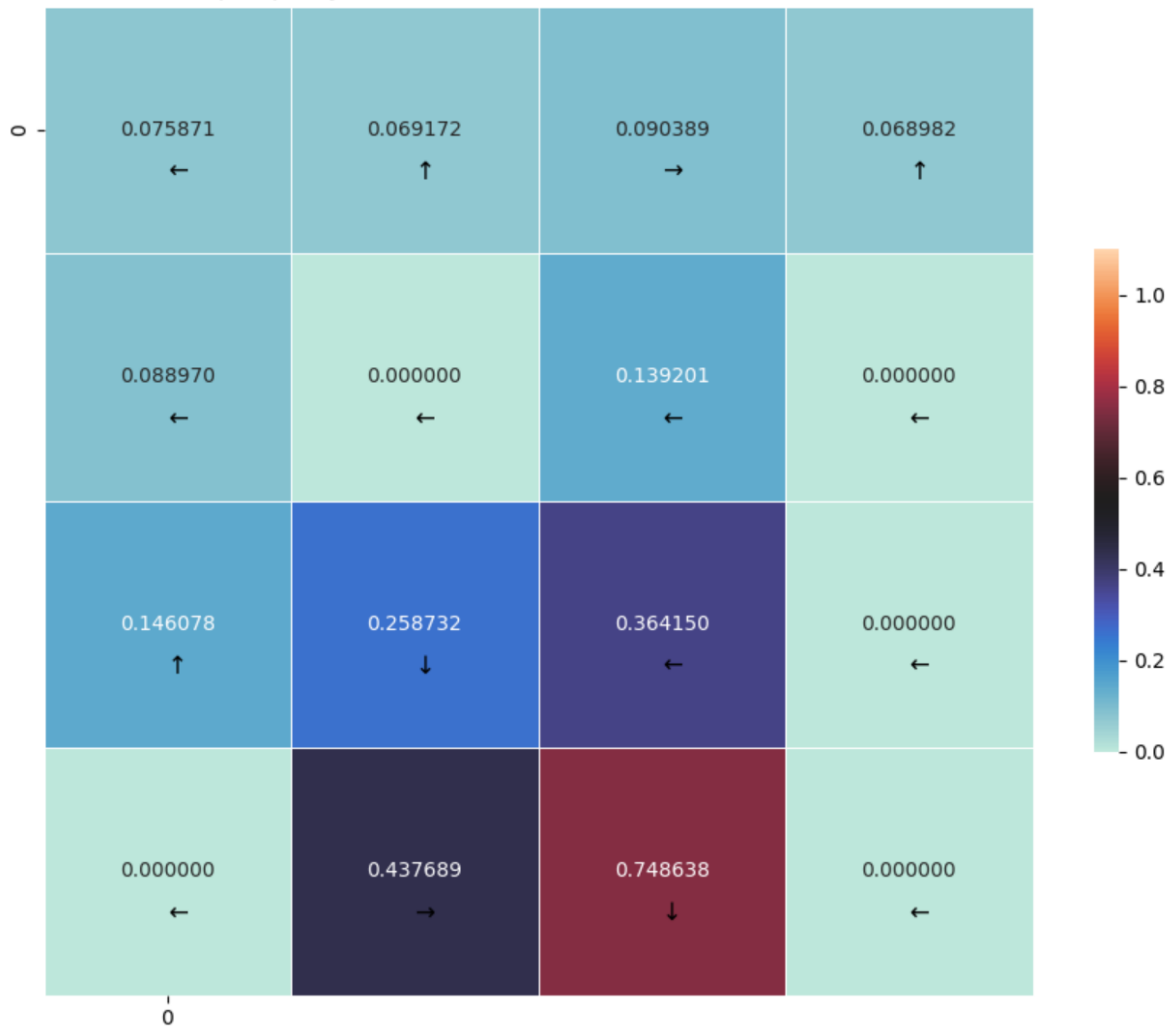
```
[0. 3. 2. 3. 0. 0. 0. 0. 3. 1. 0. 0. 0. 2. 1. 0.]
```

The policy that was computed here does not match the optimal policy found in QVI. There is one difference at state 2. This is because, when looking at the difference in the optimal value function for state 2, there is an incredibly small difference between actions 0 and 2. For something like Q-learning, which makes its decisions by sampling, this small difference could easily go either way. If you look at the values in the value function gathered by Q-learning, there is still a very minor difference between actions 0 and 2. A way to alleviate this issue would be to make more samples or to average consecutive runs.

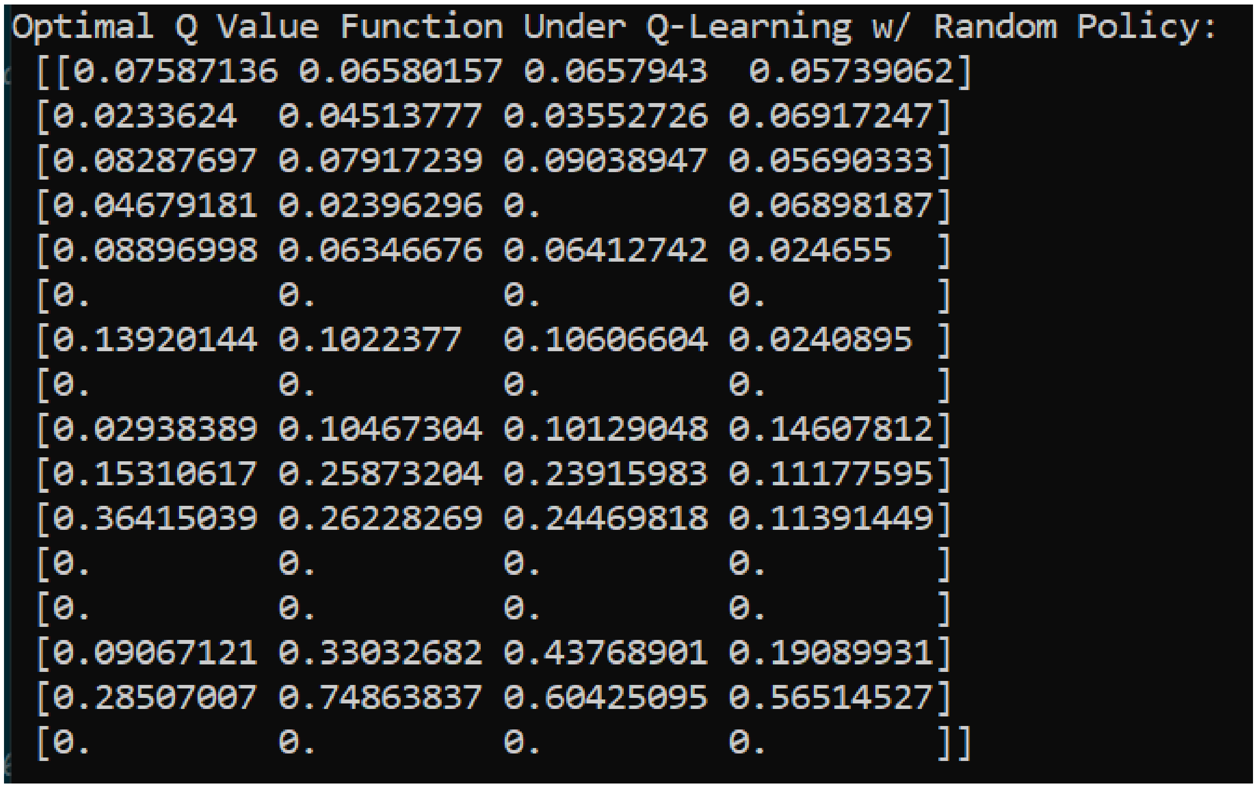
It is worth mentioning that the algorithm does work easily on non-stochastic environments and for actions that are not so close in value.

5. **Behavior Policy:** Implement tabular Q-learning with a uniformly random policy as the behavior policy. Compare the convergence with the ϵ-greedy exploration approach. Explain observations and inference. Can you implement a better behavior policy and show its effectiveness?

Random policy Q-learning value function result

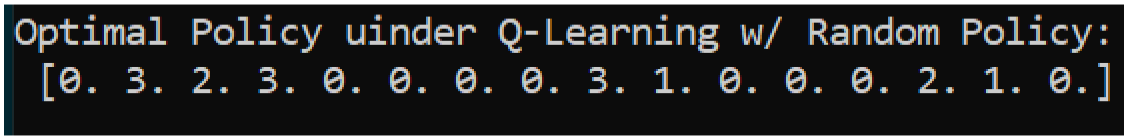


Random Policy QVF:

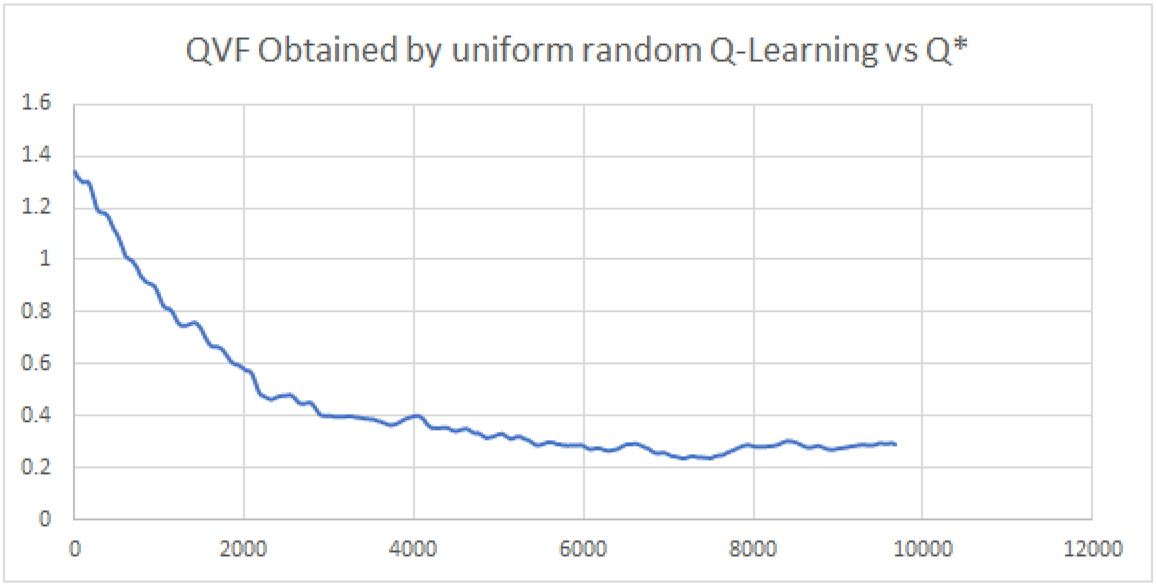


```
Optimal Q Value Function Under Q-Learning w/ Random Policy:
 [[0.07587136 0.06580157 0.0657943  0.05739062]
 [0.0233624  0.04513777 0.03552726 0.06917247]
 [0.08287697 0.07917239 0.09038947 0.05690333]
 [0.04679181 0.02396296 0.         0.06898187]
 [0.08896998 0.06346676 0.06412742 0.024655  ]
 [0.         0.         0.         0.        ]
 [0.13920144 0.1022377  0.10606604 0.0240895 ]
 [0.         0.         0.         0.        ]
 [0.02938389 0.10467304 0.10129048 0.14607812]
 [0.15310617 0.25873204 0.23915983 0.11177595]
 [0.36415039 0.26228269 0.24469818 0.11391449]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.09067121 0.33032682 0.43768901 0.19089931]
 [0.28507007 0.74863837 0.60425095 0.56514527]
 [0.         0.         0.         0.        ]]
```

Random Policy Resultant Policy:

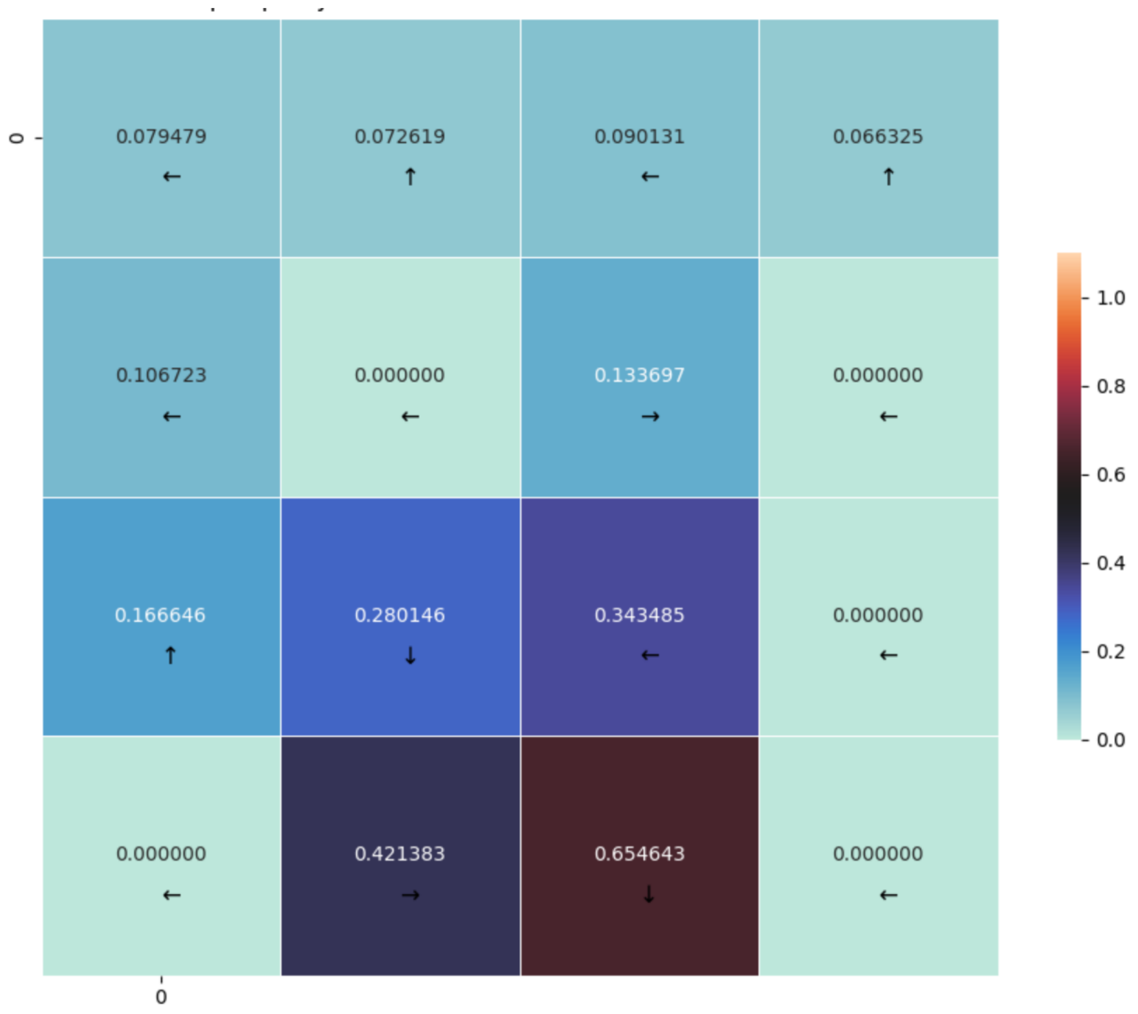


```
Optimal Policy uinder Q-Learning w/ Random Policy:
 [0. 3. 2. 3. 0. 0. 0. 0. 3. 1. 0. 0. 0. 2. 1. 0.]
```

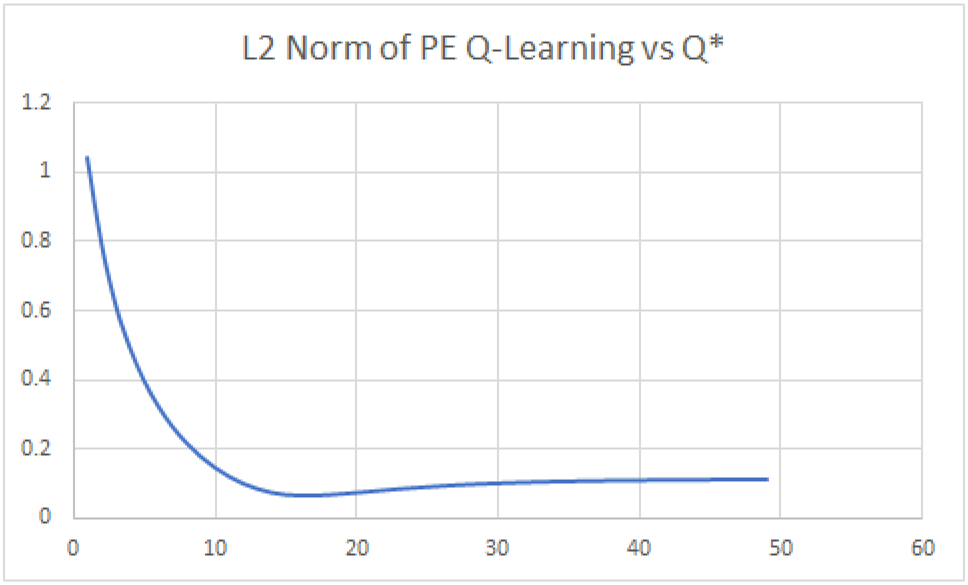


As you can see by comparing the graphs, the random Q-learning converged slower, and to an overall lesser degree. This is because it is more akin to exploring the entire state action space, which causes it to take longer to converge. It is worth mentioning that while the Q-value function from uniform random policy diverged slightly more than the Q-Value Function gained from the epsilon-greedy Q-Learning, it did result in the optimal policy. The reason that the random sampling did, while the non-random did not, is that the small amount of bias garnered in the initial decision making process was just enough to sway the decision in cases where several actions have very similar results. This is not as prevalent in random sampling, as it's decisions are not based on any prior decisions.

There were several attempts at making a better Q-Learning function. One of which involved guessing at the policy from sampling the environment. The initial 'phase' consisted of making random movements in the environment to make a guess of the transition matrix. After this estimated matrix was made, it performed Q-Value iteration. This technique ended up having a higher accuracy and converging significantly faster. A screenshot of its value function is shown below:



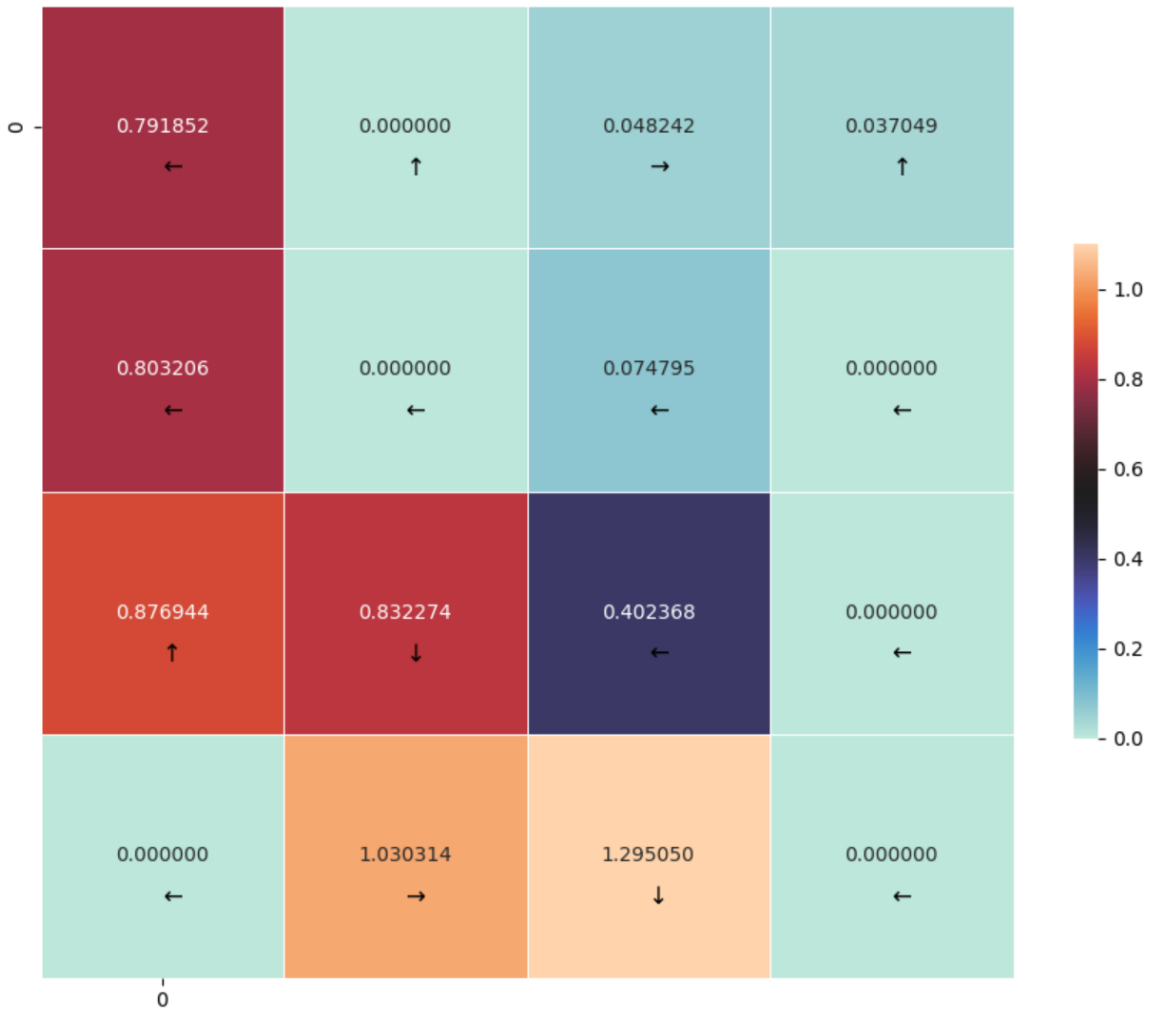
L2 Norm of the new method versus the optimal Q-Value Function:



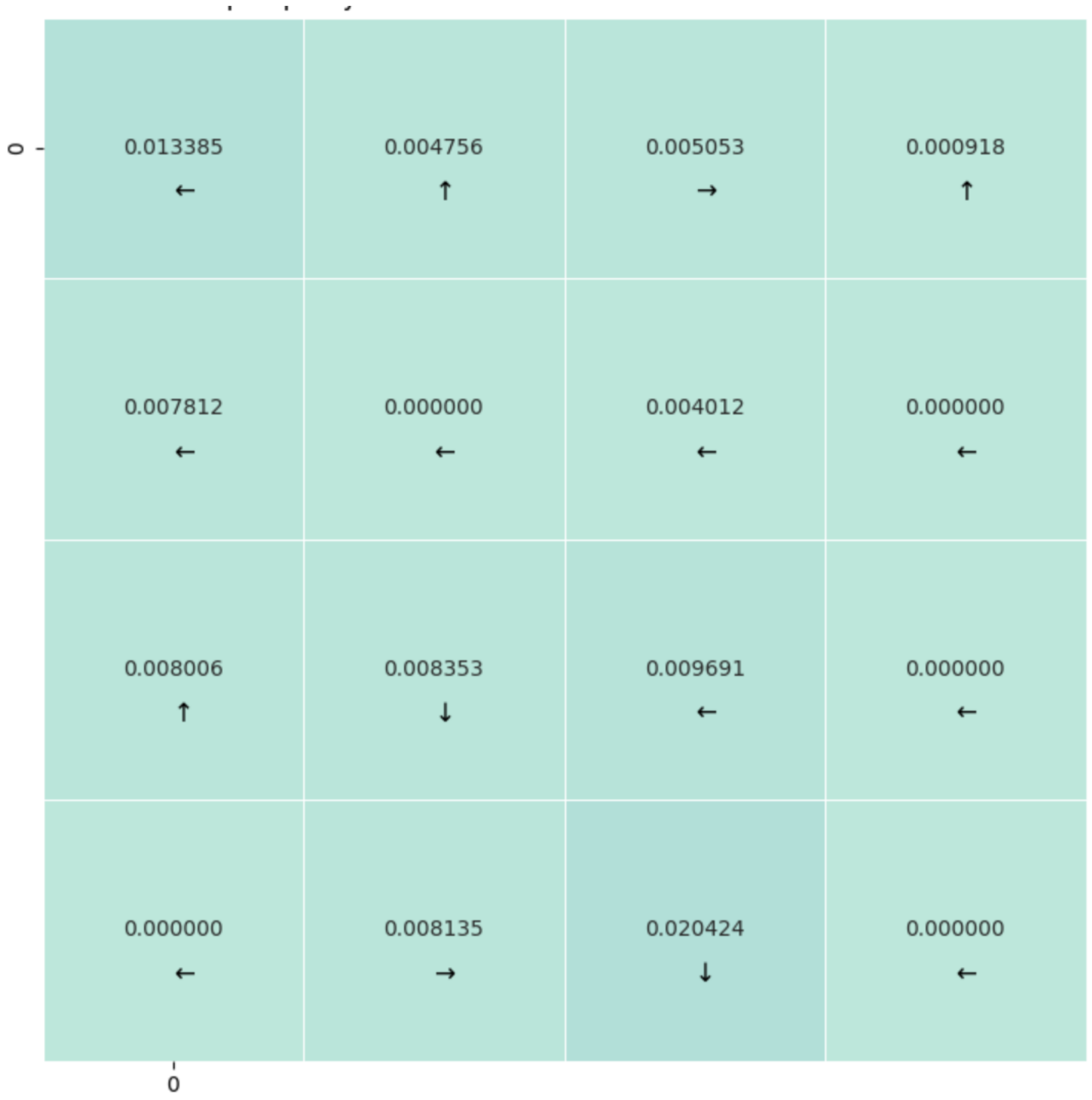
While it did approach Q* faster and was closer overall, this variation still did not result in optimal policy.

6. **TD-Learning:** Consider the following policies: (i) the optimal policy obtained from QVI, and (ii) a uniformly random policy where each action is taken with equal probability. Learn the value of these policies using:
    a. Monte Carlo (MC) Learning

Using the monte carlo learning method, I arrived at the following value functions.
Optimal Policy:



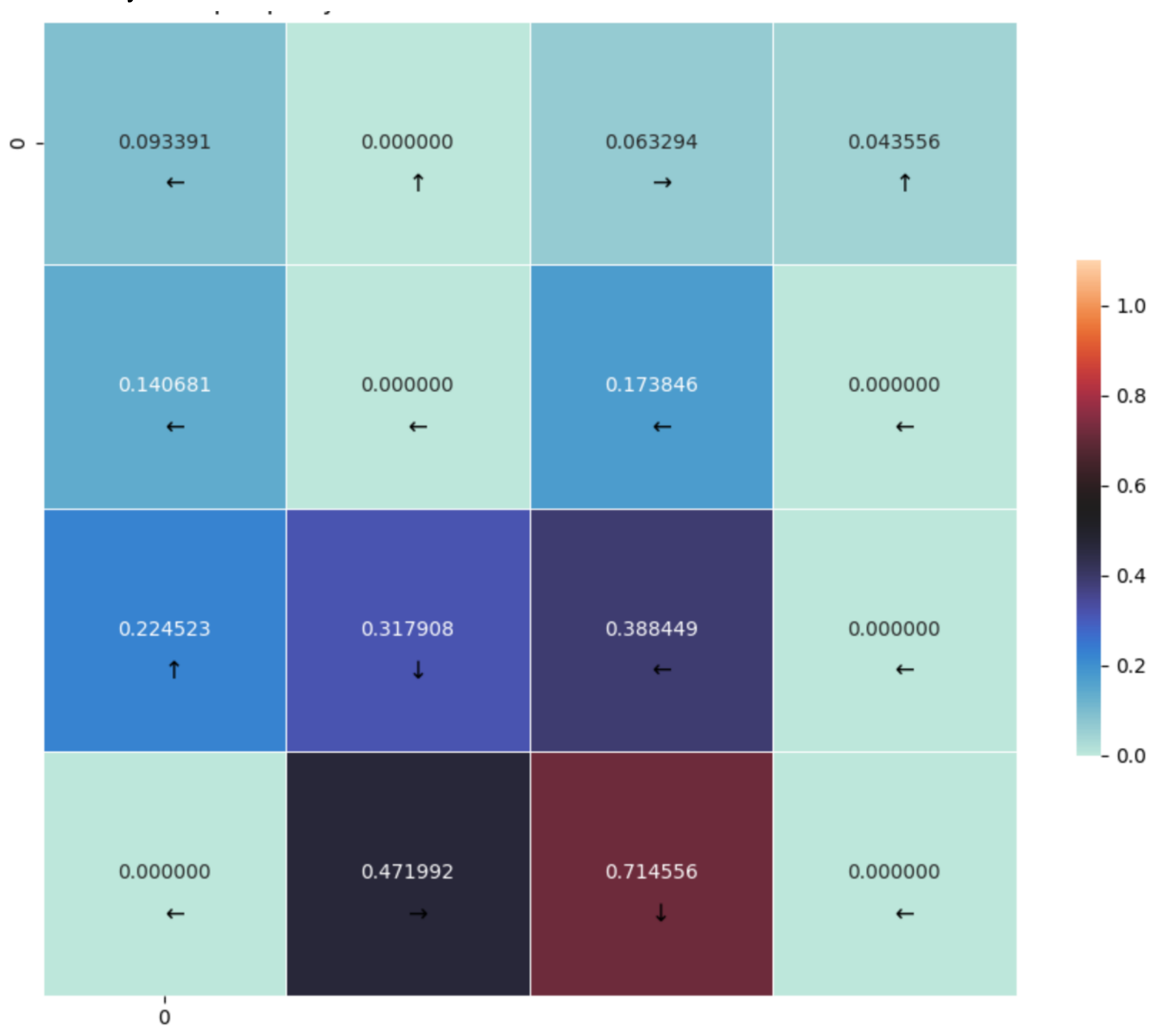
Uniform Random Policy (note that the arrows correspond to the optimal policy, not uniform random):



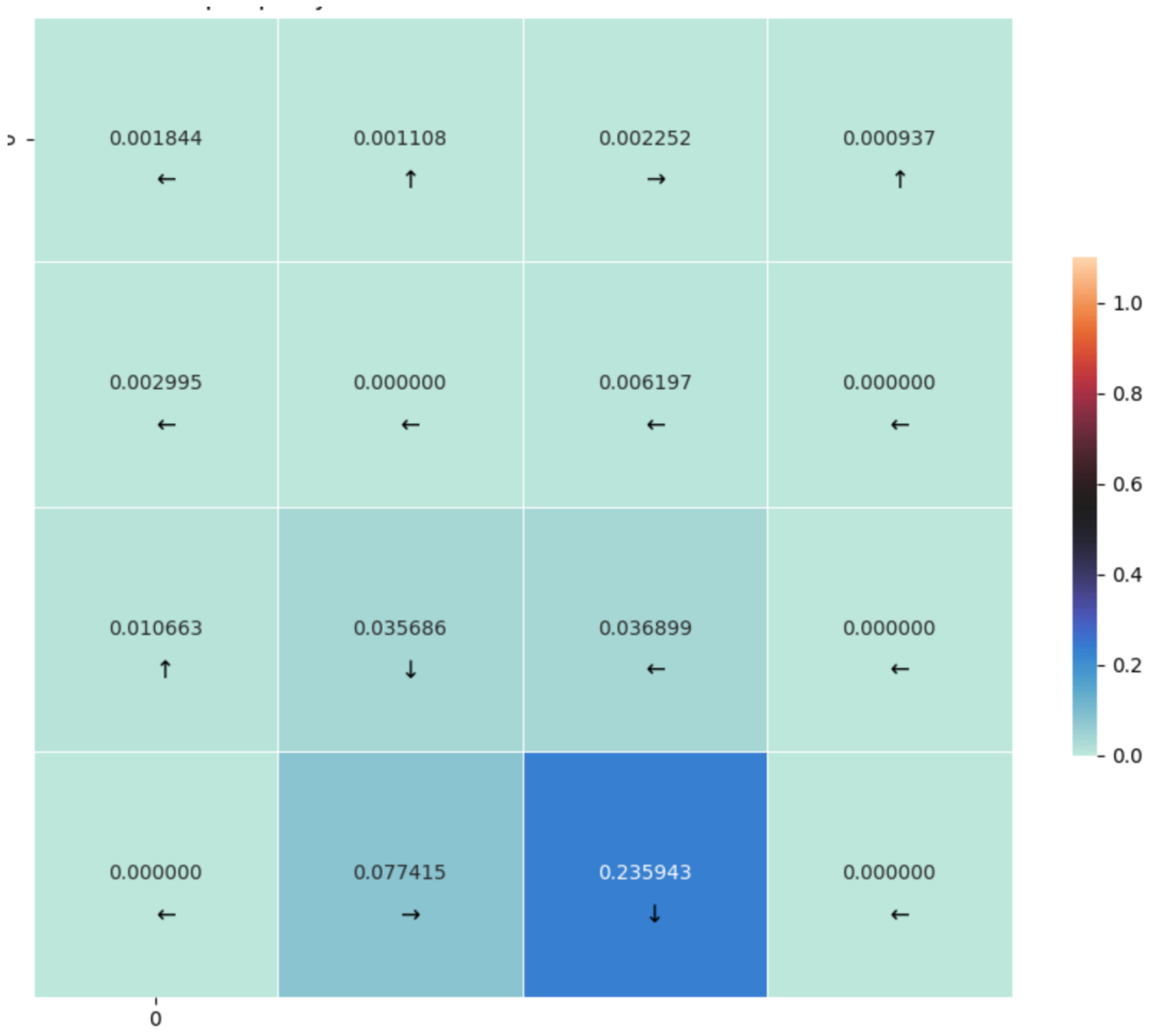
0.013385 ←	0.004756 ↑	0.005053 →	0.000918 ↑
0.007812 ←	0.000000 ←	0.004012 ←	0.000000 ←
0.008006 ↑	0.008353 ↓	0.009691 ←	0.000000 ←
0.000000 ←	0.008135 →	0.020424 ↓	0.000000 ←

b. Temporal Difference (TD) Learning (alpha was .1)

Optimal Policy:



Uniform Random (note that the arrows are shown in accordance with the optimal policy):



c. What are the trade-offs between MC and TD?

The Monte Carlo method tends to be more simplistic and easier to implement. There is no need to rely on the modification of the additional alpha parameter, and there is no need to know dynamic programming techniques.

On the other hand, while being more complicated, TD learning does not require the end of an episode to update the value function and can update the value function on the fly.

Given time, both will converge to the value function. There is no proof as to one converging quicker than the other.