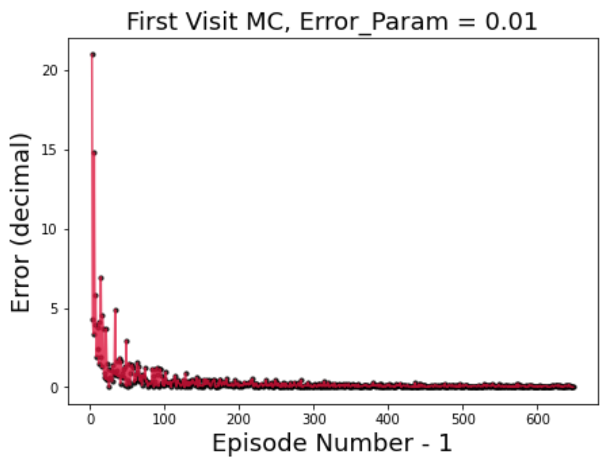
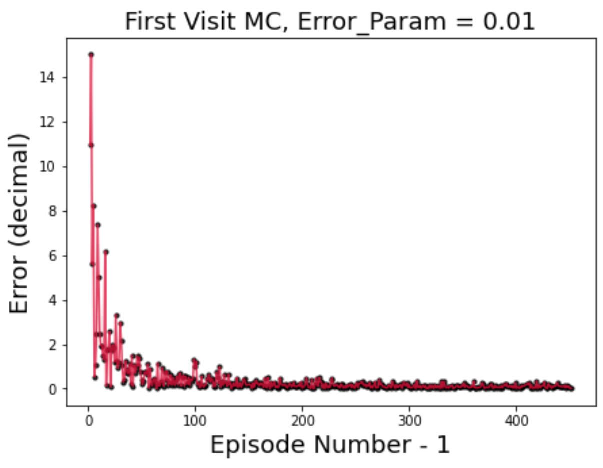
Kim Conger
EECS-658
Assignment 8
December 2022

Part One

Question 1

I ultimately settled on a somewhat arbitrary convergence condition for both MC-FirstVisit and MC-EveryVisit, since, unlike the previous assignment's policy iterations, the high variability of results for both algorithms caused nontrivial consistency difficulties. I begin with tracking the kth and k-1th Vgrids, then applying a bool flag created by requiring that if one index had a True, then that corresponding index in both Vgrids had to be a nonzero element. I then calculate the mean of the difference between the nonzero elements, and place its absolute value into an error_list array. The convergence condition necessitates that the lowest index at which an error_list[index] and error_list[index-1] are less than 0.01 will be the termination index. The first two plots below illustrate the error convergence for First Visit MC, using two different iterations as data examples. This was effectively the indication that my convergence approach at least held enough merit to continue with it.

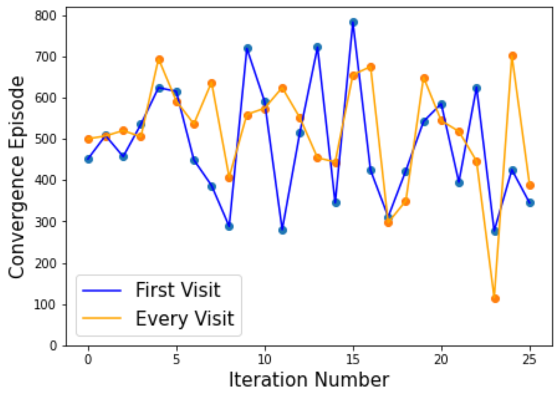




Part Two

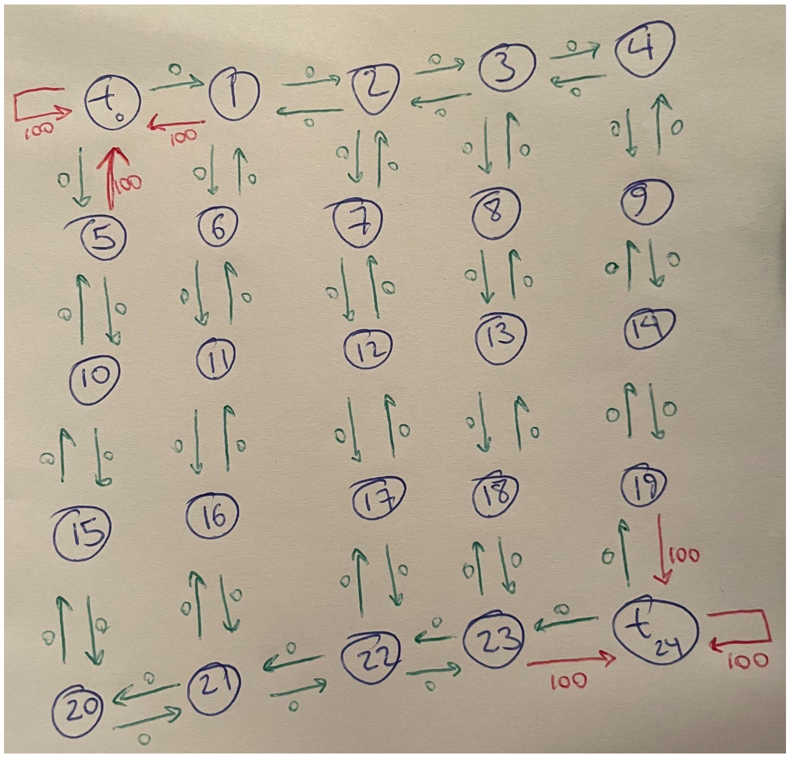
Question 2

As MC relies on random walks, I am not surprised to find that the convergence episode is not consistent between iterations for the same MC algorithm, as well as between the two algorithms. As the plot below indicates, the fruits of both EV and FV are embedded within a sea of variability that is almost entirely contained within 300 and 700. At least both are comparable with respect to the amount of generated noise about a mean of around 500.



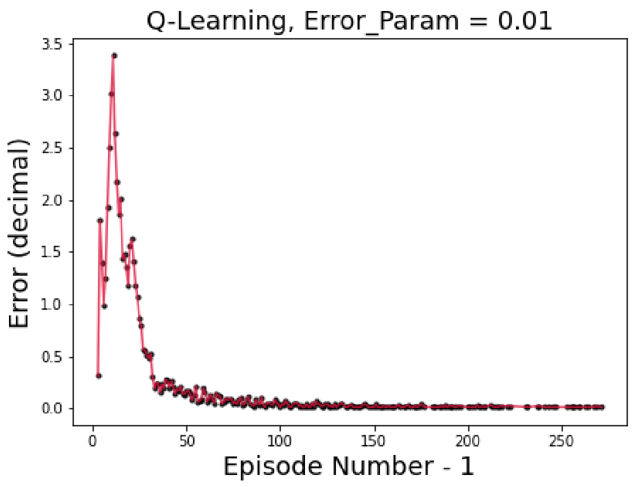
Part Three

Question 3



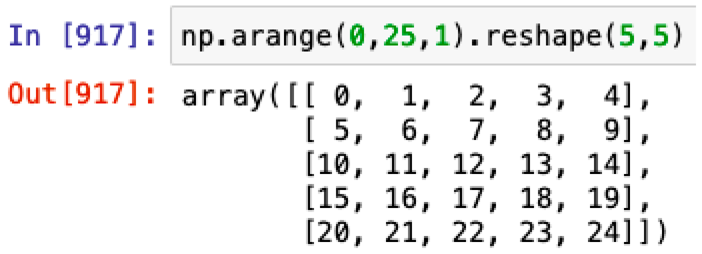
Question 4

The convergence method here is a bit similar to the previous parts in that I also calculate the mean of the difference of nonzero elements between two consecutive Qgrids. I then impose a constraint that both terminal states in the current Qgrid must present with a value > 99, in addition to the current episode error < 0.01. While there is no objective assurance here that the algorithm found an optimal rather than a suboptimal grid policy, the suboptimal versions do well enough at guiding the agent to the terminal state in the least number of moves.



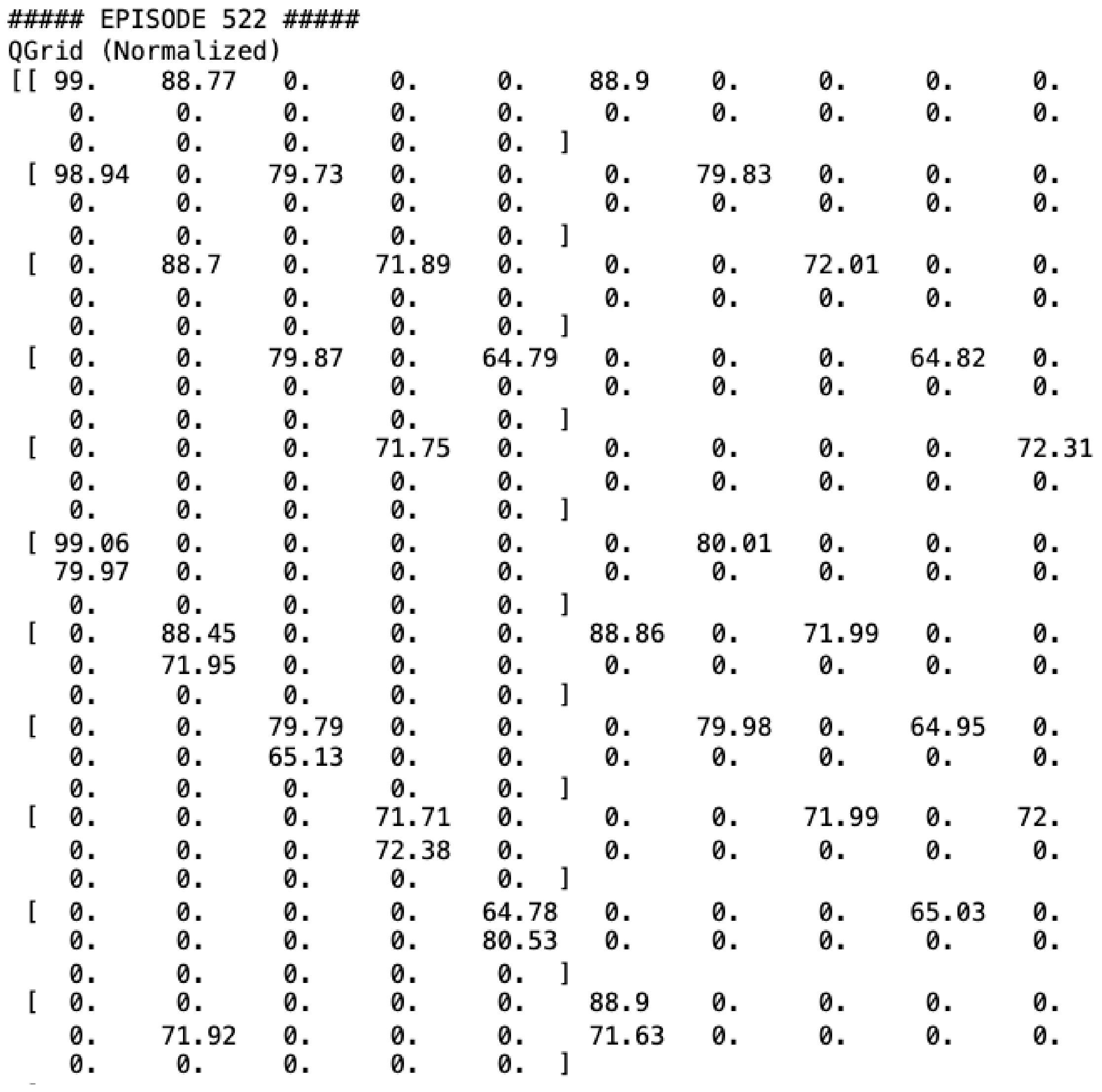
Question 5

According to a single instance of running the program, this optimal path is 7 → 6 → 5 → 0, following grid values Q[7,6] = 79.98 → Q[6,5] = 88.86 → Q[5,0] = 99.06. This indeed happens to be the optimal path, as seen below:



```
In [917]: np.arange(0,25,1).reshape(5,5)
Out[917]: array([[ 0,  1,  2,  3,  4],
                 [ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19],
                 [20, 21, 22, 23, 24]])
```

Of note is that the alternative optimal path, 7 → 2 → 1 → 0, does have Qgrid values comparable in magnitude but are nevertheless lower than those for that first path. I include the first 11 rows below, for reference purposes.



```
##### EPISODE 522 #####
QGrid (Normalized)
[[ 99.    88.77   0.     0.     0.    88.9    0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [ 98.94   0.    79.73   0.     0.     0.    79.83   0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.    88.7    0.    71.89   0.     0.     0.    72.01   0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.    79.87   0.    64.79   0.     0.     0.    64.82   0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.    71.75   0.     0.     0.     0.     0.    72.31
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [ 99.06   0.     0.     0.     0.     0.    80.01   0.     0.     0.
   79.97   0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.    88.45   0.     0.     0.    88.86   0.    71.99   0.     0.
    0.    71.95   0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.    79.79   0.     0.     0.    79.98   0.    64.95   0.
    0.     0.    65.13   0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.    71.71   0.     0.     0.    71.99   0.    72.
    0.     0.     0.    72.38   0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.    64.78   0.     0.     0.    65.03   0.
    0.     0.     0.     0.    80.53   0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.    88.9    0.     0.     0.     0.
    0.    71.92   0.     0.     0.    71.63   0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
```

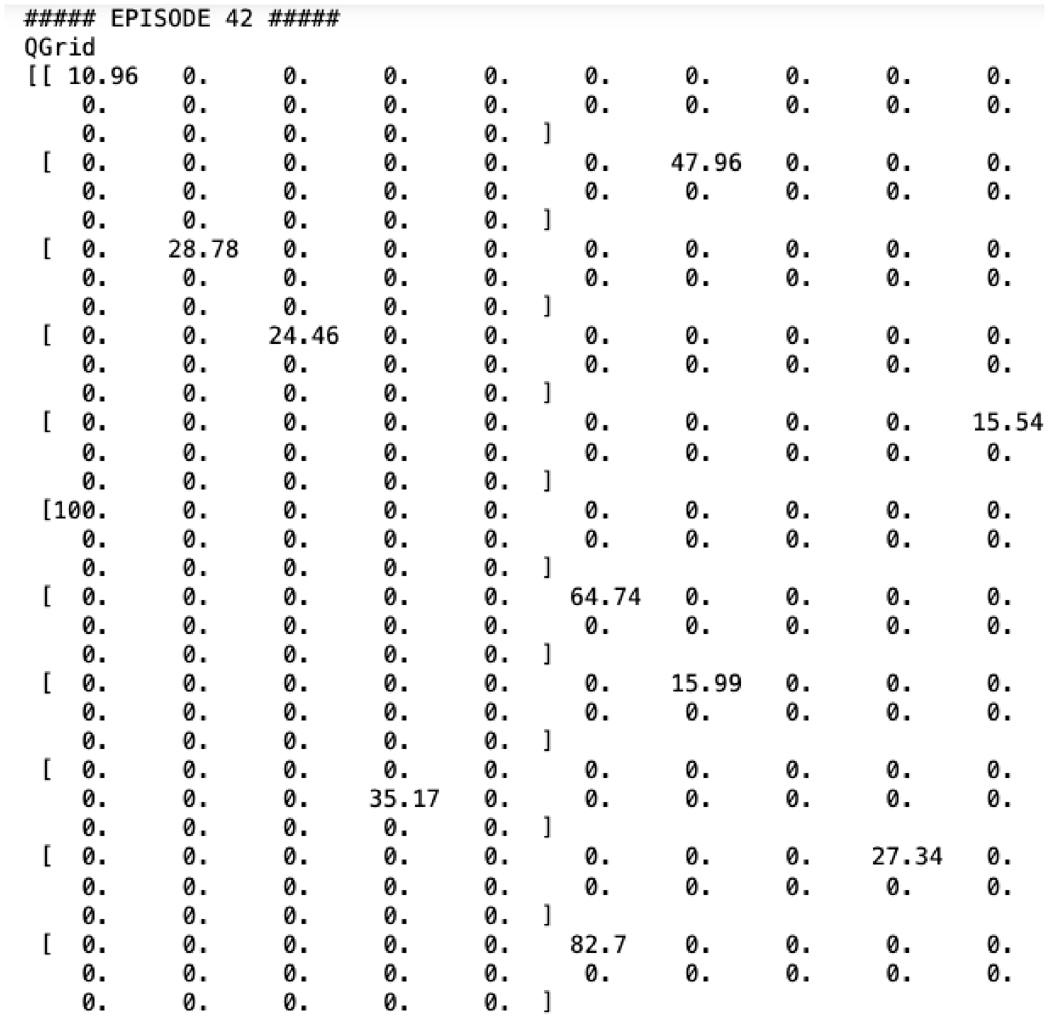
Part Four

Question 6

Following many hours of contemplation and code-revising (I wish I were joking), I have concluded that SARSA QL is just too susceptible to finding sub-optimal solutions that may not even lead to both terminal states. The high variability of each run, coupled with this aforementioned pitfall, results in a wildly different convergence criterion as compared with Part Three. I settled on terminating the program only when err_param<0.01.The minimum number of episodes must also be 50, to ensure a sufficient sampling of the grid space. There were just too many instances necessary in order to fully sample the terminal states.

Question 7

My response to this question depends on, I think, which output grid I use, as indicated. I reproduce the first 10 rows of the used Qgrid below, for which the policy's path for state 7 is 7 → 6 → 5 → 0, following grid values Q[7,6] = 15.99 → Q[6,5] = 64.74→ Q[5,0] = 100. This path is indeed optimal and matches that of Part Three; however, one should bear in mind that this same grid directs initial states beginning at [24,24], a terminal state, to [24,19] rather than back onto itself ([24,24]). To each their own.



```
##### EPISODE 42 #####
QGrid
[[ 10.96   0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.    47.96   0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.    28.78   0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.    24.46   0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.     0.    15.54
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [100.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.    64.74   0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.    15.99   0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.    35.17   0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.    27.34   0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.    82.7    0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.  ]
```

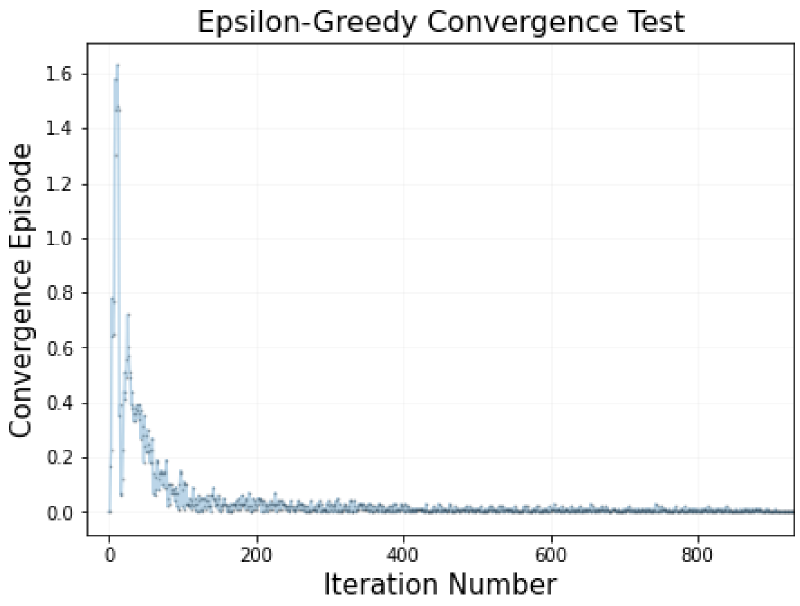
Question 8

Not surprisingly, unless strictly by happenstance during some instance of running RLAlgoComp.py, Parts Three and Four did not converge in the same number of episodes. There is, simply, too much variability in output Qgrids, especially for Part Four and its sub-optimal tendencies. In just one instance, Part Three yielded a convergence episode of 564 while Part Four managed the same error convergence in just 42 episodes. Just as with the MC comparison plot in Part Two, both QL and SARSA-QL are too inconsistent.

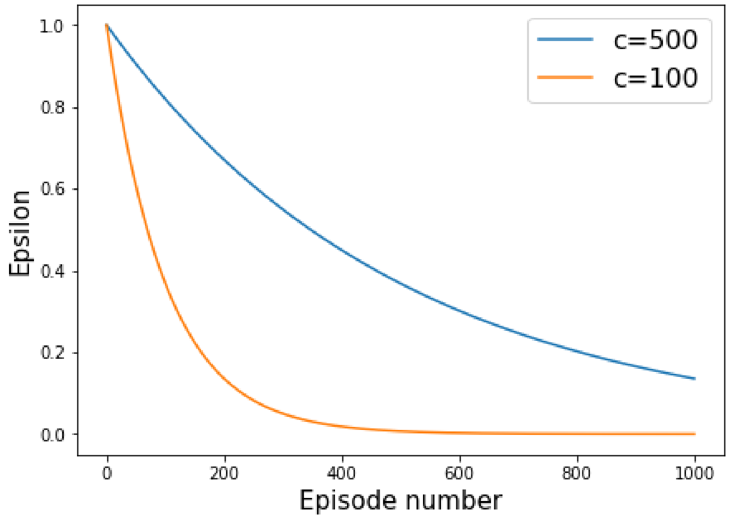
Part Five

Question 9

Success was only marginally less elusive with the epsilon-greedy algorithm compared with Part Four, as there was still the problem of . My convergence conditions here consisted of the same as Part Three, but with the minimum threshold values for the terminal states being 80 rather than 99. Even a slightly lower threshold generates an operational ‘optimal’ policy grid, but I found that 80 was decent enough insurance that the grid was reasonably close to the optimal grid, at least when the terminal states were populated.

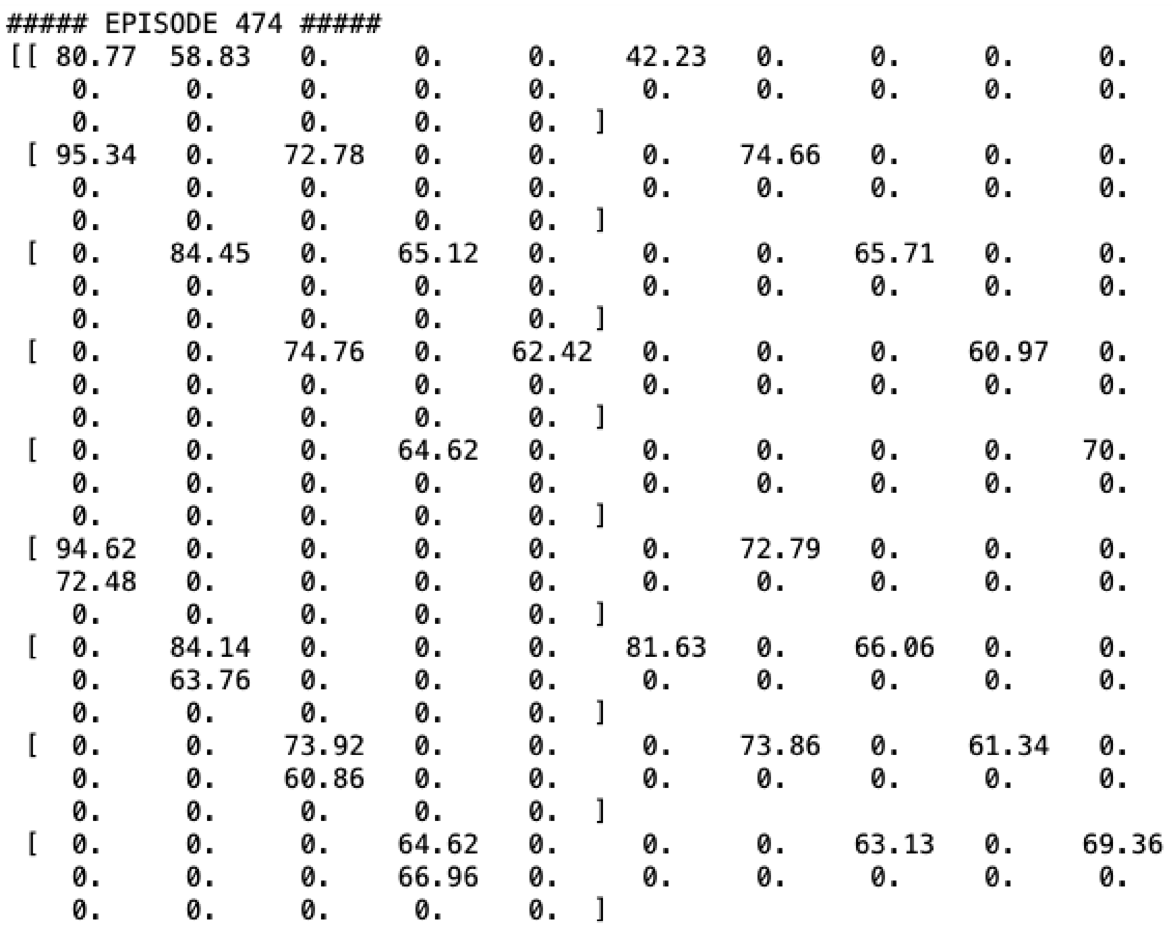


When evaluating the epsilon decay parameter c, I noticed that a sub-optimal policy rather than one with missing terminal state values was much more likely if c=500 as opposed to the recommended value c=100 from the lecture slides. As such, I set c=500 as the default.



Question 10

One instance of running the decaying epsilon-greedy code yields the 5x5 Qgrid, of which I pasted 9 lines below. This policy's path for state 7 is 7 → 2 → 1 → 0, following grid values Q[7,2] = 73.92 → Q[2,1] = 84.45→ Q[1,0] = 95.34. This path is different from Part Four but is in agreement with the 'alternative' path of Part Three. And indeed, the 'alternative path' for Part Five is in agreement with both Three and Four.



```
##### EPISODE 474 #####
[[ 80.77  58.83   0.     0.     0.    42.23   0.     0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [ 95.34   0.    72.78   0.     0.     0.    74.66   0.     0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.    84.45   0.    65.12   0.     0.     0.    65.71   0.     0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.     0.    74.76   0.    62.42   0.     0.     0.    60.97   0.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.     0.     0.    64.62   0.     0.     0.     0.     0.    70.
    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [ 94.62   0.     0.     0.     0.     0.    72.79   0.     0.     0.
   72.48   0.     0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.    84.14   0.     0.     0.    81.63   0.    66.06   0.     0.
    0.    63.76   0.     0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.     0.    73.92   0.     0.     0.    73.86   0.    61.34   0.
    0.     0.    60.86   0.     0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
 [  0.     0.     0.    64.62   0.     0.     0.    63.13   0.    69.36
    0.     0.     0.    66.96   0.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.   ]
```

Question 11

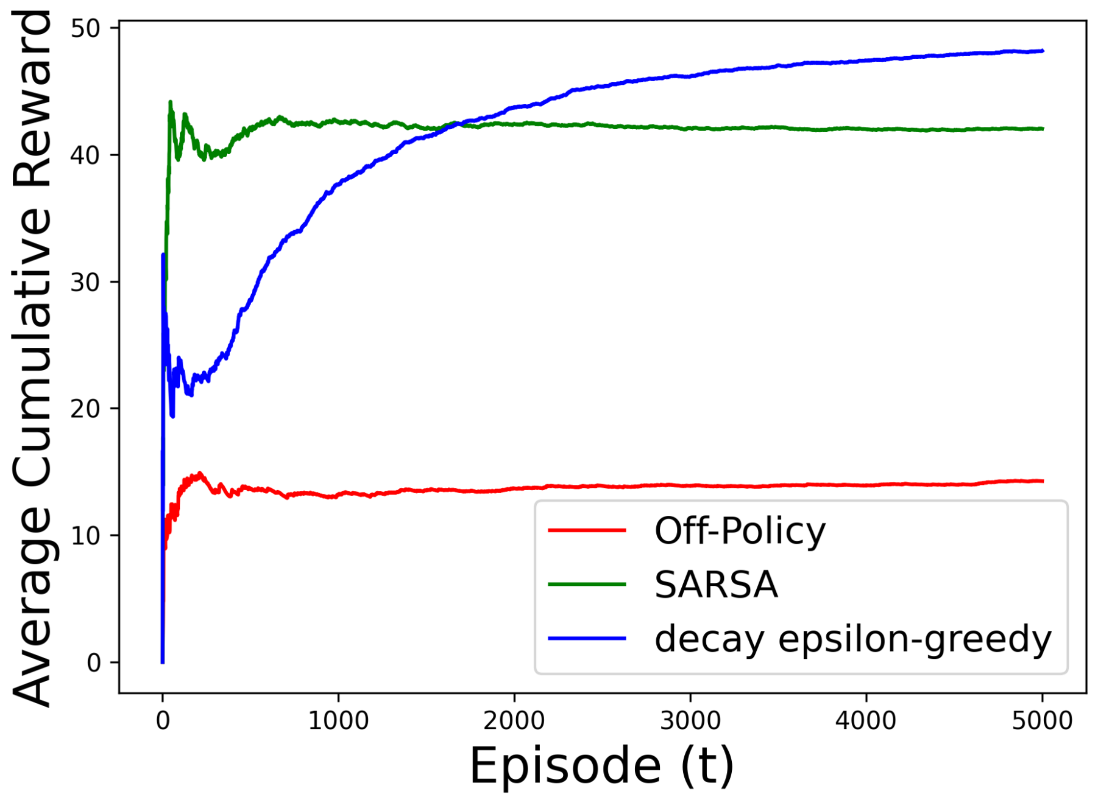
Thinking about **exact** convergence episodes, Parts Three, Four, and Five all tended to converge at different spots; however, the discrepancies were only in part due to the intrinsic variability in outputs. Due to the lenient convergence conditions I imposed on Part Four, this policy required about one magnitude less episodes to converge relative to the counterparts that incorporated off-policy QL. If I ran Part Five with c=100 and the conditions of Part Four, then the number of episodes were comparable to Part Four; and if I ran Part Five with c=500 with the conditions of Part Three, then the number of episodes were comparable to Part Three.

Part Six

Question 12

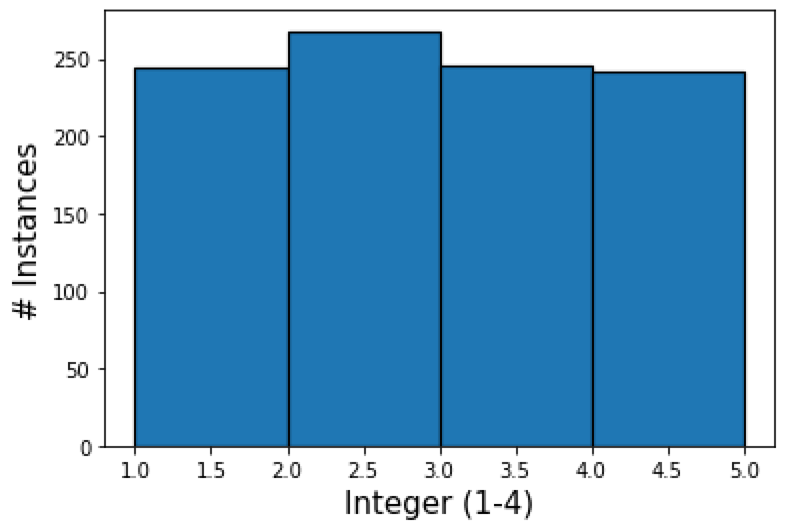
Firstly, note that the plot below is but one of a number of possible outputs due to the intrinsic variability of the algorithms. Somewhat of a consistent pattern, however, is that the off-policy curve is far below the results of QL approaches that incorporate on-policy techniques. I think this outcome is expected, since off-policy episodes consist entirely of random steps that pay no heed to the developing paths that could provide the agent with a higher reward. SARSA did precisely the opposite, settling into an optimal action quickly without the exploration that could have placed it instead in a more optimal policy with higher reward.

The decaying epsilon-greedy algorithm merges both on- and off-policy techniques, which accounts for the slower ascent into a reward plateau. In this particular instance, by around the 1600th episode, the curve rises above the sub-optimal policy of SARSA. Comparing with the plot from the lecture slides, the intersection both occurred at a much earlier episode and with the resulting gap between the two curves much larger. The different behavior in my case could simply be attributable to how the algorithm sampled the grid space in this instance, or due in part to my using <u>c = 500</u> as well as the matrices for this project being 25x25. (See Appendix for c=100 instance.) And, of course, there arises the question of whether I calculated the Average Cumulative Reward correctly.

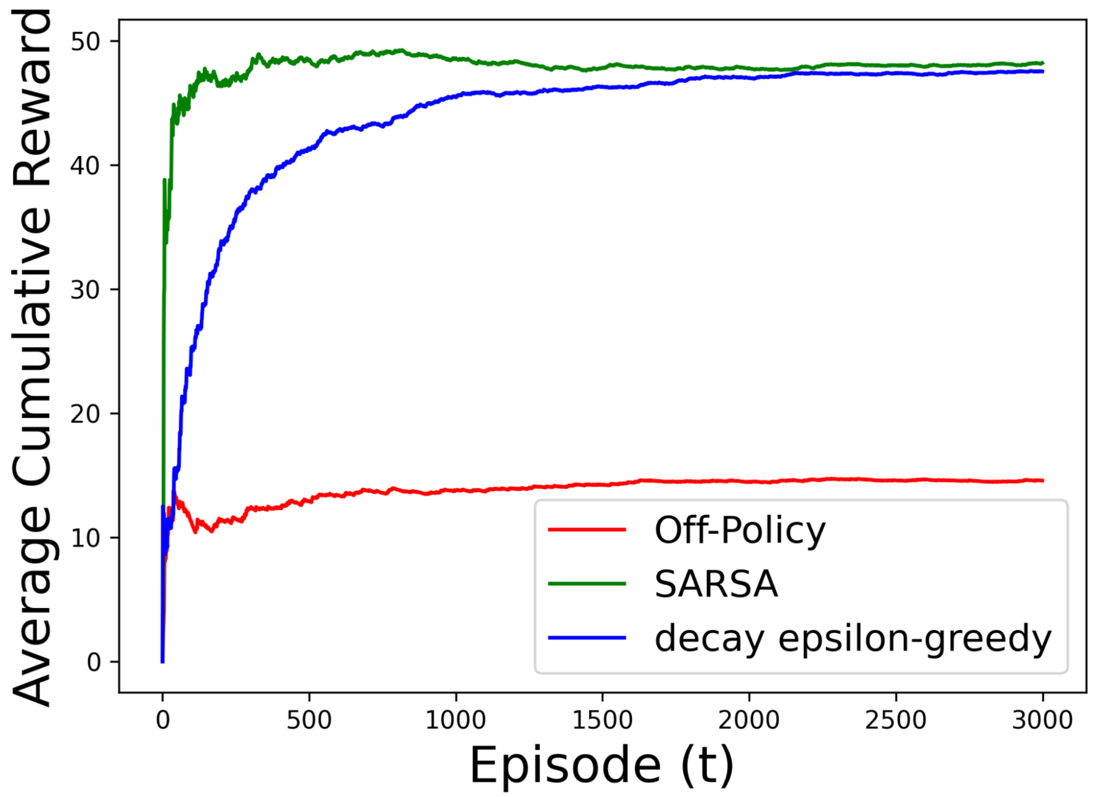


APPENDIX

1. Demonstration of relatively uniformity of random integer selections for 250 instances.



2. c=100 instance:


Average Cumulative Reward
Episode (t)
Off-Policy
SARSA
decay epsilon-greedy