**3 – LSPI**

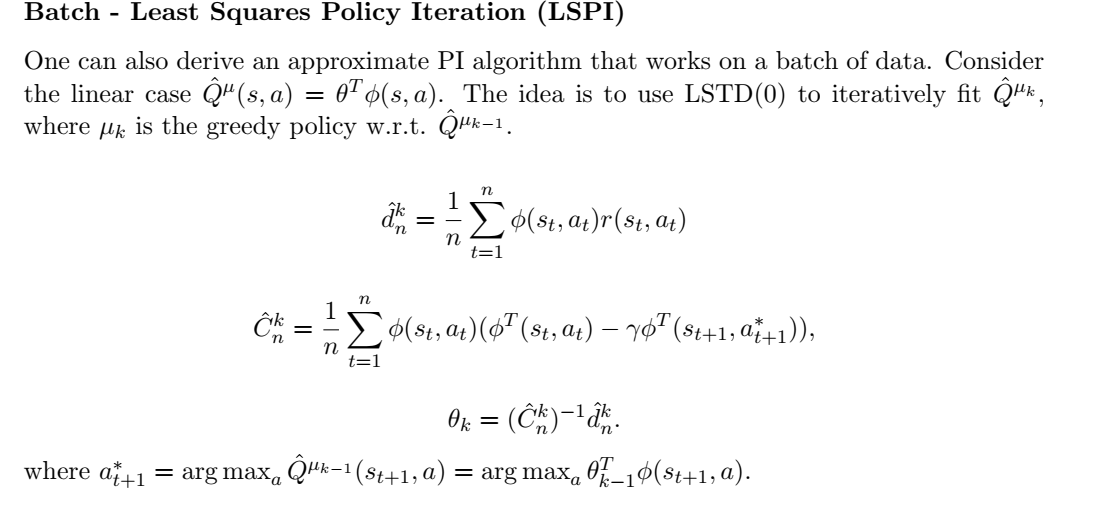
1. The data collected is with a random starting point reset for every sample

Thus, we don’t need to stop when we reach a state that is already in terminal state. However, it’s preferred to not include the future Q value estimation when reaching a done state. Meaning, not taking into account the when estimating (by LLN) the C matrix as this is a terminal state and there is no need for future rewards expectancy estimation for it (this might be another problem but not as described in our assignment).

When removing this component from C estimation for terminal states, we reach 1.0 success rate.  
In our code the C estimation update is as follows:  
C += np.dot(phi, (phi.T - (**(1 - done\_flags[i])** \* gamma \* phi\_next.T)))

1. From program prints on mean and variance:  
   data mean [-3.00931294e-01 7.02421111e-05]  
   so the mean state is at -0.3 position and 7e-05 speed.  
   data std [0.52 0.04]  
   These measures fit the expectation from a uniform sampling over the states space.
2. The size of the weights vector is as the size of the features vector . In our case the size of the features vector (as we estimate Q) is   
   In our case, when we used 12x10 features per state, we get   
   If there is a bias coefficient added to the features vector per state, we have:
3. Will be implemented before and in the LSPI iterations loop to evaluate the success rate and build the average performance per iteration plot.  
   plot:
4. See implementation in code.

In class we saw we need to do the following per iteration:  
**Critic**:

* 1. convert the state-action space to features space.
  2. By using the current and features vector for the next state in each tuple of (state, action, reward, next state) estimate for every action and apply the greedy policy over it to find the next best action (current optimal estimated Q according to current policy).
  3. Accumulate according to the below formulas the vector and matrix (From the lecture notes):  
     
  4. Apply inverse on matrix A and multiple with vector to get the new estimation of coefficients (new estimate)

**Actor**:

* 1. Apply greedy bellman operator on the new estimation and get a new greedy (or maybe close to greedy) policy.

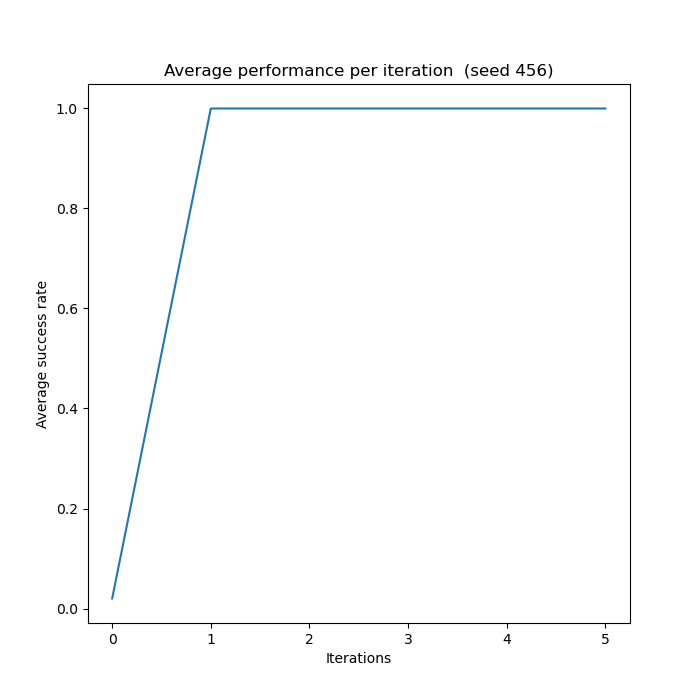
Plots of the average success rate over 50 random start states (w/ velocity 0) as described in section 4 of the question. The average success rate is evaluated vs the lspi iterations (until convergence). We tested several random seeds:  
A screenshot of a social media post

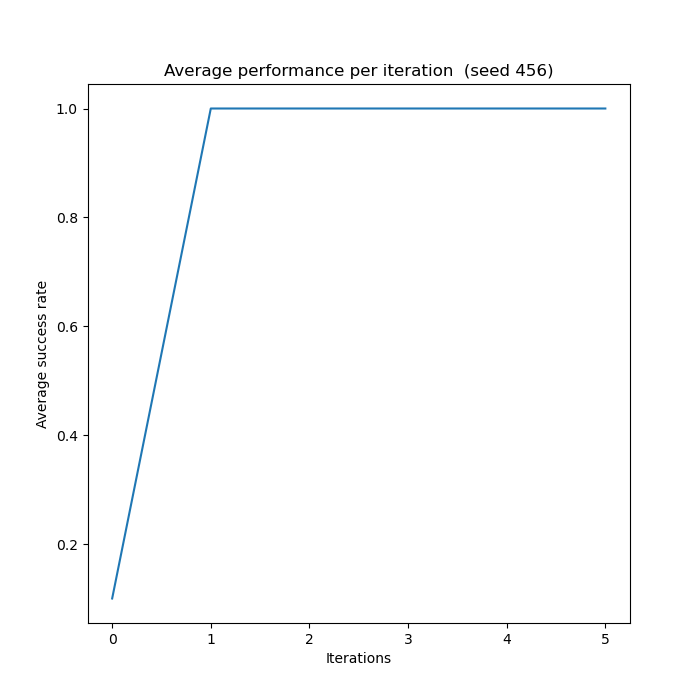
Description automatically generated  
A screenshot of a social media post

Description automatically generated  
A screenshot of a cell phone

Description automatically generated

1. We collected the final greedy policy over final Q and measured the average success rate vs the amount of data samples collected for training. The plots per seed 456:  
   For 10000 samples it seems to converge (less than 20 iterations) but the success rate is very low. We have a bias from the optimal policy and lack of data to train over. Thus, the success rate is very low.  
   A screenshot of text

   Description automatically generated  
   For 100000 samples we get:  
   

For 150000 samples we get:  
  
Since we have more data to train on, we have less overfit per iteration (as the variance error is in contrast to the amount of samples as seen in ML).

We’ve collected the final greedy policy over final Q and measured the average success rate vs the amount of data samples collected for training. We did that by adding an external loop over the lspi code that iterates over samples amount and collects the average success rate on the last policy found. Plot:  
A close up of text on a white background

Description automatically generated