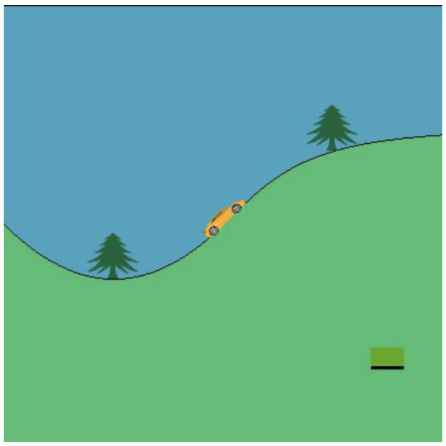
LIÈGE université
School of Engineering

INFO8003-1: Optimal Decision Making for Complex Problems



Assignment 2: Reinforcement Learning in a Continuous Domain

Staff :
ERNST Damien, *Teacher*
LOUETTE Arthur, *Teaching Assistant*
MIFTARI Bardhyl, *Teaching Assistant*

Group :
LAMBERMONT Romain
LOUIS Arthur

April 14, 2024

Contents

5 Parametric Q-Learning

5.1 Implementation

As we saw in previous sections, we successfully derived good policies from the fitted-Q learning algorithm. We can also try to use parametric Q-Learning to try to estimate Q and derive those policies. In this part, we will use a neural network as the estimator for the parametric Q-Learning iterative algorithm. The algorithm works as follows:

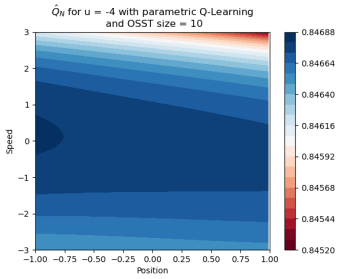
1. Initialize $\tilde{Q}(s, a, \theta)$ everywhere, we initialize α to 0.01
2. By observing the one-step system transitions (s_t, a_t, r_t, s_{t+1}), we update:
   $\delta(s_t, a_t) = r_t + \gamma \max_{a \in \mathcal{A}} \tilde{Q}(s_{t+1}, a, \theta) - \tilde{Q}(s_t, a_t, \theta)$
   $\theta \leftarrow \theta + \alpha \delta(s_t, a_t) \frac{\partial \tilde{Q}(s_t, a_t, \theta)}{\partial \theta}$
3. Jump to step 2 while the stopping criterion is not met

To allow us to compare fitted-Q and parametric Q-Learning in a correct way, we must use the same neural network architecture for both the algorithms. We used the same kind of architecture as for the previous section, with an input layer of width 3, the hidden layers following the shape $\{16, 32, 64, 32, 16, 8\}$, an output layer of width 1 and the activation function still being tanh. To train the network with the fitted-Q algorithm, we used the full generation technique (uniform start between -1 and 1 for the starting point of the OSST) and the fixed 189 iterations.

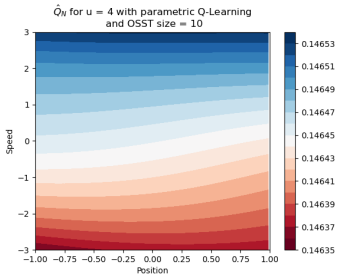
The algorithms works in a similar way compared to fitted-Q, but in this case we cannot iterate multiple times over the OSST set. We will once again use the Monte Carlo principle to test the derived policies multiple times with a random initial starting point.

5.2 Results

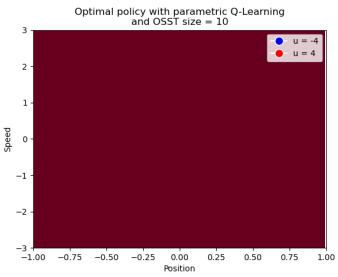
In this section, we showcase the value functions and derived policies for the different sizes of OSST sets in the next figures, sorted by the length of the OSST sets.



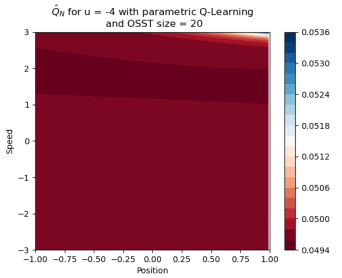
(a) Value function for action $u = -4$ with the OSST size 10



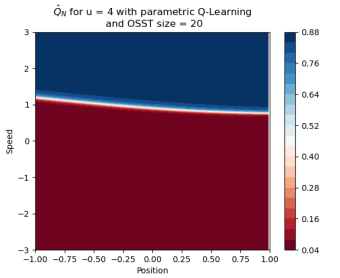
(b) Value function for action $u = 4$ with the OSST size 10



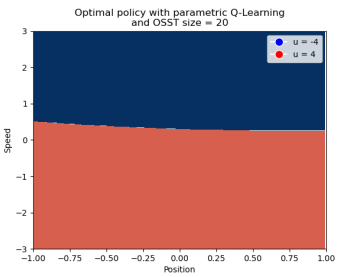
(c) Policy derived from PQL with OSST size 10



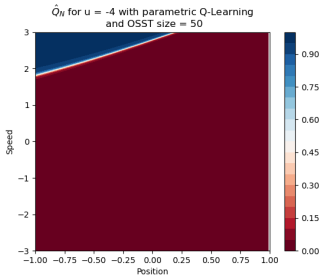
(a) Value function for action $u = -4$ with the OSST size 20



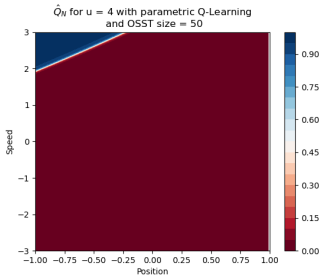
(b) Value function for action $u = 4$ with the OSST size 20



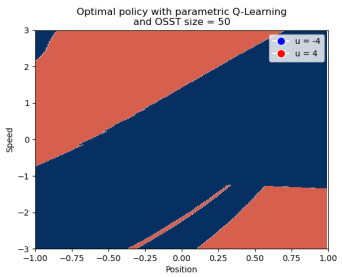
(c) Policy derived from PQL with OSST size 20



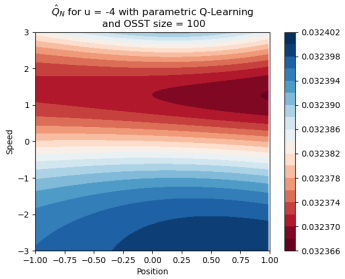
(a) Value function for action $u = -4$ with the OSST size 50



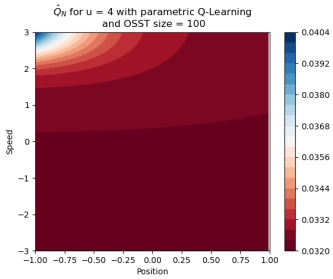
(b) Value function for action $u = 4$ with the OSST size 50



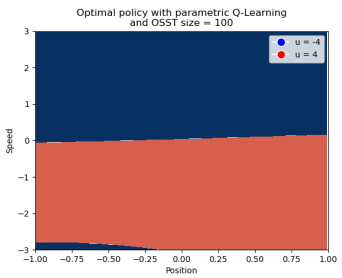
(c) Policy derived from PQL with OSST size 50



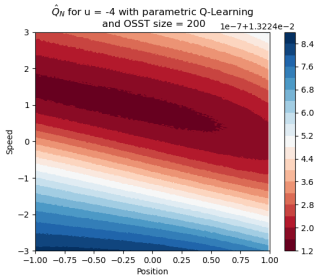
(a) Value function for action $u = -4$ with the OSST size 100



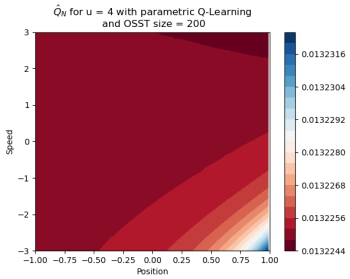
(b) Value function for action $u = 4$ with the OSST size 100



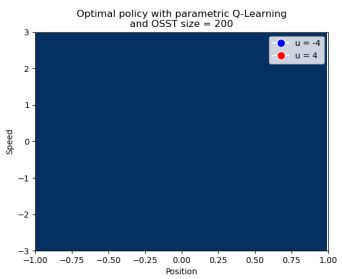
(c) Policy derived from PQL with OSST size 100



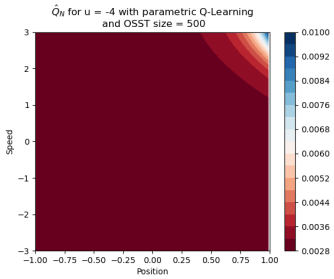
(a) Value function for action $u = -4$ with the OSST size 200



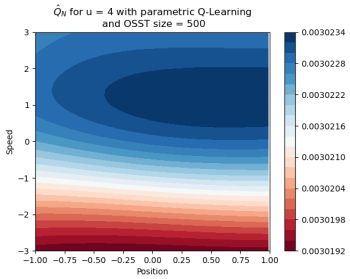
(b) Value function for action $u = 4$ with the OSST size 200



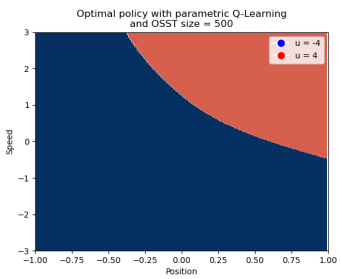
(c) Policy derived from PQL with OSST size 200



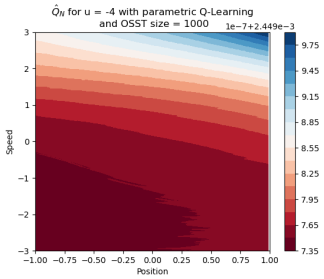
(a) Value function for action $u = -4$ with the OSST size 500



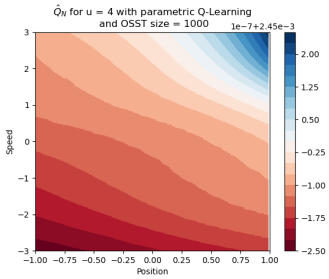
(b) Value function for action $u = 4$ with the OSST size 500



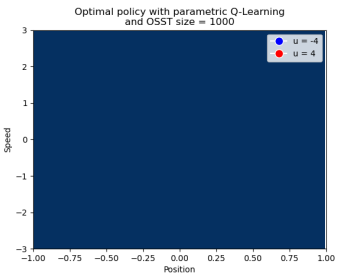
(c) Policy derived from PQL with OSST size 500



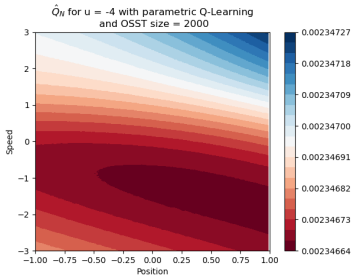
(a) Value function for action $u = -4$ with the OSST size 1000



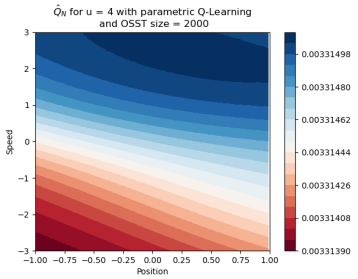
(b) Value function for action $u = 4$ with the OSST size 1000



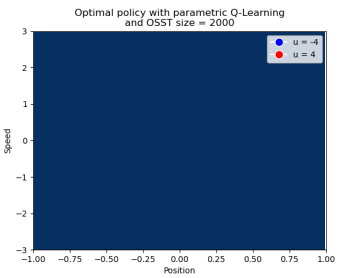
(c) Policy derived from PQL with OSST size 1000



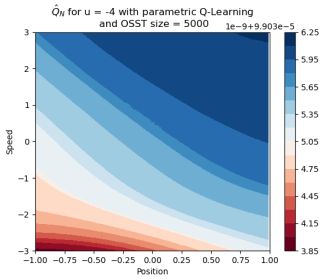
(a) Value function for action $u = -4$ with the OSST size 2000



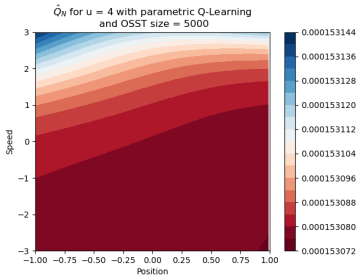
(b) Value function for action $u = 4$ with the OSST size 2000



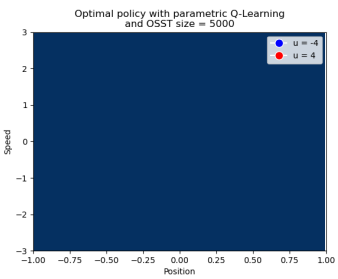
(c) Policy derived from PQL with OSST size 2000



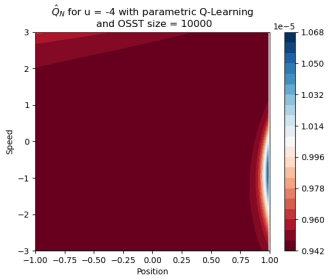
(a) Value function for action $u = -4$ with the OSST size 5000



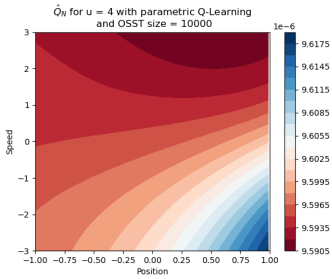
(b) Value function for action $u = 4$ with the OSST size 5000



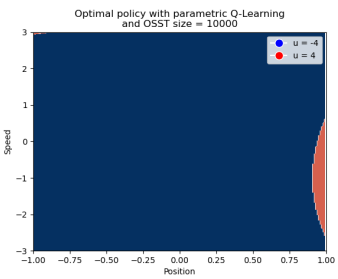
(c) Policy derived from PQL with OSST size 5000



(a) Value function for action $u = -4$ with the OSST size 10000



(b) Value function for action $u = 4$ with the OSST size 10000



(c) Policy derived from PQL with OSST size 10000

In the following figure, we compared the results obtained by FQI and PQL with the respective OSST set sizes. In that figure we can clearly see that the FQI performs much better than the PQL in that case. The

behaviour of PQL not being able to find good policies was expected from theory in this problem. Maybe with loads of data the PQL could start to perform a little bit better but with our hardware and not wanting to spend entire days to train our network, we were not allowed to deliver good results for this implementation.

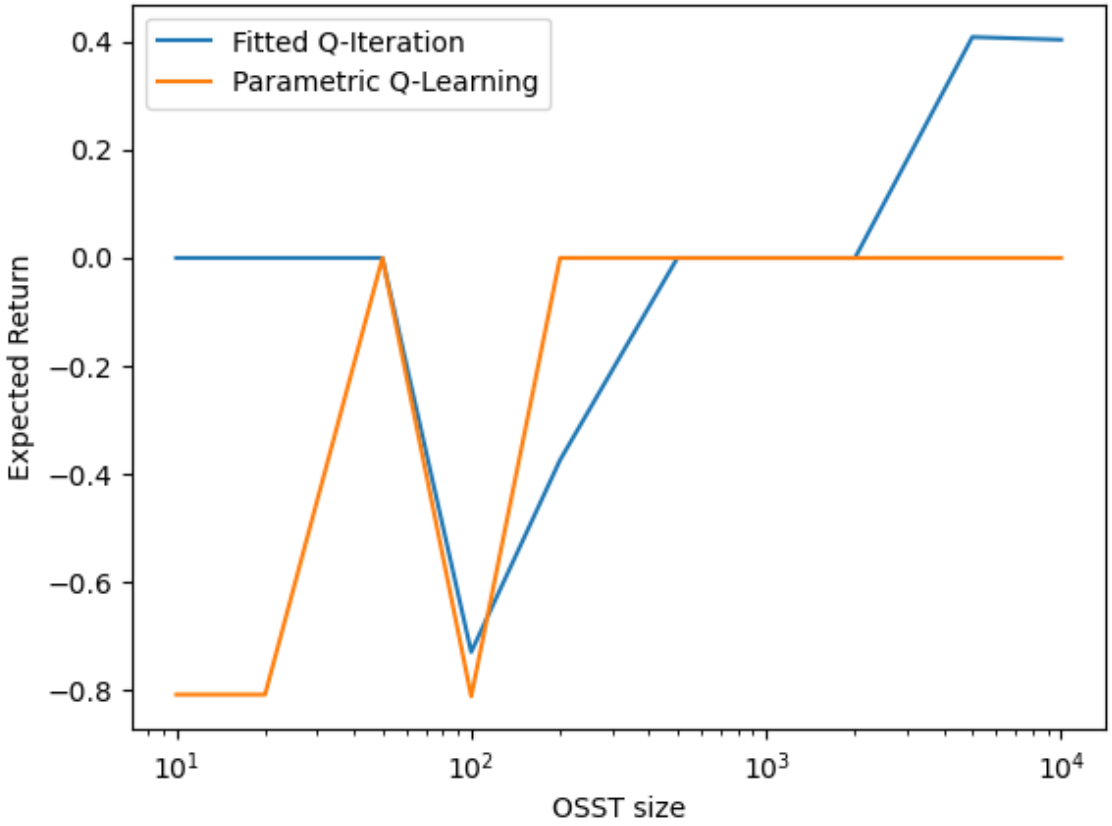


Figure 11: Evolution of the return values of both techniques with respect to the sizes of the OSST sets

We also generated GIF files as in previous sections but they are not really interesting as none of the policies reach a good terminal state. There are still available in the submission file.