Master MVA
Reinforcement Learning

TP 3: Reinforcement Learning With Function Approximation

Victor Busa
victor.busa@ens-paris-saclay.fr

January 5, 2018

1 On-Policy Reinforcement Learning with Parametric Policy

1.1 Question 1

See Python code for the implementation of the *REINFORCE* algorithm for the linear-quadratic Gaussian regulation (LQG) problem. For this problem $\theta^* = -0.59$. I used the following parameters to run the experiments:

- $N = 100$: number of episodes to collect
- $T = 100$: each trajectory have at most T time steps
- $n\_itr = 100$: number of policy parameters update
- $\gamma = 0.9$: discount factor
- $n\_runs = 5$: number of times we run the algorithm to get an average

1.1.1 Constant Step

For the constant step, I've tuned the learning rate α parameter. Figures 1.1 and 1.2 show the value of the average reward R and the $\widehat{\theta}$ parameter for different values of α:

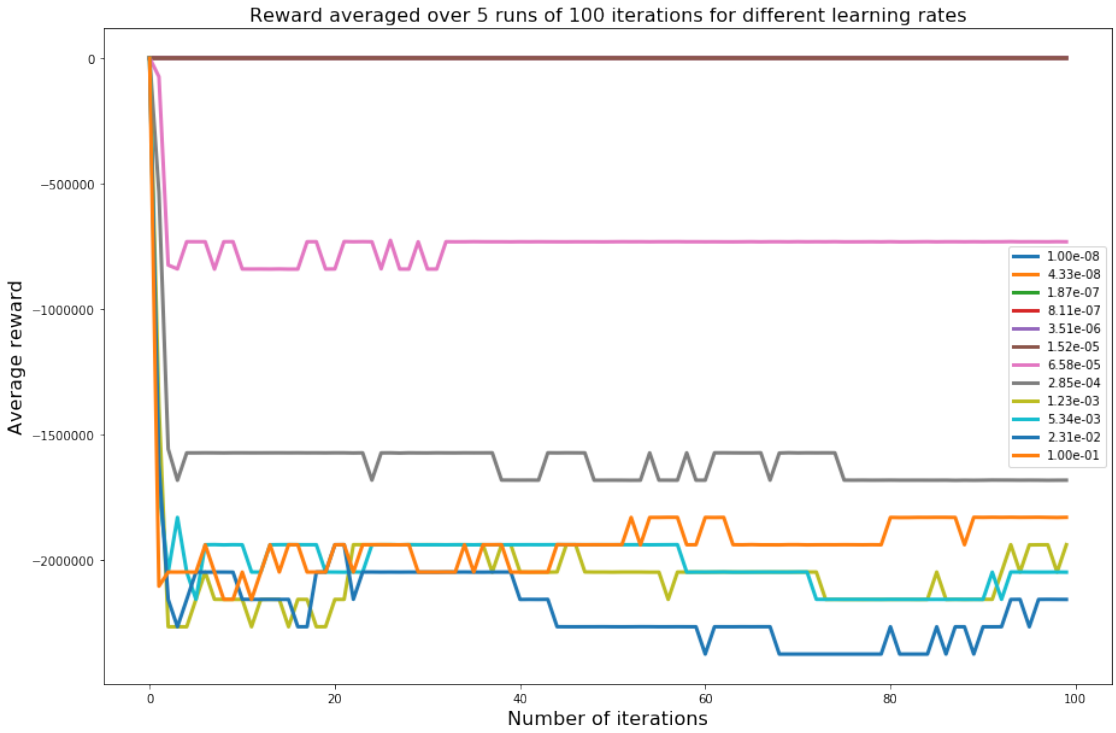

Figure 1.1: average reward over 5 runs of 100 iterations each for different values of the learning rate α

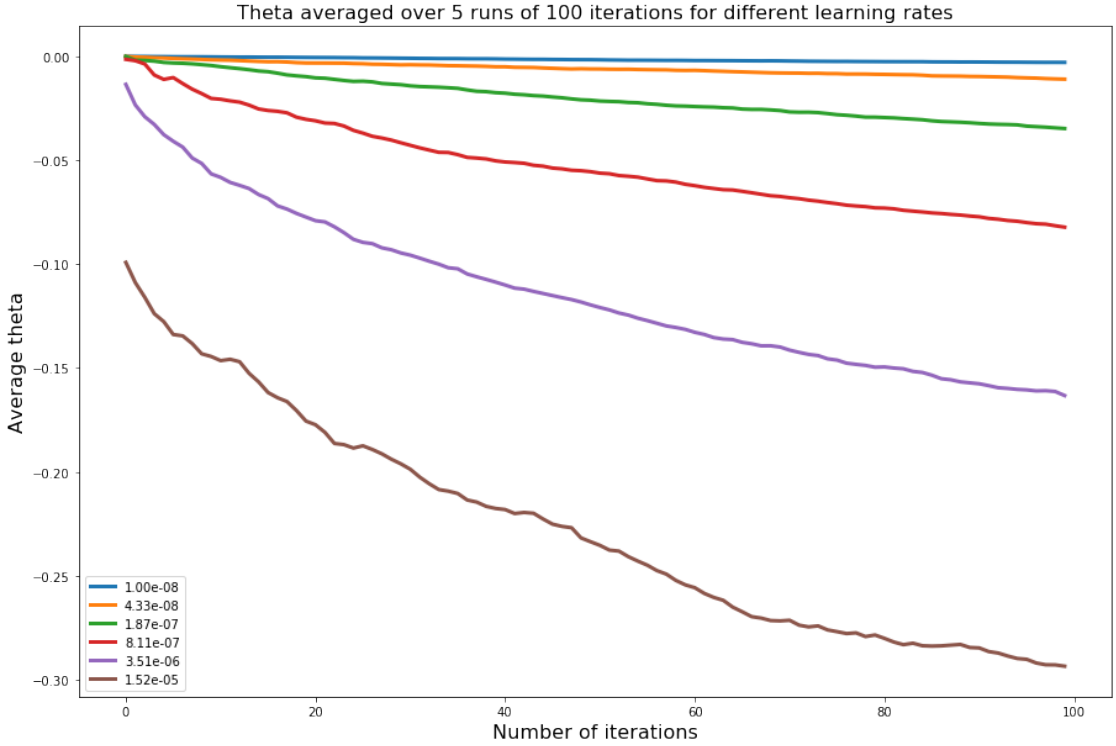

Figure 1.2: average parameter $\widehat{\theta}$ over 5 runs of 100 iterations each for different values of the learning rate α

According to the Figures we can see that the best value of α is α = 1.52e-5 as $\widehat{\theta}$ converges faster towards $\theta^* = -0.59$. We will use this value of α and we will tune the learning decay in the next section.

1.1.2 Annealing Step

Using the best value of α (α = 1.52e-5) found for the constant step, we will tune the learning decay. Figures 1.3 and 1.4 show the value of the average reward R and the $\widehat{\theta}$ parameter for different values of the learning decays:

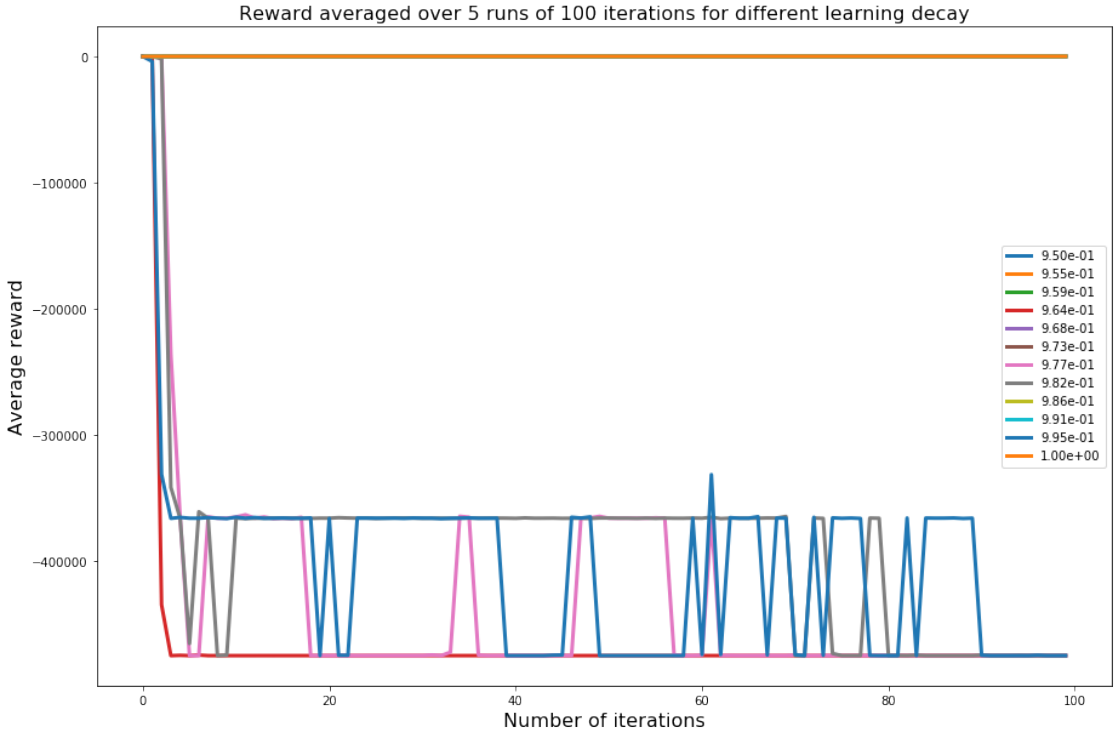

Figure 1.3: average reward over 5 runs of 100 iterations each for different values of the learning decays

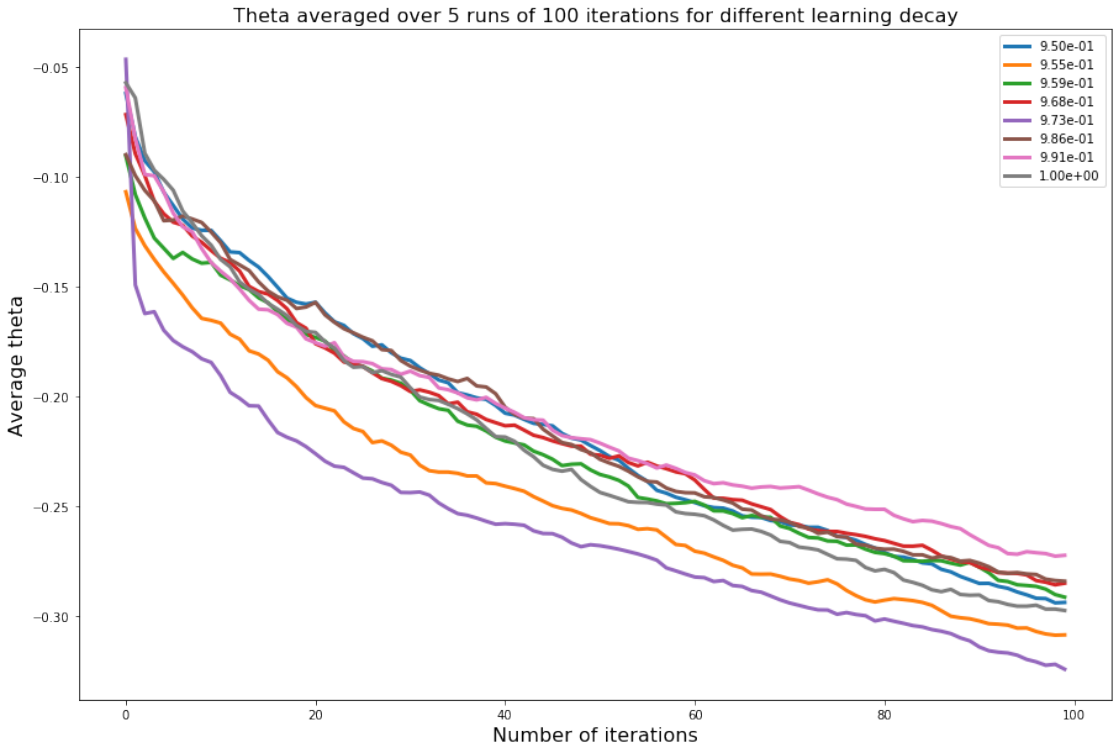

Figure 1.4: average parameter $\widehat{\theta}$ over 5 runs of 100 iterations each for different values of the learning decays

According to Figures 1.3 and 1.4, the best learning decay for this experiment is 0.973.
Note: In practice it is very difficult to tune both the learning rate and the learning decay and here I chose to tune the learning decay only for a fixed value of the learning rate. What we should instead do is: use a high learning rate α at the beginning of the experiment and a high learning decay. Then decrease the learning rate by the learning decay and decrease the learning decay at each iterations. What should be even better is not to decrease the learning decay at each iteration but instead decrease it when the norm of two successive update of the parameters θ is below a certain threshold. I don't have the time to run these experiments as the algorithm takes some times to terminate.

1.1.3 Adam Step

In this experiment I used the adam optimizer and I've tuned the learning rate $\alpha$for a fixed value of $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Here I've chosen to test the REINFORCE algorithm for 12 values of α logarithmically spaced in $[10^{-3}, 10^{-1}]$. Figures 1.5 and 1.6 show the value of the average reward R and the $\widehat{\theta}$ parameter for different values of the learning rates.

Note: I've deleted inconsistent values of the learning rates α before plotting the graphs, that is why there is no 12 different values of α.

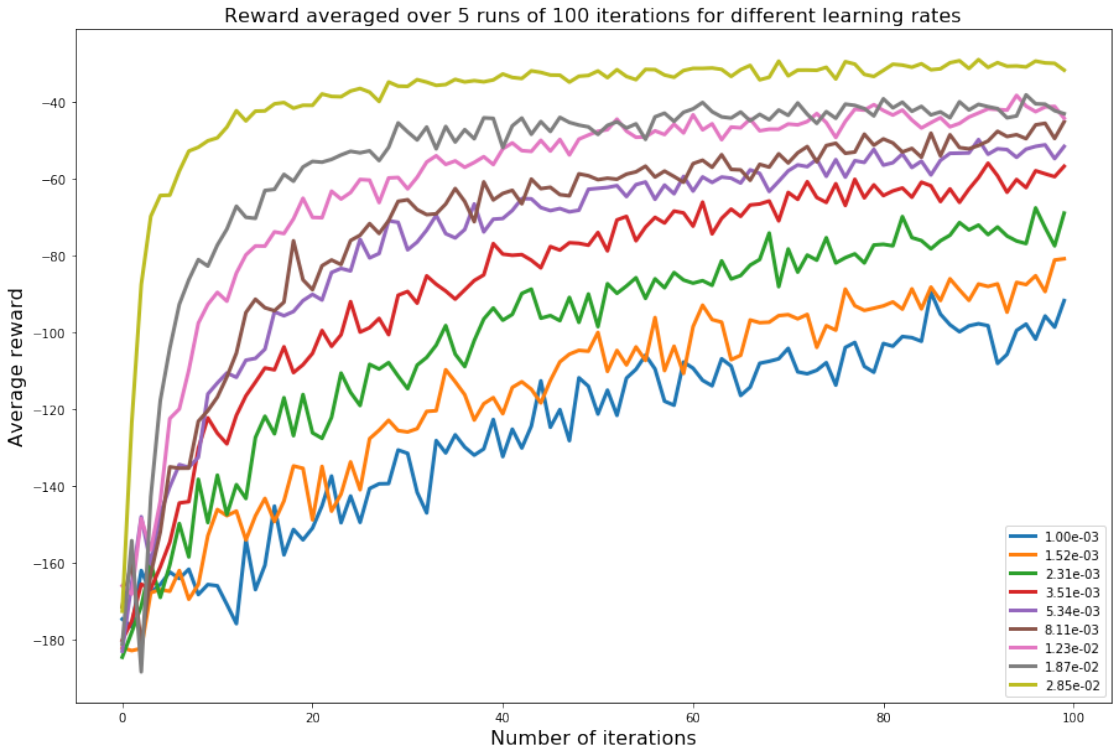

Figure 1.5: average reward over 5 runs of 100 iterations each for different values of the learning rates

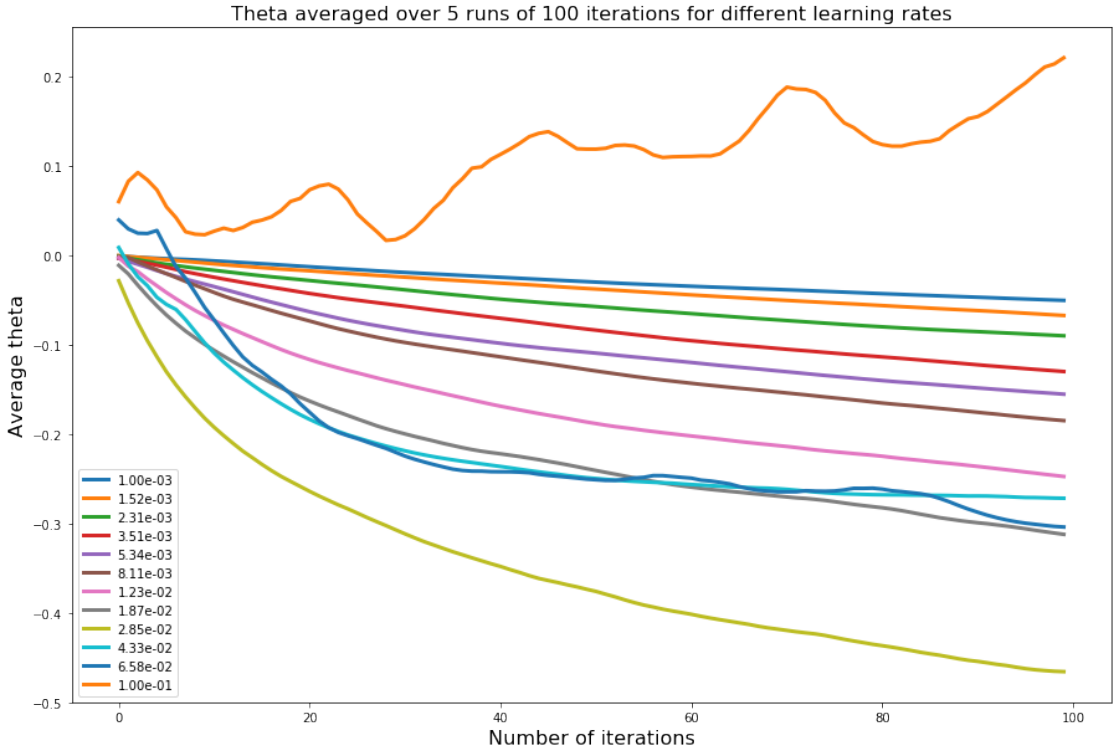

Figure 1.6: average parameter $\widehat{\theta}$ over 5 runs of 100 iterations each for different values of the learning rates

According to Figures 1.5 1.6 the best value for α is α = 2.85e-2.

1.2 Conclusion

α_t is the learning rate parameter. A high value of α_t will make a big step in the direction of the gradient of the function we computed and hence. Yet, if the step is too big, in the next iteration it might result in a value of the function we tried to minimize which is higher then at the previous iteration. On the contrary, if the value of α_t is too low, the value will likely decrease at each iteration but it won't decrease a lot and the algorithm will take lots of time to converge. Hence it is very important to tune this parameter. Adam optimizer is a gradient descent method that takes into account the moment or velocity of previous iteration in order to accelerate the convergence of the next iteration.
N (the number of trajectories generated) matters in the sense that the more trajectories we collect the more accurately the algorithm will converge at each iteration. Indeed $N = 1$ is synonym of a stochastic gradient descent and thus the algorithm run faster but will converge more slowly (and the convergence curve will have a serrated shape).

1.3 Question 2: Trying different values of N: number of trajectories

As discussed shortly in the Conclusion part, the lowest the value of N is, the fastest[1] the algorithm will run. Yet, the algorithm will converge more slowly as the direction of the gradient for few trajectories isn't representative enough of the true direction to consider to minimize the gradient. The higher the value of N is the more accurately the algorithm will converge to the true direction that minimize the gradient but it will also takes more time at each iteration due to the high number of trajectories to consider at each iteration. Figures 1.7 and 1.8 show the value of the average return and the average theta for 5 distinct runs and different values N.

[1] in term of computational time

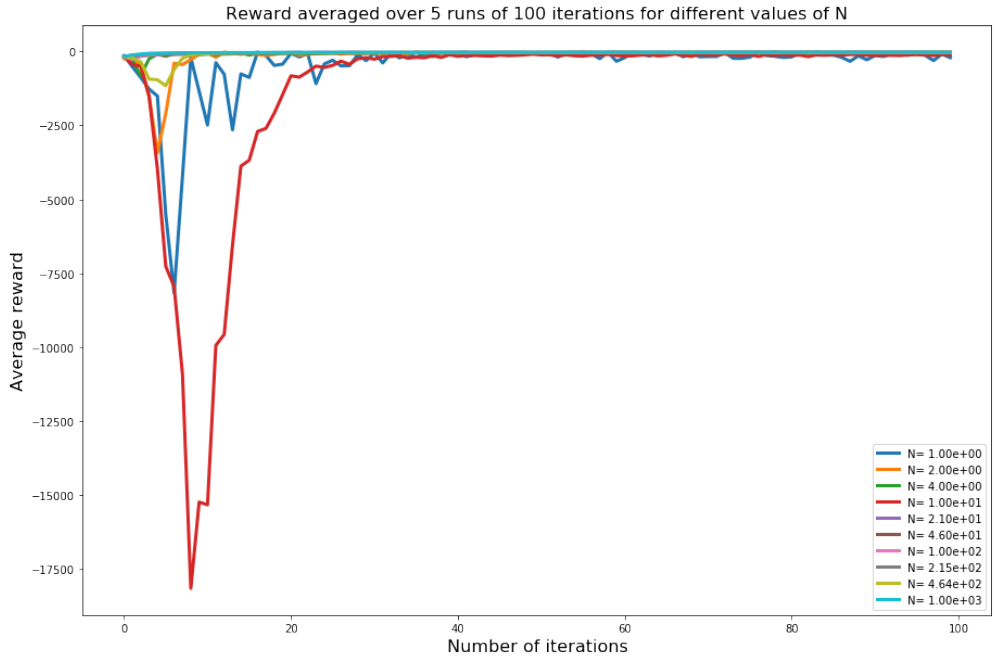

Figure 1.7: average return $J(\pi_k)$ in function of the number of iterations k obtained by the fitted-Q iteration algorithm applied to the LQG problem for different values of N (the number of trajectories considered)

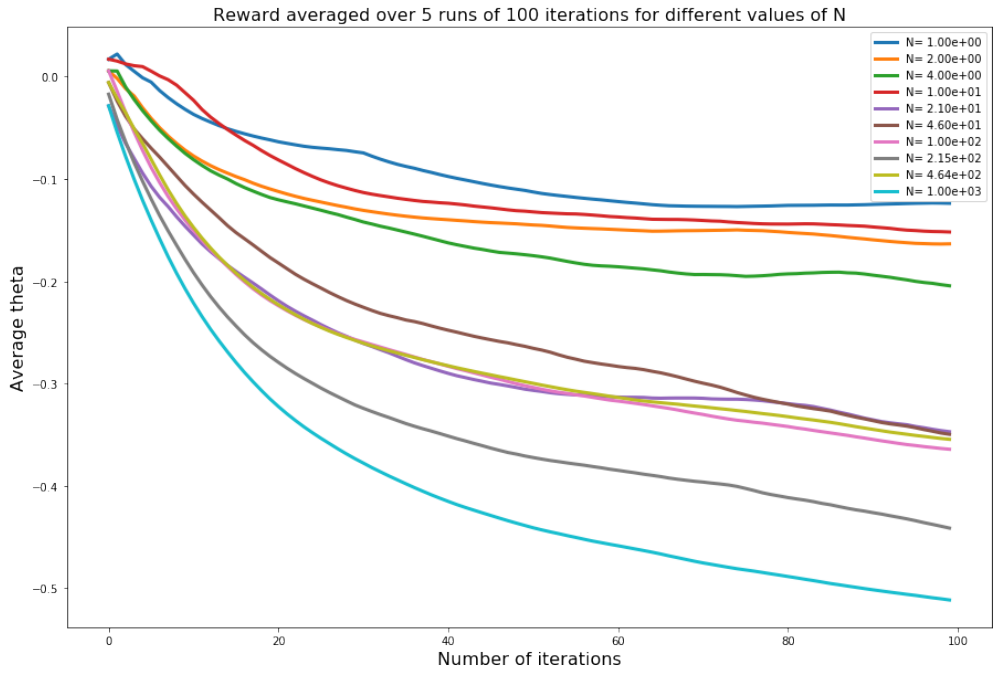

Figure 1.8: average theta $\widehat{\theta}$ in function of the number of iterations k obtained by the fitted-Q iteration algorithm applied to the LQG problem for different values of N (the number of trajectories considered)

Naturally we can see that high value of N allow the algorithm to converge more quickly to the true

value θ^* will it also takes more time to compute (6 min on my compute[2]). On the other hand lower values of N converge slowly much takes less time to compute. Hence we have to choose N carefully to balance between computational time and convergence velocity. $N \approx 200$ seems to be a good pragmatical choice.

2 On-Policy Reinforcement Learning with Parametric Policy

2.1 Implementation

See Python code for the implementation of the algorithm. I chose to stop the algorithm either went it attains a number of maximum iterations (set by default to 100) or when $\ell_2(\theta_k - \theta_{k-1}) < \epsilon$ with $\epsilon = 10^{-12}$ by default.

2.2 Experimentation

Using a discount parameter $\gamma = 0.9$ and a value of λ (the regularization parameter) of 2, the linear fitted Q iteration algorithm performs well and converges in one iteration. The Figure 2.1 shows the average reward obtained from running the fitted-Q iteration algorithm on the linear-quadratic Gaussian regulation (LQG) problem in function of the number of iterations k.

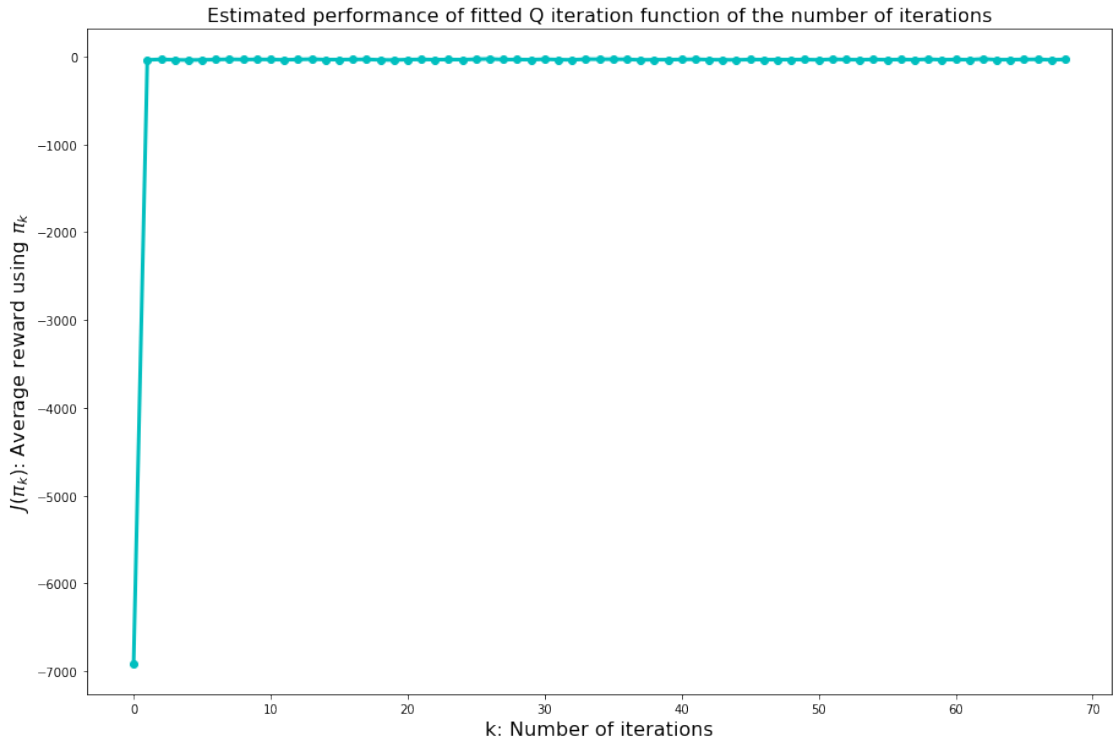

Figure 2.1: average return $J(\pi_k)$ in function of the number of iterations k obtained by the fitted-Q iteration algorithm applied to the LQG problem

Note: I've played with the parameters α and γ to see if it impacts the performance of the algorithm

[2]See code

for the LQG problem. Apparently it hasn't much influence on the performance. Graphs 2.2 and 2.3 that show the result of the fitted Q iteration in function of α or γ are displayed below.

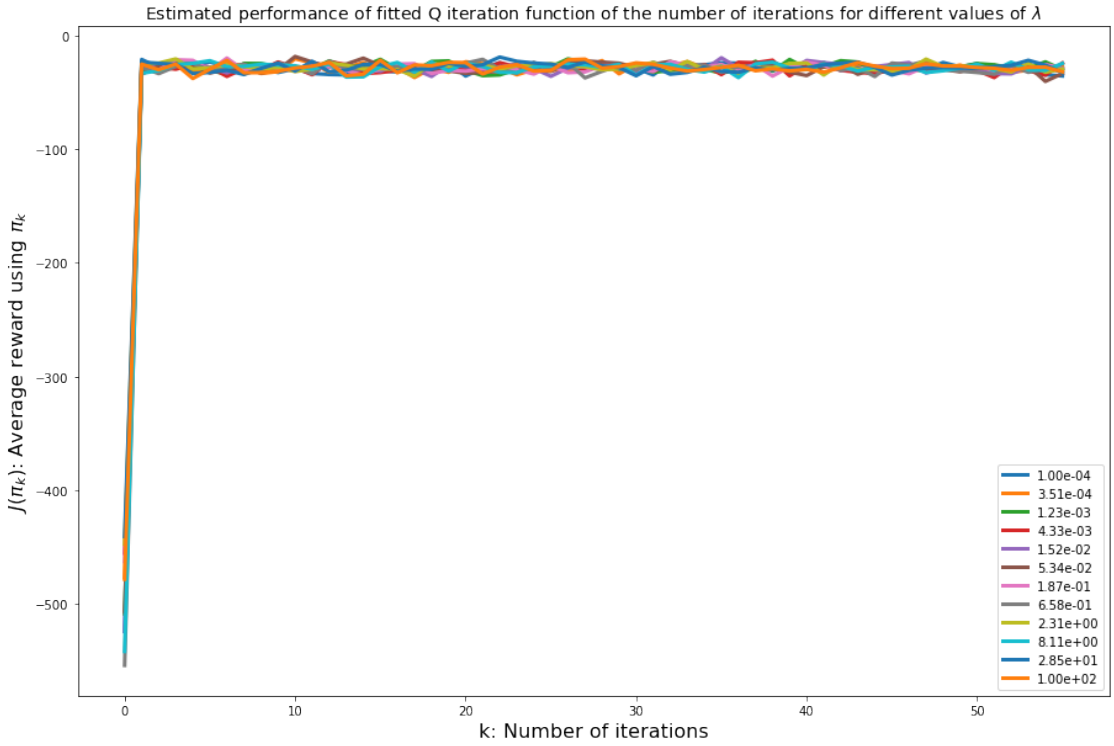

Figure 2.2: average return $J(\pi_k)$ in function of the number of iterations k obtained by the fitted-Q iteration algorithm applied to the LQG problem for different values of α

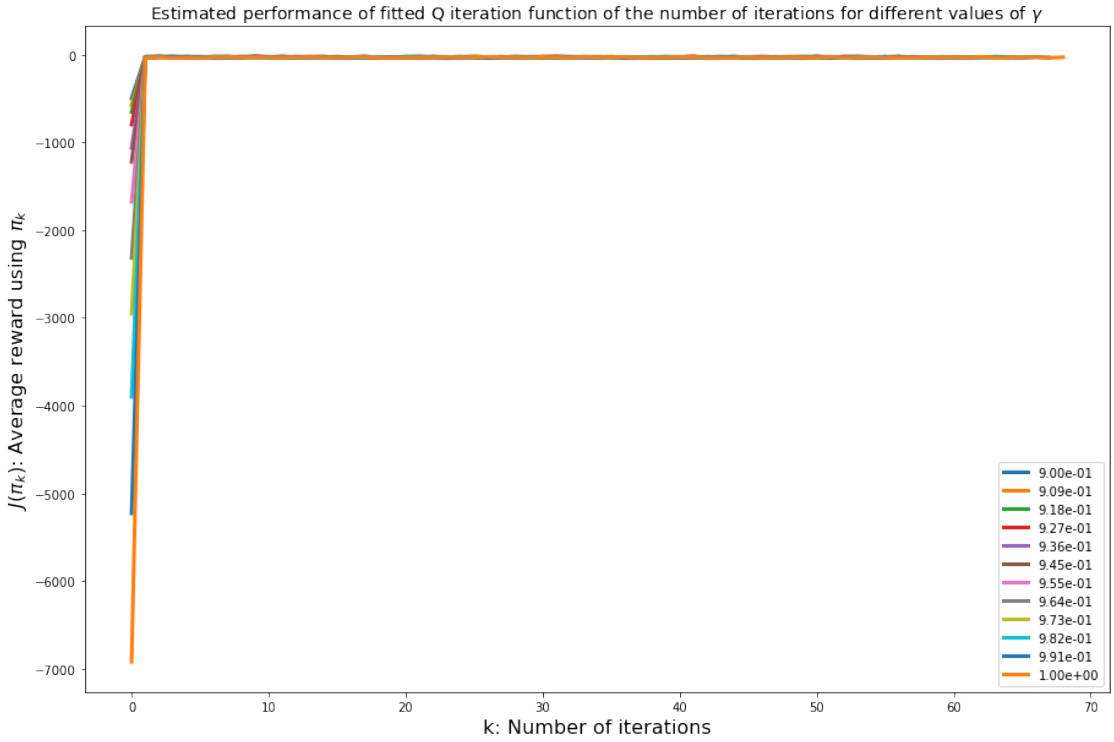

Figure 2.3: average return $J(\pi_k)$ in function of the number of iterations k obtained by the fitted-Q iteration algorithm applied to the LQG problem for different values of γ