Wet2 – 046203

Submitters:

David Valensi 342439643

Yaniv Galron 206765646

Question1

1.1) We will define the game as a Markov decision process. We will define each state as the sum of the cards of the player (X) and the value of the first dealer card (Y) (When it's the dealer turn there are no more actions to be taken so those states are not important. There are 10 states for the dealer card value, 17 Player sum values (that are lower to 21) and 1 terminal state. The number of states: $|s| = 17 * 10 + 1 = 171$
We don't care about the different terminal states because the player can't take any action in those states. Furthermore, we will model the reward function such that we receive a stochastic reward in each state based on the action taken. in the terminal state we don't receive a reward. This is fine because the value function at different terminal states is trivial. The action space (A) is to take a card (hit=1 or to not take a card (stick =0). The reward is stochastic, we will calculate the expectancy of it grammatically for the value iteration algorithm.

1.2) Like we mentioned in the previous subsection, we will not plot the terminal states, since the value function there is trivial (just the reward in each of those states).

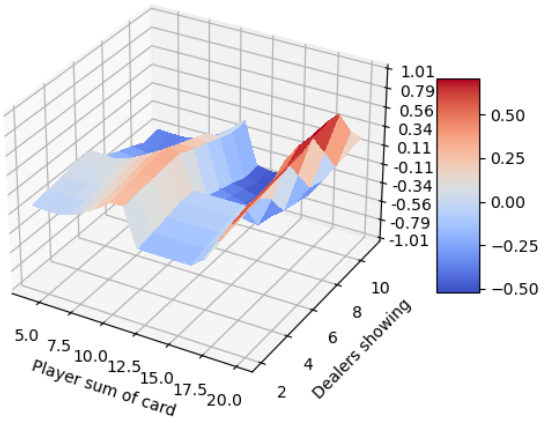


Figure: The value function for the different states of the game

We can see that the maximum value is achieved when the player has cards that sums up to 20. In those states (as we will see clearly in the next subsection), the player will stick and will have a very large change of winning. Another big value group of states are the ones where the player has cards that sums up to 10. In those states the player will hit and will have a large chance of getting a 10 (we have four cards with a value of 10) and then will stick.

1.3) We can see some interesting things from the following figure, such as:

- For a hand greater or equal to 17, we will always stick.
- For a hand lower than 12 we will always hit.
- For a Dealer showing of around 7, we prefer to hit more than other states. This is since the dealer has a higher chance of getting close to 21 in this case.

Question2

2.1) The states are the jobs left and the action is which job to take. Thus, the number of states is the number of subsets in the set of jobs, and the number of actions is the number of total jobs:

$$|S| = 2^N = 2^5 = 32, |A| = N = 5$$

We will order the states as binary code, where the state "00000" corresponds to the state which no job is left and "00011" corresponds to the state which only the fourth and the fifth jobs left.

2.2) CODE: Solved using iterative solution, rewards = -costs

2.3) The policy π_c and its value function

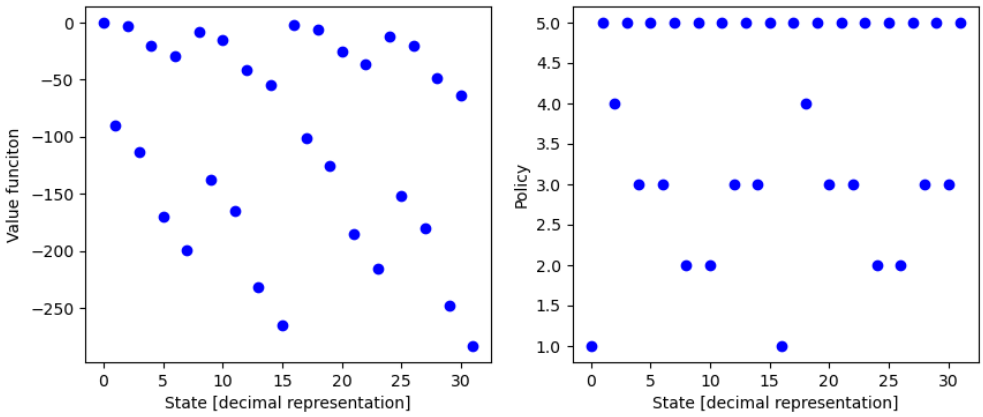


Figure: The “cost only” policy and its value function

We can see for example that the state with the lowest value function is s_0 – the state where all jobs are not finished.

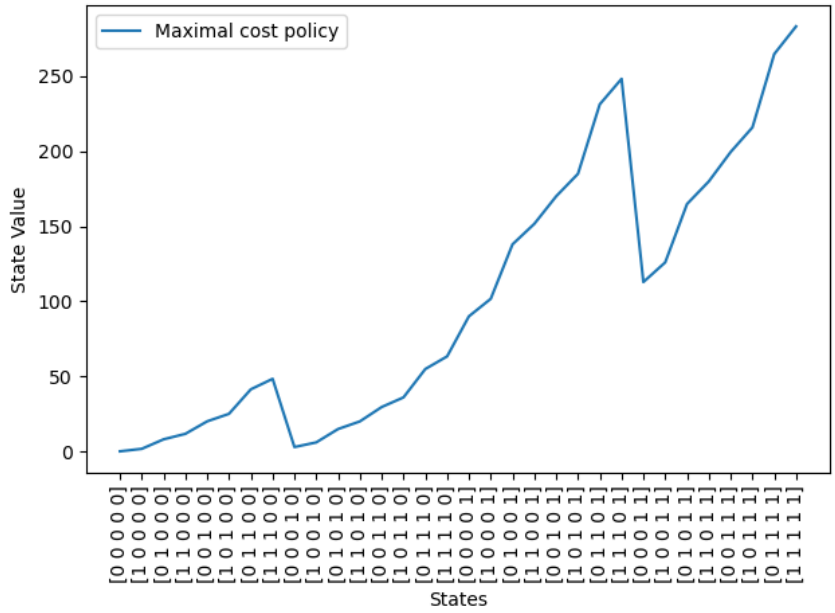
Question 2

a. Each state contains the set of unfinished jobs. Each job can be finished or unfinished. So, the number of states with N=5 is the size of the power set $P\{1,2,\dots,5\}$

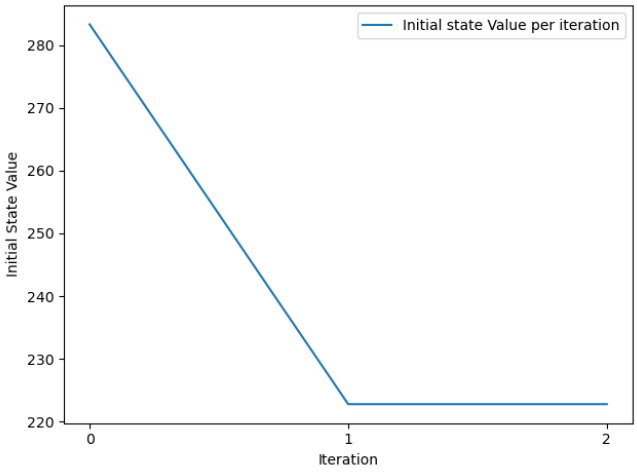
$$|S| = 2^N = 2^5 = 32, |A| = N = 5$$

b. Solved by VI in the code

c. Here are the state values for all state in S using the maximal cost policy.
Here, state [1 0 0 0 0] means that job 1 still needs to be done and all other jobs are finished.

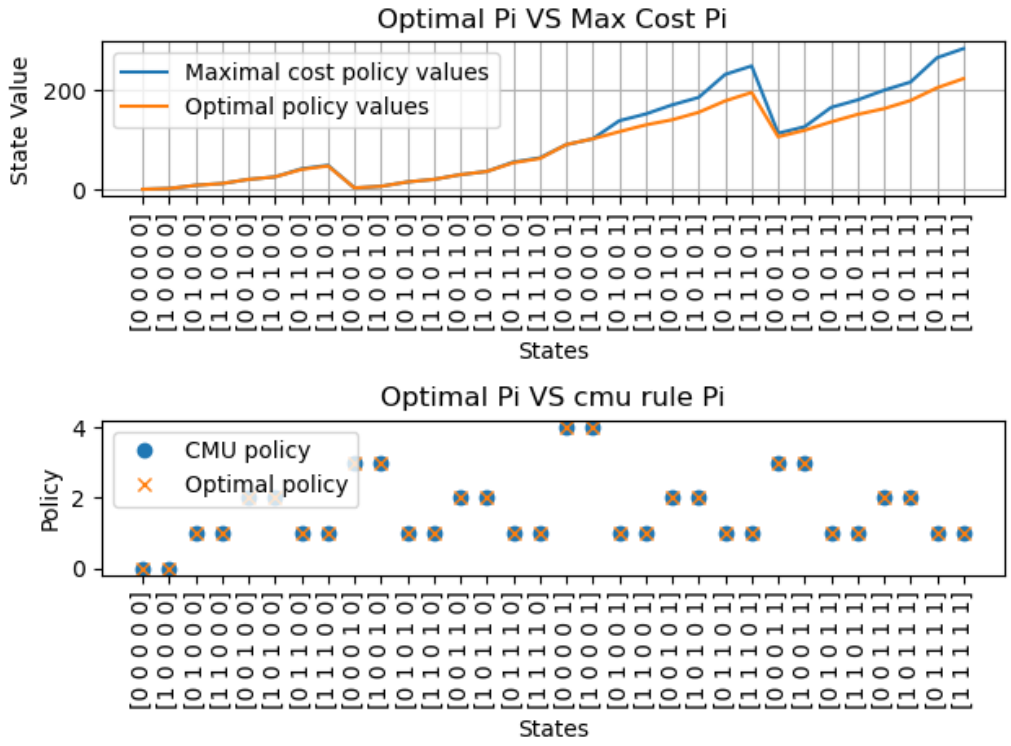


d. The plot for the initial state [1 1 1 1 1] value using PI starting from π_c:



As we can see, it takes one iteration to the PI algorithm to find the optimal policy π^*.

e. Let's compare between the optimal policy achieved by PI and the $c\mu\ rule$ policy $\pi^{c\mu}$. We observe that they are the same as expected since $\pi^{c\mu}$ is known to be the optimal policy for this problem.
Moreover, we plot $V^{\pi^*}\ vs\ V^{\pi_c}$:

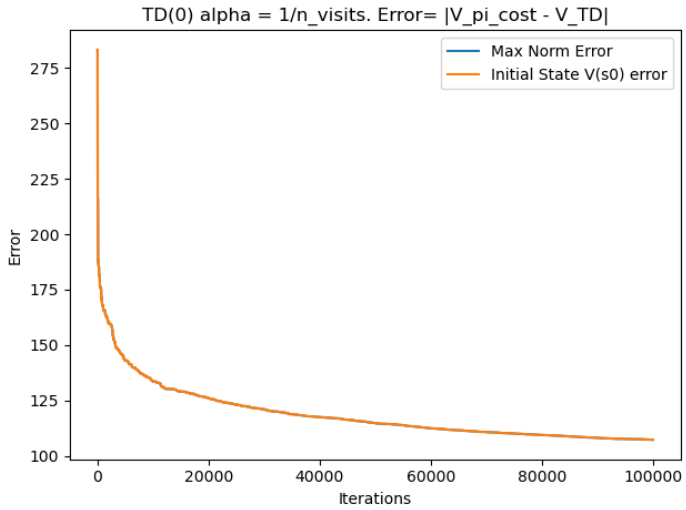


We observe as expected, that the CMU policy is equal to the optimal policy that we computer.

f. In the code

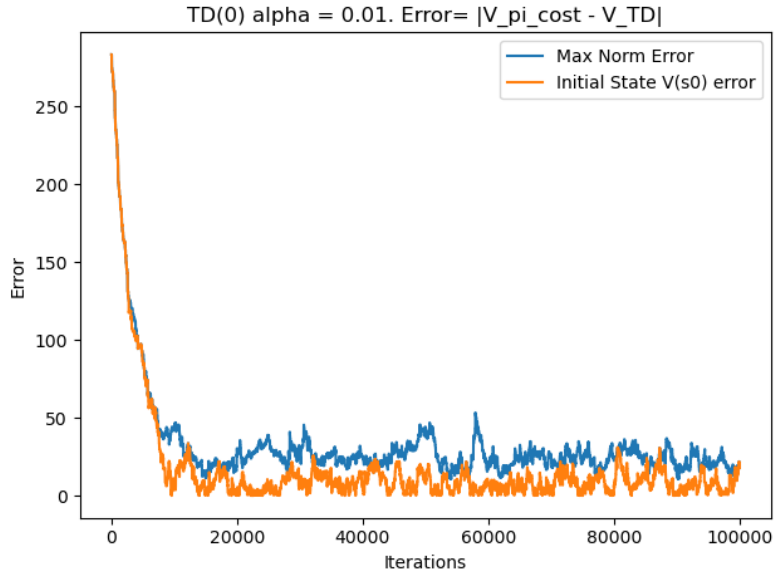
g. For each step size scheduler, we plot the max-norm of the error $V^{\pi_c} - \hat{V}_{TD}$ and the initial state error: $V^{\pi_c}(s_0) - \hat{V}_{TD}(s_0)$.

$\alpha_n = \frac{1}{\#num\ of\ visits\ to\ s_n}$:

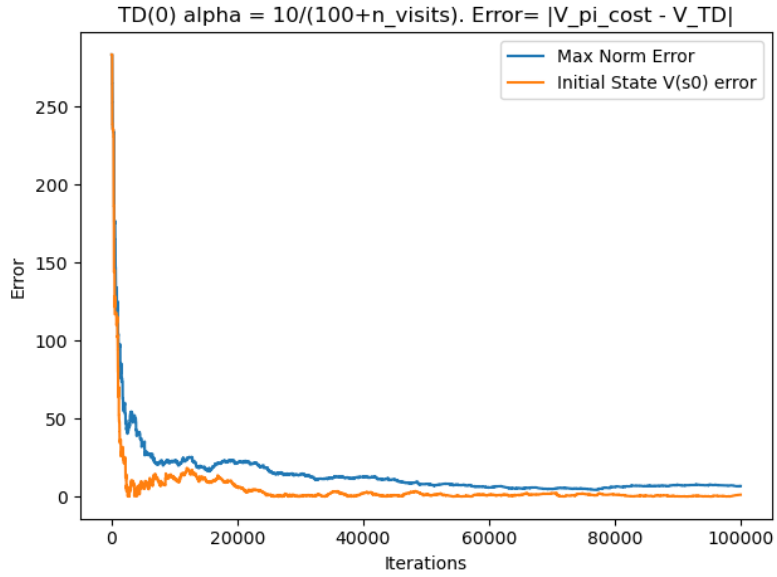


We observe that converge to 0 error is not achieved both in the Max-norm and in the Initial state values.
The idea behind this step-size is to hold the G1 requirements from the convergence theorem and thus ensures $\hat{V}_{TD}$ to converge to the real value function when $iteration \to \infty$. However, we observe that the error doesn't converge to 0 even after a huge number of iterations. The step size becomes too small at some point to enable convergence and a real change to the value function.

$\alpha_n = 0.01$: We observe that with this step size, convergence for $\hat{V}_{TD}(s_0)$ and a 0-error are achieved but is not stable around a 0-error. This is because the step size doesn't tend to 0 and always updates the value function even if it achieved the minimal error.



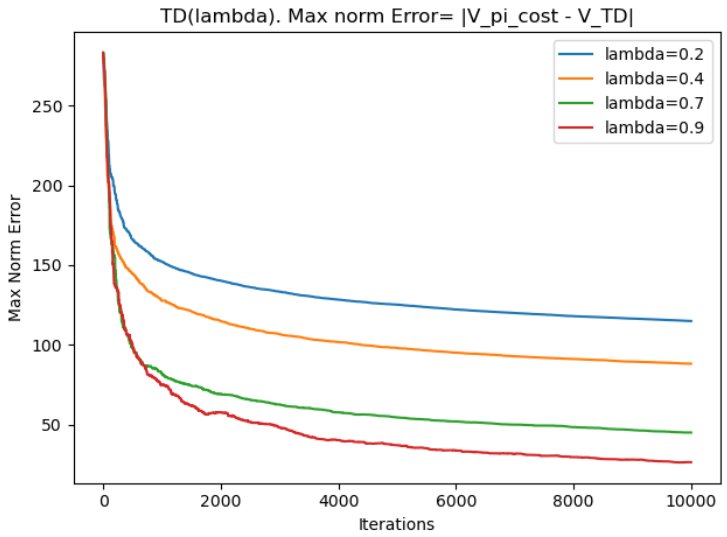
$\alpha_n = \frac{10}{100+\#num\ of\ visits\ to\ s_n}$:

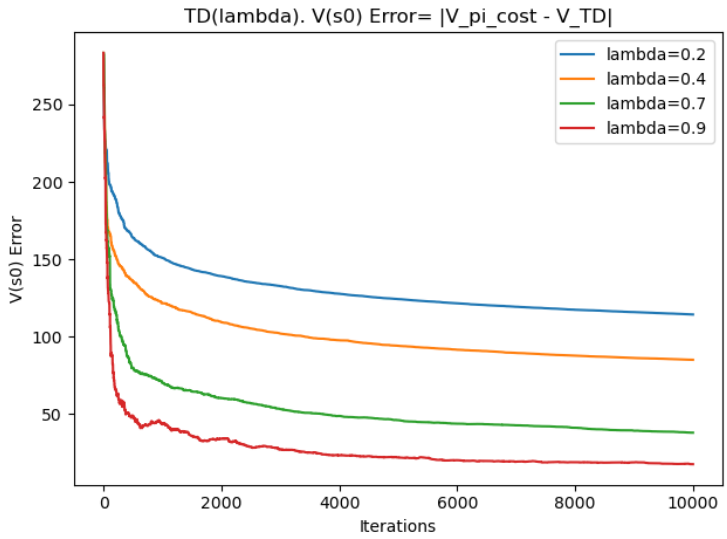


For this step-size scheduler, we observe a more stable convergence of the error. This is because the step size tends to 0 with new visits and thus, new experiences don't add noise to the computed state values. The stability occurs both in the Max-norm error and in the Initial-state error. The advantage of this step-size, is that it is higher than the $\alpha_n = \frac{1}{\#num\ of\ visits\ to\ s_n}$ from $\#num\ of\ visits\ to\ s_n \geq 11$ and thus, gives more weights to more experiences such that TD(0) has the time to converge to a better value function.

In all the step sizes, we observe that the Max-norm error doesn't converge to 0. This is because the fixed policy π_c may not visit all the states and for these states, the $\hat{V}_{TD}(s)$ values cannot converge since they're not visited (a sufficient number of times).

h. $TD(\lambda)$. We use several lambda values and plot the errors using step-size $\alpha_n = \frac{1}{\#num\ of\ visits\ to\ s_n}$. The plots show the average errors over 20 experiments/lambda.



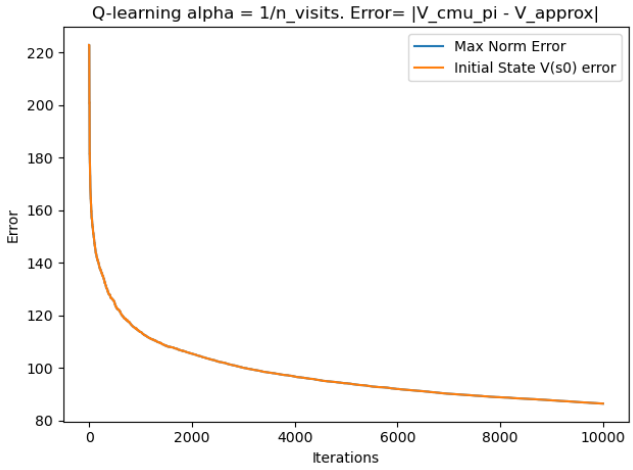


In both cases, we clearly observe that higher lambda gives better convergence values. However, we noticed that with other step-size schedulers, increasing lambda not necessarily improve the error.

i.

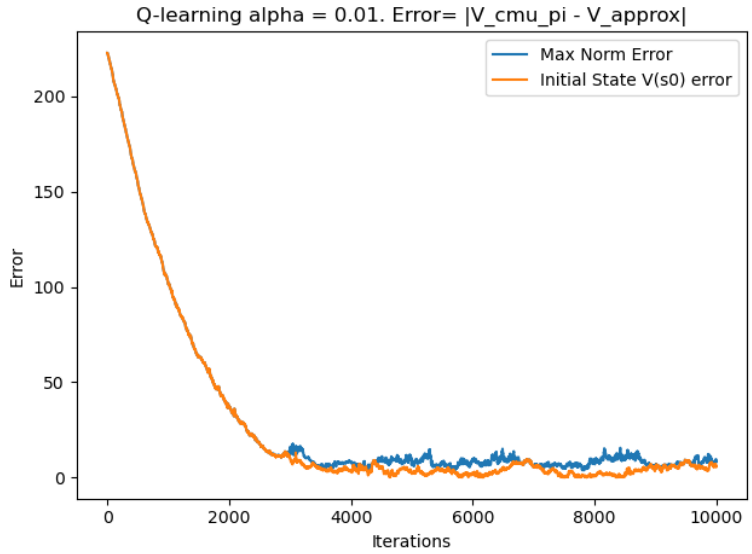
For each step size scheduler, we plot the max-norm of the error $V^* - \hat{V}_Q$ and the initial state error: $V^*(s_0) - \hat{V}_Q(s_0)$. The errors are computed every 100 iterations such that 1 iteration in the plots represent 100 iterations of the Q leaning steps.

$\alpha_n = \frac{1}{\#num\ of\ visits\ to\ s_n}$:



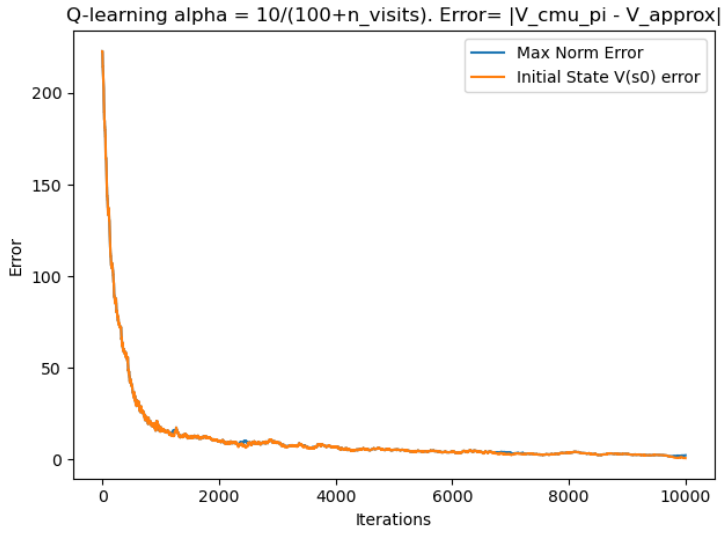
We observe that converge to 0 error is not achieved both in the Max-norm and in the Initial state values. However, this step-size enables stability as we already saw.

$\alpha_n = 0.01$:



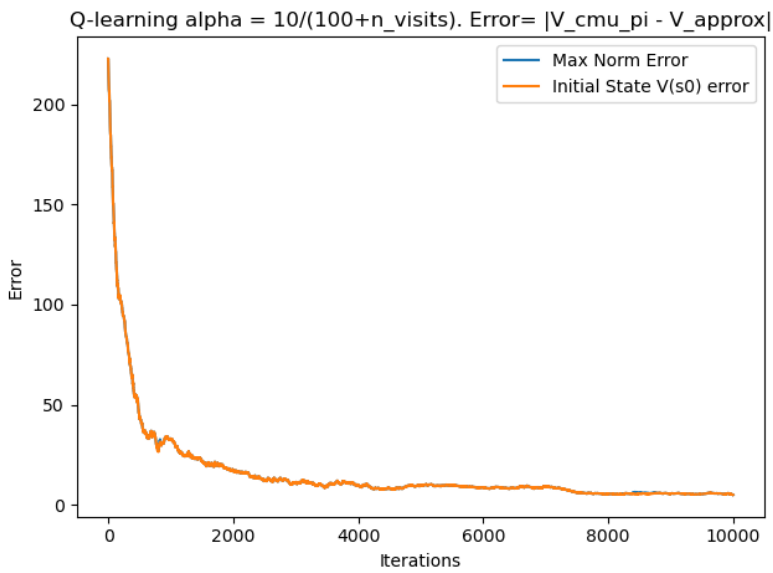
This step-size enables faster learning since it doesn't tend to 0 but the errors are not stable around some 0 error.

$\alpha_n = \frac{10}{100+\#num\ of\ visits\ to\ s_n}$:



This step-size is the best combination of both worlds. Stability and fast convergence are achieved.

j. Using $\epsilon = 0.01$, and the last step-size $\alpha_n = \frac{10}{100+\#num\ of\ visits\ to\ s_n}$, we obtain the following error plots.



As we can see, the errors are approximately the same as with $\epsilon = 0.1$.