Homework 6

Submission by: Apoorv Agnihotri (6604679), Gaurav Niranjan (6599177), Carla López Martínez (6637484)

Q1. Function Approximation

a.

Tabular methods can be viewed as a special case of linear function approximation, where each state (or state-action pair) is represented by a one-hot encoded feature vector. Here's how:

Feature Vectors: For a state s in a state space with n states, the feature vector ϕ(s) is an n-dimensional vector, where:

ϕ(s)_i = { 1 (if i corresponds to s.), 0 (otherwise) }

Linear Function Approximation: The value function V(s) is represented as:

V(s)= w^⊤ . ϕ(s)

where w is a weight vector. Since ϕ(s) is one-hot encoded, V(s) directly corresponds to the weight w_i associated with state s.

Thus, tabular methods are a special case where the features are one-hot vectors, enabling exact representation of each state or state-action value.

b.

Each s_j has different possible exponents given by $c_{i,j}$, which ranges from 0 to n. Therefore, we have n+1 different choices for $c_{i,j}$. The state space has k dimensions, and each dimension s_j can take on one of n+1 values as explained before, so in total the number of unique features x_i(s) is (n+1)^k.

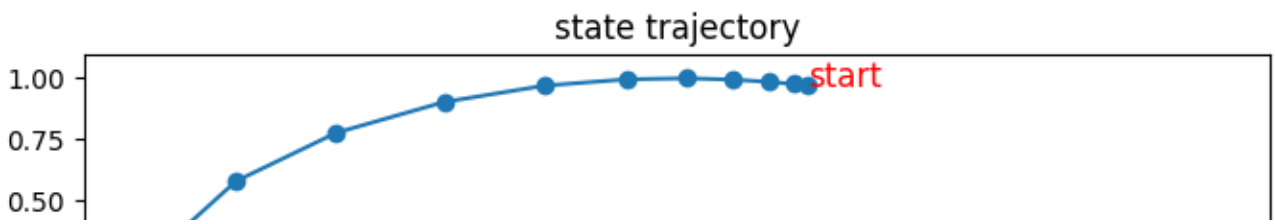
Q2. Feature Designing

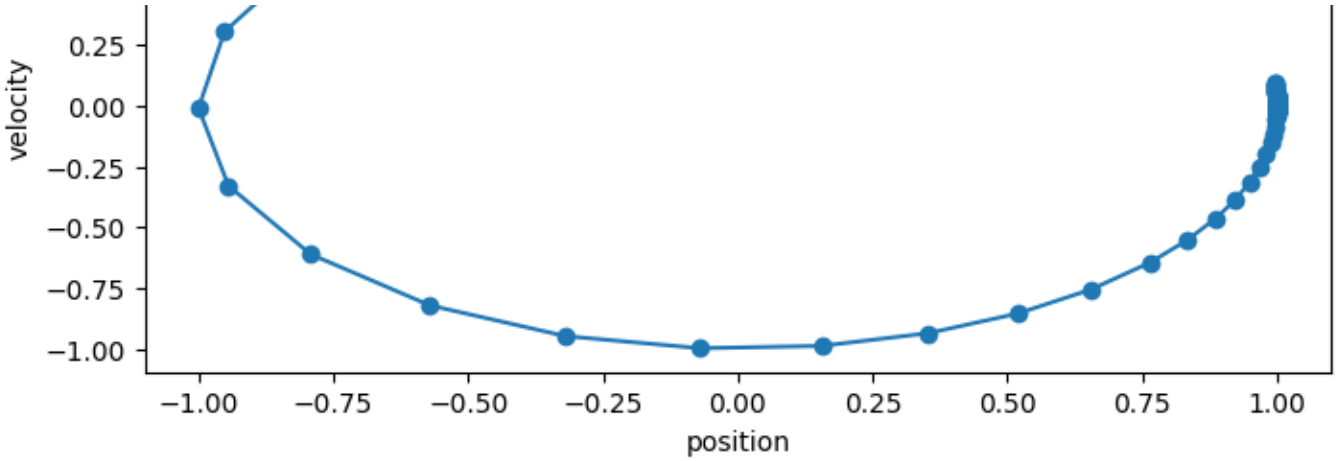
a.

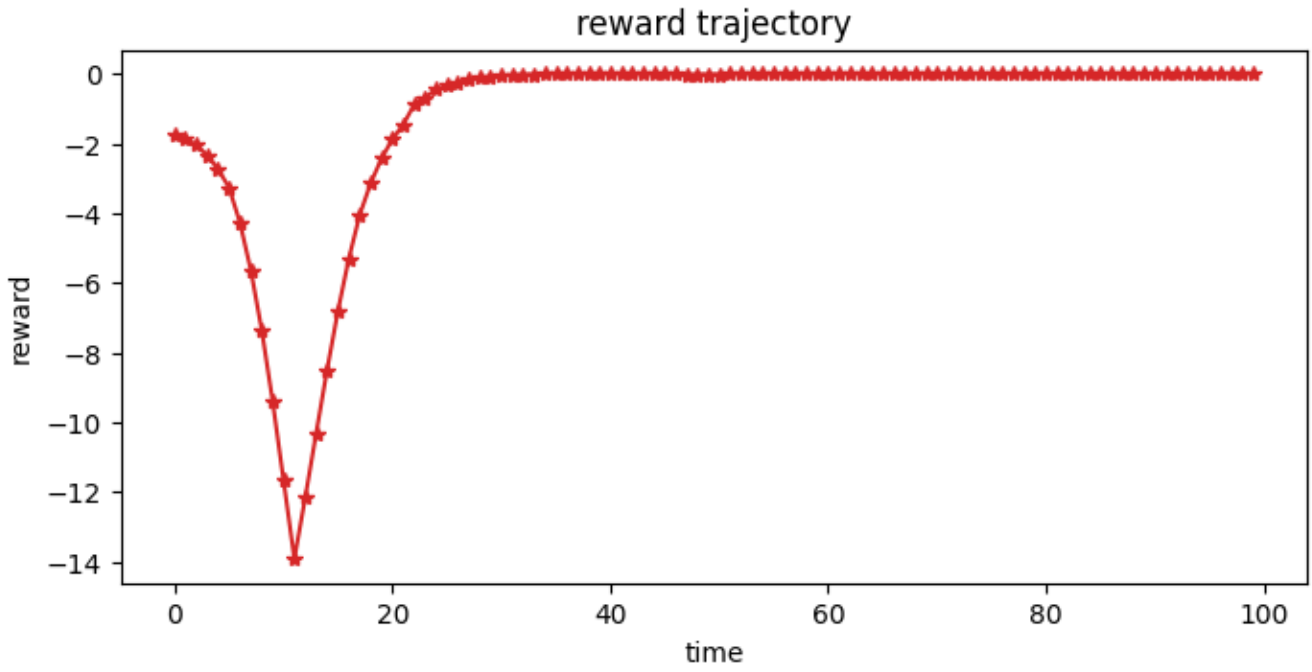
One of the most basic way to encode the state is to directly take the pixel values and treat them as the feature vector.

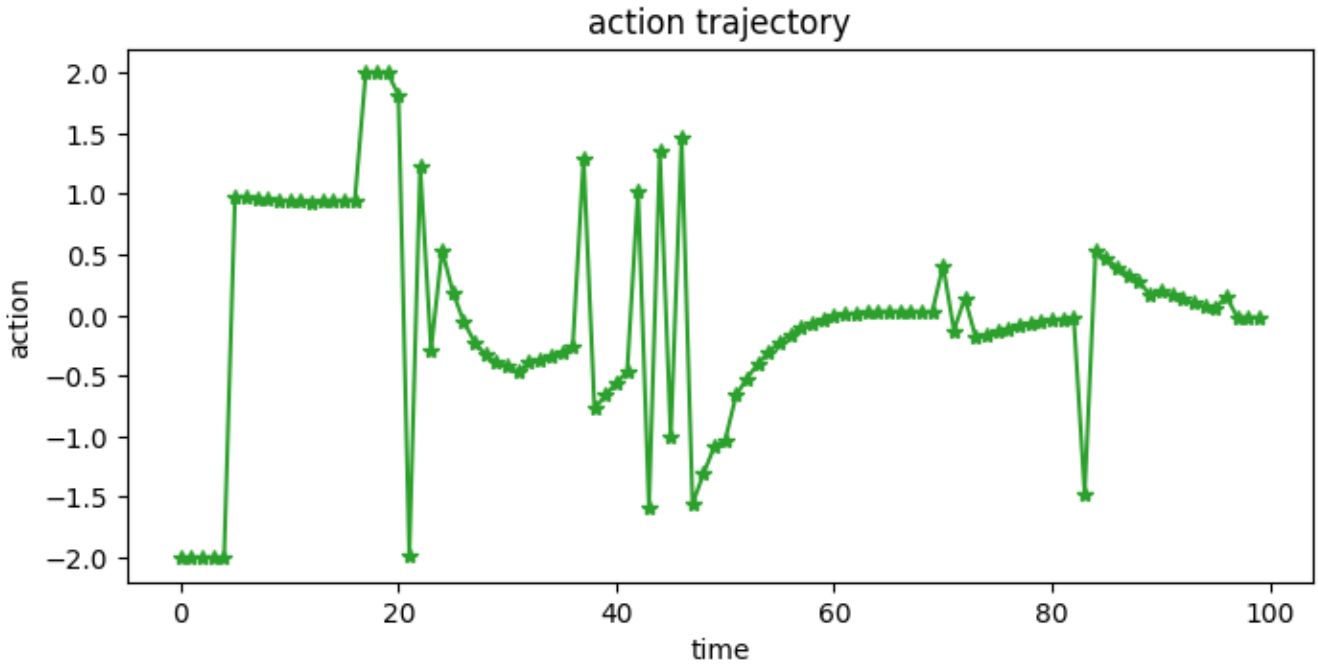
Q3.

a.




velocity
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
position


reward trajectory
reward
0
−2
−4
−6
−8
−10
−12
−14
0
20
40
60
80
100
time

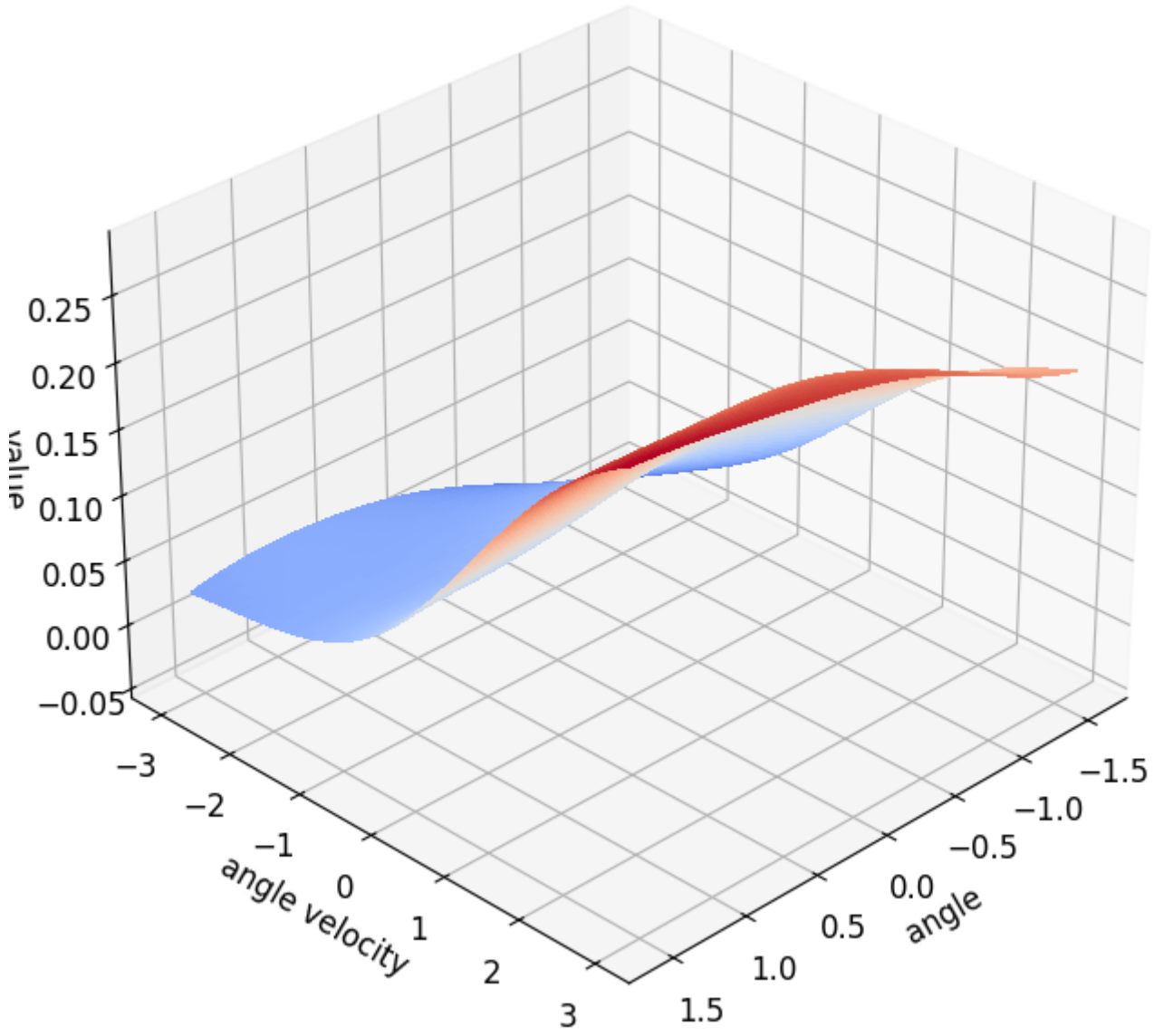

action trajectory
action
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
20
40
60
80
100
time

b.

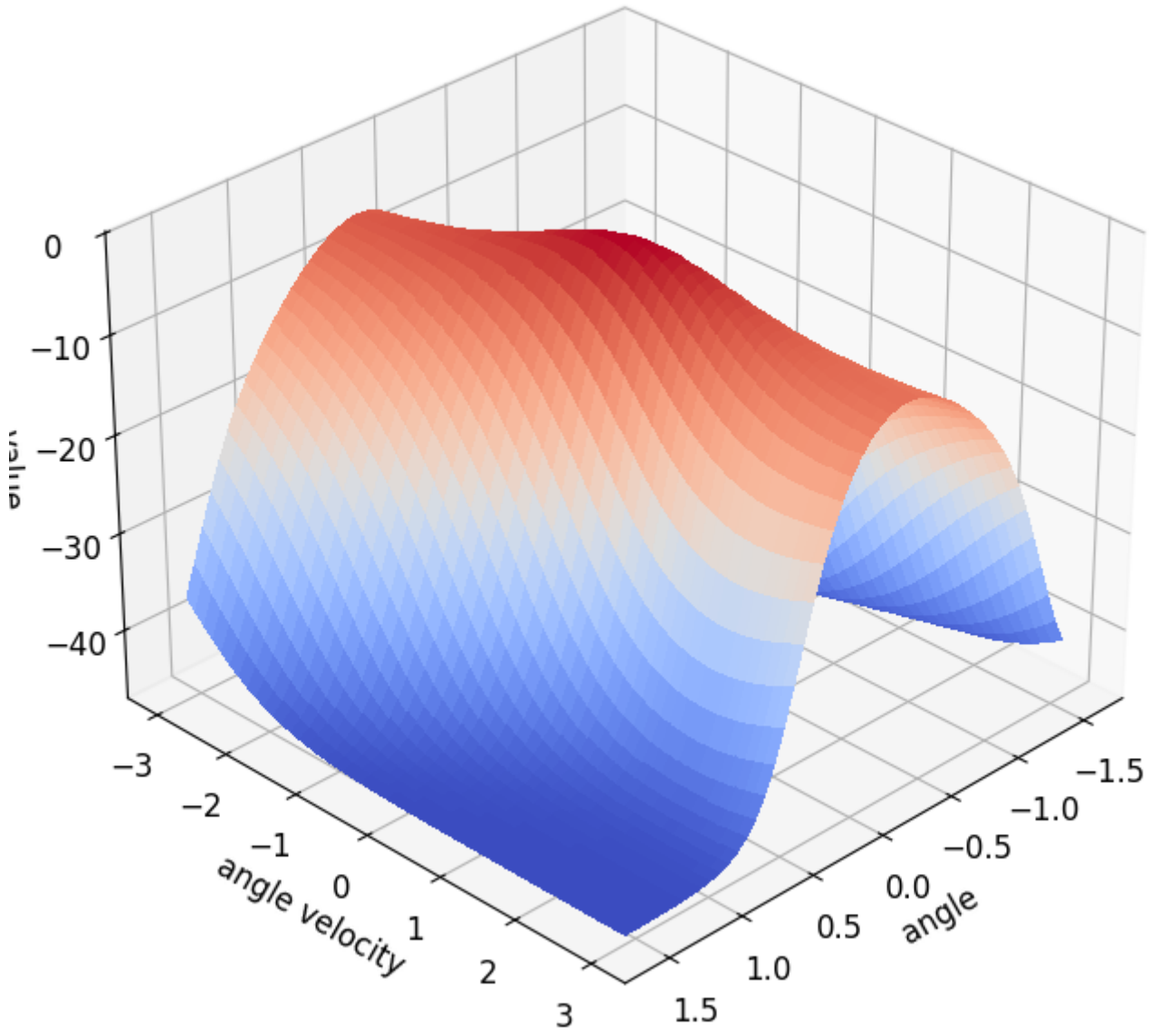
We see that the learning is happening although with a lot of noise.

c. Checking out the Value function before and after learning (gamma = 0.95)

Before


0.25
0.20
0.15
0.10
0.05
0.00
−0.05
value
−3
−2
−1
0
1
2
3
angle velocity
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
angle

After



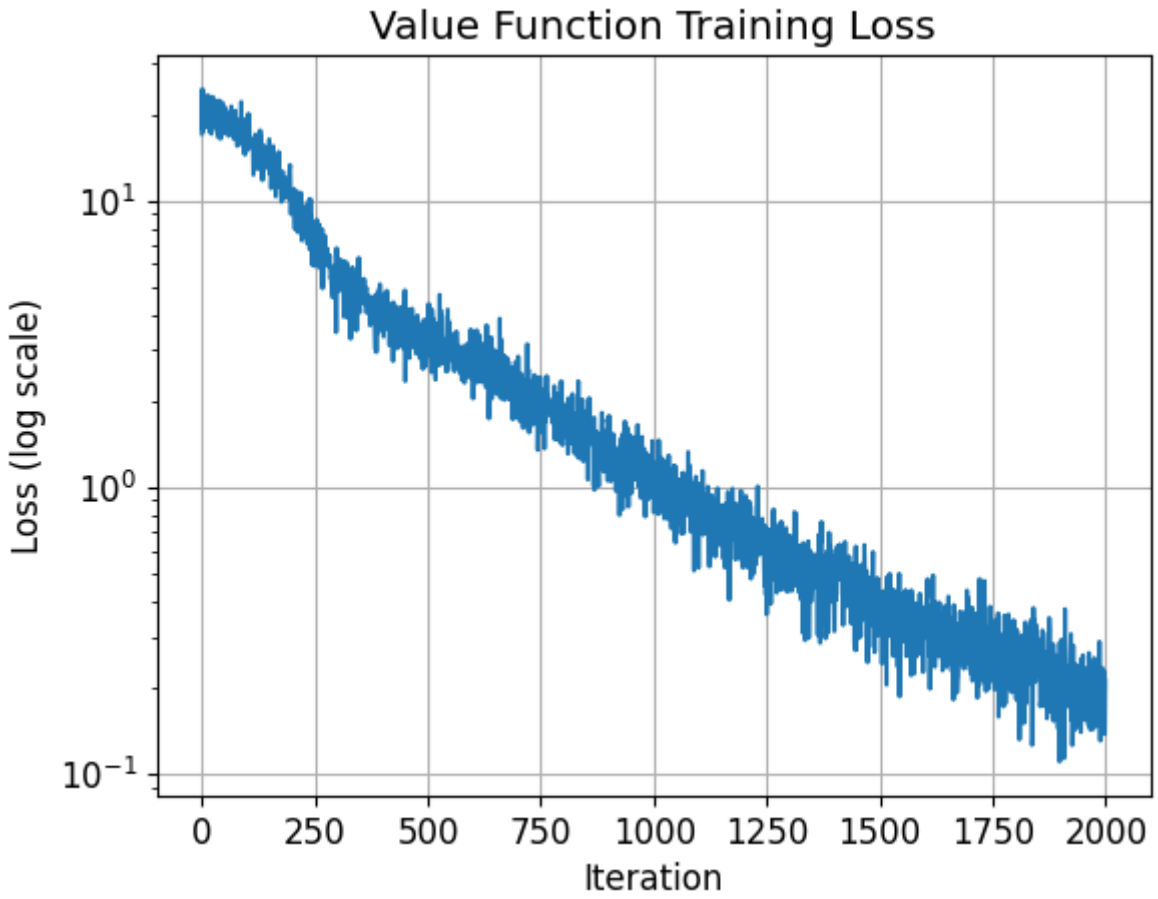
d.

We see that the the final value fuction does make sense because it (in general) assigns higher value to the states where the angle is close to 0. This is to be expected since we want the pendulam to stay upright. That's the objective. Further, if you notice, the angle valocity if it is close to 0 when the angle is 0, that's perfect because that means it will stay in the same position a bit longer.

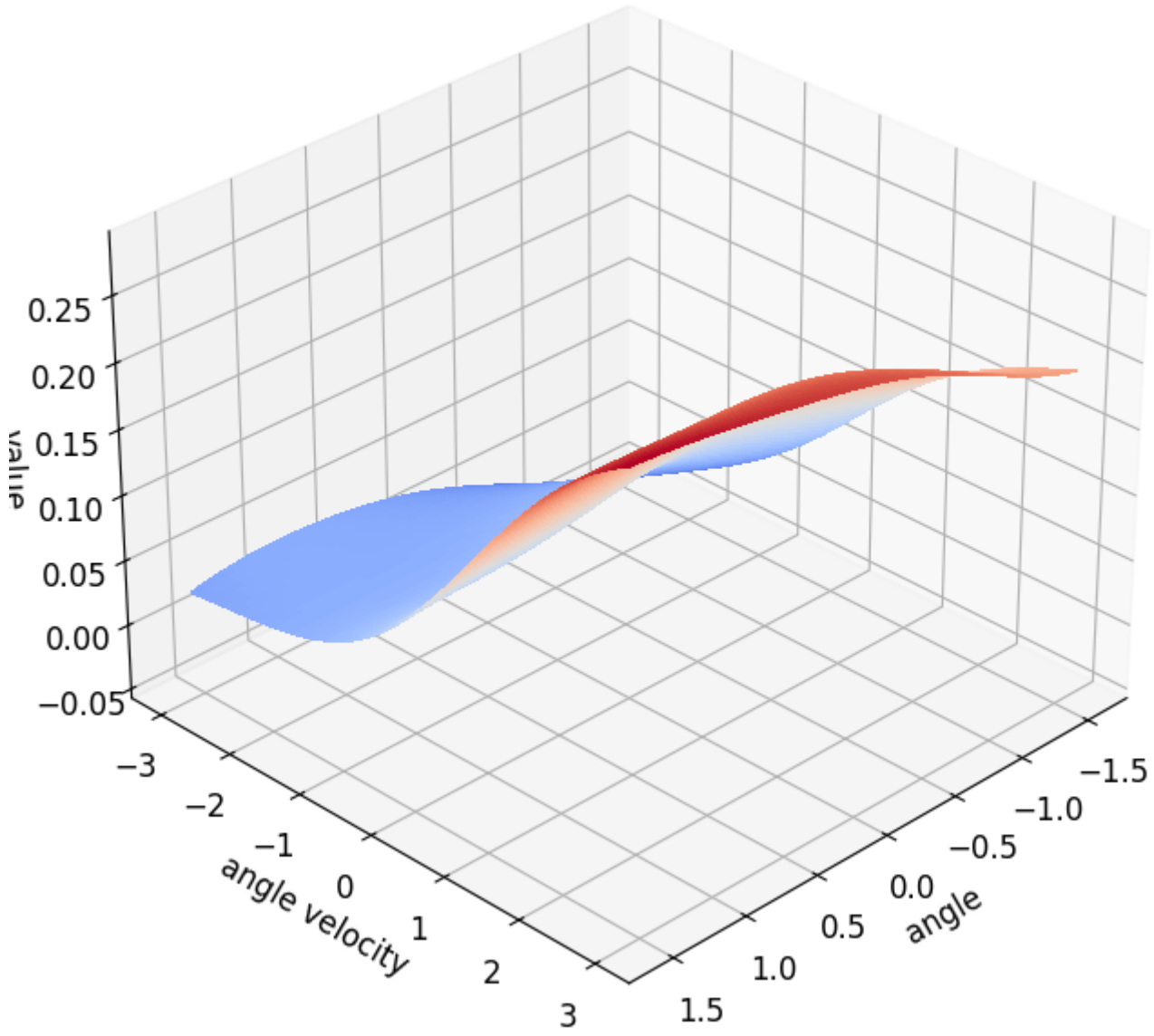
e. Checking out the Value function before and after learning (gamma = 0.5)

In general we observe that the training progresses faster and we are able to get a much lower loss with this discount factor. Further, reducing the discount factor actually makes the value function smoother as well. This makes sense because if the objective is to keep the pendulum upright, it makes sense that states closer to the "ideal" state should be closer in value to each other. Therefore having an abrupt final value function is actually suspicious in the first setting.

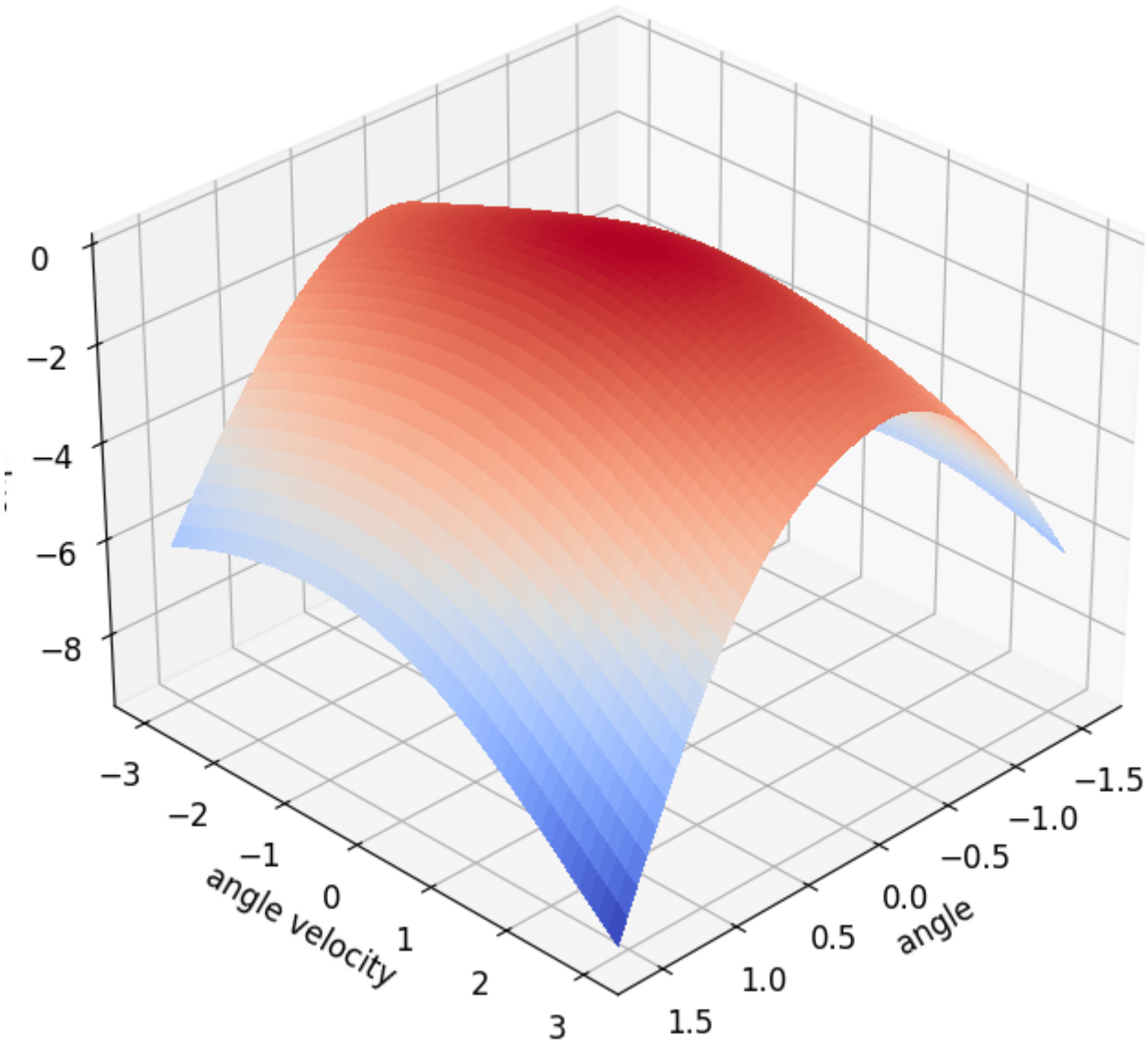
Learning Curve


Value Function Training Loss
Loss (log scale)
10^1
10^0
10^{-1}
0
250
500
750
1000
1250
1500
1750
2000
Iteration

Before


0.25
0.20
0.15
0.10
0.05
0.00
−0.05
value
−3
−2
−1
0
1
2
3
angle velocity
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
angle

After


0
−2
−4
−6
−8
−3
−2
−1
0
1
2
3
angle velocity
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
angle