Report

Policy Perfectionists - Team 9
Likhith Asapu, Aakash Terela, Sukhjinder Kumar, Parshva Bhadra

Algorithm 1 - Dynamic Programming

Explanation

The LQR problem involves finding a control policy that minimizes a cost function, which is a quadratic function of the system state and control input. The goal of dynamic programming is to find an optimal control policy by recursively solving a sequence of subproblems.
The dynamic programming algorithm for the LQR problem involves the following steps:

1. Initialization: Set the cost-to-go function to zero for the final time step.
2. Backward recursion: Starting from the final time step, compute the cost-to-go function for each previous time step by solving a quadratic optimization problem.
3. Forward recursion: Starting from the initial time step, compute the optimal control policy by selecting the control input that minimizes the cost-to-go function for the current state.

The backward recursion step involves solving the following optimization problem at each time step:

$$J_k(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + J_{k+1}(Ax_k + Bu_k)$$

where J_k(x_k,u_k) is the cost-to-go function at time step k for state x_k, and control input u_k, Q is the state cost matrix, R is the control cost matrix, A and B are the state and control matrices, and J_{k+1} is the cost-to-go function for the next time step.
The optimal control u_k is given by -

$$u_k^* = -(R + B^T P_{k+1} B)^{-1} B^T P_{k+1} A x_k$$

And p_k is given by -

$$P_k = Q + A^T P_{k+1} A - A^T P_{k+1} B (R + B^T P_{k+1} B)^{-1} B^T P_{k+1} A$$

The forward recursion step involves using the optimal control input to compute the state at the next time step, and repeating the process until the final time step is reached. The following diagram sums it up -

Backward Iteration: Compute $F(N-k)$ and $P(k)$ $(P(0)=H)$

$$\boxed{F(N-k) = -[R(N-k) + B^T P(k-1)B]^{-1} B^T P(k-1)A} \quad (35)$$

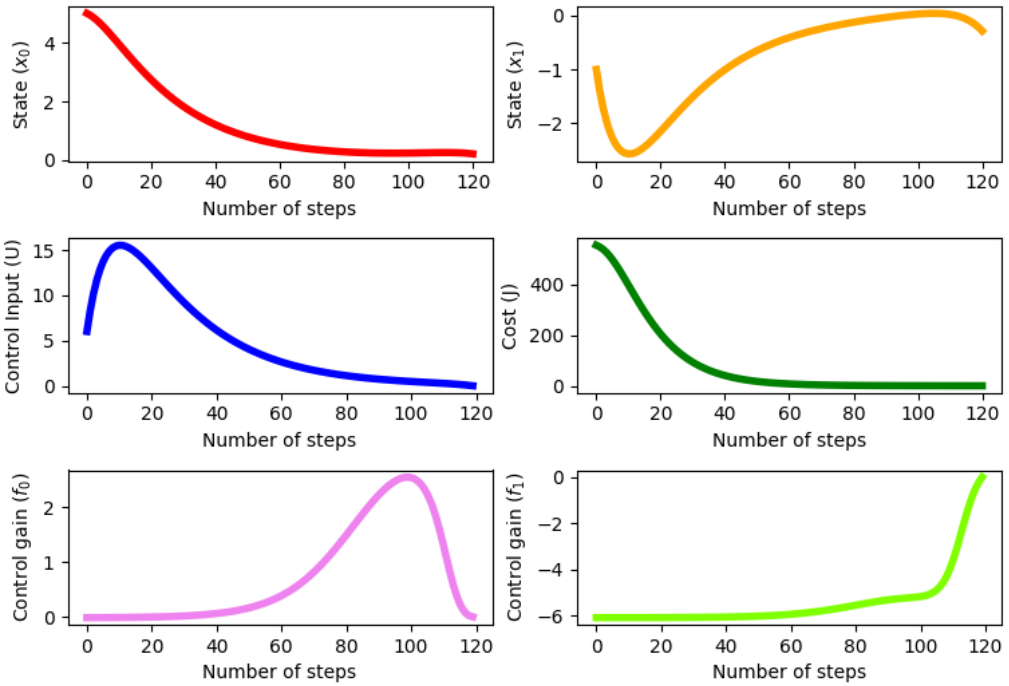
$$\boxed{\begin{aligned} P(k) = [A + BF(N-k)]^T P(k-1)[A + BF(N-k)] + \\ F^T(N-k)R(N-k)F(N-k) + Q(N-k) \end{aligned}} \quad (36)$$

Forward Iteration: Compute $U^*(k)$ and $X(k+1)$ $(X(0)$ is given)

$$\boxed{U^*(k) = F(k)X(k)} \quad (37)$$

$$\boxed{X(k+1) = AX(k) + BU^*(k)} \quad (38)$$

Results



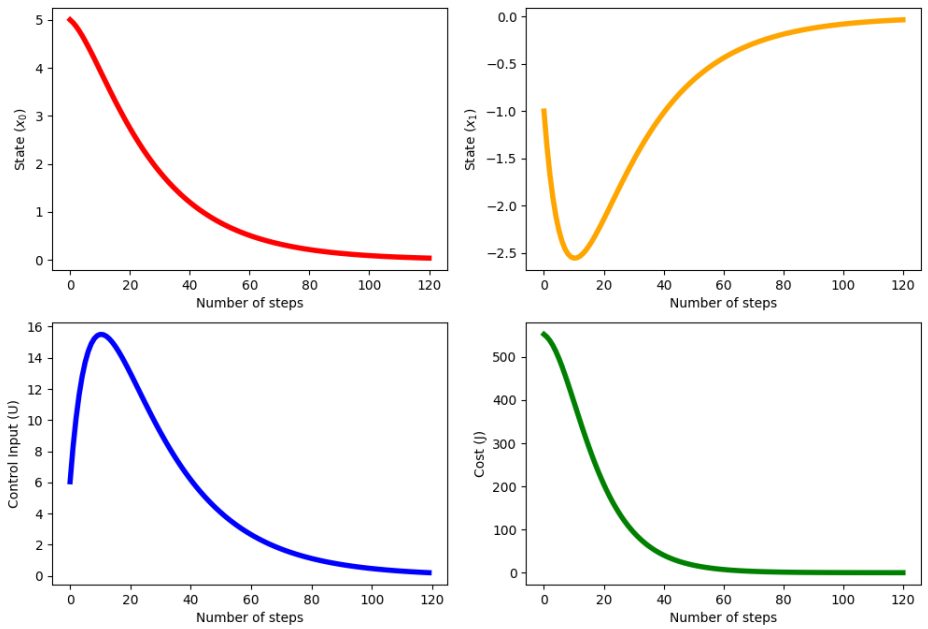
Algorithm 2 - Value Iteration Algorithm

Explanation

- Start with a stabilizing gain F_0 (i.e. $A + BF_0$ has eigenvalues inside a unit circle) and a $P_0 > 0$
- STEP 1: Solve for P_1 using $P_1 = Q + F_0^T R F_0 + (A + BF_0)^T P_0 (A + BF_0)$
- STEP 2: $F_1 = -[R + B^T P_1 B]^{-1} B^T P_1 A$
- Iterate STEP 1 and STEP 2 till P and F converge.

Value iteration algorithm like the Policy iteration algorithm is used for infinite horizon LQR problems. Here we are approximating as N=120 is pretty large. The key idea is instead of finding all P's (i.e. for each iteration) we assume it to be constant and search for that one converging value and use it to compute costs and in finding optimal control policy. We use the Riccati equation to solve for P and F. And from there we can find optimal cost by J = ½ * x^T * P * x and optimal control by u = Fx.

Results



Algorithm 3 - REINFORCE/ Vanilla Policy Gradient

Explanation

This algorithm is a model-free algorithm(where A is unknown). The algorithm learns about the environment through exploration. An agent learns through episodes which is just a state-action-rewards sequence. The rewards/cost in this method can be defined as

$$R(\tau) = (G_0, G_1, ..., G_H)$$

Where G_k is called the total return, or future return, at time step ***k*** for the transition ***k.*** The goal of this algorithm is to find the weights θ of the neural network to minimize the expected return that we denote by U(θ) and can be defined as

$$U(\theta) = \sum_{\tau} \mathbb{P}(\tau;\theta) R(\tau)$$

Given below is the pseudo code of this algorithm.

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Algorithm parameter: step size $\alpha > 0$)
Initialize the policy parameter θ at random

(1) Use the policy π_θ to collect a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, ...a_H, r_{H+1}, s_{H+1})$

(2) Estimate the Return for trajectory τ: $R(\tau) = (G_0, G_1, ..., G_H)$
where G_k is the expected return for transition k:

$$G_k \leftarrow \sum_{t=k+1}^{H+1} \gamma^{t-k-1} R_k$$

(3) Use the trajectory τ to estimate the gradient $\nabla_\theta U(\theta)$

$$\nabla_\theta U(\theta) \leftarrow \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

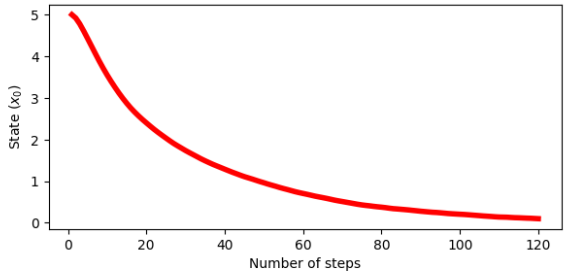
(4) Update the weights θ of the policy

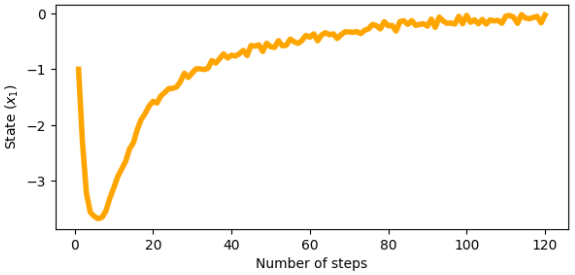
$$\theta \leftarrow \theta + \alpha \nabla_\theta U(\theta)$$

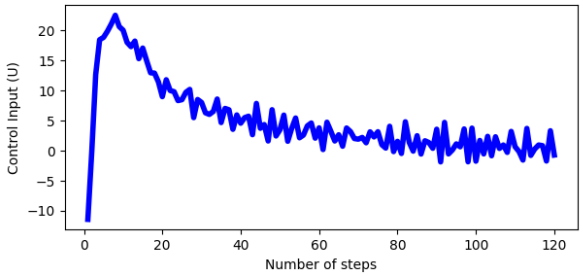
(5) Loop over steps 1-5 until not converged

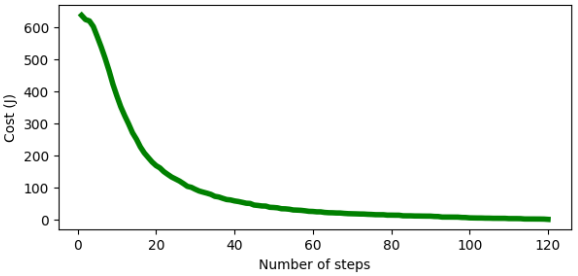
The first step involves generating the episode/trajectory using the policy π. Then calculated G_t by summing the future rewards. The third step involves calculating the gradient to update the neural network weights to minimize the cost. This is calculated by multiplying the log probability of the action being sampled under the given policy and the cumulative reward G_t. Then calculate the weights of the neural network by summing the gradients obtained in the previous step for a given episode. Do this until convergence.

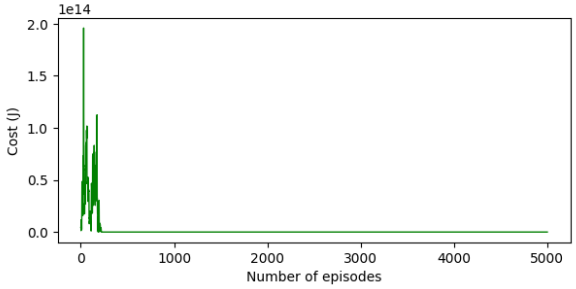
Results

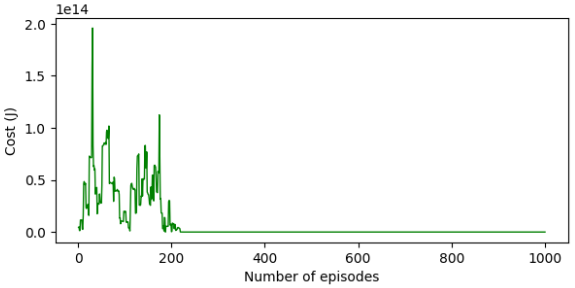












Algorithm 4 - Actor-Critic

Explanation

In the Actor-Critic method, the policy is referred to as the actor that proposes a set of possible actions given a state, and the estimated value function is referred to as the critic, which evaluates actions taken by the actor based on the given policy.

Both the Actor and Critic are represented by neural networks The Actor neural network takes input as state and outputs a random action determined by the probability distribution of all possible actions. The Critic neural network takes input as state and outputs a single value which is the predicted value function.

Except for the Loss, the Actor neural network is very similar to the neural network used in Policy Gradient (REINFORCE). The policy gradient expression for the Actor is changed as shown below,

$$\nabla J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t, s_t) A_{\pi_\theta}(s_t, a_t)$$

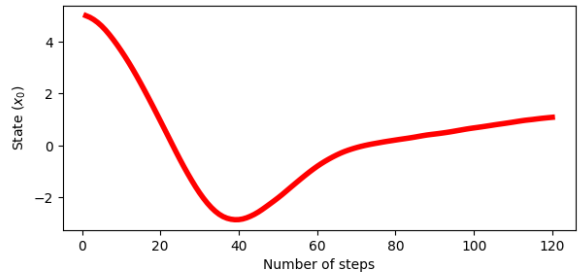
We can see that the reward is replaced by $A_\pi(s_t,a_t)$, which is called the Advantage function.

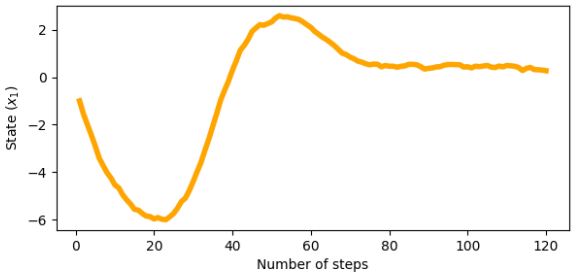
$$A_{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)$$

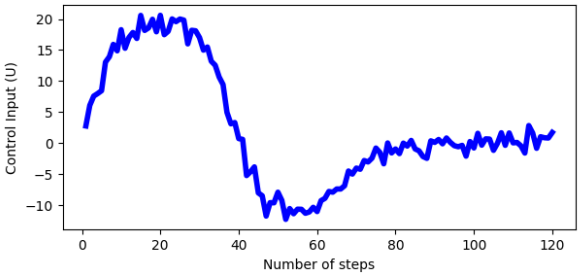
So, we update the Actor neural network parameters(θ) by the following, $\theta \leftarrow \theta + \nabla J(\theta)$.

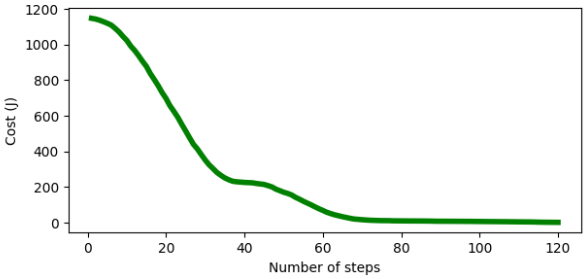
The Critic neural network follows a different Loss for learning as the goal for the Critic is to accurately predict the value function. The update for Critic neural network parameters(ω) is done by following, $\omega \leftarrow \omega + \delta_t$, where δ_t is the Loss between the predicted value function and the true value function. For our implementation, the Loss function we used is MSE Loss. The terms used in the Loss function are already present in the Advantage function, the predicted value function is given by $V_\pi(s_t)$ and the true value function is given by $r(s_t,a_t) + V_\pi(s_{t+1})$.

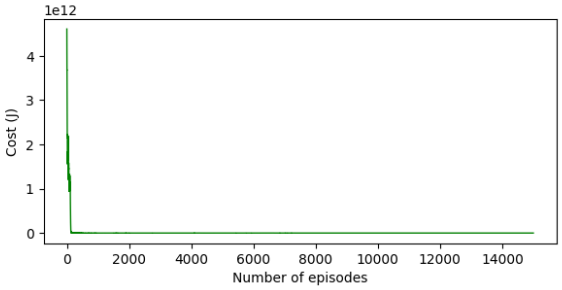
Results

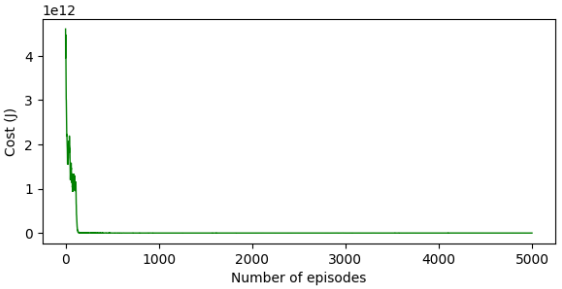

State (x_0)
Number of steps


State (x_1)
Number of steps


Control Input (U)
Number of steps


Cost (J)
Number of steps


1e12
Cost (J)
Number of episodes


1e12
Cost (J)
Number of episodes