RL Project Report

Introduction

We have solved the swing-up pendulum problem which involves controlling a pendulum to swing from its downward hanging position to an upright position and balancing it there. The goal is to apply the right amount of torque at each time step to swing the pendulum up and keep it balanced at the upright position.
The model equations are:

$$ml^2\frac{d^2\theta(t)}{dt^2} + b\frac{d\theta(t)}{dt} + mglsin(\theta(t)) = u(t)$$

We have used a model-based method: SDRE and 3 model-free methods: DDPG, SAC, and PPO.

Custom Environment

We have modified the Pendulum-v1 gym environment to account for air friction (The gym environment doesn't account for air friction). We have used the conservation of energy and conservation of angular momentum to get our step equation:
$\theta' = \theta' + ((3g/2l) * sin\theta + 3u/(ml ** 2) - b\theta'/(ml ** 2))dt$

DDPG - Deep Deterministic Policy Gradient (Off Policy)

Overview

Uses the structure of the Actor-Critic and the memory buffer. The DDPG algorithm maintains an actor network that takes in observation (state) and outputs the best action possible at that state, a critic network that takes in observation and corresponding action and outputs the Q value (this determines how good the action is at that observation) and a memory buffer to store all the transition information this contains state, action chosen in that state, observed reward, next state.

Implementation

Memory Buffer:

A memory buffer is used to store the transitions during episodes and sample some minibatch of experience and train the network.

Training of Actor Network:
To train the actor network we assume that the critic network is well trained and tune the parameters of the network to maximize the output Q value. Our objective is to maximize the below function

$$J(\theta) = \mathbb{E}[Q(s, a)|_{s=s_t, a_t=\mu(s_t)}]$$

So, the negative of the above is used as the loss function for the actor-network.

Training of Critic Network:
The critic Q(s, a) is learned using the Bellman equation as in Q-learning (Q = reward + discount-factor*Q_next). The reward + discount-factor*Q_next is the target value and Q is the predicted value. Our objective is to minimize the error between the target value and the predicted value. So, MSE loss is used to train the critic network.

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right]$$

Actor and Critic Target Networks:

In a transition stored in the memory buffer we have the following values i.e, state, action, reward, next state. In the Bellman equation, to get the Q value from the critic network we send state and action values from the transition stored and can also get the reward from the transition, and to get Q_next we use Actor and Critic Target Networks. We send the next state in the transition to the target actor-network, this next state along with the output of the target actor network is then sent to the target critic network to get Q_next.

These Target networks are not trained and their parameters are updated using the parameters of the actor and critic networks as follows

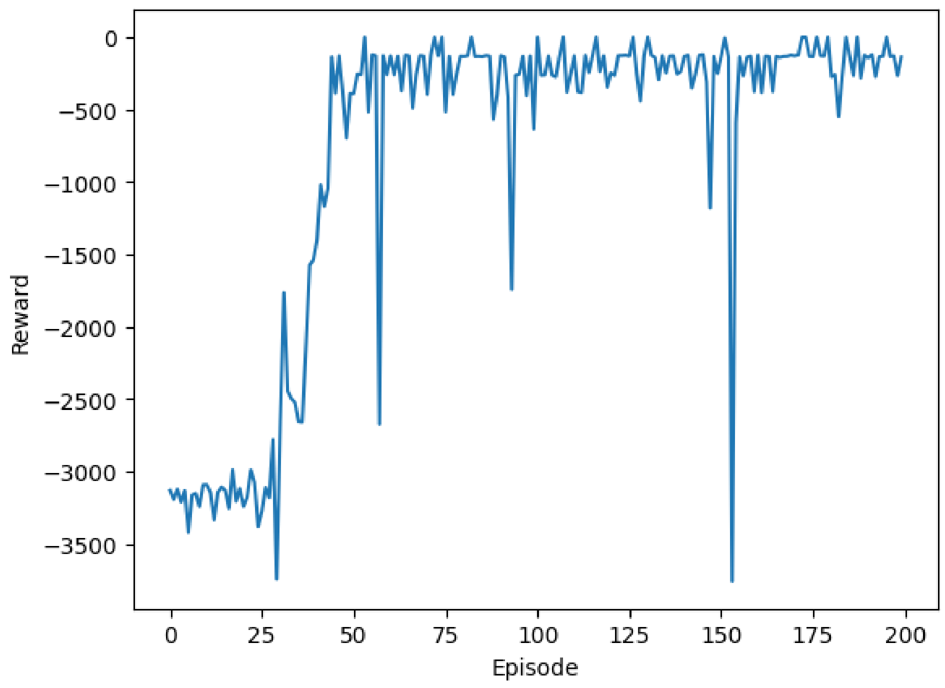
$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

Where θ^{Q} are the parameters of the critic function, $\theta^{Q'}$ are the parameters of the target critic function, θ^{μ} are the parameters of the actor function, and $\theta^{\mu'}$ are the parameters of the target actor function.We take small values for Tau, so that the target values are constrained to change slowly, greatly improving the stability of learning. Ornstein-Uhlenbeck noise is added to the action to increase exploration.

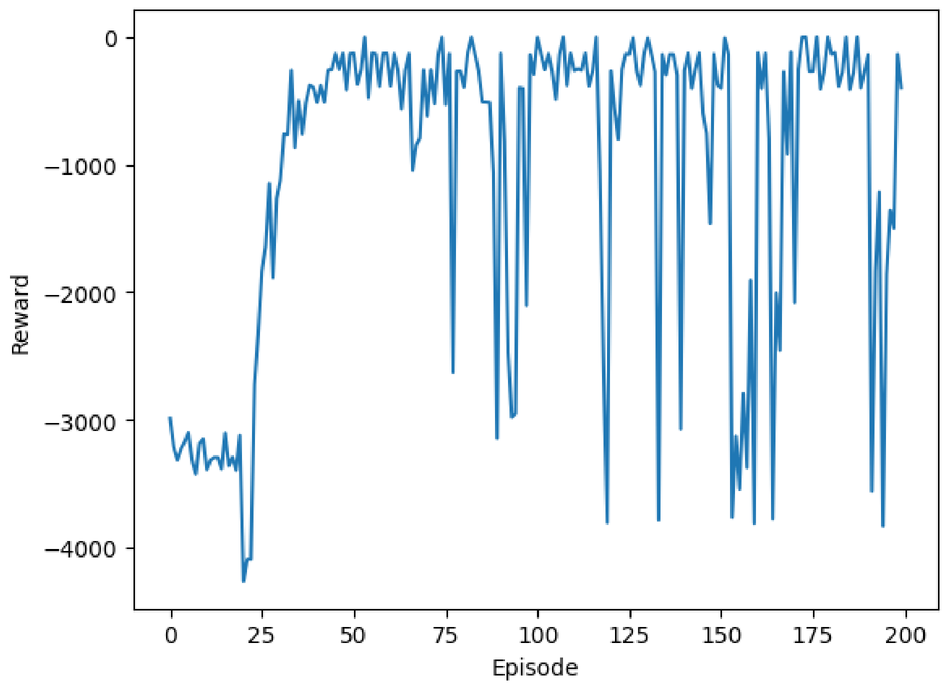
Results

Hyperparameters:
Maximum no.of steps in an episode = 500
Learning rate of actor = 0.001
Learning rate of critic = 0.002
Discount factor = 0.9
Tau (soft replacement) = 0.01
Graphs:

1) Tau = 0.01 : Convergence in around 60 episodes

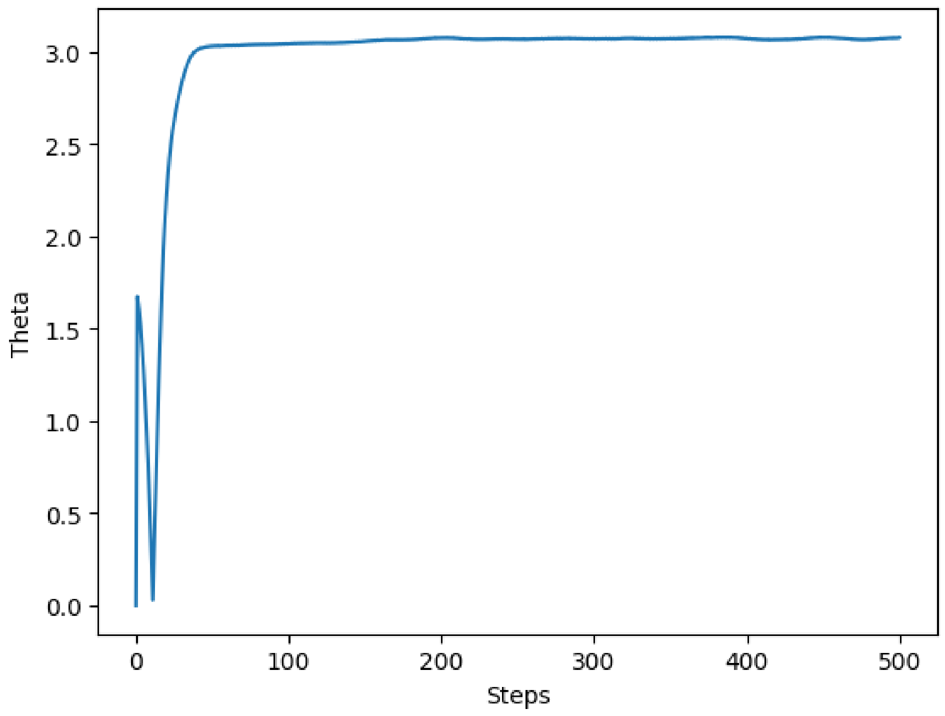


2) Lr = 0.01 : Fluctuations

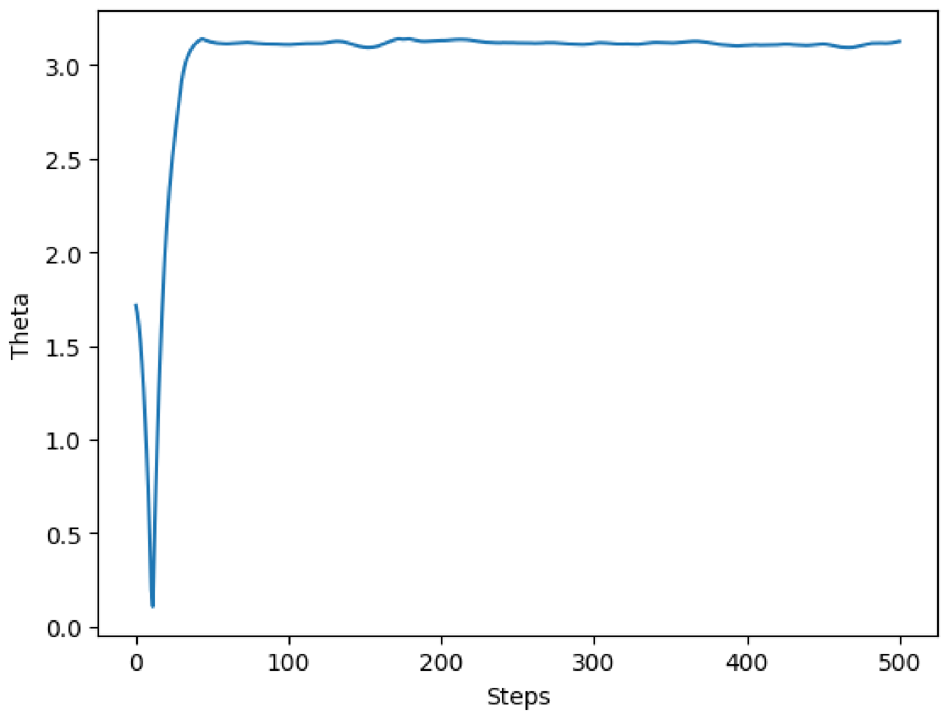


Plots for theta:

1) Initial Theta = 0 : Converging to theta = π



2) Theta = random (~1.7) : Converging to theta = π



Soft Actor Critic (Off Policy)

Overview

SAC is defined for RL tasks involving continuous actions. The biggest feature of SAC is that it uses a modified objective function. Instead of only seeking to maximize the long term rewards, SAC seeks to also maximize the entropy of the policy. Through this, SAC solves the problem of extensive hyperparameter tuning which is required for off-policy methods like DDPG.
SAC overcomes the brittleness problem by encouraging the policy network to explore and not assign a very high probability to any one part of the range of actions.

Implementation

The objective function for SAC is:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t)) \right].$$

Here H is entropy i.e, Log probability of taking an action at state s_t

SAC makes use of the following networks: a actor network, two critic networks, their corresponding target networks and another network to train the alpha parameter that controls the trade-off between exploration and exploitation.
Actor Network:
Actor Network takes in a state and returns the action and log probability for taking the action in the state. This makes SAC stochastic. And training of the actor network is done by using the above loss function.
Critic Network:
Critic Network takes in state and action and returns the Q value as in DDPG. In SAC two critic networks are used and minimum of the two returned Q values is considered in order to overcome the over-estimation bias.
We train the two critic network by minimizing the following error

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right)^2\right]$$

Q^ is the target Q value computed from the values of target critic networks as in DDPG.

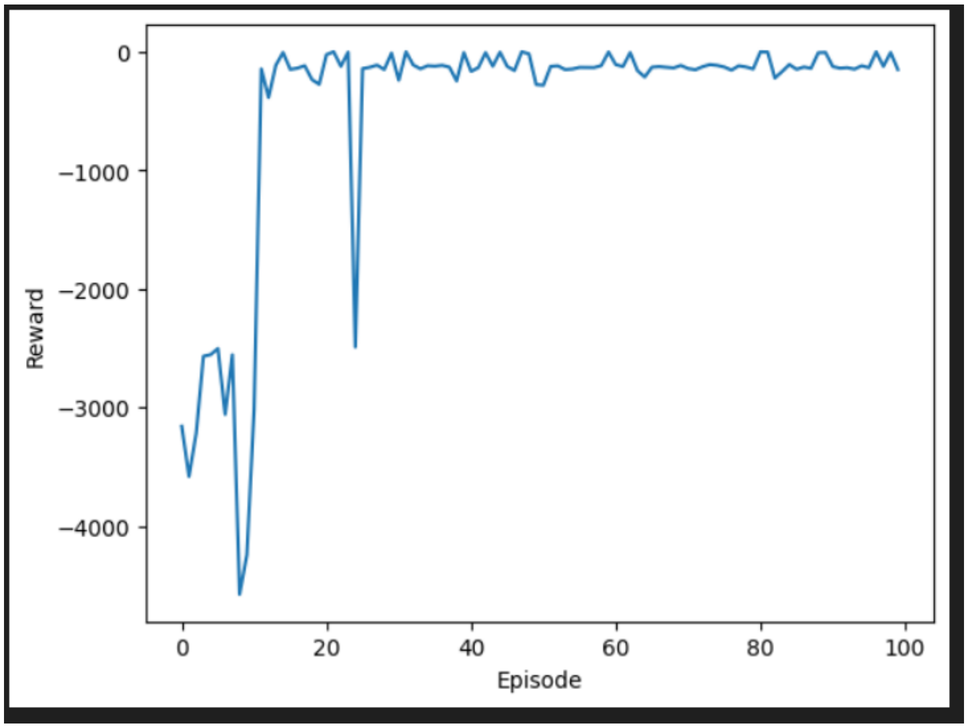
Target Actor and Target Critic networks:
The are updated in the same way as in DDPG.

Results

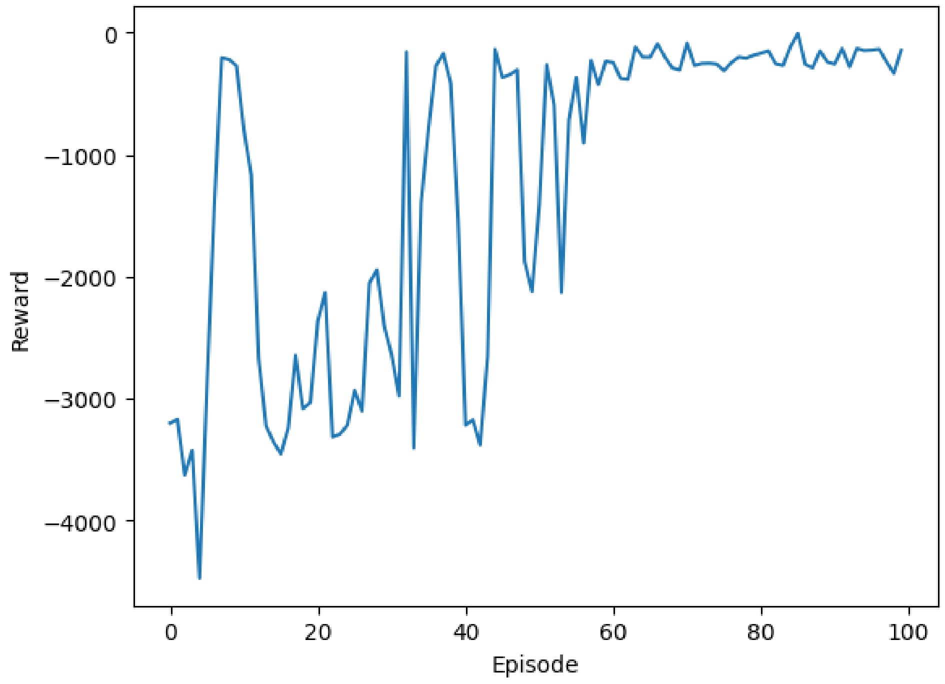
Hyperparameters:
Maximum no.of steps in an episode = 500
Learning rate of policy network = 0.001
Learning rate of Q network = 0.001
Discount factor = 0.98
Tau = 0.005

Graphs:

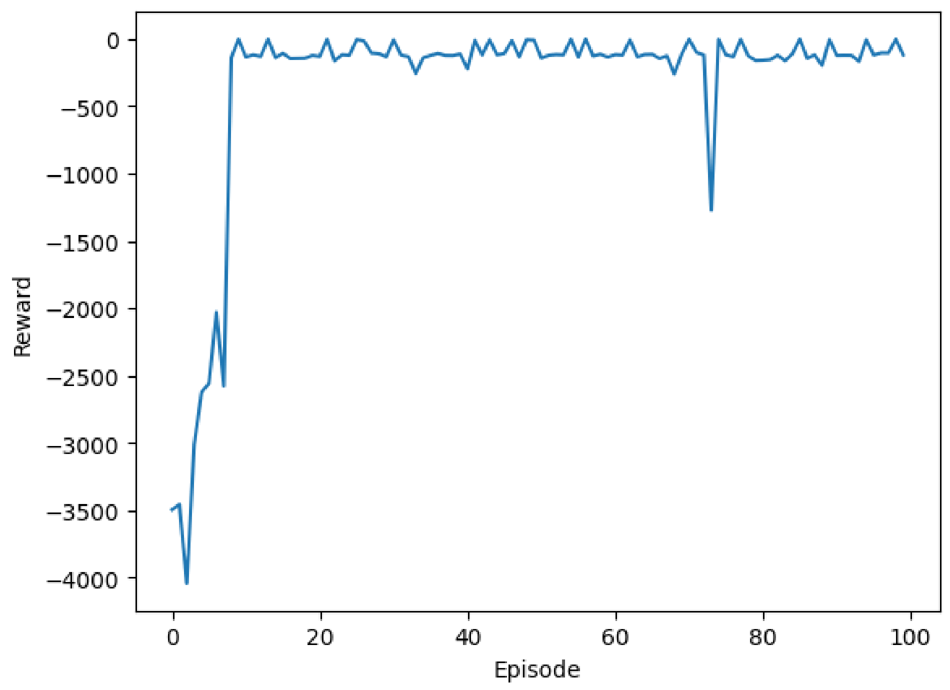
1) Tau = 0.005: Convergence in around 20 episodes



2) Tau = 0.1: Fluctuations due to unstability

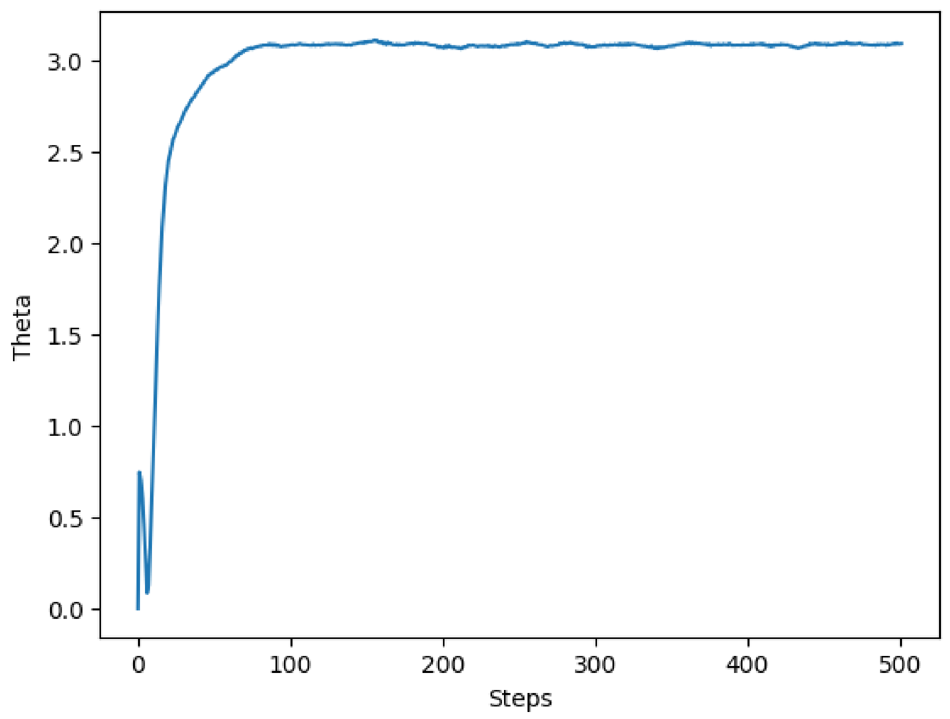


3) Tau = 0.005, lr = 0.01 : Convergence in around 10 episodes



Plots for state trajectory (theta):

1) Initial theta = 0 : Converging to theta = $\boldsymbol{\pi}$. More stable than DDPG



PPO

Overview

Actor critic methods remain unstable as large updates to the policy network can throw off the learning. PPO method ensures that policy update is constrained, thus avoiding large updates. Hence, PPO method is more stable than DDPG and SAC. However, as PPO doesn't make use of the past memory (replay buffer), thus it takes more episodes to converge.

PPO is simpler to implement than TRPO, requiring no second-order derivatives.
We have used the PPO variant with the clipped objective.

Implementation

Rather than bothering with changing penalties over time, we simply restrict the range within which the policy can change. As advantages achieved by updates outside the clipping range are not used for updating purposes, we provide an incentive to stay relatively close to the existing policy.
The objective function is:

$$\mathcal{L}_{\pi_\theta}^{CLIP}(\pi_{\theta_k}) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\left[\min\left(\rho_t(\pi_\theta,\pi_{\theta_k})A_t^{\pi_{\theta_k}}, \text{clip}(\rho_t(\pi_\theta,\pi_{\theta_k}),1-\epsilon,1+\epsilon)A_t^{\pi_{\theta_k}}\right)\right]\right]$$

Advantage basically represents how much better or worse the action taken in a state was compared to the expected value at that state.

$$A(s,a) = E_\tau[G(\tau)|s_0 = s, a_0 = a] - V(s)$$

The policy ratio term compares the probabilities of the actions selected under the current policy and the probabilities of the actions under the old policy. The policy ratio measures how the new policy deviates from the old policy for the given state-action pairs.

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)}$$

We have used GAE (Generalized Advantage Estimate) to calculate the advantages.

Results

Hyperparameters:

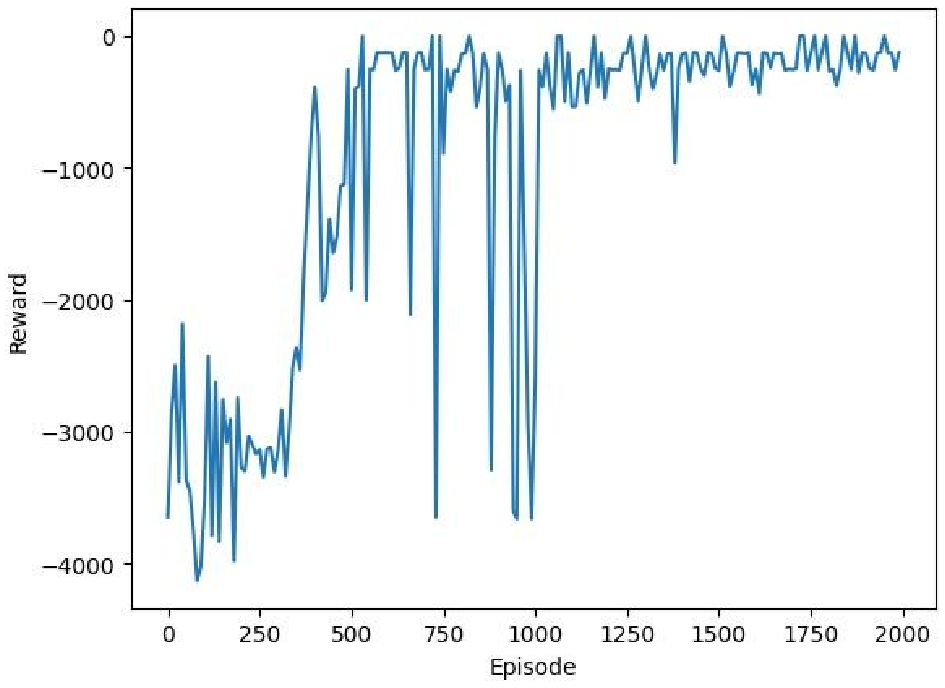
Maximum no.of steps in an episode = 1000

Policy learning rate = 3e-4

Value learning rate = 1e-2

PPO clip value = 0.2

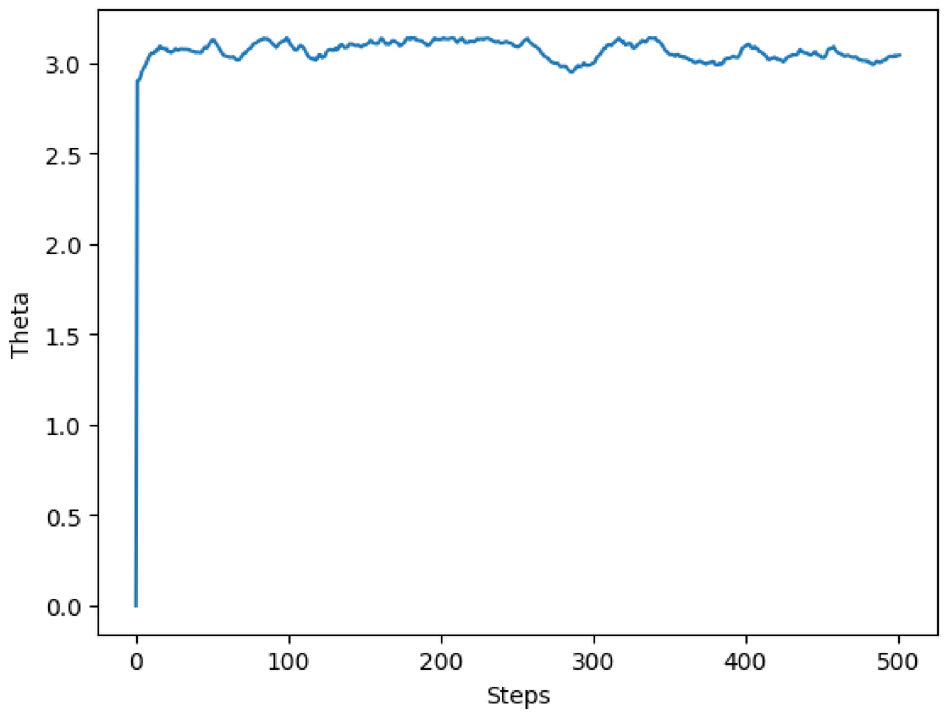
Target KL divergence = 0.02



Convergence at around 1000 episodes.

Plot for theta:

1) Initial theta = 0 : Converging to theta = π. More stable than DDPG and SAC



SDRE Method(Non-Linear LQR)

Conversion to the form X' = A(x).X+B

Note: Equations referred from this paper:
https://www.researchgate.net/publication/356213363_Modified_Infinite-Time_State-Dependent_Riccati_Equation_Method_for_Nonlinear_Affine_Systems_Quadrotor_Control

$$m\ell^2 \frac{d^2\theta(t)}{dt} + b\,\frac{d\theta(t)}{dt} + mg\ell\,\sin(\theta(t)) = u(t)$$

$$\theta_1 = \theta$$

$$\theta_2 = \dot{\theta}_1$$

$$\Rightarrow \quad m\ell^2\,\ddot{\theta}_2 + b\,\dot{\theta}_1 + mg\ell\,\sin(\theta_1(t)) = u(t)$$

$$\dot{\theta}_2 = \frac{-\,b\theta_2 - mg\ell\sin\theta_1 + u(t)}{m\ell^2}$$

$$\dot{\theta}_2 = -\frac{mg\ell}{m\ell^2}\sin\theta_1 - \frac{b\theta_2}{m\ell^2} + \frac{u(t)}{m\ell^2}$$

$$\Rightarrow \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -g/\ell \times \frac{\sin\theta_1}{\theta_1} & \frac{-b}{m\ell^2} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \frac{1}{m\ell^2}\,u(t)$$

$$\dot{X} = A(x)\,x + BU$$

Overview

The Linear Quadratic Regulator method is a popular approach for solving the optimal control for linear dynamical systems which are in the form X’ = AX+BU, where A and B are constants. However, the State Dependent Riccati equation(SDRE method) is a technique that extends the LQR method to be applied for Non-Linear systems which are in the form X’ = A(x).X + BU , where A(x) is the function of x.

The basic idea of this method is to discretize the system into finite time steps and apply LQR in every step. At a particular x, we can assume A(x) to be constant and the

system tends to be Linear, so at this particular point of time we can apply LQR on the system.
Riccati equation and input U equation are derived based on the fact to minimize the loss function J. J is given by

$$J(\mathbf{u}) = \frac{1}{2}\int_0^\infty (\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u})dt,$$

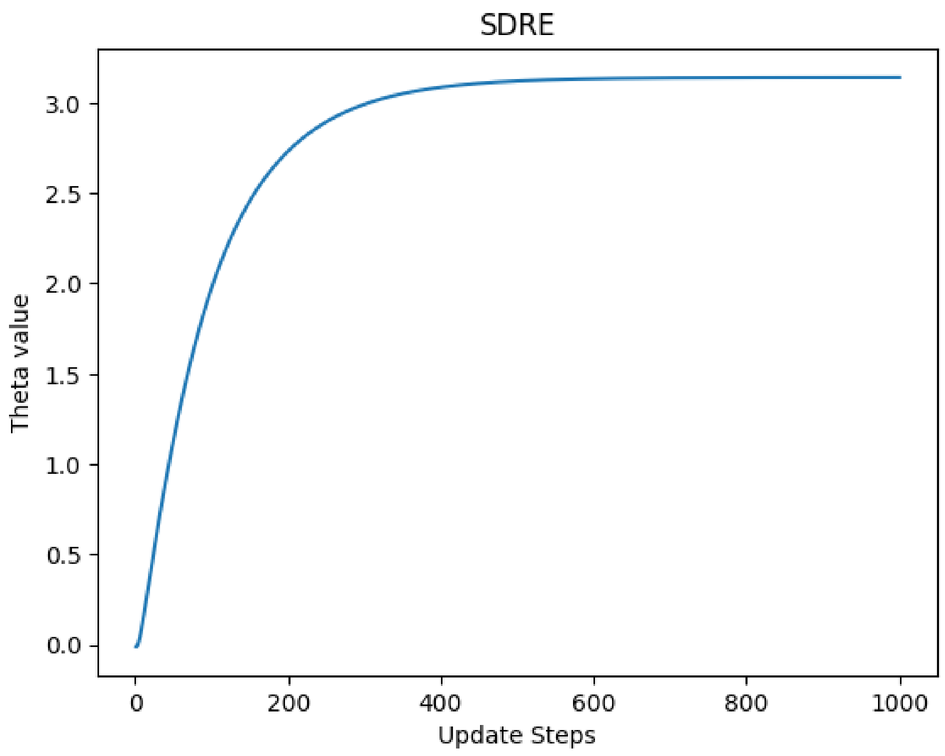
Implementation

- Start with arbitrary matrix Q and scalar R, let the initial angular velocity be a random number.
- Substituting x in this equation A(x) becomes a constant matrix.
- $$\mathbf{K}(\mathbf{x})\mathbf{A}(\mathbf{x}) + \mathbf{A}^T(\mathbf{x})\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{x})\mathbf{B}\mathbf{R}^{-1}(\mathbf{x})\mathbf{B}^T\mathbf{K}(\mathbf{x}) + \mathbf{Q}(\mathbf{x}) = \mathbf{0}$$
  $$\mathbf{u} = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{K}(\mathbf{x})\mathbf{x}$$
- So applying U, x changes so as A(x) also changes . So according to the new x we repeat the above steps.

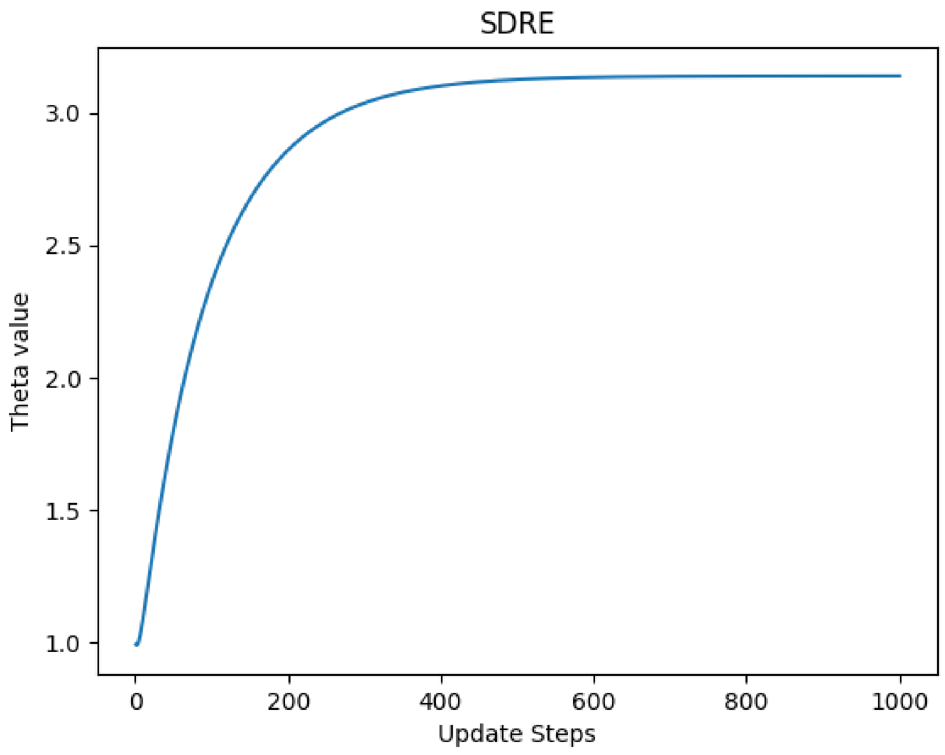
Results

Initially when we start with any angle from the vertical, the system is finally converging to the uppermost position which can be described as theta = $\boldsymbol{\pi}$.
dt = 0.01

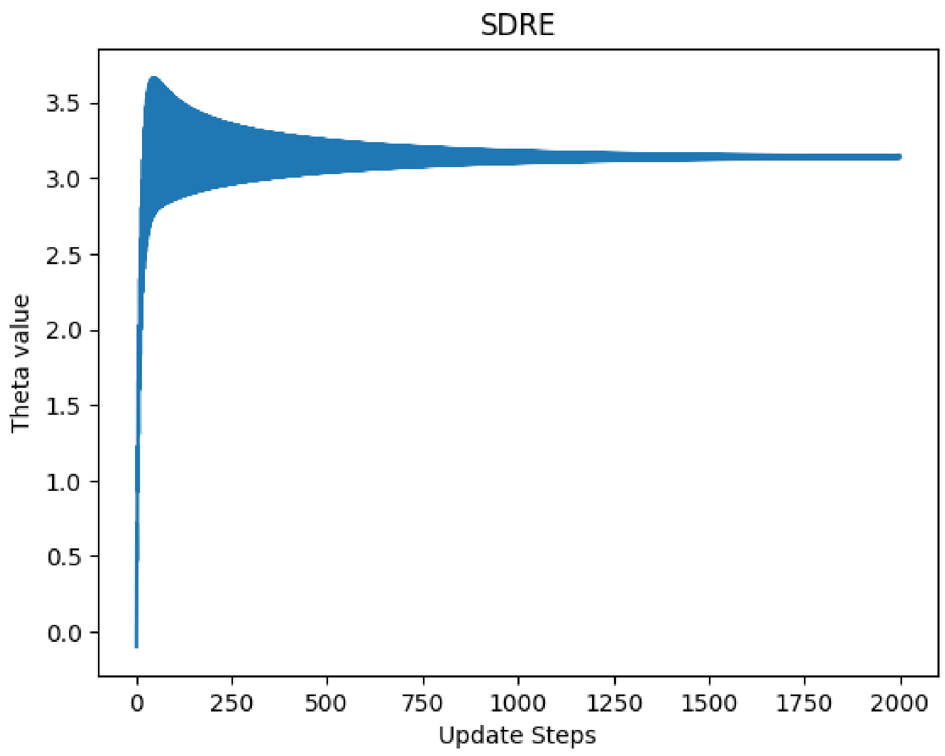
1) Theta = 0



2) Theta = 1



3) dt = 0.1



When the dt is large, we can see that the system requires a higher number of iterations to converge since a bigger time step leads to a higher level of discretization in the system.

Contribution

1. SDRE: Sudheer and Vithesh
2. DDPG: Sri Loukya and Pallavi
3. SAC: Pallavi and Sudheer
4. PPO: Vithesh and Sri Loukya