Alex Nguyen
ECE 594D

Alex Nguyen MATLAB Script Description:

This document gives a brief overview of each folder's contents and its purpose. There are three folders which I scripted for this project: DP value iteration pendulum, optimal policy, and pendulum animations.

1. DP Value Iteration Pendulum - Reference

   This folder includes multiple functions with a main script to find the optimal policy for a simple pendulum. This script was found on GitHub where I edited the script for better readability and organization. This is discussed more in the comments of the actual main MATLAB script. Also, I added an animation of the pendulum with a phase portrait and position displacement plot over time. This script does not include the energy pumping controller in the animation. The simple pendulum has a torque input act on the mass near the top at $\theta = \pi$ between $\pm\frac{\pi}{4}$ radians.

2. Pendulum Model Animations

   This folder includes code which simulates the dynamics of a simple pendulum with a controller and the dynamics of a double pendulum without a controller. There are two functions: pendyn (Equations of Motion for the simple pendulum) and sat (saturation function - approximation). There are four different .m files within the folder as well.

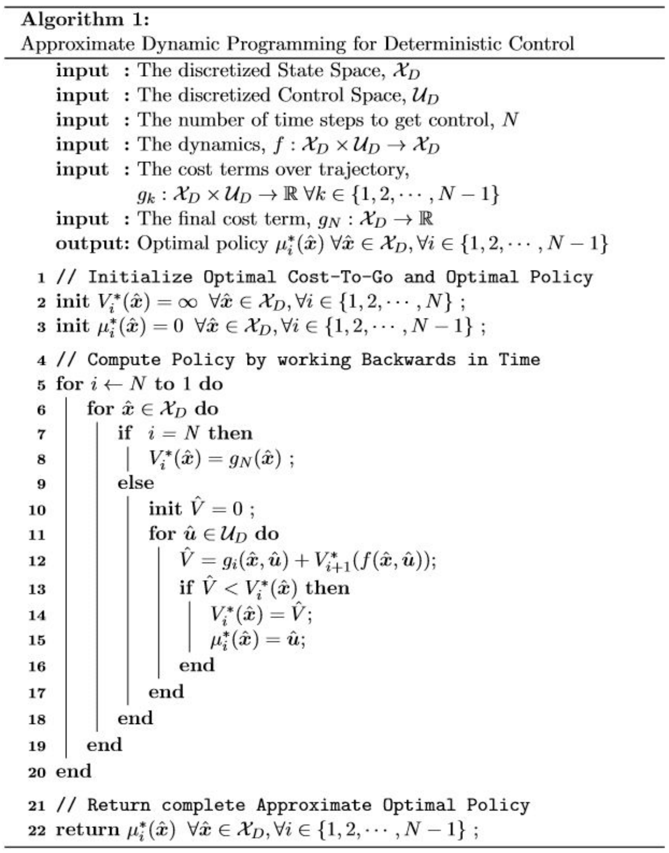
   a. *"pen_unlim_voltage.m"* assumes infinite power supply voltage so there is no limit on the torque. The stability of the pendulum is obtained using pole placement. I've included two different pole placements: only negative real poles (no oscillations) and negative real poles with complex conjugates (oscillations). One of these pole placements is commented out so the user can play around with the values and comment/uncomment which poles they would like to test.

   b. *"Pend_energy_shape_control.m"* is the torque limited case where the input value saturates which requires the pendulum to build up energy by swinging before switching controllers to either a LQR or pole placement approach. The initial conditions were set to be random to prove this model is robust to various conditions. The system starts with a controller which pumps energy into the system (due to limited voltage supply or "torque") then it switches to another controller which stabilizes the pendulum at the top at θ = π . This script is interactive so a user can specify which stability method they wish to use, either pole placement or LQR. There is an animation which shows the pendulum's full trajectory from the start using the two controllers in series.

c. "Pend_animate_no_controller.m" is a simulation of the pendulum dynamics with random initial conditions for position and velocity.

3. <u>Optimal Policy</u>

This is my version of the "DP Value Iteration Pendulum" folder. I attempted to use resources from Prof. Byl and the online web to create the optimal policy using value iteration. This was fairly difficult for me to write code for since I have not used dynamic programming (DP) or reinforcement learning (RL) before this project. My code will need to be debugged for an output like the "DP Value Iteration Pendulum - Reference' folder. There are two scripts from Prof. Byl included in the folder and two from me. One assumes the positional goal is at zero while the other assumes it to be at π . One of my scripts has commented out code which was my attempt to follow the following algorithm.



Algorithm 1:
Approximate Dynamic Programming for Deterministic Control

```
input  : The discretized State Space, $\mathcal{X}_D$
input  : The discretized Control Space, $\mathcal{U}_D$
input  : The number of time steps to get control, $N$
input  : The dynamics, $f : \mathcal{X}_D \times \mathcal{U}_D \to \mathcal{X}_D$
input  : The cost terms over trajectory,
         $g_k : \mathcal{X}_D \times \mathcal{U}_D \to \mathbb{R}\ \forall k \in \{1, 2, \cdots, N-1\}$
input  : The final cost term, $g_N : \mathcal{X}_D \to \mathbb{R}$
output: Optimal policy $\mu_i^*(\hat{x})\ \forall \hat{x} \in \mathcal{X}_D, \forall i \in \{1, 2, \cdots, N-1\}$

// Initialize Optimal Cost-To-Go and Optimal Policy
init $V_i^*(\hat{x}) = \infty\ \ \forall \hat{x} \in \mathcal{X}_D, \forall i \in \{1, 2, \cdots, N\}$ ;
init $\mu_i^*(\hat{x}) = 0\ \ \forall \hat{x} \in \mathcal{X}_D, \forall i \in \{1, 2, \cdots, N-1\}$ ;
// Compute Policy by working Backwards in Time
for $i \leftarrow N$ to 1 do
    for $\hat{x} \in \mathcal{X}_D$ do
        if $i = N$ then
            $V_i^*(\hat{x}) = g_N(\hat{x})$ ;
        else
            init $\hat{V} = 0$ ;
            for $\hat{u} \in \mathcal{U}_D$ do
                $\hat{V} = g_i(\hat{x}, \hat{u}) + V_{i+1}^*(f(\hat{x}, \hat{u}))$;
                if $\hat{V} < V_i^*(\hat{x})$ then
                    $V_i^*(\hat{x}) = \hat{V}$;
                    $\mu_i^*(\hat{x}) = \hat{u}$;
                end
            end
        end
    end
end
// Return complete Approximate Optimal Policy
return $\mu_i^*(\hat{x})\ \ \forall \hat{x} \in \mathcal{X}_D, \forall i \in \{1, 2, \cdots, N-1\}$ ;
```