ECE 276B Project 3
Infinite-Horizon Stochastic Optimal Control

Shiladitya Biswas
Dept. of Electrical and Computer Engineering
University of California,San Diego
California, USA

I. Introduction

Stochastic Optimal Control plays a major role in any robotic setup. Especially in control applications where the environment is very noisy. For example in a house hold setup the robot should be able to move from one location to another to perform several household duties without colliding with any obstacles. Hence, for a given environment (i.e. creating a Map of the environment by using techniques discussed in ECE276A) the robot's software should be able to generate optimal control sequences to go from one location to another. In optimal control planning our main objective is to find a feasible set of optimal control policies/sequence to minimize the total cost to reach from the initial state to the goal state. The cost here can be parameterized as a function control input and current state. In this project, we are given a inverted pendulum along the robots starting position and the goal position to reach to. We are asked to find the least cost /optimal path to go from the start position to the goal position. Infinite horizon Discounted optimal control in a type of Stochastic Optimal Control which takes into consideration the probability of state transition from one state to another and gives rewards on future cost by a number $0 < \gamma < 1$ to penalize the stage cost. The control scheme is very useful in noisy environments.

II. Problem Formulation

A. MDP Formulation

In order to formulate the MDP of this problem we first have to define or states. Here, we have 2 variables in the state i.e. Angle subtended by the pendulum with the vertical and the angular velocity of the pendulum at a given time, $X = [\theta, v]^T$, where θ is the angle and v is the angular velocity. The same applies to the control input as well 'U', lets call the control input as u. Since it is very challenging to work in continuous domain, we discreetize both the state space $[\theta, v]^T$ and control spaces into n_1,n_2 and n_u equally spaced regions,lets call them X_D and U_D. for this define the maximum and minimum values of angular velocity and control input i.e. v_{max} and u_{max} respectively. And we already know the angle state is naturally bounded between $-\pi$ to π. Hence total number of states possible in the state space are are $n_X = n_1 * n_2$ for every control input $u \in U$.

I would like to thank Saurabh Mirani and Ayon Biswas for the helpful discussions on the project.

Now, we form a 3D transition matrix P of size $n_X \times n_X \times n_u$ whose each cell value $P_{i,j,k}$ represents the Probability of going to state j from state i under the effect of control input k. In order to build this matrix we use the Euler discretization method with step size τ and look at the motion model of the inverted pendulum. lets call it $f(\mathbf{x}, u)$. For a given state $\mathbf{x} \in X_D$ and for all $u \in U_D$ the we find the mean of the probable next states using the equation:

$$\mathbf{x}_{next} = \mathbf{x} + f(\mathbf{x}, u) * \tau \tag{1}$$

We then fit a Gaussian on this $\mathbf{x}_{next}$ with covariance= $\sigma\sigma^T\tau$. Here $\sigma = [\sigma_1, \sigma_2]^T$, it controlls how much the brownian noise effects the pendulum states. The values obtained from this gaussian is then normalized and stored in the P matrix. This is repeated for all the states and all the controls belonging to the dicreetized state and control space respectively and the transition matrix P is built accordingly.

B. Optimization function

For this problem the stage cost given to us is as follows:

$$\ell(\mathbf{x}, u) = 1 - exp(k\cos(x_\theta)) - k) + \frac{r \times u^2}{2} \tag{2}$$

where $k, r > 0$.
Using this stage cost and the probability transition matrix built in the previous section, we can formulate this an infinite horizon shortest path discounted problem as follows:

$$V^*(\mathbf{x}) = \min_{u \in U_D} (\ell(\mathbf{x}, u) + \gamma \sum_{\mathbf{x'} \in X_D} P_{ij}(x'|x, u)V^*(x')) \tag{3}$$

Here $0 < \gamma < 1$ is the discount factor and the above equation is called the Bellman Equation. Now, Our goal is to minimize this value $V^*(\mathbf{x})$ over all the states $\mathbf{x} \in \mathrm{X}_D$ and accordingly keep track of all the control inputs $u \in U$ for each state $\mathbf{x}$. There exists several methods to solve this problem. These include:

1) Value Iteration
2) Policy Iteration

We discuss on the implementation aspect of these algorithms in the next section. Inorder to make my algo run quickly i vectorized my code. I calculated the stage cost of each state suject to each control input and stored them in a matrix L of size $(n_1 * n_2) \times n_u$.

C. Interpolation Problem

Until now we were working on a dicreetized model of the state space and accordingly extracting the optimal policies for each state of pendulum to control/balance the pendulum. This leads to jerky movement of the pendulum. Hence we have to again convert the discreetized optimal control policy obtained in section II-B into continuous domain. Thus interpolation is very necessary. I used Scipy library function interp1D to iterpolate both the control law and the angle subtended by the pendulum and then used it to generate the the the animation. The respective angle and control input values were also plotted.

III. Technical Approach

In this section, I have discussed my technical approach that I used in order to successfully implement the Value iteration and Policy iteration algorithms. The P matrix formulated in sec II-A

A. Interpolation

We use Euler discretisation to apply the discrete actions to the continuous-time/space system.Euler discretization with step τ leads to a transition model $p_f(x0|x,u)$ that is Gaussian with mean $x + f(x,u)R^2$ and covariance $^{R^{22}}$. To create transition probabilities in the discretized MDP, for each discrete state x and each discrete control u, I chose the next states/grid points around the mean $x + f(x,u)\tau$, evaluated the the Gaussian at the chosen grid points, and normalize so that the outgoing transition probabilities sum up to 1. Here $\sigma = [\sigma_1, \sigma_2]^T$, it controlls how much the brownian noise effects the pendulum states. The values obtained from this gaussian is then normalized and stored in the P matrix. This is repeated for all the states and all the controls belonging to the dicreetized state and control space respectively and the transition matrix P is built accordingly.

B. Value Iteration

Value Iteration applies the Dynamic Programming recursion with an arbitrary initialisation V_0 for all $x \in \mathcal{X}$:

$$V_{k+1}(x) = \min_{u \in \mathcal{U}(x)} [l(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x'|x,u) V_k(x')] \quad (4)$$

VI requires infinite iterations for $V_t(x)$ to converge to $V^\star(x)$ In this project we use Gauss-Seidel Value Iteration. Gauss-Seidel Value Iteration updates the values in place updates the values in place

$$V(x) = \min_{u \in \mathcal{U}(x)} [l(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x'|x,u) V(x')] \quad (5)$$

Gauss-Seidel VI often leads to faster convergence and requires less memory than VI.

1) Implementation: :

1) First an arbitrary value function vector V was defined of size $(n_1 * n_2) \times 1$.
2) Store V to a temporary variable temp.
3) Then I ran a for loop over all the control inputs and using the P matrix built in section II-A, I computed the total cost of transition from each state to the other state. Stored it in a matrix named H.

$$H[:,u] = L[:,u] + \gamma * P[:,:,u] * V \text{for all } u \in U_D \quad (6)$$

4) Then I extracted the optimal policy by taking row wise argmin of H matrix.
5) Then I updated the V vector by taking the row wise min of H matrix.
6) Then I check if the ℓ_2 norm of (V-temp) is less than a certain threshold. If not then jump to step 2 and repeat. If yes then we break out of the loop.

C. Policy Iteration

PI is an alternative algorithm to VI for computing $V^\star(x)$. PI iterates over policies instead of values. Policy iteration consists of the following two steps which are repeated until $V^\pi(x) = V^\pi(x)$ for all $x \in \mathcal{X}$:

1) Policy Evaluation: given a policy π, compute V^π

$$V(x) = l(x, \pi(x)) + \gamma \sum_{x' \in \mathcal{X}} p_f(x'|x, \pi(x)) V(x') \quad (7)$$

2) Policy Improvement: given V^π, obtain a new stationary policy π'

$$\pi' = _{u \in \mathcal{U}(x)} [l(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x'|x,u) V(x')] \quad (8)$$

1) Implementation: :

1) First an arbitrary value function vector V and policy vector P was defined of size $(n_1 * n_2) \times 1$.
2) Store V to a temporary variable temp.
3) First I evaluated the random policy defined above. For this I ran a loop over all the states and took the transition probabilities of the corresponding policies using the P matrix and stored in a new matrix PP. I then took the corresponding the policy costs and stored into a new array named LL.
4) Then I predicted the value functions of all the states using PP and LL.

$$V = ((I - \gamma * PP)^{-1}) * LL \quad (9)$$

5) Then I ran a for loop over all the control inputs and using the P matrix built in section II-A and V value from previous step, I computed the total cost of transition from each state to the other state. Stored it in a matrix named H.

$$H[:,u] = L[:,u] + \gamma * P[:,:,u] * V \text{for all } u \in U_D \quad (10)$$

6) Then I extracted the optimal policy by taking row wise argmin of H matrix.

7) Then I updated the V vector by taking the row wise min of H matrix.
8) Then I check if the ℓ_2 norm of (V-temp) is less than a certain threshold. If not then jump to step 2 and repeat. If yes then we break out of the loop.

IV. Results and Discussions

1) Variation of a: : Keeping other parameters constant, $\tau = 0.2 u_{max} = v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, k = 4, r = 0.01, b = 0.01$ and $\gamma = 0.9$ we study the performance by varying a. We observe that for small values of 'a' i.e. a= 1 etc, the controller is able to balance the pendulum. Value iteration takes 19 iterations while Policy iteration takes 8 iterations to generate the optimal value function. But for large values of a such has 10 etc it is unable to balance the pendulum. Inorder to balance a pendulum with high a value we need to increase the control input bound.

2) Variation of u: : Keeping other parameters constant, $\tau = 0.2 v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, k = 4, r = 0.01, a = 10, \gamma = 0.9$ and $b = 0.01$. If we keep $u_{max} = 3$ we see that it is unable to balance the pendulum. But with $u_{max} >= 6$ we were successfully able to balance the pendulum. This behaviour is expected, since inorder to balance a heavy pendulum we need to apply more torque at the rotation axis of the pendulum. However if we keep increasing the value of u_max we end up with an offset at the final position of the pendulum, i.e. the equilibrium point shifts a bit.

3) Variation of r: : Keeping other parameters constant, $\tau = 0.2, u_{max} = v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, k = 4, a = 1, \gamma = 0.9$ and $b = 0.01$. From the stage cost equation it can be seen that r value penalizes the control input. Hence higher the r value, higher is the penalty on the control input, as a result the controller tries to use small control values to reach the goal position i.e. balance the pendulum.

4) Variation of k: : Keeping other parameters constant, $\tau = 0.2, u_{max} = v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, a = 1, \gamma = 0.9$ and $b = 0.01$ Looking at the stage cost equation we see that the exponential expression containing k is constant for large values of K. But for low values of k we observe that the final equilibrium state is shifted from the upright position, since for low k value the exponential term varies a lot.

5) Variation of b: : Keeping other parameters constant, $\tau = 0.2, u_{max} = v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, a = 1, r = 0.01 \gamma = 0.9$ and $k = 4$ From the motion model of the pendulum we see that b is the damping constant.For $b = 1$ it is able to balance the pendulum. But for high b value we observe sever oscillations about the final state. This is because, high damping means the velocity quickly reduces and it is tough for the controller to reach the final state depending upon the initial state of the system. However for large $u_{max} = 10$ we are able to reduce oscillatory behaviour and reach the goal. This expected as more torque is required to overcome high damping.

6) Variation of τ: Keeping other parameters constant, $b = 0.01 u_{max} = v_{max} = 3, n_1 = 31, n_2 = 31, n_u = 31, a = 1, r = 0.01 \gamma = 0.9$ and $k = 4$ Although, it is inappropriate to change the τ value independently since it directly effects the noise covariance and the n_2 states. Thus no quantitive conclusion can be made about the controller w.r.t. τ. If n_2 is small VI/PI converges quickly else the opposite happens.

TABLE I
Convergence time comparison table

n_1	n_2	n_u	VI(Sec)	PI(Sec)
31	31	31	40.3	22.33
51	51	51	376.70	138.38
101	41	41	640.04	197.33

7) Variation of σ : : σ directly contributes to the motion noise. Hence a high value of σ will lead to high covariance of the Gaussian function we used to formulate the Transition probability matrix. Hence keeping all other values constant the controller finds it very difficult to balance the pendulum. However this can be overcome easily by by increasing the u_max value so that the noise would not effect the pendulum that much.

8) Variation of n_1, n_2 and n_u: Keeping other parameters constant, $\tau = 0.1 b = 0.01 u_{max} = v_{max} = 3, , a = 1, r = 0.01 \gamma = 0.9$ and $k = 4$. The time taken by VI and PI algorithm to converge is shown in table I. It is observed that as the state space discreetization increases the time taken for both PI and VI to converge increases. But as expected PI takes less time than VI.

An example plot for Policy and Value over all the states for $n_1 = 101, n_2 = 41 and n_u = 41$ is shown in figure 1,2 and3:

9) Variation with γ: It was observed that the controller wasn't able to balance the pendulum for any other value of γ other that 0.9. Although the algo took less time to converge for smaller γ, which is expected, since γ weights the future rewards. For small γ the future rewards are weighted less and for large γ the future rewards are given high weight hence more number of calculations are performed hence more time is taken by the algorithm to converge.

The Episode comparison of Value function is shown in figure 4

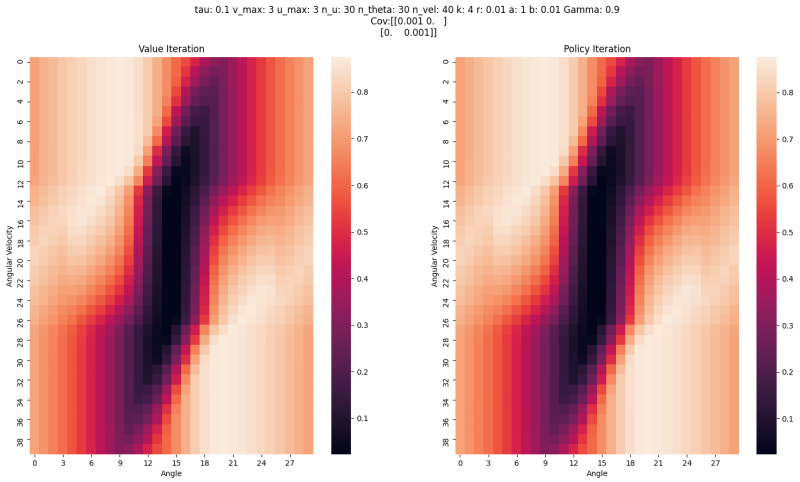

Fig. 1. Value function Plot

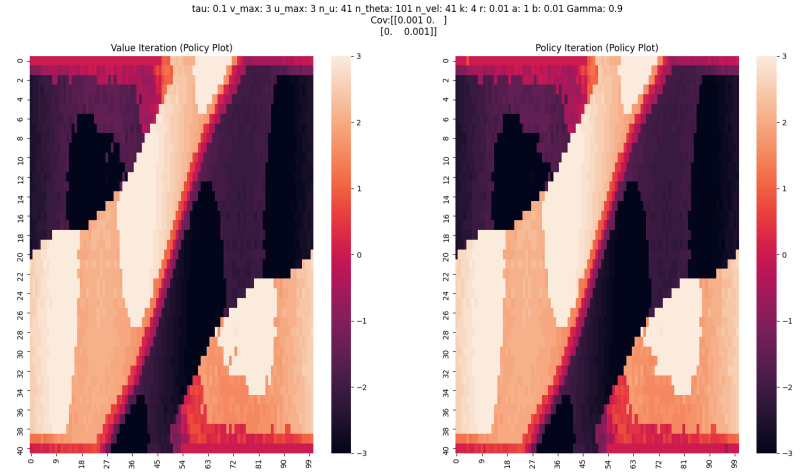

Fig. 2. Policy plot

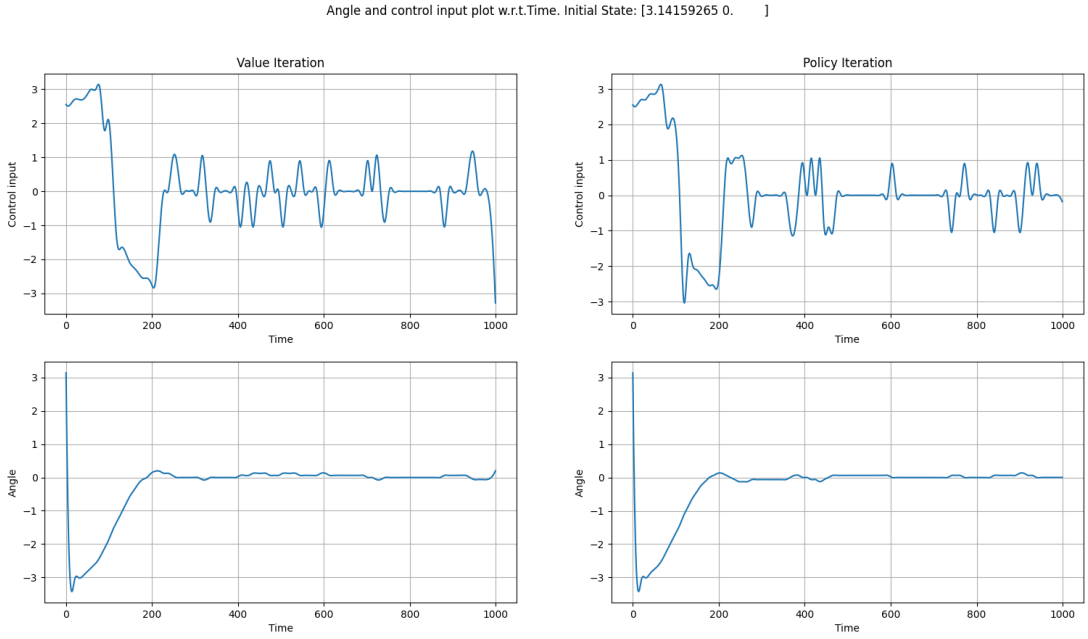

Fig. 3. Control input and θ plot

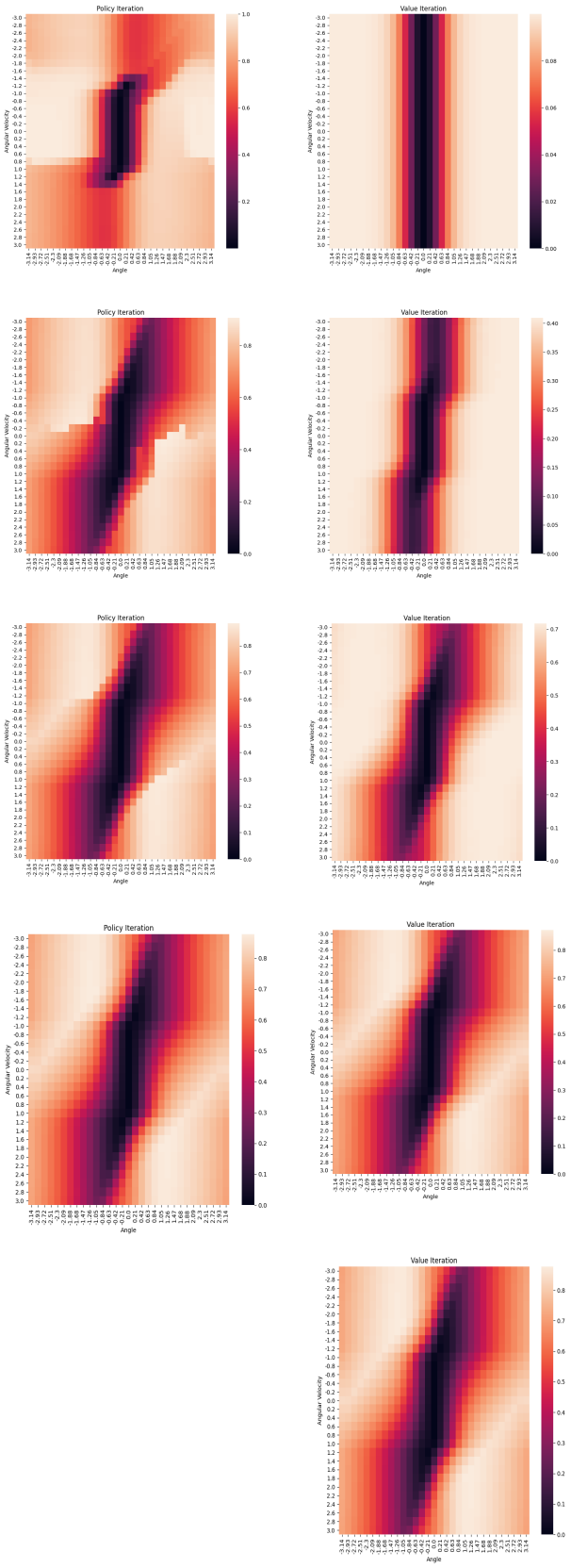

Fig. 4. Value function plot per episode for Policy(Left) and Value(Right) iteration