Spring 2020

Stanford
AA 203: Optimal and Learning-based Control
Problem set 2, due on April 27

Problem 1: In this question, we will dig into one of the first problems solved using Dynamic Programming [1]. For the first part of the question, we will examine the exact problem statement from [2].

We own two gold mines: Anaconda and Bonanza. We also have a sensitive gold-mining machine with the following characteristics:

- If the machine is used in Anaconda, it will mine, with probability p, a fixed fraction r of the gold there and be undamaged. With probability $1-p$, it will mine nothing and be permanently broken.
- If the machine is used in Bonanza, it will mine, with probability q, a fixed fraction s of the gold there and be undamaged. With probability $1-q$, it will mine nothing and be permanently broken.

At each stage, as long as the machine isn't broken, we can choose to mine Anaconda or Bonanza. Given initial amounts x_0 and y_0 in each mine, what sequence of choices maximizes the expected amount of gold mined?

Part a) We will first consider the infinite horizon case. Begin by writing the dynamic programming recursion for this case. Then, use the provided hint below to find the optimal policy.

Hint (Theorem 4, [1]): The solution of

$$f(x,y) = \max\{p_1(r_1 x + f(s_1 x, y)), p_2(r_2 y + f(x, s_2 y))\} \tag{1}$$

where $0 < p_1, 0 < p_2, s_1 < 1, s_2 < 1, 0 < r_1, 0 < r_2$, is given by

$$f(x,y) = \begin{cases} p_1(r_1 x + f(s_1 x, y)) & \text{for } \frac{p_1 r_1 x}{1-p_1} \geq \frac{p_2 r_2 y}{1-p_2} \\ p_2(r_2 x + f(x, s_2 y)) & \text{for } \frac{p_1 r_1 x}{1-p_1} \leq \frac{p_2 r_2 y}{1-p_2} \end{cases} \tag{2}$$

Part b) We will now consider the finite horizon case. Imagine that you only have two timesteps to mine as much gold as possible. Let $x_0 = 5, y_0 = 1, p = 0.5, r = 0.5, q = 0.75, s = 0.75$. Solve the DP recursion given these parameters to find the optimal value and the optimal policy.

Problem 2: In this question you will design a stabilizing controller to balance an inverted pendulum on a cart, which is a classic test problem in control. The system has state variables x, corresponding to the horizontal position of the cart, and θ, corresponding to the angle of the pendulum (where $\theta = 0$ corresponds to hanging straight down). The control moves the cart horizontally. We will write the combined state as $\boldsymbol{s} = [x, \theta, \dot{x}, \dot{\theta}]^T$ and the action as u. The continuous time dynamics of the system are given by

$$\dot{\boldsymbol{s}} = f(\boldsymbol{s}, u) = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \frac{u + m_p \sin\theta (l\dot{\theta}^2 + g\cos\theta)}{m_c + m_p \sin^2\theta} \\ \frac{-u\cos\theta - m_p l \dot{\theta}^2 \cos\theta \sin\theta - (m_c + m_p) g \sin\theta}{m_c + m_p \sin^2\theta} \end{bmatrix}. \tag{3}$$

We will discretize these dynamics via Euler integration and linearize around the upright position. Note that $f(\boldsymbol{s}^*, u^*) = 0$ for $\boldsymbol{s}^* = [0, \pi, 0, 0]^T$, $u^* = 0$, and thus this is a stationary point for this system. Linearizing around this point, we can write our discrete time dynamics

$$\delta \boldsymbol{s}_{k+1} = \underbrace{(I + dt \frac{\partial f}{\partial \boldsymbol{s}}(\boldsymbol{s}^*, u^*))}_{A} \delta \boldsymbol{s}_k + \underbrace{dt \frac{\partial f}{\partial u}(\boldsymbol{s}^*, u^*)}_{B} u_k \tag{4}$$

where dt is the timestep of our integration. In this expression,

$$\frac{\partial f}{\partial \boldsymbol{s}}(\boldsymbol{s}^*, u^*) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_1 & 0 & 0 \\ 0 & a_2 & 0 & 0 \end{bmatrix}, \quad \frac{\partial f}{\partial u}(\boldsymbol{s}^*, u^*) = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c} \\ \frac{1}{l m_c} \end{bmatrix}, \tag{5}$$

where

$$a_1 = \frac{m_p g}{m_c}, \quad a_2 = \frac{(m_c + m_p) g}{l m_c} \tag{6}$$

We will use A, B to design a stabilizing LQR controller. We will choose $Q = I, R = I$. In class, we saw that finite horizon LQR controllers could be designed which result in time-varying feedback gains. Here, we will design an infinite horizon LQR controller to minimize the cost

$$\sum_{k=0}^{\infty} \frac{1}{2} (\boldsymbol{s}_k - \boldsymbol{s}_k^*)^T Q (\boldsymbol{s}_k - \boldsymbol{s}_k^*) + \frac{1}{2} u_k^T R u_k. \tag{7}$$

After n iterations of the Riccati equation, the value matrices $\{P_k\}_{k=0}^n$ and the feedback gains $\{L_k\}_{k=0}^n$ gives the time-varying LQR controller which minimizes the quadratic cost over horizon n. We will write P_∞ and L_∞ for the value and feedback

gain of the infinite horizon LQR problem, respectively. If a linear time-invariant system (A, B) is controllable, the feedback gain matrix L_∞ will be constant for all time, and the value matrix P_∞ will be finite. Moreover, if the Riccati recursion is carried out repeatedly, P_n will asymptotically converge to P_∞ and L_n to L_∞. Therefore, a way to approximate P_∞ is to run the Riccati recursion a number of times large enough such that it has approximately converged. This is the approach we will take in this exercise.

(a) Fill in `lqr_infinite_horizon_solution.m`, first writing the A, B matrices, followed by writing the Riccati recursion. The parameter values are provided in that function. Use `simulate_p4.m` to simulate the system without disturbances.

(b) We will now simulate the system with noise to investigate disturbance rejection of the stabilizing controller. Turn on noise (by setting line 13 to true) and simulate the system again. Submit the plots for both simulations.

Problem 3: Suppose we have a machine that is either running or is broken down. If it runs throughout one week, it makes a gross profit of \$100. If it fails during the week,gross profit is zero. If it is running at the start of the week and we perform preventive maintenance, the probability that it will fail during the week is 0.4. If we do not perform such maintenance, the probability of failure is 0.7. However, maintenance will cost \$20. When the machine is broken down at the start of the week, it may either be repaired at a cost of \$40, in which case it will fail during the week with a probability of 0.4, or it may be replaced at a cost of \$150 by a new machine that is guaranteed to run through its first week of operation. Find the optimal repair,replacement, and maintenance policy that maximizes total profit over four weeks,assuming a new machine at the start of the first week.

Problem 4: In this problem, we will apply techniques for solving Markov Decision Processes to guide a quadrotor drone to its destination through a rainstorm. The world is represented as a $n \times n$ grid, meaning the state space is

$$\mathcal{S} := \{(x_1, x_2) : x_1, x_2 \in \{0, 1, ..., n-1\}\}.$$

In these coordinates, $(0, 0)$ represents the top left corner of the map and $(n-1, n-1)$ represents the bottom right corner of the map. From any location $x = (x_1, x_2) \in S$, the quadrotor has 4 possible directions $\mathcal{A} := \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$ it can move in, where

`up` - $(x_1, x_2) \mapsto (x_1 - 1, x_2)$.

`down` - $(x_1, x_2) \mapsto (x_1 + 1, x_2)$.

`left` - $(x_1, x_2) \mapsto (x_1, x_2 - 1)$.

`right` - $(x_1, x_2) \mapsto (x_1, x_2 + 1)$.

Additionally, there is a storm whose center is located at $\texttt{eye} \in \mathcal{S}$. The storm's influence is strongest at the center and decays farther from the center according to the equation $p(x) = \exp\left(-\frac{||x-\texttt{eye}||_2^2}{2\sigma^2}\right)$. Given its current state x and action a, the quadrotor's next state is determined as follows:

> With probability $p(x)$, the storm will cause the quandrotor to move in a uniformly random direction.
>
> With probability $1 - p(x)$, the quadrotor will move in the direction specified by the action.
>
> *Note:* If the resulting movement would cause the quadrotor to leave $\mathcal{S}$, then it will not move at all. For example, if the quadrotor is on the right boundary of the map, then moving right will do nothing.

Since the quadrotor's objective is to reach the goal, the reward function is given by $R(x) = \mathbb{I}_{[x=\texttt{goal}]}$. In other words, the quadrotor will receive a reward of 1 if it reaches the $\texttt{goal} \in \mathcal{S}$ and the reward for all other states is zero.

(a) For $n = 20$, $\sigma = 10$, $\texttt{eye} = (15, 15)$, and $\texttt{goal} = (19, 9)$ use value iteration to find an optimal value function for the quadrotor to navigate the storm. Submit a heatmap of the value function obtained by value iteration. You may use any programming language of your choice. A policy π is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ where $\pi(x)$ specifies the action to be taken should the quadrotor find itself in state x. Recall that value iteration repeats the following Bellman update until convergence:

$$V(x) \leftarrow \max_{a \in \mathcal{A}} \left(\sum_{x' \in \mathcal{S}} p(x'; x, a)(R(x') + V(x')) \right)$$

where $p(\cdot; x, a)$ is the pobabiility distribution of the next state when taking action a in state x, and R is the reward function.

(b) An optimal value function V^* induces an optimal policy π^* where

$$\pi^*(x) = \arg\max_{a \in \mathcal{A}} \left(\sum_{x' \in \mathcal{S}} p(x'; x, a)(R(x') + V(x')) \right)$$

Use the value function you computed in part (a) to compute an optimal policy. Then, use the policy starting from $\texttt{start} = (9, 19)$ to obtain a trajectory to the goal. Include a picture of the trajectory and briefly describe in words what the policy is doing.

Problem 5: Next, we will implement a controller to solve the more challenging "swing up" problem, in which the pendulum starts facing downwards and is brought to an upright position. For this problem, it is not sufficient to linearize around a single

stationary point, and so we will turn to iterative LQR. Our dynamics will take the form

$$\delta \boldsymbol{s}_{k+1} = \underbrace{(I + dt\frac{\partial f}{\partial \boldsymbol{s}}(\bar{\boldsymbol{s}}_k, \bar{u}_k))}_{A_k}\delta \boldsymbol{s}_k + \underbrace{dt\frac{\partial f}{\partial u}(\bar{\boldsymbol{s}}_k, \bar{u}_k)}_{B_k}\delta u_k. \tag{8}$$

Evaluating the derivatives of the dynamics can be unwieldy, so we have included a function to perform this linearization and return A_k, B_k. To implement the controller, you will fill in `ilqr_solution.m`. We will use the cost function

$$\frac{1}{2}(\boldsymbol{s}_N - \boldsymbol{s}^*)^T Q_N(\boldsymbol{s}_N - \boldsymbol{s}^*) + \sum_{k=0}^{N-1}\left(\frac{1}{2}(\boldsymbol{s}_k - \boldsymbol{s}^*)^T Q(\boldsymbol{s}_k - \boldsymbol{s}^*) + \frac{1}{2}u_k^T R u_k\right) \tag{9}$$

with Q_N large to enforce a soft terminal constraint.

(a) Rewrite the cost function in terms of deviations $\delta \boldsymbol{s}_k, \delta \boldsymbol{s}_N, \delta u_k$. This will result in a cost function with additional linear terms. Fill in these linear terms in lines 40, 51, 52 of `ilqr_solution.m`.

(b) The backward pass of iLQR consists of evaluating the Riccati recursion. Write the function `backward_riccati_recursion` to return the controller term l, L and the value terms p, P.

(c) Finally, we will implement the forward pass. Implement the nominal control update on line 63.

Run `simulate_p5.m` to run the simulations after you complete the controller. This script first performs the swing up maneuver generated by the iLQR controller, and then switches to the stabilizing infinite horizon LQR controller from the previous question. It will generate plots of the state and control trajectories. First, run the simulations without noise (the default), and observe the response. Then, turn on noise (set line 13 of `simulate_p5.m` to true) and run the simulation with noise. Submit the plots for simulations both with and without noise.

Learning goals for this problem set:

Problem 1: To applying dynamic programming and learn about its historical use cases.

Problem 2: To gain experience with implementing LQR controllers.

Problem 3: To apply dynamic programming in stochastic environments by reasoning about expected utilities.

Problem 4: To solve a stochastic optimization problem with value iteration by formulating it as a MDP.

Problem 5: To gain experience with implementing iterative LQR controllers.

References

[1] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 1952.

[2] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 1954.