Spring 2022

Stanford
AA 203: Optimal and Learning-based Control
Homework #2, due May 2 by 11:59 pm

Problem 1: Shortest path through a grid

Consider the shortest path problem in Figure 1 where it is only possible to travel to the right and the numbers represent the travel times for each segment. The control input is the decision to go "up-right" or "down-right" at each node.

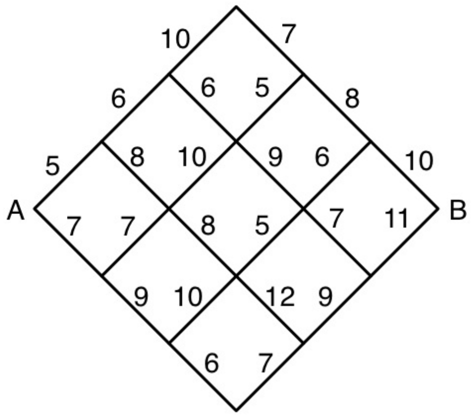

Figure 1: Shortest path problem on a grid.

(a) Use Dynamic Programming (DP) to find the shortest path from A to B.

(b) Consider a generalized version of the shortest path problem in Figure 1 where the grid has n segments on each side. Find the number of computations required by an exhaustive search algorithm (i.e., the number of routes that such an algorithm would need to evaluate) and the number of computations required by a DP algorithm (i.e., the number of DP evaluations). For example, for $n = 3$ as in Figure 1, an exhaustive search algorithm requires 20 computations, while the DP algorithm requires only 15.

Problem 2: Machine maintenance

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 3: Markovian drone

In this problem, we will apply techniques for solving a Markov Decision Process (MDP) to guide a flying drone to its destination through a storm. The world is represented as an $n \times n$ grid, i.e., the state space is

$$\mathcal{S} := \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1, x_2 \in \{0, 1 \ldots, n-1\}\}.$$

In these coordinates, $(0,0)$ represents the bottom left corner of the map and $(n-1, n-1)$ represents the top right corner of the map. From any location $x = (x_1, x_2) \in \mathcal{S}$, the drone has four possible directions it can move in, i.e.,

$$\mathcal{A} := \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}.$$

The corresponding state changes for each action are:

- `up`: $(x_1, x_2) \mapsto (x_1, x_2 + 1)$
- `down`: $(x_1, x_2) \mapsto (x_1, x_2 - 1)$
- `left`: $(x_1, x_2) \mapsto (x_1 - 1, x_2)$
- `right`: $(x_1, x_2) \mapsto (x_1 + 1, x_2)$

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

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

The quadrotor's objective is to reach $x_{\text{goal}} \in \mathcal{S}$, so the reward function is the indicator function $R(x) = I_{x_{\text{goal}}}(x)$. In other words, the drone will receive a reward of 1 if it reaches the $x_{\text{goal}} \in \mathcal{S}$, and a reward of 0 otherwise. The reward of a trajectory in this infinite horizon problem is a discounted sum of the rewards earned in each timestep, with discount factor $\gamma \in (0,1)$.

(a) Given $n = 20$, $\sigma = 10$, $\gamma = 0.95$, $x_{\text{eye}} = (15, 15)$, and $x_{\text{goal}} = (19, 9)$, write code that uses value iteration to find the optimal value function for the drone to navigate the storm. Recall that value iteration repeats the Bellman update

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

until convergence, where $p(x'; x, a)$ is the probability distribution of the next state being x' after taking action a in state x, and R is the reward function. Plot a heatmap of the optimal value function obtained by value iteration over the grid $\mathcal{S}$, with $x = (0,0)$ in the bottom left corner, $x = (n-1, n-1)$ in the top right corner, the x_1-axis along the bottom edge, and the x_2-axis along the left edge.

(b) Recall that a policy π is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ where $\pi(x)$ specifies the action to be taken should the drone find itself in state x. An optimal value function V^* induces an optimal policy π^* such that

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

Use the value function you computed in part (a) to compute an optimal policy. Then, use this policy to simulate the MDP starting from over $N = 100$ time steps starting at $x = (0, 19)$. Plot the policy as a heatmap where the actions $\{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$ correspond to the values $\{0, 1, 2, 3\}$, respectively. Plot the simulated drone trajectory overlaid on the policy heatmap, and briefly describe in words what the policy is doing.

Problem 4: Cart-pole balance

In this problem, we will design a controller to balance an inverted pendulum on a cart, i.e., the classic "cart-pole" benchmark. This system has two degrees of freedom corresponding to the horizontal position x of the cart, and the angle θ of the pendulum (where $\theta = 0$ occurs when the pendulum is handing straight down). We can apply a force $u \in \mathbb{R}$ to push the cart horizontally, where $u > 0$ corresponds to a force in the positive x-direction. With the state $s := (x, \theta, \dot{x}, \dot{\theta}) \in \mathbb{R}^4$, we can write the continuous-time dynamics of the cart-pole system as

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

where m_p is the mass of the pendulum, m_c is the mass of the cart, ℓ is the length of the pendulum, and g is the acceleration due to gravity. We can discretize the continuous-time dynamics using Euler integration with a fixed time step Δt to get the approximate discrete-time dynamics

$$s_{k+1} \approx s_k + \Delta t f(s_k, u_k),$$

where s_k and u_k are the state and control input, respectively, at time $t = k\Delta t$.

(a) Consider the upright state $s^* := (0, \pi, 0, 0)$ with $u^* := 0$, and define $\Delta s_k := s_k - s^*$. Linearizing the approximate discrete-time dynamics $s_{k+1} \approx s_k + \Delta t f(s_k, u_k)$ about (s^*, u^*) yields an approximate LTI system of the form

$$\Delta s_{k+1} \approx A\Delta s_k + Bu_k.$$

Express A and B in terms of m_p, m_c, ℓ, g, and Δt. You may use the fact that

$$\frac{\partial f}{\partial s}(s^*, u^*) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{m_p g}{m_c} & 0 & 0 \\ 0 & \frac{(m_c+m_p)g}{m_c\ell} & 0 & 0 \end{bmatrix}, \quad \frac{\partial f}{\partial u}(s^*, u^*) = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c} \\ \frac{1}{m_c\ell} \end{bmatrix}.$$

We will design a stabilizing LQR controller for this discrete-time LTI system to solve

$$\begin{aligned} \underset{\{u_k\}_{k=0}^{\infty}}{\text{minimize}} \quad & \sum_{k=0}^{\infty} \left(\frac{1}{2}\Delta s_k^\mathsf{T} Q \Delta s_k + \frac{1}{2} u_k^\mathsf{T} R u_k \right) \\ \text{subject to} \quad & \Delta s_{k+1} = A\Delta s_k + Bu_k, \ \forall k \in \mathbb{N}_{\geq 0} \end{aligned},$$

for fixed $Q, R \succ 0$. Recall that after N iterations of the discrete-time Riccati recursion

$$\begin{aligned} K_k &= -(R + B^\mathsf{T} P_{k+1} B)^{-1} B^\mathsf{T} P_{k+1} A \\ P_k &= Q + A^\mathsf{T} P_{k+1}(A + BK_k) \end{aligned},$$

the cost-to-go matrices $\{P_k\}_{k=0}^{N}$ and the time-varying feedback gains $\{K_k\}_{k=0}^{N-1}$ describe the optimal LQR controller for a finite-horizon version of the problem above. If (A, B) is stabilizable, then these iterates asymptotically converge to some $P_\infty \succ 0$ and K_∞, such that $(s_0 - s^*)^\mathsf{T} P_\infty (s_0 - s^*) > 0$ is the finite optimal cost-to-go for any initialization s_0, and $u_k = K_\infty \Delta s_k$ is the *time-invariant* feedback law for the corresponding optimal LQR controller[1].

(b) Write code to approximate P_∞ and K_∞ for the linearized, discretized cart-pole system by initializing $P_\infty = 0$ and then applying the Ricatti recursion until convergence with respect to the maximum element-wise norm condition $\|P_{k+1} - P_k\|_{\max} < 10^{-4}$. Use $m_p = 2$ kg, $m_c = 10$ kg, $\ell = 1$ m, $g = 9.81$ m/s^2, $\Delta t = 0.1$ s, $Q = I_4$, and $R = I_1$. Report the value of K_∞ up to two decimal places for each entry.

(c) Write code to simulate the continuous-time, nonlinear cart-pole system with the linear feedback controller $u = K_\infty \Delta s$. Initialize the system at $s = (0, 3\pi/4, 0, 0)$, and use a controller sampling rate of 10 Hz. Plot each state variable over time on separate plots for $t \in [0, 30]$. For your own interest, we provide the function `animate_cartpole` in `animations.py` to create a video animation of the cart-pole over time[2].

Hint: Write a function `ds = cartpole(s,t,u)` that computes the state derivative `ds` for the continuous-time, nonlinear cart-pole dynamics. To simulate the cart-pole with the fixed control input `u[k]` from state `s[k]` at time `t[k]` to state `s[k+1]` at time `t[k+1]`, you can use the following Python code:

```
from scipy.integrate import odeint
s[k+1] = odeint(cartpole, s[k], t[k:k+2], (u[k],))[1]
```

Make sure to review the documentation for `odeint`.

(d) To investigate the disturbance rejection ability of the controller, add noise to the system dynamics. Specifically, after each controller sampling period (i.e., every 0.1 s), sample a new noise vector $w \in \mathbb{R}^4$ from the Gaussian distribution with mean $\mu = 0$ and covariance $\Sigma = \mathrm{diag}(0, 0, 10^{-3}, 10^{-3})$, and add it to the state. Simulate the noisy system and plot each state variable over time for $t \in [0, 30]$.

(e) We will now use an LQR controller to track a time-varying trajectory. Specifically, we will aim to balance the pendulum upright (i.e., $\theta^*(t) \equiv \pi$) while oscillating the position of the cart to track a desired reference $x^*(t) = a\sin(2\pi t/T)$, where $a > 0$ and $T > 0$ are known constants.

   i. Normally, as derived in class when applying LQR for trajectory tracking, you would have to re-linearize the system around the desired trajectory at each time step; why is this not the case for this particular problem (i.e., why can you just reuse A and B)?

[1] The infinite-horizon LQR problem also converges for fixed $Q \succeq 0$ and $R \succ 0$, as long as (A, B) is stabilizable and (A, Q) is observable.

[2] See `https://github.com/StanfordASL/AA203-Homework`.

ii. Repeat part (c) (i.e., without noise) for this case with $a = 10$ and $T = 30$, except this time initialize the system upright at $s(0) = (0, \pi, 0, 0)$. For each state plot, overlay the corresponding entry from the reference trajectory $s^*(t)$.

iii. You may notice that this controller does not have good tracking performance. You could try increasing the state penalty matrix Q to, e.g., $Q = 10I_4$. However, this should only improve tracking for x and $\dot{x}$, while $\theta(t)$ and $\dot{\theta}(t)$ still oscillate around zero. What physical characteristic of the desired trajectory (or lack thereof) causes this to happen?

Learning goals for this problem set:

Problem 1: To familiarize with the DP algorithm and to appreciate the computational savings of DP versus an exhaustive search algorithm.

Problem 2: To apply dynamic programming in stochastic environments by reasoning about expected utilities.

Problem 3: To solve a stochastic optimization problem with value iteration by formulating it as an MDP.

Problem 4: To gain experience with implementing LQR controllers by coding them "from scratch".