AA 203: Optimal and Learning-based Control
Homework #2
Due May 5 by 11:59 pm

Learning goals for this problem set:

Problem 1: Study sequential convex programming in nonlinear trajectory optimization and gain experience with CVXPY.

Problem 2: Familiarize the DP algorithm and appreciate the computational savings of DP versus an exhaustive search algorithm.

Problem 3: Gain experience with implementing LQR controllers by coding them "from scratch".

Problem 4: Apply iLQR for nonlinear trajectory optimization and stabilization, while using JAX for dynamics linearization.

Problem 5: Apply stochastic dynamic programming in stochastic environments by reasoning about expected utilities.

2.1 Cart-pole swing-up with limited actuation. Throughout this problem set, we will consider three instantiations of the classic "cart-pole" benchmark in which we attempt to balance an inverted pendulum upright on a cart. In this problem, we will tackle the challenging cart-pole "swing up" problem, in which the pendulum begins hanging downwards and is then brought to the upright position, with constraints on the control input that moves the cart horizontally. In practice, such control constraints often arise from motor limitations. The transcribed optimal control problem we would like to solve is

$$\begin{aligned} \underset{s,u}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \Big((s_k - s_{\text{goal}})^\mathsf{T} Q (s_k - s_{\text{goal}}) + u_k^\mathsf{T} R u_k \Big) \\ \text{subject to} \quad & s_0 = \bar{s}_0 \\ & s_N = s_{\text{goal}} \\ & s_{k+1} = f_\mathsf{d}(s_k, u_k), \quad \forall k \in \{0, 1, \ldots, N-1\} \\ & u_k \in \mathcal{U}, \qquad\qquad \forall k \in \{0, 1, \ldots, N-1\} \end{aligned},$$

where $\bar{s}_0 \in \mathbb{R}^n$ is the initial state, $s_{\text{goal}} = (0, \pi, 0, 0)$ is the goal state (i.e., the upright position), $Q \succ 0$ and $R \succ 0$ are stage cost matrices, $\mathcal{U}$ is the control constraint set, and $f_\mathsf{d} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a discretized form of the cart-pole dynamics.

However, this optimal control problem is non-convex due to the nonlinear equality constraints $s_{k+1} = f_\mathsf{d}(s_k, u_k)$; even if the dynamics were linear, this optimization cannot be solved with Riccati recursion due to the constraints on actuation. We will circumvent this issue with sequential convex programming (SCP). Recall that the key idea of SCP is to iteratively linearize the dynamics around an estimate $(s^{(i)}, u^{(i)})$ and solve a convex approximation of the optimal control problem near this trajectory, at each iteration i. Since linearization provides a good approximation to the nonlinear dynamics only in a small neighborhood around $(s^{(i)}, u^{(i)})$, the accuracy of the convex model may be poor if (s, u) deviates far from $(s^{(i)}, u^{(i)})$. To ensure smooth convergence, we consider a convex trust region around the state and input that is imposed as an additional constraint in the convex

optimization problem. Specifically, we consider a box around the nominal trajectory $(s^{(i)}, u^{(i)})$ described by

$$\|s_k - s_k^{(i)}\|_\infty \leq \rho, \ \forall k \in \{0, 1, \ldots, N\}$$
$$\|u_k - u_k^{(i)}\|_\infty \leq \rho, \ \forall k \in \{0, 1, \ldots, N-1\}$$

for some constant $\rho > 0$. Moreover, the terminal constraint $s_N = s_{\text{goal}}$ is difficult to enforce during each iteration of SCP; any sub-problem for which the trust region does not contain a swing-up maneuver will be infeasible. Thus, we will remove the terminal constraint $s_N = s_{\text{goal}}$ and add the terminal state cost term $(s_N - s_{\text{goal}})^\mathsf{T} P (s_N - s_{\text{goal}})$ as a proxy, where $P \succ 0$ has large entries.

In `cartpole_swingup_constrained.py`, you will use JAX and CVXPY to construct and solve a convex sub-problem at each SCP iteration. Carefully read *all* of the provided code.

(a) Derive the convex approximation of the optimal control problem around an iterate $(s^{(i)}, u^{(i)})$, including the control constraint set $\mathcal{U} \coloneqq [-u_{\max}, u_{\max}]$ with $u_{\max} > 0$, trust region constraints, and the terminal state cost term in place of the terminal state constraint. The convex problem should have linear equality constraints of the form $s_{k+1} = A_k s_k + B_k u_k + c_k$ obtained by linearizing the dynamics around $(s^{(i)}, u^{(i)})$. Derive A_k, B_k, and c_k in terms of $s^{(i)}$, $u^{(i)}$, and $f_\mathtt{d}$.

(b) Use JAX to complete the function `affinize` such that it computes A_k, B_k, and c_k from $s^{(i)}$, $u^{(i)}$, and $f_\mathtt{d}$ in two lines of code.

(c) Complete the function `scp_iteration`. Specifically, given the current iterate $(s^{(i)}, u^{(i)})$, use CVXPY to specify and solve the convex optimization problem you derived to obtain an updated solution $(s^{(i+1)}, u^{(i+1)})$.

(d) Run `cartpole_swingup_constrained.py`. Submit the generated state, control, and SCP cost plots. You should notice that the pendulum just reaches the goal state before the simulation ends; it would be more satisfying to swing the pendulum upright and then keep it there. In words, suggest a control scheme to do this.

Submit all of the requested plots and your completed version of `cartpole_swingup_constrained.py`.

2.2 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" or "down" at each node. Given that state transitions are deterministic in a low-dimensional state space, we will use dynamic programming to develop a closed-loop policy for each node in the grid.

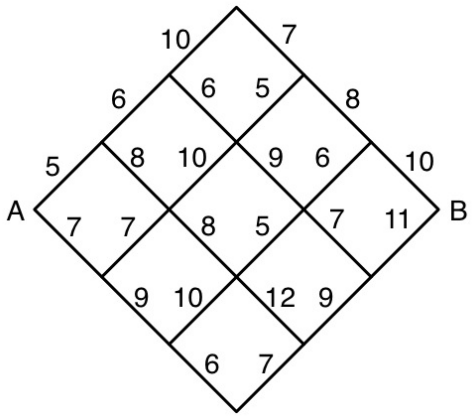

Figure 1: Shortest path problem on a grid.

(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.

2.3 Cart-pole balance. In this problem, we consider a new cart-pole formulation in which we will design a controller to balance the inverted pendulum by linearizing the dynamics around a single stationary point. Therefore, unlike the cart-pole swing-up problem, we will consider a trajectory initialized in a neighborhood about the final, upright state and the optimal control is a closed-form solution derived through Riccati recursion. 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 hanging straight downwards). 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 $s \coloneqq (x, \theta, \dot{x}, \dot{\theta}) \in \mathbb{R}^4$, 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 $\bar{s} \coloneqq (0, \pi, 0, 0)$ with $\bar{u} \coloneqq 0$, and define $\tilde{s}_k \coloneqq s_k - \bar{s}$. Linearizing the approximate discrete-time dynamics $s_{k+1} \approx s_k + \Delta t f(s_k, u_k)$ about $(\bar{s}, \bar{u})$ yields an approximate LTI system of the form

$$\tilde{s}_{k+1} \approx A\tilde{s}_k + Bu_k.$$

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

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

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

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

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_∞. In fact, $J^*(s_0) \coloneqq (s_0 - \bar{s})^\mathsf{T} P_\infty (s_0 - \bar{s})$ is the *finite* optimal cost-to-go for any initialization s_0, and $u_k = K_\infty \tilde{s}_k$ is the optimal feedback policy, which happens to be linear and *time-invariant*[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 $\|P_k - P_{k-1}\|_{\max} < 10^{-4}$. 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_∞ with two decimal places for each entry.

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

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

i. Normally, as derived in class when applying LQR for trajectory tracking to nonlinear systems, 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)?

ii. Repeat part (c) for this case with $a = 10$ and $T = 10$, 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 $\bar{s}(t)$.

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., $Q = 10 I_4$. However, this should only improve tracking for $x(t)$ and $\dot{x}(t)$, while $\theta(t)$ and $\dot{\theta}(t)$ still oscillate around π and 0, respectively. What physical characteristic of the desired trajectory (or lack thereof) causes this to happen?

Submit all of the requested plots and your complete code.

[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 detectable.

2.4 Cart-pole swing-up. In this problem, we will implement a controller to solve the cart-pole "swing up" problem without limited actuation. Recall that in the swing-up problem, the pendulum begins hanging downwards and is then brought to the upright position. Unlike in the cart-pole balancing problem, it is no longer sufficient to linearize around a single stationary point. Therefore, we will formulate the optimal control problem into a convex sub-problem and iteratively solve for optimal perturbations from a reference trajectory. Unlike the case with limited actuation, the solution to each convex sub-problem can be solved with Riccati recursion, i.e., we will iteratively develop our solution using iterative LQR (iLQR) control. In this case iLQR control provides an elegant and easy to implement closed-loop policy. However, through the standard LQR formulation at each iteration, we generally cannot reason, at least directly, over constraints on the state or control space: in the case of limited actuation, we require the more flexible and sophisticated framework provided by SCP. We provide starter code in `cartpole_swingup.py`.

Recall that the cart-pole is a continuous-time system with dynamics of the form $\dot{s} = f(s, u)$. To compute the iLQR control law, we will consider the Euler discretized dynamics

$$s_{k+1} \approx f_{\mathsf{d}}(s_k, u_k) \coloneqq s_k + \Delta t f(s_k, u_k) \tag{1}$$

with time step $\Delta t > 0$. The provided code will then simulate this control law on the original continuous-time system.

(a) For a given operating point $(\bar{s}_k, \bar{u}_k)$, suppose we define the Jacobians

$$A_k \coloneqq \frac{\partial f_{\mathsf{d}}}{\partial s}(\bar{s}_k, \bar{u}_k), \quad B_k \coloneqq \frac{\partial f_{\mathsf{d}}}{\partial u}(\bar{s}_k, \bar{u}_k). \tag{2}$$

Use JAX in the function `linearize` to write a single line of code that computes A_k and B_k, given f_{d}, $\bar{s}_k$ and $\bar{u}_k$.

For our iLQR controller, we will use the quadratic cost function

$$J(s, u) \coloneqq \frac{1}{2}(s_N - s_{\text{goal}})^{\mathsf{T}} Q_N (s_N - s_{\text{goal}}) + \frac{1}{2}\sum_{k=0}^{N-1}\Big((s_k - s_{\text{goal}})^{\mathsf{T}} Q (s_k - s_{\text{goal}}) + u_k^{\mathsf{T}} R u_k\Big), \tag{3}$$

where s_{goal} is the goal state (i.e., the upright position). The entries of $Q_N \succ 0$ are chosen to be large so that the terminal cost acts as a soft terminal "constraint".

(b) Rewrite the cost function in terms of the deviations $\tilde{s}_N \coloneqq s_N - \bar{s}_N$, $\tilde{s}_k \coloneqq s_k - \bar{s}_k$, and $\tilde{u}_k \coloneqq u_k - \bar{u}_k$. This will result in a quadratic cost function with linear terms of the form $q_N^{\mathsf{T}}\tilde{s}_N$, $q_k^{\mathsf{T}}\tilde{s}_k$, and $r_k^{\mathsf{T}}\tilde{u}_k$; identify the vectors q_N, q_k, and r_k.

(c) Use NumPy to complete the iLQR controller code in the delineated section of the function `ilqr`. Specifically, your code must update the controller gain and offset terms $\{Y_k\}_{k=0}^{N-1}$ and $\{y_k\}_{k=0}^{N-1}$, respectively[2], the nominal trajectory $(\bar{s}, \bar{u})$, and the deviations $(\tilde{s}, \tilde{u})$.

(d) Towards the end of `cartpole_swingup.py`, fill in the delineated section of the code to either apply the open-loop iLQR control input or the closed-loop iLQR policy, depending on the value of the Boolean flag `closed_loop`. Run your code for both cases to simulate the system and generate plots of the state and control input over time. You should notice that the iLQR control sequence does not accomplish the task if applied open-loop. Submit both sets of plots (i.e., you should have 10 plots in total).

Submit all of the requested plots and your completed version of `cartpole_swingup.py`.

[2]In lecture, we labelled these terms as K_t and k_t, but here we try to avoid confusion with the discrete index k.

2.5 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 (that is guaranteed to run during the first week of operation).