Spring 2021

Stanford
AA 203: Optimal and Learning-based Control
Problem Set 2, due May 3 by 5:00 pm

Problem 1: Introduction to Q-Learning

You have just purchased *Widget Co.*, a small shop selling widgets. Congratulations! Widget Co. is in the business of buying widgets wholesale, and selling them to consumers at a markup. While the previous owners were losing money, you suspect that you can apply the material that you learned in *AA203* to turn the business around.

The shop is able to store between 0 and 5 widgets at a time. We will write the number of widgets held in the shop on day k as x_k. Every day, you choose how many widgets to order from your supplier. You can order either zero widgets, a "half order" of 2 widgets, or a "full order" of 4 widgets. We write the number of widgets ordered to arrive on day k as u_k. A random number of customers (following an unknown distribution, though this distribution may be assumed to be consistent across all days) come in to Widget Co. every day, and buy a widget if there are any available. We write the demand on day k as d_k, and assume $d_k \leq 5$. As such, the dynamics are

$$x_{k+1} = \min(5, \max(x_k + u_k - d_k, 0)) \tag{1}$$

where d_k is sampled i.i.d. from $p(d)$.

Your costs are as follows. You must pay a fixed rent on your shop, $r = 1$. You also pay a cost $s = 0.05$ to store each widget overnight. You pay $b(u) = \sqrt{u}$ to order from your supplier. Finally, you earn $c = 1.2$ for each widget you sell. Thus, the total profit each day is

$$r(x_k, u_k, d_k) = c \min(d_k, x_k + u_k) - r - s x_k - b(u_k) \tag{2}$$

where $\min(d_k, x_k + u_k)$ is the "satisfied" demand.

a) The previous owners were nice enough to give you the details of their last three years of operation. In particular, they have provided a dataset of the last three years of (x_k, u_k, r_k, x_{k+1}) data. Write a Q-learning algorithm to learn an optimal policy π^* from this dataset. The starter code contains a function to generate the dataset. Submit a plot of Q-values for each state-action pair versus the number of iterations (please group these plots by state–i.e. the Q values for each action should all appear on the same plot for each state, and thus you should have six total plots). Use a discount factor of 0.95.

Once π^* has been computed, simulate this policy for five years (5∗365 days/iterations). Plot the total (aggregate) profit earned in the period from the beginning of the simulation to each day over the five year period. See `problem2_starter.py` and `problem2_env.py` for starter code.

b) Write code to do value iteration, using the knowledge that the demand model is:

$$d_k = \begin{cases} 0 & \text{with probability} \quad 0.1 \\ 1 & \text{with probability} \quad 0.3 \\ 2 & \text{with probability} \quad 0.3 \\ 3 & \text{with probability} \quad 0.2 \\ 4 & \text{with probability} \quad 0.1 \end{cases} \tag{3}$$

From this, compute the same Q values estimated in part (a). Again, use a discount factor of 0.95. Compare the values computed by value iteration to those estimated via Q-learning. What do you notice about the differences between the Q values computed by each method?

c) Do you think Q learning is the best approach to this problem? What information from the problem setting have we used in our approach? What information have we not used?

Hint: Your Q-learning algorithm will likely take on the order of 10000 steps to converge, depending on the step size used. See the starter code for details on simulating the policy and functions to use for value iteration.

Problem 2: Cart-pole swing-up

In this problem, we will implement a controller to solve the challenging "swing up" problem, in which 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, so we will use iterative LQR control. We provide starter code in `problem2_ilqr_starter.py` and `animations.py`; when submitting code, only provide the lines of code that you are asked to write in your submission.

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

$$s_{k+1} \approx f_{\text{discrete}}(s_k, u_k) \coloneqq s_k + \Delta t f(s_k, u_k), \tag{4}$$

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

$$A_k \coloneqq \frac{\partial f_{\text{discrete}}}{\partial s}(\bar{s}_k, \bar{u}_k), \quad B_k \coloneqq \frac{\partial f_{\text{discrete}}}{\partial u}(\bar{s}_k, \bar{u}_k). \tag{5}$$

Use JAX on line 22 in `problem2_ilqr_starter.py` to write a single line of code that computes A_k and B_k, given f_{discrete}, $\bar{s}_k$ and $\bar{u}_k$.

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

$$\frac{1}{2}(s_N - s^*)^\mathsf{T} Q_N (s_N - s^*) + \sum_{k=0}^{N-1} \big((s_k - s^*)^\mathsf{T} Q (s_k - s^*) + u_k^\mathsf{T} R u_k \big), \tag{6}$$

where s^* is the goal state (i.e., the upright position). The entries of $Q_N \succ 0$ are chosen to be large to enforce a soft terminal constraint.

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

c) Use NumPy to complete the iLQR controller code in the delineated section beginning on line 89 in `problem2_ilqr_starter.py`. Specifically, your code must update the controller terms $\{L_k\}_{k=0}^{N-1}$ and $\{l_k\}_{k=0}^{N-1}$, and the forward-simulated states $\{s_k\}_{k=1}^{N}$ and control inputs $\{u_k\}_{k=0}^{N-1}$.

d) Toggle the boolean on line 148 in `problem2_ilqr_starter.py` to simulate using continuous-time dynamics instead of the Euler-discretized dynamics considered by iLQR. Notice that your iLQR control sequence no longer accomplishes the task if applied open-loop. Use the fact that iLQR additionally outputs a feedback policy that stabilizes around the optimized trajectory to modify the applied control (line 159) and accomplish the swing-up task even when simulating with the "true" system.

e) Run `problem2_ilqr_starter.py` to view a plot of the cart-pole state variables over time and an animation of the swing-up maneuver. Submit the plot only.

Problem 3: Cart-pole swing-up with limited actuation

We used iterative LQR to solve the cart-pole swing-up problem in Problem 2. However, we did not account for constraints on the control (which moves the cart horizontally). In practice, control constraints often arise from motor limitations. In this problem, we consider we study the swing up problem with the following control constraints: $u \in \mathcal{U} = [-5, 3]$.[1]

We will solve this problem using a direct method, relying on Sequential Convex Programming (SCP) as seen in lecture. The key idea is iteratively re-linearizing the dynamics and constructing a convex approximation of the cost function around a nominal trajectory. Since linearization provides a good approximation to the nonlinear dynamics only in a small neighborhood around the nominal trajectory $(\bar{s}, \bar{u})$, the accuracy of the convex model may be poor if s, u deviate far from $(\bar{s}, \bar{u})$. To ensure smooth convergence, we consider a convex trust region on the state and the input, which is imposed as an additional constraint in the convex optimization problem. We consider a box around the nominal trajectory $(\bar{s}, \bar{u})$, which can be written as:

$$\mathcal{T} := \{s, u : ||s - \bar{s}||_\infty \leq \rho, ||u - \bar{u}||_\infty \leq \rho\}$$

[1] Some DC motors run with better performance (more torque) in one direction than the other, so asymmetrical constraints are definitely common in practice.

We will use CVX to solve this problem.

a) Given a nominal trajectory $(\bar{s}, \bar{u})$, where $\bar{s} = (\bar{s}_1, \bar{s}_2, ..., \bar{s}_N)$ and $\bar{u} = \bar{u}_1, \bar{u}_2, ..., \bar{u}_N)$, write down the convex approximation to the optimal control problem for iteration $k + 1$, including trust region constraints. The derived problem should have linear equality constraints obtained by linearizing the dynamics about the nominal trajectory $(\bar{s}, \bar{u})$.

b) The linearized dynamics will have the form $s_{k+1} = A_k s_k + B_k u_k + c_k$. Using JAX, complete the function `linearize` in `problem3_scp_starter.py` to compute A_k, B_k, c_k from $\bar{s}_k, \bar{u}_k$.

c) Complete the code for `scp_iteration` starting on line 59 of `problem3_scp_starter.py`. Specifically, given a nominal trajectory $(\bar{s}, \bar{u})$, use `CVX` to specify and solve the convex optimization problem derived in part a) to obtain an updated solution (s, u).

Hint: Consider using $\rho = 0.5$.

d) Run `problem3_scp_starter.py` to run a simulation after you complete the controller. This script will generate plots of the state and control trajectories. Submit the plots only.

Learning goals for this problem set:

Problem 1: To gain experience with Q-learning in a "real world" setting with implementation details left open-ended.

Problem 2: Apply iLQR for nonlinear trajectory optimization and stabilization; use JAX for dynamics linearization.

Problem 3: Apply sequential convex programming to include control constraints in nonlinear trajectory optimization; build familiarity with CVX.