DS 102 Data, Inference, and Decisions **Fall 2019**

Controls II

Lecturer: Gireeja Ranade

1 Dynamic Programming for Control

In the last lecture we looked at the model-free case of Q-learning. In that setting we did not have any model of the world and instead tried to learn the best action by filling in a Q-function, which is a table denoting the value of all state-action pairs.

In a lot of settings however we have access to a model of the world that gives us some information about how actions impact state transitions. In this lecture we look at this model-based setting. In the model-based setting we are interested in the following quantities at a given time k:

- A state: $x_k \in \mathbb{R}^n$. For example, x_k could be the position and velocity of a car in two dimensional space, in which case $n = 4$.
- An action: $u_k \in \mathbb{R}^d$. Keeping with the car example this could be the acceleration and the angle of the steering wheel of the car, in which case $d = 2$.
- A state transition function: $x_{k+1} = f(x_k, u_k)$. In the car example the function would compute where the car would be at the next timestep given the current position, velocity, acceleration and steering direction. The function f can either be deterministic or stochastic, for example there might be some amount of external random influence on the car such as the wind or the quality of the road that would impact the next state of the car.
- A cost function: $c_k(x_k, u_k)$. For example, the cost function might be based on how far the car deviates from a specific trajectory, and how much the car is accelerating.

Within this setting we are interested in minimizing the running cost over the N timesteps during which we run:

$$J(x_0) = \sum_{k=0}^{N} c_k(x_k, u_k).$$

In other words we want to find

$$\begin{aligned} J^*(x_0) &= \min_{u_0, u_1, \ldots u_N} J(x_0) \\ &= \min_{u_0} \left\{ c_k(x_0, u_0) + \min_{u_1, u_2, \ldots, u_N} \sum_{k=1}^{N} c_k(x_k, u_k) \right\} \\ &= \min_{u_0} \left\{ c_k(x_0, u_0) + J^*(f(x_0, u_0)) \right\}. \end{aligned}$$

The function J is often called the cost-to-go since it specifies the cost starting from our current state. The above expression is reminicent of the Bellman equations from Lecture 18, and ideally we would like to find the optimal cost-to-go through dynamic programming. Unfortunately the fact that the state and action space are infinite in the number of possible elements, and the fact that the function f can be arbitrarily complex means that we can't solve this without specializing the problem.

2 Linear Systems

A class of models that have found broad applicability within controls are linear systems. Linear systems limit what the class of function f can be such that we can solve the dynamic programming problem above in a tractable fashion. While the assumptions that we are going to make on f will restrict what kind of problems we can express, it's worth noting that linear systems are often used in the real world. This is in large part because we can approximate a nonlinear system with a different linear system at every timestep k. Using this method of *linearization* allows us to closely approximate a large class of nonlinear systems while keeping the tractability of linear systems[1].

A linear system is a setting in which the state transition function f can be represented as

$$x_{k+1} = Ax_k + Bu_k + w_k,$$

where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times d}$, and $w_k \in \mathbb{R}^n$ is random Gaussian noise. Furthermore assume that the cost function has the form

$$\begin{aligned} c_k(x_k, u_k) &= Qx_k^2 + Ru_k^2 \quad \text{for } k < N, \\ c_N(x_N, u_N) &= Qx_N^2. \end{aligned}$$

For the remainder of these notes we make the further simplifying assumption that everything is a scalar ($n = 1$ and $d = 1$), and that there is no noise[2]. This is mostly to simplify the proofs, but it's worth noting that our analysis would be valid for arbitrary n and d with a few linear algebraic tweaks.

We wish to find a way to compute the controller u_k that will give us the optimal cost-to-go $J^*(x_0)$ in this setting. First we'll show that the optimal cost-to-go at time k can be written as

$$J^*(x_k) = K_k x_k^2,$$

[1] The details of linearization are beyond the scope of this lecture but it's good to keep this idea in the back of your mind as we delve deeper into linear systems.

[2] The no-noise condition differs from the lectures but the result is identical, we just don't need to worry about expectations here.

where

$$K_k = Q + \left(K_{k+1} - \frac{K_{k+1}^2 B^2}{K_{k+1}^2 B^2 + R} \right) A^2 \quad \text{for } k < N,$$
$$k_N = Q.$$

We show this by induction, starting from the end-state. It's clear the base-case is true by definition since

$$J^*(x_N) = Q x_N^2.$$

Now assume the identity holds for time $k+1$, we want to show that it also holds at time k. Writing down the cost-to-go at time k and expanding it out gives us:

$$\begin{aligned} J^*(x_k) &= \min_{u_k} \left\{ Q x_k^2 + R u_k^2 + J^*(A x_k + B u_k) \right\} \\ &= \min_{u_k} \left\{ Q x_k^2 + R u_k^2 + K_{k+1} (A x_k + B u_k)^2 \right\}, \end{aligned}$$

where we have used the assumption that the identity holds for the second equality. Now taking the derivative with respect to u_k and setting it to 0 gives us that the minimizer u_k^* is:

$$\begin{aligned} & 2 R u_k^* + 2 B K_{k+1} (A x_k + B u_k^*) = 0 \\ \implies & u_k^* = -\frac{K_{k+1} A B}{R + K_{k+1} B^2} x_k. \end{aligned}$$

Substituting the value of u_k^* into the equation for $J^*(x_k)$ and rearranging gives us our result. Now we note that we now also have an optimal controller at each timestep u_k^*. Hence the algorithm to solve this linear control problem is as follows:

1. Starting from the last timestep and moving backward compute and save all K_k using K_{k+1}.
2. Start the linear system and at each state compute

   $$L_k = -\frac{K_{k+1} A B}{R + K_{k+1} B^2}$$

   using the saved value K_{k+1}.
3. Play the input $u_k^* = L_k x_k$ to obtain the next state x_{k+1}
4. Repeat until we reach timestep N.