Optimal Control: LQR

Eating your cake and having it in optimal control problems

Marin Vlastelica Pogančić May 4, 2019 · 6 min read ⋆

In this article, I am going to talk about optimal control. More specifically I am going to talk about the unbelievably awesome Linear Quadratic Regulator that is used quite often in the optimal control world and also address some of the similarities between optimal control and the recently hyped reinforcement learning. It is fascinating that they are named differently yet they address similar issues in sequential decision-making processes. That being said, a friendly math warning to readers:

> This article contains some linear algebra and calculus, but don't panic, it is easy-peasy, you can do it.

Now that we got that out of the way, let's start. First of all let us define an optimal control problem in general, or better to say an optimization problem. This merely means that we want to maximize or minimize some function subject to certain constraints on the variables. A typical optimal control problem would look like this:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i,\ i = 1, \ldots, m. \end{aligned}$$

Which is quite straight forward, minimizing a function f subject to some constraints (the s.t. is short for subject to). Now, in the optimization world this can be arbitrarily difficult based on how the objective function looks like and the constraints. The constraints can, of course, be equality constraints or inequality constraints based on the problem. Needless to say, non-convex functions in an optimization problem are hard to optimize over, but in the case where we have convex functions we can solve the problem efficiently and fast. Anyway, it is so significant that your reaction to finding convexity in your problem should look like this:

In control problems, we optimize our trajectories to minimize the cost function or rather maximize the reward as it is done in reinforcement learning. Naturally, the dynamics of the environment, i.e. the function that gives us the next state based on current action and current state, are part of the optimization constraints. So we might write our control optimization problem as follows):

$$\begin{aligned} \min_{u_{0\dots N}} \quad & \sum_{i=0}^{N-1} g(x_i, u_i) + E(x_N) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_0 - \bar{x} = 0 \end{aligned}$$

This is the case for finite horizon till N. Lets break it down in short. x is our state variable at each time step, ***u*** is our action. **E** would be the final cost of the final state, ***g*** the cost function for each state-action pair. ***x*** bar is our start state from which we want to optimize and ***f*** is our dynamics function. In this case we have no inequality constraints. As it turns out, if our function ***f*** is a linear function of ***x*** and ***u*** and the function ***g*** a quadratic function of ***x*** and ***u***, this makes the problem a lot simpler. This is how we arrive at the Linear Quadratic Regulator problem definition:

$$\begin{aligned} \min_{x,u} \quad & \sum_{i=0}^{N-1} (x_i^\intercal Q x_i + u_i^\intercal R u_i) + x_N^\intercal E x_N \\ \text{s.t.} \quad & x_{k+1} = A x_k + B u_k \\ & x_0 - \bar{x} = 0 \end{aligned}$$

Here, **Q**, **R** and **E** are the cost matrices that define our polynomial coefficients. We could also write the cost for each time step in block matrix notation to make the expression simpler:

$$g(x,u) = \begin{bmatrix} x \\ u \end{bmatrix} \begin{bmatrix} Q & S^\intercal \\ S & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}^\intercal$$

In the upper case we leaved out the S or better to say we assumed that S = 0, but this doesn't change the math greatly, S can also be some kind of matrix for interplay between x and u in the cost function.

We are going to make use of the optimality principle, which states a natural fact, namely that if you have an optimal route between points A and C, if we take a point B on this route then the subroute A to B is also the optimal route from A to B. Completely intuitive fact. Based on this, we can define the optimal cost-to-go, or total cost for our trajectory recursively. This is how we arrive to the Hamilton-Jacobi-Bellman equation:

$$J^*(x_i) = \min_u g(x_i, u) + J^*(x_{i+1})$$

$$J^*(x_k) = \min_u g(x_k, u) + J^*(x_{k+1})$$

Where J star is our optimal cost-to-go. In our case we stated the objective function as a polynomial function, so logically we can assume that our optimal cost-to-go is a polynomial function and we can write it as such:

$$J_k = x_k^\intercal P_k x_k$$

And logically our final cost would be the following based on our definition of the optimization problem :

$$J_N = x_N^\intercal P_N x_N$$
$$P_N = E$$

Now, if we plug in our definition of the function g and the environment dynamics into the Bellman equation we arrive at something like this:

$$J^*(x_k) = \min_u \begin{bmatrix} x_k \\ u \end{bmatrix}^\intercal \begin{bmatrix} Q & S^\intercal \\ S & R \end{bmatrix} \begin{bmatrix} x_k \\ u \end{bmatrix} + \begin{bmatrix} x_k \\ u \end{bmatrix}^\intercal \begin{bmatrix} A \\ B \end{bmatrix} P_{k+1} \begin{bmatrix} A \\ B \end{bmatrix}^\intercal \begin{bmatrix} x_k \\ u \end{bmatrix}$$

Thanks to the quadratic cost assumptions, how do we find the minimum of this function? Well, quite simply we take the gradient with respect to u and equate it to 0, we group all the terms into one big central matrix:

$$\nabla_u \begin{bmatrix} x_k \\ u \end{bmatrix}^\intercal \begin{bmatrix} Q + AP_{k+1}A^\intercal & S^\intercal + AP_{k+1}B^\intercal \\ S + B^\intercal P_{k+1}A & +BP_{k+1}B^\intercal \end{bmatrix} \begin{bmatrix} x_k \\ u \end{bmatrix} = 0$$

In order to save space we substitute the terms with the following matrices (this is self-explanatory):

$$\nabla_u \begin{bmatrix} x_k \\ u \end{bmatrix}^\intercal \begin{bmatrix} \bar{Q} & \bar{S}^\intercal \\ \bar{S} & \bar{R} \end{bmatrix} \begin{bmatrix} x_k \\ u \end{bmatrix} = 0$$

After multiplying everything out and looking only at the terms containing ***u***, because we want to take the derivative with respect to ***u***, we arrive at the following intermediate result:

$$\nabla_u u^\intercal \bar{S} x_k + x_k^\intercal \bar{S}^\intercal u + u^\intercal \bar{R} u = 0$$

After calculating the gradient and rearranging, we get the expression for u star that minimizes the cost, the optimal action:

$$u^* = -\bar{R}^{-1}\bar{S}x_k$$

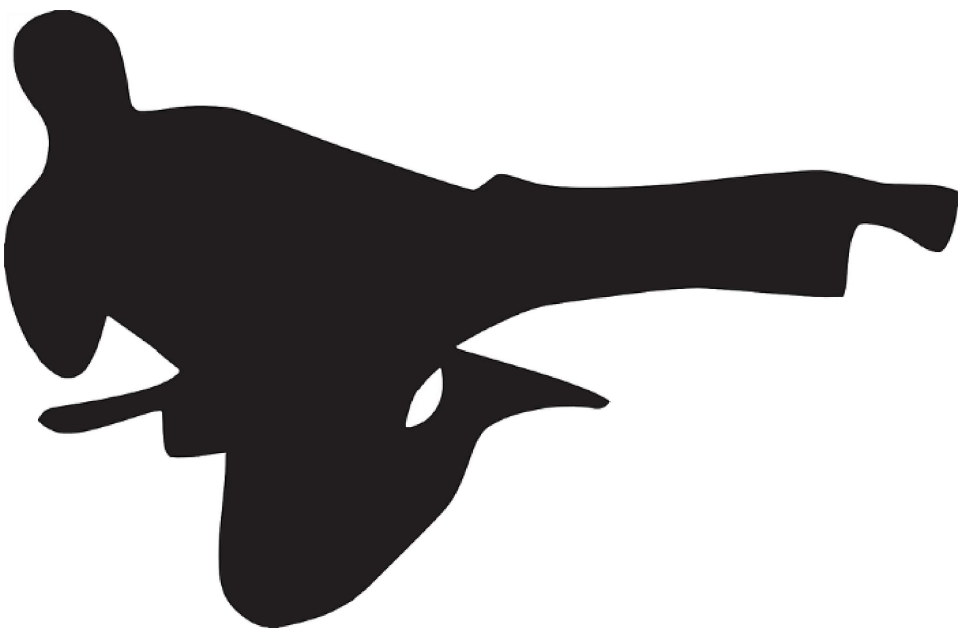
Maybe meditate on this a bit for a minute. What does this mean? Well, it means that we have a closed-form solution to the optimal action! This is pretty neat. So what do we need to solve this? We need the matrix P for time step k+1. This we can calculate based on the following equation, recursively from the last time step:

$$P_k = Q + A^\intercal P_{k+1} A + A^\intercal P_{k+1} B (B^\intercal P_{k+1} B + R)^{-1} B^\intercal P_{k+1} A$$

This is also known widely as the algebraic Riccati equation. In the case where we want a fixed point solution, for infinite horizon, the equation can be solved for a fixed **P.** In that case, we don't even need a recursion. We just get the optimal-feedback control for free.

That would be all to it basically. You have to appreciate the power of the LQR. Of course, many problems can't be simplified to linear dynamics, but it is amazing what kind of solution do we get if we make the simplification. This approach is even used in situations where our dynamics that are not linear by linearizing them around fixed points through Taylor expansion. This is an approach that is regularly used in trajectory optimization for complex problems and is called Differential Dynamic Programming (DDP), an instance of which is iLQR (iterative LQR), go figure.

Now that you obtained some LQR-fu, you have obtained the tool to understand many things in optimal control.



I hope that this explanation of LQR opened some eyes. It is a very simple yet powerful concept and a building block for many optimal control algorithms!