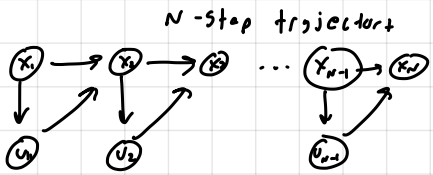
2/9 Recitation

- LQR
- Linearization
- convex.jl tutorial

$$x_{k+1} = f(x_k, u_k)$$



When we write our dynamics in this form, a trajectory with N "knot points" will have N states, and N - 1 controls. When we start solving for trajectories numerically, we are almost always going to solve for it like this. The one exception (in this class) will be direct collocation (HW3).

N x's
(N-1) u's

$$\nabla_x\left(\frac{1}{2}x^TQx + b\right) = Qx + q$$

Finite-horizon LQR

"TVLQR"

$$\min_{x_{1:N},\, u_{1:(N-1)}} \quad \underbrace{\frac{1}{2}\left(\sum_{i=1}^{N-1} x_i^T Q x_i + u_i^T R u_i\right)}_{\text{stage cost}} + \underbrace{\frac{1}{2}x_N^T Q_N x}_{\text{terminal cost}}$$

$$\text{s.t.} \quad x_0 = x_{IC} \qquad \text{Initial condition}$$

$$x_{k+1} = A_k x_k + B_k u_k, \quad \text{for } k = 1, 2, \ldots, N-1$$

Finite-Horizon LQR (FHLQR), also known as Time-Varying LQR (TVLQR) when the A's and B's are a function of time, is described by the following optimization problem. For FHLQR, there is just one A and one B. For TVLQR, there is an A and B for each timestep k.

$Q \succeq 0$, $R > 0$
PSD PD

- eq - constrained QP
- solve w/ linear system

This optimization problem is nice because it has a quadratic cost and linear equality constraints. This means we can solve for it in closed form, using just a single linear system solve (prove this to yourself by writing out the KKT conditions and solving for the primal and dual variables). We can also solve this problem with a Ricatti recursion, that gives us K_1, K_2, … , K_{N-1}, which is globally optimal for any u_k = -K_k x_k.

Infinite-horizon LQR

$$\min_{x_{1:\infty},\, u_{1:\infty}} \sum_{i=1}^{\infty} x_i^T Q x_i + u_i^T R u_i$$

$$\text{s.t. } x_0 = x_{IC}$$

$$x_{k+1} = A x_k + B u_k$$

IHLQR lets us reason about trajectories with no set length. This is equivalent to "control this thing forever". Examples of this include temperature regulation on an oven, cruise control in your car, keeping something balanced, etc. We can only solve this with Ricatti, since the trajectory is of infinite length (we can't use normal convex optimization when we have an infinite number of variables).

IHLQR can be solved w/ Ricatti:

FHLQR " " or convex optimization

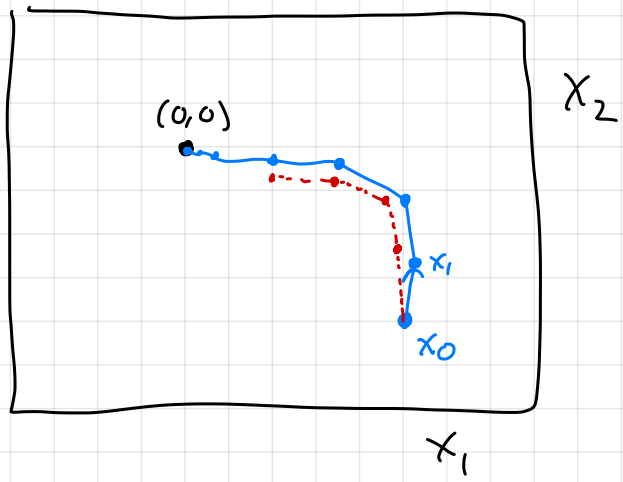
$u_i = -K_i x$ for FHLQR

$u_i = -K x$ for IHLQR

IHLQR can only be solved with Ricatti, FHLQR can be solved with either Ricatti or convex optimization. In both of the Ricatti cases, we get a linear feedback policy that does not depend on initial condition, but in the case of convex optimization, we get a set of u's that do depend on the initial condition.

TVLQR: ① $u_i = \arg\min_{x_{1:\infty},\, u_{1:\infty}}$ FHLQR problem , solve a QP

Globally optimal ② $u_i = -K_i x$, matmul



If we were to just solve a QP (solution via convex optimization), we would have to resolve this QP every time we get a new initial condition that wasn't part of our plan. This is bound to happen due to model mismatch or un-modeled disturbances. The Ricatti solution (the linear feedback policy), is globally optimal for any initial condition. This means we don't have to replan anything when the states change differently than we expected, the feedback policy is still globally optimal no matter where we are in the state space. This is a super important point so it will be reiterated multiple times.

LQR to x_{goal}

$$x_{k+1} = A x_k + B u_k \quad , \text{ I want } (x_{goal}, u_{goal})$$

$$x_{goal} = A x_{goal} + B u_{goal}$$

$\dot{x}$ at x_{goal} is 0

$\tilde{x} = x - x_g$, $\tilde{u} = u - u_g$ new tilde coords

$$\tilde{x}_{k+1} = x_{k+1} - x_g = A x_k + B u_k - (A x_g + B u_g)$$

$$\tilde{x}_{k+1} = A(x_k - x_g) + B(u_k - u_g)$$

$$\boxed{\tilde{x}_{k+1} = A \tilde{x}_k + B \tilde{u}_k}$$

$K = LQR(A, B, Q, R, Q_N)$ ← NO input w/ x_g, u_g

$$\tilde{u} = -K \tilde{x}$$

$$u - u_g = -K(x - x_g)$$

$$\boxed{u = u_g - K(x - x_g)}$$

how we use LQR to drive to any goal where $x_g = A x_g + B u_g$

Another benefit of the feedback policy is it lets us drive the system to any achievable goal. You can introduce a new set of variables (the tilde variables here), and see that we still get the same feedback matrix K. The only difference is now the feedback policy is \tilde{u} = -K \tilde{x}. With a simple substitution we see that we can drive our system to an xg and ug by simply doing u = ug - K(x - xg).

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

$$x_0 = Ax_0 + Bu_0$$

This change of variables trick can also be done with deltas, since they are just made up variables anyways. This is important once we start linearizing things.

$$x = x_0 + \Delta x \quad , \quad u = u_0 + \Delta u$$

$$x_{k+1} = x_0 + \Delta x_{k+1} = A(x_0 + \Delta x) + B(u_0 + \Delta u)$$

$$\boxed{\Delta x_{k+1} = A\Delta x_k + B\Delta u_k}$$

LQR for nonlinear

$$x_{k+1} = f(x_k, u_k)$$

In order to use LQR on a nonlinear system, we must first linearize it. Using a first-order Taylor expansion, we approximate the discrete dynamics of our system as linear in x and u. We then introduce new made up delta variables, and substitute them into our Taylor series.

$\bar{x}, \bar{u}$ is eq, $\bar{x} = f(\bar{x}, \bar{u})$

$$x_{k+1} = f(x_k, u_k)$$

1st order lin

$$x_{k+1} \approx f(\bar{x}, \bar{u}) + \underbrace{\left[\frac{\partial f}{\partial x}\Big|_{\bar{x},\bar{u}}\right]}_{A}(x - \bar{x}) + \underbrace{\left[\frac{\partial f}{\partial u}\Big|_{\bar{x},\bar{u}}\right]}_{B}(u - \bar{u})$$

$$x = \bar{x} + \Delta x \quad , \quad u = \bar{u} + \Delta u$$

Substitute in the deltas and we get an identical system to what we just showed at the top of this page.

$$x_{k+1} \approx f(\bar{x}, \bar{u}) + A(x - \bar{x}) + B(x - \bar{x})$$

$$\bar{x} + \Delta x_{k+1} = f(\bar{x}, \bar{u}) + A\Delta x + B\Delta u$$

$$\Delta x_{k+1} = A \Delta x_k + B \Delta u_k$$

After linearization + introduction of our delta variables, we now have the dynamics in a form we are familiar with. We just use LQR on this linearized system and get a feedback policy that we can use.

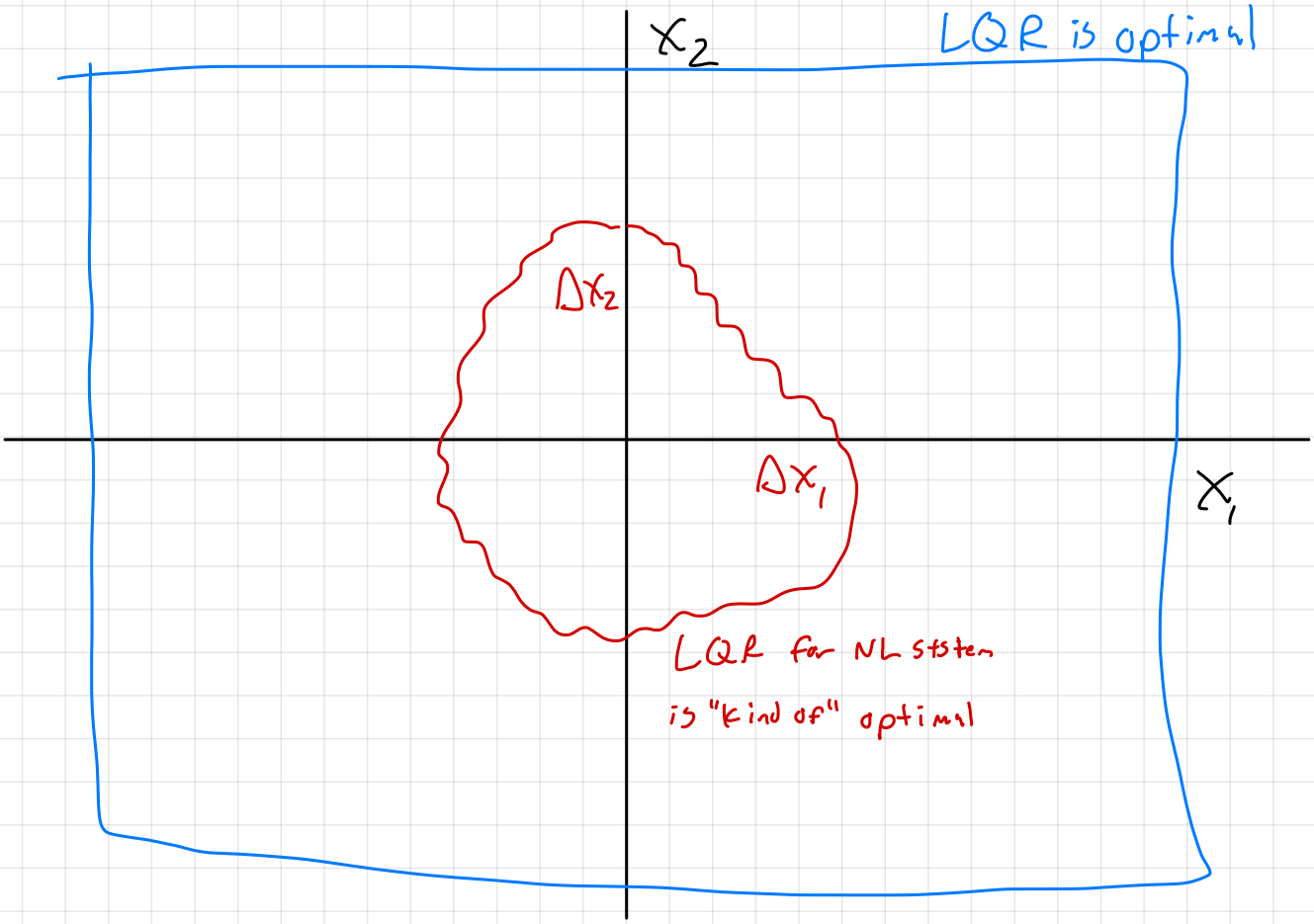
$$K = LQR(A, B, Q, R, Q_N)$$

$$\Delta u = -K \Delta x$$

$$u - \bar{u} = -K(x - \bar{x})$$

$$u = \bar{u} - K(x - \bar{x})$$

It is very important to note that LQR is optimal everywhere in the state space for linear systems, but when we use LQR on a nonlinear system that we linearize about some point, our linearized model is only accurate in the neighborhood of our linearization point. Because of this, LQR on nonlinear systems can work well in a close vicinity to the linearization point, but will often fail if the system goes far enough from this point that the linearization is no longer accurate. This is surprisingly not a very big problem because the controller is fighting to keep the system near the linearization point, and therefore also fighting to keep the linearization accurate.



TVLQR for NL sys

$\bar{x}_{k+1} = f(\bar{x}_k, \bar{u}_k)$

$\bar{x}_{1:N}$ $\bar{u}_{1:N-1}$, $x = \bar{x} + \Delta x$, $u = \bar{u} + \Delta x$

$\bar{x}_{k+1} + \Delta x_{k+1} = f(\bar{x}_k, \bar{u}_k) + A_k \Delta x_k + B_k \Delta u_k$

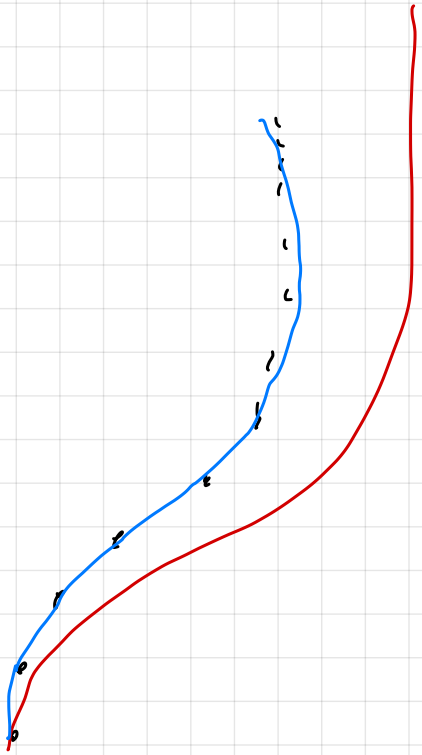
$\Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k$

$K = \text{TVLQR}(A, B, Q, R, Q_f)$

$\Delta u = -K \Delta x$

$u_k - \bar{u}_k = -K(x_k - \bar{x}_k)$

$u_k = \bar{u}_k - K(x_k - \bar{x}_k)$



We can do this same thing for trajectory tracking with TVLQR. We linearize the dynamics of the system at each knot point, introduce new delta variables, and recover a time-varying linear system (in the deltas). We then use our normal TVLQR Ricatti solution method to get a set of (N - 1) K matrices, giving us a feedback policy for each timestep. Like before, we really need to stay near the knot points we linearized about for the controller to work well.