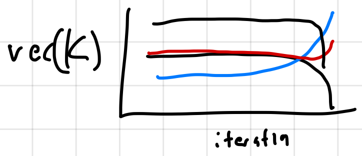
3/24 Recitation

$$u = -Kx$$

$$\begin{bmatrix} u \end{bmatrix} = -\begin{bmatrix} & K & \end{bmatrix}\begin{bmatrix} x \end{bmatrix}$$



First we talked about how the LQR gain matrix K and resulting feedback policy u = -Kx is globally optimal. Basically it odoesn't matter where we are in the state space, as long as we do u = -Kx, we are acting optimally. It's also worth noting that we did not use any state sepcific information when we solved for K using our Ricatti recursion.

DIRCOL

Direct Collocation (DIRCOL) is a way for us to solve trajectory optimization problems as constrained nonlinear programs (NLPs). We are optimizing over states and controls, and we constrain them to be dynamically feasible with either an explicit or implicit integrator. Since they end up being equality constraints in the solver, nothing about the problem setup changes if the integrator is explicit or implicit. In class, Zac went over a nice interpretation of this as approximating the solution to our dynamics ODE as a spline, this is exactly equivalent to using Hermite-Simpson (the implicit integrator) to make our dynamics constraints.

$$\min_{x_{1:N},\, u_{1:(N-1)}} \quad \overset{\text{term cost}}{\ell_N(x_N)} + \overset{\text{stage cost}}{\sum_{i=1}^{N-1} \ell(x_i, u_i)}$$

s.t.

$$x_1 = x_{IC}$$

$$☆ \quad 0 = \overset{\text{explicit integrator}}{f(x_k, u_k) - x_{k+1}} \quad \text{for } k = 1 : N-1$$

$$☆ \quad 0 = \overset{\text{implicit integrator}}{f_{imp}(x_k, u_k, x_{k+1})} \quad \text{for } k = 1 : N-1$$

$$\underline{x} \leq x \leq \overline{x}$$
$$\underline{u} \leq u \leq \overline{u}$$

$$g(u_i, x_i) \leq 0$$
$$c(u_i, x_i) = 0$$

☆ forward euler (explicit)

$$0 = x_k + dt \cdot \dot{x}_k - x_{k+1}$$

Here's an example of how these constraints would look if we used forward Euler (explicit) or backward Euler (implicit).

☆ Backwards Euler (implicit)

$$0 = x_k + dt\, \dot{x}_{k+1} - x_{k+1}$$

$$\Downarrow$$

Now we have to convert this problem to a NonLinear Program (NLP) by stacking up our X's and U's into one vector Z.

Solve w/ NLP (IPOPT/SNOPT)

$$z = \begin{bmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \\ x_{N-1} \\ u_{N-1} \\ x_N \end{bmatrix}$$

$$\min_z \quad f(z)$$

$$\text{s.t.} \quad c_{eq}(z) = 0$$

$$\underline{c} \le c_{ineq}(z) \le \overline{c}^{*}$$

$$\underline{z} \le z \le \overline{z}$$

$g(z) \le 0 \quad \downarrow \quad -\infty \le g(z) \le 0$

$|u| \le 10$

$-|u| \le -5$

$5 \le |u| \le 10$

$x_i = z[\text{IDX.x[i]}]$

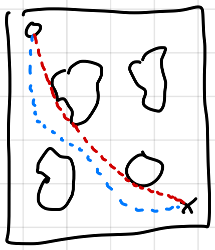
$u_i = z[\text{IDX.u[i]}]$

There are many "standard forms" for NLP's, we are going to use this form in this class.

Z_0 - initial guess

With DIRCOL (and NLP solvers in general), we want to initialize the solver with a good guess Z0. We can use this to warm start the solver with a rough trajectory that doesn't have to be dynamically feasible or feasible to the general constraints, but the closer it is to the optimal solution the easier time the solver will have. Choice of initial guess is crucial for the success/speed of DIRCOL. We can initialize states/controls independently, and our guess does not have to be a real trajectory (dynamically feasible).

ex:



Z_0 (Not dynamically feasible)

z^* (dyn. feasible)

iLQR

iLQR is a DDP based method for solving trajectory optimization problems of exactly this form. While it can be modified to work with implicit integrators, it was significantly more common to stick with explicit integrators.

term cost

stage cost

$$\min_{x_{1:N}, u_{1:(N-1)}} \quad \ell_N(x_N) + \sum_{i=1}^{N-1} \ell(x_i, u_i)$$

$$x_1 = x_{IC}$$

$$x_{k+1} = f(x_k, u_k) \quad \text{for} \quad k = 1 : N-1$$

99%, explicit integrators
(implicit is possible too)

$$x^*, u^*, K = iLQR(x_{IC}, U_{guess}, f, \ell, \ell_N)$$

TVLQR Gains about x^*, u^*

iLQR returns X, U, and K, where K is our TVLQR gains that track X and U. It is important to note that we initialize iLQR with only an initial condition and a guess control trajectory Uguess. This gives us less freedom for initializing the solver, and for some problems this can be quite limiting. An example would be bipedal locomotion where it's hard to find a guess trajectory that makes the biped walk, making the initial guess U tricky.

DIRCOL does not suffer from this, as it allows for initialization of the states and controls independently.