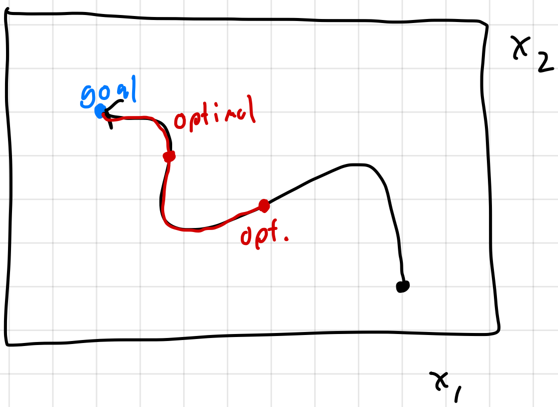
2/16 Recitation

- Bellman and Dynamic Programming
- MPC
- Code for quadrotor MPC

Bellman's Principle of Optimality



Bellman's principle of optimality basically states that any subtrajectory in an optimal trajectory is also optimal. If my optimal trajectory goes D->B->C->A, then both B->C->A and C->A are optimal as well. If they weren't and there was a better way to get from B to A, why wouldn't the original optimal trajectory have taken it?

Dynamic Programming uses this idea to work backwards from the goal to give us an optimal policy. This can give us globally optimal actions, but it really only works for very simple things (like our linear dynamics). Even for "easy" nonlinear systems, dynamic programming can be intractable.

$V(x)$ is optimal cost to go from x

V(x) is our "value function" or "optimal cost-to-go". It tells us that given our problem setup (costs and constraints), what is the cost incurred from that point forward if we act optimally. This is extremeley powerful, because if we knew this function, we would just take a step with our dynamics to the state with the lowest cost-to-go.

FHLQR (or TVLQR)

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

$$\text{s.t.} \quad x_{i+1} = A x_i + B u_i \quad \text{for } i = 1, 2, \dots N-1$$

$$x_1 = x_{IC}$$

Here is our familiar FHLQR problem, we can either solve this as a convex optimization problem for x's and u's (this depends on xIC), or we can solve it with dynamic programming (Ricatti) to get a simple feedback policy that does not depend on xIC.

1. Solve with convex solver for x's and u's ← this depends on x_0
2. Solve with Ricatti for $u_i = -K_i x_i$ ← this does not depend on x_0

Ricatti via DP for $x_{k+1} = Ax_k + Bu_k$

value $$V_N(x) = x^T Q_N x$$

Let's start at the last timestep, where we only have one cost term left, the terminal cost. This means we can write down our value function for the final timestep as just the terminal cost. From here, we define an action-value function (sometimes called Q function), that is the stage cost to get from x to xk+1, and the value function at v(xk+1). The optimal policy u = argmin_u S(x, u).

action value $$S(x,u) = \underbrace{\ell(x,u)}_{\text{stage cost}} + \underbrace{V(x_{k+1})}_{\text{value function}}$$

$$S(x,u) = \underbrace{\ell(x,u)}_{\text{stage cost}} + \underbrace{V(Ax+Bu)}_{\text{value function}}$$

$$V_k(x) = \min_u S_k(x,u) = \min_u \ell_k(x,u) + V_{k+1}(\overbrace{Ax+Bu}^{x_{k+1}})$$

assume quadratic value function $V_k(x) = x^T S_k x$, so $S_N = Q_N$

$$V_{N-1}(x) = \min_u \overbrace{x^T Q x + u^T R u}^{\text{stage cost}} + V_N(Ax+Bu)$$

$$V_{N-1}(x) = \min_u \underbrace{\left(x^T Q x + u^T R u + (Ax+Bu)^T S_N (Ax+Bu)\right)}_{S_{N-1}(x,u)}$$

$$\nabla_u S_{N-1}(x,u) = 2Ru + 2B^T S_N(Ax+Bu) = 0$$

$$(R + B^T S_N B)u = -B^T S_N A x$$

$$u = \underbrace{-(R+B^T S_N B)^{-1} B^T S_N A}_{K} x$$

Since we know V_N(x), let's solve for u_N-1. By plugging in V_N(x) to our action-value function at N-1, we have a nice quadratic function to minimize. The minimizing argument of this function is below, which is conveniently just a feedback policy on the current value of x.

$$u_{N-1} = \underbrace{-(R + B^T S_N B)^{-1} B^T S_N A}_{K_{N-1}} x_{N-1}$$

plug $[u_{N-1} = -K x_{N-1}]$ back into $V_{N-1}(x)$

$$V_{N-1}(x) = x^T Q x + u^T R u + (Ax + Bu)^T S_N (Ax + Bu)$$

$$V_{N-1}(x) = x^T Q x + x^T K^T R K x + ((A-BK)x)^T S_N ((A-BK)x)$$

$$\boxed{V_{N-1}(x) = x^T \underbrace{\left[Q + K^T R K + (A-BK)^T S_N (A-BK) \right]}_{S_{N-1}} x}$$

If we plug in our optimal feedback policy u into our action value function, we get an expression for the value function at N-1. This showed us that we can form this feedback policy and value function quadratic term in a recursive fashion. This gives us our Ricatti recursion.

$S_N = Q_N$

for $i = [N-1, N-2, .. 1]$

if TVLQR $A = A_i$, $B = B_i$

$K_i = (R + B^T S_{i+1} B)^{-1} B^T S_{i+1} A$

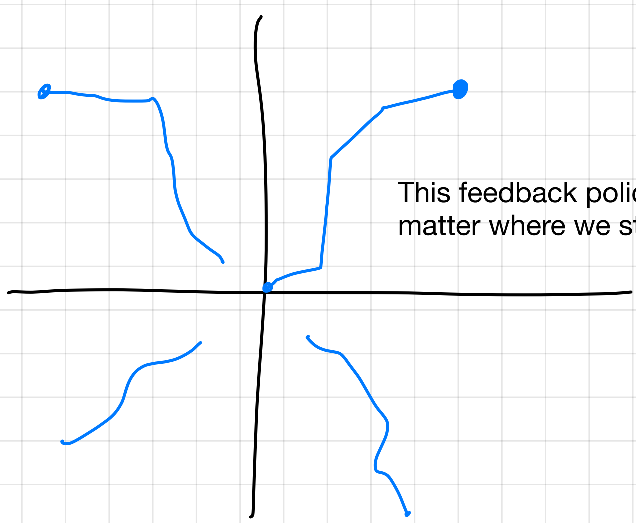
$S_i = Q + K_i^T R K_i + (A - BK_i)^T S_{i+1} (A - BK_i)$

end

We can solve TVLQR by adding A_i, B_i

For FHLQR, A and B are our linear dynamics. For TVLQR, there are A_i, B_i that are specific to each timestep.

$U_i = -K x_i$



This feedback policy is optimal everywhere, doesn't matter where we start.

MPC

what to do if we have constraints or costs not in LQR

Sometimes our problems have costs/constraints that don't fit into the LQR framework. This is ok, especially if the resulting problem is still convex. Turns out, solving convex trajectory optimization problems is very fast and easy, though not as fast as a simple linear feedback policy.

if x_1 is my current timestep

$$\min_{\substack{x_{1:N} \\ u_{1:N-1}}} \underbrace{\left(\sum_{i=1}^{N-1} x_i^T Q x_i + u_i^T R u_i\right)}_{\text{stage cost}} + \underbrace{x_N^T Q_N x_N}_{\text{terminal cost}} + 10\|u_3\|_1$$

$$\text{s.t.} \quad x_{i+1} = A x_i + B u_i \quad \text{for } i = 1, 2 \ldots N-1$$

$$x_1 = x_{IC}$$

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

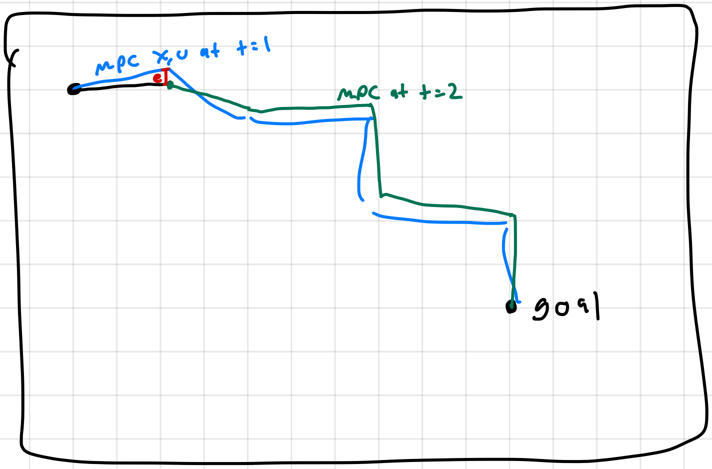
$$\underline{u} \leq u \leq \overline{u}$$

MPC in a nutshell:

- take our current position xIC
- solve a trajectory optimization problem looking at the next N time steps
- execute the first control U_1 from this problem on the system
- repeat this whole process at the next time step

LQR: $u = -Kx$ Feedback policy

MPC: $u = \text{solve\_mpc}(x_0)$ Solve a small trajopt problem and return first control input



$$\boxed{\text{TVLQR}}$$

$$\bar{x}, \bar{u}$$

If we linearize a nonlinear system about a nominal trajectory Xbar Ubar, we can track a given reference trajectory assuming the linearization is accurate. This gives us a time-varying linear system, and we can use TVLQR to track a reference.

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

$$\cancel{\bar{x}_{k+1}} + \Delta x_{k+1} \approx \cancel{f(\bar{x}_k, \bar{u}_k)} + \overbrace{\left[\frac{\partial f}{\partial x}\bigg|_{\bar{x}_k, \bar{u}_k} \right]}^{A_k} \Delta x_k + \overbrace{\left[\frac{\partial f}{\partial u}\bigg|_{\bar{x}_k, \bar{u}_k} \right]}^{B_k} \Delta u_k$$

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

$$A_k = \frac{\partial f}{\partial x}\bigg|_{\bar{x}_k, \bar{u}_k}$$

$$B_k = \frac{\partial f}{\partial u}\bigg|_{\bar{x}_k, \bar{u}_k}$$

$$S_N = Q_N$$

for k: $(N-1, N-2, N-3, \dots, 3, 2, 1)$

$$K_k = (R + B_k^T S_{k+1} B_k)^{-1} B_k^T S_{k+1} A_k$$

$$S_k = Q + K_k^T R K_k + (A_k - B_k K_k)^T S_{k+1} (A_k - B_k K_k)$$