Last Time:

- Merit functions
- Control History
- Deterministic Optimal Control
- Pontryagin

Today:

- LQR Intro
- LQR via Shooting
- LQR as a QP
- Riccati Recursion

---

* LQR Problem

$$\min_{\substack{x_{1:N} \\ u_{1:N-1}}} \quad J = \sum_{n=1}^{N-1} \frac{1}{2} x_n^T Q_n x_n + \frac{1}{2} u_n^T R_n u_n + \frac{1}{2} x_N^T Q_N x_N$$

$$\text{s.t.} \quad x_{n+1} = A_n x_n + B_n u_n$$

$$Q \succeq 0 \quad , \quad R > 0$$

- Can (locally) approximate many nonlinear problems
- Computationally tractable
- Many extensions e.g. infinite horizon, stochastic, etc.
- "Time invariant" if $A_n = A$, $B_n = B$, $Q_n = Q$, $R_n = R$ $\forall t$, "time varying" otherwise.

* LQR with Indirect Shooting:

$$x_{n+1} = \nabla_\lambda H(x_n, u_n, \lambda_{n+1}) = Ax_n + Bu_n$$

$$\lambda_n = \nabla_x H(x_n, u_n, \lambda_{n+1}) = Qx_n + A^T \lambda_{n+1}$$

$$\lambda_N = Q_N x_N$$

$$u_n = \underset{u}{\operatorname{argmin}}\ H(x_n, u, \lambda_{n+1})$$

$$= -R^{-1} B^T \lambda_{n+1}$$

- Procedure:

1) Start with an initial guess $u_{1:N-1}$
2) Simulate/rollout to get $x_{1:N}$
3) Backward pass to compute λ and δu
4) Rollout with line search on δu
5) GoTo 3 until convergence

* Example:

- "Double Integrator"

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

($\dot{q}$: velocity; $\ddot{q}$: acceleration; q: position; u: force)

- Think of this as a brick sliding on ice (no friction)

$$x_{n+1} = \underbrace{\begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} q_n \\ \dot{q}_n \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix}}_{B} u_n$$

(h: time step)

* LQR as a QP

- Assume x_1 (initial state) is given (not a decision variable).

- Define $z = \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}$

- Define $H = \begin{bmatrix} R_1 & & & \\ & Q_2 & & 0 \\ & & R_2 & \\ & 0 & & Q_3 \ddots \\ & & & & Q_N \end{bmatrix}$

such that $J = \frac{1}{2} z^T H z$

- Define C and d:

$$\underbrace{\begin{bmatrix} B & -I & 0 & 0 & \cdots & & \\ 0 & A & B & -I & 0 & \cdots & \\ & & \ddots & \ddots & & & \\ & & & & A_{N-1} & B_M & -I \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_N \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} -A x_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{d}$$

$$\Rightarrow Cz = d$$

- Now we can write the LQR problem as a standard QP:

$$\min_z \ \frac{1}{2} z^T H z$$
$$\text{s.t.} \quad Cz = d$$

- The Lagrangian of this QP is:

$$L(z, \lambda) = \frac{1}{2} z^T H z + \lambda^T (Cz - d)$$

- KKT Conditions

$$\nabla_z L = Hz + C^T \lambda = 0$$
$$\nabla_\lambda L = Cz - d = 0$$

$$\Rightarrow \begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

- We get the exact solution by solving one linear system!

* Example:

  - Double integrator
  - Much better than shooting!

* A closer look at the LQR QP:

- The KKT system for LQR is very sparse (lots of zeros) and has lots of structure.

$$\begin{bmatrix} R & & & & B^T & & & & \\ & Q & & & -I & A^T & & & \\ & & R & & & B^T & & & \\ & & & Q & & -I & A^T & & \\ & & & & R & & B^T & & \\ & & & & & Q_N & -I & & \\ \hline B & -I & & & & & & & \\ A & B & -I & & & & 0 & & \\ & & A & B & -I & & & & \end{bmatrix} \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \hline \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ \\ \\ \\ \\ \hline -Ax_1 \\ 0 \\ 0 \end{bmatrix}$$

$$Q_N x_4 - \lambda_4 = 0 \Rightarrow \lambda_4 = Q_N x_4$$

$$R u_3 + B^T \lambda_4 = R u_3 + B^T Q_N x_4 = 0$$

(Plug in dynamics for x_4)

$$\Rightarrow R u_3 + B^T Q_N (A x_3 + B u_3) = 0$$

$$\Rightarrow u_3 = -\underbrace{(R + B^T Q_N B)^{-1} B^T Q_N A}_{K_3} x_3$$

$$Q x_3 - \lambda_3 + A^T \lambda_4 = 0$$

(Plug in λ_4)

$$Q x_3 - \lambda_3 + A^T Q_N x_4 = 0$$

(Plug in dynamics)

$$Q x_3 - \lambda_3 + A^T Q_N (A x_3 + B u_3) = 0$$

(Plug in $u_3 = -K_3 x_3$)

$$\lambda_3 = \underbrace{(Q + A^T Q_N (A - B K_3))}_{P_3} x_3$$

- Now we have a recursion for K and P:

$$P_N = Q_N$$

$$K_n = (R + B^T P_{n+1} B)^{-1} B^T P_{n+1} A$$

$$P_n = Q + A^T P_{n+1} (A - B K_n)$$

- This is called the Riccati equation / recursion
- We can solve the QP by doing a backward Riccati recursion followed by a forward rollout to compute $x_{1:N}$ and $u_{1:N-1}$
- General (dense) QP has complexity $O(N^3(n^3+m^3))$
  - time horizon
  - state dim.
  - control dim.
- Riccati solution is $O(N(n^3+m^3))$
- Even more important: we now have a feedback policy $u = -Kx$ instead of an open-loop trajectory

* Example
  - Riccat solution exactly matches QP
  - Feedback policy lets us change x_0 and reject noise / disturbances