Last Time:

- Hybrid methods for legged locomotion

Today:

- Iterative Learning Control

---

* What happens when our model has errors?
  - Models are approximate
  - Simpler models are often preferred even if they're less accurate
  - Feedback (e.g. LQR/MPC) can often compensate for model errors
  - Sometimes that isn't enough (e.g. very tight constraints, performance, safety).

* Several Options:

1) Parameter Estimation: Classical "system ID" / "grey-box" modeling. Fit e.g. masses, lengths, etc. in your model from data.

   + Very sample efficient
   + Generalizes well
   + Interpretable
   - Assumes model structure

2) Learn Model: Fit generic function approximator to the full dynamics or residual. Classical "black-box" modeling / System ID

+ Doesn't assume model structure
+ Generalizes
- Not sample efficient. Requires lots of data

Improve the model

3) Learn a Policy: Standard model-free RL approach: Optimize a function approximation of the controller

+ Makes few assumptions
- Doesn't generalize
- Not sample efficient. Requires lots of "rollouts"

4) "Transfer": Assume we have a reference trajectory computed with a nominal model. Improve it with data from the real system.

+ Makes few assumptions
- Assumes a decent prior model
- Doesn't generalize (task specific)
+ Very sample efficient

Improve the controller

Iterative Learning Control (ILC)

- Can think of this as a very specialized policy-gradient method on the policy class:

$$u_n(\bar{x}_n) = \bar{u}_n - K_n(x_n - \bar{x}_n)$$

($\bar{u}_n$: reference inputs; K_n: any tracking controller; $\bar{x}_n$: reference states)

* we only update $\bar{u}_n$

- Can think of this as SQP where we get the RHS vector from a rollout on the real system.

- Assume we have a reference trajectory $\bar{x}, \bar{u}$ that we want to track:

$$\min_{\substack{x_{1:N} \\ u_{1:N-1}}} J = \sum_{n=1}^{N-1} \frac{1}{2}(x_n - \bar{x}_n)^T Q (x_n - \bar{x}_n) + \frac{1}{2}(u_n - \bar{u}_n)^T R (u_n - \bar{u}_n) + \frac{1}{2}(x_N - \bar{x}_N)^T Q (x_N - \bar{x}_N)$$

$$\text{s.t.} \quad x_{n+1} = f(x_n, u_n)$$

(other constraints)

- The KKT system for this problem looks like:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla J \\ -c(z) \end{bmatrix}$$

where $\Delta z = \begin{bmatrix} \Delta x_1 \\ \Delta u_1 \\ \vdots \\ \Delta x_N \end{bmatrix}$, $c(z) = \begin{bmatrix} \vdots \\ f(x_n, u_n) - x_{n+1} \\ \vdots \end{bmatrix}$, $C = \frac{\partial c}{\partial z}$

($f(x_n, u_n)$: nominal dynamics model)

$$H = \begin{bmatrix} Q & & \\ & R & \\ & & Q_N \end{bmatrix}$$ ← Gauss-Newton Hessian

- Two important observations:

  1) If we do a rollout on the real system, $c(z) = 0$ always (for the true dynamics)

  2) Since we know J, given x_n, u_n from a rollout, we can compute ∇J

- Now we have RHS vector from the KKT system

- We also know H from the cost

- We can compute $C = \frac{\partial c}{\partial z}$ using x_n, u_n and the nominal model

- However, since our nominal model is approximate and assuming x_n, u_n is already close to $\bar{x}, \bar{u}$, we can just use $C = \frac{\partial c}{\partial z}|_{\bar{x}, \bar{u}}$, which can be computed offline

- Now just solve KKT system for δz, update: $\bar{u} \leftarrow \bar{u} + \delta u$, and repeate

- Can easily add inequality constraints (e.g. torque limits) and solve a QP

* ILC Algorithm:

- Given nominal $\bar{x}, \bar{u}$

do:

$$x_{1:N}, u_{1:N-1} \leftarrow \underbrace{\text{rollout}(x_0, \bar{u})}_{\text{on real system}}$$

$$\delta x, \delta u \leftarrow \underbrace{\left\{ \begin{array}{l} \text{argmin } J(\delta x, \delta u) \\ \text{s.t. } \delta x_{n+1} = A_n \delta x_n + B_n \delta u_n \\ u_{min} \le u_n \le u_{max} \end{array} \right.}_{\text{This is a QP}}$$

$$\bar{u} \leftarrow \bar{u} + \delta u$$

while $\underbrace{\|x_N - \bar{x}_N\| \ge tol}_{\text{whatever you care about}}$

* Why should ILC work?

- We've already seen approximations of Newton's method (e.g. Gauss-Newton)

- In general these are called "quasi-Newton" or "inexact Newton" methods. Many variants e.g. BFGS / Newton-CG. Well developed theory.

- For a generic root-finding problem:

$$f(x+\delta x) \approx \underbrace{f(x) + \frac{\partial f}{\partial x}\delta x}_{\text{exact Newton step}} = 0$$

- As long as δx satisfies:

  $\|f(x) + J\delta x\| \leq \eta \|f(x)\|$ for some $\eta < 1$,
  an inexact Newton method will converge

- Convergence is slower than exact Newton

- This means we can use $J \approx \frac{\partial f}{\partial x} \neq \frac{\partial f}{\partial x}$ to compute δx