Last Time:

- Hybrid methods for legged locomotion

Today:

- Review: Convex vs. Non-Convex Optimization
- Iterative Learning Control

---

* Convex vs. Non-Convex Optimization:
  - Convex Optimization:
    - Minimize a convex objective function over a convex set (convex constraints)
    - Generally means linear equalities and linear and/or conic inequalities.
  - Non-Convex Optimization
    - Everything Else

| | Convex | Non-Convex |
|---|---|---|
| Need a Good Initial Guess | ✗ | ✓ |
| Global Optimality Guarantee | ✓ | ✗ |
| Local Convergence Guarantee | ✓ | ✗ |
| Infeasibility Certificate | ✓ | ✗ |
| Bounded Solution time | ✓ | ✗ |

* 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 in your model from data.

   + Very sample efficient
   + Generalizes well
   - Assumes model structure

2) Learn Model: Fit a 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 RL approach: Optimize a function approximation for control policy.

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

4) Improve a trajectory: Assume we have a reference 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:

$$u_n(\bar{u}_n) = \underbrace{\bar{u}_n}_{\text{reference inputs}} - \underbrace{K_n(x_n - \bar{x}_n)}_{\text{can be any tracking controller}}$$

where we are updating $\bar{u}_n$.

- 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:

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

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

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

$$H = \begin{bmatrix} Q & & & \\ & R & & \\ & & \ddots & \\ & & & Q_N \end{bmatrix}$$ ← 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 $\nabla J$

- Now we have RHS vector for the KKT system.
- We also know it from the cost
- We can compute $C = \frac{\partial c}{\partial z}$ using $x_u, u_u$ and the nominal model.
- However, since our nominal model is approximate and assuming $x_u, u_u$ is already close to $\bar{x}, \bar{u}$, we can just use $C = \frac{\partial c}{\partial z}\Big|_{\bar{x},\bar{u}}$, which can be computed offline.
- Now just solve KKT system for $\delta z$, 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 trajectory $\bar{x}, \bar{u}$

do:

$x_{1:N}, u_{1:N-1} \leftarrow$ rollout$(\bar{x}_0, \bar{u})$ ← note can be different from $\bar{x}$ due to tracking controller

↳ on real system

$\delta x, \delta u \leftarrow$ argmin $J(\delta x, \delta u)$

This is a QP

s.t. $\delta x_{n+1} = A_n \delta x + B_n \delta u_n$

$u_{min} \le u_n \le u_{max}$

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

while $\|x_n - \bar{x}_n\| \ge$ tol

(many options)

* Why Should ILC Work?

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

- In general, these are called "inexact" and/or "quasi-Newton" methods. Many variants (BFGS, Newton-CG), well developed theory.

- For a generic root-finding problem:

$f(x+\delta x) \approx f(x) + \frac{\partial f}{\partial x}\delta x = 0$

exact Newton step

- As long as $\delta x$ 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 $J \approx \frac{\partial f}{\partial x}$ to compute $\delta x$