

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 convex 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:

$$U_n(\bar{U}_n) = \underbrace{\bar{U}_n - K_n(X_n - \bar{X}_n)}_{\substack{\text{reference} \\ \text{inputs}}}$$

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:

$$\begin{array}{ll} \min_{\substack{x_{1:N} \\ u_{1:N}}} & J = \sum_{n=1}^N \frac{1}{2} (x_n - \bar{x}_n)^T Q (x_n - \bar{x}_n) + \frac{1}{2} (u_n - \bar{u}_n)^T R (u_n - \bar{u}_n) \\ & + \frac{1}{2} (x_0 - \bar{x}_0)^T Q_0 (x_0 - \bar{x}_0) \\ \text{s.t. } & x_{n+1} = f(x_n, u_n) \end{array}$$

- The KKT system for this problem looks like:

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

where $0z = \begin{bmatrix} 0x_1 \\ 0u_1 \\ \vdots \\ 0x_N \end{bmatrix}$, $C(z) = \begin{bmatrix} \vdots \\ f(x_N, u_N) - x_N \\ \vdots \end{bmatrix}$, $C = \frac{\partial}{\partial z}$

$$H = \begin{bmatrix} Q & & \\ & R_{11} & \\ & \ddots & \\ & & Q_N \end{bmatrix} \quad \leftarrow \text{Gauss-Newton Hessian}$$

- 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 for the KKT system.
- We also know it 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 repeat
- Can easily add inequality constraints (e.g. torque limits) and solve a QP.

* ILC Algorithm:

- Given nominal trajectory \bar{X}, \bar{U}

do:

$$X_{1:N}, U_{1:N} \leftarrow \underbrace{\text{rollout}(\bar{X}_0, \bar{U})}_{\text{on real system}}$$

note can be different from it due to tracking controller

$$\Delta X, \Delta U \leftarrow \underset{\text{Q.P}}{\text{argmin}} \mathcal{J}(X, U)$$

$$\begin{cases} \text{s.t. } \Delta X_{n+1} = A_n \Delta X + B_n \Delta U \\ U_{\min} \leq U_n \leq U_{\max} \end{cases}$$

$$\bar{U} \leftarrow \bar{U} + \Delta U$$

$$\text{while } \|X_n - \bar{X}_n\| \geq \text{tol}$$

(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 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 $J \approx \frac{\partial f}{\partial x}$ to compute ΔX