16-745 Optimal Control Lecture 18

Reid Graves

March 25, 2025

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	X	✓
Global optimality guarantee	✓	X
Local convergence guarantee	✓	X
Infeasibility Certificate	✓	X
Bounded solution time	✓	X

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:

- improve the model:
  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
     - - 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 controller
  1. **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"
  2. **"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

Iterative Learning Control (ILC)

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

$$u_k(\bar{u}_k) = \underbrace{\bar{u}_k}_{\text{reference inputs}} - \underbrace{K_k(x_k - \bar{x}_k)}_{\text{can be any tracking controller}}$$

  *We only update $\bar{u}_k$

- 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:

$$\min_{x_{1:N}, u_{1:N-1}} \quad J = \sum_{n=1}^{N-1} \frac{1}{2}(x_k - \bar{x}_k)^T Q (x_k - \bar{x}_k) + \frac{1}{2}(u_k - \bar{u}_k)^T R (u_k - \bar{u}_k) + \frac{1}{2}(x_N - \bar{x}_N)^T Q_N (x_N - \bar{x}_N)$$
$$s.t. \quad x_{k+1} = f(x_k, u_k)$$

- 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:

$$\Delta z = \begin{bmatrix} \Delta x_1 \\ \Delta u_1 \\ \vdots \\ \Delta x_N \end{bmatrix} \qquad c(z) = \underbrace{\begin{bmatrix} \vdots \\ f(x_k, u_k) - x_{k+1} \\ \vdots \end{bmatrix}}_{\text{nominal dynamics model}} \qquad C = \frac{\partial c}{\partial z}$$

$$H = \begin{bmatrix} Q & & & \\ & R & & \\ & & \ddots & \\ & & & Q_N \end{bmatrix} \leftarrow \text{Gauss-newton hessian}$$

- **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_k, u_k from 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_k, u_k and the nominal model
- However, since our nominal model is approximate and assuming x_k, u_k 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:

$$x_{1:N}, u_{1:N-1} \leftarrow \underbrace{rollout(\bar{x}_0, \bar{u})}_{\text{on real system}}$$

$u_{1:N-1}$ can be different from $\bar{u}$ due to tracking controller

This is a QP:

$$\Delta x, \Delta u \leftarrow \text{argmin } J(\Delta x, \Delta u)$$
$$s.t. \Delta x_{k+1} = A_k \Delta x + B_k \Delta u_k$$
$$u_{min} \leq u_k \leq u_{max}$$
$$\bar{u} \leftarrow \bar{u} + \Delta u$$

while $\underbrace{||x_N - \bar{x}_N|| \geq \text{tol}}_{\text{(many options)}}$

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:

$$f(x + \Delta x) \underbrace{f(x) + \frac{\partial f}{\partial x}\Delta x = 0}_{\text{exact Newton Step}}$$

- As long as Δx satisfies:

$$||f(x) + J\Delta x|| \leq \eta ||f(x)|| \text{ for some } \eta < 1,$$
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