16-745 Optimal Control Lecture 13

Reid Graves

March 3, 2025

Last Time:

- DDP details
- Constraints

Today

Other domain of nonlinear control- Direct methods. (Before did indirect/shooting). Indirect optimizes first, then discretizes. Direct discretizes first, then optimizes.

- Minimum / Free-Time problems
- Direct Trajectory Optimization
- Direct Collocation
- Sequential Quadratic Programming

Handling Free / Minimum-Time Problems

free time- instead of defining time horizon, let the controller figure it out. The Minimum time problem is:

$$\min_{x(t),u(t),T_f} \quad J = \int_0^{T_f} 1\, dt$$

subject to:

$$\dot{x} = f(x,u)$$

Add goal constraint- need to get to goal:

$$x(T_f) = x_{\text{goal}}$$

Add input constraints to ensure well posed.

$$u_{\min} \leq u(t) \leq u_{\max}$$

- We don't want to change the number of knot points. So change length of each timestep instead of number of knot points.

- Make h (time step) from RK a control input:

$$x_{k+1} = f_{\text{RK4}}(x_k, \bar{u}_k), \quad \bar{u}_k = \begin{bmatrix} u_k \\ h_k \end{bmatrix}$$

- Also want to scale the cost by h, e.g.,

$$J(x, u) = \sum_{u=1}^{k} h_k l(x_k, u_k) + l_N(x_N)$$

- Requires constraints on h. Otherwise, the solver can "cheat physics" by making h very large or negative to exploit discretization errors.
- Always nonlinear/nonconvex, even if the dynamics are linear. h is multiplying the dynamics, so get quadratic terms in dynamics, making problem nonlinear.

Direct Trajectory Optimization

- **Basic strategy:** Discretize / "transcribe" continuous-time optimal control problem into a standard nonlinear program (NLP):

  "Standard" NLP

$$\begin{aligned} \min_x \quad & f(x) \quad \text{(cost function)} \\ \text{s.t.} \quad & c(x) = 0 \quad \text{(dynamics constraints)} \\ & d(x) \leq 0 \quad \text{(other constraints- actuator limits, etc)} \end{aligned}$$

- All functions (f, c, d) assumed C^2 smooth.
- Lots of off-the-shelf solvers for large-scale NLP.
- Most common:
  - IPOPT (free)
  - SNOPT (commercial)
  - KNITRO (commercial)
- Common solution strategy: Sequential Quadratic Programming (SQP)-this is what SNOPT does. Also most common technique for nonlinear MPC.

SQP: Sequential Quadratic Programming

- **Strategy:** use 2^{nd}-order Taylor expansion of the Lagrangian and linearize $c(x)$, $d(x)$ to approximate the NLP as a QP:

$$\min_{\Delta x} \quad f(x) + g^T \Delta x + \frac{1}{2} \Delta x^T H \Delta x$$

  subject to:

$$C(x) + C\Delta x = 0$$

$$d(x) + D\Delta x \leq 0$$

where:

$$H = \frac{\partial^2 L}{\partial x^2}, \quad g = \frac{\partial L}{\partial x}, \quad C = \frac{\partial C}{\partial x}, \quad D = \frac{\partial d}{\partial x}$$

$$L(x, \lambda, \mu) = f(x) + \lambda^T c(x) + \mu^T d(x)$$

- Solve QP to compute primal-dual search direction:

$$\Delta z = \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \end{bmatrix}$$

- Perform line search with merit function.
- With only equality constraints, reduces to Newton's method on KKT conditions:

$$\underbrace{\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix}}_{\text{KKT System}} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -g \\ -c(x) \end{bmatrix}$$

- Think of SQP as a generalization of Newton's method to handle **inequalities**.
- Can use any QP solver for sub-problems, but good implementations typically warm start using the previous QP iteration.
- For good performance on trajectory optimization problems, taking advantage of sparsity in KKT systems is crucial.
- If inequalities are convex (e.g., conic), we can generalize SQP to **SCP** (Sequential Convex Programming), where inequalities are passed directly to the sub-problem solver.
- SCP is still an active research area.

Direct Collocation

Direct collocation is gold standard direct method.

- So far, we've used explicit **RK** methods:

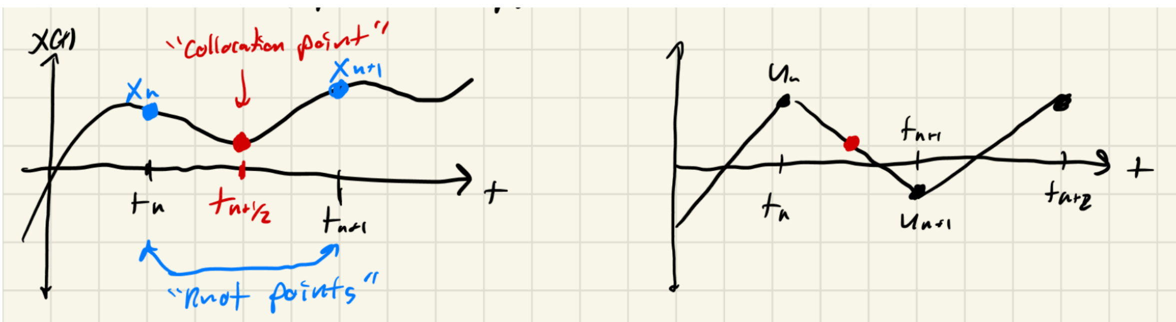
$$\dot{x} = f(x, u) \quad \Rightarrow \quad x_{k+1} = f(x_k, u_k)$$

- This makes sense if you're doing rollout.

- However, in a direct method, we're just enforcing dynamics as **equality constraints** between knot points:

$$C_k(x_k, u_k, x_{k+1}, u_{k+1}) = 0$$

- $\Rightarrow$ Implicit integration is "free" in this formulation.
- Collocation methods use polynomial splines to represent trajectories and enforce dynamics as constraints on spline derivatives
- Classic **DIRCOL** algorithm uses cubic splines for states and piecewise linear interpolation for $u(t)$.
- Very high-order polynomials are sometimes used (e.g., spacecraft trajectories), but this is not common.
- **DIRCOL Spline Approximations:**



$$x(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3$$

$$\dot{x}(t) = C_1 + 2C_2 t + 3C_3 t^2$$

Matrix Formulation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} x_k \\ \dot{x}_k \\ x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix}$$

Inverted System

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{h^2} & -\frac{2}{h} & \frac{3}{h^2} & -\frac{1}{h} \\ \frac{2}{h^3} & \frac{1}{h^2} & -\frac{2}{h^3} & \frac{1}{h^2} \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

- Evaluate at $t_{k+\frac{1}{2}}$

$$x_{k+\frac{1}{2}} = x(t_k + \frac{h}{2}) = \frac{1}{2}(x_k + x_{k+1}) + \frac{h}{8}(\dot{x}_k - \dot{x}_{k+1})$$

$$= \frac{1}{2}(x_k + x_{k+1}) + \frac{h}{8}\big(\underset{\uparrow}{f(x_k, u_k)} - \underset{\uparrow}{f(x_{k+1}, u_{k+1})}\big)$$

(Continuous-time dynamics)

$$\dot{x}_{k+\frac{1}{2}} = \dot{x}(t_k + \frac{h}{2}) = -\frac{3}{2h}(x_k - x_{k+1}) - \frac{1}{4}(\dot{x}_k + \dot{x}_{k+1})$$

$$= -\frac{3}{2h}(x_k - x_{k+1}) - \frac{1}{4}\big(f(x_k, u_k) + f(x_{k+1}, u_{k+1})\big)$$

$$u_{k+\frac{1}{2}} = u(t_k + \frac{h}{2}) = \frac{1}{2}(u_k + u_{k+1})$$

- We can enforce Dynamics Constraints

$$C_i(x_k, u_k, x_{k+1}, u_{k+1}) =$$

$$\underset{\uparrow}{f(x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}})} - \left[-\frac{3}{2h}(x_k - x_{k+1}) - \frac{1}{4}\big(f(x_k, u_k) + f(x_{k+1}, u_{k+1})\big)\right] = 0$$

(Continuous dynamics)

- Note that only x_k, u_k are decision variables (not $x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}}$).
- Called **"Hermite-Simpson" integration**.
- Achieves **3rd order** integration accuracy like RK3.
- Requires **fewer dynamics calls** than explicit RK3!

Explicit RK3

$$f_1 = f(x_k, u_k)$$

$$f_2 = f(x_k + \frac{1}{2}hf_1, u_k)$$

$$f_3 = f(x_k + 2hf_1 - hf_2, u_k)$$

$$x_{k+1} = x_k + \frac{h}{6}(f_1 + 4f_2 + f_3)$$

$\Rightarrow$ **3 dynamics evaluations per time step**

Hermite-Simpson Integration

$$f(x_{k+\frac{1}{2}}, u_{k+\frac{1}{2}}) + \frac{3}{2h}(x_k - x_{k+1})$$

$$-\frac{1}{4}\big(f(x_k, u_k) + f(x_{k+1}, u_{k+1})\big) = 0$$

$$\uparrow \qquad\qquad\qquad \uparrow$$

These get re-used at adjacent steps!

$\Rightarrow$ **Only 2 dynamics calls per time step!**

- Since dynamics calls often dominate total compute cost, this results in a $\sim$**50% savings**.

Example

- Acrobot with **DIRCOL**
- Warm-starting with dynamically infeasible guesses can help a lot!