Last Time:

- DDP details
- Constraints

Today:

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

---

* Handling Free / Minimum-Time Problems

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

$$\text{s.t.} \quad \dot{x} = f(x,u)$$

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

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

Minimum-time Problem

- We don't want to change the number of knot points
- Make h (time step) from RK a control input

$$x_{n+1} = f_{RK4}(x_n, \tilde{u}_n), \quad \tilde{u}_n = \begin{bmatrix} u_n \\ h_n \end{bmatrix}$$

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

$$J(x,u) = \sum_{n=1}^{N-1} h_n \, l(x_n, u_n) + l_N(x_N)$$

- Always nonlinear / nonconvex even if the dynamics are linear

- Requires constraints on h. Otherwise the solver can "cheat physics" by making h very large or negative to exploit discretization errors.

Direct Traj Opt

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

"Standard" NLP

$$\begin{aligned} \min_{x} \quad & f(x) \quad \leftarrow \text{cost function} \\ \text{s.t.} \quad & c(x) = 0 \quad \leftarrow \text{dynamics constraints} \\ & d(x) \leq 0 \quad \leftarrow \text{other constraints} \end{aligned}$$

- All functions 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)

* SQP:

- Strategy: Use 2nd-ord Taylor expansion of the Lagrangian and linearize $c(x)$, $d(x)$ to approximate the NLP as a QP:

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

$$\text{s.t.} \quad c(x) + C\Delta x = 0$$

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

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

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

- 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

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

- Think of SQP as a generalization of Newton to handle inequalities.

- Can use any QP solver for sub-problems but good implementations typically warm start using previous QP iteration

- For good performance on traj opt problems, taking advantage of sparsity in KKT systems is crucial.

- If inequalities are convex (e.g. conic) 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

- So far we've used explicit RK methods!

$$\dot{x} = f(x,u) \quad \rightarrow \quad x_{n+1} = f(x_n, u_n)$$

- This makes sense if you're doing rollout

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

$$c_n(x_n, u_n, x_{n+1}, u_{n+1}) = 0$$

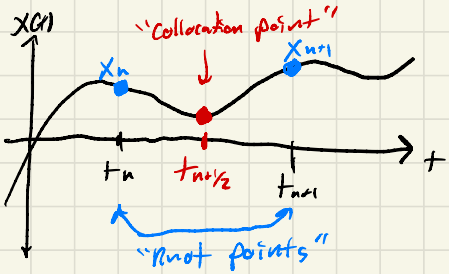
$\Rightarrow$ implicit integration is "free"

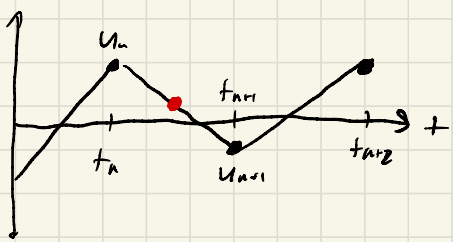
- Collocation methods represent trajectories as polynomial splines and enforce dynamics on spline derivatives.

- 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 not common.

- DIRCOL Spline Approximations:





$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

$$\Downarrow$$

$$\dot{x}(t) = c_1 + 2c_2 t + 3c_3 t^2$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_n \\ \dot{x}_n \\ x_{n+1} \\ \dot{x}_{n+1} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3/h^2 & -2/h & 3/h^2 & -1/h \\ 2/h^3 & 1/h^2 & -2/h^3 & 1/h^2 \end{bmatrix} \begin{bmatrix} x_n \\ \dot{x}_n \\ x_{n+1} \\ \dot{x}_{n+1} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

- Evaluate at $t_{n+1/2}$:

$$x_{n+1/2} = x(t_n + h/2) = \frac{1}{2}(x_n + x_{n+1}) + \frac{h}{8}(\dot{x}_n - \dot{x}_{n+1})$$

$$= \frac{1}{2}(x_n + x_{n+1}) + \frac{h}{8}(f(x_n, u_n) - f(x_{n+1}, u_{n+1}))$$

(continuous-time dynamics)

$$\dot{x}_{n+1/2} = \dot{x}(t_n + 1/2) = \frac{-3}{2h}(x_n - x_{n+1}) - \frac{1}{4}(\dot{x}_n + \dot{x}_{n+1})$$

$$= \frac{-3}{2h}(x_n - x_{n+1}) - \frac{1}{4}(f(x_n, u_n) + f(x_{n+1}, u_{n+1}))$$

$$u_{n+1/2} = u(t_n + h/2) = \frac{1}{2}(u_n + u_{n+1})$$

- We can enforce dynamics constraints!

$$c_i(x_n, u_n, x_{n+1}, u_{n+1}) =$$

$$f(x_{n+1/2}, u_{n+1/2}) - \left[\frac{-3}{2h}(x_n - x_{n+1}) - \frac{1}{4}(f(x_n, u_n) + f(x_{n+1}, u_{n+1}))\right]$$

(f: continuous dynamics)

$$= 0$$

- Note that only x_n, u_n are decision variables (not $x_{n+1/2}$, $u_{n+1/2}$)
- Called "Hermite-Simpson" integration
- Achieves 3rd order integration accuracy like RK3
- Requires fewer dynamics calls than explicit RK3!

Explicit RK3:

$$f_1 = f(x_n, u_n)$$
$$f_2 = f(x_n + \tfrac{1}{2}hf_1, u_n)$$
$$f_3 = f(x_n + 2hf_1 - hf_2, u_n)$$

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

$\Rightarrow$ 3 dynamics evals per time step

Hermite Simpson:

$$f(x_{n+1/2}, u_{n+1/2}) + \tfrac{3}{2h}(x_n - x_{n+1}) - \tfrac{1}{4}(f(x_n, u_n) + f(x_{n+1}, u_{n+1})) = 0$$

These get re-used at adjacent steps!

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

- Since dynamics calls often dominate total compute cost, this is a ~50% savings!

* Example:
  - Acrobot w/ DIRCOL
  - Warm starting with dynamically infeasible guess can help a lot!