Last Time:

- DDP details/extensions
- Constraints

Today

- Minimum/free time w/ DDP
- Direct Traject optimization
- Sequential Quadratic Programming
- Direct Collocation

---

* Notes on Linearization w/ LQR/MPC

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

$$\Downarrow$$

$$x_{n+1} + \delta x_{n+1} \simeq f(x_n, u_n) + A\delta x_n + B\delta u_n$$

($A = \frac{\partial f}{\partial x}$, $B = \frac{\partial f}{\partial u}$)

$$\Downarrow$$

$$\delta x_{n+1} = A\delta x_n + B\delta u_n$$

- When applying eg. LQR:

$$\delta u = -K\delta x$$

$$\Downarrow$$

$$u = u_0 - K(x - x_0)$$

✱ Handling Free / Minimum Time Problems:

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

$$\text{s.t.} \quad \dot{x} = f(x, u)$$
$$x(T) = x_{goal}$$
$$u_{min} \le u(t) \le u_{max}$$

minimum time Problem

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

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

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

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

- Always nonlinear / nonconvex even if the continuous dynamics are linear.
- Requires constraints on h, otherwise the solver can "cheat physics" by making h very large or negative to exploit discretization error.

Direct Traj Opt

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

"Standard form" NLP

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

- Several off-the-shelf large-scale NLP solvers can be used to solve the problem
- Most common solvers: IPOPT (free), SNOPT (commercial), KNITRO (commercial).
- Common solution strategy: Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming

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

$$\begin{aligned} \min_{\delta x} \quad & \tfrac{1}{2}\delta x^T H \delta x + g^T \delta x \\ \text{s.t.} \quad & c(x) + C\delta x = 0 \\ & d(x) + D\delta x \leq 0 \end{aligned}$$

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

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

- 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 method on the Lagrangian:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -\frac{\partial \mathcal{L}}{\partial x} \\ -c(x) \end{bmatrix}$$

"KKT System"

- Think of SQP as generalization of Newton to inequality constrained problems.
- Can use any QP solver to solve sub-problems, but good implementations typically use active-set strategies with warm-starting tricks.
- For good performance on traj opt problems, taking advantage of sparsity in KKT system is crucial. SNOPT (sparse nonlinear optimizer) does this well.

- If inequalities are convex (e.g. conics) 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 to discretize dynamics:

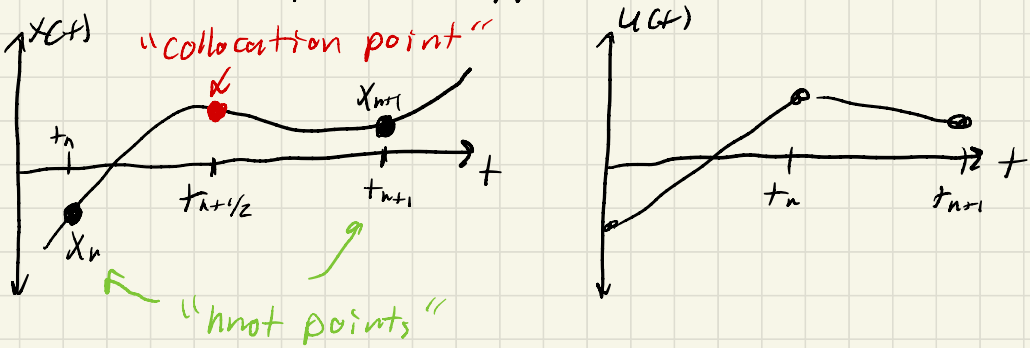
$$\dot{x} = f(x, u) \rightarrow x_{n+1} = f(x_n, u_n)$$

  - This makes sense if you're doing rollouts
  - However in a direct method where dynamics are enforced as equality constraints between knot points, this is not an advantage:

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

  - Collocation methods represent trajectories with polynomial splines and enforce dynamics on the spline derivatives
  - Classic DIRCOL algorithm uses cubic splines for state trajectories and piecewise linear interpolation for u(t)
  - Very high-order polynomials are sometimes used (e.g. spacecraft trajectories) but not common.

- DFRCOL 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 $t_{n+1/2}$

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

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

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

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

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

- We can now enforce dynamics constraints:

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

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

$$= 0$$

Continuous dynamics

- Note that only x_k, u_k are decision variables (not $x_{k+1/2}$, $u_{k+1/2}$)
- Called "Hermite-Simpson" integration.
- Achieves 3rd order 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 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 evals per time step!

- Since dynamics calls often dominate computational cost, this can be a ~50% savings.

* Example:

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