

---

---

---

---

---



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

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



$$\frac{\partial f}{\partial x}$$

$$\frac{\partial f}{\partial u}$$

$$\cancel{x_{n+1}} + \Delta x_{n+1} \approx f(x_n, u_n) \cancel{+ A \Delta x_n + B \Delta u_n}$$



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

- When applying e.g. LQR:

$$\Delta u = -K \Delta x$$



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

## \* Handling Free/Minimum Time Problems:

$$\begin{aligned} \min_{\substack{x(t) \\ u(t) \\ T}} \quad & J = \int_0^T 1 \, dt \\ \text{s.t.} \quad & \dot{x} = f(x, u) \\ & x(T) = x_{\text{goal}} \\ & U_{\min} \leq u(t) \leq U_{\max} \end{aligned}$$

} 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.

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

- 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):

$$\left. \begin{array}{l} \min_x f(x) \quad \text{← objective (cost) function} \\ \text{"standard form"} \\ \text{NLP} \end{array} \right\} \begin{array}{l} \text{s.t. } \begin{array}{l} c(x) = 0 \quad \text{← dynamics constraints} \\ d(x) \leq 0 \quad \text{← other constraints} \end{array} \end{array}$$

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

$$\min_{\Delta x} \frac{1}{2} \Delta x^T H \Delta x + g^T \Delta x$$

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

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

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

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

- Solve QP to compute primal-dual search direction:

$$\mathbf{Oz} = \begin{bmatrix} \mathbf{Ox} \\ \mathbf{O}\lambda \\ \mathbf{O}\mu \end{bmatrix}$$

- Perform line search with merit function
- With only equality constraints reduces to Newton method on the Lagrangian:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Ox} \\ \mathbf{O}\lambda \end{bmatrix} = \begin{bmatrix} -\frac{\partial L}{\partial x} \\ -C(x) \end{bmatrix}$$

“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 trj opt problems, taking advantage of sparsity in KKT system is crucial. SNOPT (sparse nonlinear optimizer) does this well.

- If inequalities are convex (e.g. cones), 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:

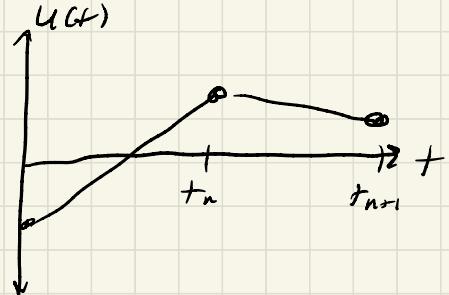
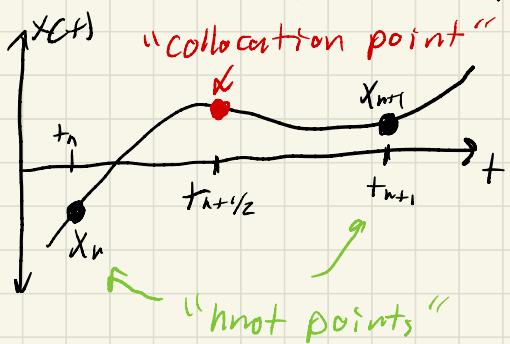
$$\dot{x} = f(x, u) \rightarrow x_{n+1} = f(x_n, u_n)$$

- 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!



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

↓

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} X_n \\ \dot{X}_n \\ X_{n+1} \\ \dot{X}_{n+1} \end{bmatrix}$$

↓

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3/h^2 & -2/h & 3/h^2 & -1/h \\ 2/h^3 & 1/h^2 & -2/h^3 & 1/h^2 \end{bmatrix} \begin{bmatrix} X_n \\ \dot{X}_n \\ X_{n+1} \\ \dot{X}_{n+1} \end{bmatrix} = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

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

$$\begin{aligned} X_{n+1/2} &= X(t_n + h/2) = \frac{1}{2}(X_n + X_{n+1}) + \frac{h}{8}(\dot{X}_n - \dot{X}_{n+1}) \\ &= \frac{1}{2}(X_n + X_{n+1}) + \frac{h}{8}(f(X_n, u_n) - f(X_{n+1}, u_{n+1})) \end{aligned}$$

$$\begin{aligned} \dot{X}_{n+1/2} &= \dot{X}(t_n + h/2) = -\frac{3}{2}h(X_n - X_{n+1}) - \frac{1}{4}(\dot{X}_n + \dot{X}_{n+1}) \\ &= -\frac{3}{2}h(X_n - X_{n+1}) - \frac{1}{4}(f(X_n, u_n) + f(X_{n+1}, u_{n+1})) \end{aligned}$$

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

- We can now enforce dynamics constraints:

$$C_0(X_n, u_n, X_{n+1}, u_{n+1}) =$$

$$f(X_{n+1/2}, U_{n+1/2}) - \left( -\frac{3}{2}h(X_n - X_{n+1}) - \frac{1}{4}(f(X_n, u_n) + f(X_{n+1}, u_{n+1})) \right) = 0$$

Continuous dynamics

- Note that only  $X_n, u_n$  are decision variables (not  $X_{n+1/2}, u_{n+1/2}$ )

- Called "Hermite-Simpson" integration.

- Achieves 3<sup>rd</sup> order accuracy like RK 3

- Requires fewer dynamics calls than explicit RK 3!

## Explicit RK3:

$$f_1 = f(x_n, u_n)$$

$$f_2 = f(x_n + \frac{1}{2}hf_1, u_n)$$

$$f_3 = f(x_n + 2hf_1 - hf_2, u_n)$$

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

$\Rightarrow$  3 dynamics evals per step

## Hermite-Simpson:

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

$$- \frac{1}{4}(f(x_n, u_n) + f(x_{n+1}, u_{n+1})) = 0$$



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% saving.

## \* Example:

- Acrobot w/ DRCOL

- Warm-starting with dynamically infeasible guess can help a lot!

