Last Time:

- DDP details
- Constraints

Today:

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

---

* Handling Free/Minimum-Time Problems:

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

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

} min time

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

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

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

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

- Requires constraints on h_n otherwise solver can "cheat physics" by making h very large or negative.
- Always nonlinear/nonconvex even if dynamics are linear

Direct Traj Opt

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

"Standard" NLP

$$\min_x \; f(x) \quad \leftarrow \text{cost function}$$
$$\text{s.t.} \quad c(x) = 0 \quad \leftarrow \text{"dynamics" constraints}$$
$$d(x) \le 0 \quad \leftarrow \text{other constraints}$$

- 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: Sequential Quadratic Programming (SQP)

SQP:

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

$$\min_{\delta x} \; f(x) + g^T \delta x + \frac{1}{2} \delta x^T H \delta x$$
$$\text{s.t.} \quad c(x) + C\delta x = 0$$
$$d(x) + D\delta x \le 0$$

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:

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

- Think of SQP as a generalization of Newton's method to handle inequalities

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

- For good performance on TrajOpt problems, taking advantage of sparsity in KKT systems is critical.

- If inequalities are convex (e.g. conic) can generalize SQP to SCP (sequential convex programming) where inequalities passed directly to the sub-problem solver.

Direct Collocation

- So far we've used explicit RK methods:

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

- This makes sense if you're doing a rollout

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

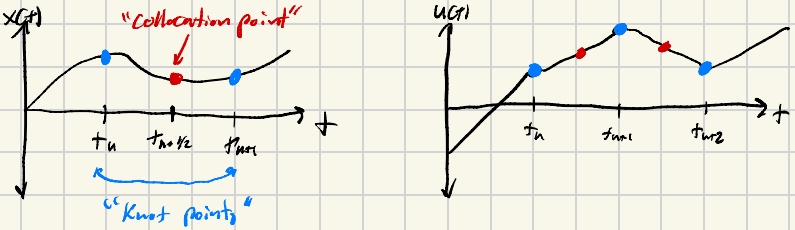
$$C_k(x_k, u_k, x_{k+1}, u_{k+1}) = 0$$

⇒ implicit integration is "free"

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

- Classic DIRCOL algorithm uses cubic splines for state and piecewise linear interpolation for u(t)

- DIRCOL spline approximations:



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

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

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

⇓

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

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

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

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

(continuous-time dynamics)

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

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

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

- We can enforce the dynamics constraints!

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

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

(continuous dynamics)

$$= 0$$

- 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 integration accuracy like RK3
- Requires fewer dynamics calls than explicit RK3!

- Explicit RK3:

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

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

$\Rightarrow$ 3 dynamics calls per time step

- Hermite-Simpson:

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

(these get reused at adjacent steps!)

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

- Since dynamics calls often dominate total compute cost, this is a ~30~50% savings!

* Example:

- Acrobot w/ DIRCOL

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