

Last Time:

- DDP details
- Constraints

Today:

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

\* Handling Free/Minimum-Time Problems:

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

} min time

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

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

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

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

- 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"  $\min_x f(x) \leftarrow \text{cost function}$   
 NLP  $\text{s.t. } \begin{array}{l} C(x) = 0 \\ d(x) \leq 0 \end{array} \leftarrow \begin{array}{l} \text{"dynamics" constraints} \\ \text{other constraints} \end{array}$

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

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

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

$$d(\bar{x}) + \bar{D} \Delta x \leq 0$$

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

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

- Solve QP to compute primal-dual search direction:

$$\delta z = \begin{bmatrix} 0x \\ 0\lambda \\ 0M \end{bmatrix}$$

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

$$\underbrace{\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} 0x \\ 0\lambda \end{bmatrix}}_V = \begin{bmatrix} -g(x, \lambda) \\ -Cx \end{bmatrix}$$

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.

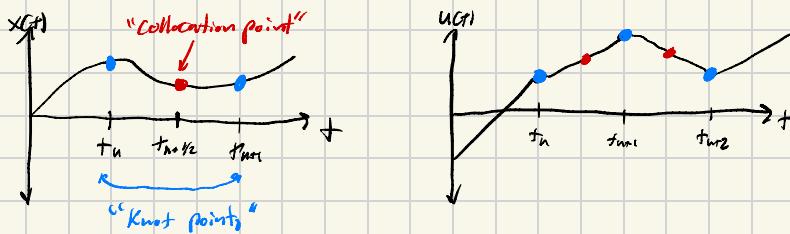
## \* Direct Collocation

- 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:
- $$C_a(x_n, \dot{x}_n, x_{n+1}, \dot{x}_{n+1}) = 0$$
- $\Rightarrow$  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 states and piecewise linear interpolation for u(t)
- DIRCOL Spline approximations:



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

$\Downarrow$

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

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

$\Downarrow$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{h^2} & -\frac{2}{h} & \frac{3}{h^2} & \frac{1}{h} \\ \frac{2}{h^3} & \frac{1}{h^2} & -\frac{2}{h^3} & \frac{1}{h^2} \end{bmatrix} \begin{bmatrix} x_u \\ \dot{x}_u \\ x_{unr} \\ \dot{x}_{unr} \end{bmatrix} = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

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

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

$\nwarrow$  continuous-time dynamics

$$\begin{aligned} \dot{x}_{n+1/2} &= \dot{x}(t_n + h/2) = -\frac{3}{h^2}(x_n - x_{unr}) - \frac{1}{4}(\dot{x}_n + \dot{x}_{unr}) \\ &= \frac{3}{2h}(x_n - x_{unr}) - \frac{1}{4}(f(x_n, u_n) + f(x_{unr}, u_{unr})) \end{aligned}$$

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

- We can enforce the dynamics constraints!

$$C_u(x_{unr}, x_{n+1/2}, u_{unr}) =$$

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

$\nwarrow$  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:

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

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

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

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

⇒ 3 dynamics calls per time step

## - Hermite-Simpson:

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

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

These get reused at adjacent steps!

⇒ 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!