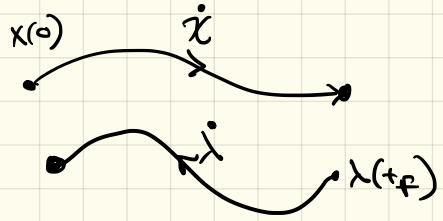
Lecture 13: Trajectory optimization via Shooting

Last time: Pontryagin's Principle



$$\min \int_0^{t_f} l(t, x, u)\, dt + l_f(x(t_f))$$

if $x^*(t), u^*(t)$ optimal $\exists \lambda(t)$, $\lambda(t_f) = \left[\left.\frac{\partial l_f}{\partial x}\right|_{x(t_f)}\right]^T$

① $\dot{\lambda} = -\nabla_x H(t, x^*(t), u^*(t), \lambda(t))$

② $\dot{x}^* = \nabla_\lambda H(t, x^*(t), u^*(t), \lambda(t))$

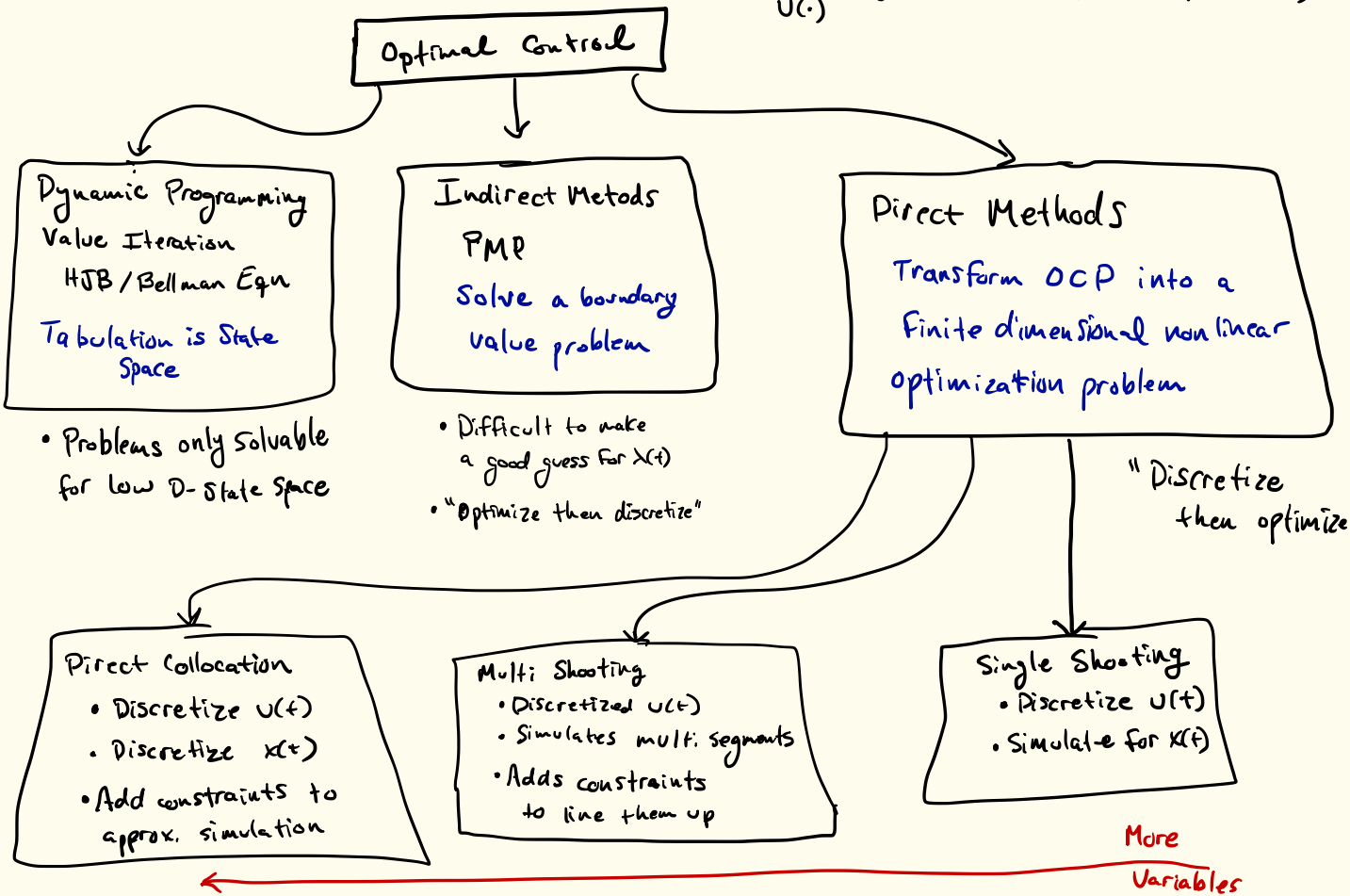
③ $u^* = \underset{\tilde{u}}{\operatorname{argmin}}\ H(t, x^*(t), \tilde{u}, \lambda(t))$

Today:

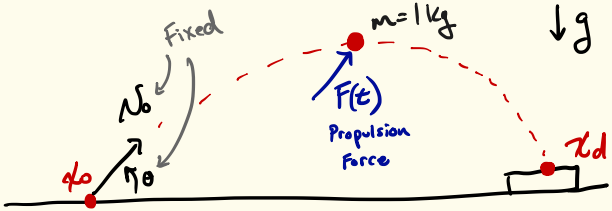
- A more "direct" way of finding optimal open-loop controls
- Trajectory optimization overview
- Shooting details
- Sensitivity analysis

Optimal Control Family Tree

$$\min_{u(\cdot)} \int_0^{t_f} \ell(t, x(t), u(t))dt + \ell_f(x(t_f))$$



Example: The rocket-propelled projectile.



State:

$$x = \begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix} \in \mathbb{R}^4$$

Dynamics:

$$\dot{x} = \begin{bmatrix} \text{velocity} \\ g + F(t) \end{bmatrix} := f(t, x, p)$$

Discretize Control:

$$u(t) = F(t) = \sum_{i=0}^{3} F_i t^i \qquad F_i \in \mathbb{R}^2$$

$$= F_0 + F_1 t + F_2 t^2 + F_3 t^3$$

$$p = \begin{bmatrix} F_0 ; F_1 ; F_2 ; F_3 \end{bmatrix} \in \mathbb{R}^8$$ Parameters for optimization

Finite Dimensional NLP:

$$\min_{p, t_f} \int_0^{t_f} \underbrace{\|F(t)\|^2}_{:= \ell(t, x, p)} dt$$

s.t. $x(0) = x_0$

$x(t_f, p) = x_d$ ← Evaluated by simulation.

Slight Change: Fixed limits of integration

Old Form:

$$\min_{p,t_f} \int_0^{t_f} l(t,x,p)\, dt$$

s.t. $x(0) = x_0$

$x(t_f, p) = x_d$

Evaluated with Simulation:

$\dot{x} = f(t, x, p)$

"Speeding up" the integration allows fixed limits of integ. without changing the final state/cost.

Change of variables: $p' = [p; t_f]$ $\quad t = t' \cdot t_f$

New Form:

$$\min_{p'} \int_0^1 \underbrace{t_f\, l(t', x(t', p'), p')}_{:= l'(t', x, p')}\, dt'$$

s.t. $x(0) = x_0$

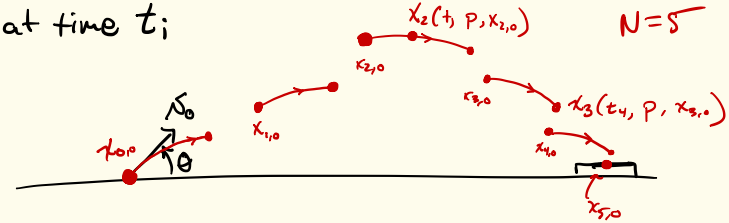
$x(1, p') = x_d$

Evaluated with Simulation:

$$\frac{d}{dt'} x = \underbrace{t_f\, f(t', x(t', p'), p')}_{:= f'(t', x, p')}$$

Multiple Shooting: N segments

- Consider $[0, t_1, \dots, t_{N-1}, t_F]$ $\quad t_i = \Delta t \cdot i \quad \Delta t = \frac{t_F}{N}$
- Let $x_{i,0} \in \mathbb{R}^n$ an optimization variable for the state at the beginning of segment i
- p discretized representation for control
- $x_i(t, p, x_{i,0})$ denotes the state @ time t when simulating from $x_{i,0}$ starting at time t_i



Multi-Shooting Formulation:

$$\min_{\{x_{i,0}\}_{i=0}^{N}, p, t_F} \sum_{i=0}^{N-1} \int_{t_i}^{t_i + \Delta t} l(t, x_i(t, p, x_{i,0}), p)\, dt$$

s.t. $x_{0,0} = x_0$, $x_{N,0} = x_d$

$$x_i(t_{i+1}, p, x_{i,0}) = x_{i+1,0} \qquad i = 0, 1, \dots, N-1$$

↖ Enforces continuity between segments

Note: The same tricks can be used to scale time in a MS formulation.

Multiple Shooting

- Simulations over smaller intervals
  - Less sensitive than simulations over long intervals
  - Leads to more variables
  - However, nonlinearities are less pronounced
    $\Rightarrow$ Often easier to optimize than single shooting
- Constraints
  - Constraints on controls are easily handled
  - Constraints on states at segment boundaries are easily handled
  - Constraints on states within segments are difficult to enforce...
- By Default, MATLAB evaluates derivatives via finite differences
  - Finite differences have roundoff/truncation error
  - Bad gradients $\Rightarrow$ Bad directions for optimizer to follow
  - Issues are especially pronounced for single shooting

Getting Better Gradients: Forward sensitivity analysis

$x \in \mathbb{R}^n \quad p \in \mathbb{R}^s$

$$x(0) = x_0(p) \qquad x(t) = \int_0^t f(t, x(t), p)\, dt$$

How does $x(t)$ change with p:

$$\frac{\partial x(t)}{\partial p} = \int_0^t \frac{\partial}{\partial p}\left[f(t, x(t), p)\right] dt = \int_0^t \left[\frac{\partial f}{\partial p} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial p}\right] dt$$

System of $n \times s$ ODEs for $\partial x(t)/\partial p$

$$\frac{d}{dt}\left[\frac{\partial x(t)}{\partial p}\right] = \frac{\partial f}{\partial p} + \frac{\partial f}{\partial x}\left[\frac{\partial x(t)}{\partial p}\right]$$

$$\frac{\partial x(0)}{\partial p} = \frac{\partial x_0}{\partial p}$$

Evaluating Gradients: Forward sensitivity analysis

Cost Function

$$L_f(p) = \int_0^{t_F} l(t, x, p)\, dt + l_f(x(t_F))$$

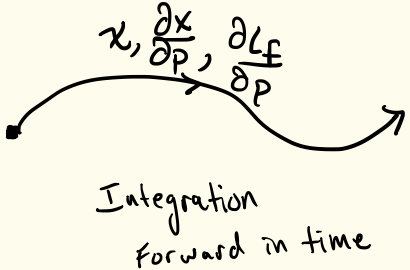
Gradient of cost:

$$\frac{\partial L_f}{\partial p} = \int_0^{t_F} \left[\frac{\partial l}{\partial p} + \frac{\partial l}{\partial x} \frac{\partial x}{\partial p} \right] dt + \frac{\partial l_f}{\partial x} \cdot \frac{\partial x(t_F)}{\partial p} \Leftarrow S \text{ integrals to compute}$$

Known from solution of ODEs on the previous slide

Can be computed analytically from $x(t), p$

Summary:



Overall:

- One pass fwd in time
- $nxs + s = (n+1)s$ integrals to finally compute

$$\frac{\partial L_f}{\partial p} \in \mathbb{R}^s$$

... costly when s large...

Summary

1. Shooting is a "direct" method for trajectory optimization
   - "discretize then optimize"
   - Single shooting is simplest, but intractable for complex systems and long horizons due to sensitivity issues
   - Multiple shooting addresses these issues
     - more variables
     - Nonlinear effects are less pronounced
2. Accurate gradients via
   - Forward mode sensitivity
   - Reverse mode sensitivity ~ Next time