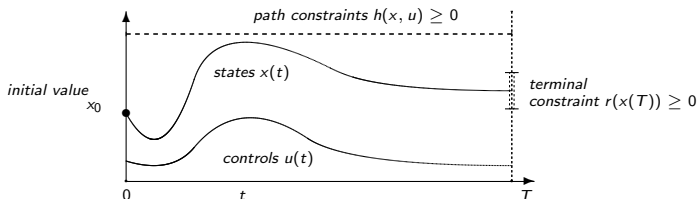


# Numerical Optimal Control

Moritz Diehl

# Simplified Optimal Control Problem in ODE



$$\text{minimize}_{x(\cdot), u(\cdot)} \int_0^T L(x(t), u(t)) dt + E(x(T))$$

subject to

$$x(0) - x_0 = 0, \quad (\text{fixed initial value})$$

$$\dot{x}(t) - f(x(t), u(t)) = 0, \quad t \in [0, T], \quad (\text{ODE model})$$

$$h(x(t), u(t)) \geq 0, \quad t \in [0, T], \quad (\text{path constraints})$$

$$r(x(T)) \geq 0 \quad (\text{terminal constraints})$$

# More general optimal control problems

Many features left out here for simplicity of presentation:

- ▶ multiple dynamic stages
- ▶ differential algebraic equations (DAE) instead of ODE
- ▶ explicit time dependence
- ▶ constant design parameters
- ▶ multipoint constraints  $r(x(t_0), x(t_1), \dots, x(t_{\text{end}})) = 0$

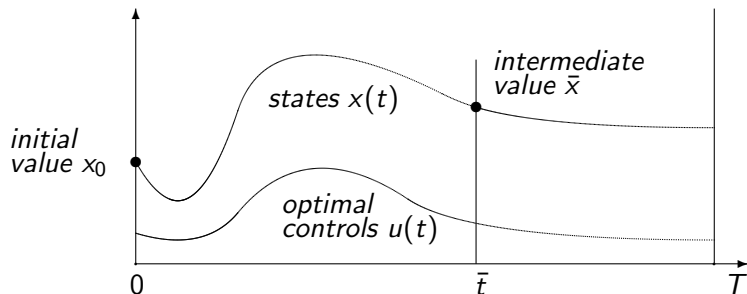
# Optimal Control Family Tree

Three basic families:

- ▶ Hamilton-Jacobi-Bellmann equation / dynamic programming
- ▶ Indirect Methods / calculus of variations / Pontryagin
- ▶ Direct Methods (control discretization)

# Principle of Optimality

Any subarc of an optimal trajectory is also optimal.



Subarc on  $[\bar{t}, T]$  is optimal solution for initial value  $\bar{x}$ .

# Dynamic Programming Cost-to-go

IDEA:

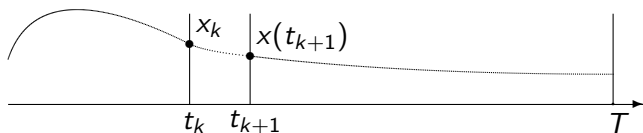
- ▶ Introduce **optimal-cost-to-go** function on  $[\bar{t}, T]$

$$J(\bar{x}, \bar{t}) := \min_{x, u} \int_{\bar{t}}^T L(x, u) dt + E(x(T)) \quad \text{s.t.} \quad x(\bar{t}) = \bar{x}, \dots$$

- ▶ Introduce grid  $0 = t_0 < \dots < t_N = T$ .
- ▶ Use **principle of optimality** on intervals  $[t_k, t_{k+1}]$ :

$$J(x_k, t_k) = \min_{x, u} \int_{t_k}^{t_{k+1}} L(x, u) dt + J(x(t_{k+1}), t_{k+1})$$

s.t.  $x(t_k) = x_k, \dots$

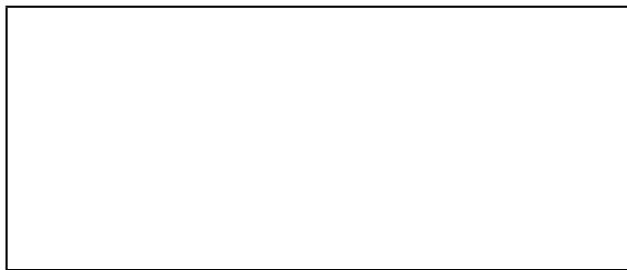


# Dynamic Programming Recursion

Starting from  $J(x, t_N) = E(x)$ , compute recursively backwards, for  $k = N - 1, \dots, 0$

$$J(x_k, t_k) := \min_{x, u} \int_{t_k}^{t_{k+1}} L(x, u) dt + J(x(t_{k+1}), t_{k+1}) \quad \text{s.t.} \quad x(t_k) = x_k, \dots$$

by solution of short horizon problems for all possible  $x_k$  and tabulation in state space.



# Dynamic Programming Recursion

Starting from  $J(x, t_N) = E(x)$ , compute recursively backwards, for  $k = N - 1, \dots, 0$

$$J(x_k, t_k) := \min_{x, u} \int_{t_k}^{t_{k+1}} L(x, u) dt + J(x(t_{k+1}), t_{k+1}) \quad \text{s.t.} \quad x(t_k) = x_k, \dots$$

by solution of short horizon problems for all possible  $x_k$  and tabulation in state space.



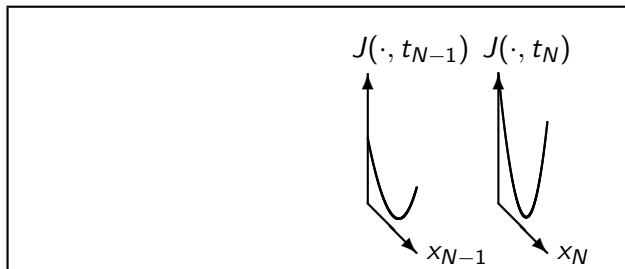


# Dynamic Programming Recursion

Starting from  $J(x, t_N) = E(x)$ , compute recursively backwards, for  $k = N - 1, \dots, 0$

$$J(x_k, t_k) := \min_{x, u} \int_{t_k}^{t_{k+1}} L(x, u) dt + J(x(t_{k+1}), t_{k+1}) \quad \text{s.t.} \quad x(t_k) = x_k, \dots$$

by solution of short horizon problems for all possible  $x_k$  and tabulation in state space.

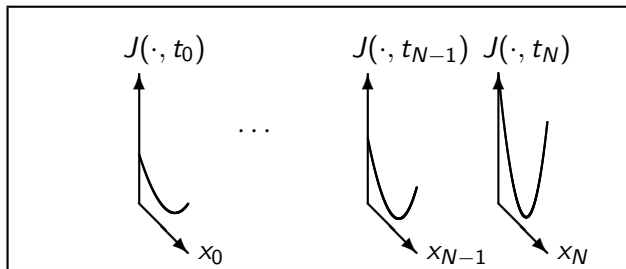


# Dynamic Programming Recursion

Starting from  $J(x, t_N) = E(x)$ , compute recursively backwards, for  $k = N - 1, \dots, 0$

$$J(x_k, t_k) := \min_{x, u} \int_{t_k}^{t_{k+1}} L(x, u) dt + J(x(t_{k+1}), t_{k+1}) \quad \text{s.t.} \quad x(t_k) = x_k, \dots$$

by solution of short horizon problems for all possible  $x_k$  and tabulation in state space.



# Hamilton-Jacobi-Bellman (HJB) Equation

- ▶ Dynamic Programming with infinitely small timesteps leads to **Hamilton-Jacobi-Bellman (HJB) Equation**:

$$-\frac{\partial J}{\partial t}(x, t) = \min_u \left( L(x, u) + \frac{\partial J}{\partial x}(x, t) f(x, u) \right) \quad \text{s.t.} \quad h(x, u) \geq 0.$$

- ▶ Solve this partial differential equation (PDE) backwards for  $t \in [0, T]$ , starting at the end of the horizon with

$$J(x, T) = E(x).$$

- ▶ **NOTE:** Optimal controls for state  $x$  at time  $t$  are obtained from

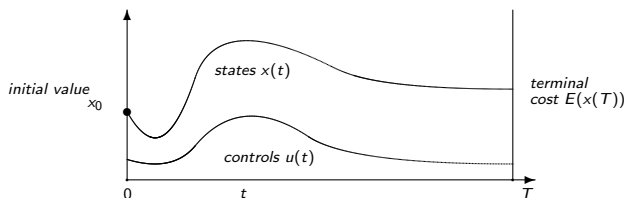
$$u^*(x, t) = \arg \min_u \left( L(x, u) + \frac{\partial J}{\partial x}(x, t) f(x, u) \right) \quad \text{s.t.} \quad h(x, u) \geq 0.$$

# Dynamic Programming / HJB

- ▶ “Dynamic Programming” applies to discrete time, “HJB” to continuous time systems.
- ▶ Pros and Cons
  - + Searches whole state space, finds global optimum.
  - + Optimal feedback controls precomputed.
  - + Analytic solution to some problems possible (linear systems with quadratic cost → Riccati Equation)
- ▶ “Viscosity solutions” (Lions et al.) exist for quite general nonlinear problems.
  - But: in general intractable, because partial differential equation (PDE) in high dimensional state space: “curse of dimensionality”.
    - ▶ Possible remedy: Approximate  $J$  e.g. in framework of neuro-dynamic programming [Bertsekas 1996].
- ▶ Used for practical optimal control of small scale systems e.g. by Bonnans, Zidani, Lee, Back, ...

# Indirect Methods

For simplicity, regard only problem without inequality constraints:



$$\text{minimize}_{x(\cdot), u(\cdot)} \int_0^T L(x(t), u(t)) dt + E(x(T))$$

subject to

$$x(0) - x_0 = 0, \quad (\text{fixed initial value})$$

$$\dot{x}(t) - f(x(t), u(t)) = 0, \quad t \in [0, T], \quad (\text{ODE model})$$

# Pontryagin's Minimum Principle

**OBSERVATION:** In HJB, optimal controls

$$u^*(t) = \arg \min_u \left( L(x, u) + \frac{\partial J}{\partial x}(x, t) f(x, u) \right)$$

depend only on derivative  $\frac{\partial J}{\partial x}(x, t)$ , not on  $J$  itself!

**IDEA:** Introduce **adjoint variables**

$$\lambda(t) \triangleq \frac{\partial J}{\partial x}(x(t), t)^T \in \mathbb{R}^{n_x}$$

and get controls from **Pontryagin's Minimum Principle**

$$u^*(t, x, \lambda) = \arg \min_u \left( \underbrace{L(x, u) + \lambda^T f(x, u)}_{\text{Hamiltonian} =: H(x, u, \lambda)} \right)$$

**QUESTION:** How to obtain  $\lambda(t)$ ?

# Adjoint Differential Equation

- ▶ Differentiate HJB Equation

$$-\frac{\partial J}{\partial t}(x, t) = \min_u H(x, u, \frac{\partial J}{\partial x}(x, t)^T)$$

with respect to  $x$  and obtain:

$$-\dot{\lambda}^T = \frac{\partial}{\partial x} (H(x(t), u^*(t, x, \lambda), \lambda(t))).$$

- ▶ Likewise, differentiate  $J(x, T) = E(x)$  and obtain terminal condition

$$\lambda(T)^T = \frac{\partial E}{\partial x}(x(T)).$$

# How to obtain explicit expression for controls?

- ▶ In simplest case,

$$u^*(t) = \arg \min_u H(x(t), u, \lambda(t))$$

is defined by

$$\frac{\partial H}{\partial u}(x(t), u^*(t), \lambda(t)) = 0$$

(Calculus of Variations, Euler-Lagrange).

- ▶ In presence of path constraints, expression for  $u^*(t)$  changes whenever active constraints change. This leads to state dependent switches.
- ▶ If minimum of Hamiltonian locally not unique, “singular arcs” occur. Treatment needs higher order derivatives of  $H$ .



# Necessary Optimality Conditions

Summarize optimality conditions as **boundary value problem**:

$$x(0) = x_0, \quad \textit{initial value}$$

$$\dot{x}(t) = f(x(t), u^*(t)), \quad t \in [0, T], \quad \textit{ODE model}$$

$$-\dot{\lambda}(t) = \frac{\partial H}{\partial x}(x(t), u^*(t), \lambda(t))^T, \quad t \in [0, T], \quad \textit{adjoint equations}$$

$$u^*(t) = \arg \min_u H(x(t), u, \lambda(t)), \quad t \in [0, T], \quad \textit{minimum principle}$$

$$\lambda(T) = \frac{\partial E}{\partial x}(x(T))^T. \quad \textit{adjoint final value.}$$

Solve with so called

- ▶ gradient methods,
- ▶ shooting methods, or
- ▶ collocation.

# Indirect Methods

- ▶ “First optimize, then discretize”
- ▶ Pros and Cons
  - + Boundary value problem with only  $2 \times n_x$  ODE.
  - + Can treat large scale systems.
    - Only necessary conditions for local optimality.
    - Need explicit expression for  $u^*(t)$ , singular arcs difficult to treat.
    - ODE strongly nonlinear and unstable.
    - Inequalities lead to ODE with state dependent switches.  
Possible remedy: Use interior point method in function space inequalities, e.g. Weiser and Deuffhard, Bonnans and Laurent-Varin
- ▶ Used for optimal control e.g. in satellite orbit planning at CNES...

# Direct Methods

- ▶ “First discretize, then optimize”
- ▶ Transcribe infinite problem into finite dimensional, **Nonlinear Programming Problem (NLP)**, and solve NLP.
- ▶ Pros and Cons:
  - + Can use state-of-the-art methods for NLP solution.
  - + Can treat inequality constraints and multipoint constraints much easier.
  - Obtains only suboptimal/approximate solution.
- ▶ Nowadays most commonly used methods due to their easy applicability and robustness.

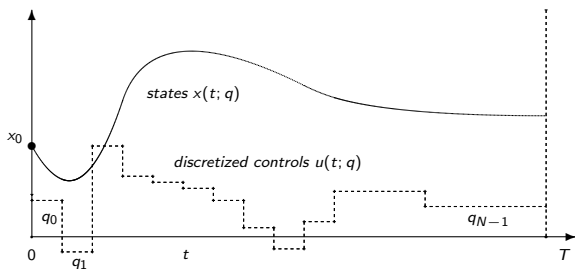
# Direct Methods Overview

We treat three direct methods:

- ▶ Direct Single Shooting (sequential simulation and optimization)
- ▶ Direct Collocation (simultaneous simulation and optimization)
- ▶ Direct Multiple Shooting (simultaneous resp. hybrid)

# Direct Single Shooting [Hicks1971,Sargent1978]

Discretize controls  $u(t)$  on fixed grid  $0 = t_0 < t_1 < \dots < t_N = T$ , regard states  $x(t)$  on  $[0, T]$  as dependent variables.



Use numerical integration to obtain state as function  $x(t; q)$  of finitely many control parameters  $q = (q_0, q_1, \dots, q_{N-1})$

# NLP in Direct Single Shooting

After control discretization and numerical ODE solution, obtain NLP:

$$\underset{q}{\text{minimize}} \quad \int_0^T L(x(t; q), u(t; q)) dt + E(x(T; q))$$

subject to

$$h(x(t_i; q), u(t_i; q)) \geq 0, \quad (\text{discretized path constraints})$$
$$i = 0, \dots, N,$$

$$r(x(T; q)) \geq 0. \quad (\text{terminal constraints})$$

Solve with finite dimensional optimization solver, e.g. Sequential Quadratic Programming (SQP).

# Solution by Standard SQP

Summarize problem as

$$\min_q F(q) \text{ s.t. } H(q) \geq 0.$$

Solve e.g. by Sequential Quadratic Programming (SQP), starting with guess  $q^0$  for controls.  $k := 0$

1. Evaluate  $F(q^k), H(q^k)$  by ODE solution, and derivatives!
2. Compute correction  $\Delta q^k$  by solution of QP:

$$\min_{\Delta q} \nabla F(q_k)^T \Delta q + \frac{1}{2} \Delta q^T A^k \Delta q \text{ s.t. } H(q^k) + \nabla H(q^k)^T \Delta q \geq 0.$$

3. Perform step  $q^{k+1} = q^k + \alpha_k \Delta q^k$  with step length  $\alpha_k$  determined by line search.

# ODE Sensitivities

How to compute the sensitivity  $\frac{\partial x(t; q)}{\partial q}$  of a numerical ODE solution  $x(t; q)$  with respect to the controls  $q$ ?

Four ways:

1. External Numerical Differentiation (END)
2. Variational Differential Equations
3. Automatic Differentiation
4. Internal Numerical Differentiation (IND)



# Numerical Test Problem

$$\begin{aligned} & \underset{x(\cdot), u(\cdot)}{\text{minimize}} && \int_0^3 x(t)^2 + u(t)^2 dt \\ & \text{subject to} && \end{aligned}$$

$$x(0) = x_0, \quad \text{(initial value)}$$

$$\dot{x} = (1 + x)x + u, \quad t \in [0, 3], \quad \text{(ODE model)}$$

$$\begin{bmatrix} 1 - x(t) \\ 1 + x(t) \\ 1 - u(t) \\ 1 + u(t) \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad t \in [0, 3], \quad \text{(bounds)}$$

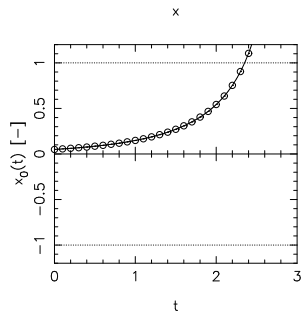
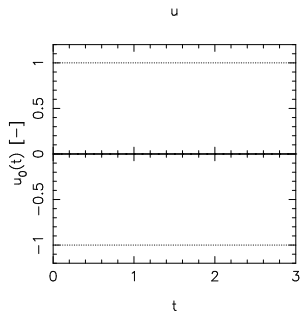
$$x(3) = 0. \quad \text{(zero terminal constraint).}$$

**Remark:** Uncontrollable growth for  
 $(1 + x_0)x_0 - 1 \geq 0 \Leftrightarrow x_0 \geq 0.618$ .

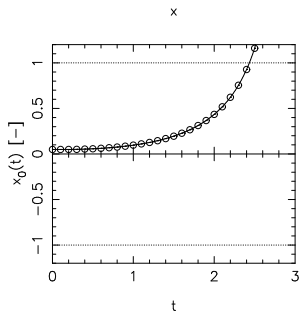
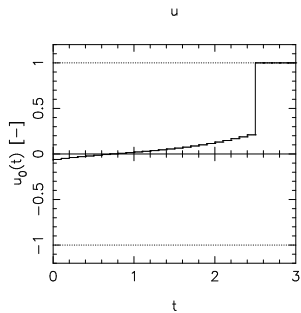
## Single Shooting Optimization for $x_0 = 0.05$

- ▶ Choose  $N = 30$  equal control intervals.
- ▶ Initialize with steady state controls  $u(t) \equiv 0$ .
- ▶ Initial value  $x_0 = 0.05$  is the maximum possible, because initial trajectory explodes otherwise.

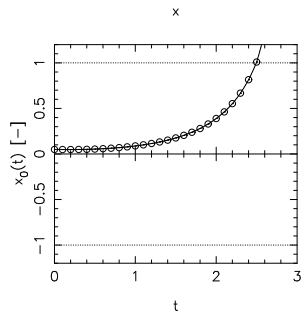
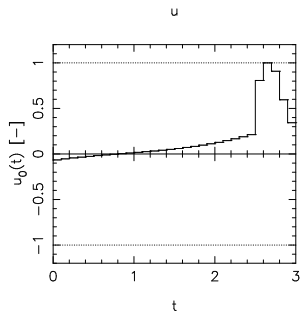
# Single Shooting: Initialization



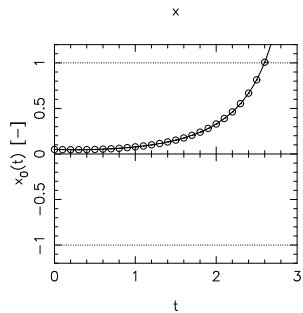
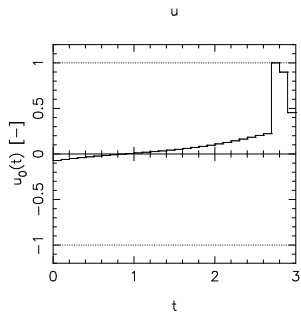
# Single Shooting: First Iteration



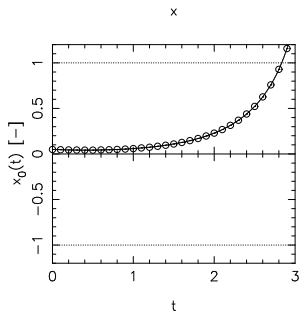
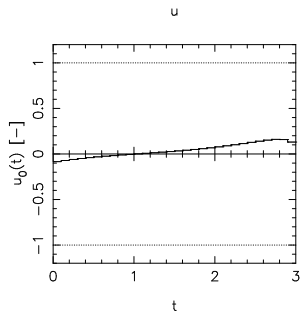
# Single Shooting: 2nd Iteration



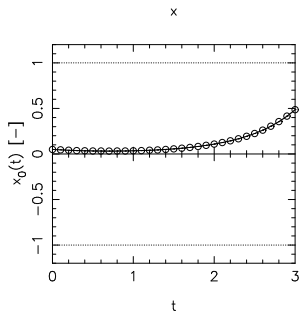
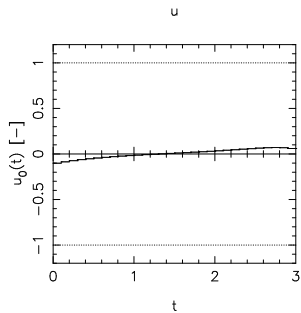
# Single Shooting: 3rd Iteration



# Single Shooting: 4th Iteration

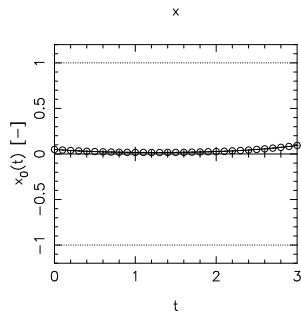
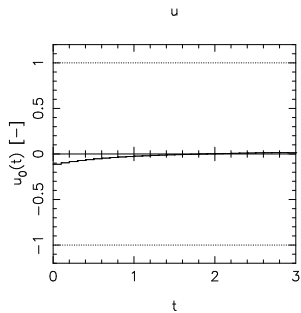


# Single Shooting: 5th Iteration

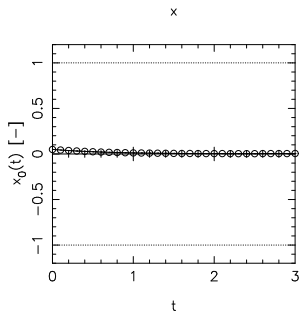
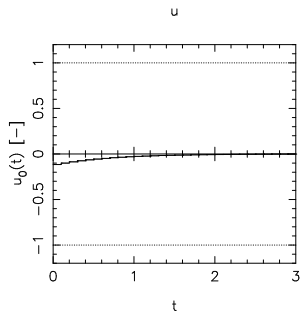




# Single Shooting: 6th Iteration



# Single Shooting: 7th Iteration and Solution



# Direct Single Shooting: Pros and Cons

- ▶ **Sequential** simulation and optimization.
- + Can use state-of-the-art ODE/DAE solvers.
- + Few degrees of freedom even for large ODE/DAE systems.
- + Active set changes easily treated.
- + Need only initial guess for controls  $q$ .
  - Cannot use knowledge of  $x$  in initialization (e.g. in tracking problems).
  - ODE solution  $x(t; q)$  can depend very nonlinearly on  $q$ .
  - Unstable systems difficult to treat.
- ▶ Often used in engineering applications e.g. in packages gOPT (PSE), DYOS (Marquardt), ...

## Direct Collocation (Sketch) [Tsang1975]

- ▶ Discretize controls and states on **fine** grid with node values  $s_i \approx x(t_i)$ .
- ▶ Replace infinite ODE

$$0 = \dot{x}(t) - f(x(t), u(t)), \quad t \in [0, T]$$

by finitely many equality constraints

$$c_i(q_i, s_i, s_{i+1}) = 0, \quad i = 0, \dots, N-1,$$

$$\text{e.g. } c_i(q_i, s_i, s_{i+1}) := \frac{s_{i+1} - s_i}{t_{i+1} - t_i} - f\left(\frac{s_i + s_{i+1}}{2}, q_i\right)$$

- ▶ Approximate also integrals, e.g.

$$\int_{t_i}^{t_{i+1}} L(x(t), u(t)) dt \approx l_i(q_i, s_i, s_{i+1}) := L\left(\frac{s_i + s_{i+1}}{2}, q_i\right) (t_{i+1} - t_i)$$

# NLP in Direct Collocation

After discretization obtain large scale, but sparse NLP:

$$\underset{s, q}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(q_i, s_i, s_{i+1}) + E(s_N)$$

subject to

$$s_0 - x_0 = 0, \quad (\text{fixed initial value})$$

$$c_i(q_i, s_i, s_{i+1}) = 0, \quad i = 0, \dots, N-1, \quad (\text{discretized ODE model})$$

$$h(s_i, q_i) \geq 0, \quad i = 0, \dots, N, \quad (\text{discretized path constraint})$$

$$r(s_N) \geq 0. \quad (\text{terminal constraints})$$

Solve e.g. with SQP method for sparse problems.

# What is a sparse NLP?

General NLP:

$$\begin{aligned} \min_w F(w) \quad \text{s.t.} \\ G(w) = 0, \\ H(w) \geq 0. \end{aligned}$$

is called sparse if the Jacobians (derivative matrices)

$$\nabla_w G^T = \frac{\partial G}{\partial w} = \left( \frac{\partial G}{\partial w_j} \right)_{ij} \quad \text{and} \quad \nabla_w H^T$$

contain many zero elements.

In SQP methods, this makes QP much cheaper to build and to solve.

# Direct Collocation: Pros and Cons

- ▶ **Simultaneous** simulation and optimization.
- + Large scale, but very sparse NLP.
- + Can use knowledge of  $x$  in initialization.
- + Can treat unstable systems well.
- + Robust handling of path and terminal constraints.
  - Adaptivity needs new grid, changes NLP dimensions.
- ▶ Successfully used for practical optimal control e.g. by Biegler and Wächter (IPOPT), Betts,

# Direct Multiple Shooting [Bock 1984]

- ▶ Discretize controls piecewise on a coarse grid

$$u(t) = q_i \quad \text{for } t \in [t_i, t_{i+1}]$$

- ▶ Solve ODE on each interval  $[t_i, t_{i+1}]$  numerically, starting with artificial initial value  $s_i$ :

$$\begin{aligned}\dot{x}_i(t; s_i, q_i) &= f(x_i(t; s_i, q_i), q_i), \quad t \in [t_i, t_{i+1}], \\ x_i(t_i; s_i, q_i) &= s_i.\end{aligned}$$

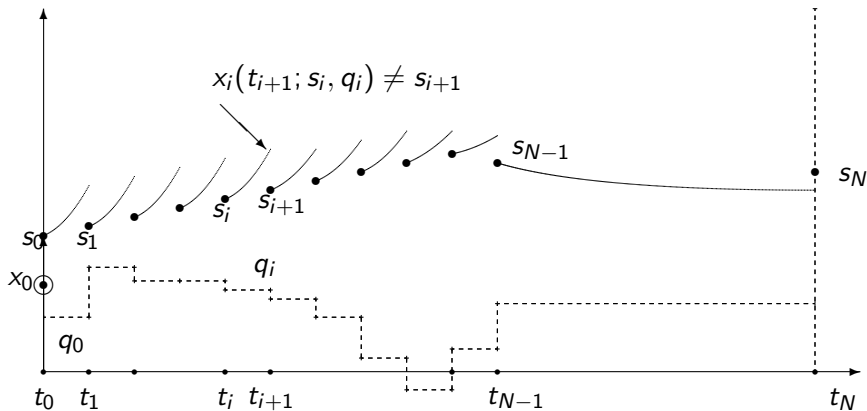
Obtain trajectory pieces  $x_i(t; s_i, q_i)$ .

- ▶ Also numerically compute integrals

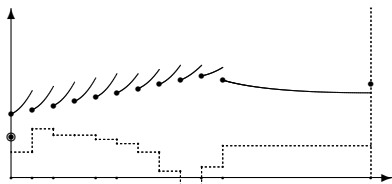
$$l_i(s_i, q_i) := \int_{t_i}^{t_{i+1}} L(x_i(t; s_i, q_i), q_i) dt$$



# Sketch of Direct Multiple Shooting



# NLP in Direct Multiple Shooting



$$\underset{s, q}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, q_i) + E(s_N)$$

subject to

$$s_0 - x_0 = 0, \quad (\text{initial value})$$

$$s_{i+1} - x_i(t_{i+1}; s_i, q_i) = 0, \quad i = 0, \dots, N-1, \quad (\text{continuity})$$

$$h(s_i, q_i) \geq 0, \quad i = 0, \dots, N, \quad (\text{discretized path constraints})$$

$$r(s_N) \geq 0. \quad (\text{terminal constraints})$$

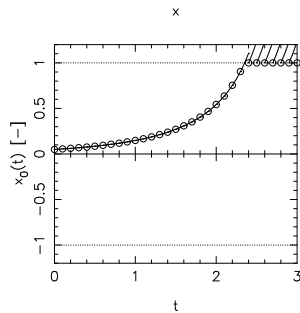
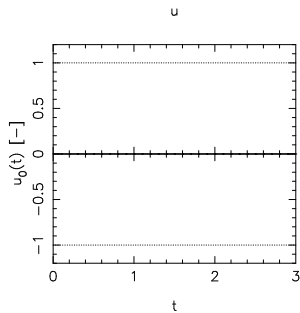
# Structured NLP

- ▶ Summarize all variables as  $w := (s_0, q_0, s_1, q_1, \dots, s_N)$ .
- ▶ Obtain structured NLP

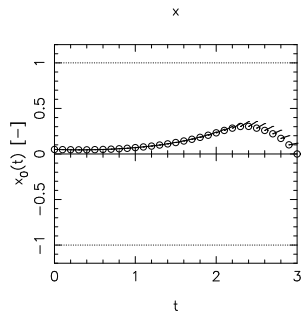
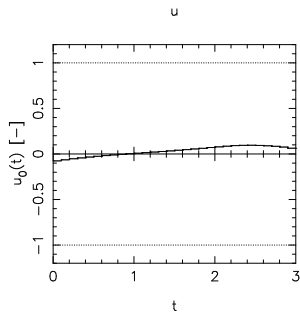
$$\min_w F(w) \quad \text{s.t.} \quad \begin{cases} G(w) = 0 \\ H(w) \geq 0. \end{cases}$$

- ▶ Jacobian  $\nabla G(w^k)^T$  contains dynamic model equations.
- ▶ Jacobians and Hessian of NLP are block sparse, can be exploited in numerical solution procedure.

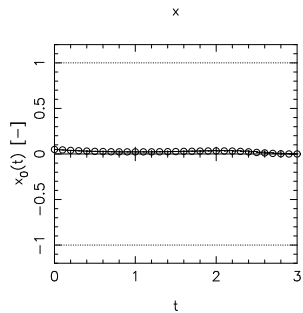
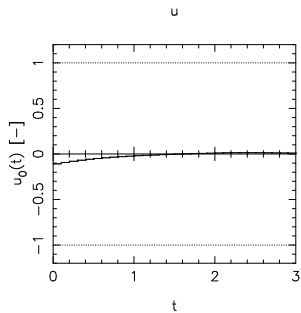
# Test Example: Initialization with $u(t) \equiv 0$



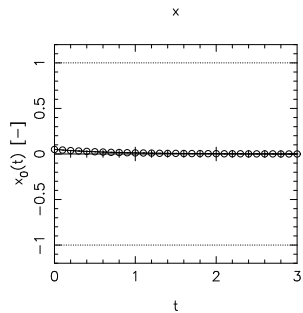
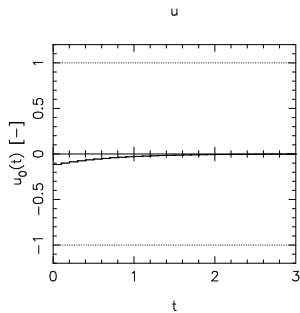
# Multiple Shooting: First Iteration



# Multiple Shooting: 2nd Iteration



# Multiple Shooting: 3rd Iteration and Solution

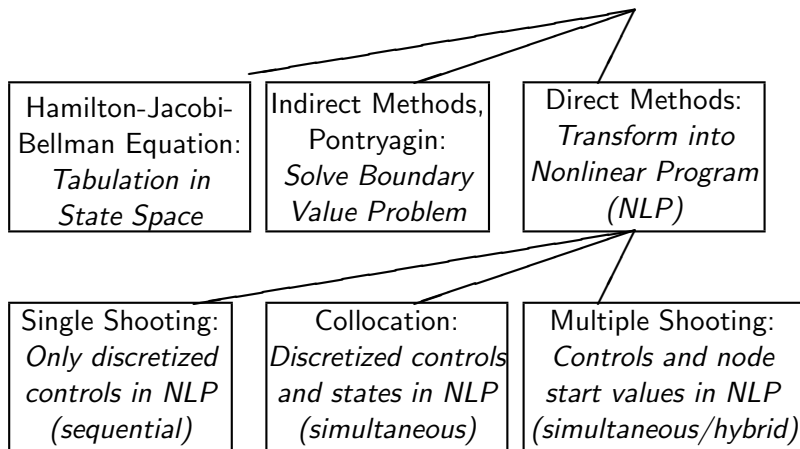


# Direct Multiple Shooting: Pros and Cons

- ▶ **Simultaneous** simulation and optimization.
- + uses **adaptive** ODE/DAE solvers
- + but NLP has **fixed dimensions**
- + can use knowledge of  $x$  in initialization (here bounds; more important in online context).
- + can treat unstable systems well.
- + robust handling of path and terminal constraints.
- + easy to parallelize.
  - not as sparse as collocation.
- ▶ Used for practical optimal control e.g by Franke (ABB) (“HQP”), Terwen (Daimler); Bock et al. (“MUSCOD-II”); in ACADO Toolkit; ...



# Conclusions: Optimal Control Family Tree



## Literature

- ▶ T. Binder, L. Blank, H. G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt and J. P. Schler, and O. v. Stryk: Introduction to Model Based Optimization of Chemical Processes on Moving Horizons. In Grötschel, Krumke, Rambau (eds.): Online Optimization of Large Scale Systems: State of the Art, Springer, 2001. pp. 295–340.
- ▶ John T. Betts: Practical Methods for Optimal Control Using Nonlinear Programming. SIAM, Philadelphia, 2001. ISBN 0-89871-488-5
- ▶ Dimitri P. Bertsekas: Dynamic Programming and Optimal Control. Athena Scientific, Belmont, 2000 (Vol I, ISBN: 1-886529-09-4) & 2001 (Vol II, ISBN: 1-886529-27-2)
- ▶ A. E. Bryson and Y. C. Ho: Applied Optimal Control, Hemisphere/Wiley, 1975.