Numerical Optimal Control

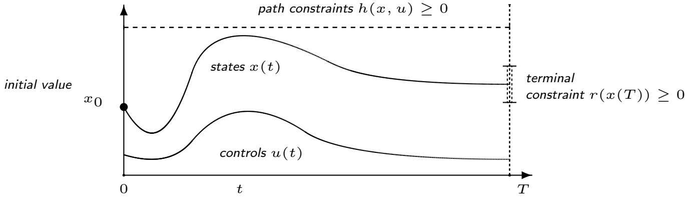
Moritz Diehl

Systems Control and Optimization Laboratory, University of Freiburg, Germany

Robust Nonlinear MPC Course, University of California Santa Barbara
March 25-28, 2024

universität freiburg

Simplified Optimal Control Problem in ODE



Continuous Time Optimal Control Problem

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

subject to

$$\begin{aligned}
x(0) - x_0 &= 0, && && \text{(fixed initial value)}\\
\dot{x}(t) - f(x(t),u(t)) &= 0, && t \in [0,T], && \text{(ODE model)}\\
h(x(t),u(t)) &\geq 0, && t \in [0,T], && \text{(path constraints)}\\
r\left(x(T)\right) &\geq 0 && && \text{(terminal constraints)}
\end{aligned}$$

(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 $r(x(t_0), x(t_1), \ldots, x(t_{\text{end}})) = 0$

Discrete Time Problem

Discrete Time Optimal Control Problem

$$\begin{aligned}
\min_{x,u} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad x_0 &= \bar{x}_0 \\
x_{k+1} &= f(x_k, u_k) \\
h(x_k, u_k) &\geq 0, \qquad\qquad k = 0, \ldots, N-1 \\
r(x_N) &\geq 0
\end{aligned}$$

Optimal Control Family Tree

Three basic families:

- Dynamic Programming / Hamilton-Jacobi-Bellmann Equation
- Indirect Methods / Calculus of Variations / Pontryagin's Maximum Principle
- Direct Methods, i.e., discretization combined with nonlinear programming

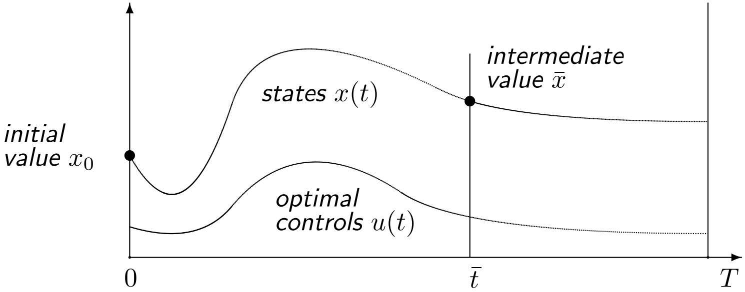
Optimal Control Family Tree

Three basic families:

- **Dynamic Programming** (/ Hamilton-Jacobi-Bellmann Equation)
- (Indirect Methods / Calculus of Variations / Pontryagin's Maximum Principle)
- **Direct Methods, i.e., discretization combined with nonlinear programming**

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 (discrete time, unconstrained)

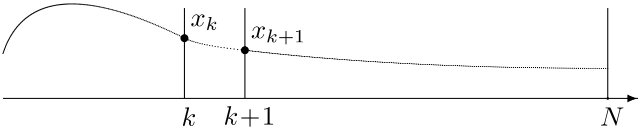
IDEA:

- Introduce **optimal-cost-to-go** function on $[k, N]$

$$J_k(x) := \min_{s_k, u_k, \ldots, s_N} \sum_{i=k}^{N-1} L(s_i, u_i) + E(s_N) \quad \text{s.t.} \quad s_k = x, \ldots$$

- Use **principle of optimality** on intervals $[k, k+1]$:

$$J_k(x_k) = \min_{s_k, u_k, s_{k+1}} L(s_k, a_k) + J_{k+1}(s_{k+1})$$
$$\text{s.t.} \quad s_k = x_k,\ s_{k+1} = f(s_k, u_k)$$



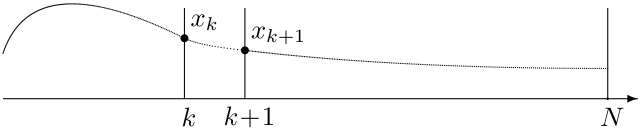
Dynamic Programming Step

Can simplify

$$J_k(x_k) = \min_{s_k, u_k, s_{k+1}} L(s_k, u_k) + J_{k+1}(s_{k+1})$$
$$\text{s.t.} \quad s_k = x_k, \; s_{k+1} = f(s_k, u_k)$$

by trivial elimination of s_k, s_{k+1} to

$$J_k(x_k) \;=\; \min_{u_k} \; L(x_k, u_k) + J_{k+1}(f(x_k, u_k))$$



Dynamic Programming Recursion

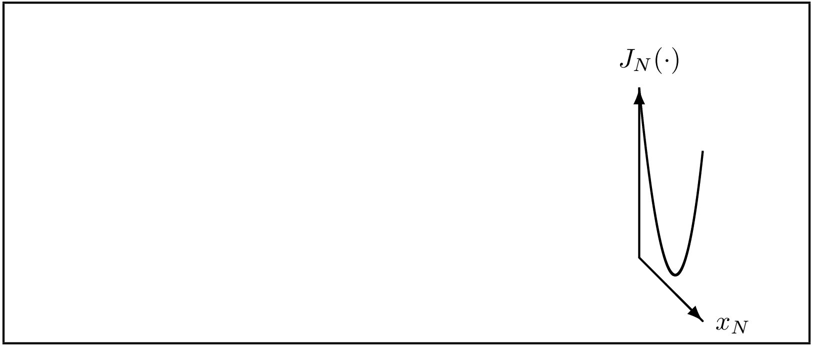
Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{R}^{n_x}$
for $k = N-1, N-2, \ldots$

$$J_k(x) = \min_u L(x,u) + J_{k+1}(f(x,u))$$

Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{R}^{n_x}$
for $k = N-1, N-2, \ldots$

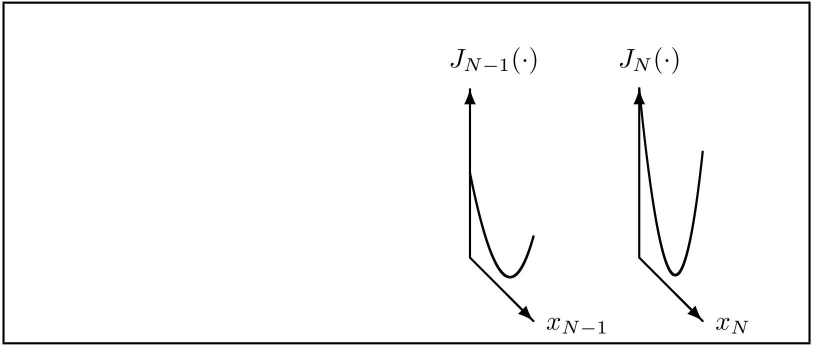
$$J_k(x) = \min_u L(x,u) + J_{k+1}(f(x,u))$$



Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{R}^{n_x}$
for $k = N-1, N-2, \ldots$

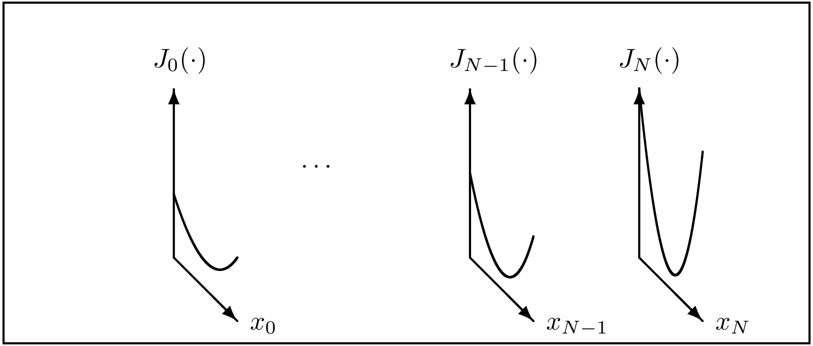
$$J_k(x) = \min_u L(x,u) + J_{k+1}(f(x,u))$$



Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{R}^{n_x}$
for $k = N-1, N-2, \dots$

$$J_k(x) = \min_u L(x,u) + J_{k+1}(f(x,u))$$



The optimal feedback control policy

The **optimal feedback control law** π_k^* at time k is defined by

$$\pi_k^*(x) \quad := \quad \arg\min_u \; L(x,u) + J_{k+1}(f(x,u))$$

These feedback laws together define the **optimal feedback control policy** $(\pi_0^*, \ldots, \pi_{N-1}^*)$ which tells us for any state x at any time index k what would be the optimal control action.

How to obtain optimal trajectories ?

The optimal policy $(\pi_0^*, \dots, \pi_{N-1}^*)$ allows us to solve the original optimal control problem.

Starting with $x_0^* := \bar{x}_0$, we simulate the closed loop system for $k = 0, 1, \dots, N-1$:

$$\begin{aligned} u_k^* &:= \pi_k^*(x_k^*) \\ x_{k+1}^* &:= f(x_k^*, u_k^*) \end{aligned}$$

yielding the optimal trajectories $x^* = (x_0^*, \dots, x_N^*)$ and $u^* = (u_0^*, \dots, u_N^*)$ that solve problem (2).

Optimal Control Problem

$$\begin{aligned} \min_{x,u} \; & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) \\ \text{s.t.} \; & x_0 = \bar{x}_0 \\ & x_{k+1} = f(x_k, u_k), \\ & \qquad k = 0, \dots, N-1 \end{aligned} \tag{2}$$

How to obtain optimal trajectories ?

The optimal policy $(\pi_0^*, \ldots, \pi_{N-1}^*)$ allows us to solve the original optimal control problem.

Starting with $x_0^* := \bar{x}_0$, we simulate the closed loop system for $k = 0, 1, \ldots, N-1$:

$$\begin{aligned} u_k^* &:= \pi_k^*(x_k^*) \\ x_{k+1}^* &:= f(x_k^*, u_k^*) \end{aligned}$$

yielding the optimal trajectories $x^* = (x_0^*, \ldots, x_N^*)$ and $u^* = (u_0^*, \ldots, u_N^*)$ that solve problem (2).

Optimal Control Problem

$$\begin{aligned} \min_{x,u} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) \\ \text{s.t.} \quad & x_0 = \bar{x}_0 \\ & x_{k+1} = f(x_k, u_k), \\ & \qquad k = 0, \ldots, N-1 \end{aligned} \tag{2}$$

Note: MPC applies only $\pi_0^*(\bar{s}_0)$. The MPC law can be generated in one of three ways:

(a) via dynamic programming,
(b) via online solution of (2) in classical MPC, or
(c) via offline solution of (2) based on parametric programming in *explicit MPC*.

Robust Dynamic Programming

Dynamic Programming can straightforwardly be extended to games like chess, or to closed loop robust min-max optimal control problems, which are not easily treatable with other robust optimization methods.

Here, in each time step, we first choose the controls u_k, but then an adverse player choses disturbances w_k, and both influence the system dynamics $x_{k+1} = f(x_k, u_k, w_k)$.

Robust DP Recursion

Iterate backwards, from $k = N - 1$ down to $k = 0$, using the robust Bellman equation

$$J_k(x) = \min_{u} \max_{w \in \mathbb{W}} \left(\, L(x, u) + J_{k+1}(f(x, u, w)) \, \right)$$

starting with terminal cost

$$J_N(x) = E(x)$$

The only additional effort are the evaluations of the worst-cases in each DP step.

(Hamilton-Jacobi-Bellman (HJB) Equation)

- DP with infinitesimal steps leads to **Hamilton-Jacobi-Bellman (HJB) Equation**:

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

- This is a partial differential equation (PDE) for $t \in [0,T]$ with terminal condition

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

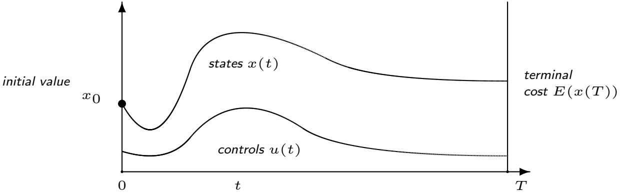
- **NOTE:** Optimal controls for state x at time t are obtained from

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

Dynamic Programming: Pros and Cons

- "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 sometimes possible (linear systems with quadratic cost)
  - + can easily be extended to robust min-max or stochastic optimal control
  - - but: in general intractable, because need to tabulate value function in state space: Bellman's "curse of dimensionality"
- possible remedy: Approximate J e.g. in framework of Neuro-Dynamic Programming [Bertsekas 1996], closely related to Reinforcement Learning [Barton and Sutto, 2018]

For simplicity, regard only problem without inequality constraints:



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

subject to

$$\begin{aligned} x(0) - x_0 &= 0, & & \text{(fixed initial value)} \\ \dot{x}(t) - f(x(t),u(t)) &= 0, \quad t \in [0,T], & & \text{(ODE model)} \end{aligned}$$

(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**

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

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

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

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:

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

- 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**:

$$
\begin{aligned}
x(0) &= x_0, && \textit{initial value} \\
\dot{x}(t) &= f(x(t), u^*(t)), \quad t \in [0, T], && \textit{ODE model} \\
-\dot{\lambda}(t) &= \frac{\partial H}{\partial x}(x(t), u^*(t), \lambda(t))^T, \quad t \in [0, T], && \textit{adjoint equations} \\
u^*(t) &= \arg\min_u H(x(t), u, \lambda(t)), \quad t \in [0, T], && \textit{minimum principle} \\
\lambda(T) &= \frac{\partial E}{\partial x}(x(T))^T. && \textit{adjoint final value.}
\end{aligned}
$$

Solve with so called

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

(Indirect Methods / Pontryagin: Pros and Cons)

- "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: interior point method in function space e.g. Weiser and Deuflhard, Bonnans and Laurent-Varin
- used for optimal control e.g. in satellite orbit planning (at French space agency)

Direct Methods

- "first discretize, then optimize"
- transcribe infinite problem into finite **Nonlinear Programming Problem (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

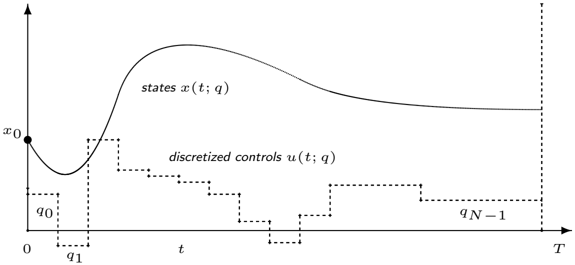
Direct Methods Overview

We treat three direct methods:

- Direct Single Shooting (sequential simulation and optimization)
- Direct Collocation (fully simultaneous simulation and optimization)
- Direct Multiple Shooting (simultaneous simulation and optimization)

Direct Single Shooting [Hicks1971,Sargent1978]

Discretize controls $u(t)$ on fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$, 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 $q = (q_0, q_1, \ldots, q_{N-1}) \in \mathbb{R}^{N \cdot n_u}$

NLP in Direct Single Shooting

After control discretization and numerical ODE solution, obtain NLP:

NLP resulting from Direct Single Shooting

$$\begin{aligned}
&\underset{q \in \mathbb{R}^{N \cdot n_u}}{\text{minimize}} \quad \int_0^T L(x(t;q), u(t;q))\, dt + E\left(x(T;q)\right) \\
&\text{subject to} \\
&\quad h(x(t_i;q), u(t_i;q)) \geq 0, \\
&\qquad i = 0, \ldots, N, && \textit{(discretized path constraints)} \\
&r\left(x(T;q)\right) \geq 0. && \textit{(terminal constraints)}
\end{aligned}$$

Solve with nonlinear programming solver, e.g. Sequential Quadratic Programming (SQP)

Solution by Standard SQP

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

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 Δq^k by solution of QP:

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

3. Perform step $q^{k+1} = q^k + \alpha_k \Delta q^k$ with step length $\alpha_k \in (0, 1]$ determined by line search

ODE Sensitivities

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

many ways, for example:

- External Numerical Differentiation (END)
- Variational Differential Equations
- Automatic Differentiation (AD) of integration code
- Internal Numerical Differentiation (IND)

cf. [Rien Quirynen, Numerical simulation methods for embedded optimization, PhD thesis, KU Leuven and Freiburg University, 2017]

Numerical Test Problem

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad \int_0^3 x(t)^2 + u(t)^2 \, dt$$

subject to

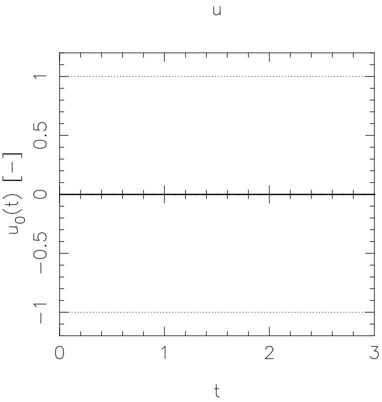
$$\begin{aligned}
x(0) &= x_0, && && \text{(initial value)}\\
\dot{x} &= (1+x)x + u, && t \in [0,3], && \text{(ODE model)}\\
\begin{bmatrix} 1 - x(t) \\ 1 + x(t) \\ 1 - u(t) \\ 1 + u(t) \end{bmatrix} &\geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, && t \in [0,3], && \text{(bounds)}\\
x(3) &= 0. && && \text{(zero terminal constraint)}
\end{aligned}$$

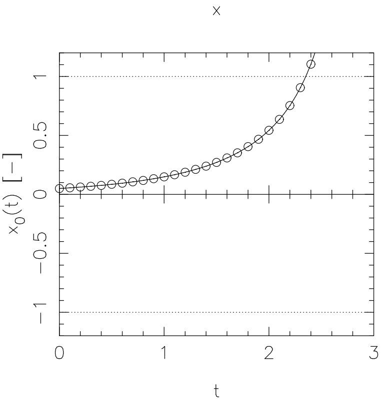
Remark: Uncontrollable growth for $(1+x_0)x_0 - 1 \geq 0 \Leftrightarrow x_0 \geq 0.618.$

Single Shooting Optimization for $x_0 = 0.05$

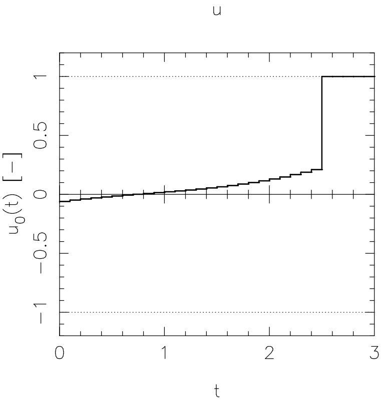
- choose $N = 30$ equal control intervals
- initialize with steady state controls $u(t) \equiv 0$
- initial value $x_0 = 0.05$ is the maximum possible for the problem to be solved by single shooting, because the initial trajectory explodes otherwise

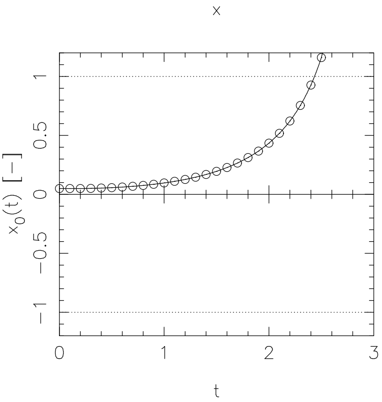
Single Shooting: Initialization



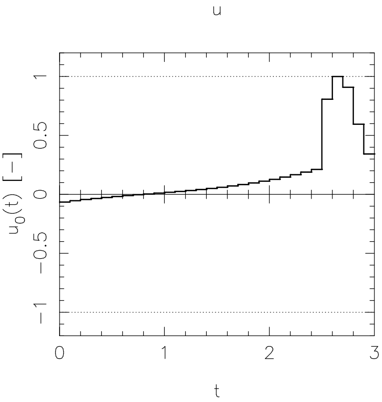


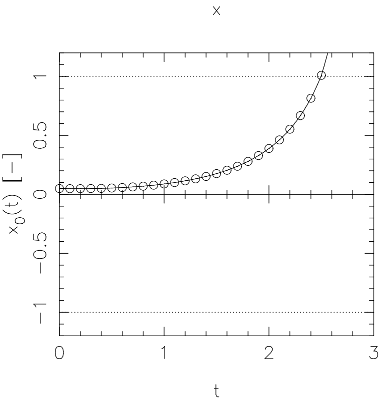
Single Shooting: First Iteration



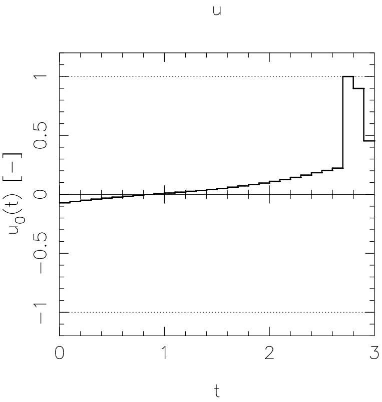


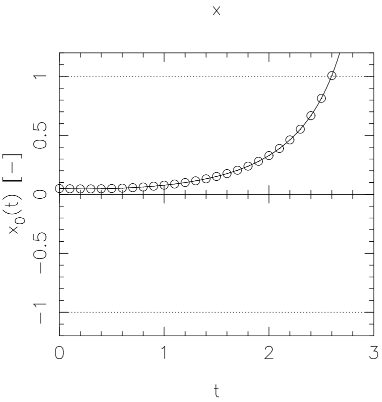
Single Shooting: 2nd Iteration



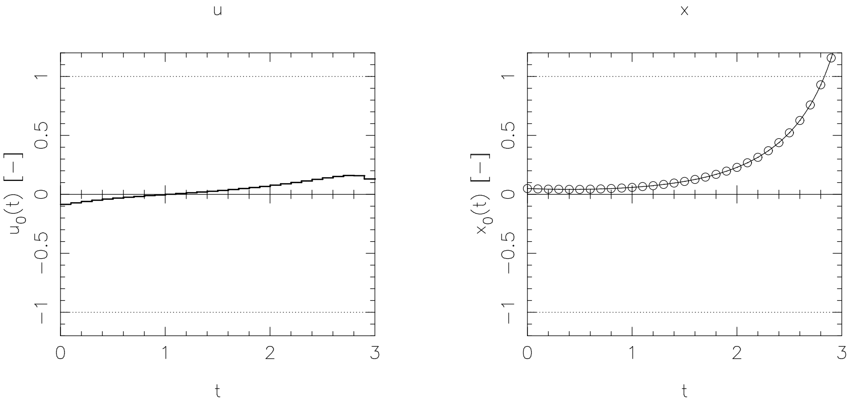


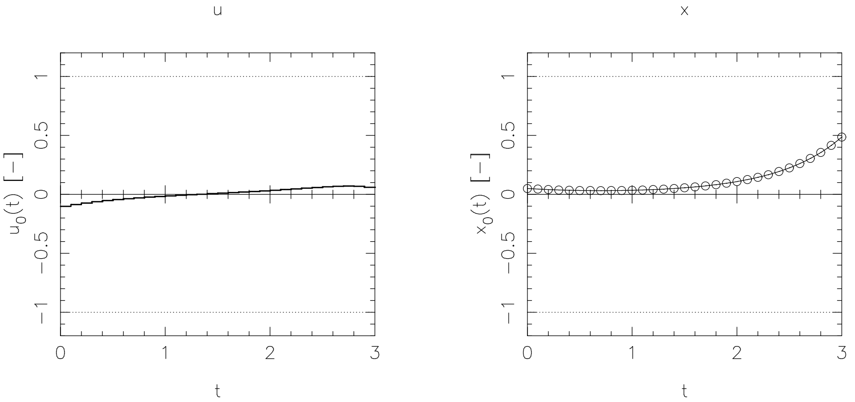
Single Shooting: 3rd Iteration



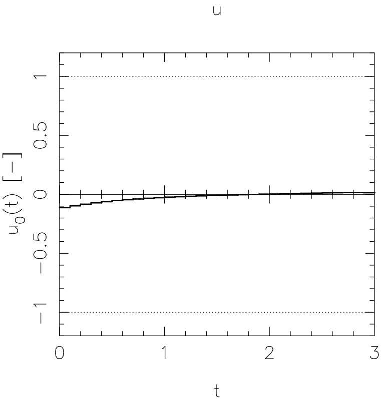


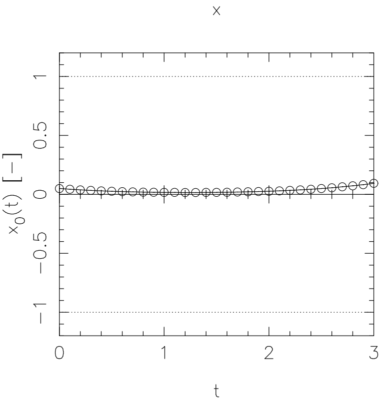
Single Shooting: 4th Iteration



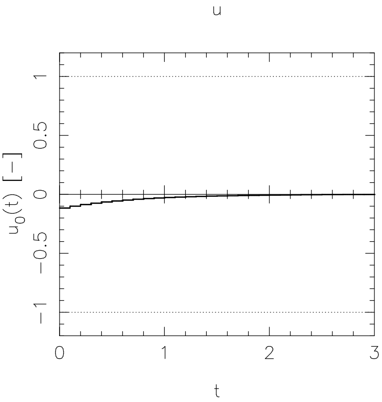

u
x
$u_0(t)$ [−]
$x_0(t)$ [−]
1
0.5
0
−0.5
−1
0
1
2
3
t
t

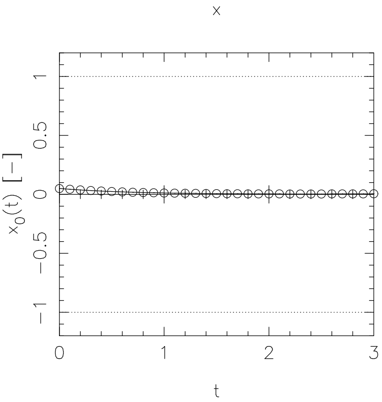
Single Shooting: 6th Iteration





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 self-made optimal control codes in engineering applications

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 local intermediate integration variables z_i and finitely many equality constraints

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

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

- Approximate also integrals, e.g.

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

NLP in Direct Collocation

After discretization, obtain large scale, but sparse NLP:

Sparse NLP resulting from direct collocation

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

subject to

$$\begin{aligned}
s_0 - x_0 &= 0, & & \text{(fixed initial value)}\\
c_i(s_i, q_i, z_i, s_{i+1}) &= 0, & i = 0, \ldots, N-1, & \quad \text{(discretized ODE model)}\\
h(s_i, q_i, z_i) &\geq 0, & i = 0, \ldots, N-1, & \quad \text{(discretized path constraints)}\\
r\left(s_N\right) &\geq 0. & & \text{(terminal constraints)}
\end{aligned}$$

solve NLP with sparsity exploiting SQP or nonlinear interior point method (e.g. ipopt)

What is a sparse NLP?

General NLP:

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

is called sparse if the Jacobians (derivative matrices)

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

contain many zero elements.

In SQP and IP methods, this makes the linear systems 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 by many experienced researchers

Direct Multiple Shooting [Bock 1984]

- Discretize controls piecewise on a coarse grid

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

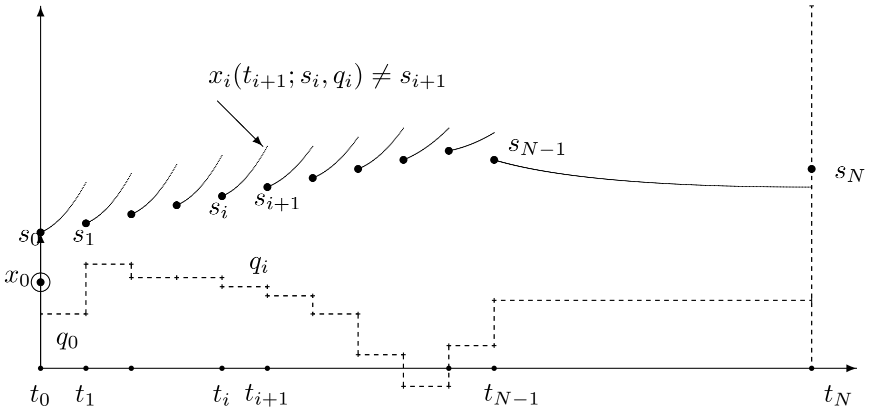
- Solve ODE on each interval $[t_i, t_{i+1}]$ numerically, starting with artificial initial value s_i:

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

  Obtain trajectory pieces $x_i(t; s_i, q_i)$.
- Also numerically compute integrals

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

Sketch of Direct Multiple Shooting



NLP in Direct Multiple Shooting

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

subject to

$$
\begin{aligned}
s_0 - x_0 &= 0, && \text{(initial value)}\\
s_{i+1} - x_i(t_{i+1}; s_i, q_i) &= 0,\ i = 0, \ldots, N-1, && \text{(continuity)}\\
h(s_i, q_i) &\geq 0,\ i = 0, \ldots, N, && \text{(discretized path constraints)}\\
r\left(s_N\right) &\geq 0. && \text{(terminal constraints)}
\end{aligned}
$$

Multiple Shooting NLP = Discrete Time Optimal Control Problem

Discrete Time Optimal Control Problem

$$\begin{aligned}
\min_{x,u} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{k+1} = f(x_k, u_k) \\
& h(x_k, u_k) \geq 0, \qquad k = 0, \dots, N-1 \\
& r(x_N) \geq 0
\end{aligned}$$

Nonlinear Program

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

summarize all variables as $w := (s_0, q_0, s_1, q_1, \dots, s_N)$

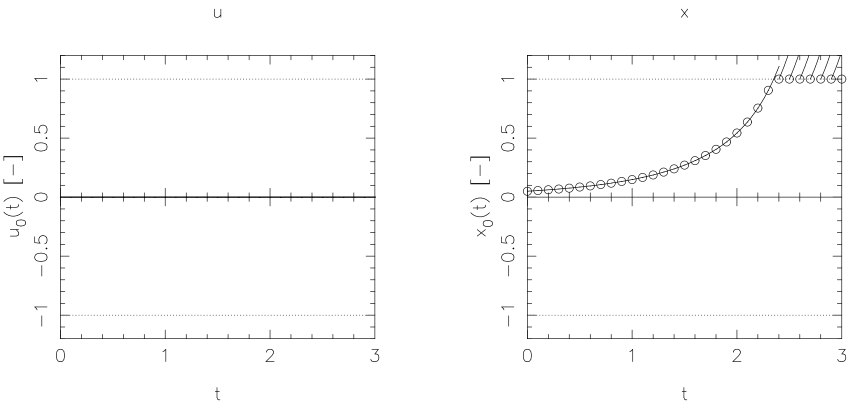
Structured Block-Sparse NLP

Nonlinear Program

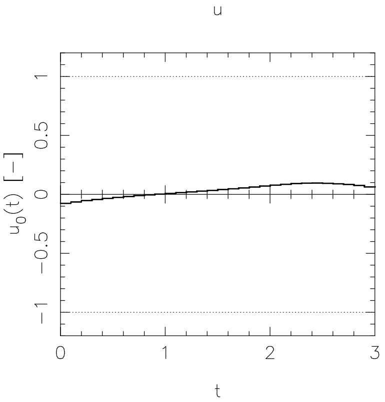
$$\min_{w} \ F(w)$$
$$\text{s.t.} \quad G(w) = 0$$
$$H(w) \geq 0$$

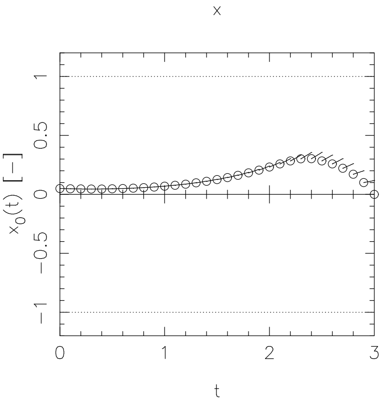
- Jacobian $\nabla G(w)^\top$ contains linearized dynamic model equations
- Jacobians and Hessian of NLP are block sparse, can be exploited in numerical solution
- NLPs of single and direct multiple shooting are equivalent (same solutions in control space)
- but "lifting" of the state variables of multiple shooting reduces the nonlinearity, as observed by many practitioners and investigated theoretically by [Albersmeyer and Diehl, The Lifted Newton Method and Its Application to Optimization, SIAM J. Opt., 2010]

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

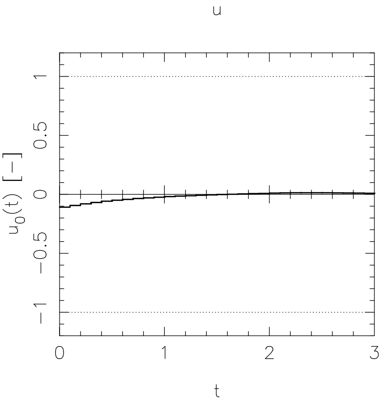


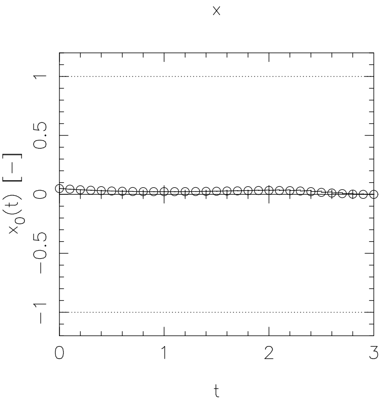
Multiple Shooting: First Iteration



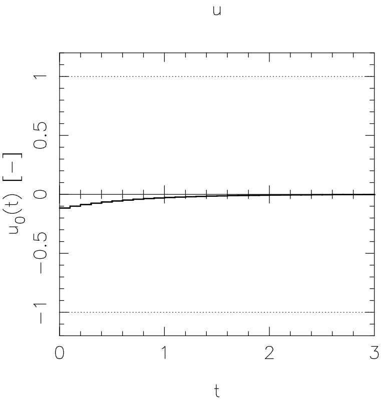


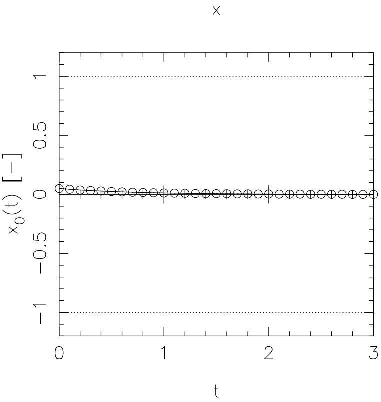
Multiple Shooting: 2nd Iteration





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 (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 in many codes e.g MUSCOD (Bock), HQP (Franke), MUSCOD-II (Leineweber et al.), ACADO Toolkit (Houska, Ferreau et al.), acados (Verschueren, Frey, Frison, Kouzoupis, Quirynen et al.), ...

Conclusions: Optimal Control Family Tree

- Dynamic Programming / Hamilton-Jacobi-Bellman Equation: *Tabulation in State Space*
- Indirect Methods, Pontryagin: *Solve Boundary Value Problem*
- Direct Methods: *Transform into Nonlinear Program (NLP)*
  - Direct Single Shooting: *Only discretized controls in NLP (sequential)*
  - Direct Collocation: *Discretized controls and states in NLP (fully simultaneous)*
  - Direct Multiple Shooting: *Controls and node start values in NLP (simultaneous)*

Literature

- Moritz Diehl, Sébastien Gros: Numerical Optimal Control (Draft), 2024
- John T. Betts: Practical Methods for Optimal Control Using Nonlinear Programming. SIAM, Philadelphia, 2001. ISBN 0-89871-488-5
- A. E. Bryson and Y. C. Ho: Applied Optimal Control, Hemisphere/Wiley, 1975.
- Dimitri P. Bertsekas: Dynamic Programming and Optimal Control. Athena Scientific (2001)
- Dimitri P. Bertsekas: Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control. Athena Scientific (2022).
- J. Björnberg and M. Diehl: Approximate robust dynamic programming and robustly stable MPC. Automatica (2006)
- T. Binder, L. Blank, H. G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt and J. P. Schlöder, 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.