# 2. Direct methods for smooth nonlinear optimal control

Armin Nurkanović

Systems Control and Optimization Laboratory, University of Freiburg, Germany
(slides create jointly with **Moritz Diehl**)

**Winter School on Numerical Methods for Optimal Control of Nonsmooth Systems**
École des Mines de Paris
February 3-5, 2025, Paris, France

universität freiburg

# Outline of the lecture

# Continuous-time optimal control problems (OCP)

**Continuous-time optimal control problem**

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

- decision variables $x(\cdot)$, $u(\cdot)$ in infinite dimensional function space
- infinitely many constraints for $t \in [0,T]$
- smooth ordinary differential equations (ODE)

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

- dynamic model can be more general e.g., nonsmooth
- OCP can be convex or nonconvex

# Classification of optimal control methods

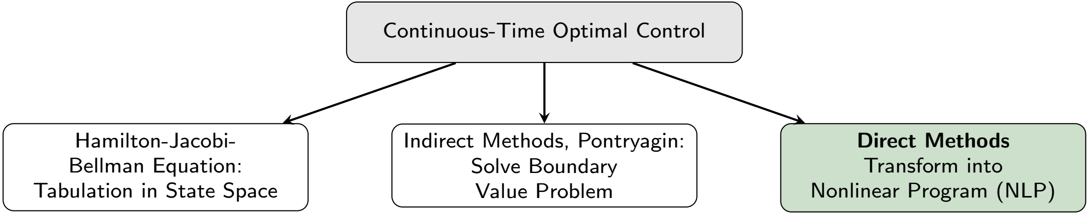



Figure inspired by Figure 9.2. in Moritz Diehl, Sébastien Gros. "Numerical optimal control (Draft)," Lecture notes, 2024

# Classification of optimal control methods

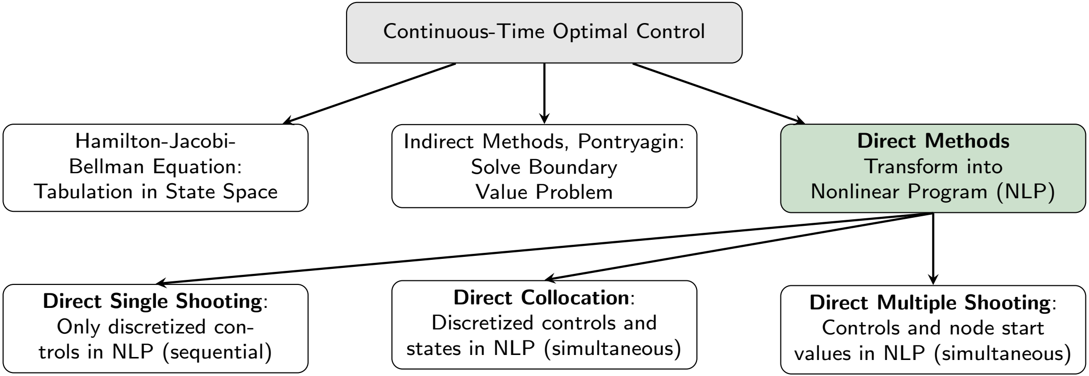
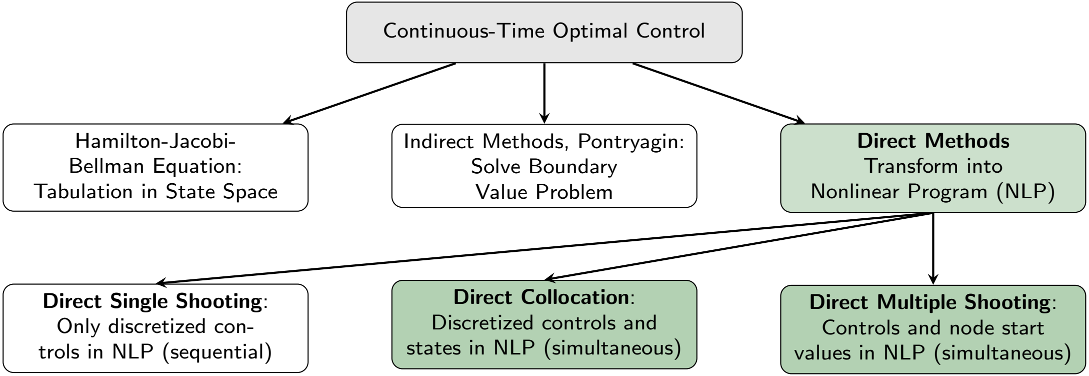



Figure inspired by Figure 9.2. in Moritz Diehl, Sébastien Gros. "Numerical optimal control (Draft)," Lecture notes, 2024

# Classification of optimal control methods

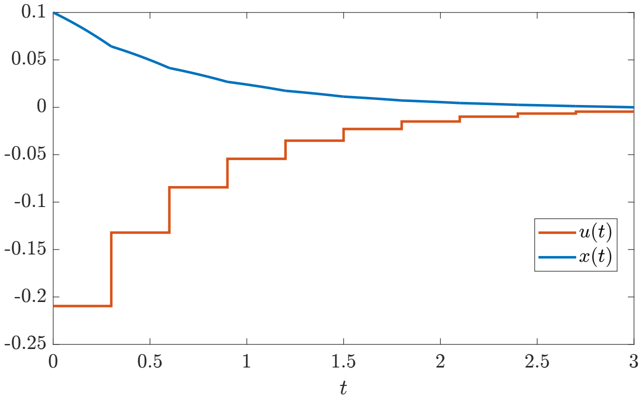



- indirect methods: first optimize, then discretize
- direct methods: first discretize, then optimize

Figure inspired by Figure 9.2. in Moritz Diehl, Sébastien Gros. "Numerical optimal control (Draft)," Lecture notes, 2024

# Direct single shooting

- discretize controls $u(t) \in \mathbb{R}^{n_u}$ on a fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$, and regard $x(t)$ as depended variables
- parametrize controls by $q = (q_0, \ldots, q_{N-1}) \in \mathbb{R}^{N \cdot N_u}$
- use numerical integration and obtain state $x(t; q)$ as function of $q$



# Direct single shooting

## NLP resulting from direct single shooting

$$
\begin{aligned}
\min_{q \in \mathbb{R}^{N \cdot n_u}} \quad & \int_0^T L(x(t;q), u(t;q))\, \mathrm{d}t + E(x(T;q)) \\
\text{s.t.} \quad & 0 \geq h(x(t_i;q), u(t_i;q)),\ i = 1, \dots, N \\
& 0 \geq r(x(T;q))
\end{aligned}
$$

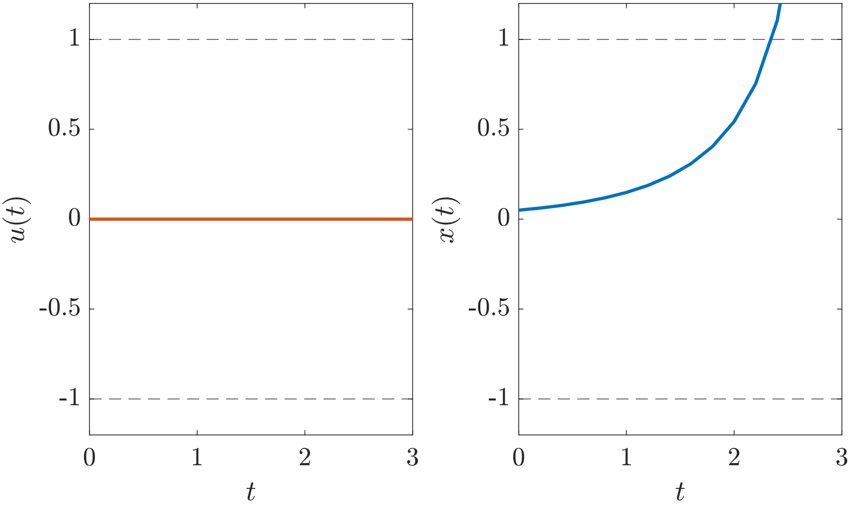
- This is a standard NLP, can be solved with SQP or interior-point method
- Convergence can be difficult for unstable systems

# Numerical example with single shooting

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^3 x(t)^2 + u(t)^3 \,\mathrm{d}t \\
\text{s.t.} \quad & x(0) = x_0 && && \text{(initial value)} \\
& \dot{x}(t) = (1+x)x + u, && t \in [0,3] && \text{(ODE)} \\
& -1 \leq x(t) \leq 1, && t \in [0,3] && \text{(path constraint)} \\
& -1 \leq u(t) \leq 1, && t \in [0,3] && \text{(path constraint)} \\
& x(3) = 0. && && \text{(terminal constraint)}
\end{aligned}
$$

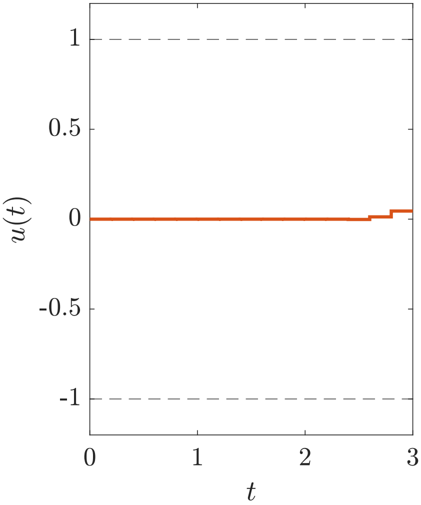
- For $(1+x_0)x_0 \geq 1$, i.e., $x_0 \geq 0.618$ uncontrollable growth
- Choose $N = 15$ equidistant control intervals
- Initialize with steady state control $u(t) = 0$
- Initial value $x_0 = 0.05$ (for higher value trajectory explodes)
- Solve OCP with `IPOPT` via `CasADi`

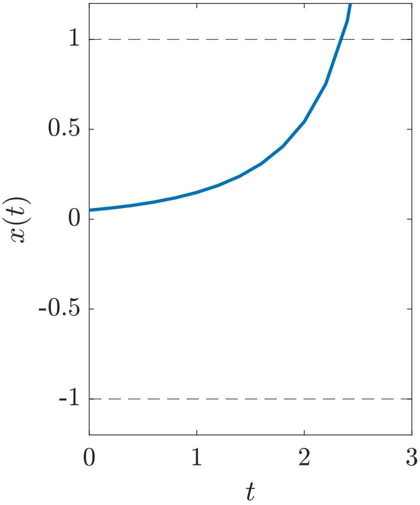
# Numerical example with single shooting: iterations




initialization

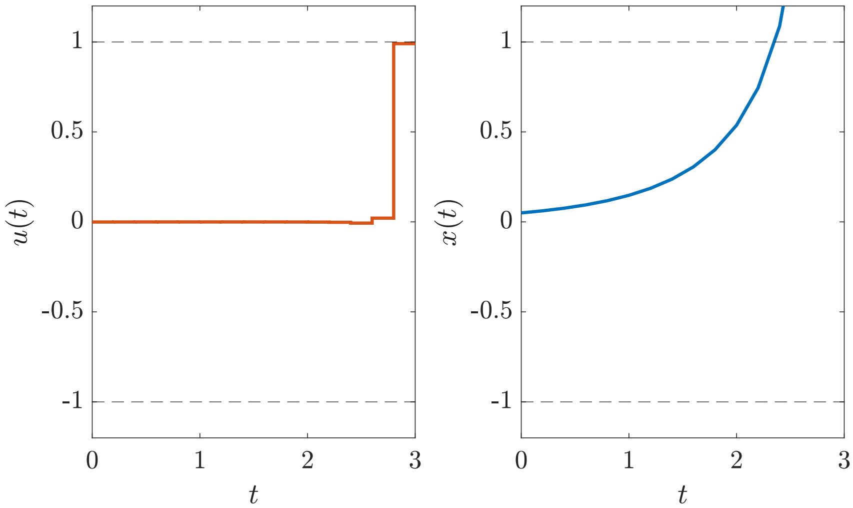
# Numerical example with single shooting: iterations



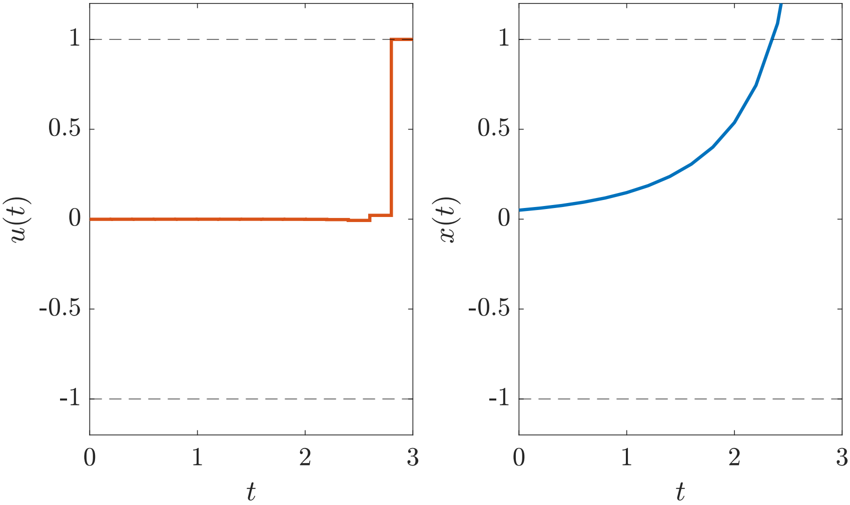

1st iteration

# Numerical example with single shooting: iterations



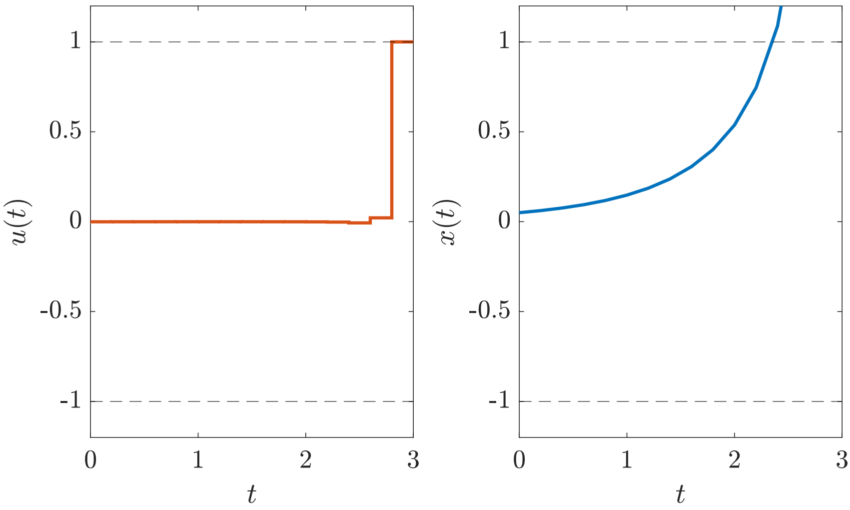

2nd iteration

# Numerical example with single shooting: iterations



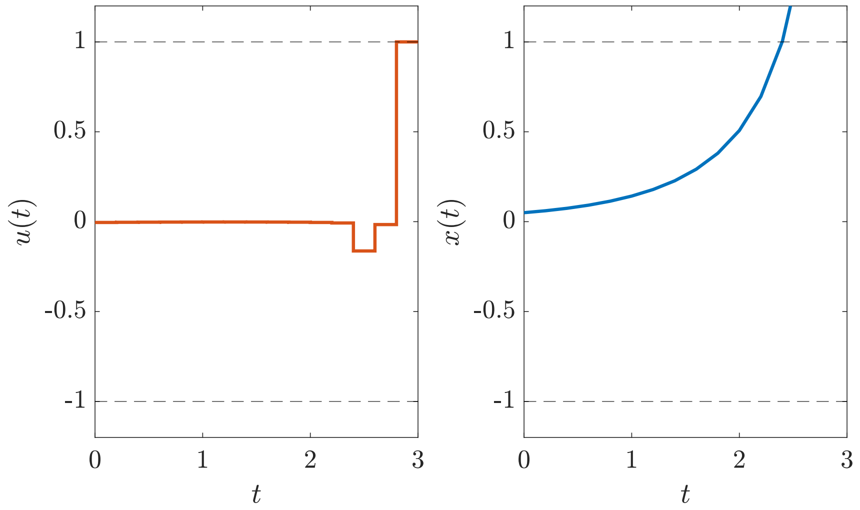

3rd iteration

# Numerical example with single shooting: iterations



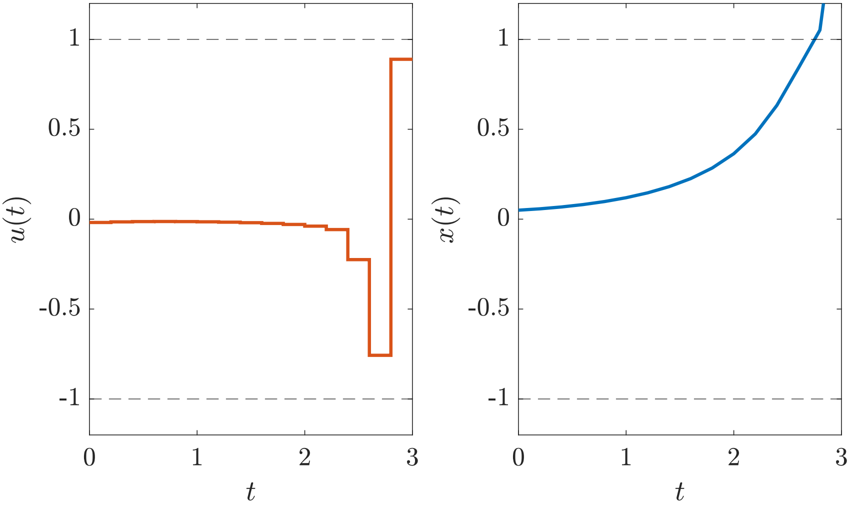

4th iteration

# Numerical example with single shooting: iterations



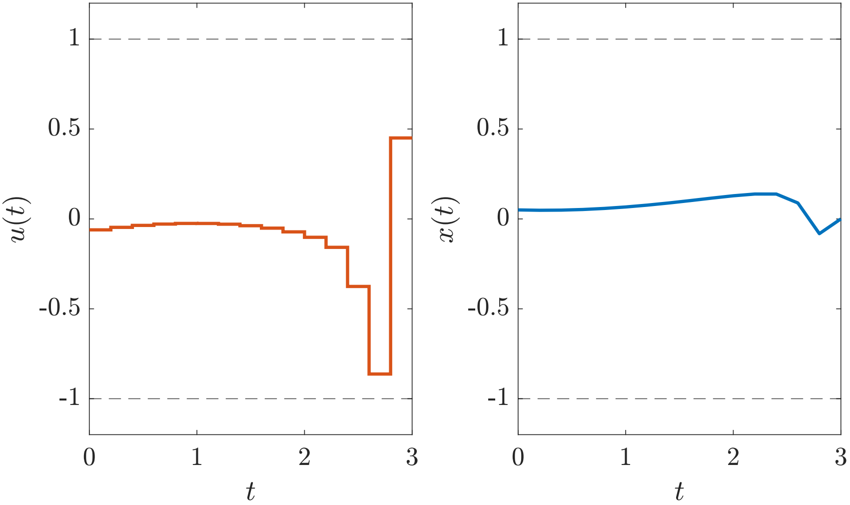

5th iteration

# Numerical example with single shooting: iterations



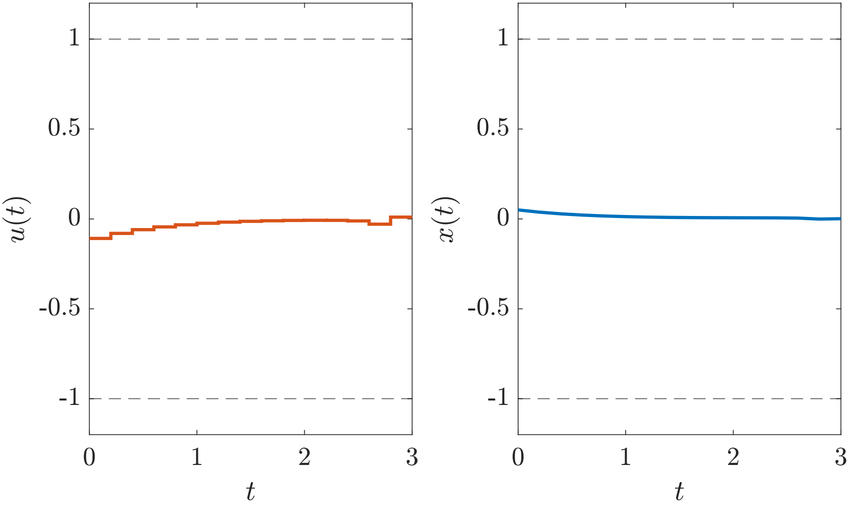

10th iteration

# Numerical example with single shooting: iterations



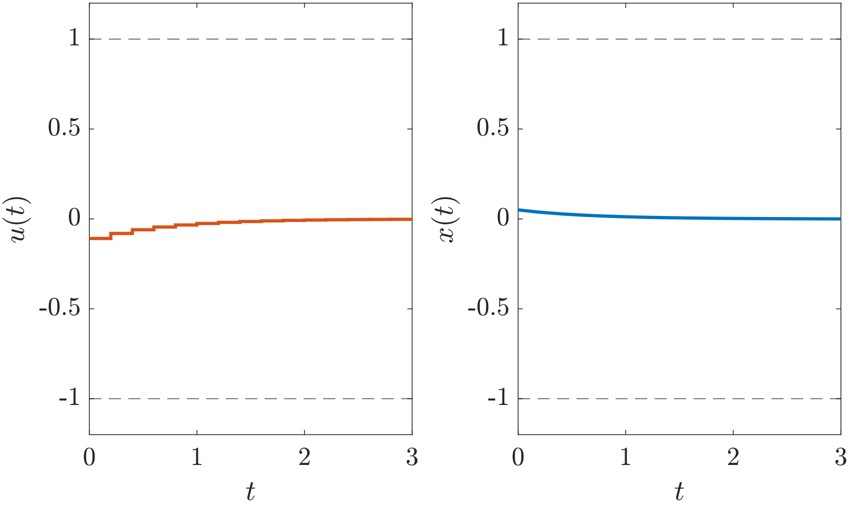

15th iteration

# Numerical example with single shooting: iterations




20th iteration

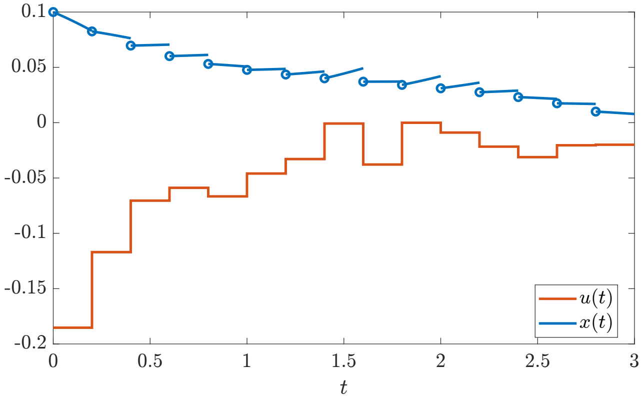
# Numerical example with single shooting: iterations




23rd iteration - **converged!**

# Direct multiple shooting

- discretize controls $u(t) \in \mathbb{R}^{n_u}$ on a fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$
- parametrize controls by $\mathbf{u} = (u_0, \ldots, u_{N-1}) \in \mathbb{R}^{N \cdot n_u}$
- *numerically* solve ODE $\dot{x}(t) = f_c(x(t), u_n)$ on each $[t_n, t_{n+1}]$ with artificial initial value $x(t_n) = x_n$
- new degrees of freedom: $\mathbf{x} = (x_0, \ldots, x_N) \in \mathbb{R}^{(N+1) \cdot n_x}$



# Direct multiple shooting

## Continuous time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

# Direct multiple shooting

**Continuous time OCP**

$$\begin{aligned}\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\; t \in [0,T] \\ & 0 \geq r(x(T))\end{aligned}$$

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.

# Direct multiple shooting

**Continuous time OCP**

$$\min_{x(\cdot),u(\cdot)} \int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T))$$
$$\text{s.t.} \quad x(0) = \bar{x}_0$$
$$\dot{x}(t) = f(x(t),u(t))$$
$$0 \geq h(x(t),u(t)),\ t \in [0,T]$$
$$0 \geq r(x(T))$$

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics via numerical simulation method

$$L_{\mathrm{d}}(x_n,u_n) = \int_{t_n}^{t_{n+1}} L(x(t),u(t))\,\mathrm{d}t.$$

Replace $\dot{x} = f(x,u)$ by

$$x_{n+1} = \psi_f(x_n,u_n)$$

# Direct multiple shooting

**Continuous time OCP**

$$\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \textstyle\int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}$$

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics via numerical simulation method

$$L_{\mathrm{d}}(x_n,u_n) = \int_{t_n}^{t_{n+1}} L(x(t),u(t))\,\mathrm{d}t.$$

Replace $\dot{x} = f(x,u)$ by

$$x_{n+1} = \psi_f(x_n,u_n)$$

3. Relax path constraints, e.g., evaluate only at $t = t_n$

$$0 \geq h(x_n,u_n),\ n = 0,\dots N-1.$$

# Direct multiple shooting

## Continuous time OCP

$$\begin{aligned}\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T))\end{aligned}$$

## Discrete time OCP (an NLP)

$$\begin{aligned}\min_{\mathbf{x},\mathbf{u}} \quad & \sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k,u_k) + E(x_N) \\ \text{s.t.} \quad & x_0 = \bar{x}_0 \\ & x_{n+1} = \psi_f(x_n,u_n) \\ & 0 \geq h(x_n,u_n),\ n = 0,\dots,N-1 \\ & 0 \geq r(x_N)\end{aligned}$$

Variables $\mathbf{x} = (x_0,\dots,x_N)$, and $\mathbf{u} = (u_0,\dots,u_{N-1})$.

# Direct optimal control methods solve Nonlinear Programs (NLP)

**Discrete time OCP (an NLP)**

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \ & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \psi_f(x_n, u_n) \\
& 0 \geq h(x_n, u_n), \ n = 0, \dots, N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $w = (\mathbf{x}, \mathbf{u})$

# Direct optimal control methods solve Nonlinear Programs (NLP)

**Discrete time OCP (an NLP)**

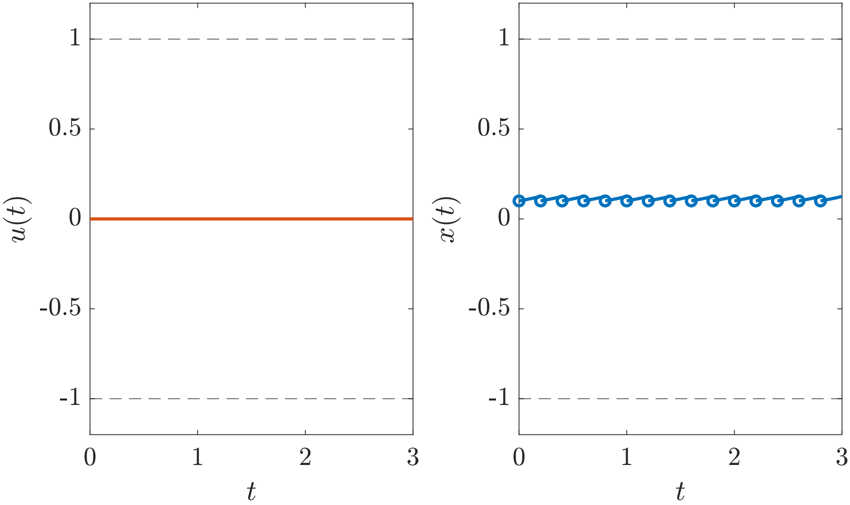
$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \ & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \psi_f(x_n, u_n) \\
& 0 \geq h(x_n, u_n), \ n = 0, \dots, N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $w = (\mathbf{x}, \mathbf{u})$

**Nonlinear Program (NLP)**

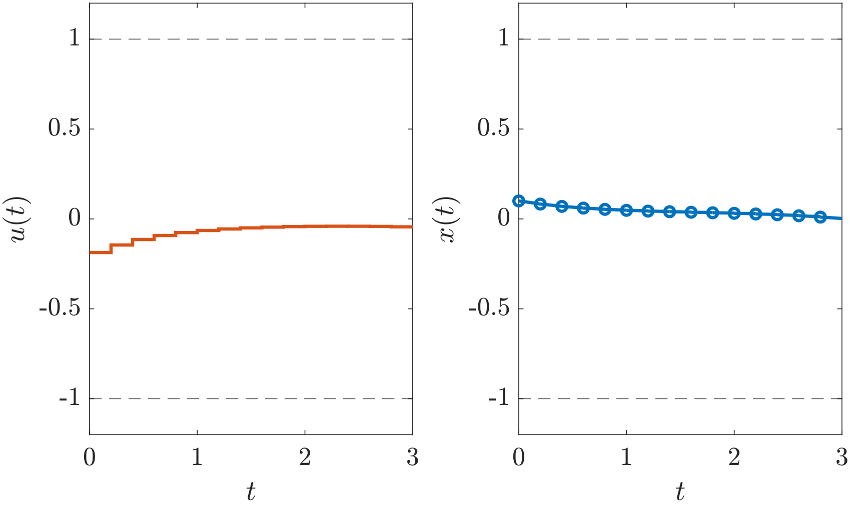
$$
\begin{aligned}
\min_{w \in \mathbb{R}^{n_x}} \ & F(w) \\
\text{s.t.} \ & G(w) = 0 \\
& H(w) \geq 0
\end{aligned}
$$

Obtain larger but sparse NLP



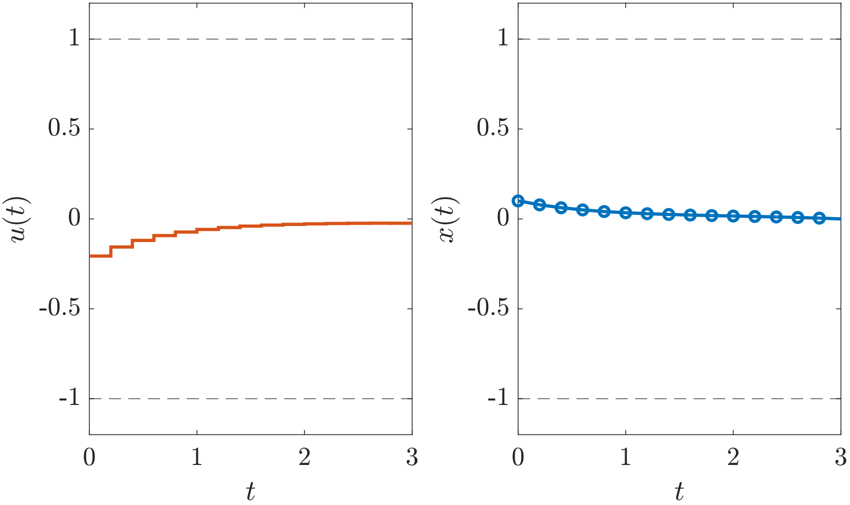

initialization

# Numerical example with multiple shooting: iterations



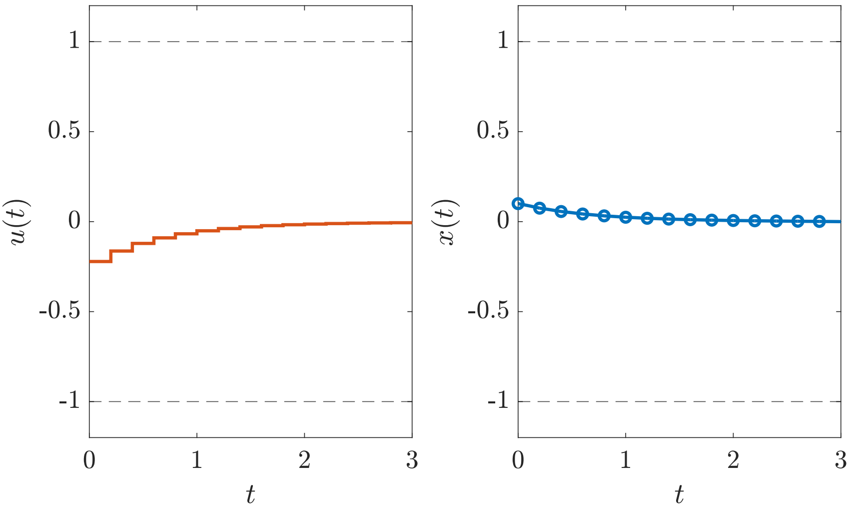

1st iteration

# Numerical example with multiple shooting: iterations



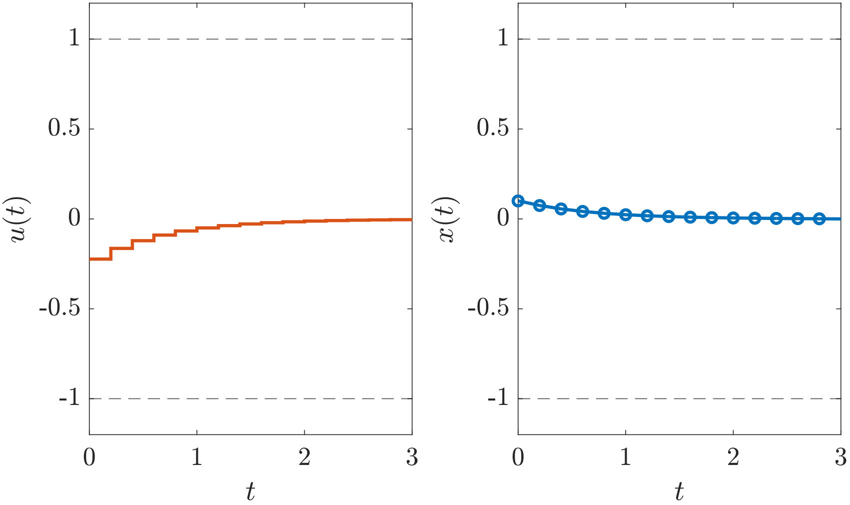

2nd iteration

# Numerical example with multiple shooting: iterations




3rd iteration

# Numerical example with multiple shooting: iterations




5th iteration - **converged!**

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$
- **cannot** use knowledge of $x$ in initialization

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$
+ **can** use knowledge of $x$ in initialization

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$
- **cannot** use knowledge of $x$ in initialization
- ODE solution can depend very non-linearly on $u$

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$
+ **can** use knowledge of $x$ in initialization
+ usually less nonlinear = faster convergence)

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$
- **cannot** use knowledge of $x$ in initialization
- ODE solution can depend very non-linearly on $u$
- unstable systems difficult to treat

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$
+ **can** use knowledge of $x$ in initialization
+ usually less nonlinear = faster convergence)
+ unstable systems easier to treat

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$
- **cannot** use knowledge of $x$ in initialization
- ODE solution can depend very non-linearly on $u$
- unstable systems difficult to treat
- difficult to parallelize

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$
+ **can** use knowledge of $x$ in initialization
+ usually less nonlinear = faster convergence)
+ unstable systems easier to treat
+ easy to parallelize

# Single vs multiple shooting: pros and cons

Single shooting: **sequential** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
+ few degrees of freedom even for large ODE/DAE systems
+ need only initial guess for controls $u$
- **cannot** use knowledge of $x$ in initialization
- ODE solution can depend very non-linearly on $u$
- unstable systems difficult to treat
- difficult to parallelize
- often used in prototype engineering implementations

Multiple shooting: **simultaneous** optimization and simulation:

+ use state-of-the-art ODE/DAE solvers
- more optimization variables, but fixed dimension
+ can pass initial guess for controls $u$
+ **can** use knowledge of $x$ in initialization
+ usually less nonlinear $=$ faster convergence)
+ unstable systems easier to treat
+ easy to parallelize
+ used in many great software packages: MUSCOD-II, ACADO, acados

# Multiple shooting as lifted Newton's method (1/2)

- Just like in our example, it is often observed that multiple shooting converges better and faster than single shooting.
- The multiple shooting discretization leads to more degrees of freedom, and the "nonlinearity is distributed over the variables".
- This is a manifestation of the lifted Newton's method, which was studied in: Albersmeyer, J., & Diehl, M. (2010). The lifted Newton method and its application in optimization. SIAM Journal on Optimization, 20(3), 1655-1684., PDF: `https://publications.syscop.de/Albersmeyer2010.pdf`
- To gain more intuition we illustrate this effect on a small root-finding problem.
- Careful, it may also happen that lifting makes the situation worse. In multiple shooting, this is almost never observed.

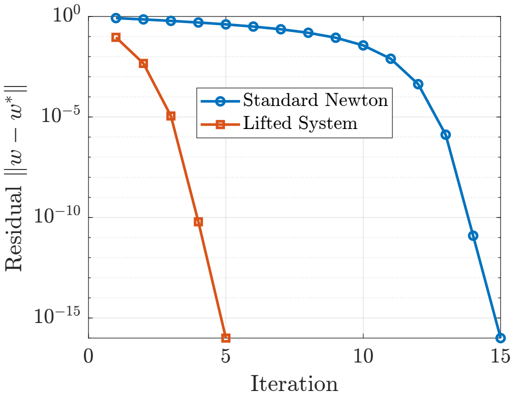
The problem:

$$w^{16} - 2 = 0,$$

has only one variable and is quite nonlinear. An equivalent lifted problem, with more variables but less nonlinear is:

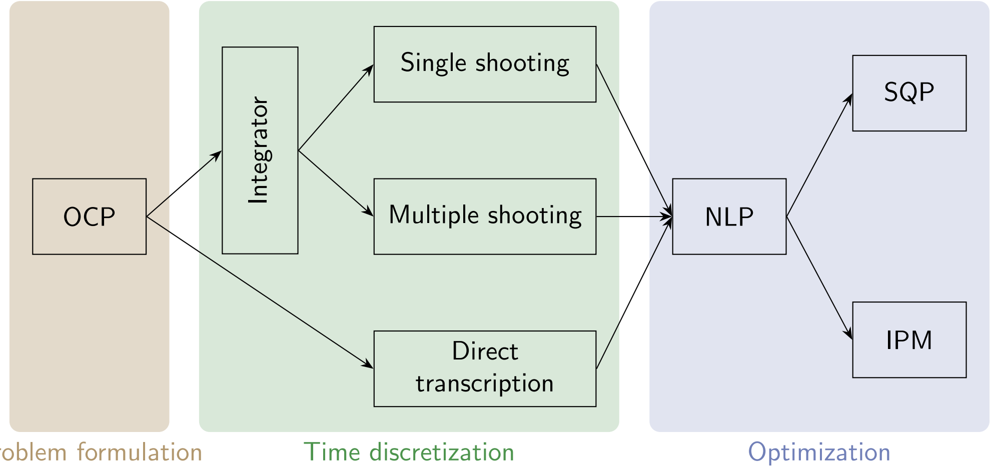
$$\begin{aligned} z_1^2 - 2 &= 0, \\ z_2^2 - z_1 &= 0, \\ z_3^2 - z_2 &= 0, \\ z_4^2 - z_3 &= 0. \end{aligned}$$

- The fist is initialized with $w^0 = 1.5$, the second with $z_i^0 = 1.5, i = 1, \dots, 4$.
- The lifted formulation is less nonlinear and Newton's method converges faster.



# Work flow in smooth direct optimal control

First discretize, then optimize.




OCP = Optimal Control Problem

NLP = Nonlinear Program

SQP = Sequential Quadratic Programming

IPM = Interior-Point Method

Figure inspired by Lecture 1, Numerical Methods for Optimal Control: Introduction, 2022, by Mario Zanon and Sébastien Gros.

# Outline of the lecture

# Ordinary differential equations and controlled dynamical system

Let:

- $t \in \mathbb{R}$ be the time
- $x(t) \in \mathbb{R}^{n_x}$ the differential states
- $u(t) \in \mathbb{R}^{n_u}$ a given control function
- denote by $\dot{x}(t) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$

## Ordinary differential equations

# Ordinary differential equations and controlled dynamical system

Let:

- $t \in \mathbb{R}$ be the time
- $x(t) \in \mathbb{R}^{n_x}$ the differential states
- $u(t) \in \mathbb{R}^{n_u}$ a given control function
- denote by $\dot{x}(t) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$

## Ordinary differential equations

- Let $F : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ be a function such that the Jacobian $\frac{\partial F}{\partial \dot{x}}(\cdot)$ is invertible. The system of equations:

$$F(t, \dot{x}(t), x(t), u(t)) = 0,$$

is called an Ordinary Differential Equation (ODE).

# Ordinary differential equations and controlled dynamical system

Let:

- $t \in \mathbb{R}$ be the time
- $x(t) \in \mathbb{R}^{n_x}$ the differential states
- $u(t) \in \mathbb{R}^{n_u}$ a given control function
- denote by $\dot{x}(t) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$

## Ordinary differential equations

- Let $F : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ be a function such that the Jacobian $\frac{\partial F}{\partial \dot{x}}(\cdot)$ is invertible. The system of equations:

$$F(t, \dot{x}(t), x(t), u(t)) = 0,$$

  is called an Ordinary Differential Equation (ODE).
- Given a function $f : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ then a system of equations:

$$\dot{x}(t) = f(t, x(t), u(t)) \tag{1}$$

  is called an explicit ODE.

# Sufficient conditions for existence and uniqueness

**Theorem (Picard-Lindelöf / Cauchy–Lipschitz )**

*An initial value problem in ODE*

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u(t)), \quad t \in [0, T], \\ x(0) &= x_0 \end{aligned}$$

- *with given initial state* $x_0$*, and controls* $u(t)$*,*
- $f(t, x(t), u(t)) = \hat{f}(t, x(t))$ *is continuous in* $t$ *and Lipschitz continuous in* $x$

# Sufficient conditions for existence and uniqueness

**Theorem (Picard-Lindelöf / Cauchy–Lipschitz )**

*An initial value problem in ODE*

$$\begin{aligned}\dot{x}(t) &= f(t, x(t), u(t)), \quad t \in [0, T],\\ x(0) &= x_0\end{aligned}$$

- *with given initial state* $x_0$, *and controls* $u(t)$,
- $f(t, x(t), u(t)) = \hat{f}(t, x(t))$ *is continuous in* $t$ *and Lipschitz continuous in* $x$

*has a unique solution* $x(t),\ t \in [0, T]$.

- $f$ is Lipschitz if $\|f(x) - f(y)\| \leq L\|x - y\|$
- smooth ODEs modeling physics usually Lipschitz

# Sufficient conditions for existence and uniqueness

## Theorem (Picard-Lindelöf / Cauchy–Lipschitz )

*An initial value problem in ODE*

$$\dot{x}(t) = f(t, x(t), u(t)), \quad t \in [0, T],$$
$$x(0) = x_0$$

- *with given initial state* $x_0$, *and controls* $u(t)$,
- $f(t, x(t), u(t)) = \hat{f}(t, x(t))$ *is continuous in* $t$ *and Lipschitz continuous in* $x$

*has a unique solution* $x(t)$, $t \in [0, T]$.

- $f$ is Lipschitz if $\|f(x) - f(y)\| \leq L\|x - y\|$
- smooth ODEs modeling physics usually Lipschitz
- if $f$ is only continuous, existence but not uniqueness can be guaranteed, e.g. $\dot{x}(t) = \sqrt{|x(t)|}, x(0) = 0$, solutions: $x(t) = 0$ and $x(t) = \frac{t^2}{4}$

# Sufficient conditions for existence and uniqueness

**Theorem (Picard-Lindelöf / Cauchy–Lipschitz )**

*An initial value problem in ODE*

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u(t)), \quad t \in [0, T], \\ x(0) &= x_0 \end{aligned}$$

- *with given initial state* $x_0$*, and controls* $u(t)$*,*
- $f(t, x(t), u(t)) = \hat{f}(t, x(t))$ *is continuous in* $t$ *and Lipschitz continuous in* $x$

*has a unique solution* $x(t),\ t \in [0, T]$*.*

- $f$ is Lipschitz if $\|f(x) - f(y)\| \leq L\|x - y\|$
- smooth ODEs modeling physics usually Lipschitz
- if $f$ is only continuous, existence but not uniqueness can be guaranteed, e.g. $\dot{x}(t) = \sqrt{|x(t)|}, x(0) = 0$, solutions: $x(t) = 0$ and $x(t) = \frac{t^2}{4}$
- Conditions are only sufficient, ODEs with a non-Lipschitz r.h.s. can have unique solutions

A collection of results in: *Agarwal, Ratan Prakash, Ravi P. Agarwal, and V. Lakshmikantham. Uniqueness and nonuniqueness criteria for ordinary differential equations. Vol. 6. World Scientific, 1993.*

# Basic definitions of numerical simulation

- IVPs have only in special cases a closed form solution
- Instead, compute numerically a solution approximation $\tilde{x}(t)$ that approximately satisfies:

$$\dot{\tilde{x}}(t) \approx f(t, \tilde{x}(t), u(t)), \quad t \in [0, T]$$
$$\tilde{x}(0) = x(0) = x_0$$

# Basic definitions of numerical simulation

- IVPs have only in special cases a closed form solution
- Instead, compute numerically a solution approximation $\tilde{x}(t)$ that approximately satisfies:

$$\begin{aligned} \dot{\tilde{x}}(t) &\approx f(t, \tilde{x}(t), u(t)), \quad t \in [0, T] \\ \tilde{x}(0) &= x(0) = x_0 \end{aligned}$$

- Recursively generate solution approximation $x_n := \tilde{x}(t_n) \approx x(t_n)$ at $N$ discrete time points $0 = t_0 < t_1 < \ldots < t_N = T$
- Integration interval $[0, T]$ split into subintervals $[t_n, t_{n+1}]$ where $h = t_{n+1} - t_n$
- $h$ - integration step size can be constant, different for every interval, or adaptive

# Single step numerical simulation as discrete time system

## Single step abstract integration method

$$
\begin{aligned}
x_{n+1} &= \phi_f(x_n, z_n, u_n),\\
0 &= \phi_{\mathrm{int}}(x_n, z_n, u_n),\;\; n = 0, \ldots, N-1.
\end{aligned}
$$

- $\phi_f$ - state transition - compute next integration step
- $\phi_{\mathrm{int}}$ - internal computations, e.g., stages of a Runge-Kutta method (next section)
- $z_n$ collects all interval variables of the integration method

# Single step numerical simulation as discrete time system

## Single step abstract integration method

$$
\begin{aligned}
x_{n+1} &= \phi_f(x_n, z_n, u_n), \\
0 &= \phi_{\text{int}}(x_n, z_n, u_n), \;\; n = 0, \dots, N-1.
\end{aligned}
$$

- $\phi_f$ - state transition - compute next integration step
- $\phi_{\text{int}}$ - internal computations, e.g., stages of a Runge-Kutta method (next section)
- $z_n$ collects all interval variables of the integration method

Example (Explicit Euler):

$$x_{n+1} = x_n + hf(x_n, u_n),$$

# Single step numerical simulation as discrete time system

## Single step abstract integration method

$$
\begin{aligned}
x_{n+1} &= \phi_f(x_n, z_n, u_n), \\
0 &= \phi_{\mathrm{int}}(x_n, z_n, u_n), \ \ n = 0, \dots, N-1.
\end{aligned}
$$

- $\phi_f$ - state transition - compute next integration step
- $\phi_{\mathrm{int}}$ - internal computations, e.g., stages of a Runge-Kutta method (next section)
- $z_n$ collects all interval variables of the integration method

Example (Explicit Euler):

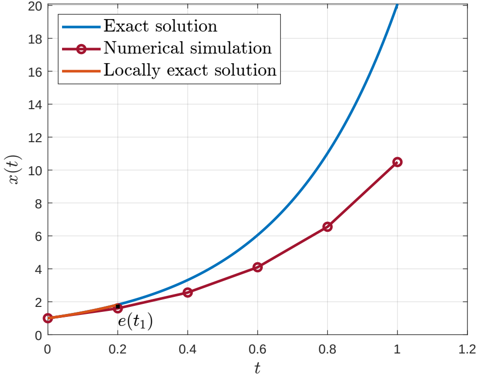
$$
\begin{aligned}
x_{n+1} &= x_n + h z_n, \\
0 &= f(x_n, u_n) - z_n.
\end{aligned}
$$

# Integration error

## Local and global error

- Local integration error at $t_{n+1}$:

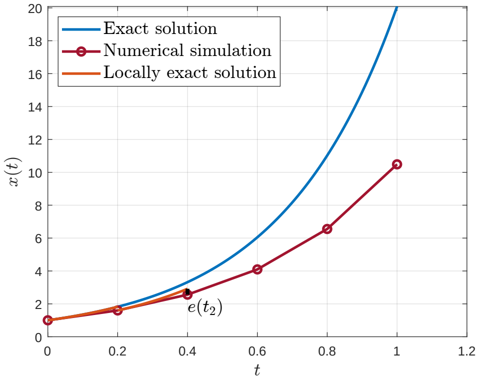
$$e(t_{n+1}) = \|x(t_{n+1}) - \phi_f(x(t_n), z_n, u_0)\|.$$



### Local and global error

- Local integration error at $t_{n+1}$:

$$e(t_{n+1}) = \|x(t_{n+1}) - \phi_f(x(t_n), z_n, u_0)\|.$$

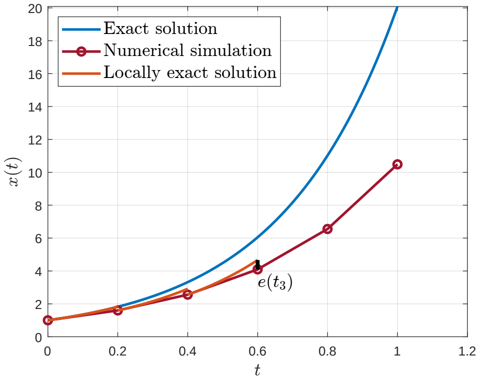


# Integration error

## Local and global error

- Local integration error at $t_{n+1}$:

$$e(t_{n+1}) = \|x(t_{n+1}) - \phi_f(x(t_n), z_n, u_0)\|.$$

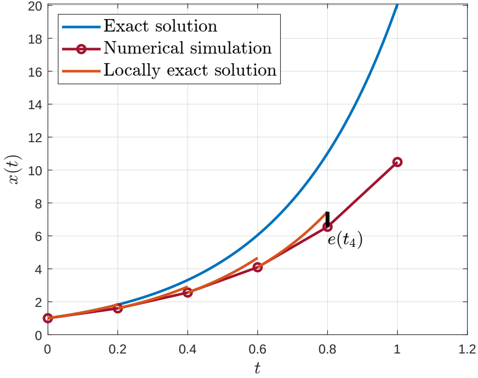


# Integration error

## Local and global error

- Local integration error at $t_{n+1}$:

$$e(t_{n+1}) = \|x(t_{n+1}) - \phi_f(x(t_n), z_n, u_0)\|.$$



# Integration error

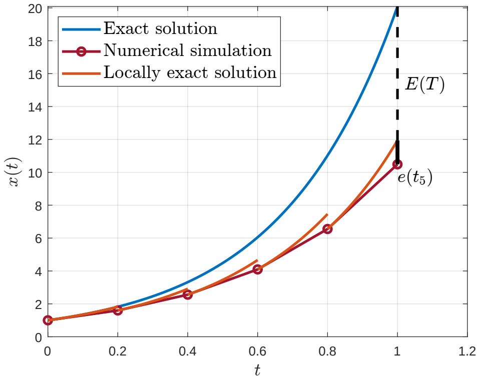
## Local and global error

- Local integration error at $t_{n+1}$:

$$e(t_{n+1}) = \|x(t_{n+1}) - \phi_f(x(t_n), z_n, u_0)\|.$$

- Global integration error at $t = T$:

$$E(T) = \|x(T) - x_N\|.$$




- Global error - accumulation of local errors

# Convergence and integrator order

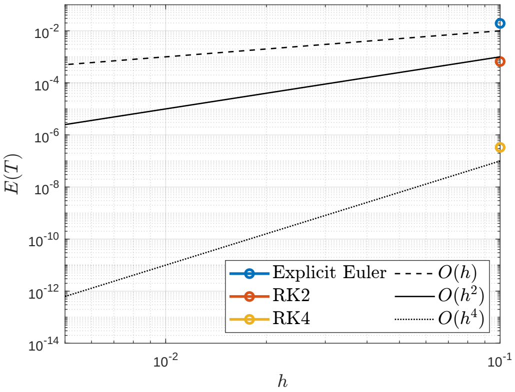
## Integrator convergence and accuracy

- Convergence

$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$




- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy

# Convergence and integrator order

## Integrator convergence and accuracy

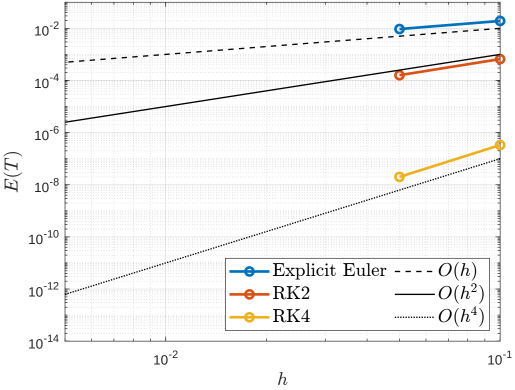
- Convergence

$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy



# Convergence and integrator order

## Integrator convergence and accuracy

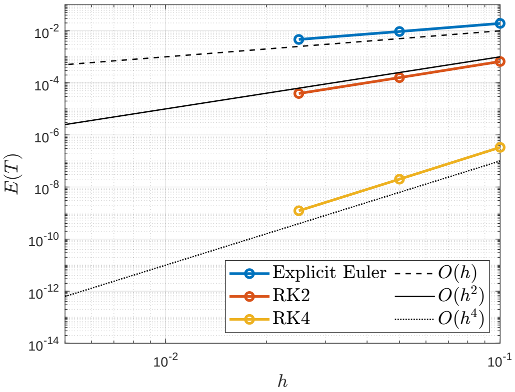
- Convergence

$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy



# Convergence and integrator order

## Integrator convergence and accuracy

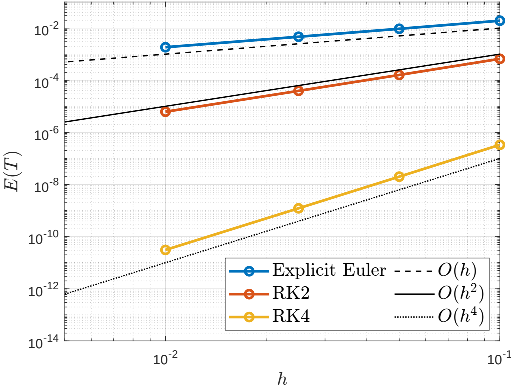
- Convergence

$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy



# Convergence and integrator order

## Integrator convergence and accuracy

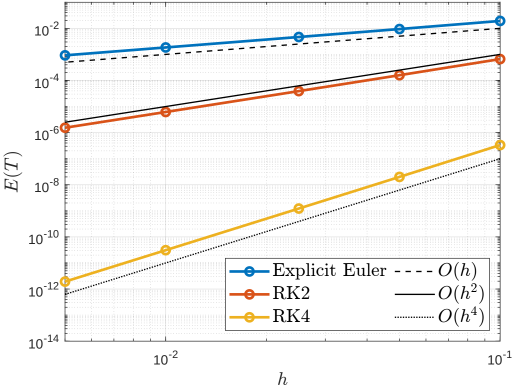
- Convergence

$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq Ch^{p+1} = O(h^{p+1}), C > 0$$

- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy



# Convergence and integrator order

## Integrator convergence and accuracy

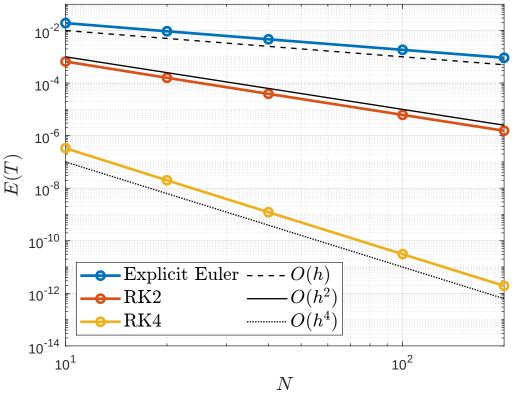
- Convergence

$$\lim_{h\to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h\to 0} e(t_i) \leq Ch^{p+1} = O(h^{p+1}), C > 0$$

- Higher order $p$:
  - less, but more expensive steps for same accuracy
  - in total fewer r.h.s. evaluations for same accuracy




Alternatively one can plot the error over $N \propto \frac{1}{h}$ instead of $h$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

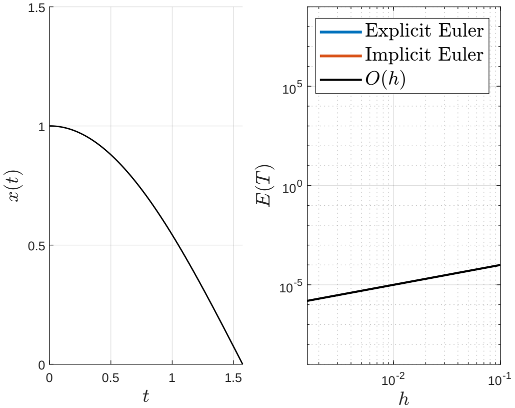
$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \; t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

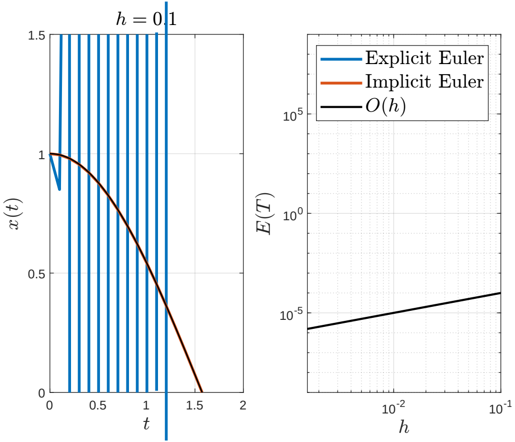
$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \;\; t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

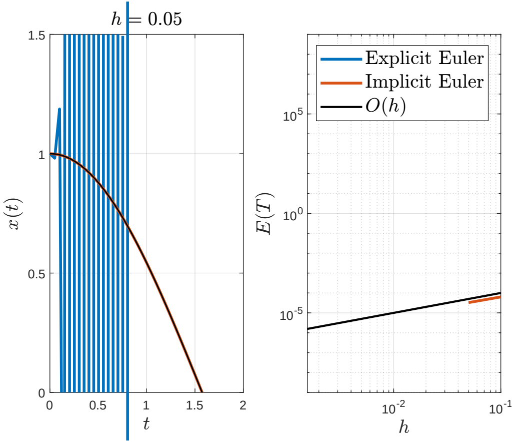
$$\lim_{h\to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h\to 0} e(t_i) \leq Ch^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \;\; t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

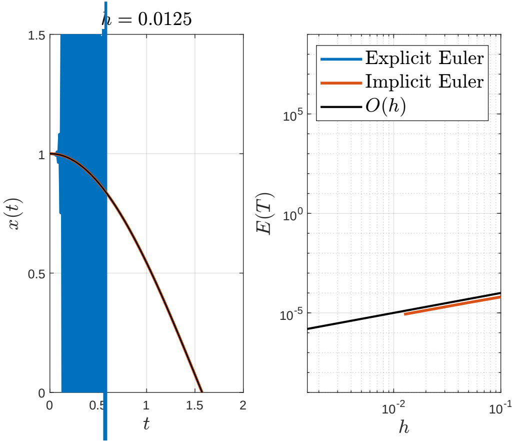
$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \ t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

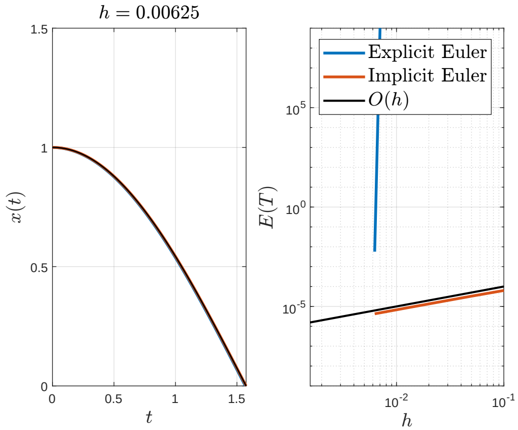
$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \ t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

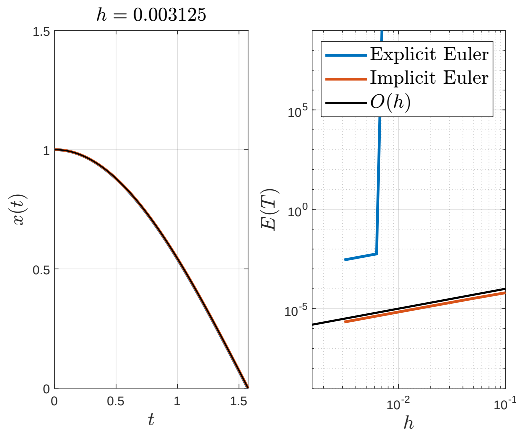
$$\lim_{h \to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h \to 0} e(t_i) \leq C h^{p+1} = O(h^{p+1}), C > 0$$

- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)), \ t \in [0, 2]$$
$$x(0) = 1$$

# Stability and convergence

## Integrator convergence and accuracy

- Convergence

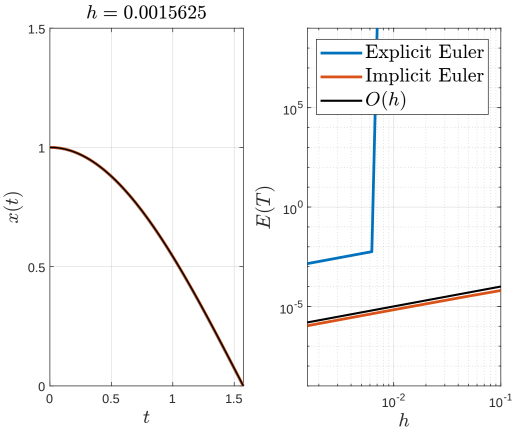
$$\lim_{h\to 0} E(T) = 0$$

- Integrator has order $p$ if

$$\lim_{h\to 0} e(t_i) \leq Ch^{p+1} = O(h^{p+1}), C > 0$$

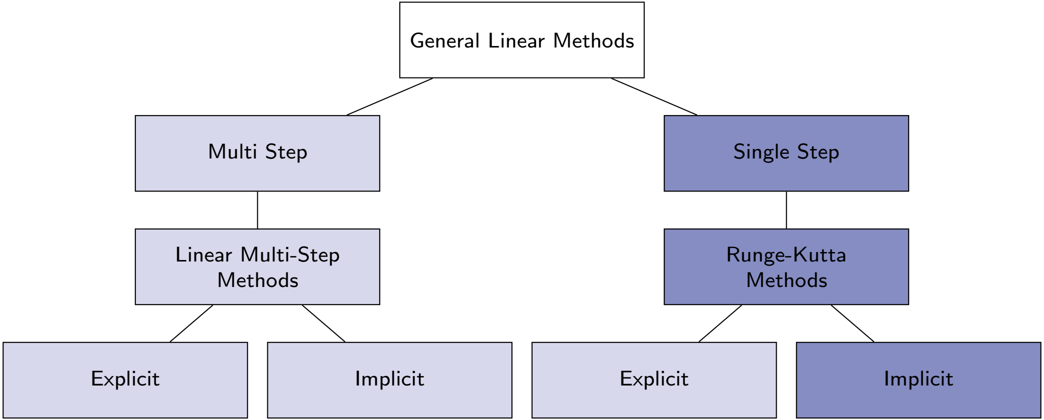
- Stability: damping of errors, does it work for $h \gg 0$?

- If integrator is unstable, it does not converge and has $p = 0$, unless $h$ very small




$$\dot{x}(t) = -300(x(t) - \cos(t)),\ \ t \in [0, 2]$$
$$x(0) = 1$$

# Classes of numerical simulation methods



# Runge-Kutta method definition

## Unknowns are derivatives at stage points

### Definition (Runge-Kutta method in differential form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \ldots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$
\begin{aligned}
k_{n,i} &= f(t_{n,i}, x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, u_n), \ i = 1, \ldots, n_\mathrm{s} \\
x_{n+1} &= x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}
\end{aligned}
$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in the *differential form*.

# Runge-Kutta method definition

Unknowns are derivatives at stage points

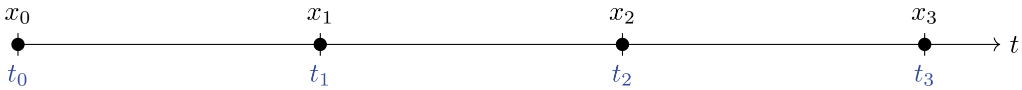
## Definition (Runge-Kutta method in differential form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \dots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$k_{n,i} = f(t_{n,i}, x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, u_n), \; i = 1, \dots, n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in the *differential form*.



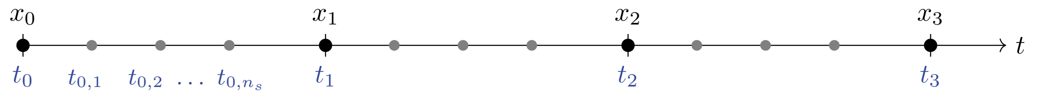
## Definition (Runge-Kutta method in differential form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i,j = 1,\dots,n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$k_{n,i} = f(t_{n,i}, x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, u_n),\ i = 1,\dots,n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in the *differential form*.



# Runge-Kutta method definition

Unknowns are derivatives at stage points

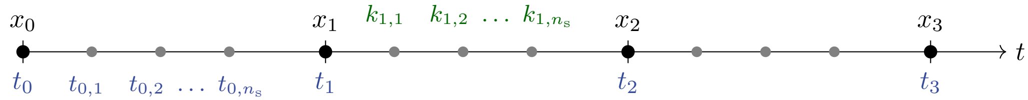
## Definition (Runge-Kutta method in differential form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \dots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$k_{n,i} = f(t_{n,i}, x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, u_n),\ i = 1, \dots, n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in the *differential form*.



# Runge-Kutta method definition

## Unknowns are derivatives at stage points

### Definition (Runge-Kutta method in differential form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \dots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$k_{n,i} = f(t_{n,i}, x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, u_n),\ i = 1, \dots, n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in the *differential form*.

| **Time grid** | **Butcher tableau** | **Data** | **Variables** |
|---|---|---|---|
| $h,\ t_n,\ t_{n,i}$ | $a_{i,j},\ b_i,\ c_i$ | $x_n,\ u_n,\ f(\cdot)$ | $x_{n+1},\ k_{n,i}$ |
| $i = 1, \dots, n_\mathrm{s}$ | $i, j = 1, \dots, n_\mathrm{s}$ | | $i = 1, \dots, n_\mathrm{s}$ |

# Runge-Kutta method examples

### Explicit Runge-Kutta 4

$$
\begin{aligned}
k_{n,1} &= f(t_n, x_n) \\
k_{n,2} &= f(t_n + \frac{h}{2}, x_n + h\frac{k_{n,1}}{2}) \\
k_{n,3} &= f(t_n + \frac{h}{2}, x_n + h\frac{k_{n,2}}{2}) \\
k_{n,4} &= f(t_n + h, x_n + hk_{n,3}) \\
x_{n+1} &= x_n + h(\frac{1}{6}k_{n,1} + \frac{2}{6}k_{n,2} + \frac{2}{6}k_{n,3} + \frac{1}{6}k_{n,4})
\end{aligned}
$$

- All $k_{n,i}$ can be found by explicit function evaluations.

# Runge-Kutta method examples

## Explicit Runge-Kutta 4

$$
\begin{aligned}
k_{n,1} &= f(t_n, x_n)\\
k_{n,2} &= f(t_n + \frac{h}{2}, x_n + h\frac{k_{n,1}}{2})\\
k_{n,3} &= f(t_n + \frac{h}{2}, x_n + h\frac{k_{n,2}}{2})\\
k_{n,4} &= f(t_n + h, x_n + hk_{n,3})\\
x_{n+1} &= x_n + h(\frac{1}{6}k_{n,1} + \frac{2}{6}k_{n,2} + \frac{2}{6}k_{n,3} + \frac{1}{6}k_{n,4})
\end{aligned}
$$

- All $k_{n,i}$ can be found by explicit function evaluations.

## Implicit Euler Method

$$
\begin{aligned}
k_{n,1} &= f(t_{n+1}, x_n + hk_{n,1})\\
x_{n+1} &= x_n + hk_{n,1}
\end{aligned}
$$

- All $k_{n,1}$ is found implicitly by solving $k_{n,1} - f(t_n, x_n + hk_{n,1}) = 0.$

# Explicit vs implicit Runge-Kutta methods

## The Butcher tableau

### Explicit Runge-Kutta method

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{2,1} & & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}}
\end{array}
$$

# Explicit vs implicit Runge-Kutta methods

## The Butcher tableau

### Explicit Runge-Kutta method

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{2,1} & & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}}
\end{array}
$$

- $a_{i,j} \neq 0$ only for $j < i$
- Explicit function evaluations to compute stage values and next step
- Computationally cheap
- Order: $p = n_\mathrm{s}$ if $n_\mathrm{s} \leq 4$ and $p < n_\mathrm{s}$ otherwise

# Explicit vs implicit Runge-Kutta methods

## The Butcher tableau

### Explicit Runge-Kutta method

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{2,1} & & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}}
\end{array}$$

- $a_{i,j} \neq 0$ only for $j < i$
- Explicit function evaluations to compute stage values and next step
- Computationally cheap
- Order: $p = n_\mathrm{s}$ if $n_\mathrm{s} \leq 4$ and $p < n_\mathrm{s}$ otherwise

### Implicit Runge-Kutta method

$$\begin{array}{c|ccccc}
c_1 & a_{1,1} & a_{1,2} & \dots & a_{1,n_\mathrm{s}-1} & a_{1,n_\mathrm{s}} \\
c_2 & a_{2,1} & a_{2,2} & \dots & a_{2,n_\mathrm{s}-1} & a_{2,n_\mathrm{s}} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & a_{n_\mathrm{s},n_\mathrm{s}} \\
\hline
 & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}}
\end{array}$$

# Explicit vs implicit Runge-Kutta methods

## The Butcher tableau

### Explicit Runge-Kutta method

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{2,1} & & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & \\ \hline & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}} \end{array}$$

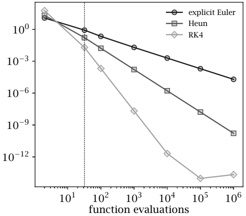
- $a_{i,j} \neq 0$ only for $j < i$
- Explicit function evaluations to compute stage values and next step
- Computationally cheap
- Order: $p = n_\mathrm{s}$ if $n_\mathrm{s} \leq 4$ and $p < n_\mathrm{s}$ otherwise

### Implicit Runge-Kutta method

$$\begin{array}{c|ccccc} c_1 & a_{1,1} & a_{1,2} & \dots & a_{1,n_\mathrm{s}-1} & a_{1,n_\mathrm{s}} \\ c_2 & a_{2,1} & a_{2,2} & \dots & a_{2,n_\mathrm{s}-1} & a_{2,n_\mathrm{s}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n_\mathrm{s}} & a_{n_\mathrm{s},1} & a_{n_\mathrm{s},2} & \dots & a_{n_\mathrm{s},n_\mathrm{s}-1} & a_{n_\mathrm{s},n_\mathrm{s}} \\ \hline & b_1 & b_2 & \dots & b_{n_\mathrm{s}-1} & b_{n_\mathrm{s}} \end{array}$$

- Requires solving nonlinear rootfinding problem with Newton's method
- Expensive but good for stiff systems
- Order: $p = 2n_\mathrm{s}$, $p = 2n_\mathrm{s} - 1$, ...
- Famous representative: collocation methods - treated next!

# Butcher tableau, six examples




Euler

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Heun

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

RK4

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Implicit Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Midpoint rule (GL2)

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

Gauss-Legendre of order 4 (GL4)

$$\begin{array}{c|cc} 1/2-\sqrt{3}/6 & 1/4 & 1/4-\sqrt{3}/6 \\ 1/2+\sqrt{3}/6 & 1/4+\sqrt{3}/6 & 1/4 \\ \hline & 1/2 & 1/2 \end{array}$$

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ c_2 & a_{21} & \cdots & a_{2s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

# Differential-Algebraic Equations (DAE) - semi-explicit form

Extend the ODE by algebraic equations $g$ and algebraic states $z$:

$$\begin{aligned}\dot{x}(t) &= f(t, x(t), u(t), z(t))\\ 0 &= g(t, x(t), z(t), u(t))\end{aligned}$$

- differential states: $x(t) \in \mathbb{R}^{n_x}$
- algebraic states: $z(t) \in \mathbb{R}^{n_z}$
- control input: $u(t) \in \mathbb{R}^{n_u}$

- Source problems: conservation laws, fast dynamics: $\epsilon \dot{z}(t) = g(x(t), z(t), u(t))$, $\epsilon \to 0$.
- The usual case is index one, i.e. the Jacobian $\frac{\partial g}{\partial z}$ is invertible. If this is not the case, replace $g(x, z, u) = 0$ by $\frac{\mathrm{d}^d}{\mathrm{d}t^d} g = 0$, until it is (higher index).

# Runge-Kutta methods for differential algebraic equations

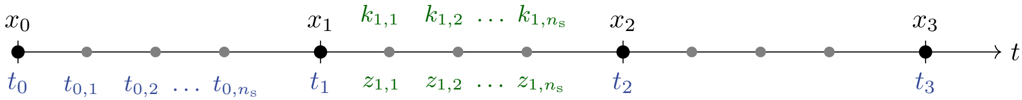
Unknowns are derivatives of states $k_{i,j}$ and algebraic states $z_{i,j}$ at stage points

## Definition (RK method for index 1 DAEs)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \ldots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$.

$$
\begin{aligned}
k_{n,i} &= f\big(t_{n,i}, x_n + h \textstyle\sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, z_{n,i}, u_n\big), && i = 1, \ldots, n_\mathrm{s} \\
0 &= g\big(t_{n,i}, x_n + h \textstyle\sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, z_{n,i}, u_n\big), && i = 1, \ldots, n_\mathrm{s} \\
x_{n+1} &= x_n + h \textstyle\sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}, \\
0 &= g(t_{n+1}, x_{n+1}, z_{n+1}, u_n).
\end{aligned}
$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method for DAEs of index 1. Here $z_{n,i}$, $i = 1, \ldots, n_\mathrm{s}$ are the stage values for the algebraic variables and $z_{n+1}$ is the approximation of $z(t_{n+1})$.



# Runge-Kutta methods for differential algebraic equations

Unknowns are derivatives of states $k_{i,j}$ and algebraic states $z_{i,j}$ at stage points

## Definition (RK method for index 1 DAEs)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \dots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$.

$$
\begin{aligned}
k_{n,i} &= f(t_{n,i}, x_n + h \textstyle\sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, z_{n,i}, u_n), && i = 1, \dots, n_\mathrm{s} \\
0 &= g(t_{n,i}, x_n + h \textstyle\sum_{j=1}^{n_\mathrm{s}} a_{i,j} k_{n,j}, z_{n,i}, u_n), && i = 1, \dots, n_\mathrm{s} \\
x_{n+1} &= x_n + h \textstyle\sum_{i=1}^{n_\mathrm{s}} b_i k_{n,i}, \\
0 &= g(t_{n+1}, x_{n+1}, z_{n+1}, u_n).
\end{aligned}
$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method for DAEs of index 1. Here $z_{n,i},\ i = 1, \dots, n_\mathrm{s}$ are the stage values for the algebraic variables and $z_{n+1}$ is the approximation of $z(t_{n+1})$.

| **Time grid** | **Butcher tableau** | **Data** | **Variables** |
|---|---|---|---|
| $h,\ t_n,\ t_{n,i}$ | $a_{i,j},\ b_i,\ c_i$ | $x_n,\ u_n,\ f(\cdot)$ | $x_{n+1},\ k_{n,i},\ z_{n+1},\ z_{n,i}$ |
| $i = 1, \dots, n_\mathrm{s}$ | $i, j = 1, \dots, n_\mathrm{s}$ | | $i = 1, \dots, n_\mathrm{s}$ |

# Outline of the lecture

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_{\mathrm{s}}$

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_\mathrm{s}$
- Pick $n_\mathrm{s}$ distinct numbers: $0 \leq c_1 < c_2 < \ldots < c_{n_\mathrm{s}} \leq 1$

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_\mathrm{s}$
- Pick $n_\mathrm{s}$ distinct numbers: $0 \leq c_1 < c_2 < \ldots < c_{n_\mathrm{s}} \leq 1$
- Define **collocation points** $t_{n,i} = t_n + c_i h \ , \ i = 1, \ldots, n_\mathrm{s}$

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_\mathrm{s}$
- Pick $n_\mathrm{s}$ distinct numbers: $0 \leq c_1 < c_2 < \ldots < c_{n_\mathrm{s}} \leq 1$
- Define **collocation points** $t_{n,i} = t_n + c_i h \ , \ i = 1, \ldots, n_\mathrm{s}$
- The polynomial $q_n(t) \approx x(t)$ satisfies the ODE on the collocation points:

## Collocation equations

$$\begin{aligned} q_n(t_n) &= x_n \\ \dot{q}_n(t_n + c_i h) &= f(t_n + c_i h, q_n(t_n + c_i h), u_n), \quad i = 1, \ldots, n_\mathrm{s} \end{aligned}$$

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_\mathrm{s}$
- Pick $n_\mathrm{s}$ distinct numbers: $0 \leq c_1 < c_2 < \ldots < c_{n_\mathrm{s}} \leq 1$
- Define **collocation points** $t_{n,i} = t_n + c_i h \ , \ i = 1, \ldots, n_\mathrm{s}$
- The polynomial $q_n(t) \approx x(t)$ satisfies the ODE on the collocation points:

**Collocation equations**

$$
\begin{aligned}
q_n(t_n) &= x_n \\
\dot{q}_n(t_n + c_i h) &= f(t_n + c_i h, q_n(t_n + c_i h), u_n), \quad i = 1, \ldots, n_\mathrm{s}
\end{aligned}
$$

- Polynomial of degree $n_\mathrm{s}$: $n_\mathrm{s} + 1$ coefficient and $n_\mathrm{s} + 1$ equations

# Collocation

**Main ideas:**

- Approximate $x(t)$ on $t \in [t_n, t_{n+1}]$ with a **polynomial** $q_n(t)$ of degree $n_\mathrm{s}$
- Pick $n_\mathrm{s}$ distinct numbers: $0 \leq c_1 < c_2 < \ldots < c_{n_\mathrm{s}} \leq 1$
- Define **collocation points** $t_{n,i} = t_n + c_i h$ , $i = 1, \ldots, n_\mathrm{s}$
- The polynomial $q_n(t) \approx x(t)$ satisfies the ODE on the collocation points:

**Collocation equations**

$$
\begin{aligned}
q_n(t_n) &= x_n \\
\dot{q}_n(t_n + c_i h) &= f(t_n + c_i h, q_n(t_n + c_i h), u_n), \quad i = 1, \ldots, n_\mathrm{s}
\end{aligned}
$$

- Polynomial of degree $n_\mathrm{s}$: $n_\mathrm{s} + 1$ coefficient and $n_\mathrm{s} + 1$ equations
- Next value - simple evaluation: $x_{n+1} = q_n(t_{n+1})$

# Collocation - how to implement it?

**How to parameterize $q_n(t)$?**

Two common (equivalent) choices

# Collocation - how to implement it?

## How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \ldots, k_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \ldots, n_\mathrm{s}$ (**this lecture**).

# Collocation - how to implement it?

## How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \ldots, k_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \ldots, n_\mathrm{s}$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through $x_n$ (at $t_n$) and **state values** $x_{n,1}, \ldots, x_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \ldots, n_\mathrm{s}$ (in Appendix).

# Collocation - how to implement it?

### How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \dots, k_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \dots, n_\mathrm{s}$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through $x_n$ (at $t_n$) and **state values** $x_{n,1}, \dots, x_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \dots, n_\mathrm{s}$ (in Appendix).

- $q_n(t)$ is recovered via:

$$q_n(t) = x_n + \int_{t_n}^{t} \dot{q}_n(\tau; k_{n,1}, \dots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau.$$

# Collocation - how to implement it?

## How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \dots, k_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \dots, n_\mathrm{s}$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through $x_n$ (at $t_n$) and **state values** $x_{n,1}, \dots, x_{n,n_\mathrm{s}}$ at collocation points $t_{n,i},\ i = 1, \dots, n_\mathrm{s}$ (in Appendix).

- $q_n(t)$ is recovered via:

$$q_n(t) = x_n + \int_{t_n}^{t} \dot{q}_n(\tau; k_{n,1}, \dots, k_{n,n_\mathrm{s}})\mathrm{d}\tau.$$

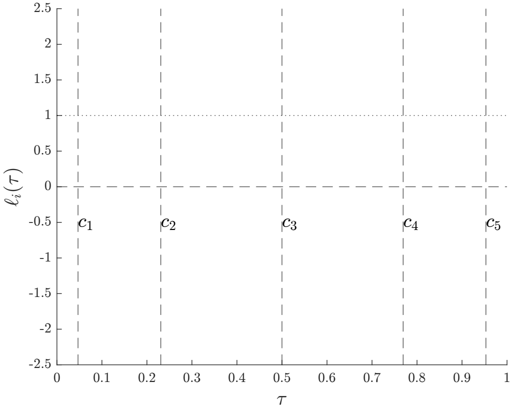
- with:

$$\begin{aligned}\dot{q}_n(t) &= \ell_1\left(\frac{t-t_n}{h}\right)k_{n,1} + \ell_2\left(\frac{t-t_n}{h}\right)k_{n,2} + \cdots + \ell_{n_\mathrm{s}}\left(\frac{t-t_n}{h}\right)k_{n,n_\mathrm{s}} \\ &= \sum_{i=1}^{n_\mathrm{s}} \ell_i\left(\frac{t-t_n}{h}\right)\underbrace{f(t_n + c_i, q_n(t_n + c_i h), u_0)}_{=k_{n,i}}\end{aligned}$$

# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i \neq j}^{n_s} \frac{\tau - c_j}{c_i - c_j}.$$



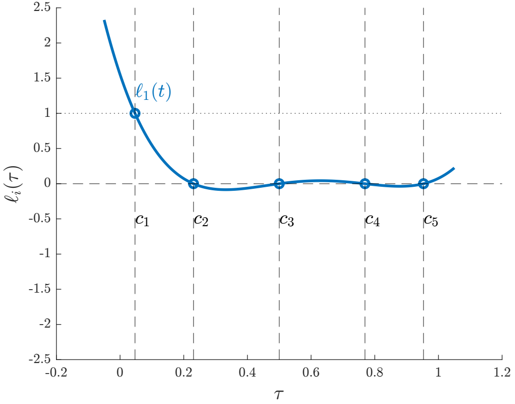
# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i\neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$



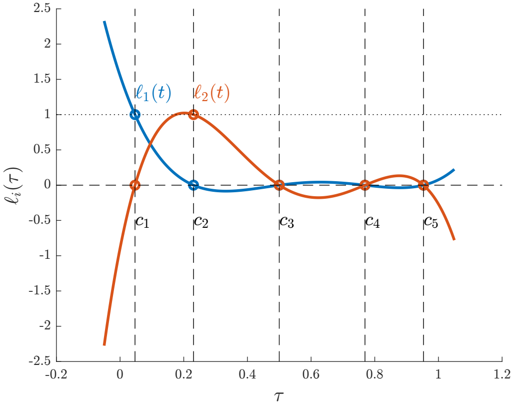
# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i\neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$



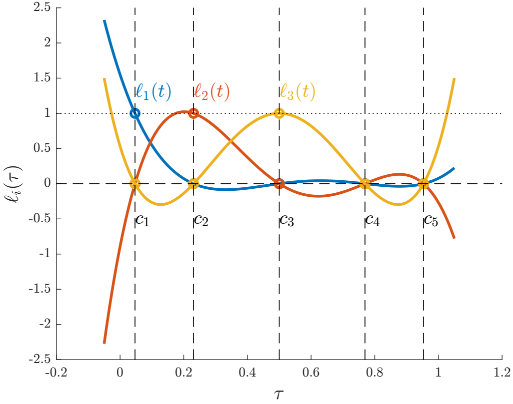
# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i \neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$



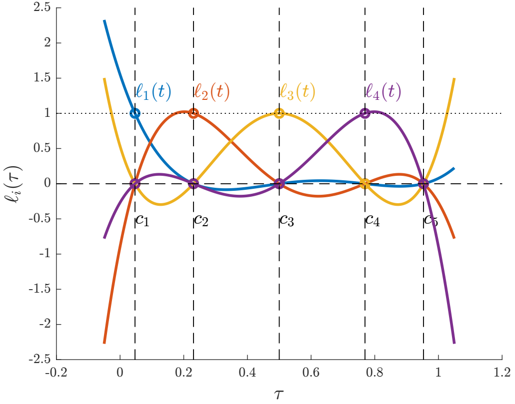
# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i\neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$



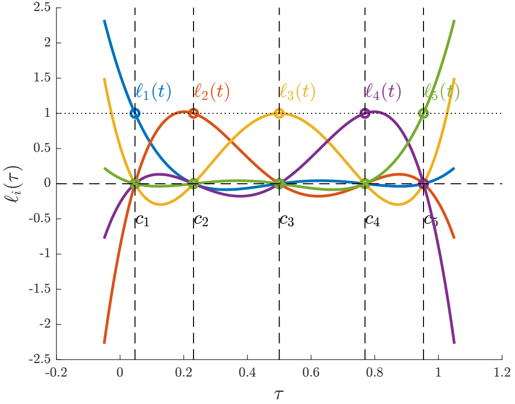
# The Lagrange polynomials $\ell_i(\tau)$

## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i\neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$



# The Lagrange polynomials $\ell_i(\tau)$

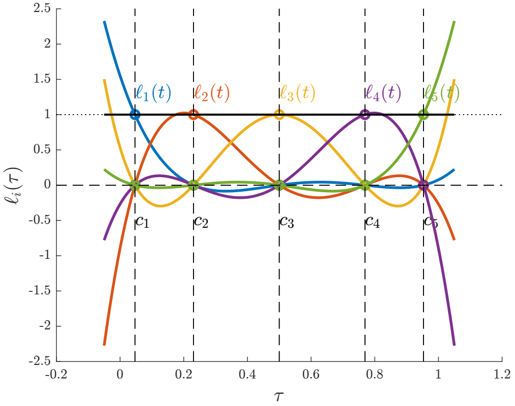
## Lagrange polynomial basis

$$\ell_i(\tau) = \prod_{j=1, i\neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}.$$

Properties:

$$\ell_i(c_j) = \begin{cases} 1 & \text{if} \quad j = i \\ 0 & \text{if} \quad j \neq i \end{cases}$$

$$\sum_{i=1}^{n_\mathrm{s}} \ell_i(t) = 1$$



# Collocation - how to implement it - continued

- Evaluate $q_n(t)$ at collocation points

$$q_n(t_n + c_i h) = x_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau$$

# Collocation - how to implement it - continued

- Evaluate $q_n(t)$ at collocation points

$$
\begin{aligned}
q_n(t_n + c_i h) &= x_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau \\
&= x_n + \int_{t_n}^{t_n + c_i h} \sum_{j=1}^{n_\mathrm{s}} \ell_j \Big( \frac{\tau - t_n}{h} \Big) k_{n,j} \mathrm{d}\tau
\end{aligned}
$$

# Collocation - how to implement it - continued

- Evaluate $q_n(t)$ at collocation points

$$
\begin{aligned}
q_n(t_n + c_i h) &= x_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau \\
&= x_n + \int_{t_n}^{t_n + c_i h} \sum_{j=1}^{n_\mathrm{s}} \ell_j \left( \frac{\tau - t_n}{h} \right) k_{n,j} \mathrm{d}\tau \\
&= x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} \underbrace{\int_0^{c_i} \ell_j(\sigma) \mathrm{d}\sigma}_{:= a_{i,j}}
\end{aligned}
$$

# Collocation - how to implement it - continued

- Evaluate $q_n(t)$ at collocation points

$$
\begin{aligned}
q_n(t_n + c_i h) &= x_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau \\
&= x_n + \int_{t_n}^{t_n + c_i h} \sum_{j=1}^{n_\mathrm{s}} \ell_j\left(\frac{\tau - t_n}{h}\right) k_{n,j} \mathrm{d}\tau \\
&= x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} \underbrace{\int_0^{c_i} \ell_j(\sigma) \mathrm{d}\sigma}_{:= a_{i,j}} \\
&= x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{i,j}
\end{aligned}
$$

# Collocation - how to implement it - continued

- Evaluate $q_n(t)$ at collocation points

$$
\begin{aligned}
q_n(t_n + c_i h) &= x_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \dots, k_{n,n_\mathrm{s}}) \mathrm{d}\tau \\
&= x_n + \int_{t_n}^{t_n + c_i h} \sum_{j=1}^{n_\mathrm{s}} \ell_j \left( \frac{\tau - t_n}{h} \right) k_{n,j} \mathrm{d}\tau \\
&= x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} \underbrace{\int_0^{c_i} \ell_j(\sigma) \mathrm{d}\sigma}_{:= a_{i,j}} \\
&= x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{i,j}
\end{aligned}
$$

Similarly $q_n(t)$ evaluated at $t_{n+1} = t_n + h$:

$$
q_n(t_n + h) = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} \underbrace{\int_0^1 \ell_i(\sigma) \mathrm{d}\sigma}_{:= b_i} = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} b_i
$$

# All collocation methods are implicit Runge-Kuta method

## Collocation equations

$$q_n(t_n) = x_n \qquad \text{(initial value)}$$

$$\dot{q}_n(t_n + c_i h) = f(t_n + c_i, q_n(t_n + c_i h), u_n), \quad i = 1, \dots, n_\mathrm{s} \qquad \text{(stage eqs.)}$$

$$x_{n+1} = q_n(t_{n+1}) \qquad \text{(next value)}$$

# All collocation methods are implicit Runge-Kuta method

## Collocation equations

$$
\begin{aligned}
&q_n(t_n) = x_n && \text{(initial value)}\\
&k_{n,i} = f(t_n + c_i h, q_n(t_n + c_i h), u_n), \quad i = 1, \ldots, n_{\mathrm{s}} && \text{(stage eqs.)}\\
&x_{n+1} = q_n(t_{n+1}) && \text{(next value)}
\end{aligned}
$$

# All collocation methods are implicit Runge-Kuta method

## Collocation equations

$$q_n(t_n) = x_n \qquad \text{(initial value)}$$

$$k_{n,i} = f(t_n + c_i h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{i,j}, u_n), \quad i = 1, \dots, n_\mathrm{s} \qquad \text{(stage eqs.)}$$

$$x_{n+1} = q_n(t_{n+1}) \qquad \text{(next value)}$$

# All collocation methods are implicit Runge-Kuta method

## Collocation equations

$$q_n(t_n) = x_n \qquad \text{(initial value)}$$

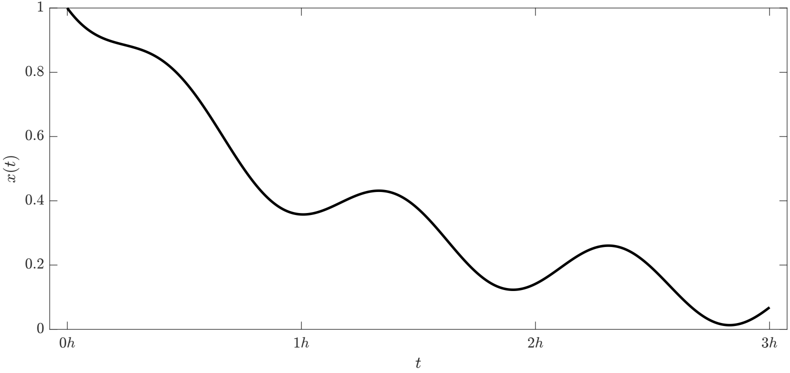
$$k_{n,i} = f(t_n + c_i h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{i,j}, u_n), \quad i = 1, \dots, n_\mathrm{s} \qquad \text{(stage eqs.)}$$

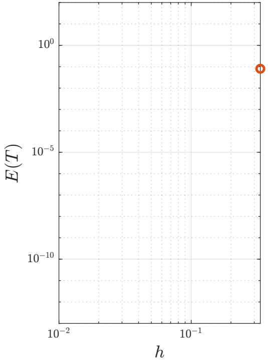
$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} b_i \qquad \text{(next value)}$$

- We arrived at the implicit RK equations in differential form
- Unknowns: $x_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} n_x}$
- $(n_\mathrm{s} + 1)n_x$ equations and $(n_\mathrm{s} + 1)n_x$ variables - solve via Newton's methods

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

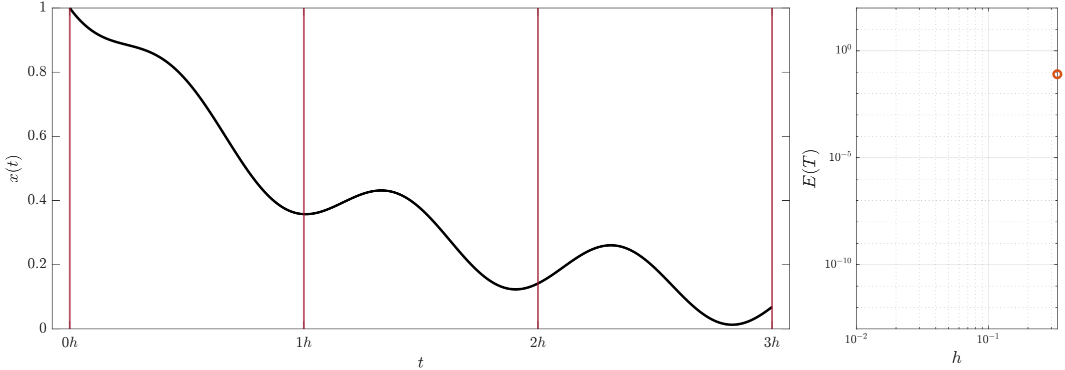



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

Visualization inspired by Leo Simpson's talk at the European control conference 2023

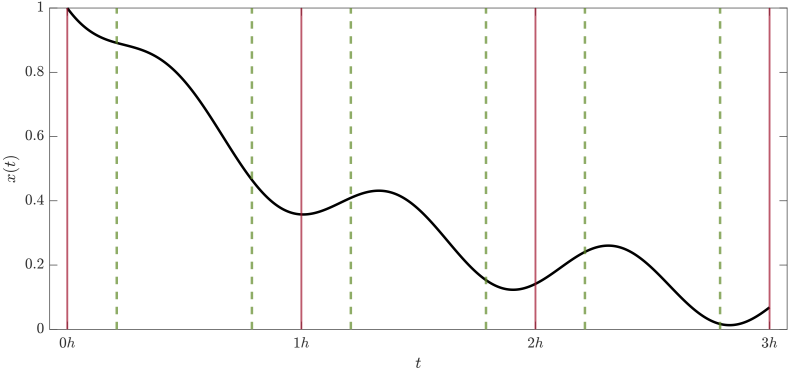
- Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.

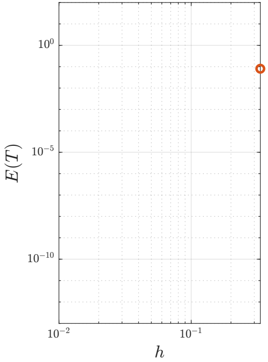



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

Visualization inspired by Leo Simpson's talk at the European control conference 2023

- Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



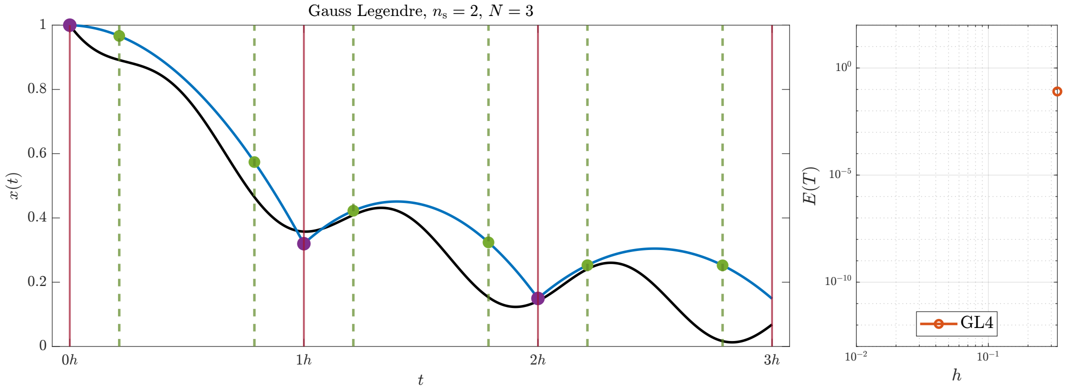




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

Visualization inspired by Leo Simpson's talk at the European control conference 2023

# Collocation - visualization

- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

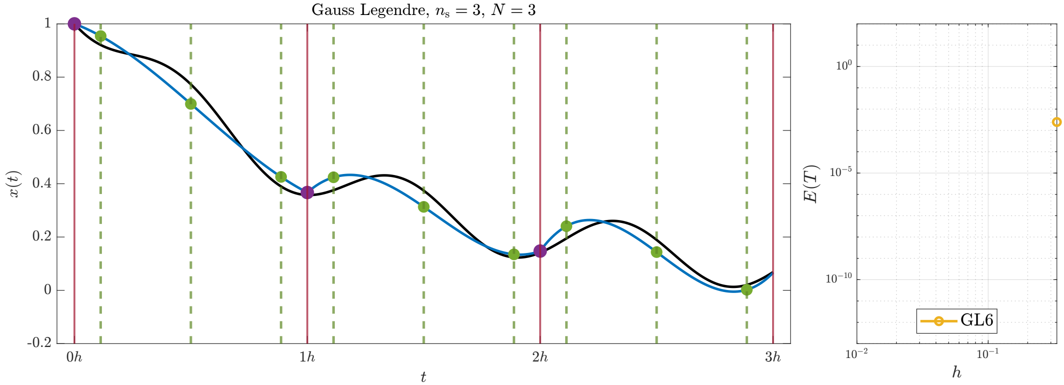



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

Visualization inspired by Leo Simpson's talk at the European control conference 2023

# Collocation - visualization

- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

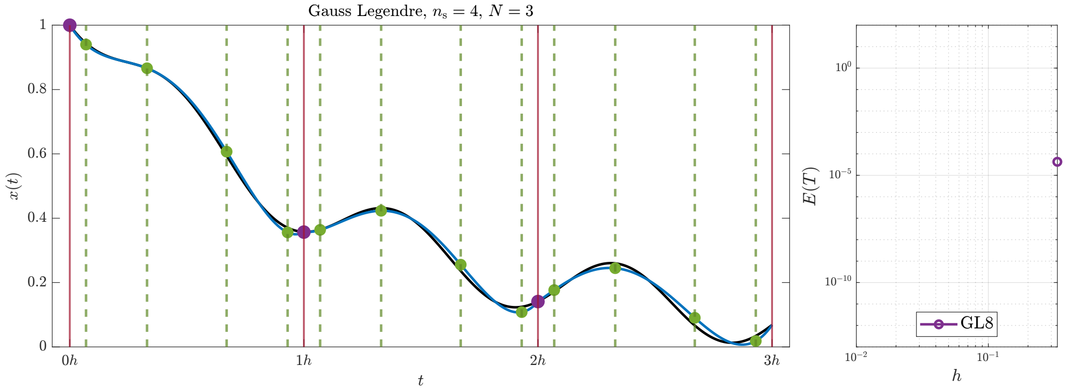



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

Visualization inspired by Leo Simpson's talk at the European control conference 2023

# Collocation - visualization

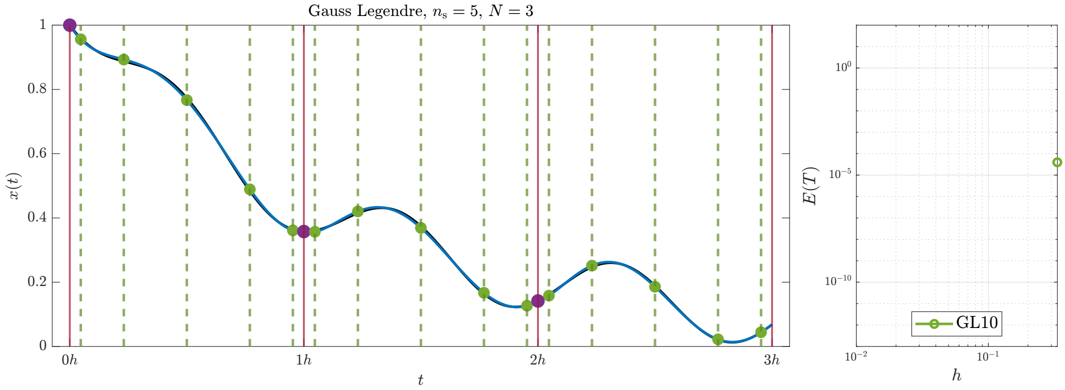
- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

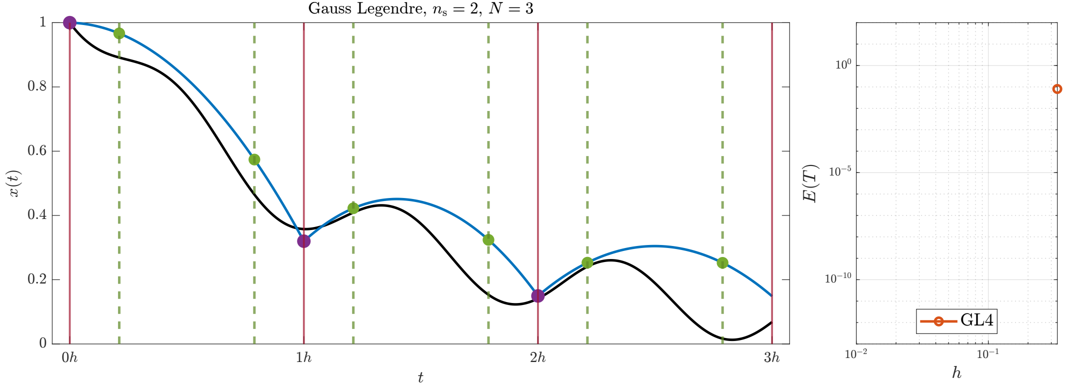
- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

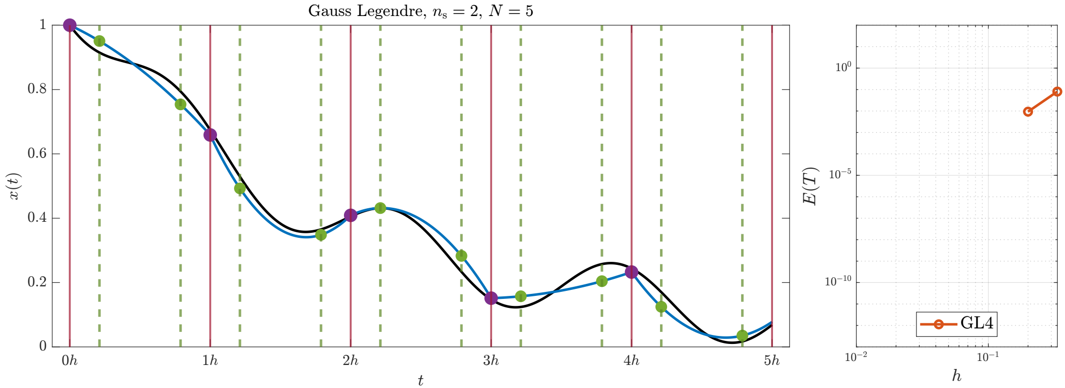
# Collocation - visualization

- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

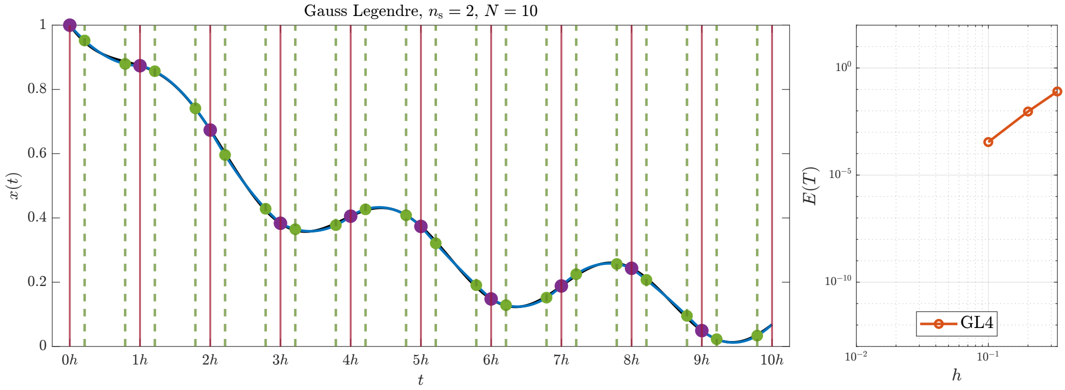
- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

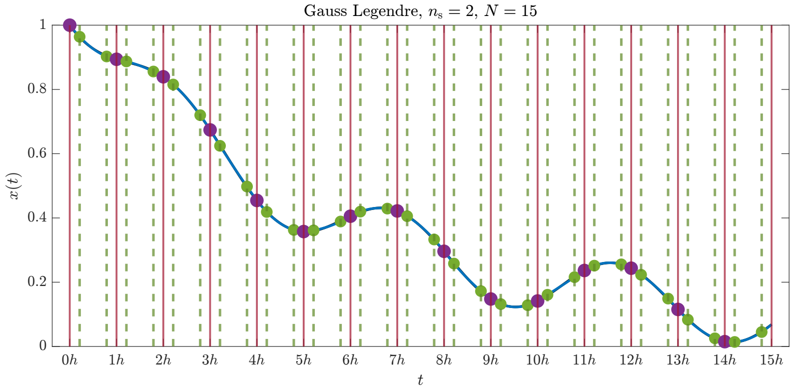
- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

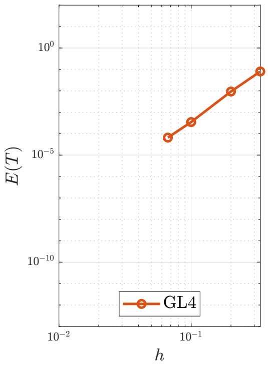



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

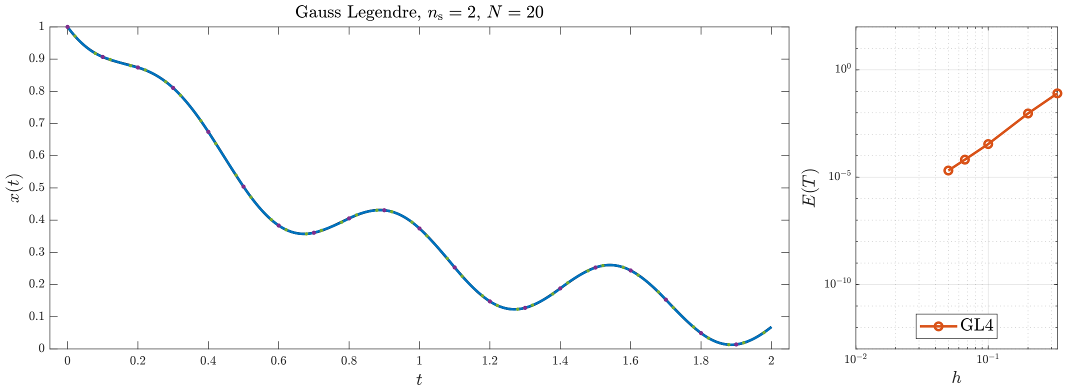



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

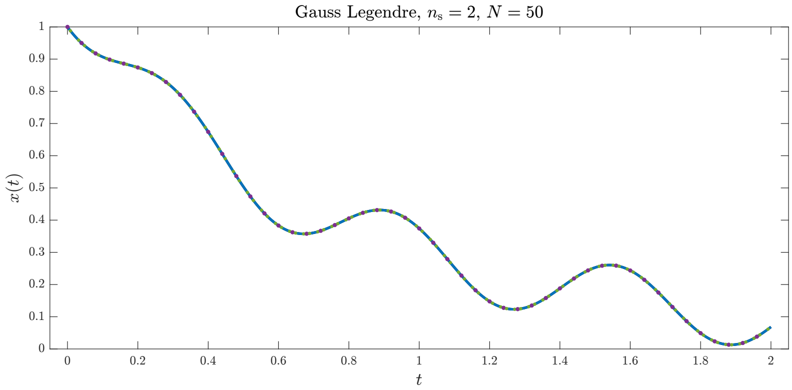
# Collocation - visualization

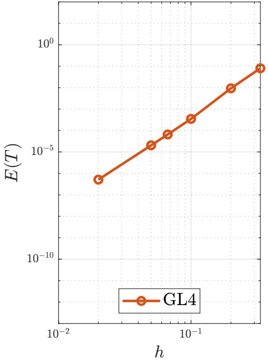
- Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

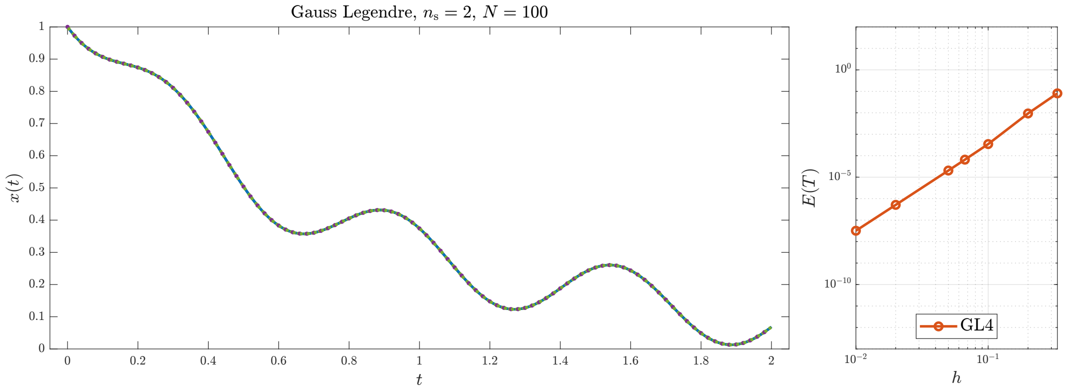
- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

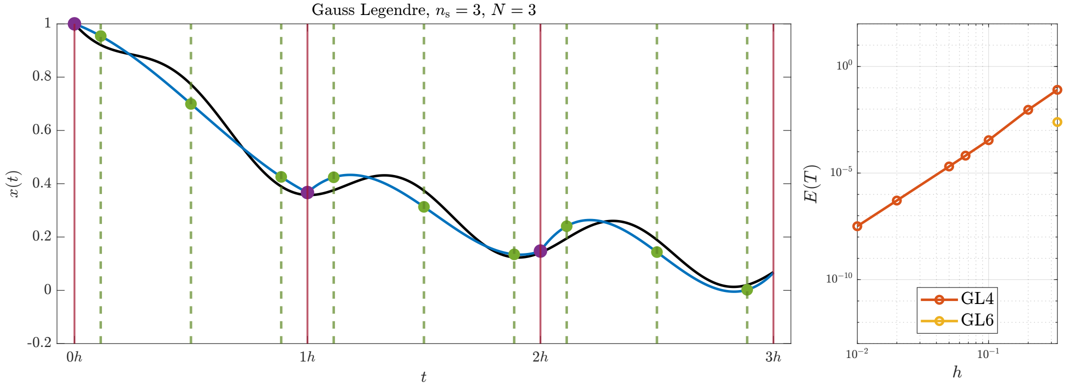
- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

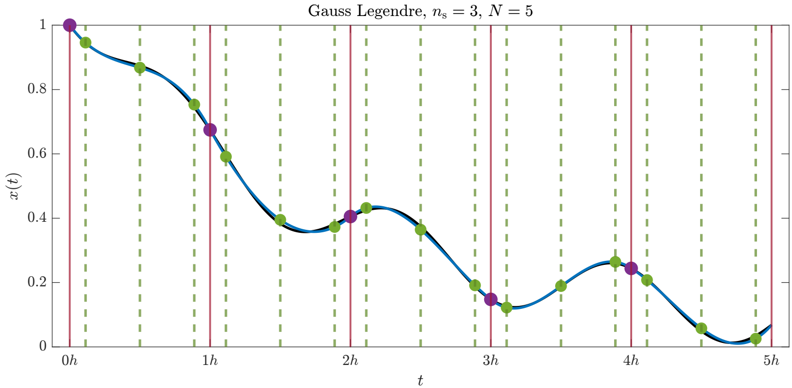
- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

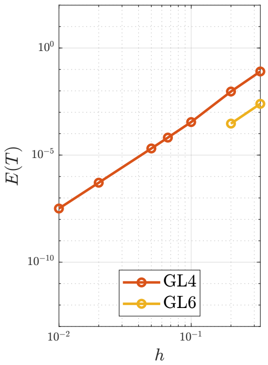



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

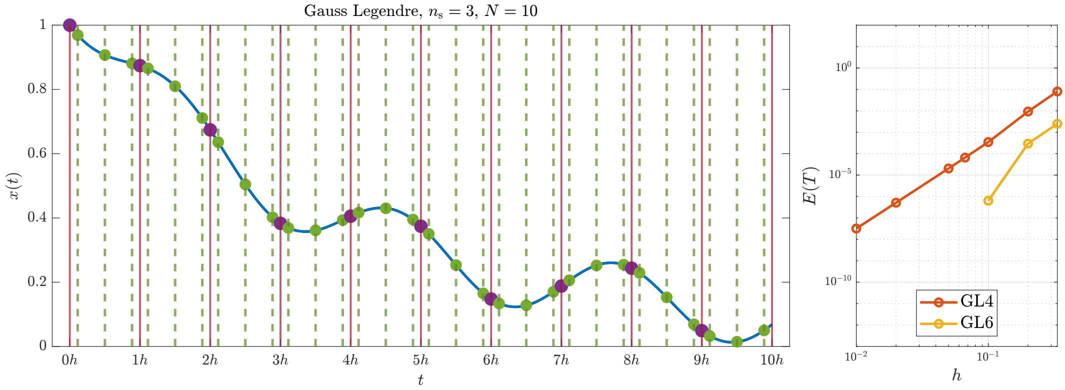



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

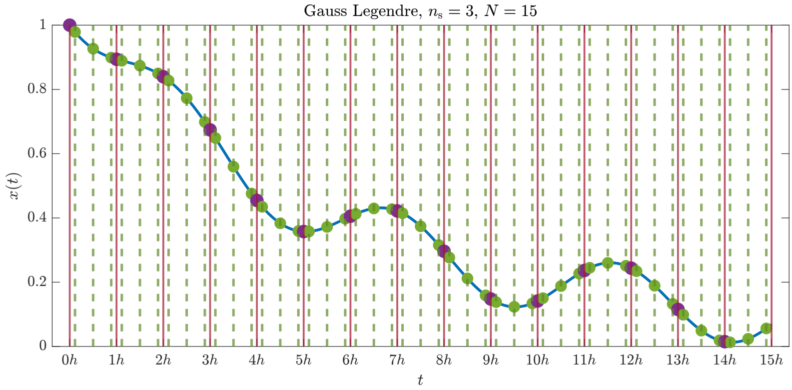
- Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.

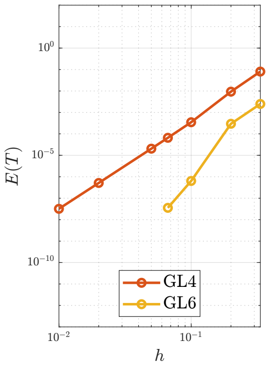



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

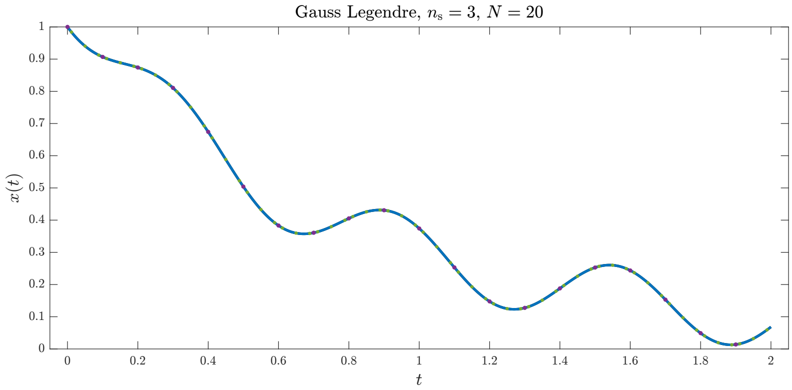


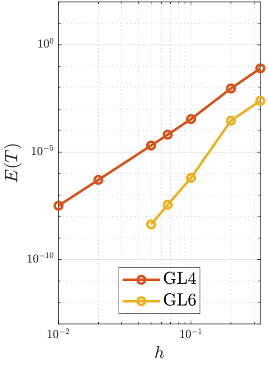




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

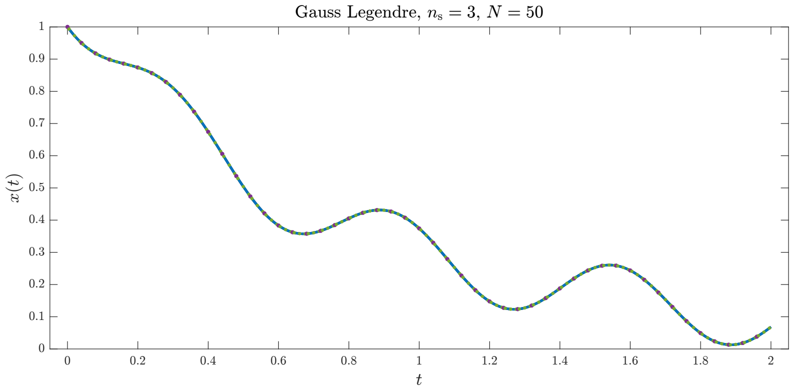
- Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.

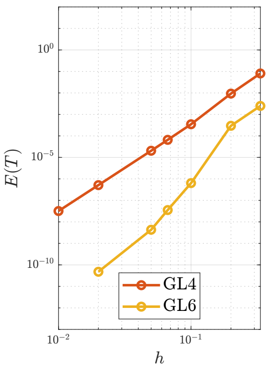



$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

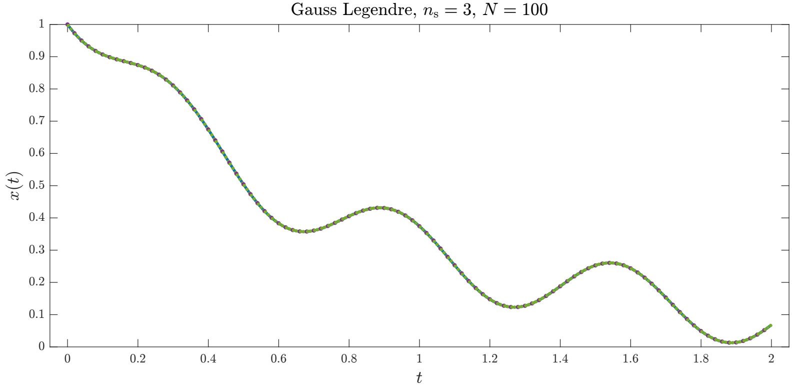


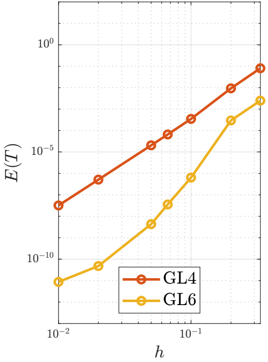




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

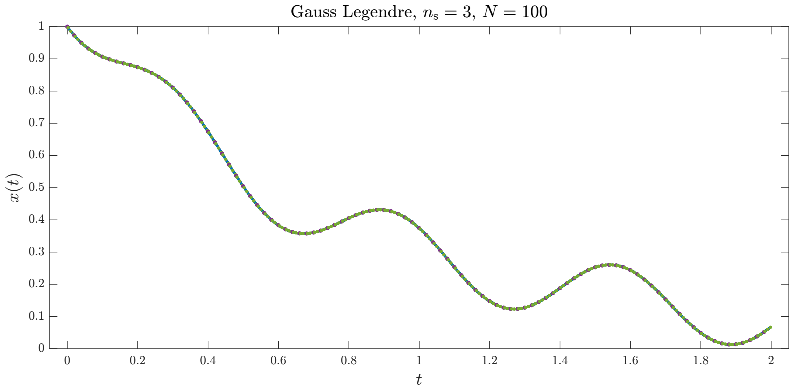


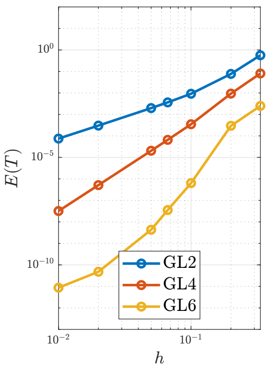




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \dots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.

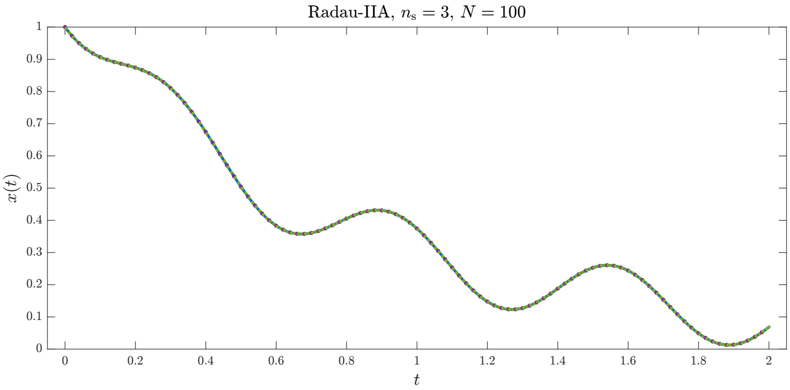


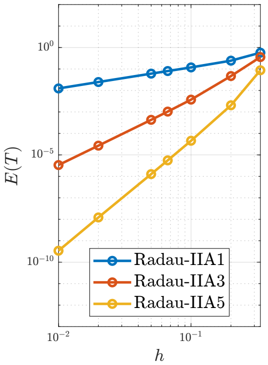




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Collocation - visualization

- Choice of points $c_1, \ldots, c_{n_\mathrm{s}}$ determines properties of method.
- Gauss-Legendre $p = 2n_\mathrm{s}$, Radau-IIA $p = 2n_\mathrm{s} - 1$ good for stiff systems, Lobatto family $p = 2n_\mathrm{s} - 2$.




$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

# Direct collocation in optimal control

Variables $x_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} n_x}$

## Collocation equations

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} b_i \qquad \text{(next value)}$$

$$k_{n,1} = f(t_n + c_1 h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{1,j}, u_n) \qquad \text{(stage Eq. 1)}$$

$$\vdots$$

$$k_{n,n_\mathrm{s}} = f(t_n + c_{n_\mathrm{s}} h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{n_\mathrm{s},j}, u_n), \qquad \text{(stage Eq. } n_\mathrm{s}\text{)}$$

Variables $x_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} n_x}$

## Collocation equations

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} b_i \qquad \text{(next value)}$$

$$0 = k_{n,1} - f(t_n + c_1 h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{1,j}, u_n) \qquad \text{(stage Eq. 1)}$$

$$\vdots$$

$$0 = k_{n,n_\mathrm{s}} - f(t_n + c_{n_\mathrm{s}} h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{n_\mathrm{s},j}, u_n), \qquad \text{(stage Eq. } n_\mathrm{s}\text{)}$$

# Direct collocation in optimal control

Variables $x_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \ldots, k_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} n_x}$

## Collocation equations

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} k_{n,i} b_i =: \phi_f(x_n, z_n, u_n) \qquad \text{(next value)}$$

$$0 = \begin{bmatrix} k_{n,1} - f(t_n + c_1 h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{1,j}, u_n) \\ \vdots \\ k_{n,n_\mathrm{s}} - f(t_n + c_{n_\mathrm{s}} h, x_n + h \sum_{j=1}^{n_\mathrm{s}} k_{n,j} a_{n_\mathrm{s},j}, u_n) \end{bmatrix} =: \phi_{\mathrm{int}}(x_n, z_n, u_n) \qquad \text{(stage Eqs.)}$$

# Direct collocation in optimal control

Variables $x_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_\mathrm{s}}) \in \mathbb{R}^{n_\mathrm{s} n_x}$

## Collocation equations

$$\begin{aligned} x_{n+1} &= \phi_f(x_n, z_n, u_n) && \text{(next value)} \\ 0 &= \phi_\mathrm{int}(x_n, z_n, u_n) && \text{(stage Eqs.)} \end{aligned}$$

- Use to discretize optimal control problem

# Continious time OCP into Nonlinear Programs (NLP)

## Continuous time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

- Direct methods: first discretize, then optimize

# Continious time OCP into Nonlinear Programs (NLP)

**Continuous time OCP**

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

- Direct methods: first discretize, then optimize

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.

# Continious time OCP into Nonlinear Programs (NLP)

**Continuous time OCP**

$$\min_{x(\cdot),u(\cdot)} \quad \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T))$$

$$\begin{aligned} \text{s.t.} \quad x(0) &= \bar{x}_0 \\ \dot{x}(t) &= f(x(t),u(t)) \\ 0 &\geq h(x(t),u(t)),\ t \in [0,T] \\ 0 &\geq r(x(T)) \end{aligned}$$

- Direct methods: first discretize, then optimize

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics via collocation

$$L_{\mathrm{d}}(x_n, u_n) = \int_{t_n}^{t_{n+1}} L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t.$$

Replace $\dot{x} = f(x,u)$ by

$$\begin{aligned} x_{n+1} &= \phi_f(x_n, z_n, u_n), \\ 0 &= \phi_{\mathrm{int}}(x_n, z_n, u_n). \end{aligned}$$

# Continious time OCP into Nonlinear Programs (NLP)

**Continuous time OCP**

$$\min_{x(\cdot),u(\cdot)} \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T))$$

$$\text{s.t.} \quad x(0) = \bar{x}_0$$

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

$$0 \geq h(x(t),u(t)), \; t \in [0,T]$$

$$0 \geq r(x(T))$$

- Direct methods: first discretize, then optimize

1. Parametrize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics via collocation

$$L_\mathrm{d}(x_n,u_n) = \int_{t_n}^{t_{n+1}} L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t.$$

   Replace $\dot{x} = f(x,u)$ by

$$x_{n+1} = \phi_f(x_n,z_n,u_n),$$
$$0 = \phi_\mathrm{int}(x_n,z_n,u_n).$$

3. Relax path constraints, e.g., evaluate only at $t = t_n$

$$0 \geq h(x_n,u_n), \; n = 0,\ldots N-1.$$

# Continious time OCP into Nonlinear Programs (NLP)

### Continuous time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

- Direct methods: first discretize, then optimize

### Discrete time OCP (an NLP)

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \quad & \sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k,u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \phi_f(x_n,z_n,u_n) \\
& 0 = \phi_{\mathrm{int}}(x_n,z_n,u_n) \\
& 0 \geq h(x_n,u_n),\ n = 0,\dots,N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $\mathbf{x} = (x_0,\dots,x_N)$, $\mathbf{z} = (z_0,\dots,z_N)$ and $\mathbf{u} = (u_0,\dots,u_{N-1})$.

# Direct optimal control methods solve Nonlinear Programs (NLP)

**Discrete time OCP (an NLP)**

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \quad & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \phi_f(x_n, z_n, u_n) \\
& 0 = \phi_{\mathrm{int}}(x_n, z_n, u_n) \\
& 0 \geq h(x_n, u_n), \; n = 0, \dots, N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $w = (\mathbf{x}, \mathbf{z}, \mathbf{u})$

# Direct optimal control methods solve Nonlinear Programs (NLP)

**Discrete time OCP (an NLP)**

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \quad & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k, u_k) + E(x_N) \\
\text{s.t.} \quad x_0 &= \bar{x}_0 \\
x_{n+1} &= \phi_f(x_n, z_n, u_n) \\
0 &= \phi_{\mathrm{int}}(x_n, z_n, u_n) \\
0 &\geq h(x_n, u_n), \;\; n = 0, \ldots, N-1 \\
0 &\geq r(x_N)
\end{aligned}
$$

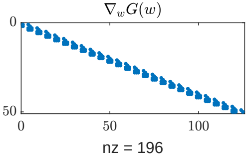
Variables $w = (\mathbf{x}, \mathbf{z}, \mathbf{u})$

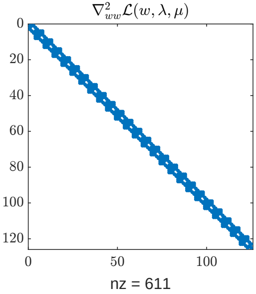
**Nonlinear Program (NLP)**

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

Obtain large and sparse NLP

# Direct optimal control methods solve Nonlinear Programs (NLP)




Variables $w = (\mathbf{x}, \mathbf{z}, \mathbf{u})$

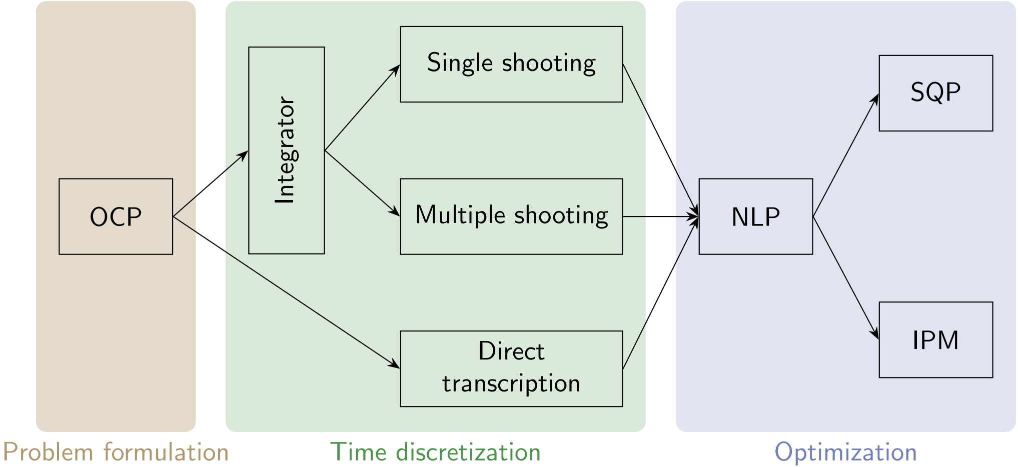
**Nonlinear Program (NLP)**

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

Obtain large and sparse NLP

# Work flow in smooth direct optimal control

First discretize, then optimize.




OCP = Optimal Control Problem

NLP = Nonlinear Program

SQP = Sequential Quadratic Programming

IPM = Interior-Point Method

Figure inspired by Lecture 1, Numerical Methods for Optimal Control: Introduction, 2022, by Mario Zanon and Sébastien Gros.

# Direct collocation vs direct multiple shooting

## Direct collocation NLP

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \ & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k,u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \phi_f(x_n,z_n,u_n) \\
& 0 = \phi_{\mathrm{int}}(x_n,z_n,u_n) \\
& 0 \geq h(x_n,u_n), \ n = 0,\dots,N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

- Variables $w = (\mathbf{x},\mathbf{z},\mathbf{u})$
- Only fixed number of integration steps
- Internal computations of integrator done by optimizer
- More variables, but sparser

## Multiple shooting NLP

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \ & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k,u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \psi_f(x_n,u_n) \\
& 0 \geq h(x_n,u_n), \ n = 0,\dots,N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

- Variables $w = (\mathbf{x},\mathbf{u})$
- Can use adaptive integrators
- Internal computations of integrator hidden from optimizer
- Less variables

# Summary

- Numerical simulation methods used to solve ODEs approximately.
- Integration accuracy order and stability play key roles.
- All collocation methods are IRK methods, the converse is not true.
- Choice of discretization method has huge influence on efficacy and reliability of NLP solution.
- Best choice is problem dependent and often requires lot of care.
- Many good software packages exist: `CasADi`, `pyomo.DAE`, `acados`, `MUSCOD-II`, `ACADO`, `ForcesPRO`, `IPOPT`, ... (the list is far from complete)

# Reading recommendations

Direct optimal control:

- James B. Rawlings, David Q. Mayne, and Moritz Diehl. Model predictive control: theory, computation, and design. Vol. 2. Madison, WI: Nob Hill Publishing, 2017. - Chapter 8. Online: `https://sites.engineering.ucsb.edu/~jbraw/mpc/`
- Biegler, Lorenz T. Nonlinear programming: concepts, algorithms, and applications to chemical processes. Society for Industrial and Applied Mathematics, 2010.
- Moritz Diehl, Sébastien Gros. "Numerical optimal control (Draft)," Lecture notes, 2014. Online: `https://www.syscop.de/files/2024ws/NOC/book-NOCSE.pdf`

Optimal control, MPC vs reinforcement learning:

- Recht, B. (2019). A tour of reinforcement learning: The view from continuous control. Annual Review of Control, Robotics, and Autonomous Systems, 2(1), 253-279.
- Bertsekas, D. P. (2024). Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming. arXiv preprint arXiv:2406.00592.

# Free online video lectures

Numerical optimal control video lectures by Moritz Diehl (**highly recommended!**):

- Videos: `https://www.syscop.de/teaching/ss2020/numerical-optimal-control-online`
- Lecture notes: `https://www.syscop.de/files/2024ws/NOC/book-NOCSE.pdf`

Lecture notes/slides by Mario Zanon Sébastien Gros

- `https://mariozanon.wordpress.com/teaching/numerical-methods-for-optimal-control/`

Optimal control software:

- `CasADi` - general purpose modeling and optimization - `https://web.casadi.org/get`
- acados - fast embedded model predictive control - `https://github.com/acados/acados`
- `https://www.syscop.de/research/software`

# References used for this lecture


- Moritz Diehl, Sébastien Gros. "Numerical optimal control (Draft)," Lecture notes, 2024.
- James B. Rawlings, David Q. Mayne, and Moritz Diehl. Model predictive control: theory, computation, and design. Vol. 2. Madison, WI: Nob Hill Publishing, 2017.
- Gerhard Wanner, Ernst Hairer. "Solving ordinary differential equations II." Vol. 375. New York: Springer Berlin Heidelberg, 1996.

# Direct collocation via state values

To discretize an optimal control problem with *direct collocation* we replace the continuous-time dynamics

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

by the discrete-time collocation equations.

- split the control horizon $[0, T]$ into $N$ control intervals with a uniform time discretization grid $t_n = nh$, $n = 0, \dots, N$, $h = \frac{T}{N}$
- state values are $x_n = x(t_n)$
- control discretization: $u(t) = u_n, t \in [t_n, t_{n+1}]$, $n = 1, \dots, N$.
- on every control interval the state trajectory is approximated by polynomials $q_n(t)$, $n = 1, \dots, N$.

Next, on each control interval $[t_n, t_{n+1}]$, we compute the coefficients of these polynomials to ensure that the ODE is exactly satisfied at the *collocation points* $t_{n,i} = t_n + hc_i, i = 1, \dots, n_\mathrm{s}$, where, $n_\mathrm{s}$ is the number of stages.

# Direct collocation via state values

- choice of $0 = c_0 < c_1 < \ldots < c_{n_\mathrm{s}} \leq 1$ determines the accuracy and stability properties of the resulting method
- popular choices for $c_i$ are the Radau IIA or Gauss-Legendre points.
- Previously treated: interpolating polynomial $\dot{q}_n(t)$ through the state derivatives $k_{n,1}, \ldots, k_{n,n_\mathrm{s}}$.
- Here, finding the interpolating polynomial $q_n(t)$ through the initial value $x_n$ and *state values* $x_{n,1}, \ldots, x_{n,n_\mathrm{s}}$ at the stage points.
- We use of the Lagrangian polynomial basis. Using these time points, we define a basis for our polynomials:

$$\ell_i(\tau) = \prod_{j=0,\, i \neq j}^{n_\mathrm{s}} \frac{\tau - c_j}{c_i - c_j}, \qquad i = 0, \ldots, n_\mathrm{s}. \tag{2}$$

- Remark: in contrast to using state derivatives $k_{n,i}$, the counter starts from $i = 0$, as we include the point $c_0 = 0$, since we interpolate through $x_n$.

# Direct collocation via state values

We approximate the state trajectory on $[t_n, t_{n+1}]$ by a linear combination of the basis functions:

$$q_n(t) = \sum_{j=0}^{n_\mathrm{s}} \ell_j \left( \frac{t - t_n}{h} \right) x_{n,j}. \tag{3}$$

By differentiation, we obtain an approximation of the time derivative at each collocation point:

$$\dot{q}_n(t_{n,i}) = \frac{1}{h} \sum_{j=0}^{n_\mathrm{s}} \dot{\ell}_j(c_i)\, x_{n,j} := \frac{1}{h} \sum_{j=0}^{n_\mathrm{s}} C_{j,i}\, x_{n,j}, \quad i = 0, \dots, n_\mathrm{s}. \tag{4}$$

The collocation equations must satisfy the ODE at every collocation point $t_{n,i}$:

$$\dot{q}_n(t_{n,i}) = f(q_n(t_{n,i}), u_n), \quad i = 1, \dots n_s$$

That is:

$$\frac{1}{h} \sum_{j=0}^{n_s} C_{j,i}\, x_{n,j} = f(x_{n,i}, u_n), \quad i = 1, \dots n_s. \tag{5}$$

# Collocation: next step and cost integral approximation

The expression for the state at the end of an interval reads as:

$$x_{n+1} = \sum_{i=0}^{n_\mathrm{s}} \ell_i(1)\, x_{n,i} := \sum_{i=0}^{n_\mathrm{s}} D_i\, x_{n,i} \tag{6}$$

Moreover, using the obtained approximation $q_n(t)$ we can integrate the *stage cost*

$$\int_0^T L(x(t), u(t))\mathrm{d}t,$$

over every control interval and obtain a formula for *quadratures*:

$$\int_{t_n}^{t_{n+1}} \sum_{j=0}^{n_\mathrm{s}} \ell_j\left(\frac{t - t_n}{h}\right) L(x_{n,j}, u_n)\,\mathrm{d}t = h \sum_{j=0}^{n_\mathrm{s}} \int_0^1 \ell_j(t)\,\mathrm{d}t\, L(x_{n,j}, u_n) := h \sum_{j=0}^{n_\mathrm{s}} B_j\, L(x_{n,j}, u_n). \tag{7}$$

# Runge-Kutta method definition

Unknowns are states at stage points, cannot treat case of $c_1 = 0$

## Definition (Runge-Kutta method in integral form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \ldots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$x_{n,i} = x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} f(t_{n,i}, x_{n,j}, u_n), \; i = 1, \ldots, n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i f(t_{n,i}, x_{n,i}, u_n),$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in *integral form.*

| **Time grid** | **Butcher tableau** | **Data** | **Variables** |
|---|---|---|---|
| $h$, $t_n$, $t_{n,i}$ | $a_{i,j}$, $b_i$, $c_i$ | $x_n$, $u_n$, $f(\cdot)$ | $x_{n+1}$, $x_{n,i}$ |
| $i = 1, \ldots, n_\mathrm{s}$ | $i, j = 1, \ldots, n_\mathrm{s}$ | | $i = 1, \ldots, n_\mathrm{s}$ |

# Runge-Kutta method definition

Unknowns are states at stage points, cannot treat case of $c_1 = 0$

## Definition (Runge-Kutta method in integral form)

Let $n_\mathrm{s}$ be the number of stages. Given the matrix $A \in \mathbb{R}^{n_\mathrm{s} \times n_\mathrm{s}}$ with the entries $a_{i,j}$ for $i, j = 1, \dots, n_\mathrm{s}$, and the vectors $b, c \in \mathbb{R}^{n_\mathrm{s}}$. Let $t_{n,i} = t_n + c_i h$. The system of equations:

$$x_{n,i} = x_n + h \sum_{j=1}^{n_\mathrm{s}} a_{i,j} f(t_{n,i}, x_{n,j}, u_n), \; i = 1, \dots, n_\mathrm{s}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{n_\mathrm{s}} b_i f(t_{n,i}, x_{n,i}, u_n),$$

is called a $n_\mathrm{s}$-stage Runge-Kutta (RK) method in *integral form.*