

Model Predictive Control and Reinforcement Learning

– Lecture 01: Dynamic Systems and Simulation –

Armin Nurkanović

University of Freiburg

Slides jointly developed with Moritz Diehl, Andrea Ghezzi

School on Model Predictive Control and Reinforcement Learning
Freiburg, 7-16 October 2026

universität freiburg



NTNU

Norwegian University of
Science and Technology

Dynamic System Models

From Continuous to Discrete Time

Implicit Runge-Kutta Methods

Input Output Models

Stochastic Models



- ▶ optimal control = optimization of dynamic systems
- ▶ each optimal control problem (OCP) is characterized by three ingredients:
 - ▶ dynamic system model
 - ▶ constraints
 - ▶ objective function, i.e., cost or reward



- ▶ optimal control = optimization of dynamic systems
- ▶ each optimal control problem (OCP) is characterized by three ingredients:
 - ▶ **dynamic system model** (focus of this lecture)
 - ▶ constraints
 - ▶ objective function, i.e., cost or reward



- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)



- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
 - ▶ control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)



- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
 - ▶ control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
 - ▶ random **disturbance** ϵ from some disturbance space $\mathbb{D}$



- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
 - ▶ control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
 - ▶ random **disturbance** ϵ from some disturbance space $\mathbb{D}$
- ▶ examples:

- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
 - ▶ control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
 - ▶ random **disturbance** ϵ from some disturbance space $\mathbb{D}$
- ▶ examples:
 - ▶ **stochastic discrete time system**, for $k = 0, 1, 2, \dots$

$$\boxed{s_{k+1} = f(s_k, a_k, \epsilon_k)} \quad \text{with "evolution function" } f : \mathbb{S} \times \mathbb{A} \times \mathbb{D} \rightarrow \mathbb{S}$$

- ▶ system model describes evolution of system as function of
 - ▶ system **state** s from state space $\mathbb{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
 - ▶ control **action** a from action space $\mathbb{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
 - ▶ random **disturbance** ϵ from some disturbance space $\mathbb{D}$

- ▶ examples:

- ▶ **stochastic discrete time system**, for $k = 0, 1, 2, \dots$

$$\boxed{s_{k+1} = f(s_k, a_k, \epsilon_k)} \quad \text{with "evolution function" } f : \mathbb{S} \times \mathbb{A} \times \mathbb{D} \rightarrow \mathbb{S}$$

- ▶ deterministic continuous time **ordinary differential equation (ODE)**, for $t \in [0, \infty)$

$$\boxed{\frac{ds}{dt}(t) = f_c(s(t), a(t))} \quad \text{with "right hand side function" } f_c : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}^{n_s}$$



- ▶ denote $\frac{ds}{dt}(t)$ by $\dot{s}(t)$
- ▶ drop time argument, abbreviate $\dot{s}(t) = f_c(s(t), a(t))$ by

$$\dot{s} = f_c(s, a)$$

- ▶ In this course, we use the RL notation: s for state and a for control action
- ▶ But in control engineering, one uses: x for state and u for control action, i.e.,

$$\dot{x} = f_c(x, u)$$

Mass m with spring constant k and friction coefficient β :

$$\begin{aligned}\dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= -\frac{k}{m}(x_2(t) - u(t)) - \frac{\beta}{m}x_1(t)\end{aligned}$$

- state $x(t) \in \mathbb{R}^2$
- position of mass $x_1(t)$ $\longleftarrow$ measured
- velocity of mass $x_2(t)$
- control action: spring position $u(t) \in \mathbb{R}$ $\longleftarrow$ manipulated

Can summarize as $\dot{x} = f_c(x, u)$ with

$$f_c(x, u) = \begin{bmatrix} x_2 \\ -\frac{k}{m}(x_2 - u) - \frac{\beta}{m}x_1 \end{bmatrix}$$

Mass m with spring constant k and friction coefficient β :

$$\begin{aligned}\dot{s}_1(t) &= s_2(t) \\ \dot{s}_2(t) &= -\frac{k}{m}(s_2(t) - a(t)) - \frac{\beta}{m}s_2(t)\end{aligned}$$

- state $s(t) \in \mathbb{R}^2$
- position of mass $s_1(t)$ $\longleftarrow$ measured
- velocity of mass $s_2(t)$
- control action: spring position $a(t) \in \mathbb{R}$ $\longleftarrow$ manipulated

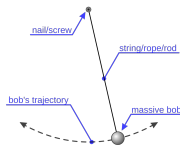
Can summarize as $\dot{s} = f_c(s, a)$ with

$$f_c(s, a) = \begin{bmatrix} s_2 \\ -\frac{k}{m}(s_2 - a) - \frac{\beta}{m}s_2 \end{bmatrix}$$

Some ODE Examples – what are their state vectors?

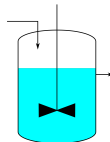


- ▶ Pendulum
- ▶ Hot plate with pot
- ▶ Continuously Stirred Tank Reactors (CSTR)
- ▶ Robot arms
- ▶ Mobile robots
- ▶ Race cars
- ▶ Airplanes in free flight



Pendulum

Commons: Chris-martin, Public domain



CSTR

Commons: Razorbliss, CC0



Mobile robots

Commons: Jiuguang Wang, CC BY-SA 3.0



Hot plate with pot

Commons: Carnildo, Public domain



Robot arms

Commons: Mixabest, CC BY-SA 3.0



Race cars

Commons: edvvc, CC BY 2.0

Dynamic System Models

From Continuous to Discrete Time

Implicit Runge-Kutta Methods

Input Output Models

Stochastic Models



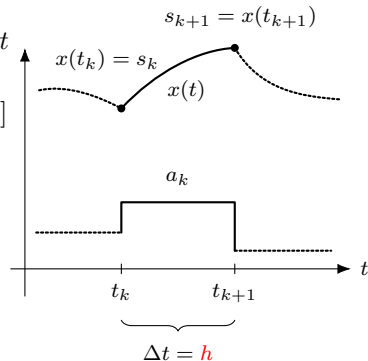
Transform continuous $\dot{s}(t) = f_c(s(t), a(t))$ into discrete time $s_{k+1} = f(s_k, a_k)$ as follows:

Transform continuous $\dot{s}(t) = f_c(s(t), a(t))$ into discrete time $s_{k+1} = f(s_k, a_k)$ as follows:

1. define $s_k := s(t_k)$ on **equidistant time grid** $t_k = k \Delta t$ with sampling time $\Delta t = h$
2. use **zero order hold** control $a(t) = a_k$ on $t \in [t_k, t_{k+1}]$
3. use **numerical simulation** to compute ODE solution $x(t) \equiv x(t; s_k, a_k)$ satisfying

$$\begin{aligned}x(t_k) &= s_k \\ \dot{x}(t) &= f_c(x(t), a_k) \quad \text{for } t \in [t_k, t_{k+1}]\end{aligned}$$

4. define $f(s_k, a_k) := x(t_{k+1}; s_k, a_k)$
(discrete-time dynamics)



From Continuous to Discrete Time via Direct Multiple Shooting



Transform continuous $\dot{s}(t) = f_c(s(t), a(t))$ into discrete time $s_{k+1} = f(s_k, a_k)$ as follows:

$$s_{k+1} = x(t_{k+1})$$

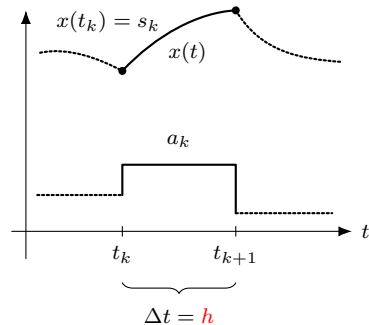
Exact ODE solution

$$x(0) = s,$$

$$\dot{x}(t) = f_c(x(t), a),$$

$$\text{for } t \in [0, h]$$

$$f(s, a) := x(h)$$



How to simulate ODE numerically?

- ▶ simplest (but not recommended) implementation is a single step of an Euler integrator:

$$f(s, a) := s + h f_c(s, a)$$

- ▶ more accurate are N steps of an Euler integrator:

```
 $x_0 := s$   
for  $i = 0$  to  $N - 1$  do  
     $x_{i+1} := x_i + (h/N) f_c(x_i, a)$   
 $f(s, a) := x_N$ 
```

- ▶ more efficient are higher order **Runge Kutta (RK)** methods, e.g. a single RK4 step:

```
 $v_1 := f_c(s, a)$   
 $v_2 := f_c(s + (h/2) v_1, a)$   
 $v_3 := f_c(s + (h/2) v_2, a)$   
 $v_4 := f_c(s + h v_3, a)$   
 $f(s, a) := s + (h/6) (v_1 + 2v_2 + 2v_3 + v_4)$ 
```

Euler vs 4th Order Runge Kutta Method (RK4) for Test Problem



Aim: solve $\dot{s} = s + a$ for $\Delta t = 1, s = 1, a = 0$. Exact solution is $f(s, a) = e = 2.718$.

► Four Euler steps give

$$x_0 := 1$$

$$x_1 := x_0 + 1/4 x_0 \quad [= (1 + 1/4)x_0]$$

$$x_2 := (1 + 1/4)x_1$$

$$x_3 := (1 + 1/4)x_2$$

$$x_4 := (1 + 1/4)x_3$$

$$f_{\text{Euler}}(s, a) := x_4 \quad [= (1 + 1/4)^4 = 2.441], \text{ error} > 10\%$$

► One RK4 step gives

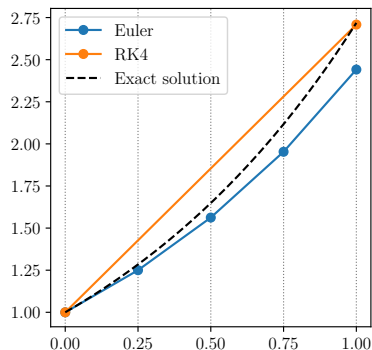
$$v_1 := 1$$

$$v_2 := 1 + 1/2 v_1 \quad [= 6/4]$$

$$v_3 := 1 + (1/2)v_2 \quad [= 7/4]$$

$$v_4 := 1 + v_3 \quad [= 11/4]$$

$$f_{\text{RK4}}(s, a) := 1 + (1/6) (v_1 + 2v_2 + 2v_3 + v_4) \quad [= 2.708]$$



Euler vs 4th Order Runge Kutta Method (RK4) for Test Problem



Aim: solve $\dot{s} = s + a$ for $\Delta t = 1, s = 1, a = 0$. Exact solution is $f(s, a) = e = 2.718$.

► Four Euler steps give

$$x_0 := 1$$

$$x_1 := x_0 + 1/4 x_0 \quad [= (1 + 1/4)x_0]$$

$$x_2 := (1 + 1/4)x_1$$

$$x_3 := (1 + 1/4)x_2$$

$$x_4 := (1 + 1/4)x_3$$

$$f_{\text{Euler}}(s, a) := x_4 \quad [= (1 + 1/4)^4 = 2.441], \text{ error} > 10\%$$

► One RK4 step gives

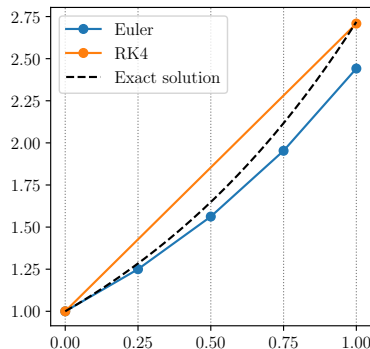
$$v_1 := 1$$

$$v_2 := 1 + 1/2 v_1 \quad [= 6/4]$$

$$v_3 := 1 + (1/2)v_2 \quad [= 7/4]$$

$$v_4 := 1 + v_3 \quad [= 11/4]$$

$$f_{\text{RK4}}(s, a) := 1 + (1/6) (v_1 + 2v_2 + 2v_3 + v_4) \quad [= 2.708]$$



Integration order

RK4 is 27x more accurate than Euler for same number $M = 4$ of function evaluations!



From now on, throughout the course, we exclusively focus on discrete time models

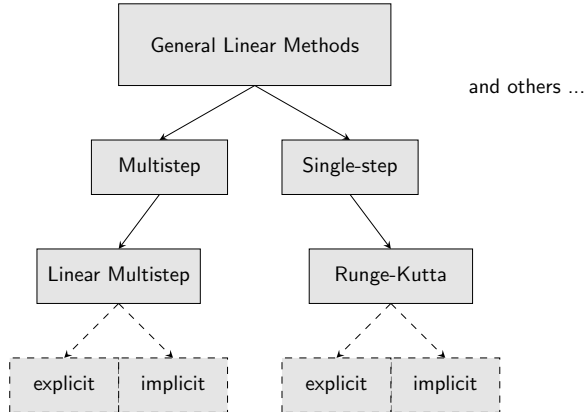
$$s_{k+1} = f(s_k, a_k)$$

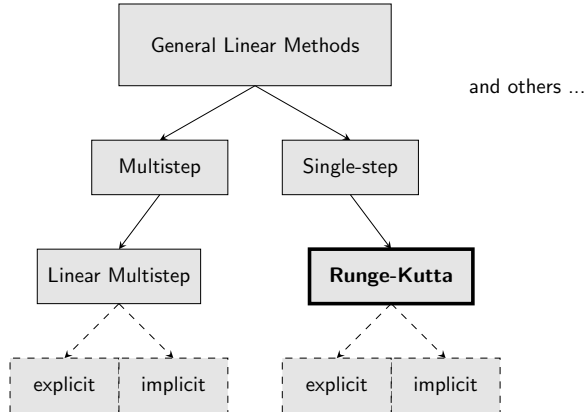
with integer time index $k = 0, 1, 2, \dots$. We often simplify notation to

$$s^+ = f(s, a)$$

Aim of optimal feedback control (including both MPC and RL) is to design a map, or **policy**, $\pi : \mathbb{S} \rightarrow \mathbb{A}$, $s \mapsto a := \pi(s)$ such that **closed-loop system** $s^+ = f(s, \pi(s))$ has desirable properties, such as respecting constraints and minimizing a cost.

In practice, however, we might not be able to directly measure the state s ...





Dynamic System Models

From Continuous to Discrete Time

Implicit Runge-Kutta Methods

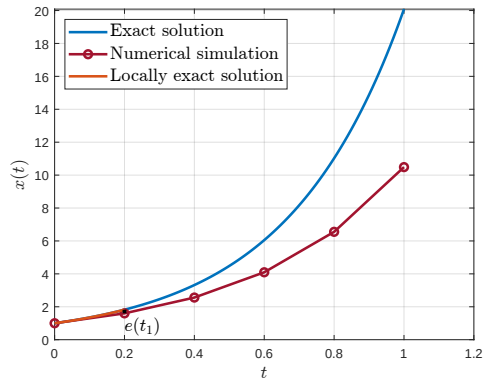
Input Output Models

Stochastic Models

Local and global error

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

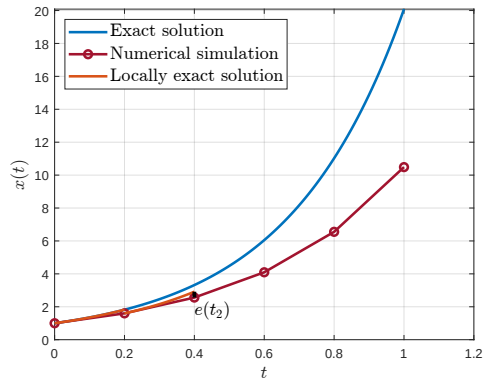
$$e(t_{n+1}) = \|x(t_{n+1}) - f(x(t_n), a_0)\|.$$



Local and global error

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

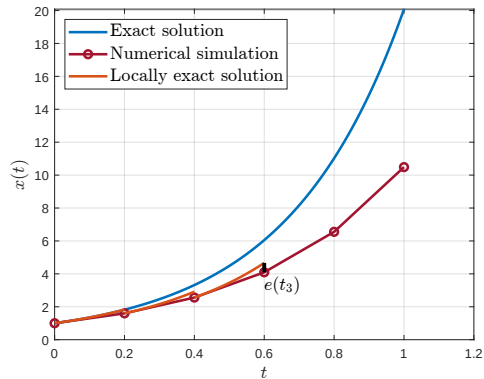
$$e(t_{n+1}) = \|x(t_{n+1}) - f(x(t_n), a_0)\|.$$



Local and global error

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

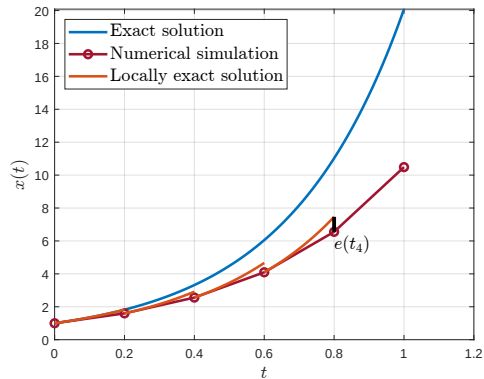
$$e(t_{n+1}) = \|x(t_{n+1}) - f(x(t_n), a_0)\|.$$



Local and global error

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

$$e(t_{n+1}) = \|x(t_{n+1}) - f(x(t_n), a_0)\|.$$



Local and global error

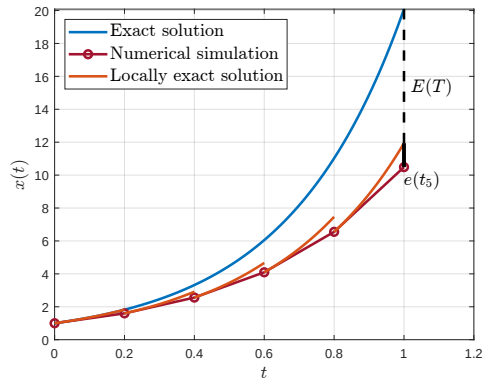
- ▶ Local integration error at t_{n+1} :

$$e(t_{n+1}) = \|x(t_{n+1}) - f(x(t_n), a_0)\|.$$

- ▶ Global integration error at $t = T$:

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

- ▶ The global error is the accumulation of local errors.



Integrator convergence and accuracy

► Convergence

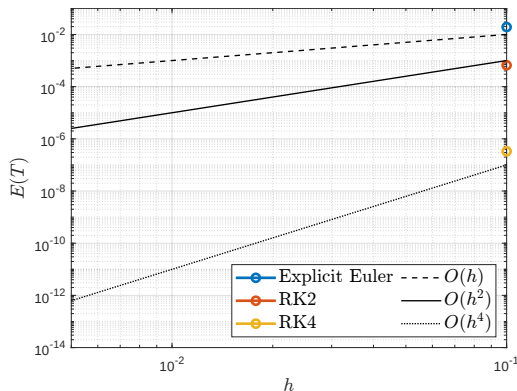
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► Higher order p :

- less, but more expensive steps for same accuracy
- in total fewer r.h.s. evaluations for same accuracy



Integrator convergence and accuracy

► Convergence

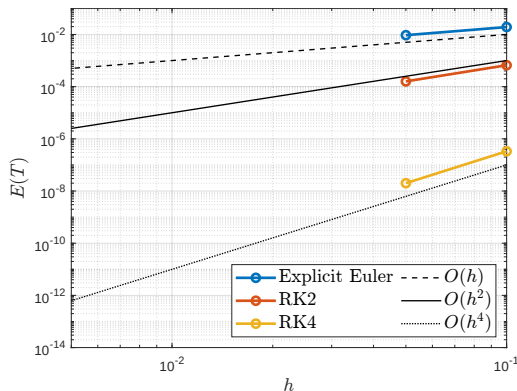
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► Higher order p :

- less, but more expensive steps for same accuracy
- in total fewer r.h.s. evaluations for same accuracy



Integrator convergence and accuracy

► Convergence

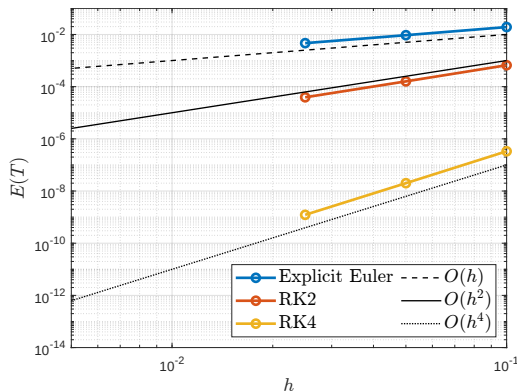
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► Higher order p :

- less, but more expensive steps for same accuracy
- in total fewer r.h.s. evaluations for same accuracy



Integrator convergence and accuracy

► Convergence

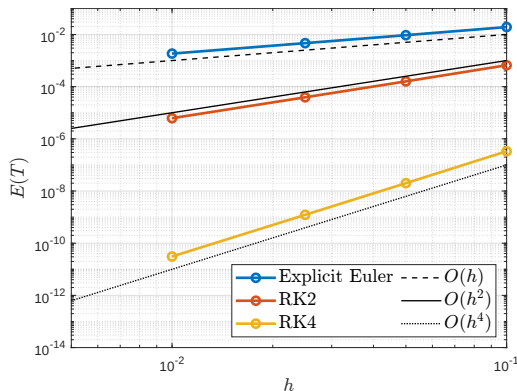
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► Higher order p :

- less, but more expensive steps for same accuracy
- in total fewer r.h.s. evaluations for same accuracy



Integrator convergence and accuracy

► Convergence

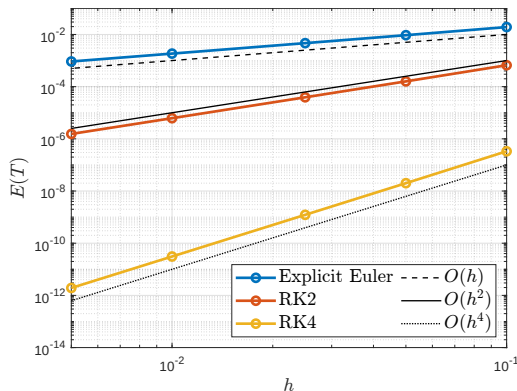
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► Higher order p :

- less, but more expensive steps for same accuracy
- in total fewer r.h.s. evaluations for same accuracy



Integrator convergence and accuracy

► Convergence

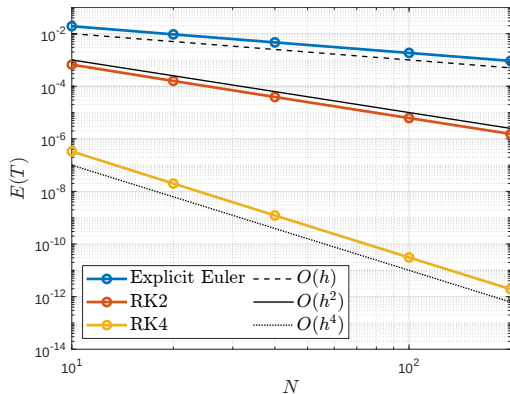
$$\lim_{h \rightarrow 0} E(T) = 0$$

► Integrator has order p if

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

► 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

Definition (Runge-Kutta method in differential form)

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

$$k_{n,i} = f_c(t_{n,i}, s_n + h \sum_{j=1}^{n_s} a_{i,j} k_{n,j}, a_n), \quad i = 1, \dots, n_s$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} b_i k_{n,i}$$

is called a n_s -stage Runge-Kutta (RK) method in the *differential form*.

Notation alert: for the stage “velocities” we use the symbols $k_{i,j}$ or $v_{i,j}$. Do not confuse the Butcher tableau “ $a_{i,j}$ ” with actions “ a_n ”.

Definition (Runge-Kutta method in differential form)

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

$$k_{n,i} = f_c(t_{n,i}, s_n + h \sum_{j=1}^{n_s} a_{i,j} k_{n,j}, a_n), \quad i = 1, \dots, n_s$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} b_i k_{n,i}$$

is called a n_s -stage Runge-Kutta (RK) method in the *differential form*.



Definition (Runge-Kutta method in differential form)

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

$$k_{n,i} = f_c(t_{n,i}, s_n + h \sum_{j=1}^{n_s} a_{i,j} k_{n,j}, a_n), \quad i = 1, \dots, n_s$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} b_i k_{n,i}$$

is called a n_s -stage Runge-Kutta (RK) method in the *differential form*.



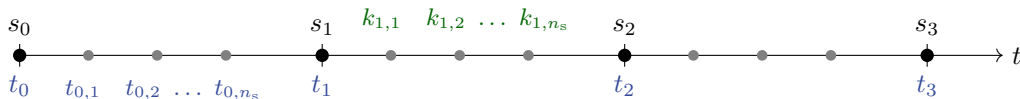
Definition (Runge-Kutta method in differential form)

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

$$k_{n,i} = f_c(t_{n,i}, s_n + h \sum_{j=1}^{n_s} a_{i,j} k_{n,j}, a_n), \quad i = 1, \dots, n_s$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} b_i k_{n,i}$$

is called a n_s -stage Runge-Kutta (RK) method in the *differential form*.



Definition (Runge-Kutta method in differential form)

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

$$k_{n,i} = f_c(t_{n,i}, s_n + h \sum_{j=1}^{n_s} a_{i,j} k_{n,j}, a_n), \quad i = 1, \dots, n_s$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} b_i k_{n,i}$$

is called a n_s -stage Runge-Kutta (RK) method in the *differential form*.

Time grid

$$h, t_n, t_{n,i} \\ i = 1, \dots, n_s$$

Butcher tableau

$$a_{i,j}, b_i, c_i \\ i, j = 1, \dots, n_s$$

Data

$$s_n, a_n, f_c(\cdot)$$

Variables

$$s_{n+1}, k_{n,i} \\ i = 1, \dots, n_s$$

Explicit Runge-Kutta 4

$$k_{n,1} = f_c(t_n, s_n)$$

$$k_{n,2} = f_c(t_n + \frac{h}{2}, s_n + h \frac{k_{n,1}}{2})$$

$$k_{n,3} = f_c(t_n + \frac{h}{2}, s_n + h \frac{k_{n,2}}{2})$$

$$k_{n,4} = f_c(t_n + h, s_n + h k_{n,3})$$

$$s_{n+1} = s_n + h(\frac{1}{6}k_{n,1} + \frac{2}{6}k_{n,2} + \frac{2}{6}k_{n,3} + \frac{1}{6}k_{n,4})$$

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

Explicit Runge-Kutta 4

$$k_{n,1} = f_c(t_n, s_n)$$

$$k_{n,2} = f_c(t_n + \frac{h}{2}, s_n + h\frac{k_{n,1}}{2})$$

$$k_{n,3} = f_c(t_n + \frac{h}{2}, s_n + h\frac{k_{n,2}}{2})$$

$$k_{n,4} = f_c(t_n + h, s_n + hk_{n,3})$$

$$s_{n+1} = s_n + h(\frac{1}{6}k_{n,1} + \frac{2}{6}k_{n,2} + \frac{2}{6}k_{n,3} + \frac{1}{6}k_{n,4})$$

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

Implicit Euler Method

$$k_{n,1} = f_c(t_{n+1}, s_n + hk_{n,1})$$

$$s_{n+1} = s_n + hk_{n,1}$$

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

Explicit Runge-Kutta method

0					
c_2	$a_{2,1}$				
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

Explicit Runge-Kutta method

0					
c_2	$a_{2,1}$				
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

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

Explicit Runge-Kutta method

0					
c_2	$a_{2,1}$				
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

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

Implicit Runge-Kutta method

c_1	$a_{1,1}$	$a_{1,2}$	$\dots$	a_{1,n_s-1}	a_{1,n_s}
c_2	$a_{2,1}$	$a_{2,2}$	$\dots$	a_{2,n_s-1}	a_{2,n_s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	a_{n_s,n_s}
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

Explicit Runge-Kutta method

0					
c_2	$a_{2,1}$				
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

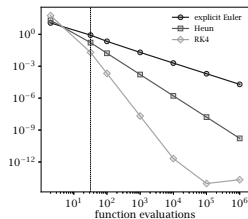
- ▶ $a_{i,j} \neq 0$ only for $j < i$
- ▶ Explicit function evaluations to compute stage values and next step
- ▶ Computationally cheap
- ▶ Order: $p = n_s$ if $n_s \leq 4$ and $p < n_s$ otherwise

Implicit Runge-Kutta method

c_1	$a_{1,1}$	$a_{1,2}$	$\dots$	a_{1,n_s-1}	a_{1,n_s}
c_2	$a_{2,1}$	$a_{2,2}$	$\dots$	a_{2,n_s-1}	a_{2,n_s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
c_{n_s}	$a_{n_s,1}$	$a_{n_s,2}$	$\dots$	a_{n_s,n_s-1}	a_{n_s,n_s}
	b_1	b_2	$\dots$	b_{n_s-1}	b_{n_s}

- ▶ Requires solving nonlinear rootfinding problem with Newton's method
- ▶ Expensive but good for stiff systems
- ▶ Order: $p = 2n_s$, $p = 2n_s - 1$, ...

Butcher tableau, six examples



Euler

0		
1		1

Heun

0			
1		1	
		1/2	1/2

RK4

0					
1/2		1/2			
1/2		0	1/2		
1		0	0	1	
		1/6	2/6	2/6	1/6

Implicit
Euler

1		1
		1

Midpoint
rule (GL2)

1/2		1/2
		1

Gauss-Legendre
of order 4 (GL4)

$1/2 - \sqrt{3}/6$		1/4	$1/4 - \sqrt{3}/6$
$1/2 + \sqrt{3}/6$		$1/4 + \sqrt{3}/6$	1/4
		1/2	1/2

c_1		a_{11}	$\dots$	a_{1s}
c_2		a_{21}	$\dots$	a_{2s}
$\vdots$		$\vdots$		$\vdots$
c_s		a_{s1}	$\dots$	a_{ss}
		b_1	$\dots$	b_s

But how do we obtain a Butcher tableau?



Main ideas:

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



Main ideas:

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



Main ideas:

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

Main ideas:

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

Collocation equations

$$\begin{aligned} q_n(t_n) &= s_n \\ \dot{q}_n(t_n + c_i h) &= f_c(t_n + c_i h, q_n(t_n + c_i h), a_n), \quad i = 1, \dots, n_s \end{aligned}$$

Main ideas:

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

Collocation equations

$$\begin{aligned} q_n(t_n) &= s_n \\ \dot{q}_n(t_n + c_i h) &= f_c(t_n + c_i h, q_n(t_n + c_i h), a_n), \quad i = 1, \dots, n_s \end{aligned}$$

- ▶ Polynomial of degree n_s : $n_s + 1$ coefficient and $n_s + 1$ equations

Main ideas:

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

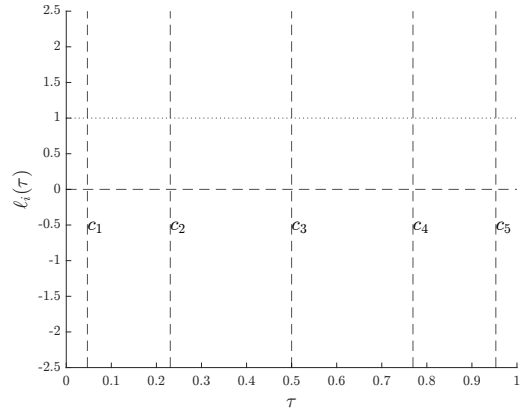
Collocation equations

$$\begin{aligned} q_n(t_n) &= s_n \\ \dot{q}_n(t_n + c_i h) &= f_c(t_n + c_i h, q_n(t_n + c_i h), a_n), \quad i = 1, \dots, n_s \end{aligned}$$

- ▶ Polynomial of degree n_s : $n_s + 1$ coefficient and $n_s + 1$ equations
- ▶ Next value by simple evaluation: $s_{n+1} = q_n(t_{n+1})$

Lagrange polynomial basis

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

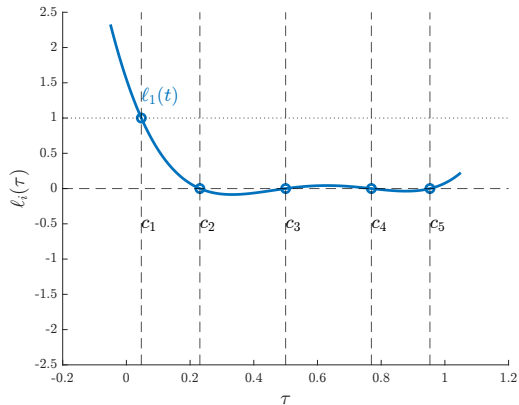


Lagrange polynomial basis

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

Properties:

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

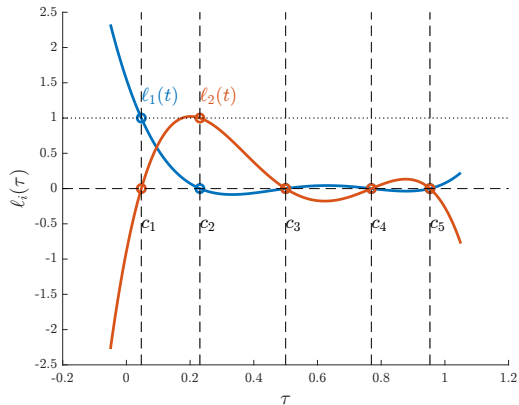


Lagrange polynomial basis

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

Properties:

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

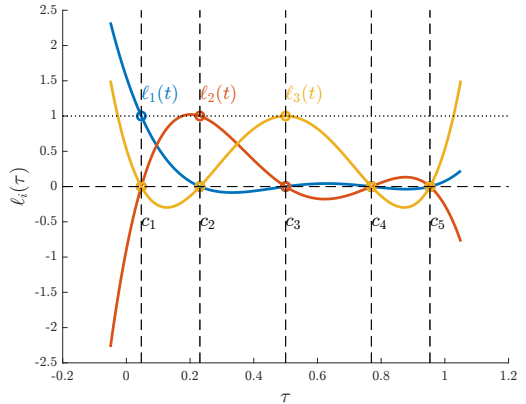


Lagrange polynomial basis

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

Properties:

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

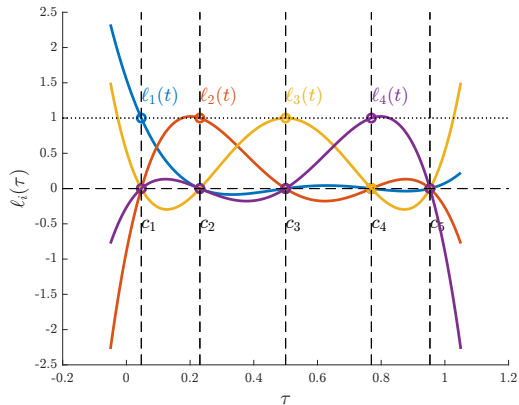


Lagrange polynomial basis

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

Properties:

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

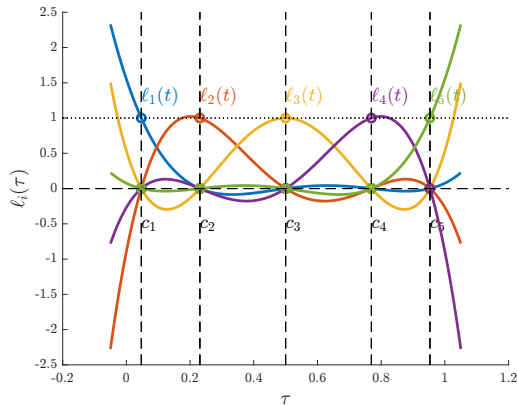


Lagrange polynomial basis

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

Properties:

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



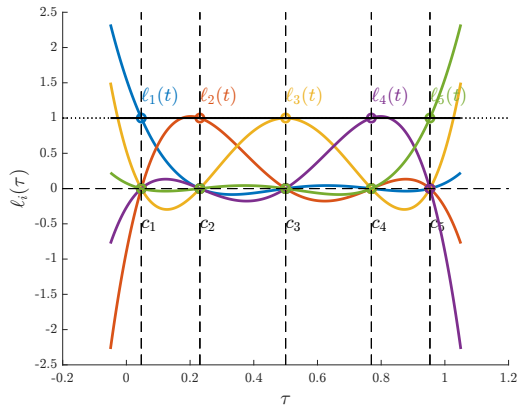
Lagrange polynomial basis

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

Properties:

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

$$\sum_{i=1}^{n_s} \ell_i(t) = 1$$



How to parameterize $q_n(t)$?

Two common (equivalent) choices

How to parameterize $q_n(t)$?

Two common (equivalent) choices

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

How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \dots, k_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through s_n (at t_n) and **state values** $s_{n,1}, \dots, s_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (homework).

How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \dots, k_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through s_n (at t_n) and **state values** $s_{n,1}, \dots, s_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (homework).

► $q_n(t)$ is recovered via:

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

How to parameterize $q_n(t)$?

Two common (equivalent) choices

1. Find $\dot{q}_n(t)$ interpolating polynomial through **state derivatives** $k_{n,1}, \dots, k_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (**this lecture**).
2. Find interpolating polynomial $q_n(t)$ through s_n (at t_n) and **state values** $s_{n,1}, \dots, s_{n,n_s}$ at collocation points $t_{n,i}$, $i = 1, \dots, n_s$ (homework).

► $q_n(t)$ is recovered via:

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

► with:

$$\begin{aligned} \dot{q}_n(t) &= \ell_1 \left(\frac{t - t_n}{h} \right) k_{n,1} + \ell_2 \left(\frac{t - t_n}{h} \right) k_{n,2} + \dots + \ell_{n_s} \left(\frac{t - t_n}{h} \right) k_{n,n_s} \\ &= \sum_{i=1}^{n_s} \ell_i \left(\frac{t - t_n}{h} \right) \underbrace{f_c(t_n + c_i h, q_n(t_n + c_i h), a_0)}_{=k_{n,i}} \end{aligned}$$



- Evaluate $q_n(t)$ at collocation points

$$q_n(t_n + c_i h) = s_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \dots, k_{n,n_s}) d\tau$$



- Evaluate $q_n(t)$ at collocation points

$$\begin{aligned} q_n(t_n + c_i h) &= s_n + \int_{t_n}^{t_n + c_i h} \dot{q}_n(\tau; k_{n,1}, \dots, k_{n,n_s}) d\tau \\ &= s_n + \int_{t_n}^{t_n + c_i h} \sum_{j=1}^{n_s} \ell_j\left(\frac{\tau - t_n}{h}\right) k_{n,j} d\tau \end{aligned}$$

- Evaluate $q_n(t)$ at collocation points

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

- Evaluate $q_n(t)$ at collocation points

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

- Evaluate $q_n(t)$ at collocation points

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

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

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

Collocation equations

$$q_n(t_n) = s_n \quad \text{(initial value)}$$

$$\dot{q}_n(t_n + c_i h) = f_c(t_n + c_i, q_n(t_n + c_i h), a_n), \quad i = 1, \dots, n_s \quad \text{(stage eqs.)}$$

$$s_{n+1} = q_n(t_{n+1}) \quad \text{(next value)}$$

Collocation equations

$$q_n(t_n) = s_n \quad \text{(initial value)}$$

$$\mathbf{k}_{n,i} = f_c(t_n + c_i h, q_n(t_n + c_i h), a_n), \quad i = 1, \dots, n_s \quad \text{(stage eqs.)}$$

$$s_{n+1} = q_n(t_{n+1}) \quad \text{(next value)}$$

Collocation equations

$$q_n(t_n) = s_n \quad (\text{initial value})$$

$$k_{n,i} = f_c(t_n + c_i h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{i,j}, a_n), \quad i = 1, \dots, n_s \quad (\text{stage eqs.})$$

$$s_{n+1} = q_n(t_{n+1}) \quad (\text{next value})$$

Collocation equations

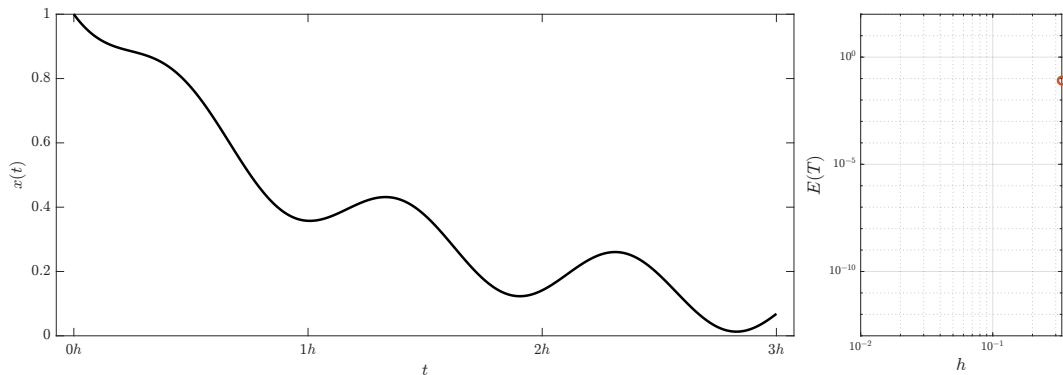
$$q_n(t_n) = s_n \quad (\text{initial value})$$

$$k_{n,i} = f_c(t_n + c_i h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{i,j}, a_n), \quad i = 1, \dots, n_s \quad (\text{stage eqs.})$$

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} k_{n,i} b_i \quad (\text{next value})$$

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

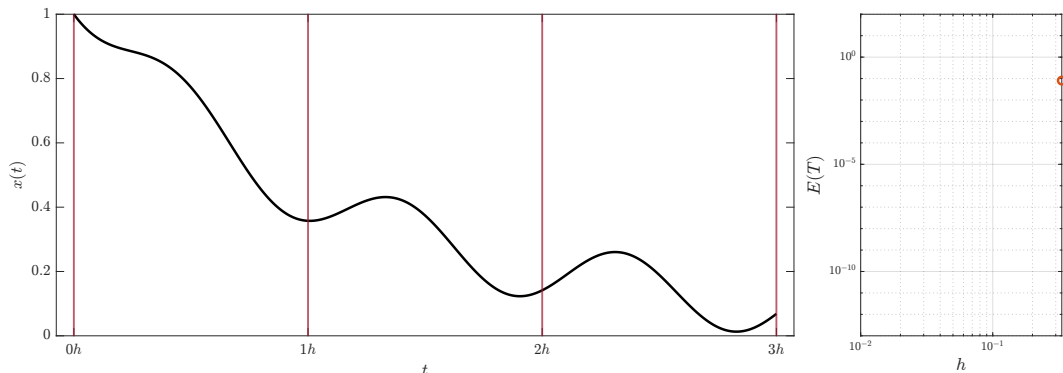
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

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

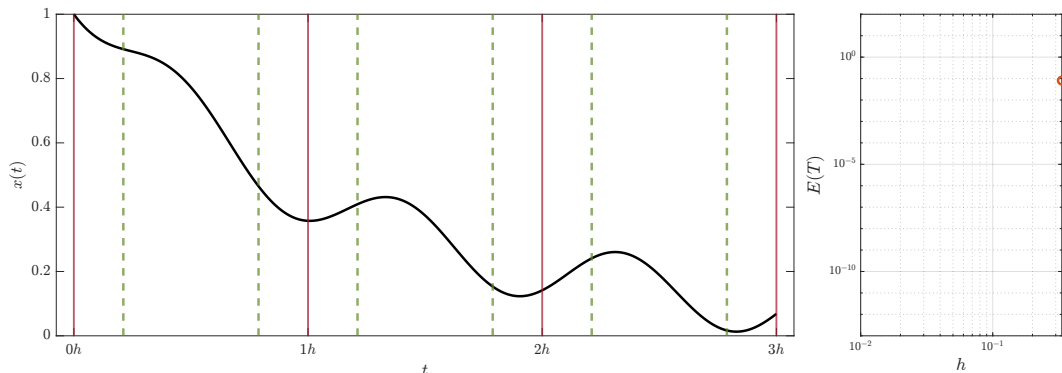
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

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

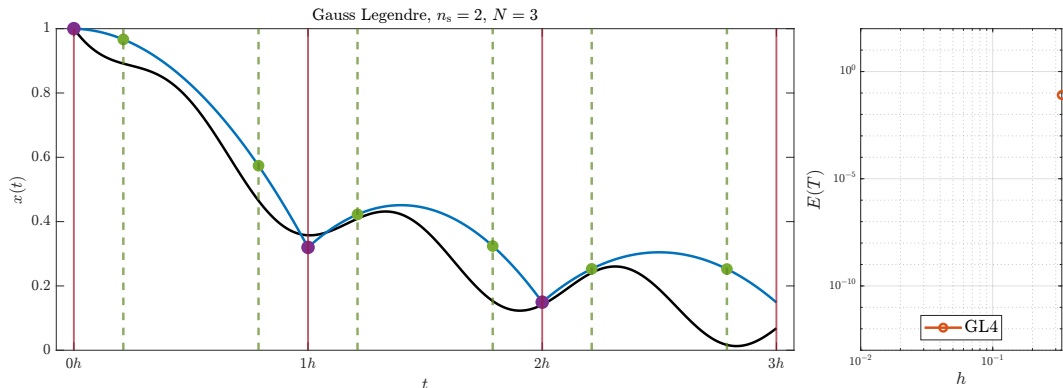
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

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

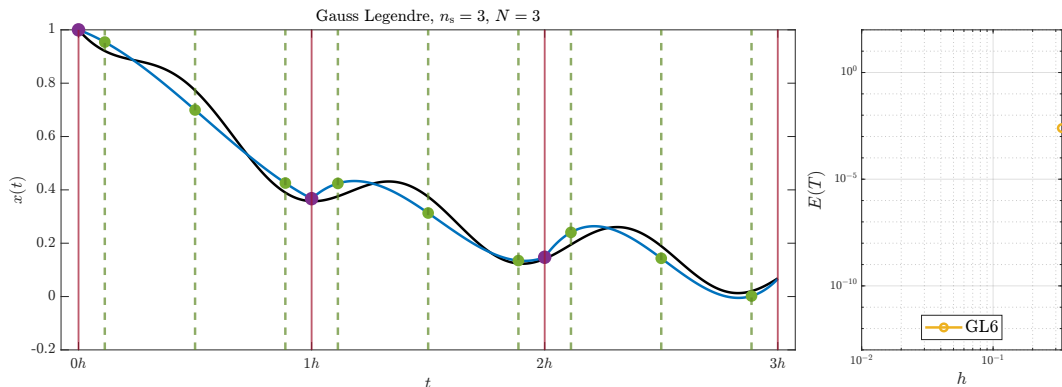
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

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

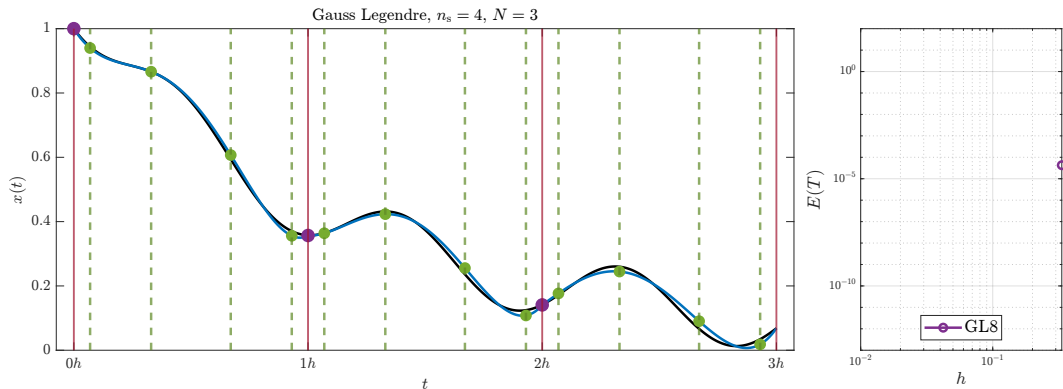
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

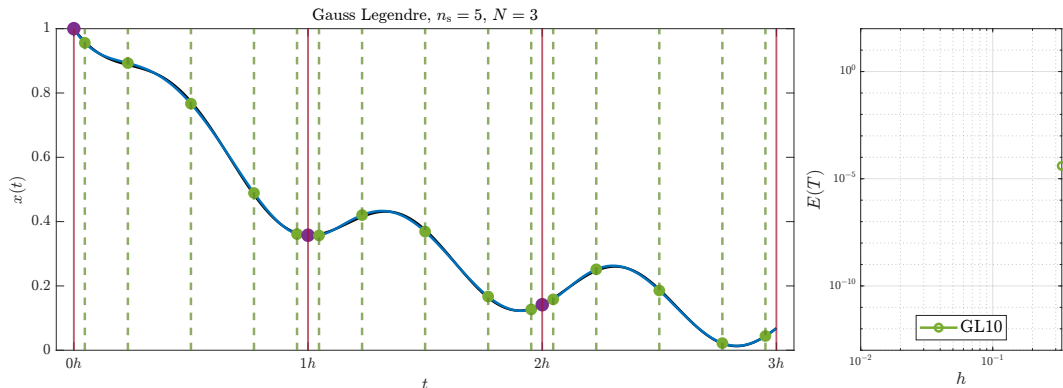
Visualization inspired by Leo Simpson's talk at the European control conference 2023

- ▶ 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 $p = 2n_s - 2$.



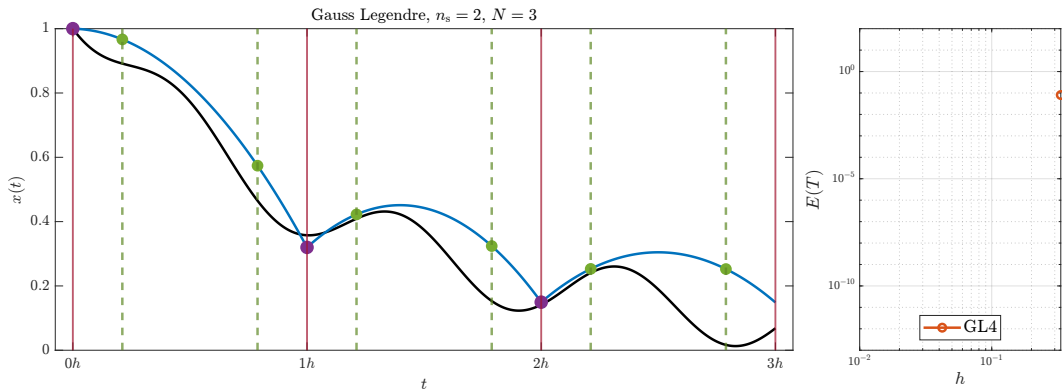
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

- ▶ 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 $p = 2n_s - 2$.



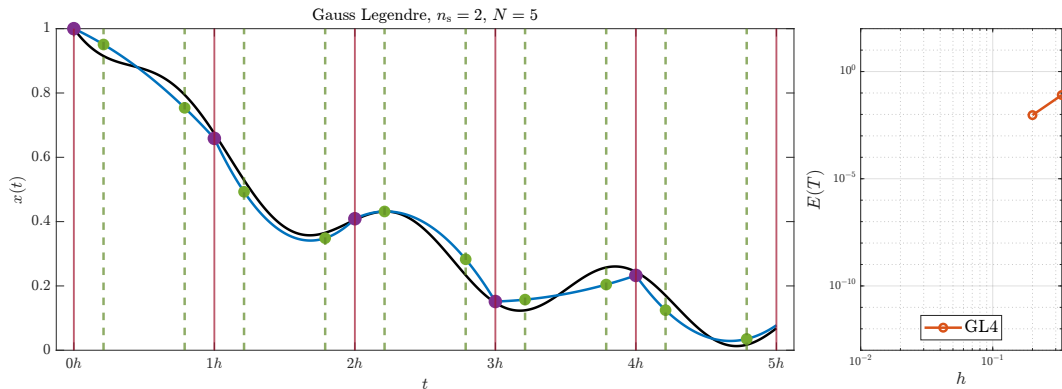
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



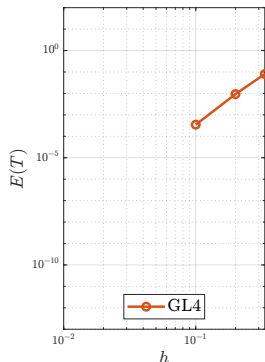
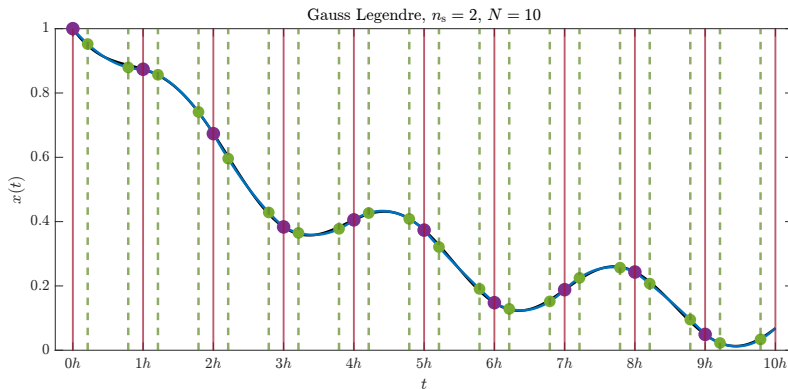
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



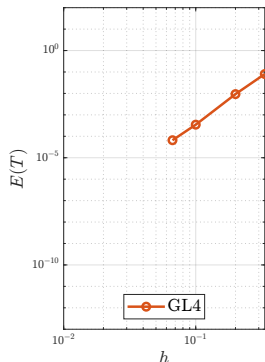
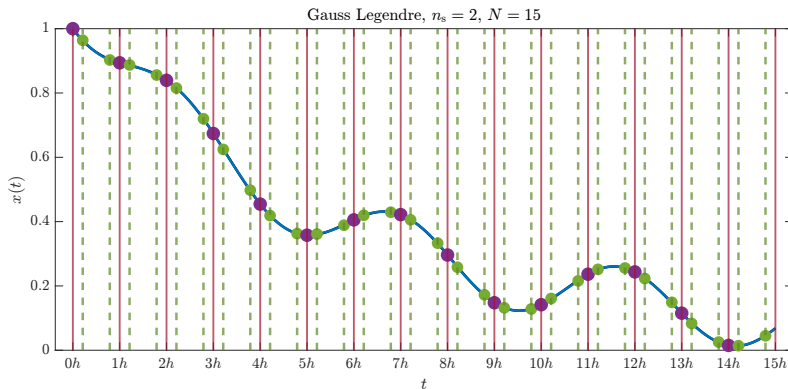
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



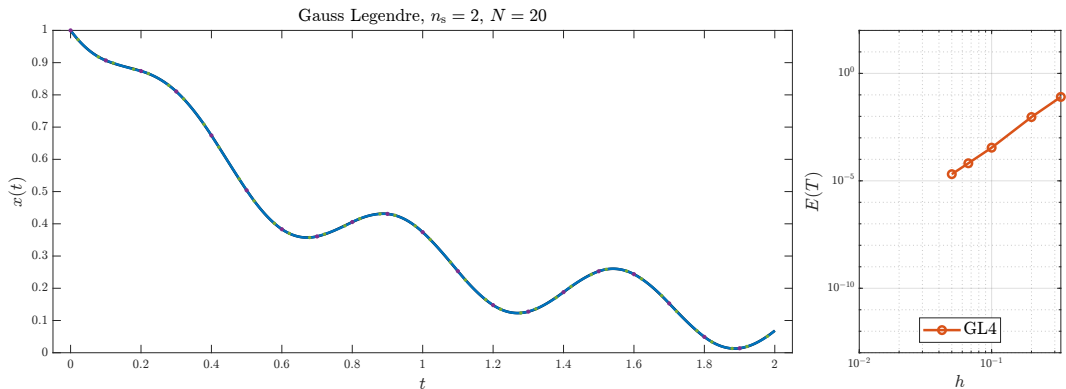
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



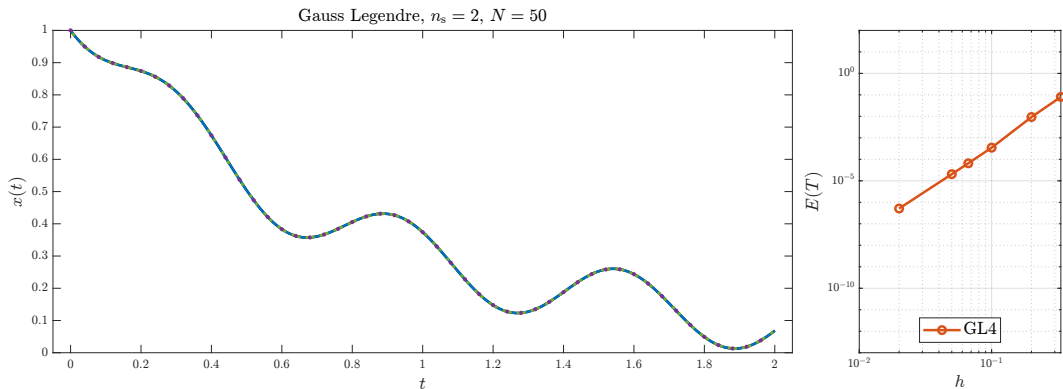
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



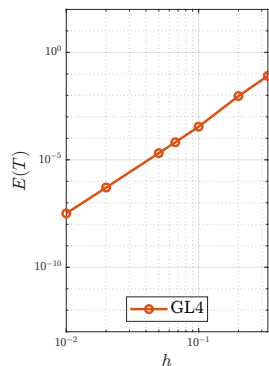
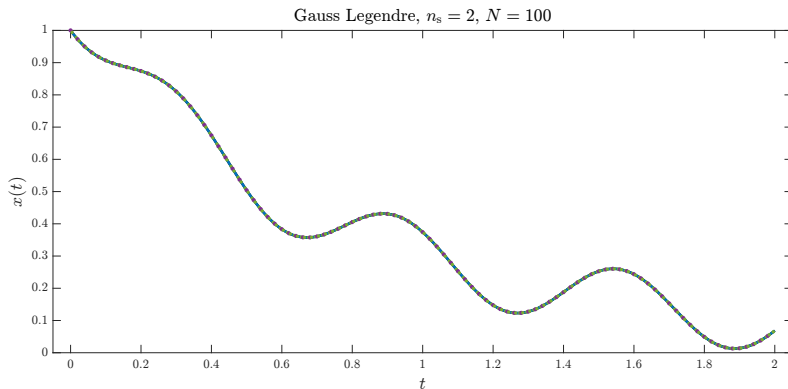
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



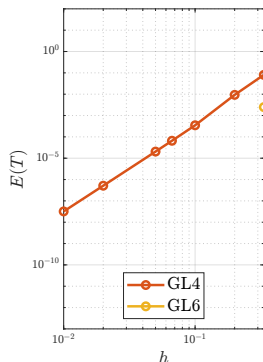
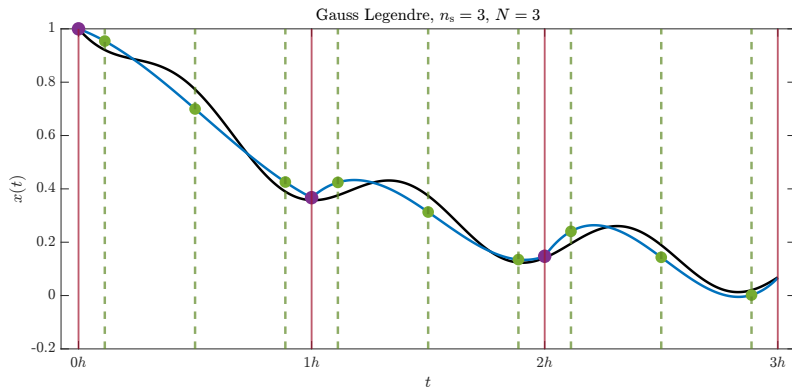
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



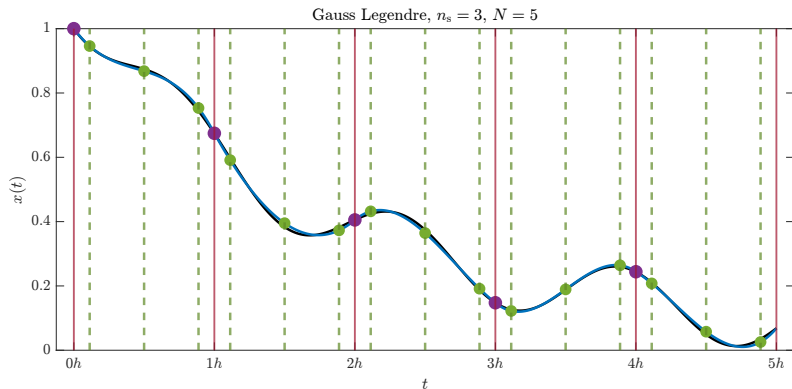
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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

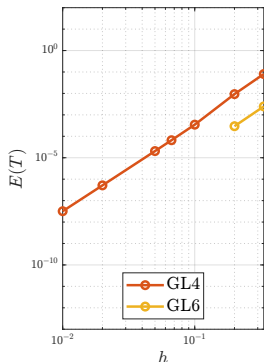


$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

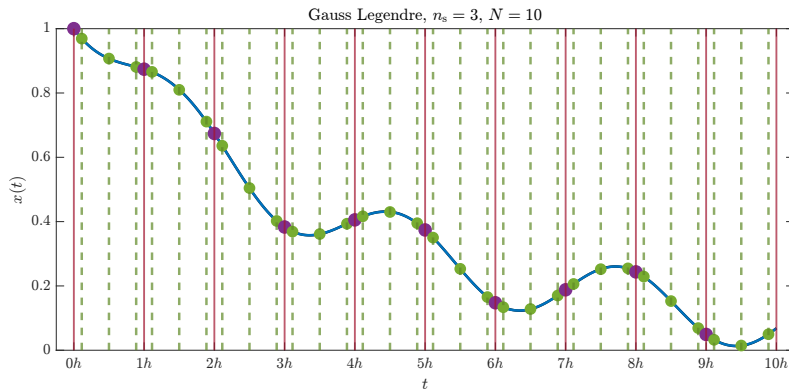
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



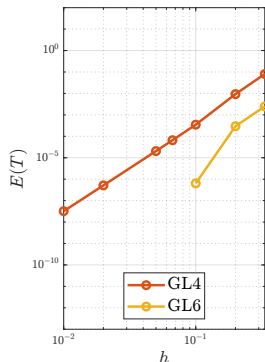
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$



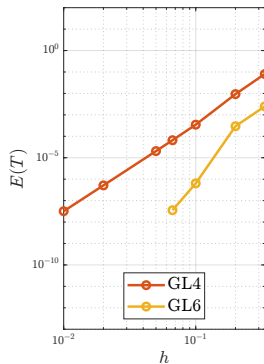
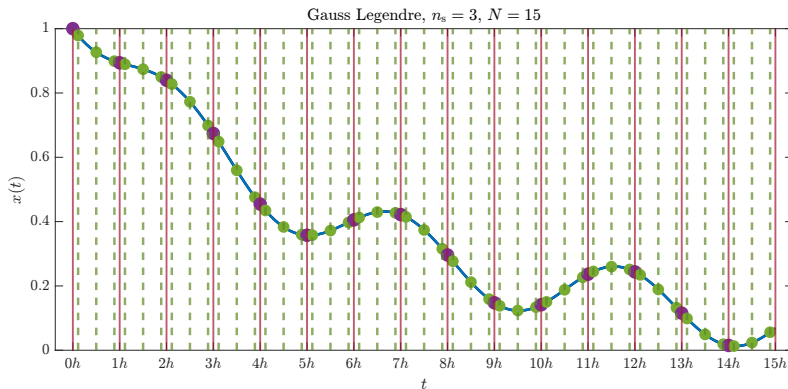
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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

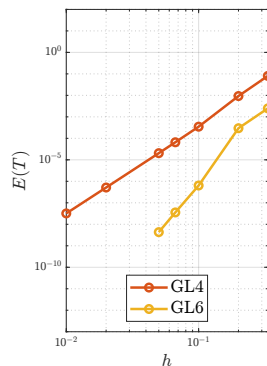
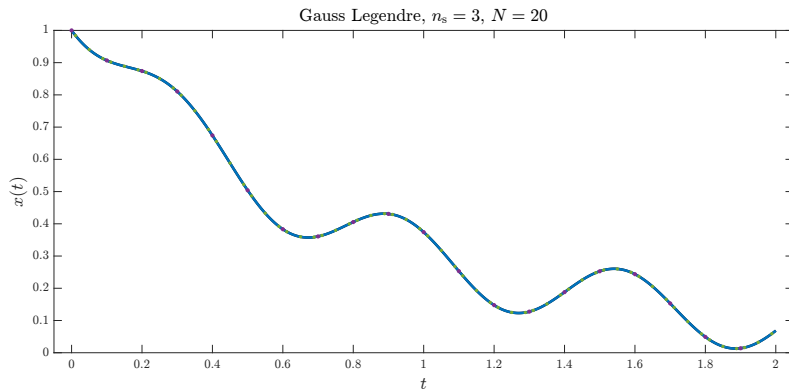


- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



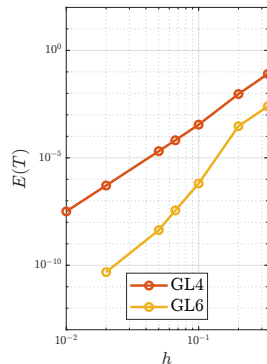
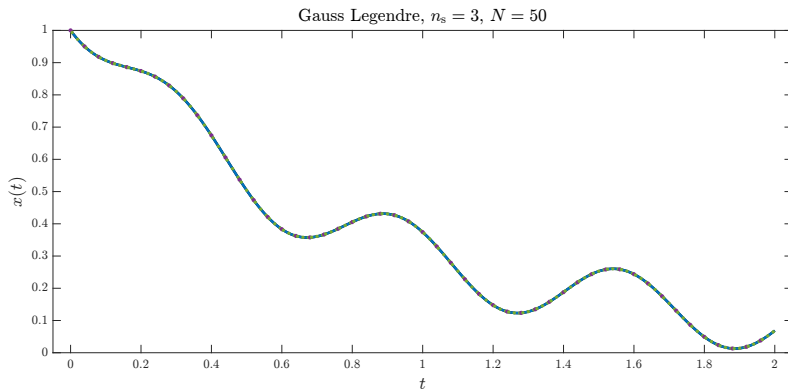
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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



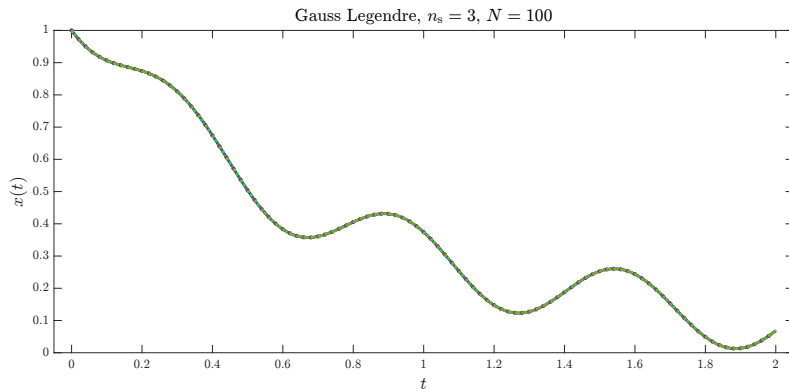
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

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

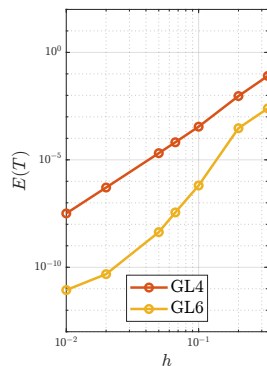


$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$

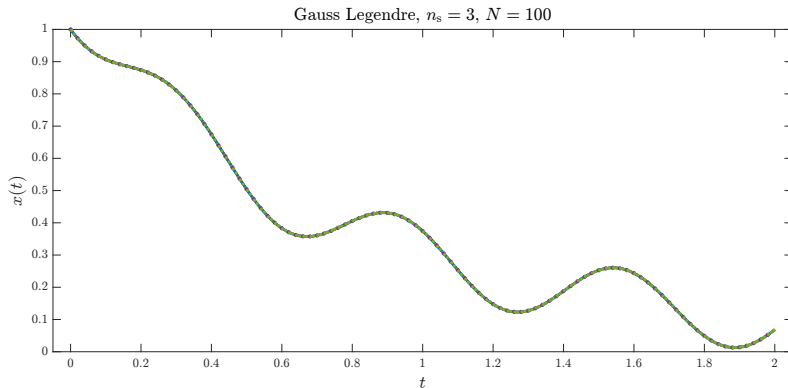
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



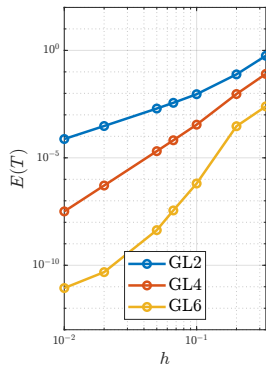
$$\dot{x}(t) = -0.5x(t)^2 - x(t) + \sin(10t), x(0) = 1$$



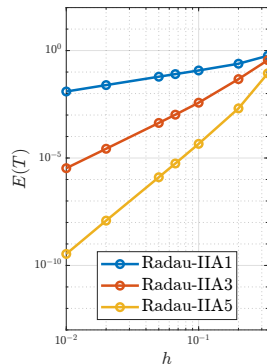
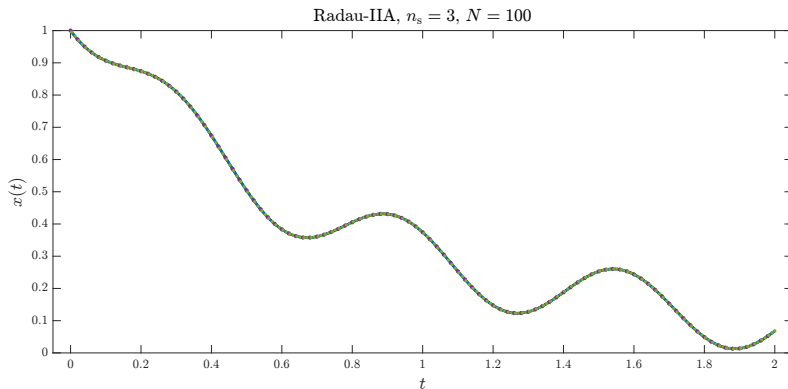
- ▶ Choice of points $c_1, \dots, c_{n_s}$ determines properties of method.
- ▶ Gauss-Legendre $p = 2n_s$, Radau-IIA $p = 2n_s - 1$ good for stiff systems, Lobatto family $p = 2n_s - 2$.



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



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



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

Variables $s_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_s}) \in \mathbb{R}^{n_s n_x}$

Collocation equations

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} k_{n,i} b_i \quad (\text{next value})$$

$$k_{n,1} = f_c(t_n + c_1 h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{1,j}, a_n) \quad (\text{stage Eq. 1})$$

$$\vdots$$

$$k_{n,n_s} = f_c(t_n + c_{n_s} h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{n_s,j}, a_n), \quad (\text{stage Eq. } n_s)$$

Variables $s_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_s}) \in \mathbb{R}^{n_s n_x}$

Collocation equations

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} k_{n,i} b_i \quad (\text{next value})$$

$$0 = k_{n,1} - f_c(t_n + c_1 h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{1,j}, a_n) \quad (\text{stage Eq. 1})$$

$$\vdots$$

$$0 = k_{n,n_s} - f_c(t_n + c_{n_s} h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{n_s,j}, a_n), \quad (\text{stage Eq. } n_s)$$

Variables $s_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_s}) \in \mathbb{R}^{n_s n_x}$

Collocation equations

$$s_{n+1} = s_n + h \sum_{i=1}^{n_s} k_{n,i} b_i =: \phi_f(s_n, z_n, a_n) \quad (\text{next value})$$

$$0 = \begin{bmatrix} k_{n,1} - f_c(t_n + c_1 h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{1,j}, a_n) \\ \vdots \\ k_{n,n_s} - f_c(t_n + c_{n_s} h, s_n + h \sum_{j=1}^{n_s} k_{n,j} a_{n_s,j}, a_n) \end{bmatrix} =: g(s_n, z_n, a_n) \quad (\text{stage Eqs.})$$

Variables $s_{n+1} \in \mathbb{R}^{n_x}$ and $z_n = (k_{n,1}, \dots, k_{n,n_s}) \in \mathbb{R}^{n_s n_x}$

Collocation equations

$$\begin{aligned} s_{n+1} &= f(s_n, z_n, a_n) && \text{(next value)} \\ 0 &= g(s_n, z_n, a_n) && \text{(stage Eqs.)} \end{aligned}$$

- Use to discretize optimal control problem



Continuous time OCP

$$\begin{aligned} \min_{s(\cdot), a(\cdot)} \quad & \int_0^T L_c(s(t), a(t)) dt + E(s(T)) \\ \text{s.t.} \quad & s(0) = \bar{s}_0, \\ & \dot{s}(t) = f_c(s(t), a(t)), \\ & h(s(t), a(t)) \leq 0, \quad t \in [0, T], \\ & r(s(T)) \leq 0 \end{aligned}$$

- Direct methods: first discretize, then optimize

Continuous time OCP

$$\begin{aligned} \min_{s(\cdot), a(\cdot)} \quad & \int_0^T L_c(s(t), a(t)) dt + E(s(T)) \\ \text{s.t.} \quad & s(0) = \bar{s}_0, \\ & \dot{s}(t) = f_c(s(t), a(t)), \\ & h(s(t), a(t)) \leq 0, \quad t \in [0, T], \\ & r(s(T)) \leq 0 \end{aligned}$$

- Direct methods: first discretize, then optimize

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

Continuous time OCP

$$\begin{aligned} \min_{s(\cdot), a(\cdot)} \quad & \int_0^T L_c(s(t), a(t)) dt + E(s(T)) \\ \text{s.t.} \quad & s(0) = \bar{s}_0, \\ & \dot{s}(t) = f_c(s(t), a(t)), \\ & h(s(t), a(t)) \leq 0, \quad t \in [0, T], \\ & r(s(T)) \leq 0 \end{aligned}$$

- Direct methods: first discretize, then optimize

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

$$L_d(s_n, a_n) = \int_{t_n}^{t_{n+1}} L_c(s(t), a(t)) dt.$$

Replace $\dot{s} = f_c(s, a)$ by

$$\begin{aligned} s_{n+1} &= f(s_n, z_n, a_n), \\ 0 &= g(s_n, z_n, a_n). \end{aligned}$$

Continuous time OCP

$$\begin{aligned} \min_{s(\cdot), a(\cdot)} \quad & \int_0^T L_c(s(t), a(t)) dt + E(s(T)) \\ \text{s.t.} \quad & s(0) = \bar{s}_0, \\ & \dot{s}(t) = f_c(s(t), a(t)), \\ & h(s(t), a(t)) \leq 0, \quad t \in [0, T], \\ & r(s(T)) \leq 0 \end{aligned}$$

- Direct methods: first discretize, then optimize

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

$$L_d(s_n, a_n) = \int_{t_n}^{t_{n+1}} L_c(s(t), a(t)) dt.$$

Replace $\dot{s} = f_c(s, a)$ by

$$\begin{aligned} s_{n+1} &= f(s_n, z_n, a_n), \\ 0 &= g(s_n, z_n, a_n). \end{aligned}$$

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

$$0 \geq h(s_n, a_n), \quad n = 0, \dots, N-1.$$

Continuous time OCP

$$\begin{aligned} \min_{s(\cdot), a(\cdot)} \quad & \int_0^T L_c(s(t), a(t)) dt + E(s(T)) \\ \text{s.t.} \quad & s(0) = \bar{s}_0, \\ & \dot{s}(t) = f_c(s(t), a(t)), \\ & h(s(t), a(t)) \leq 0, \quad t \in [0, T], \\ & r(s(T)) \leq 0 \end{aligned}$$

- Direct methods: first discretize, then optimize

Discrete time OCP (an NLP)

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{z}, \mathbf{a}} \quad & \sum_{k=0}^{N-1} L_d(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{n+1} = f(s_n, z_n, a_n), \\ & 0 = g(s_n, z_n, a_n), \\ & h(s_n, a_n) \leq 0, \quad n = 0, \dots, N-1, \\ & r(s_N) \leq 0 \end{aligned}$$

Variables $\mathbf{s} = (s_0, \dots, s_N)$, $\mathbf{z} = (z_0, \dots, z_N)$
and $\mathbf{a} = (a_0, \dots, a_{N-1})$.

Discrete time OCP (an NLP)

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{z}, \mathbf{a}} \quad & \sum_{k=0}^{N-1} L_d(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{n+1} = f(s_n, z_n, a_n), \\ & 0 = g(s_n, z_n, a_n), \\ & h(s_n, a_n) \leq 0, \quad n = 0, \dots, N-1, \\ & r(s_N) \leq 0 \end{aligned}$$

Variables $w = (\mathbf{s}, \mathbf{z}, \mathbf{a})$

Discrete time OCP (an NLP)

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{z}, \mathbf{a}} \quad & \sum_{k=0}^{N-1} L_d(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{n+1} = f(s_n, z_n, a_n), \\ & 0 = g(s_n, z_n, a_n), \\ & h(s_n, a_n) \leq 0, \quad n = 0, \dots, N-1, \\ & r(s_N) \leq 0 \end{aligned}$$

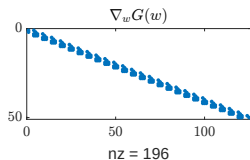
Variables $w = (\mathbf{s}, \mathbf{z}, \mathbf{a})$

Nonlinear Program (NLP)

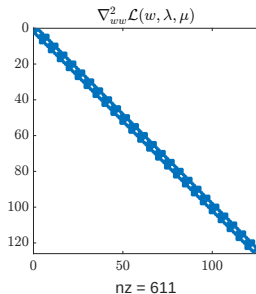
$$\begin{aligned} \min_{w \in \mathbb{R}^n} \quad & F(w) \\ \text{s.t.} \quad & G(w) = 0, \\ & H(w) \geq 0 \end{aligned}$$

Obtain large and sparse NLP

Direct optimal control methods solve Nonlinear Programs (NLP)



Variables $w = (\mathbf{s}, \mathbf{z}, \mathbf{a})$



Nonlinear Program (NLP)

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

Obtain large and sparse NLP

Direct collocation NLP

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{z}, \mathbf{a}} \quad & \sum_{k=0}^{N-1} L_d(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{n+1} = f(s_n, z_n, a_n), \\ & 0 = g(s_n, z_n, a_n), \\ & h(s_n, a_n) \leq 0, \quad n = 0, \dots, N-1, \\ & r(s_N) \leq 0 \end{aligned}$$

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

Multiple shooting NLP

$$\begin{aligned} \min_{\mathbf{s}, \mathbf{a}} \quad & \sum_{k=0}^{N-1} L_d(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{n+1} = f(s_n, a_n), \\ & h(s_n, a_n) \leq 0, \quad n = 0, \dots, N-1, \\ & r(s_N) \leq 0 \end{aligned}$$

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

Dynamic System Models

From Continuous to Discrete Time

Implicit Runge-Kutta Methods

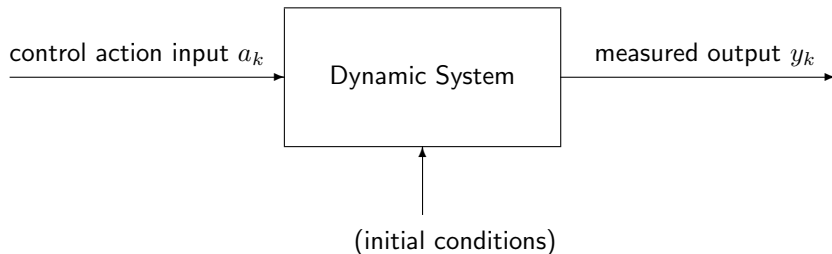
Input Output Models

Stochastic Models

The (realistic) Input Output Perspective



- ▶ In practice, we cannot measure the state. And the state representation is not even unique.
- ▶ A system model should allow us to predict, for any horizon length N and sequence of control actions $(a_1, \dots, a_N)$, the sequence of measured **outputs** $(y_0, \dots, y_N)$.
- ▶ Typically, we need to also specify some **initial conditions** (e.g. the initial state s_0)



- State Space Models with outputs:

$$\begin{aligned} s_{k+1} &= f(s_k, a_k) \\ y_k &= g(s_k, a_k) \end{aligned} \quad \text{for } k = 0, 1, 2, \dots$$

Initial conditions = initial state s_0 .

- Input Output Models (of order n):

$$y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n}) \quad \text{for } k = n, n+1, n+2, \dots$$

Initial conditions: $y_0, \dots, y_{n-1}$ and $a_0, \dots, a_{n-1}$.

Visualization of recurrence $y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, a_{k-1}, \dots, a_{k-n})$:

Control action sequence:



Output sequence:



"state s_k "

- ▶ can **always transform input-output to state-space models**:
- ▶ state: $s_k = (y_{k-1}, a_{k-1}, \dots, y_{k-n}, a_{k-n})$ (defined for $k \geq n$)
- ▶ state transition $s \mapsto s^+ = f(s, a)$ described by

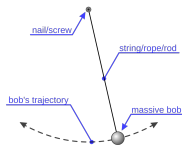
$$s_k = \begin{bmatrix} y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \\ y_{k-n} \\ a_{k-n} \end{bmatrix} \mapsto s_{k+1} = \begin{bmatrix} y_k \\ a_k \\ y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \end{bmatrix} = f(s_k, a_k) := \begin{bmatrix} h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n}) \\ a_k \\ y_{k-1} \\ a_{k-1} \\ \vdots \\ y_{k-n+1} \\ a_{k-n+1} \end{bmatrix}$$

- ▶ output equation: $y_k = g(s_k, a_k) := h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n})$.
- ▶ conversely, we can **sometimes transform state-space to input-output models**, e.g. in case of observable and controllable linear time invariant (LTI) models

Some ODE Examples – what can be measured?

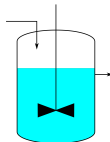


- ▶ Pendulum
- ▶ Hot plate with pot
- ▶ Continuously Stirred Tank Reactors (CSTR)
- ▶ Robot arms
- ▶ Mobile robots
- ▶ Race cars
- ▶ Airplanes in free flight



Pendulum

Commons: Chris-martin, Public domain



CSTR

Commons: Razorbliss, CC0



Mobile robots

Commons: Jiuguang Wang, CC BY-SA 3.0



Hot plate with pot

Commons: Carnildo, Public domain



Robot arms

Commons: Mixabest, CC BY-SA 3.0



Race cars

Commons: edvvc, CC BY 2.0

Dynamic System Models

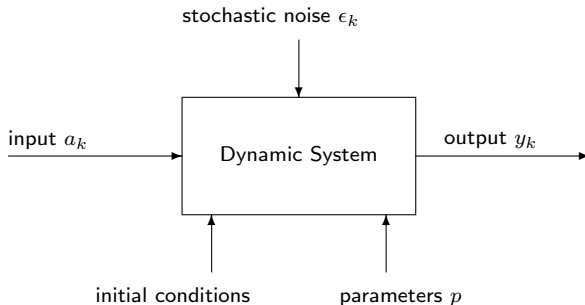
From Continuous to Discrete Time

Implicit Runge-Kutta Methods

Input Output Models

Stochastic Models

- ▶ in reality, we always have some random noise ϵ_k , e.g., disturbances or measurement errors
- ▶ also, we usually have unknown, but constant system parameters p



(parameters can be seen as states that obey the dynamics $p_{k+1} = p_k$ and will often be omitted)

General Form (with random ϵ_k):

Stochastic State Space Model

$$s_{k+1} = f(s_k, a_k, \epsilon_k)$$

$$y_k = g(s_k, a_k, \epsilon_k)$$

Special Cases:

► State Noise and Output Errors:

$$s_{k+1} = f(s_k, a_k) + \epsilon_k^{\text{SN}}$$

$$y_k = g(s_k, a_k) + \epsilon_k^{\text{OE}}$$

Stochastic Input Output Model

$$y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n}, \epsilon_k, \dots, \epsilon_{k-n})$$

► Equation Errors:

$$y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n}) + \epsilon_k^{\text{EE}}$$

Prior to implementing an MPC controller, one needs to address two tasks:

- ▶ **System Identification (offline):**

use a long sequence of recorded input and output data, $(a_0, \dots, a_N)$ and $(y_0, \dots, y_N)$, to identify parameters p using e.g. least squares optimization or subspace identification

- ▶ **State Estimation (online):**

estimate the state s_k by using the previous control actions $(\dots, a_{k-2}, a_{k-1})$ and the past measurements $(\dots, y_{k-2}, y_{k-1})$ using e.g. Extended Kalman Filter (EKF) or moving horizon estimation (MHE) (MHE uses a fixed window of past data for fitting)

Learning-based MPC typically refers to an online model adaptation, i.e., to estimating parameters online (for which MHE is particularly suitable) (“learning a model” = “system identification”)

Note: need state estimation only for *partially observable markov decision processes (POMDP)*

State Space View:

Partially Observable MDP

$$\begin{aligned}s_{k+1} &= f(s_k, a_k, \epsilon_k) \\ y_k &= g(s_k, a_k, \epsilon_k)\end{aligned}$$

with independent identically distributed ϵ_k

Fully Observable MDP

$$\begin{aligned}s_{k+1} &= f(s_k, a_k, \epsilon_k) \\ y_k &= s_k\end{aligned}$$

with $y_k \in \mathbb{R}^{n_s}$

Probabilistic View:

Partially Observable MDP

$$\begin{aligned}P_{\text{state}}(s_{k+1} | s_k, a_k) \\ P_{\text{meas}}(y_k | s_k, a_k)\end{aligned}$$

with probability density functions $P(\cdot)$

Fully Observable MDP

$$\begin{aligned}P_{\text{state}}(s_{k+1} | s_k, a_k) \\ P_{\text{meas}}(y_k | s_k, a_k) &= \delta(y_k - s_k)\end{aligned}$$

with Dirac's Delta function $\delta(\cdot)$ in $\mathbb{R}^{n_s}$

- ▶ We can avoid estimation task by assuming input-output (I/O) models of fixed order n
- ▶ This assumption leads to a **fully observable** markov decision process (MDP)
- ▶ State s_k at time k is then given by $s_k = (y_{k-1}, a_{k-1}, \dots, y_{k-n}, a_{k-n})$
- ▶ Reinforcement Learning (RL) algorithms often use I/O-models (“end-to-end learning”)
- ▶ I/O-models also used in some *linear MPC* implementations based on LTI models, e.g.

$$y_k = \sum_{i=0}^n b_i a_{k-i} + (y_{k-1} - \sum_{i=0}^n b_i a_{k-i-1}) + \epsilon_k$$

- ▶ I/O-models also used for nonlinear black-box MPC or model-based RL which use neural networks for the mapping $y_k = h(y_{k-1}, \dots, y_{k-n}, a_k, \dots, a_{k-n})$

- ▶ We distinguish different model types
 - ▶ continuous vs discrete state and control
 - ▶ continuous vs discrete time
 - ▶ linear vs nonlinear
 - ▶ state space vs input output
 - ▶ deterministic vs stochastic
 - ▶ fully or partially observable
(not to be confused with "observability" in systems theory)
- ▶ We transform differential equations to discrete time via *numerical simulation*
- ▶ We denote *deterministic discrete time models* and *Markov Decision Processes (MDP)* by

$$\boxed{s^+ = f(s, a)} \quad \text{and} \quad \boxed{P(s^+|s, a)}$$

with state $s \in \mathbb{R}^{n_s}$ and control action $a \in \mathbb{R}^{n_a}$