Numerics of Machine Learning
Lecture 06
Solving Ordinary Differential Equations

Nathanael Bosch & Jonathan Schmidt
24 November 2022

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Science
Department of Computer Science
Chair for the Methods of Machine Learning

Where are we in the course?

- Last week: **State-space models and extended Kalman filters/smoothers** (“How to estimate the *state* of a dynamical system from *observations*”)
- This week: **Ordinary differential equations and how to solve them** (“How to *simulate*, approximately, the evolution of a deterministic dynamical system”)

Today:

- What is an ordinary differential equation (ODE) and why should we care?
- **How to numerically solve an ODE:** From Euler (forward and backward) to Runge–Kutta
- **Parameter inference in ODEs** (and *neural* ODEs)

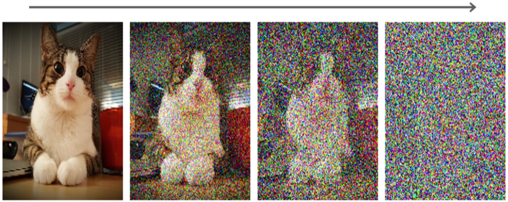
Ordinary differential equation:

$$\dot{x}(t) = f(x(t), t), \qquad t \in \mathbb{T} \subset \mathbb{R},$$

where

- $x : \mathbb{T} \to \mathbb{R}^d$ is the unknown function
- $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the *vector field*
- $\mathbb{T}$ is the time domain; typically $\mathbb{T} = [0, T]$

- **Diffusion Models**
  ODEs and SDEs for generative modeling

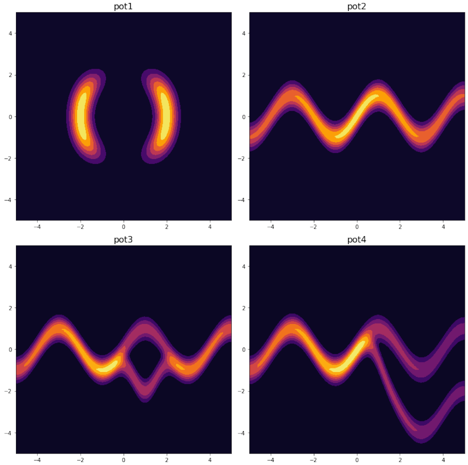


https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/

Differential Equations in Machine Learning

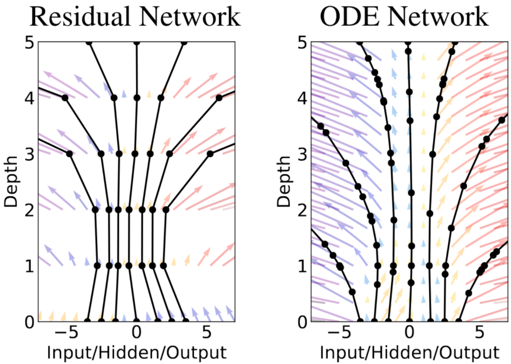
Differential Equations can be found *everywhere*

- **Diffusion Models**
  ODEs and SDEs for generative modeling
- **Normalizing Flows**
  ODEs as bijectors to model distributions



https://docs.pymc.io/en/v3/pymc-examples/examples/variational_inference/normalizing_flows_overview.html

- **Diffusion Models**
  ODEs and SDEs for generative modeling
- **Normalizing Flows**
  ODEs as bijectors to model distributions
- **Neural ODEs**
  ResNets as discretized ODEs

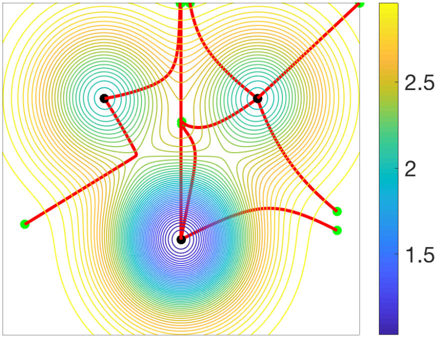


Chen et al, "Neural Ordinary Differential Equations", NeurIPS 2018

Differential Equations in Machine Learning

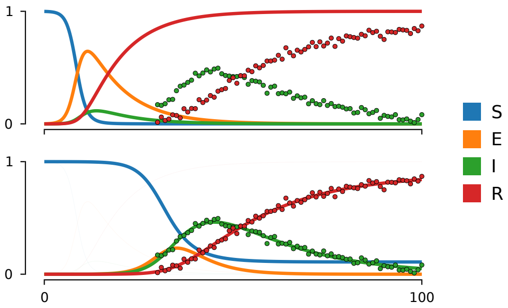
Differential Equations can be found *everywhere*

- **Diffusion Models**
  ODEs and SDEs for generative modeling
- **Normalizing Flows**
  ODEs as bijectors to model distributions
- **Neural ODEs**
  ResNets as discretized ODEs
- **Optimization Theory**
  Gradient descent follows ODE dynamics



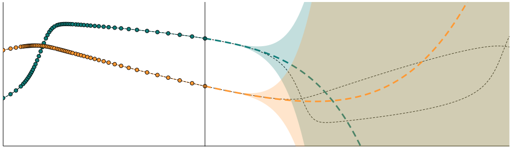
https://francisbach.com/gradient-flows/

- **Diffusion Models**
  ODEs and SDEs for generative modeling
- **Normalizing Flows**
  ODEs as bijectors to model distributions
- **Neural ODEs**
  ResNets as discretized ODEs
- **Optimization Theory**
  Gradient descent follows ODE dynamics
- **Parameter Inference** (later this lecture!)
  ODEs as inductive bias



Tronarp, Bosch, Hennig, "Fenrir: Physics-Enhanced Regression for Initial Value Problems", ICML 2022

- **Diffusion Models**
  ODEs and SDEs for generative modeling
- **Normalizing Flows**
  ODEs as bijectors to model distributions
- **Neural ODEs**
  ResNets as discretized ODEs
- **Optimization Theory**
  Gradient descent follows ODE dynamics
- **Parameter Inference** (later this lecture!)
  ODEs as inductive bias
- **Probabilistic Numerics** (next lecture!)
  ODE solving as *learning*



`https://raw.githubusercontent.com/nathanaelbosch/ProbNumDiffEq.jl/main/examples/banner.svg`

Ordinary differential equation :

$$\dot{x}(t) = f(x(t), t), \qquad t \in \mathbb{T} \subset \mathbb{R},$$

where

- $x : \mathbb{T} \to \mathbb{R}^d$ is the unknown function
- $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the *vector field*
- $\mathbb{T}$ is the time domain; typically $\mathbb{T} = [0, T]$

Ordinary differential equation :

$$\dot{x}(t) = f(x(t), t), \qquad t \in \mathbb{T} \subset \mathbb{R},$$

where

- $x : \mathbb{T} \to \mathbb{R}^d$ is the unknown function
- $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the *vector field*
- $\mathbb{T}$ is the time domain; typically $\mathbb{T} = [0, T]$

Solution (fundamental theorem of calculus):

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau) \, \mathrm{d}\tau$$

Ordinary Differential Equations

Ordinary differential equation :

$$\dot{x}(t) = f(x(t), t), \qquad t \in \mathbb{T} \subset \mathbb{R},$$

where

- $x : \mathbb{T} \to \mathbb{R}^d$ is the unknown function
- $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the *vector field*
- $\mathbb{T}$ is the time domain; typically $\mathbb{T} = [0, T]$

Solution (fundamental theorem of calculus):

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau) \, \mathrm{d}\tau$$

$\Rightarrow$ Solutions depend on the initial value $x(0)$

Ordinary Differential Equations / Initial Value Problems

Ordinary differential equation **initial value problem**:

$$\dot{x}(t) = f(x(t), t), \qquad t \in \mathbb{T} \subset \mathbb{R}, \qquad x(0) = x_0,$$

where

- $x : \mathbb{T} \to \mathbb{R}^d$ is the unknown function
- $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the *vector field*
- $\mathbb{T}$ is the time domain; typically $\mathbb{T} = [0, T]$
- x_0 is the initial value

Solution (fundamental theorem of calculus):

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau)\, \mathrm{d}\tau$$

$\Rightarrow$ Solutions depend on the initial value $x(0)$

https://upload.wikimedia.org/wikipedia/commons/0/04/Pierre_Francois_Verhulst.jpg

- **Logistic ODE:**

$$\dot{P}(t) = rP(t)\left(1 - \frac{P(t)}{K}\right),$$

where P is the population size, r is the growth rate, and K is the carrying capacity (bottleneck).

Pierre-François Verhulst

https://upload.wikimedia.org/wikipedia/commons/0/04/Pierre_Francois_Verhulst.jpg

- **Logistic ODE:**

$$\dot{P}(t) = rP(t)\left(1 - \frac{P(t)}{K}\right),$$

where P is the population size, r is the growth rate, and K is the carrying capacity (bottleneck).

- **Solution:**

$$P(t) = \frac{K}{1 + \frac{K-P(0)}{P(0)} e^{-rt}}.$$

(You can verify for yourself by taking its derivative!)

Pierre-François Verhulst

https://upload.wikimedia.org/wikipedia/commons/0/04/Pierre_Francois_Verhulst.jpg

- **Logistic ODE:**

$$\dot{P}(t) = rP(t)\left(1 - \frac{P(t)}{K}\right),$$

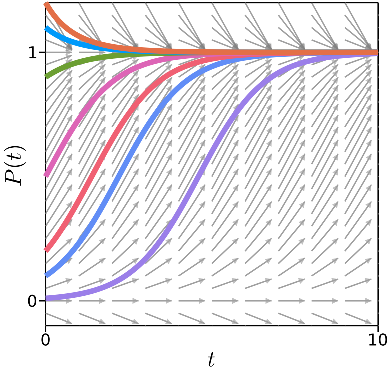
where P is the population size, r is the growth rate, and K is the carrying capacity (bottleneck).

- **Solution:**

$$P(t) = \frac{K}{1 + \frac{K-P(0)}{P(0)} e^{-rt}}.$$

(You can verify for yourself by taking its derivative!)

- **Example** with $K = 1$ and $r = 1$:



- **Logistic ODE:**

$$\dot{P}(t) = rP(t)\left(1 - \frac{P(t)}{K}\right),$$

where P is the population size, r is the growth rate, and K is the carrying capacity (bottleneck).

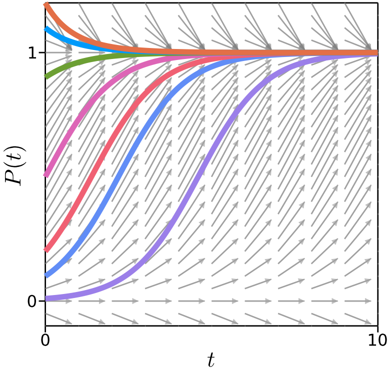
- **Solution:**

$$P(t) = \frac{K}{1 + \frac{K-P(0)}{P(0)} e^{-rt}}.$$

(You can verify for yourself by taking its derivative!)

- **Initial condition:** $P(0) = 0.01$.

- **Example** with $K = 1$ and $r = 1$:



A simple example: Modeling population growth

The logistic ODE

https://upload.wikimedia.org/wikipedia/commons/0/04/Pierre_Francois_Verhulst.jpg

- **Logistic ODE:**

$$\dot{P}(t) = rP(t)\left(1 - \frac{P(t)}{K}\right),$$

  where P is the population size, r is the growth rate, and K is the carrying capacity (bottleneck).

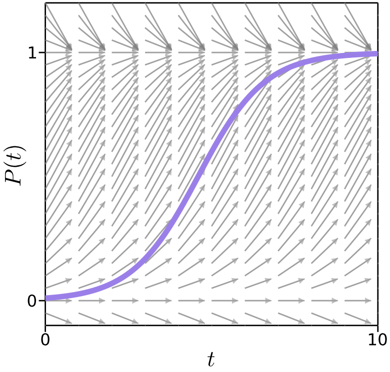
- **Solution:**

$$P(t) = \frac{K}{1 + \frac{K-P(0)}{P(0)} e^{-rt}}.$$

  (You can verify for yourself by taking its derivative!)

- **Initial condition:** $P(0) = 0.01$.

- **Example** with $K = 1$ and $r = 1$:



Next: **How can we solve ODEs in general?**

How to *numerically* solve ODEs — the general case

Is there a way we can solve ODEs in general?

Recall: The initial value problem

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

has the solution

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau) \, \mathrm{d}\tau.$$

How to *numerically* solve ODEs — the general case

Is there a way we can solve ODEs in general?

Recall: The initial value problem

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

has the solution

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Numerical solvers extrapolate step by step: If we know $x(t)$, then $x(t+h)$ is given by

$$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

$\Rightarrow$ At each step, we need to approximate $\int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau$.

How to *numerically* solve ODEs — the general case

Is there a way we can solve ODEs in general?

Recall: The initial value problem

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

has the solution

$$x(t) = x(0) + \int_0^t f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Numerical solvers extrapolate step by step: If we know $x(t)$, then $x(t+h)$ is given by

$$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

$\Rightarrow$ At each step, we need to approximate $\int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau$.
How?

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Recall: $$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

How to *numerically* solve ODEs

Taylor series expansions to the rescue

Recall: $$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, d\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \cdots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!} (\tau - t_0)^n.$$

How to *numerically* solve ODEs

Taylor series expansions to the rescue

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \dots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!}(\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t$:

How to *numerically* solve ODEs

Taylor series expansions to the rescue

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau)\, d\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \cdots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!}(\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t$:

$$f(x(\tau), \tau) \approx f(x(t), t) + \mathcal{O}(h)$$

How to *numerically* solve ODEs

Taylor series expansions to the rescue

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \cdots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!}(\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t$:

$$f(x(\tau), \tau) \approx f(x(t), t) + \mathcal{O}(h)$$

$$\Rightarrow \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau \approx \int_t^{t+h} f(x(t), t) \, \mathrm{d}\tau = h \cdot f(x(t), t).$$

How to *numerically* solve ODEs

Taylor series expansions to the rescue

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \cdots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!} (\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t$:

$$f(x(\tau), \tau) \approx f(x(t), t) + \mathcal{O}(h)$$

$$\Rightarrow \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau \approx \int_t^{t+h} f(x(t), t) \, \mathrm{d}\tau = h \cdot f(x(t), t).$$

(Explicit) Forward Euler: $\hat{x}(t+h) = \hat{x}(t) + h \cdot f(\hat{x}(t), t)$.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \cdots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!} (\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t + h$:

$$f(x(\tau), \tau) \approx f(x(t+h), t+h) + \mathcal{O}(h)$$

$$\Rightarrow \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau \approx \int_t^{t+h} f(x(t+h), t+h) \, \mathrm{d}\tau = h \cdot f(x(t+h), t+h).$$

(Implicit) Backward Euler: $\hat{x}(t+h) = \hat{x}(t) + h \cdot f(\hat{x}(t+h), t+h)$.

How to *numerically* solve ODEs

Taylor series expansions to the rescue

Recall: $$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau)\, \mathrm{d}\tau.$$

Definition (Taylor Series Expansion)

Let $g : \mathbb{R} \to \mathbb{R}$ be a function. Then the *Taylor series expansion* of g at t_0 is given by

$$g(\tau) = g(t_0) + g^{(1)}(t_0)(\tau - t_0) + \frac{1}{2} g^{(2)}(t_0)(\tau - t_0)^2 + \dots = \sum_{n=0}^{\infty} \frac{g^{(n)}(t_0)}{n!}(\tau - t_0)^n.$$

- Zeroth-order Taylor series approximation around $\tau = t + h$:

$$f(x(\tau), \tau) \approx f(x(t+h), t+h) + \mathcal{O}(h)$$

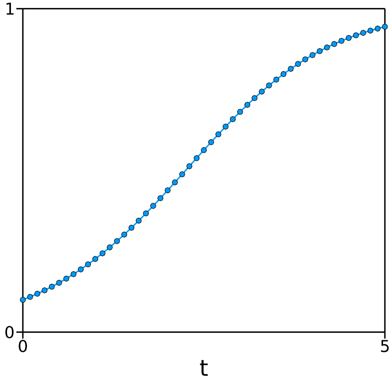
$$\Rightarrow \int_t^{t+h} f(x(\tau), \tau)\, \mathrm{d}\tau \approx \int_t^{t+h} f(x(t+h), t+h)\, \mathrm{d}\tau = h \cdot f(x(t+h), t+h).$$

(Implicit) Backward Euler: $\hat{x}(t+h) = \hat{x}(t) + h \cdot f(\hat{x}(t+h), t+h)$.

Forward Euler in Code

Julia lang is best lang

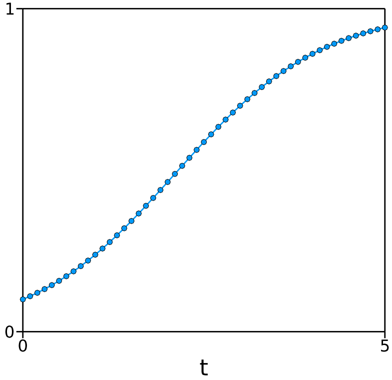
```julia
using Plots

f(x, t) = x * (1 - x)
x0, tspan = 0.1, (0, 5)

h = 1 // 10
x, out = x0, [x0]
for t in tspan[1]:h:(tspan[2]-h)
    x = x + h * f(x, t)
    push!(out, x)
end

plot(tspan[1]:h:tspan[2], out, marker=:o)
```



Backward Euler in Code

Julia lang is best lang

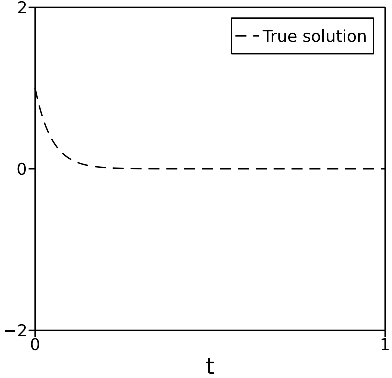

```julia
using Plots, Roots

f(x, t) = x * (1 - x)
x0, tspan = 0.1, (0, 5)

h = 1 // 10
x, out = x0, [x0]
for t in (tspan[1]+h):h:tspan[2]
    x = find_zero(y -> y - (x + h*f(y, t)), x)
    push!(out, x)
end

plot(tspan[1]:h:tspan[2], out, marker=:o)
```



Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ?
(here $\lambda = -21$).

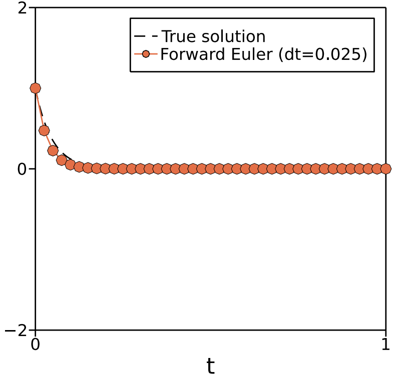


Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ? (here $\lambda = -21$).

- **Forward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t)$
  $\Rightarrow \hat{x}(t+h) = (1 + h\lambda) \cdot \hat{x}(t)$
  $\Rightarrow$ For $\hat{x}(t)$ to remain bounded, we need $|1 + h\lambda| \leq 1$.



Stability: The difference between forward and backward Euler

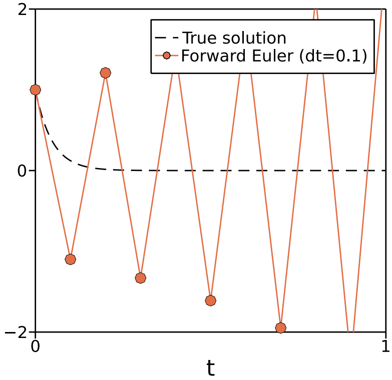
Sometimes explicit methods are just not that great

Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ? (here $\lambda = -21$).

- **Forward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t)$
  $\Rightarrow \hat{x}(t+h) = (1 + h\lambda) \cdot \hat{x}(t)$
  $\Rightarrow$ For $\hat{x}(t)$ to remain bounded, we need $|1 + h\lambda| \leq 1$.



Stability: The difference between forward and backward Euler

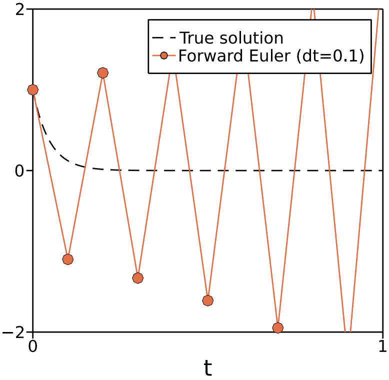
Sometimes explicit methods are just not that great

Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ? (here $\lambda = -21$).

- **Forward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t)$
  $\Rightarrow \hat{x}(t+h) = (1 + h\lambda) \cdot \hat{x}(t)$
  $\Rightarrow$ For $\hat{x}(t)$ to remain bounded, we need $|1 + h\lambda| \leq 1$.
- **Backward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t+h)$
  $\Rightarrow \hat{x}(t+h) = \frac{1}{(1-h\lambda)} \cdot \hat{x}(t)$
  $\Rightarrow \frac{1}{|1-h\lambda|} \leq 1$

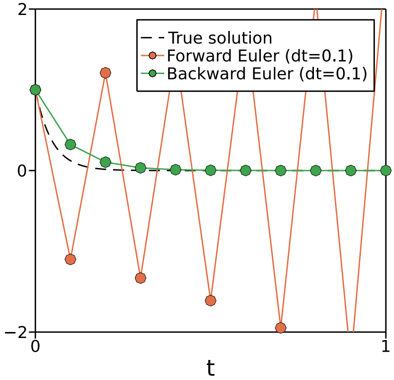


Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ? (here $\lambda = -21$).

- **Forward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t)$
  $\Rightarrow \hat{x}(t+h) = (1 + h\lambda) \cdot \hat{x}(t)$
  $\Rightarrow$ For $\hat{x}(t)$ to remain bounded, we need $|1 + h\lambda| \leq 1$.
- **Backward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t+h)$
  $\Rightarrow \hat{x}(t+h) = \frac{1}{(1-h\lambda)} \cdot \hat{x}(t)$
  $\Rightarrow \frac{1}{|1-h\lambda|} \leq 1$



Stability: The difference between forward and backward Euler

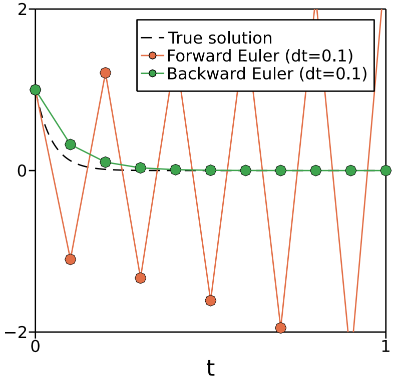
Sometimes explicit methods are just not that great

Consider the following scalar ODE (test equation)

$$\dot{x}(t) = \lambda x(t).$$

How small do we have to make the steps, depending on λ?
(here $\lambda = -21$).

- **Forward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t)$
  $\Rightarrow \hat{x}(t+h) = (1 + h\lambda) \cdot \hat{x}(t)$
  $\Rightarrow$ For $\hat{x}(t)$ to remain bounded, we need $|1 + h\lambda| \leq 1$.
- **Backward Euler:** $\hat{x}(t+h) = \hat{x}(t) + h \cdot \lambda \hat{x}(t+h)$
  $\Rightarrow \hat{x}(t+h) = \frac{1}{(1-h\lambda)} \cdot \hat{x}(t)$
  $\Rightarrow \frac{1}{|1-h\lambda|} \leq 1$



$\Rightarrow$ Different algorithms have different stability properties!

Next: **Runge–Kutta solvers**

Recall: $$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau)\, \mathrm{d}\tau.$$

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Numerical Quadrature

Let $g : \mathbb{R} \to \mathbb{R}^d$ be a function. Then *numerical quadrature* (or *numerical integration*) approximates

$$\int_l^r g(\tau) \, \mathrm{d}\tau \approx \sum_{i=1}^n w_i g(t_i),$$

where t_i are the quadrature nodes and w_i are the quadrature weights.

How to *numerically* solve ODEs — continued

Building better solvers via numerical quadrature

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau)\, \mathrm{d}\tau.$$

Numerical Quadrature

Let $g : \mathbb{R} \to \mathbb{R}^d$ be a function. Then *numerical quadrature* (or *numerical integration*) approximates

$$\int_l^r g(\tau)\, \mathrm{d}\tau \approx \sum_{i=1}^{n} w_i g(t_i),$$

where t_i are the quadrature nodes and w_i are the quadrature weights.

- This motivates (explicit) **Runge–Kutta**:

$$\int_t^{t+h} f(x(\tau), \tau)\, \mathrm{d}\tau \approx h \cdot \sum_{i=1}^{s} w_i f(\hat{x}(\tau_i), \tau_i).$$

How to *numerically* solve ODEs — continued

Building better solvers via numerical quadrature

$$\text{Recall:} \quad x(t+h) = x(t) + \int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau.$$

Numerical Quadrature

Let $g : \mathbb{R} \to \mathbb{R}^d$ be a function. Then *numerical quadrature* (or *numerical integration*) approximates

$$\int_l^r g(\tau) \, \mathrm{d}\tau \approx \sum_{i=1}^n w_i g(t_i),$$

where t_i are the quadrature nodes and w_i are the quadrature weights.

- This motivates (explicit) **Runge–Kutta**:

$$\int_t^{t+h} f(x(\tau), \tau) \, \mathrm{d}\tau \approx h \cdot \sum_{i=1}^s w_i f(\hat{x}(\tau_i), \tau_i).$$

Next: How to choose weights w_i and nodes τ_i? And how to construct $\hat{x}(\tau_i)$?

Runge–Kutta Methods — Definition

Definition ((Explicit) Runge–Kutta method)

An explicit Runge–Kutta method is given by

$$\hat{x}(t+h) = \hat{x}(t) + h \sum_{i=1}^{s} b_i k_i,$$

where

$$\begin{aligned}
k_1 &= f(\hat{x}(t), t), \\
k_2 &= f(\hat{x}(t) + h(a_{21} k_1), t + h c_2), \\
k_3 &= f(\hat{x}(t) + h(a_{31} k_1 + a_{32} k_2), t + h c_3), \\
&\vdots \\
k_s &= f\left(\hat{x}(t) + h \sum_{j=1}^{s-1} a_{sj} k_j, t + h c_s\right).
\end{aligned}$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Definition ((Explicit) Runge–Kutta method)

An explicit Runge–Kutta method is given by

$$\hat{x}(t+h) = \hat{x}(t) + h \sum_{i=1}^{s} b_i k_i,$$

where

$$\begin{aligned} k_1 &= f(\hat{x}(t), t), \\ k_2 &= f(\hat{x}(t) + h(a_{21}k_1), t + hc_2), \\ k_3 &= f(\hat{x}(t) + h(a_{31}k_1 + a_{32}k_2), t + hc_3), \\ &\vdots \\ k_s &= f\left(\hat{x}(t) + h \sum_{j=1}^{s-1} a_{sj}k_j, t + hc_s\right). \end{aligned}$$

"Butcher tableau": A compact representation of *a specific* Runge–Kutta method

$$\begin{array}{c|ccccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Runge–Kutta Methods — Examples

Forward Euler is a Runge–Kutta method!

Runge–Kutta Methods — Examples

Forward Euler is a Runge–Kutta method!

Runge–Kutta in general:

$$\hat{x}(t+h) = \hat{x}(t) + h \sum_{i=1}^{s} b_i k_i, \qquad \text{with} \quad k_i = f\left(\hat{x}(t) + h \sum_{j=1}^{i-1} a_{ij} k_j, t + hc_i\right).$$

Runge–Kutta Methods — Examples

Forward Euler is a Runge–Kutta method!

Runge–Kutta in general:

$$\hat{x}(t+h) = \hat{x}(t) + h \sum_{i=1}^{s} b_i k_i, \qquad \text{with} \quad k_i = f\left(\hat{x}(t) + h \sum_{j=1}^{i-1} a_{ij} k_j, t + h c_i\right).$$

Turns out **forward Euler is actually a Runge–Kutta method:**

$$\begin{aligned} k_1 &= f(\hat{x}(t), t), \\ \hat{x}(t+h) &= \hat{x}(t) + h k_1. \end{aligned}$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Runge–Kutta in general:

$$\hat{x}(t+h) = \hat{x}(t) + h\sum_{i=1}^{s} b_i k_i, \qquad \text{with} \quad k_i = f\left(\hat{x}(t) + h\sum_{j=1}^{i-1} a_{ij}k_j, t + hc_i\right).$$

Turns out **forward Euler is actually a Runge–Kutta method:**

$$\begin{aligned} k_1 &= f(\hat{x}(t+0h), t+0h), \\ \hat{x}(t+h) &= \hat{x}(t) + h \cdot 1 k_1. \end{aligned}$$

Butcher tableau:

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

Runge–Kutta Methods — Examples

Backward Euler is a Runge–Kutta method as well!

Runge–Kutta in general:

$$\hat{x}(t+h) = \hat{x}(t) + h \sum_{i=1}^{s} b_i k_i, \qquad \text{with} \quad k_i = f\left(\hat{x}(t) + h \sum_{j=1}^{i-1} a_{ij} k_j, t + hc_i\right).$$

Turns out **backward Euler is actually a (implicit) Runge–Kutta method:**

$$k_1 = f(\hat{x}(t + 1h), t + 1h),$$
$$\hat{x}(t+h) = \hat{x}(t) + h \cdot 1 k_1.$$

Butcher tableau:

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

(the 1 makes it implicit!)

The **explicit midpoint rule** aims to improve the accuracy of the forward Euler method by selecting

$$\hat{x}(t+h) = \hat{x}(t) + hf\left(\hat{x}\left(t+\frac{h}{2}\right), t+\frac{h}{2}\right).$$

But how to choose $\hat{x}\left(t+\frac{h}{2}\right)$?

Runge–Kutta Methods — Examples

Improving on forward Euler with the *explicit midpoint rule*

The **explicit midpoint rule** aims to improve the accuracy of the forward Euler method by selecting

$$\hat{x}(t+h) = \hat{x}(t) + hf\left(\hat{x}\left(t+\frac{h}{2}\right), t+\frac{h}{2}\right).$$

But how to choose $\hat{x}\left(t+\frac{h}{2}\right)$?
With another Euler step!

Runge–Kutta Methods — Examples

Improving on forward Euler with the *explicit midpoint rule*

The **explicit midpoint rule** aims to improve the accuracy of the forward Euler method by selecting

$$\hat{x}(t+h) = \hat{x}(t) + hf\left(\hat{x}\left(t+\frac{h}{2}\right), t+\frac{h}{2}\right).$$

But how to choose $\hat{x}\left(t+\frac{h}{2}\right)$?
With another Euler step!

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

This leads to the scheme:

$$\begin{aligned} k_1 &= f\left(\hat{x}(t), t+0h\right), \\ k_2 &= f\left(\hat{x}(t) + h \cdot \frac{1}{2}k_1, t+\frac{1}{2}h\right), \\ \hat{x}(t+h) &= \hat{x}(t) + h\left(0k_1 + 1k_2\right). \end{aligned}$$

Runge–Kutta Methods — Examples

The original Runge–Kutta methods of order 4 ($s = 4$)

The **classic fourth-order Runge–Kutta method** selects

$$
\begin{aligned}
k_1 &= f\left(\hat{x}(t), t + 0h\right), \\
k_2 &= f\left(\hat{x}(t) + h \cdot \frac{1}{2} k_1, t + \frac{1}{2} h\right), \\
k_3 &= f\left(\hat{x}(t) + h \cdot \frac{1}{2} k_2, t + \frac{1}{2} h\right), \\
k_4 &= f\left(\hat{x}(t) + h \cdot 1 k_3, t + 1h\right),
\end{aligned}
$$

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

and then

$$
\hat{x}(t + h) = \hat{x}(t) + h\left(\frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4\right).
$$

(Further reading: "Solving Ordinary Differential Equations I" by Hairer, Norsett and Wanner, Chapter II.1; includes derivations for the coefficients!)

The Dormand–Prince method

This is one of the most used ODE solvers of all time! A.k.a. `DOPRI5` in SciPy, `DP5` in Julia, or `ode45` in MATLAB

The Dormand–Prince method

This is one of the most used ODE solvers of all time! A.k.a. `DOPRI5` in SciPy, `DP5` in Julia, or `ode45` in MATLAB

`DOPRI5` has a much more complicated Butcher tableau:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
\frac{1}{2} & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & 0 & \frac{11}{84} \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

The Dormand–Prince method

This is one of the most used ODE solvers of all time! A.k.a. **DOPRI5** in SciPy, **DP5** in Julia, or **ode45** in MATLAB

DOPRI5 has a much more complicated Butcher tableau:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
\frac{1}{2} & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & 0 & \frac{11}{84} \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

This is the reason for the SciPy code:

```
      C2=0.2D0
      C3=0.3D0
      C4=0.8D0
      C5=8.D0/9.D0
      A21=0.2D0
      A31=3.D0/40.D0
      A32=9.D0/40.D0
      A41=44.D0/45.D0
      A42=-56.D0/15.D0
      A43=32.D0/9.D0
      A51=19372.D0/6561.D0
      A52=-25360.D0/2187.D0
      A53=64448.D0/6561.D0
      A54=-212.D0/729.D0
      A61=9017.D0/3168.D0
      A62=-355.D0/33.D0
      A63=46732.D0/5247.D0
      A64=49.D0/176.D0
      A65=-5103.D0/18656.D0
      A71=35.D0/384.D0
      A73=500.D0/1113.D0
      A74=125.D0/192.D0
      A75=-2187.D0/6784.D0
      A76=11.D0/84.D0
      E1=71.D0/57600.D0
      E3=-71.D0/16695.D0
      E4=71.D0/1920.D0
      E5=-17253.D0/339200.D0
      E6=22.D0/525.D0
      E7=-1.D0/40.D0
```

`https://github.com/scipy/scipy/blob/main/scipy/integrate/dop/dopri5.f`

The Dormand–Prince method

This is one of the most used ODE solvers of all time! A.k.a. **DOPRI5** in SciPy, **DP5** in Julia, or **ode45** in MATLAB

DOPRI5 has a much more complicated Butcher tableau:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
\frac{1}{2} & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & 0 & \frac{11}{84} \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

This is the reason for the SciPy code:

```
      C2=0.2D0
      C3=0.3D0
      C4=0.8D0
      C5=8.D0/9.D0
      A21=0.2D0
      A31=3.D0/40.D0
      A32=9.D0/40.D0
      A41=44.D0/45.D0
      A42=-56.D0/15.D0
      A43=32.D0/9.D0
      A51=19372.D0/6561.D0
      A52=-25360.D0/2187.D0
      A53=64448.D0/6561.D0
      A54=-212.D0/729.D0
      A61=9017.D0/3168.D0
      A62=-355.D0/33.D0
      A63=46732.D0/5247.D0
      A64=49.D0/176.D0
      A65=-5103.D0/18656.D0
      A71=35.D0/384.D0
      A73=500.D0/1113.D0
      A74=125.D0/192.D0
      A75=-2187.D0/6784.D0
      A76=11.D0/84.D0
      E1=71.D0/57600.D0
      E3=-71.D0/16695.D0
      E4=71.D0/1920.D0
      E5=-17253.D0/339200.D0
      E6=22.D0/525.D0
      E7=-1.D0/40.D0
```

`https://github.com/scipy/scipy/blob/main/scipy/integrate/dop/dopri5.f`

(the two bottom lines are there because of "error estimation" which is not covered in this lecture; if interested, check out Chapter II.4 in "Solving Ordinary Differential Equations I" by Hairer et al.)

Intermediate summary on classical numerical ODE solvers:

- There are A LOT of numerical ODE solvers!

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.
    - The Dormand–Prince method: $p = 5$.

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.
    - The Dormand–Prince method: $p = 5$.
- There is a lot of stuff happening under the hood when calling `scipy.integrate.ode` or similar:

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.
    - The Dormand–Prince method: $p = 5$.
- There is a lot of stuff happening under the hood when calling `scipy.integrate.ode` or similar:
  - *Step-size selection:* Discretize on the fly instead of using a fixed step size. (exercise sheet)

Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.
    - The Dormand–Prince method: $p = 5$.
- There is a lot of stuff happening under the hood when calling `scipy.integrate.ode` or similar:
  - *Step-size selection:* Discretize on the fly instead of using a fixed step size. (exercise sheet)
  - (sometimes) *Automatic solver selection:* Use heuristics to decide which solver to use.

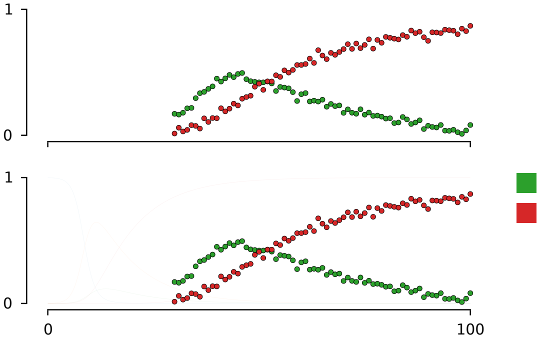
Intermediate summary on classical numerical ODE solvers:

- **There are A LOT of numerical ODE solvers!** Differences between solvers include:
  - *Stability*: Explicit vs implicit methods.
  - *Order* and convergence rates:
    A Runge–Kutta method has *order p* if the local truncation error is of order $O(h^{p+1})$. Examples:
    - Forward Euler: $p = 1$ (exercise sheet)
    - Explicit midpoint method: $p = 2$
    - The classical fourth-order Runge–Kutta method: $p = 4$.
    - The Dormand–Prince method: $p = 5$.
- There is a lot of stuff happening under the hood when calling `scipy.integrate.ode` or similar:
  - *Step-size selection:* Discretize on the fly instead of using a fixed step size. (exercise sheet)
  - (sometimes) *Automatic solver selection:* Use heuristics to decide which solver to use.

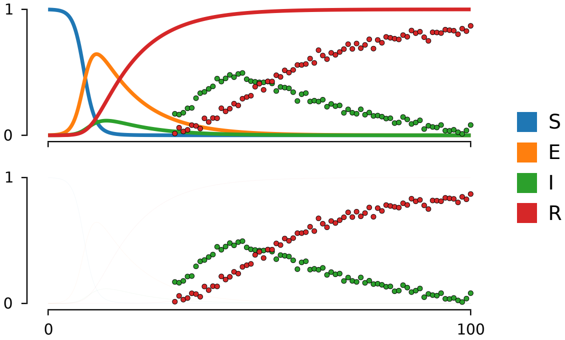
Next block: **What if we don't know *f* but instead have to estimate it from data?**

Parameter Inference

Learning unknown dynamics from data.



Tronarp, Bosch, Hennig, “Fenrir: Physics-Enhanced Regression for Initial Value Problems”, ICML 2022



Tronarp, Bosch, Hennig, “Fenrir: Physics-Enhanced Regression for Initial Value Problems”, ICML 2022

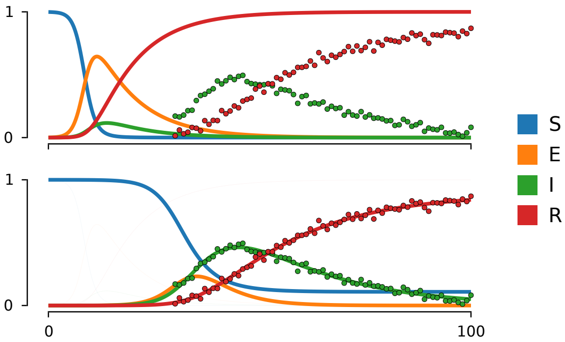
- **Typical goal:** “Fit the data”.
- **Parameter inference:** Learn the parameters of a *Mechanistic model*, e.g. here the SEIR model

$$\dot{S} = -\beta_E SE/N$$
$$\dot{E} = \beta_E SE/N - \gamma E$$
$$\dot{I} = \gamma E - \lambda I$$
$$\dot{R} = \lambda I$$

such that the solution fits the data.

Parameter Inference

Learning unknown dynamics from data.



Tronarp, Bosch, Hennig, “Fenrir: Physics-Enhanced Regression for Initial Value Problems”, ICML 2022

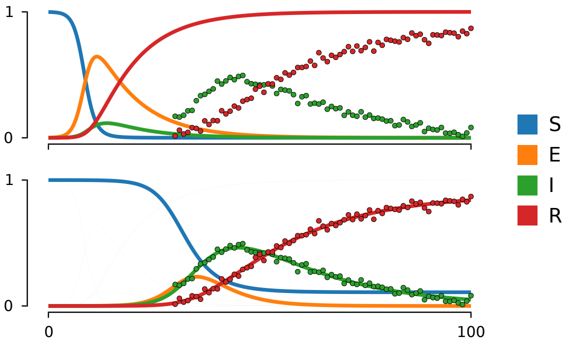
- **Typical goal:** “Fit the data”.
- **Parameter inference:** Learn the parameters of a *Mechanistic model*, e.g. here the SEIR model

$$\dot{S} = -\beta_E SE/N$$
$$\dot{E} = \beta_E SE/N - \gamma E$$
$$\dot{I} = \gamma E - \lambda I$$
$$\dot{R} = \lambda I$$

such that the solution fits the data.
$\Rightarrow$ *Provides interpretable results*

Parameter Inference

Learning unknown dynamics from data.



Tronarp, Bosch, Hennig, "Fenrir: Physics-Enhanced Regression for Initial Value Problems", ICML 2022

- **Typical goal:** "Fit the data".
- **Parameter inference:** Learn the parameters of a *Mechanistic model*, e.g. here the SEIR model

$$\dot{S} = -\beta_E SE/N$$
$$\dot{E} = \beta_E SE/N - \gamma E$$
$$\dot{I} = \gamma E - \lambda I$$
$$\dot{R} = \lambda I$$

such that the solution fits the data.
$\Rightarrow$ *Provides interpretable results*

Up to this point *f* was always given – *now it needs to be estimated!*

Parameter Inference

Learning unknown dynamics from data.

Setup: Consider an initial value problem

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

where the parameters $\theta \in \mathbb{R}^d$ are unknown.

Parameter Inference

Learning unknown dynamics from data.

Setup: Consider an initial value problem

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

where the parameters $\theta \in \mathbb{R}^d$ are unknown.
Assume noisy observations $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where

$$y_i = H \cdot x(t_i) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \Sigma).$$

Parameter Inference

Learning unknown dynamics from data.

Setup: Consider an initial value problem

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

where the parameters $\theta \in \mathbb{R}^d$ are unknown.
Assume noisy observations $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where

$$y_i = H \cdot x(t_i) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \Sigma).$$

Goal: Estimate θ from $\mathcal{D}$:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta) p(\theta)}{p(\mathcal{D})}.$$

Parameter Inference

Learning unknown dynamics from data.

Setup: Consider an initial value problem

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

where the parameters $\theta \in \mathbb{R}^d$ are unknown.
Assume noisy observations $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where

$$y_i = H \cdot x(t_i) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \Sigma).$$

Goal: Estimate θ from $\mathcal{D}$:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta) p(\theta)}{p(\mathcal{D})}.$$

Cheaper goal: Compute the *maximum-likelihood estimate*

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta} p(\mathcal{D} \mid \theta).$$

Parameter Inference with numerical ODE solvers

Assuming i.i.d. data, the likelihood is given by

$$p(\mathcal{D} \mid \theta) = \prod_{i=1}^{n} \mathcal{N}(y_i; Hx_\theta(t_i), \Sigma).$$

Issue: The likelihood is intractable, since it depends on the *true* solution $x_\theta(t)$ of the ODE.

Assuming i.i.d. data, the likelihood is given by

$$p(\mathcal{D} \mid \theta) = \prod_{i=1}^{n} \mathcal{N}(y_i; Hx_\theta(t_i), \Sigma).$$

Issue: The likelihood is intractable, since it depends on the *true* solution $x_\theta(t)$ of the ODE.
Solution: Use a numerical ODE solver to approximate the solution:

$$x_\theta(t) \approx \hat{x}_\theta(t),$$

where $\hat{x}_\theta(t)$ is the numerical solution of the ODE with parameters θ.

Assuming i.i.d. data, the likelihood is given by

$$p(\mathcal{D} \mid \theta) = \prod_{i=1}^{n} \mathcal{N}(y_i; Hx_\theta(t_i), \Sigma).$$

Issue: The likelihood is intractable, since it depends on the *true* solution $x_\theta(t)$ of the ODE.
Solution: Use a numerical ODE solver to approximate the solution:

$$x_\theta(t) \approx \hat{x}_\theta(t),$$

where $\hat{x}_\theta(t)$ is the numerical solution of the ODE with parameters θ.
Then

$$p(\mathcal{D} \mid \theta) \approx \prod_{i=1}^{n} \mathcal{N}(y_i; H\hat{x}_\theta(t_i), \Sigma),$$

which is tractable.

Parameter Inference with numerical ODE solvers

Assuming i.i.d. data, the likelihood is given by

$$p(\mathcal{D} \mid \theta) = \prod_{i=1}^{n} \mathcal{N}(y_i; Hx_\theta(t_i), \Sigma).$$

Issue: The likelihood is intractable, since it depends on the *true* solution $x_\theta(t)$ of the ODE.
Solution: Use a numerical ODE solver to approximate the solution:

$$x_\theta(t) \approx \hat{x}_\theta(t),$$

where $\hat{x}_\theta(t)$ is the numerical solution of the ODE with parameters θ.
Then

$$p(\mathcal{D} \mid \theta) \approx \prod_{i=1}^{n} \mathcal{N}(y_i; H\hat{x}_\theta(t_i), \Sigma),$$

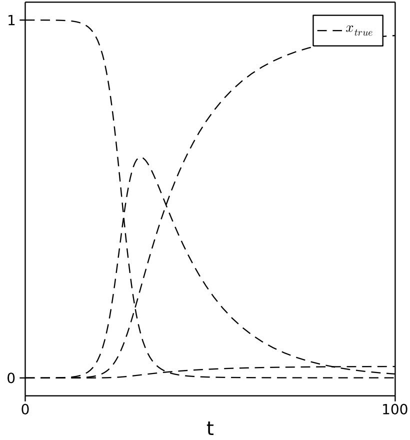
which is tractable.
Maximizing the likelihood is equivalent to minimizing the *negative log-likelihood*:

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{n} (H\hat{x}_\theta(t_i) - y_i)^T \Sigma^{-1} (H\hat{x}_\theta(t_i) - y_i).$$

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.



Parameter Inference with numerical ODE solvers

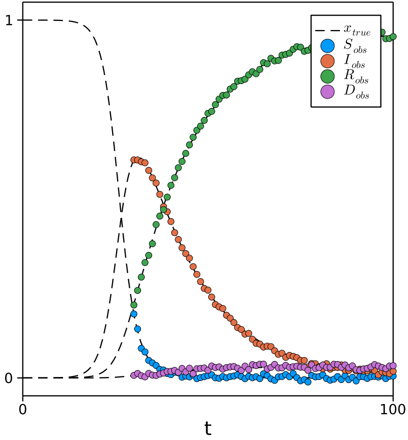
Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.



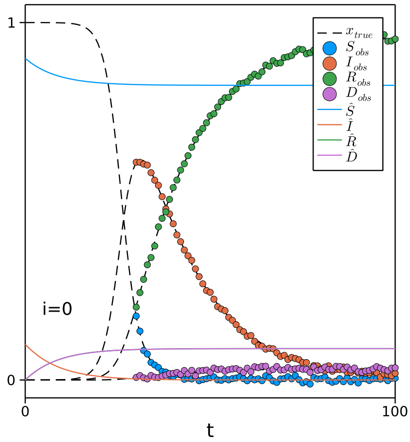
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.

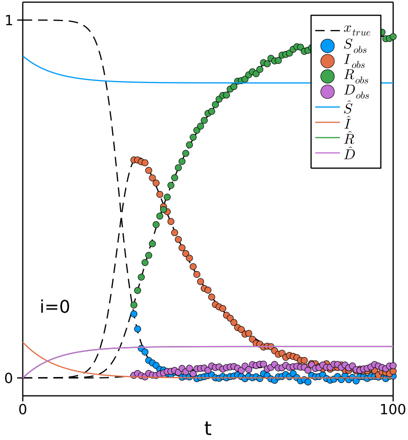


- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.

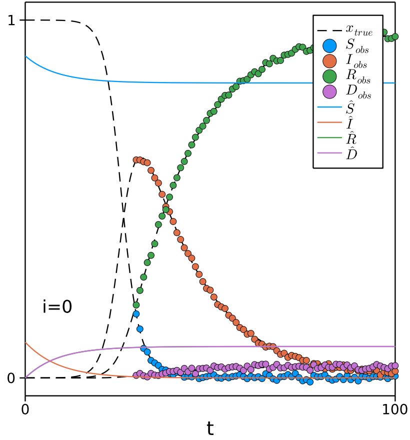


- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)

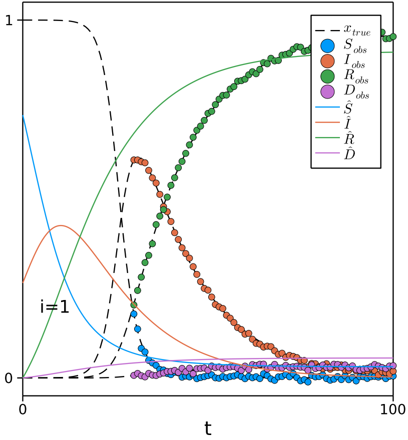


- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



Parameter Inference with numerical ODE solvers

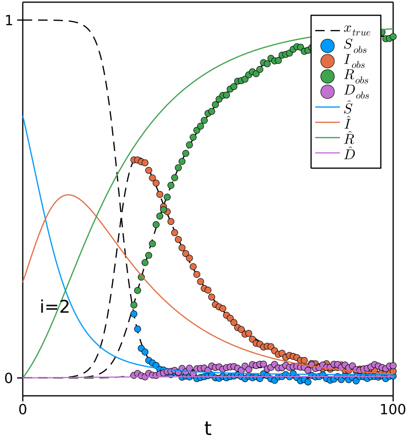
Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



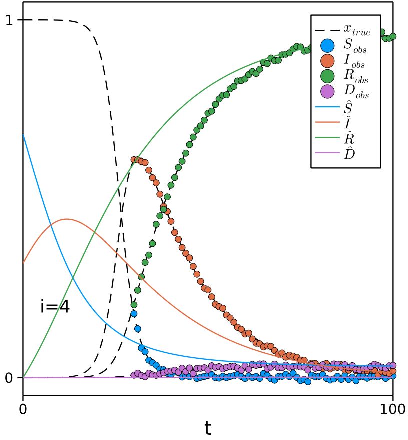
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



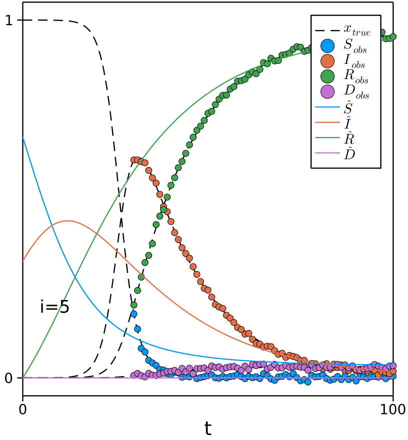
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



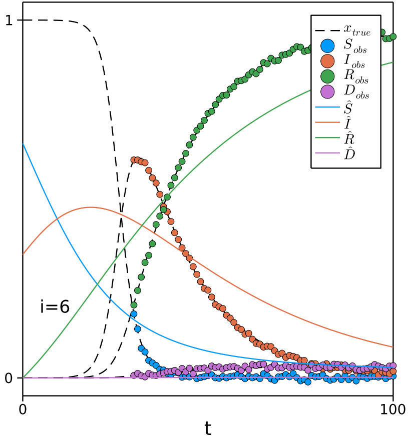
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



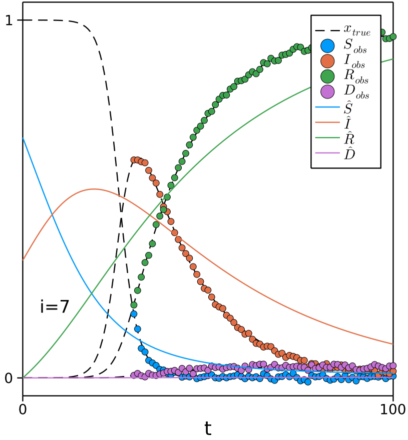
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)

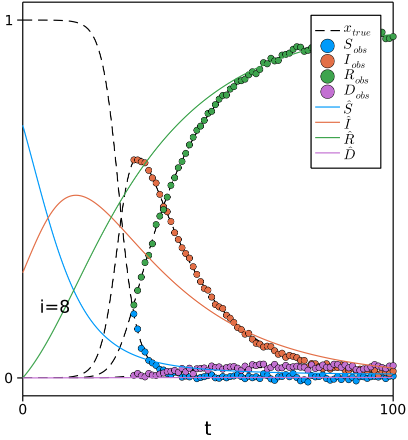


- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



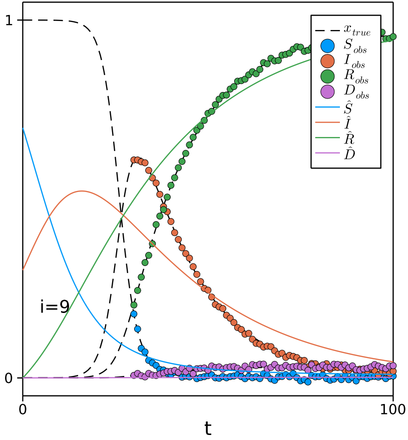
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

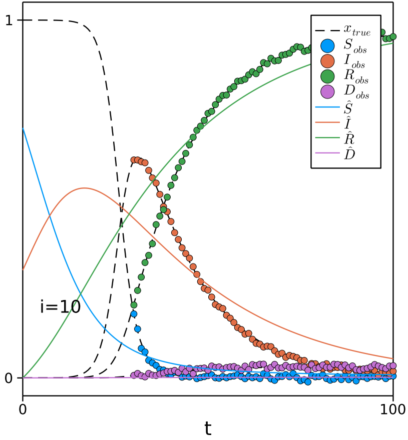
  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^{n}$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^{n} \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

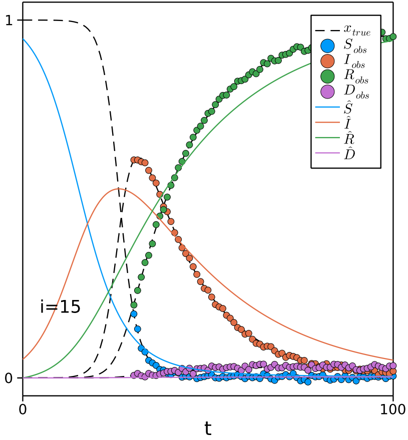
  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



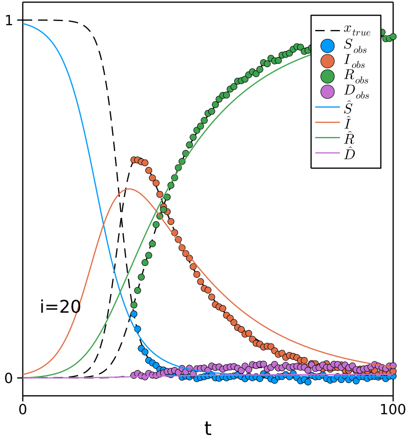
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



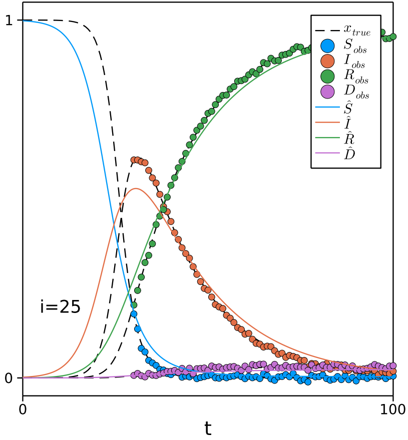
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

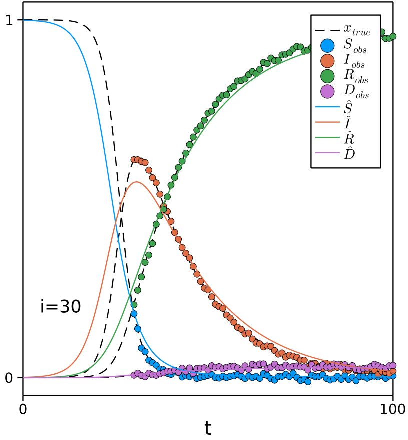
for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^{n}$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^{n} \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)

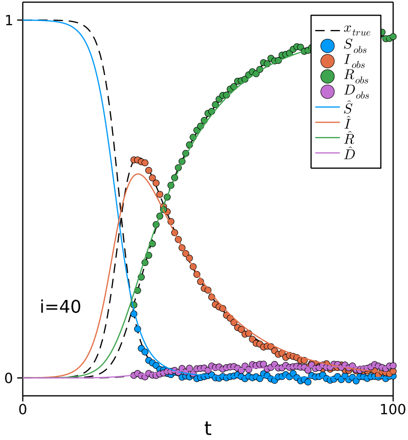


- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



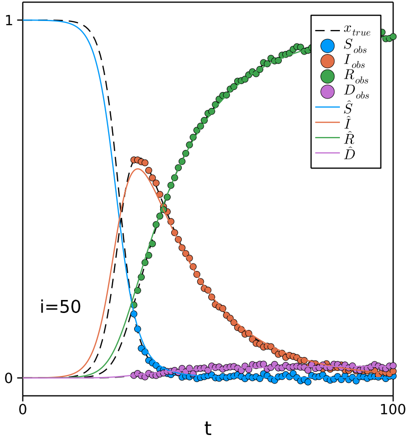
Parameter Inference with numerical ODE solvers

Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta S I \quad \dot{I} = \beta S I - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

  for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.
- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)



Parameter Inference with numerical ODE solvers

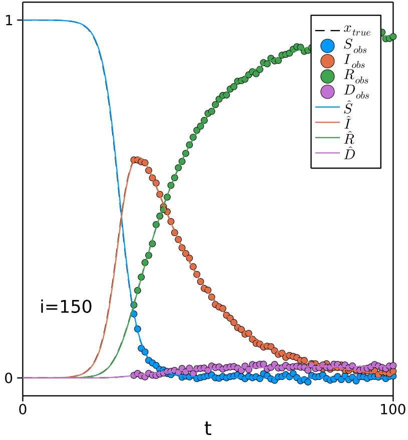
Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)
- **Result:** $\hat{\theta}_{\text{ML}} = (9.46 \cdot 10^{-6}, 0.5014, 0.0603, 0.0020)$. (took $\sim$ 3.5 seconds)



Parameter Inference with numerical ODE solvers

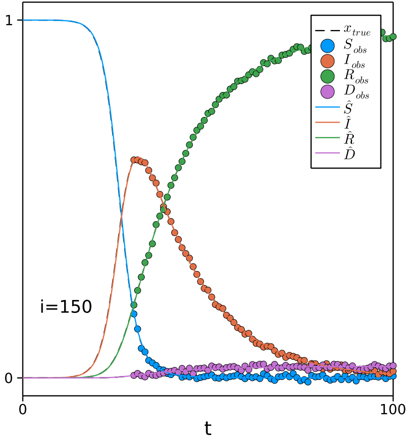
Example: Simulating epidemics with the SIRD model

- **Dynamics:** (simplified) SIRD model

$$\dot{S} = -\beta SI \quad \dot{I} = \beta SI - \gamma I - \eta I \quad \dot{R} = \gamma I \quad \dot{D} = \eta I$$

for $t \in [0, 100]$, with $SIRD(0) = [1 - I0, I0, 0, 0]$ and four unknown parameters $I_0, \beta, \gamma, \eta \in \mathbb{R}$.

- **Data:** $\mathcal{D} = \{(y_i, t_i)\}_{i=1}^n$, where $y_i \sim \mathcal{N}(x(t_i), 0.1 \cdot I)$; generated with $\theta^* = (10^{-5}, 0.5, 0.06, 0.002)$.
- **Loss** (as in last slide): $L(\theta) = \frac{1}{2} 0.1 \sum_{i=1}^n \|\hat{x}_\theta(t_i) - y_i\|_2^2$.
- **Initial guess:** $\theta_0 = (0.1, 0.1, 0.1, 0.1)$.
- **Optimize:** with the optimizer of your choice (e.g. L-BFGS)
- **Result:** $\hat{\theta}_{\text{ML}} = (9.46 \cdot 10^{-6}, 0.5014, 0.0603, 0.0020)$. (took $\sim$ 3.5 seconds)



We can learn system parameters from data via (local) optimization!

ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

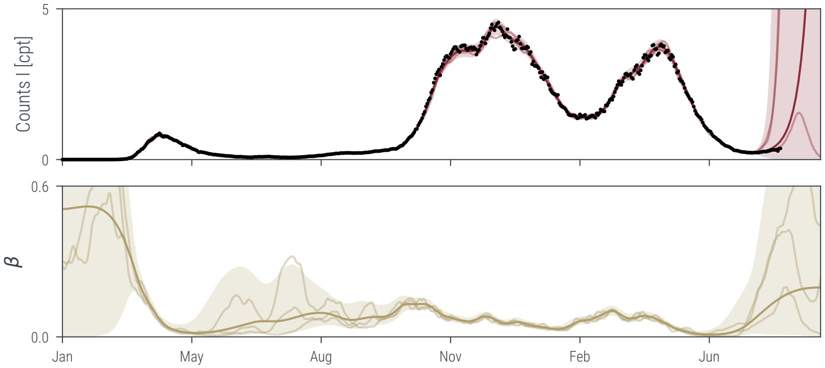
$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany.

ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany. **Related result** as shown in lecture 1:



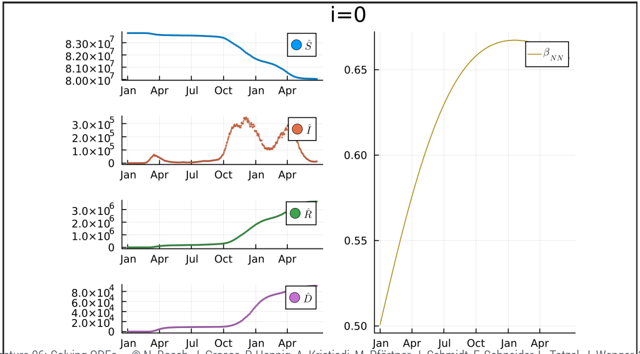
ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany.

Idea today: Just model $\beta(t)$ with a neural network $\beta_\theta^{\mathrm{NN}}$, and do parameter inference on θ as before!

Result:



Parameter Inference *on real COVID data*

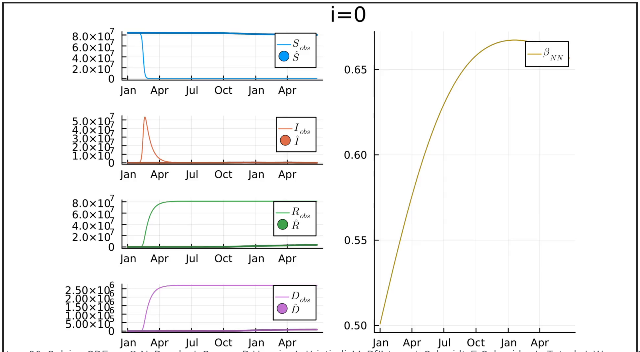
You saw this example in the first lecture - so let's revisit it!

Figure from: Schmidt, Krämer, Hennig, NeurIPS2021

ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany.
Idea today: Just model $\beta(t)$ with a neural network β_θ^{NN}, and do parameter inference on θ as before!
Result:



Parameter Inference *on real COVID data*

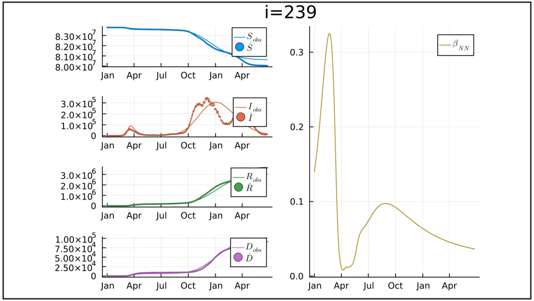
You saw this example in the first lecture - so let's revisit it!

Figure from: Schmidt, Krämer, Hennig, NeurIPS2021

ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany.
Idea today: Just model $\beta(t)$ with a neural network $\beta_\theta^{\mathrm{NN}}$, and do parameter inference on θ as before!
Result:



Disclaimer: I only had limited time and it might very well be possible to do this much better!

Parameter Inference *on real COVID data*

You saw this example in the first lecture - so let's revisit it!

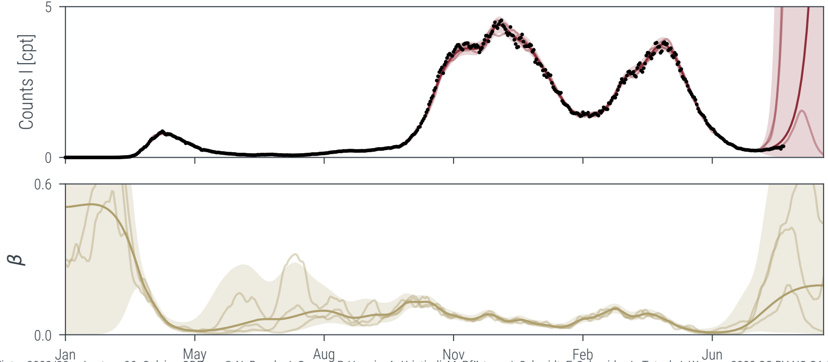
Figure from: Schmidt, Krämer, Hennig, NeurIPS2021

ODE dynamics as before, but this time with time-varying contact rate $\beta(t)$:

$$\dot{S} = -\beta(t)SI/N, \qquad \dot{I} = \beta(t)SI/N - \gamma I - \eta I, \qquad \dot{R} = \gamma I, \qquad \dot{D} = \eta I.$$

Data are the real COVID counts from Germany.

Next week: $\beta(t) \sim \mathcal{GP}$!



Summary

- ODEs play an important role in machine learning.
- In general, solving an ODE requires a *numerical* solver, e.g. Euler or Runge–Kutta
- … of which there are many! With different properties (stability, order, …).
- We can *learn* ODE parameters via (local) optimization, even neural networks!

Please cite this course, as

```
@techreport{NoML22,
    title = {Numerics of Machine Learning},
    author = {N. Bosch and J. Grosse
    and P. Hennig and A. Kristiadi
    and M. Pförtner and J. Schmidt
    and F. Schneider and L. Tatzel
    and J. Wenger},
    series = {Lecture Notes in Machine Learning},
    year = {2022},
    institution = {Tübingen AI Center},
}
```

Summary

- ODEs play an important role in machine learning.
- In general, solving an ODE requires a *numerical* solver, e.g. Euler or Runge–Kutta
- ... of which there are many! With different properties (stability, order, ...).
- We can *learn* ODE parameters via (local) optimization, even neural networks!

Next week: ***Probabilistic* numerical ODE solvers!**
Combining ODEs and Bayesian state estimation.

Please cite this course, as

```
@techreport{NoML22,
    title = {Numerics of Machine Learning},
    author = {N. Bosch and J. Grosse
    and P. Hennig and A. Kristiadi
    and M. Pförtner and J. Schmidt
    and F. Schneider and L. Tatzel
    and J. Wenger},
    series = {Lecture Notes in Machine Learning},
    year = {2022},
    institution = {Tübingen AI Center},
}
```