A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

Nathanael Bosch

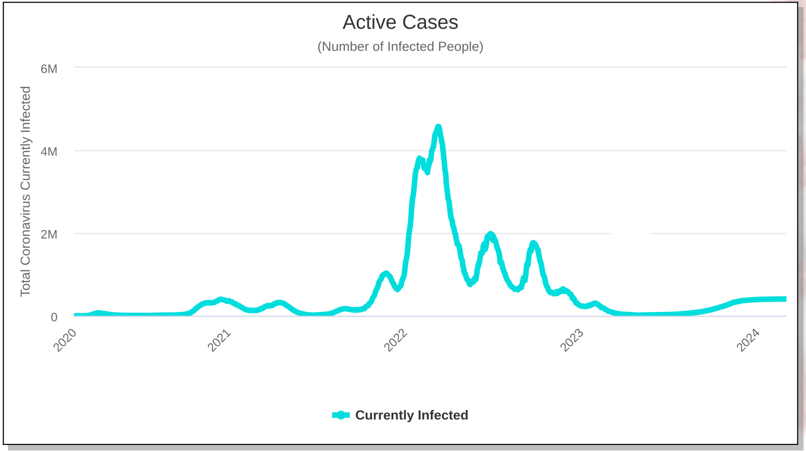
26. February 2025

Eberhard Karls Universität Tübingen

imprs-is

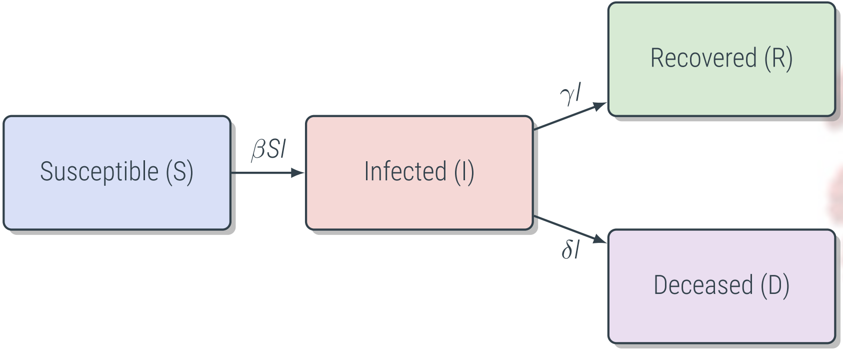
erc some of the presented work is supported by the European Research Council.

The COVID-19 pandemic — A real-world dynamical system



https://worldometers.info/coronavirus/country/germany/

SIRD – A simple model for infectious diseases



The SIRD model as an ordinary differential equation

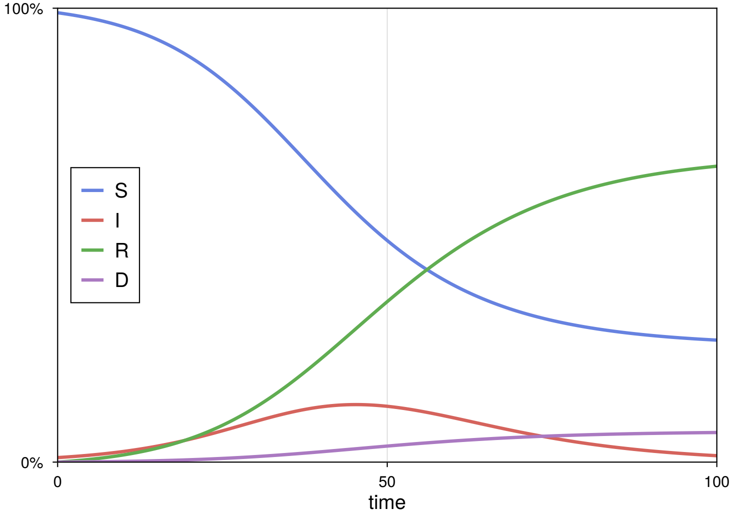
$$\dot{S}(t) = -\beta\, S(t)\, I(t)$$

$$\dot{I}(t) = \beta\, S(t)\, I(t) - \gamma\, I(t) - \delta\, I(t)$$

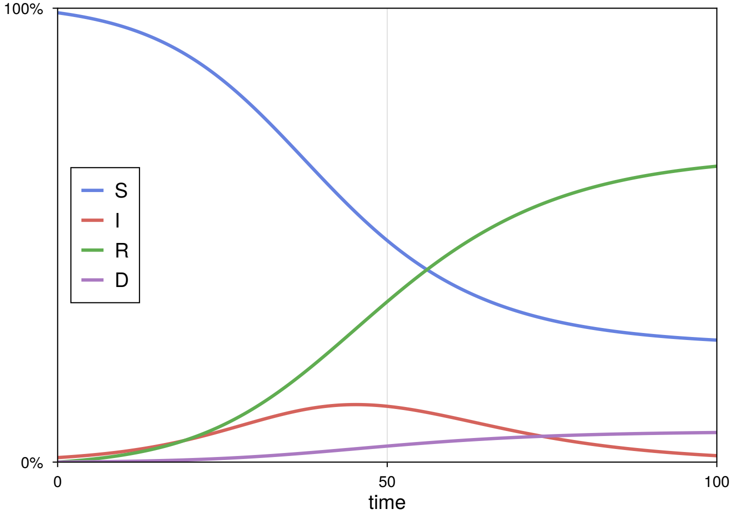
$$\dot{R}(t) = \gamma\, I(t)$$

$$\dot{D}(t) = \delta\, I(t)$$

Numerical simulation of the SIRD model

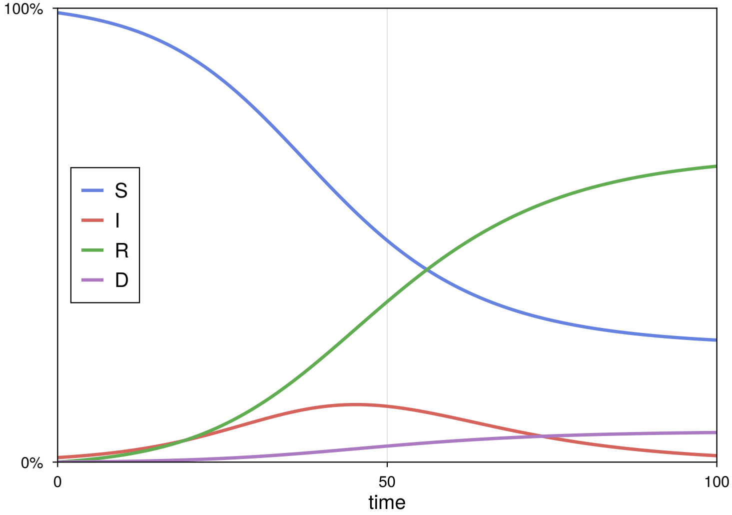


Numerical simulation of the SIRD model



How do we simulate dynamical systems?

Numerical simulation of the SIRD model



How do we simulate dynamical systems?

How accurate is the simulation?

Can we trust it?

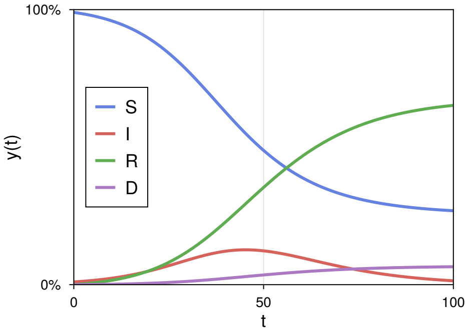
How to simulate ordinary differential equations

Ordinary Differential Equations and traditional simulators

Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".



Ordinary Differential Equations and traditional simulators

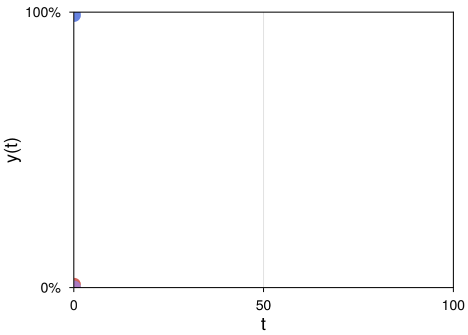
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

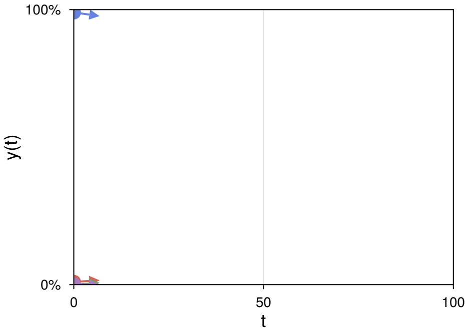


$$\dot{y}(t) = f\left(y(t), t\right), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

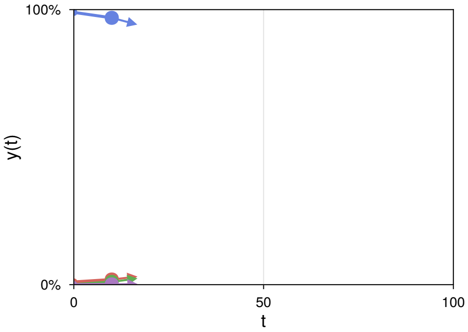


$$\dot{y}(t) = f\left(y(t), t\right), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

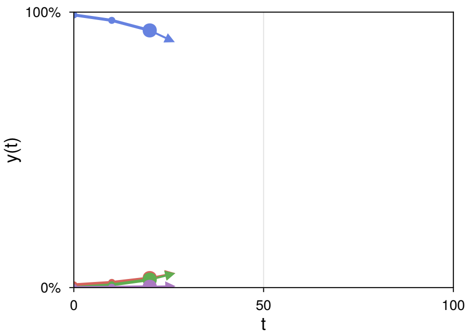
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

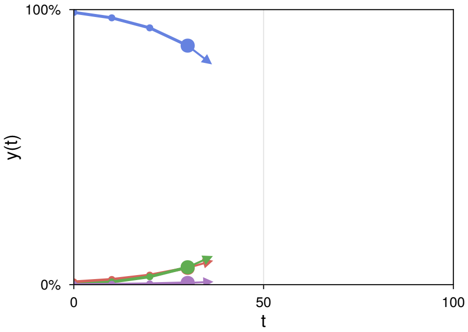


$$\dot{y}(t) = f\left(y(t), t\right), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

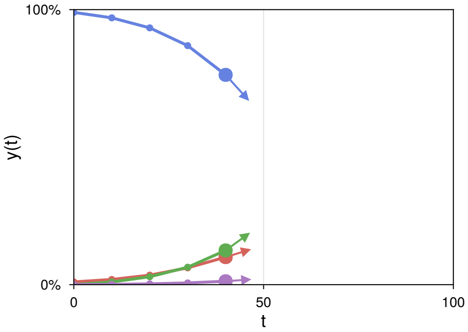


$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

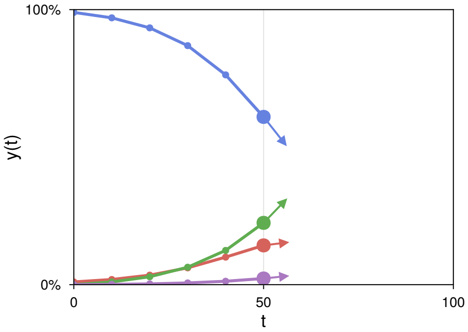
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

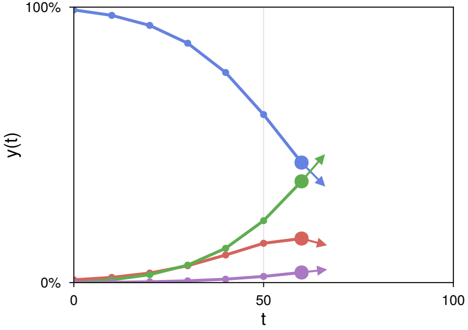


$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t + h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

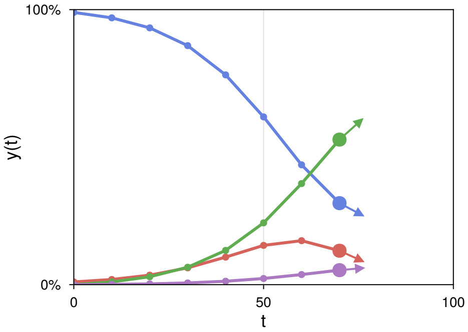
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t + h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

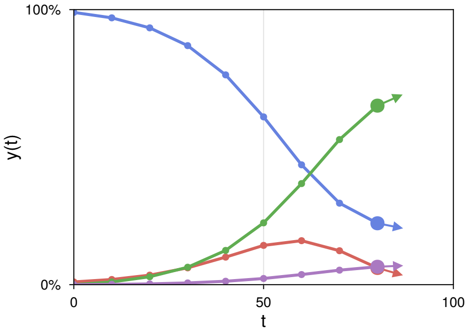


$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t + h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

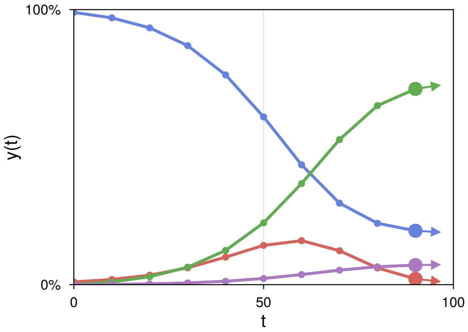
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

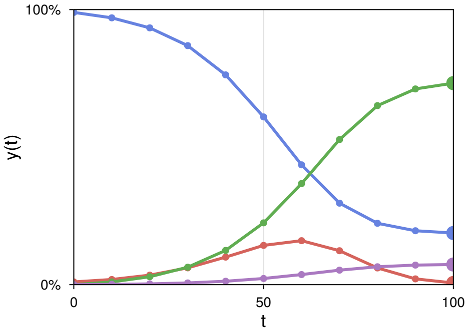
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

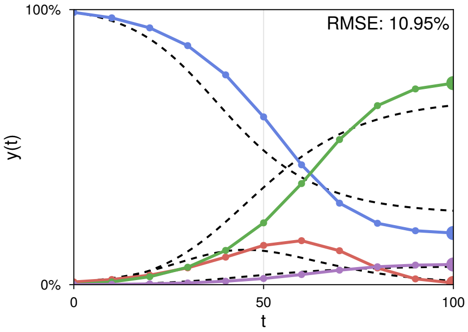
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



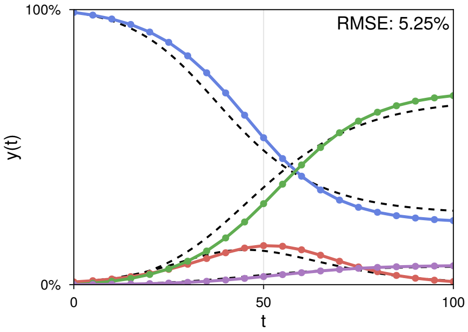
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

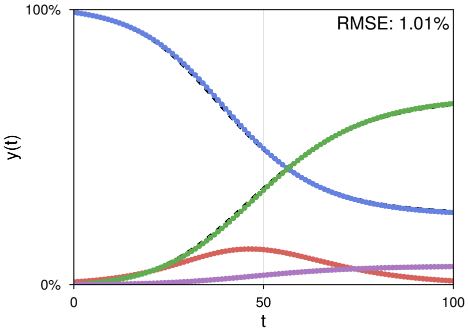
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f\left(y(t), t\right), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: “Find y”.

A simple numerical ODE solver: “Forward Euler”

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Ordinary Differential Equations and traditional simulators

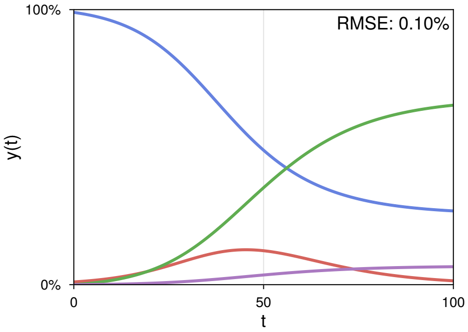
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f\left(y(t), t\right), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$



Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

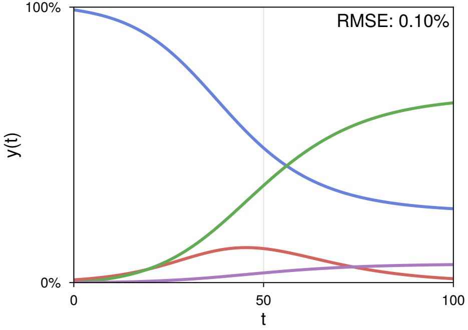
$$\dot{y}(t) = f(y(t), t), \qquad y(0) = y_0.$$

with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t + h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

The simulation $\hat{y}$ is only an *estimate* of y.
The error depends on the solver and step size.



Ordinary Differential Equations and traditional simulators

Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

$$\dot{y}(t) = f\,(y(t), t)\,, \qquad y(0) = y_0.$$

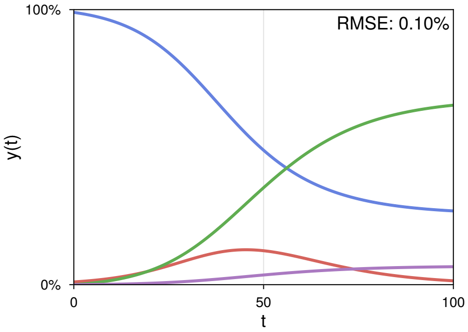
with $t \in [0, T]$, vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and initial value $y_0 \in \mathbb{R}^d$. Goal: "Find y".

A simple numerical ODE solver: "Forward Euler"

$$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t).$$

The simulation $\hat{y}$ is only an *estimate* of y.
The error depends on the solver and step size.

Traditional simulators do not quantify their *estimation error*.



$$p\left(y(t) \,\middle|\, y(0) = y_0, \{\dot{y}(t_n) = f(y(t_n), t_n)\}_{n=1}^{N}\right)$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, initial value y_0, and time discretization $\{t_n\}_{n=1}^{N}$.

$$p\left(y(t) \,\middle|\, y(0) = y_0, \{\dot{y}(t_n) = f(y(t_n), t_n)\}_{n=1}^{N}\right)$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, initial value y_0, and time discretization $\{t_n\}_{n=1}^{N}$.

Prior

$$p\left(y(t) \,\middle|\, y(0) = y_0, \{\dot{y}(t_n) = f(y(t_n), t_n)\}_{n=1}^{N}\right)$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, initial value y_0, and time discretization $\{t_n\}_{n=1}^{N}$.

Prior + **Likelihood & Data**

Probabilistic numerical ODE solvers

or *How to treat ODEs as the state estimation problem that they really are*

$$p\left(y(t) \,\middle|\, y(0) = y_0, \{\dot{y}(t_n) = f(y(t_n), t_n)\}_{n=1}^{N}\right)$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, initial value y_0, and time discretization $\{t_n\}_{n=1}^{N}$.

Prior + **Likelihood & Data** + **Inference**

Building blocks of probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Prior

Likelihood & Data

Inference

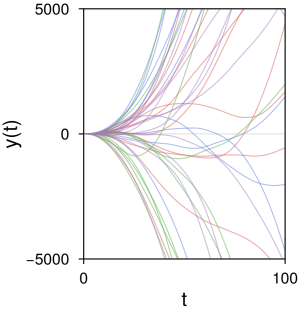
Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

Likelihood & Data

Inference



Building blocks of probabilistic numerical ODE solvers

Prior

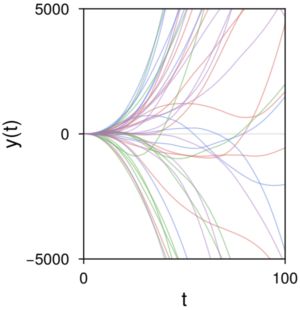
$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

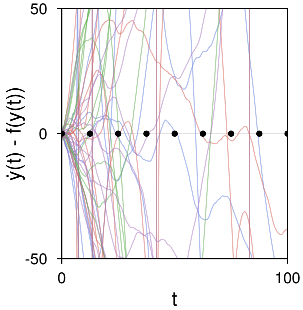
Likelihood & Data

$$z(t) = \dot{y}(t) - f(y(t), t)$$

$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$$

Inference





Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

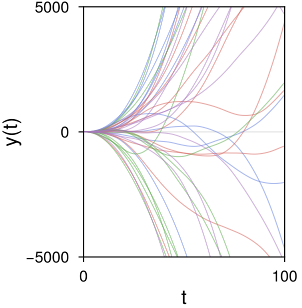
Likelihood & Data

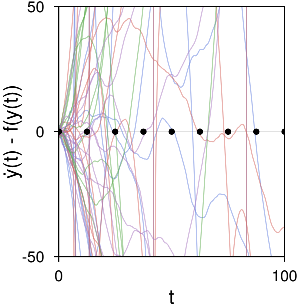
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

```
Initial distribution p(y(t_0))
for i = 1:N do
    Predict:
        p_f(y(t_{i-1})) ↦ p_p(y(t_i))
    Linearize f at E_{p_p}[y(t_i)]
    Update:
        p_p(y(t_i)), z(t_i) ↦ p_f(y(t_i))
end for
```

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

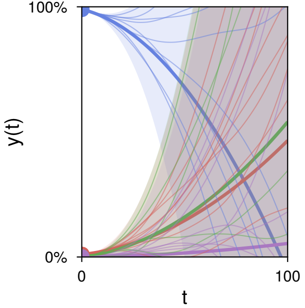
Likelihood & Data

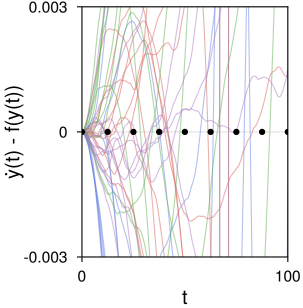
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a
Gauss–Markov process

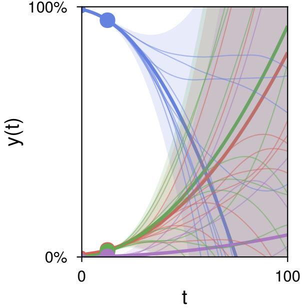
Likelihood & Data

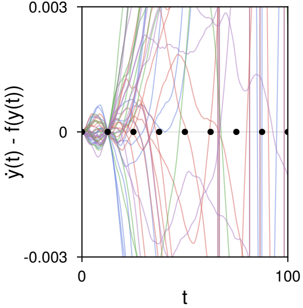
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering
and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

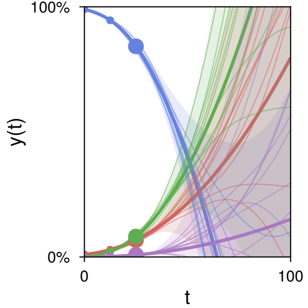
Likelihood & Data

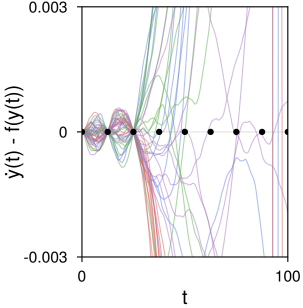
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1{:}N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1{:}N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

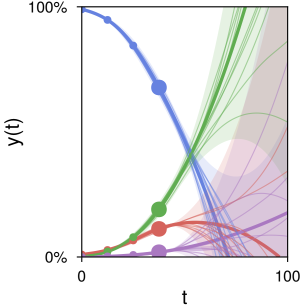
Likelihood & Data

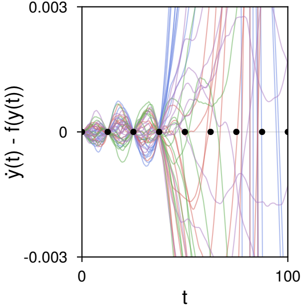
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

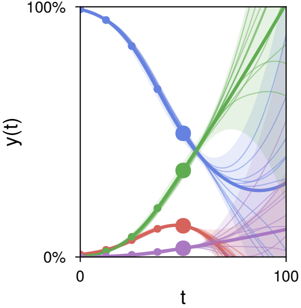
Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Prior

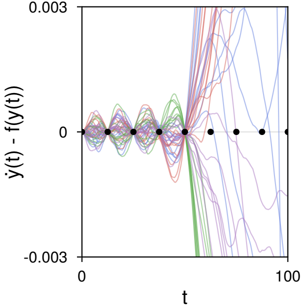
$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process



Likelihood & Data

$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$



Inference

Bayesian filtering and smoothing

Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

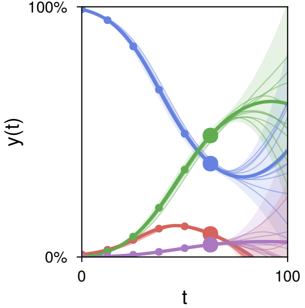
Likelihood & Data

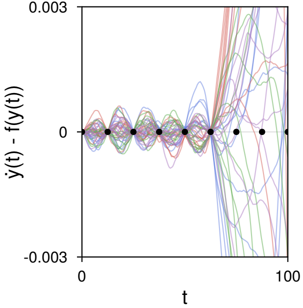
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

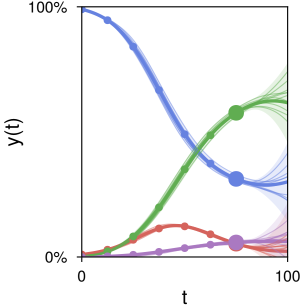
Likelihood & Data

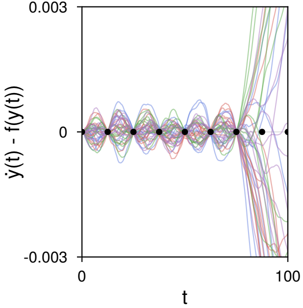
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1{:}N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1{:}N$ **do**
  Predict:
    $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
    $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

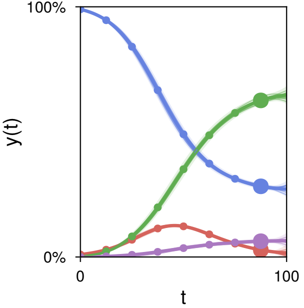
Likelihood & Data

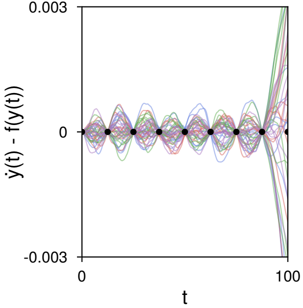
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$

for $i = 1:N$ **do**

  Predict:

  $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$

  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$

  Update:

  $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$

end for

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

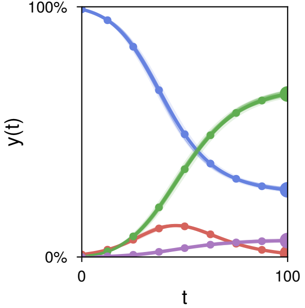
Likelihood & Data

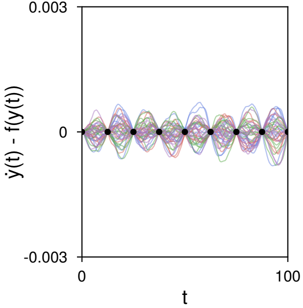
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

Initial distribution $p(y(t_0))$
for $i = 1:N$ **do**
  Predict:
   $p_f(y(t_{i-1})) \mapsto p_p(y(t_i))$
  Linearize f at $\mathbb{E}_{p_p}[y(t_i)]$
  Update:
   $p_p(y(t_i)), z(t_i) \mapsto p_f(y(t_i))$
end for

Building blocks of probabilistic numerical ODE solvers

Prior

$y(t) \sim \mathcal{GP}$ is a Gauss–Markov process

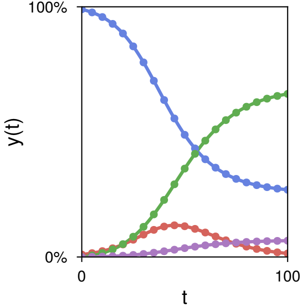
Likelihood & Data

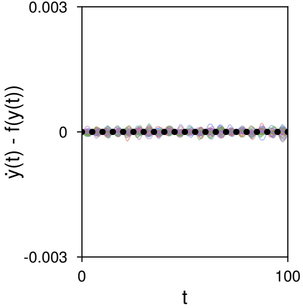
$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

Inference

Bayesian filtering and smoothing





Algorithm Extended Kalman Filter

```
Initial distribution p(y(t_0))
for i = 1:N do
    Predict:
      p_f(y(t_{i-1})) ↦ p_p(y(t_i))
    Linearize f at E_{p_p}[y(t_i)]
    Update:
      p_p(y(t_i)), z(t_i) ↦ p_f(y(t_i))
end for
```

ODE filtering as a *flexible* and *efficient* framwork for simulation *and inference*

Prior

$y(t)$ is a Gauss–Markov process

+

Likelihood & Data

$$z(t) = \dot{y}(t) - f(y(t), t)$$

$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$$

+

Inference

Bayesian filtering and smoothing

3) **Probabilistic exponential integrators**
NB, Hennig, Tronarp (NeurIPS'23)

1) **Pick-and-mix information operators for probabilistic ODE solvers**
NB, Tronarp, Hennig (AISTATS'22)

2) **Parallel-in-Time Probabilistic Numerical ODE Solvers**
NB, Corenflos, Yaghoobi, Tronarp, Hennig, Särkkä (JMLR'24)

Calibrated & adaptive solvers (AISTATS'21)

ODE filters in millions of dimensions (ICML'22)

Parameter Inference

$\dot{y}(t) = f_\theta(y(t), t)$
Find $p(\theta \mid y(t_{1:N}))$

Fenrir: Physics-enhanced regression (ICML'22)

Diffusion tempering (ICML'24)

ProbNumDiffEq.jl (JOSS'24)

Software

Prior

$y(t)$ is a Gauss–Markov process

3) **Probabilistic exponential integrators**
NB, Hennig, Tronarp (NeurIPS'23)

\+

Likelihood & Data

$$z(t) = \dot{y}(t) - f(y(t), t)$$

$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$$

1) **Pick-and-mix information operators for probabilistic ODE solvers**
NB, Tronarp, Hennig (AISTATS'22)

\+

Inference

Bayesian filtering and smoothing

2) **Parallel-in-Time Probabilistic Numerical ODE Solvers**
NB, Corenflos, Yaghoobi, Tronarp, Hennig, Särkkä (JMLR'24)

Calibrated & adaptive solvers (AISTATS'21)

ODE filters in millions of dimensions (ICML'22)

Parameter Inference

$\dot{y}(t) = f_\theta(y(t), t)$
Find $p(\theta \mid y(t_{1:N}))$

Fenrir: Physics-enhanced regression (ICML'22)

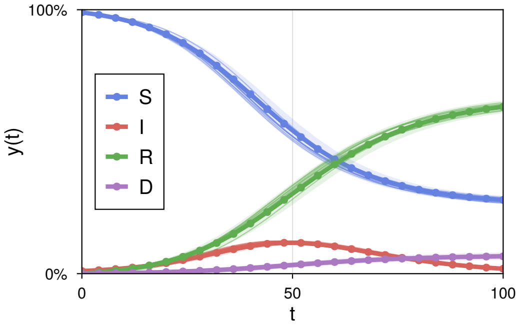
Diffusion tempering (ICML'24)

ProbNumDiffEq.jl (JOSS'24)

Software

The ODE is often not the full story

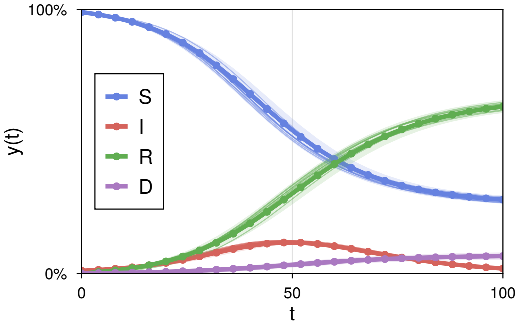
ODE: $\frac{\mathrm{d}}{\mathrm{d}t}[S, I, R, D](t) = f\left([S, I, R, D](t), t\right),$ Initial value: $[S, I, R, D](0) = [0.99, 0.01, 0, 0]$



The ODE is often not the full story

ODE: $\frac{\mathrm{d}}{\mathrm{d}t}[S, I, R, D](t) = f([S, I, R, D](t), t)$, Initial value: $[S, I, R, D](0) = [0.99, 0.01, 0, 0]$

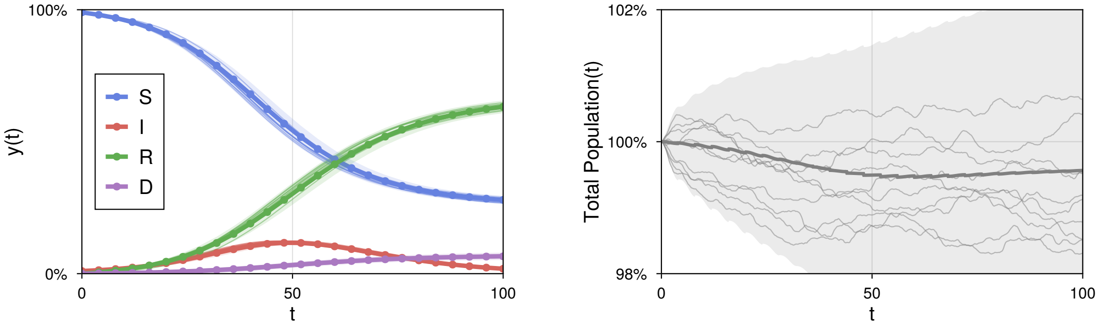
Conserved quantity: $\text{TotalPopulation}(t) := S(t) + I(t) + R(t) + D(t) = 1$



The ODE is often not the full story

ODE: $\frac{\mathrm{d}}{\mathrm{d}t}[S, I, R, D](t) = f([S, I, R, D](t), t)$, Initial value: $[S, I, R, D](0) = [0.99, 0.01, 0, 0]$

Conserved quantity: $\text{TotalPopulation}(t) := S(t) + I(t) + R(t) + D(t) = 1$



Conserved quantities are not actually conserved in the simulation.

Including additional information into the likelihood model

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Ordinary Differential Equation

$$\dot{y}(t) = f(y(t), t)$$

encode as →

Likelihood Model

$$z(t) = \dot{y}(t) - f(y(t), t)$$

$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1 : N$$

Including additional information into the likelihood model

Ordinary Differential Equation with *conserved quantity*

$$\dot{y}(t) = f(y(t), t)$$

$$\boldsymbol{g(y(t)) = g(y_0)}$$

encode as →

Likelihood Model

$$z(t) = \dot{y}(t) - f(y(t), t) \; ?$$

$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$$

Ordinary Differential Equation with *conserved quantity*

$$\dot{y}(t) = f(y(t), t)$$
$$\boldsymbol{g}(\boldsymbol{y}(\boldsymbol{t})) = \boldsymbol{g}(\boldsymbol{y}_0)$$

encode as →

Likelihood Model

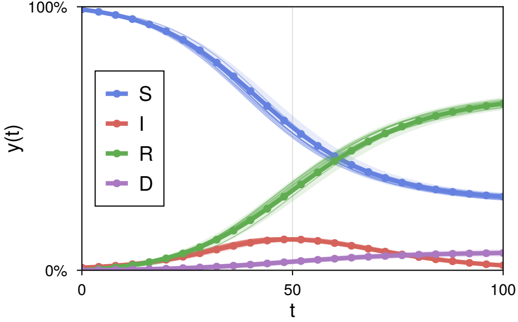
$$z(t) = \begin{bmatrix} \dot{y}(t) - f(y(t), t) \\ \boldsymbol{g}(\boldsymbol{y}(\boldsymbol{t})) - \boldsymbol{g}(\boldsymbol{y}_0) \end{bmatrix}$$

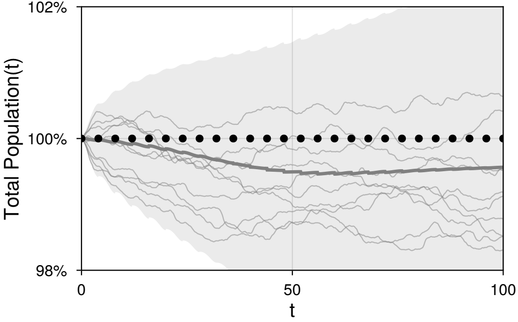
$$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$$

ODE simulation with conservation laws

SIRD initial value problem: $\frac{\mathrm{d}}{\mathrm{d}t}[S, I, R, D](t) = f([S, I, R, D](t), t), \qquad [S, I, R, D](0) = [0.99, 0.01, 0, 0]$

Conserved quantity: $P(t) := S(t) + I(t) + R(t) + D(t) = 1$



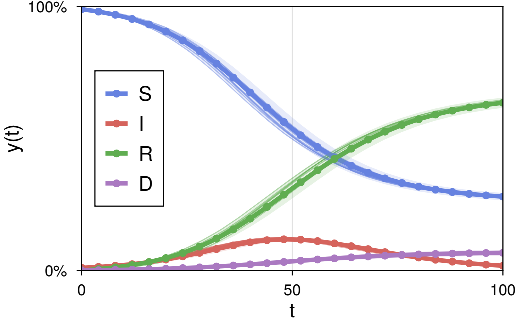


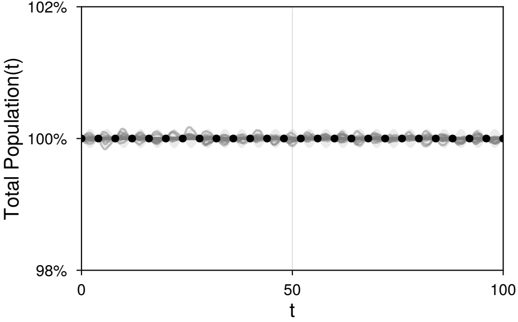
Before incorporating the conservation law.

ODE simulation with conservation laws

SIRD initial value problem: $\frac{\mathrm{d}}{\mathrm{d}t}[S, I, R, D](t) = f\left([S, I, R, D](t), t\right), \qquad [S, I, R, D](0) = [0.99, 0.01, 0, 0]$

Conserved quantity: $P(t) := S(t) + I(t) + R(t) + D(t) = 1$

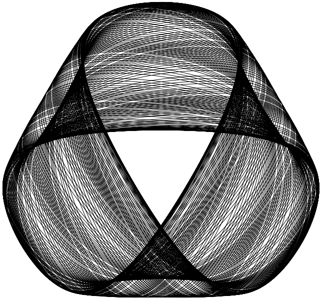




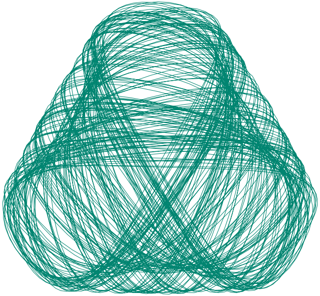
After incorporating the conservation law.

Conserved quantities stabilize long-term simulations

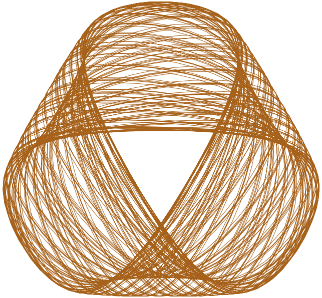
Simulation of the Henon–Heiles system which models a star moving around a galactic center.



Fine-grained simulation



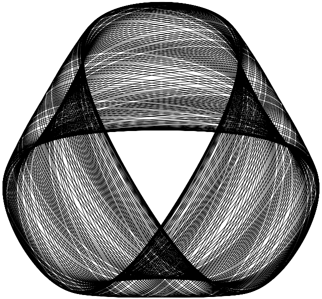
Coarse simulation



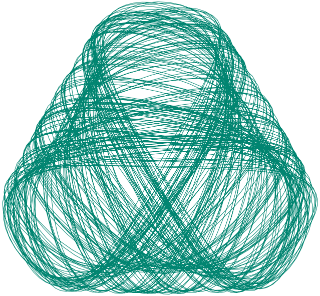
Coarse simulation with conservation of energy

Conserved quantities stabilize long-term simulations

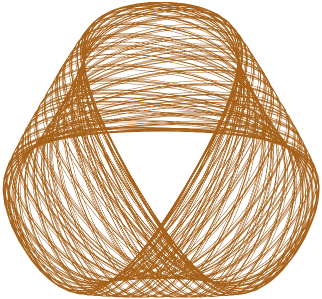
Simulation of the Henon–Heiles system which models a star moving around a galactic center.



Fine-grained simulation



Coarse simulation



Coarse simulation with conservation of energy

ODE filters can easily include additional information by adjusting their *likelihood model*.

Prior

$y(t)$ is a Gauss–Markov process

+

Likelihood & Data

$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

+

Inference

Bayesian filtering and smoothing

3) **Probabilistic exponential integrators**
NB, Hennig, Tronarp (NeurIPS'23)

1) **Pick-and-mix information operators for probabilistic ODE solvers**
NB, Tronarp, Hennig (AISTATS'22)

2) **Parallel-in-Time Probabilistic Numerical ODE Solvers**
NB, Corenflos, Yaghoobi, Tronarp, Hennig, Särkkä (JMLR'24)

Calibrated & adaptive solvers (AISTATS'21)

ODE filters in millions of dimensions (ICML'22)

Parameter Inference

$\dot{y}(t) = f_\theta(y(t), t)$
Find $p(\theta \mid y(t_{1:N}))$

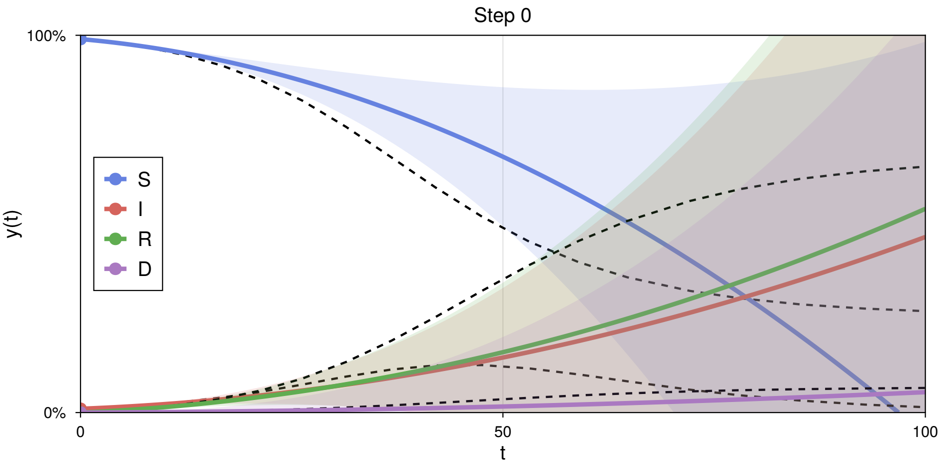
Fenrir: Physics-enhanced regression (ICML'22)

Diffusion tempering (ICML'24)

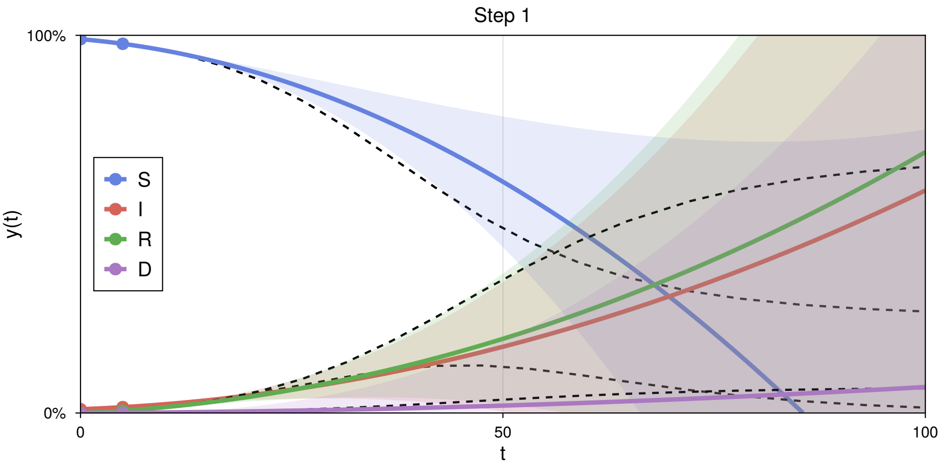
ProbNumDiffEq.jl (JOSS'24)

Software

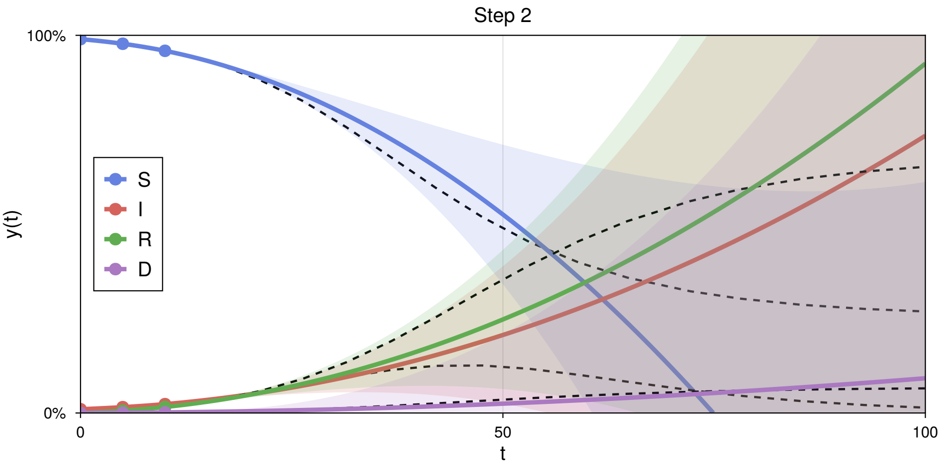
Another step-by-step simulation of the SIRD model



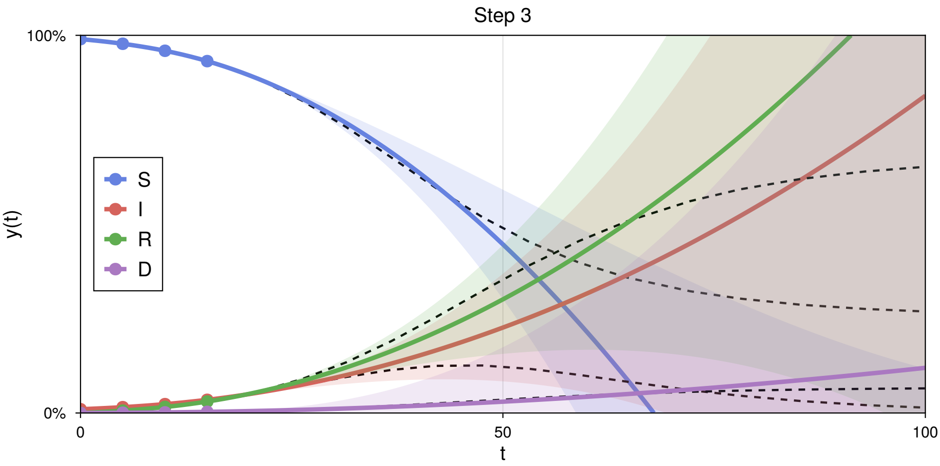
Another step-by-step simulation of the SIRD model



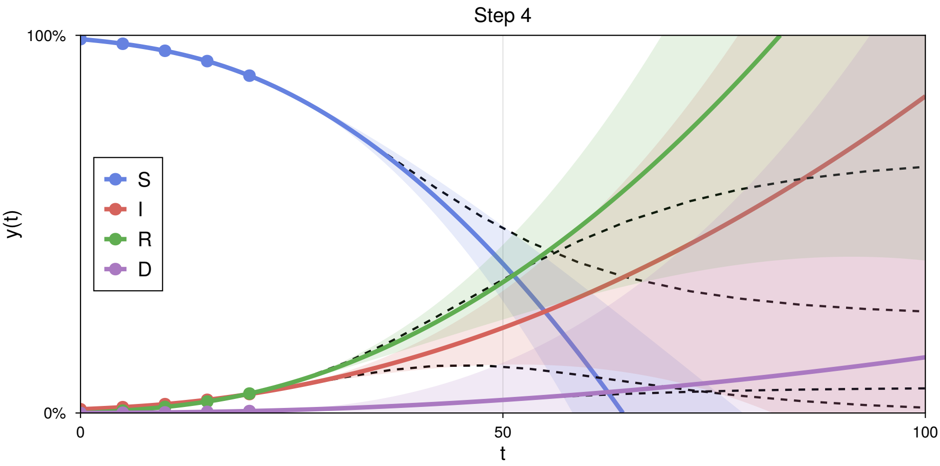
Another step-by-step simulation of the SIRD model



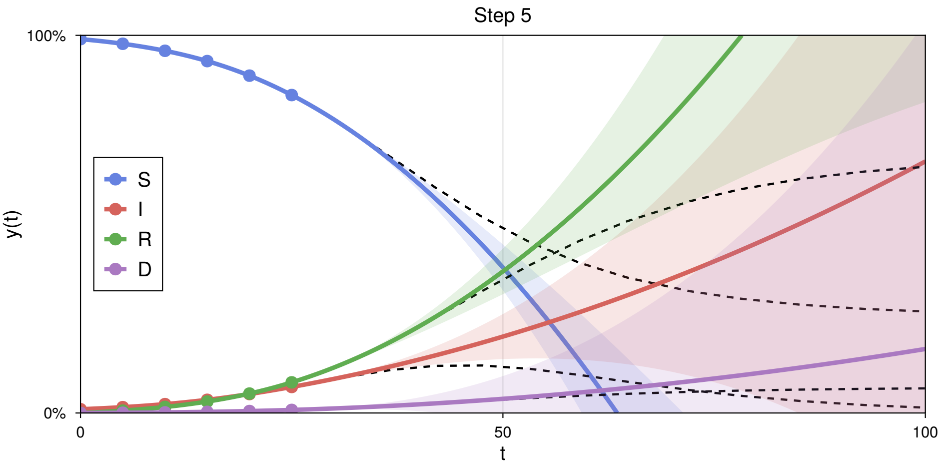
Another step-by-step simulation of the SIRD model



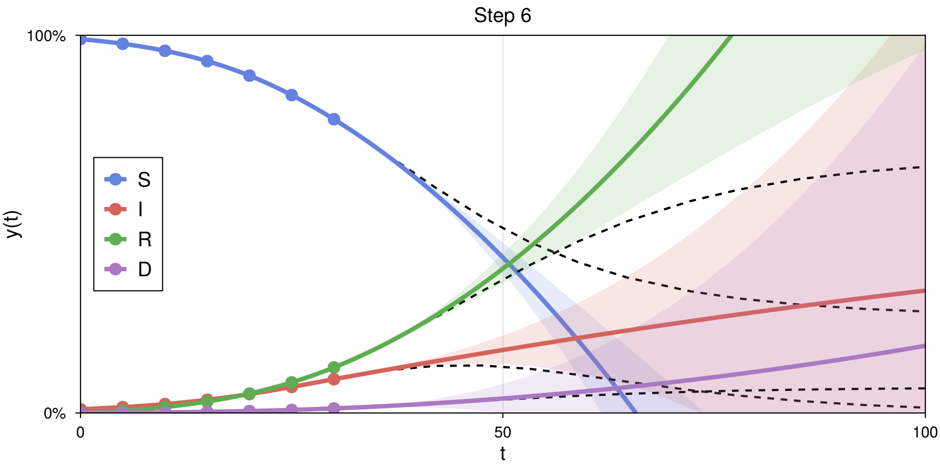
Another step-by-step simulation of the SIRD model



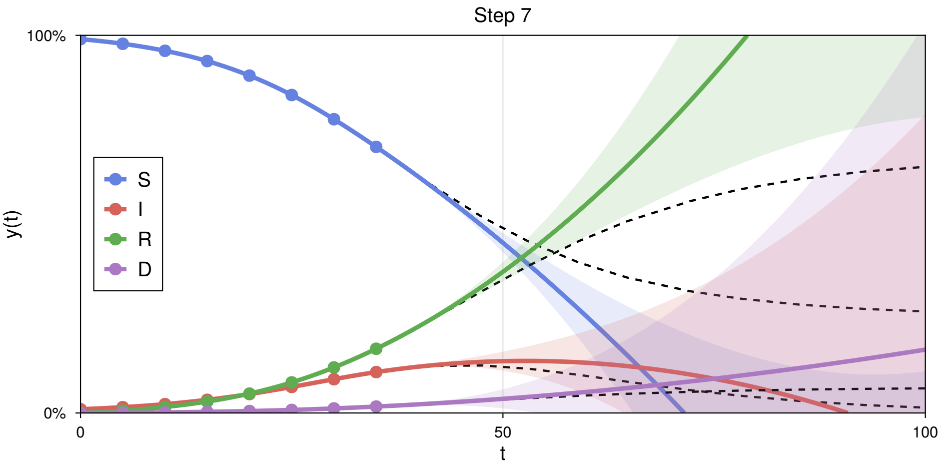
Another step-by-step simulation of the SIRD model



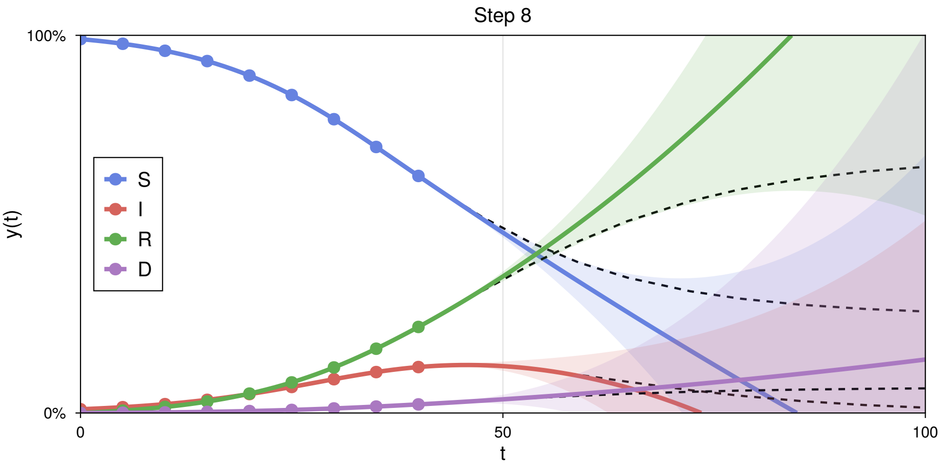
Another step-by-step simulation of the SIRD model



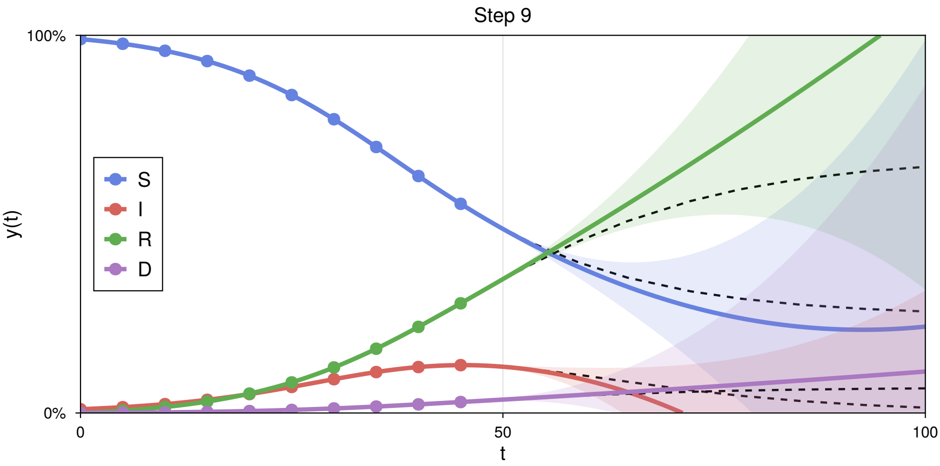
Another step-by-step simulation of the SIRD model



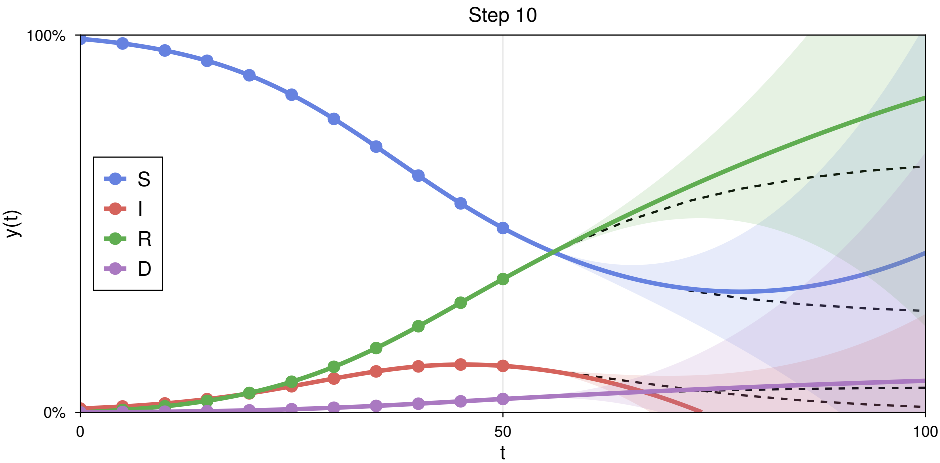
Another step-by-step simulation of the SIRD model



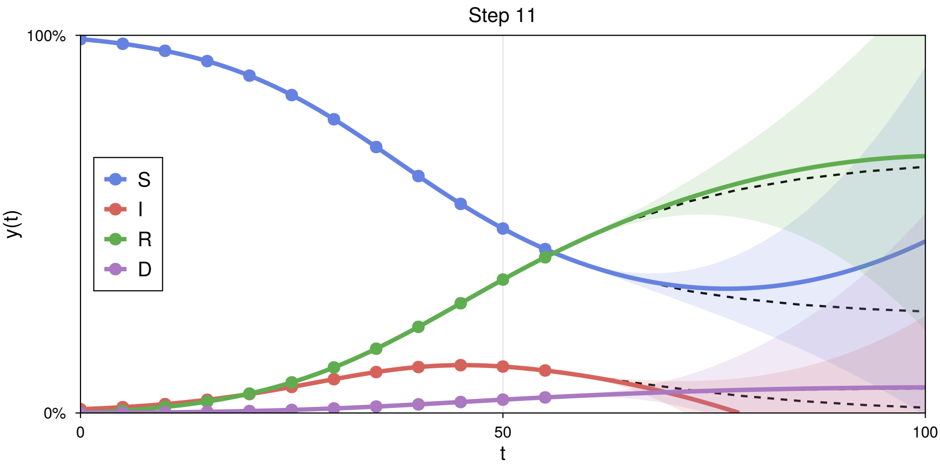
Another step-by-step simulation of the SIRD model



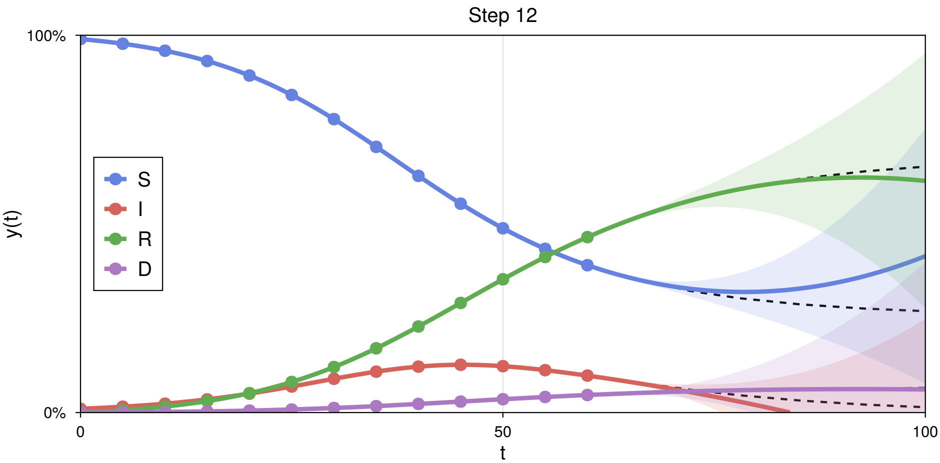
Another step-by-step simulation of the SIRD model



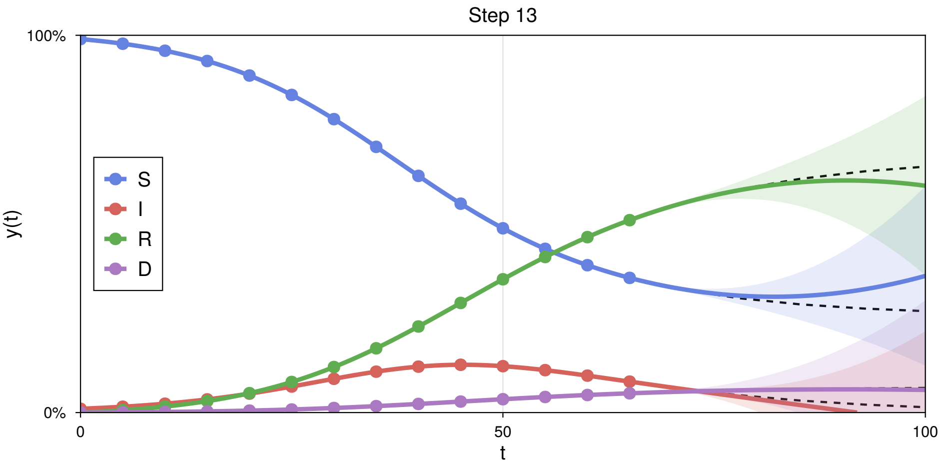
Another step-by-step simulation of the SIRD model



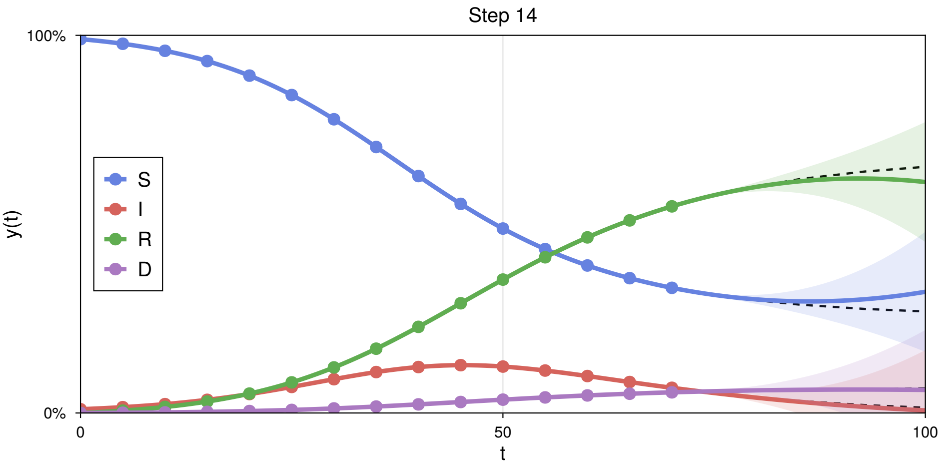
Another step-by-step simulation of the SIRD model



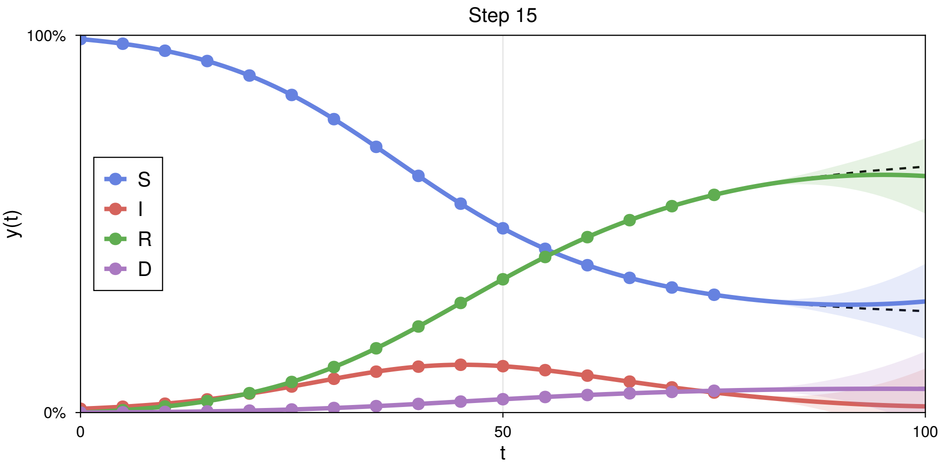
Another step-by-step simulation of the SIRD model



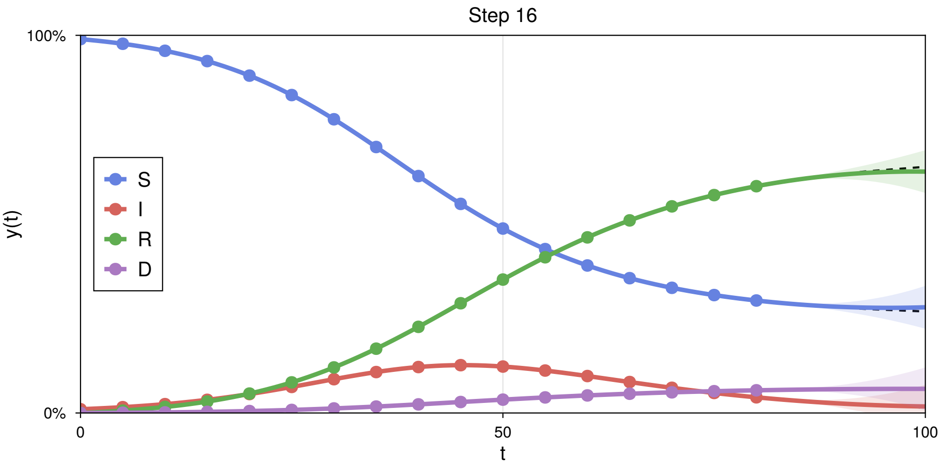
Another step-by-step simulation of the SIRD model



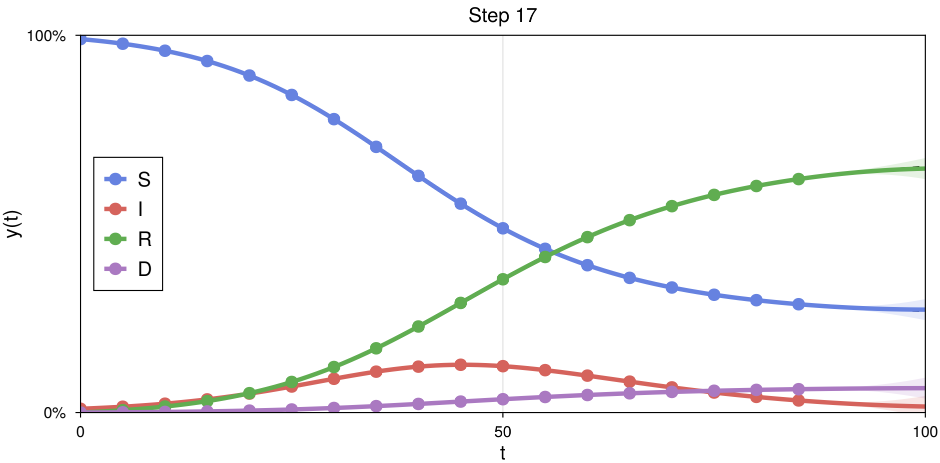
Another step-by-step simulation of the SIRD model



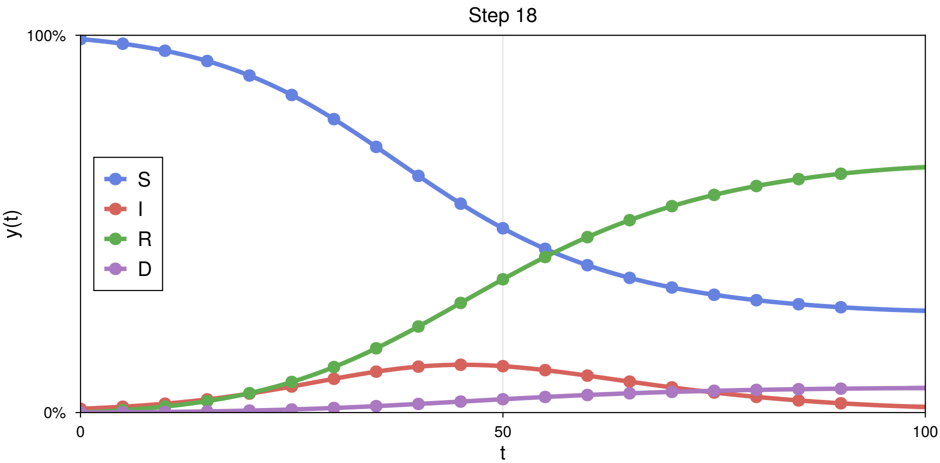
Another step-by-step simulation of the SIRD model



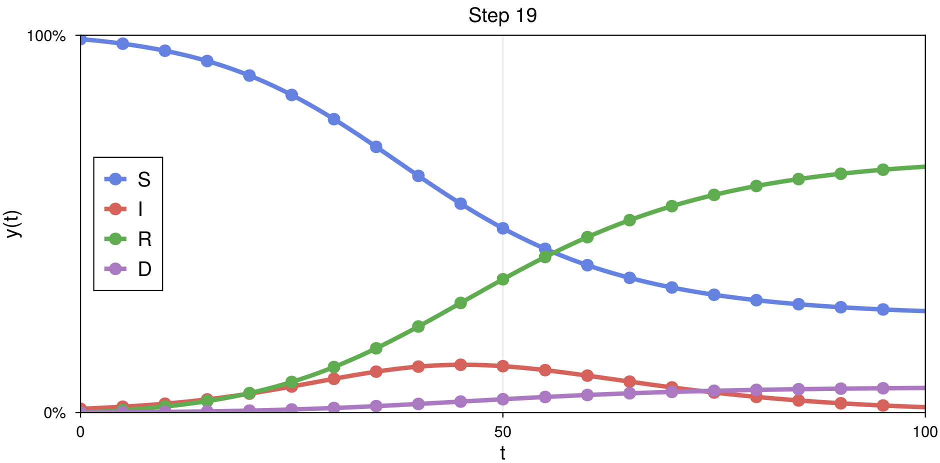
Another step-by-step simulation of the SIRD model



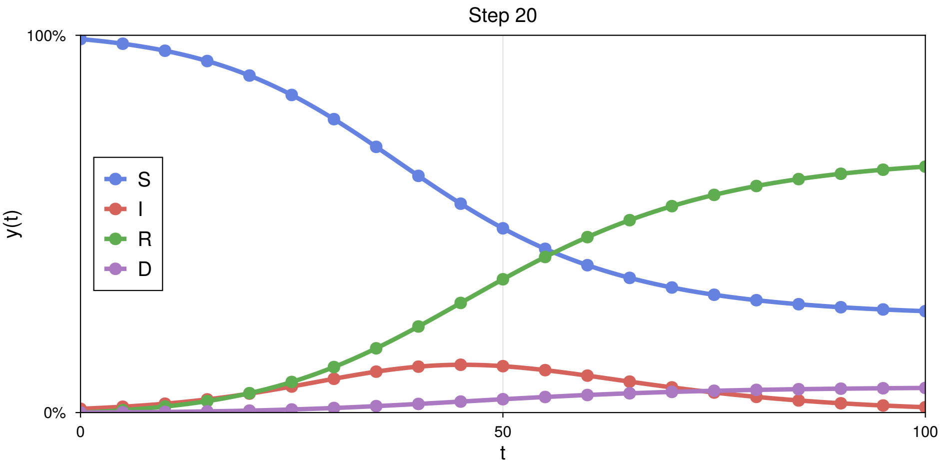
Another step-by-step simulation of the SIRD model



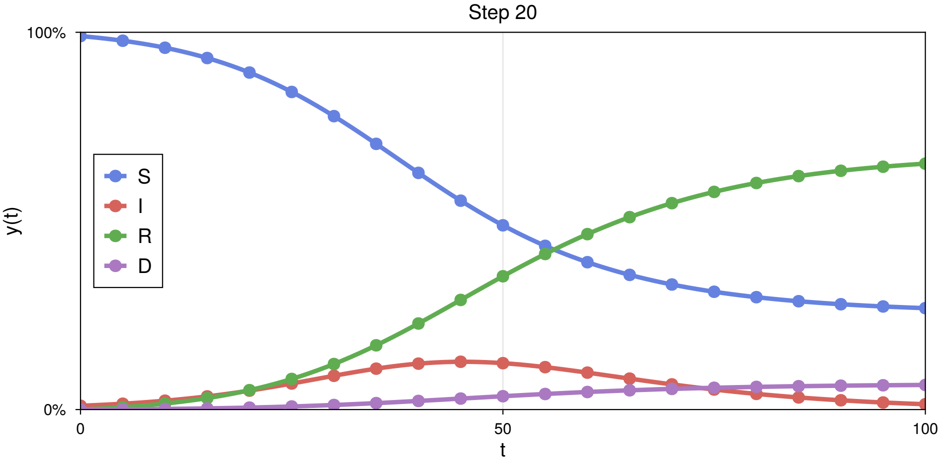
Another step-by-step simulation of the SIRD model



Another step-by-step simulation of the SIRD model

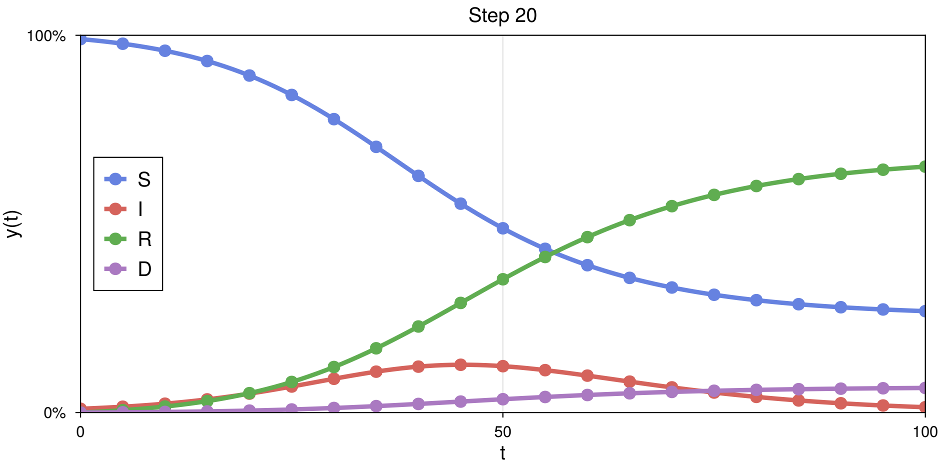


Another step-by-step simulation of the SIRD model



Inference is sequential and scales $\mathcal{O}(N)$.

Another step-by-step simulation of the SIRD model



Inference is sequential and scales $\mathcal{O}(N)$. Can we do better?

From time-parallel Bayesian filters to parallel-in-time ODE solvers

- [Särkkä and García-Fernández, 2021]:

  > Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

From time-parallel Bayesian filters to parallel-in-time ODE solvers

- [Särkkä and García-Fernández, 2021]:

  > Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  > Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

From time-parallel Bayesian filters to parallel-in-time ODE solvers

- [Särkkä and García-Fernández, 2021]:

  Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

Algorithm Time-parallel Iterated Extended Kalman Smoother

Initial trajectory $p(y(t_{1:N}))$
while not converged **do**
    (i) *Linearize the model* globally *along the trajectory.*
    (ii) *Run the time-parallel Kalman smoother on the linearized model.*
end while

From time-parallel Bayesian filters to parallel-in-time ODE solvers

- [Särkkä and García-Fernández, 2021]:

  Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

Algorithm Time-parallel Iterated Extended Kalman Smoother

Initial trajectory $p(y(t_{1:N}))$
while not converged **do**
  (i) *Linearize the model* globally *along the trajectory.*
  (ii) *Run the time-parallel Kalman smoother on the linearized model.*
end while

From time-parallel Bayesian filters to parallel-in-time ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- [Särkkä and García-Fernández, 2021]:

  Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

Algorithm Time-parallel Iterated Extended Kalman Smoother

Initial trajectory $p(y(t_{1:N}))$
while not converged **do**
  (i) *Linearize the model* globally *along the trajectory.*
  (ii) *Run the time-parallel Kalman smoother on the linearized model.*
end while

From time-parallel Bayesian filters to parallel-in-time ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- [Särkkä and García-Fernández, 2021]:

  Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

Algorithm Time-parallel Iterated Extended Kalman Smoother

```
Initial trajectory p(y(t_{1:N}))
while not converged do
    (i) Linearize the model globally along the trajectory.
    (ii) Run the time-parallel Kalman smoother on the linearized model.
end while
```

From time-parallel Bayesian filters to parallel-in-time ODE solvers

- [Särkkä and García-Fernández, 2021]:

  Kalman smoothing for **linear** Gaussian models can be done in parallel time ($\mathcal{O}(\log N)$).

- [Yaghoobi et al., 2023]:

  Iterated extended Kalman smoothing for **nonlinear** models in parallel time ($\mathcal{O}(k \log N)$).

Algorithm Time-parallel Iterated Extended Kalman Smoother

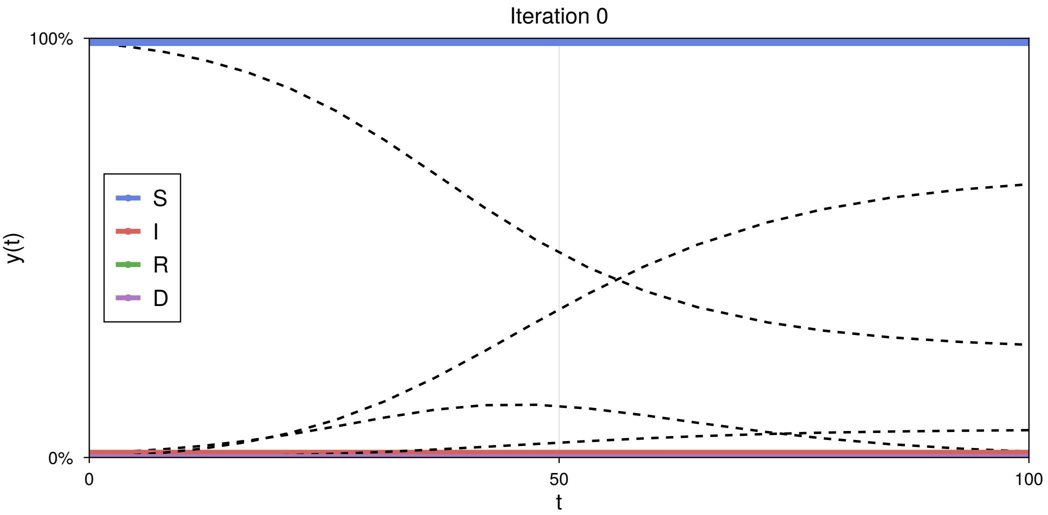
```
Initial trajectory p(y(t_{1:N}))
while not converged do
    (i) Linearize the model globally along the trajectory.
    (ii) Run the time-parallel Kalman smoother on the linearized model.
end while
```

- [Bosch et al., 2024]:

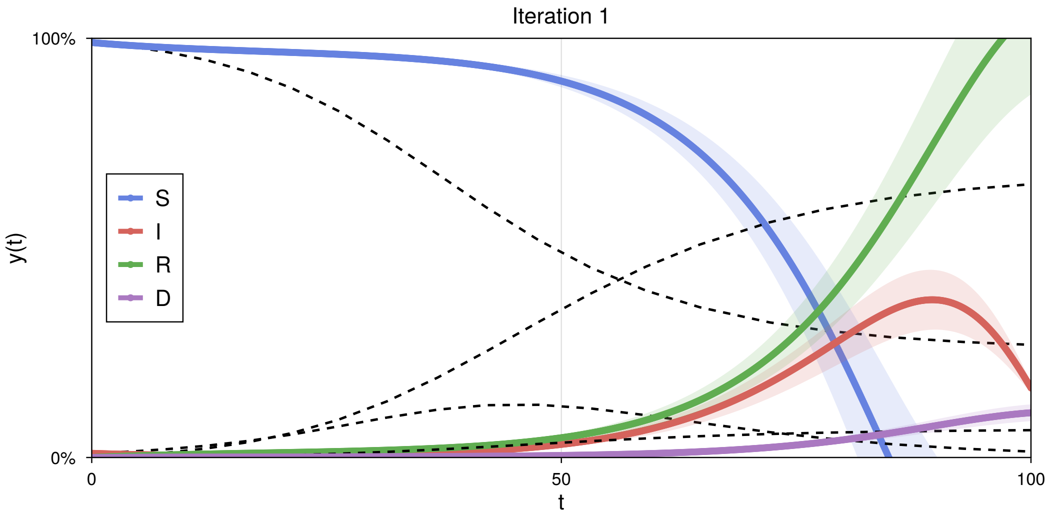
  Parallel-in-time probabilistic numerical ODE solvers in $\mathcal{O}(k \log N)$ time.

Simulating 1000 steps of the SIRD model *parallel-in-time*

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

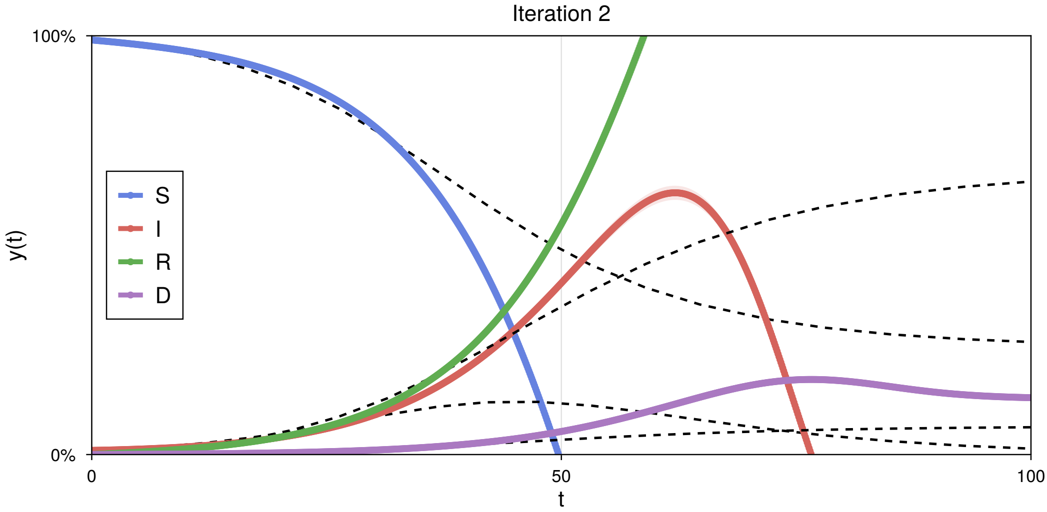


Simulating 1000 steps of the SIRD model *parallel-in-time*

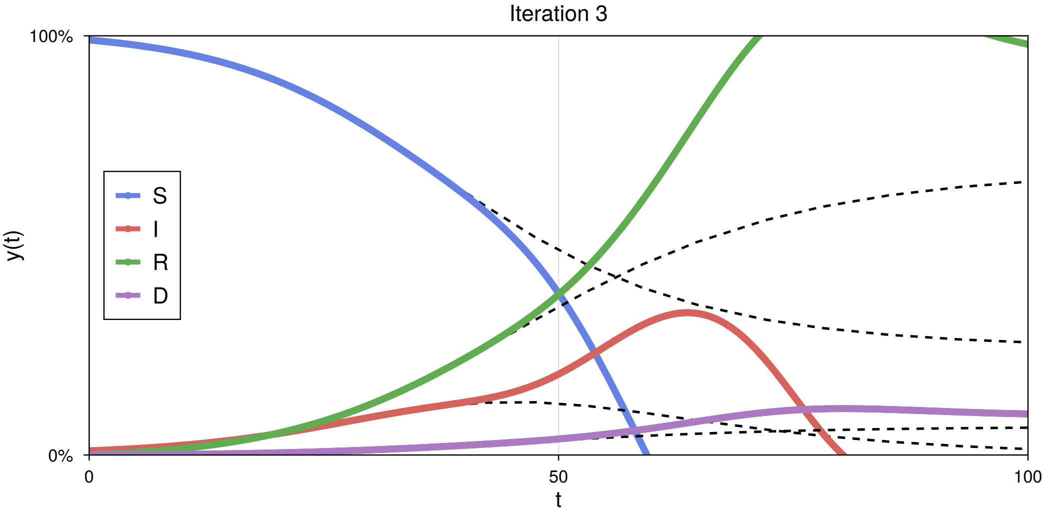


Simulating 1000 steps of the SIRD model *parallel-in-time*

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

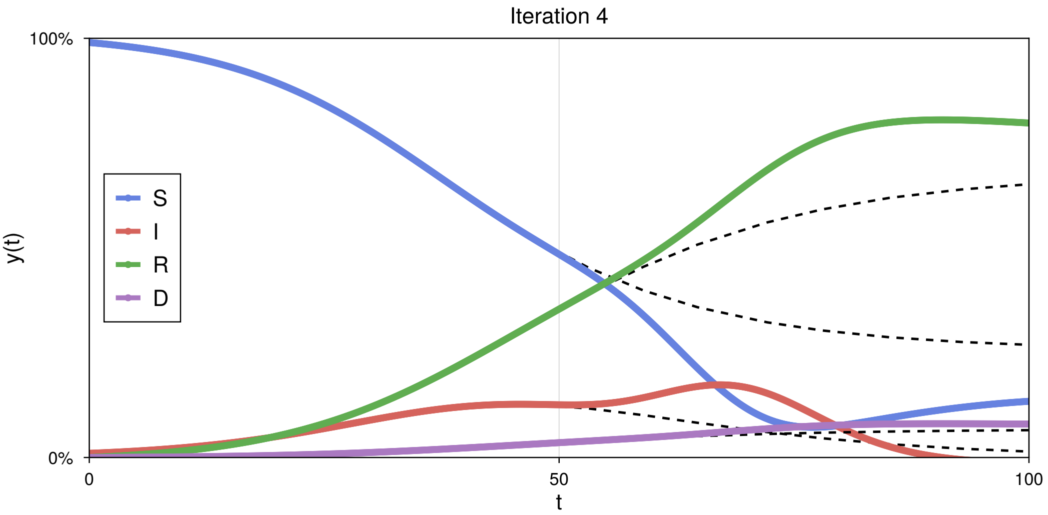


Simulating 1000 steps of the SIRD model *parallel-in-time*

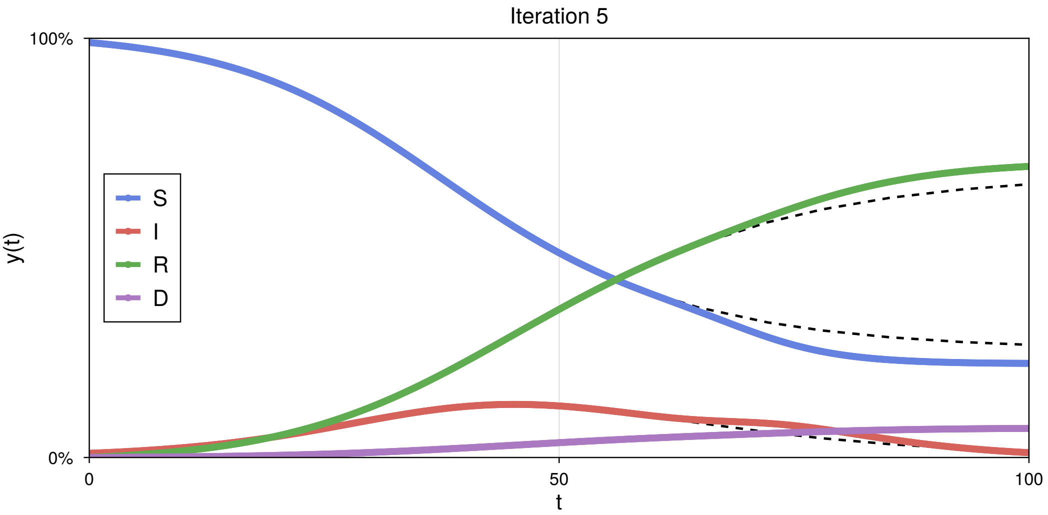


Simulating 1000 steps of the SIRD model *parallel-in-time*

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

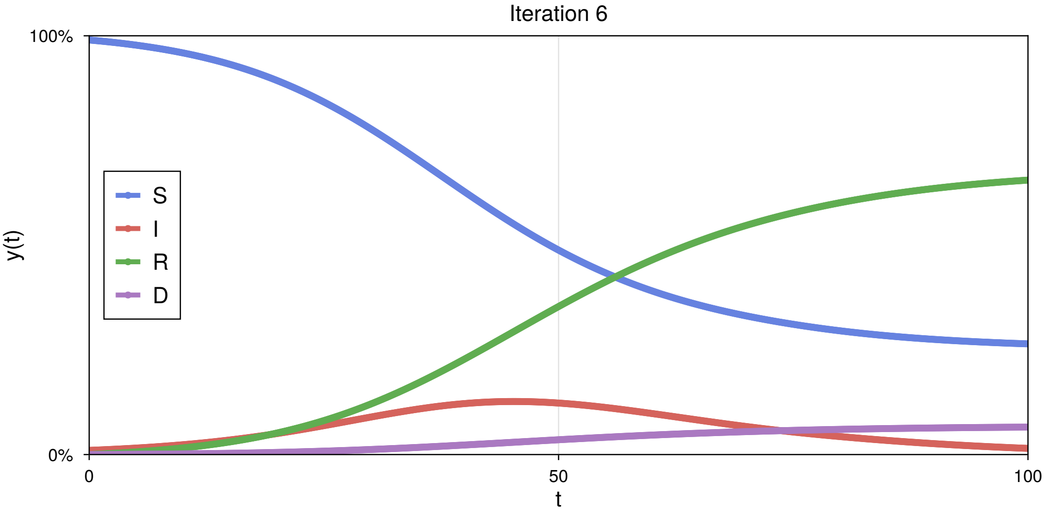


Simulating 1000 steps of the SIRD model *parallel-in-time*

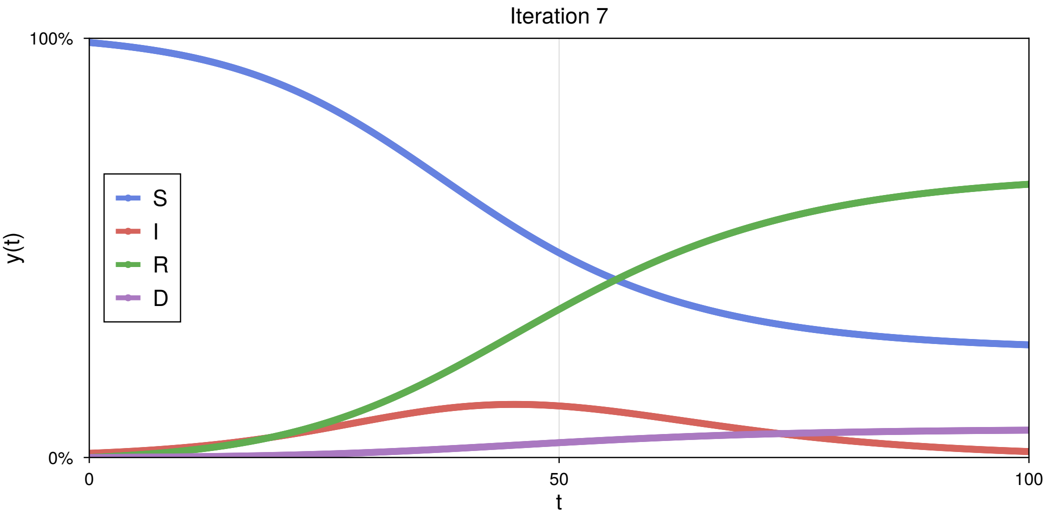


Simulating 1000 steps of the SIRD model *parallel-in-time*

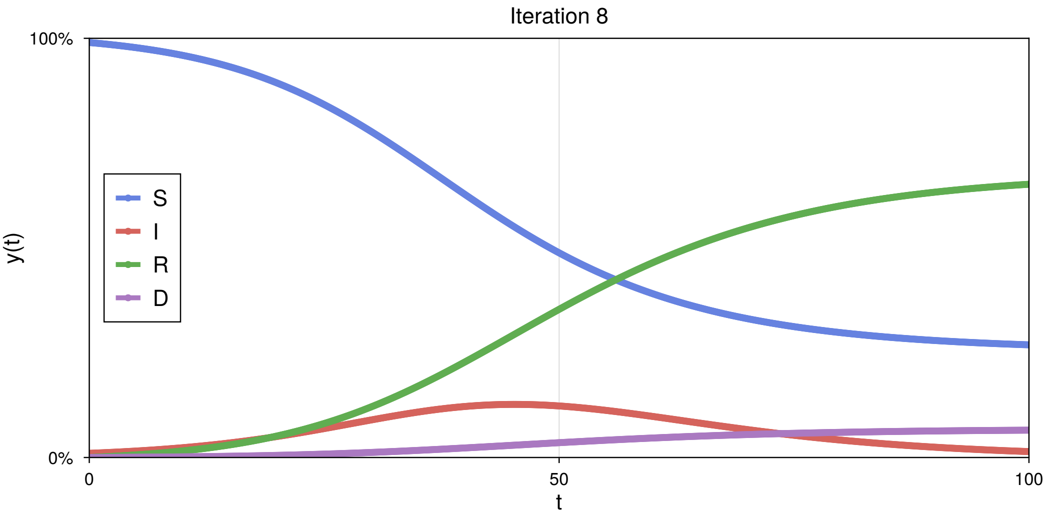
EBERHARD KARLS UNIVERSITÄT TÜBINGEN



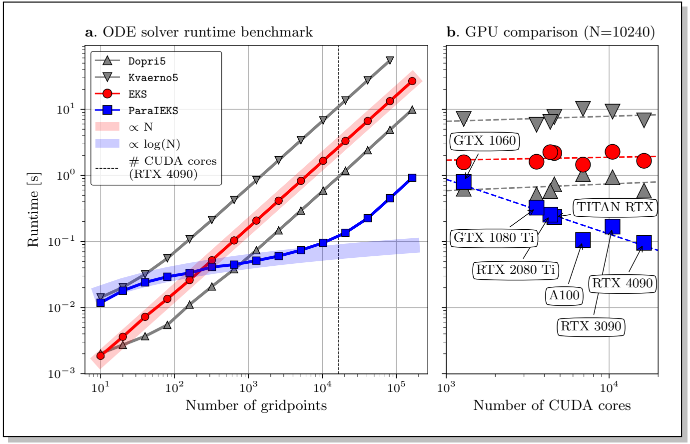
Simulating 1000 steps of the SIRD model *parallel-in-time*



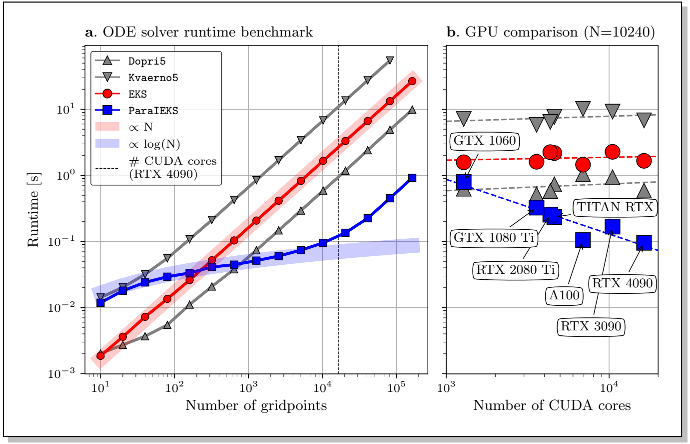
Simulating 1000 steps of the SIRD model *parallel-in-time*



Benefits of parallel-in-time simulation



Benefits of parallel-in-time simulation



Inference in ODE filters can be performed parallel-in-time at logarithmic cost.
⇒ Significant speedups for large ODE simulations on GPUs.

Prior

$y(t)$ is a Gauss–Markov process

3) **Probabilistic exponential integrators**
NB, Hennig, Tronarp
(NeurIPS'23)

+

Likelihood & Data

$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

1) **Pick-and-mix information operators for probabilistic ODE solvers**
NB, Tronarp, Hennig
(AISTATS'22)

+

Inference

Bayesian filtering and smoothing

2) **Parallel-in-Time Probabilistic Numerical ODE Solvers**
NB, Corenflos, Yaghoobi, Tronarp, Hennig, Särkkä
(JMLR'24)

Calibrated & adaptive solvers (AISTATS'21)

ODE filters in millions of dimensions (ICML'22)

Parameter Inference

$\dot{y}(t) = f_\theta(y(t), t)$
Find $p(\theta \mid y(t_{1:N}))$

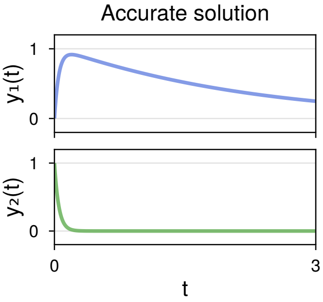
Fenrir: Physics-enhanced regression (ICML'22)

Diffusion tempering (ICML'24)

ProbNumDiffEq.jl (JOSS'24)

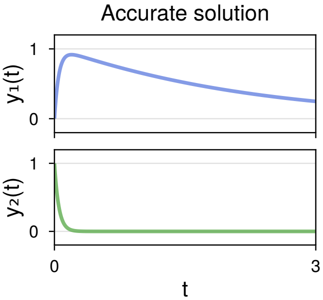
Software

$$\dot{y}_1(t) = 20 y_2(t) - 0.5 \sin(y_1(t)) \qquad y_1(0) = 0$$
$$\dot{y}_2(t) = -20 y_2(t) \qquad y_2(0) = 1$$



Stiff ordinary differential equations

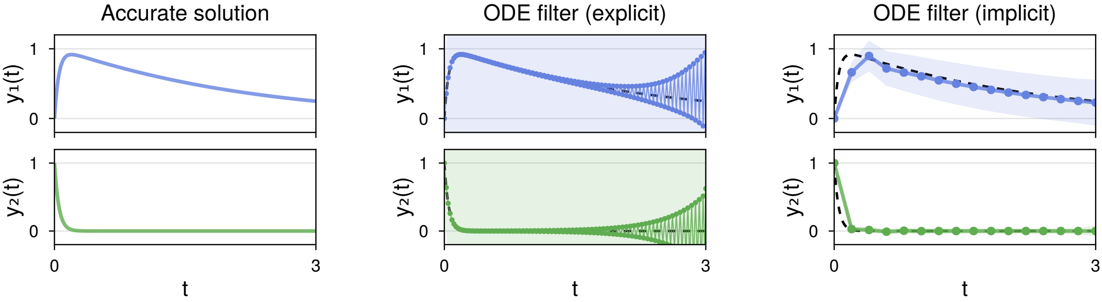
$$\dot{y}_1(t) = 20y_2(t) - 0.5\sin(y_1(t)) \qquad y_1(0) = 0$$
$$\dot{y}_2(t) = -20y_2(t) \qquad y_2(0) = 1$$



Stiff ODEs combine fast and slow dynamics ⇒ challenging to simulate

Stiff ordinary differential equations

$$\dot{y}_1(t) = 20y_2(t) - 0.5\sin(y_1(t)) \qquad y_1(0) = 0$$
$$\dot{y}_2(t) = -20y_2(t) \qquad y_2(0) = 1$$

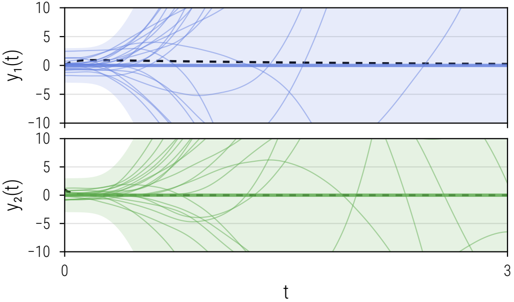


Stiff ODEs combine fast and slow dynamics $\Rightarrow$ challenging to simulate

Improving stability by adjusting the prior

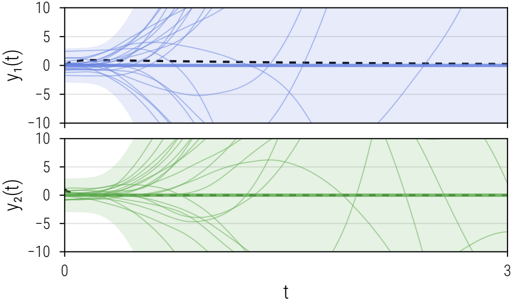
q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

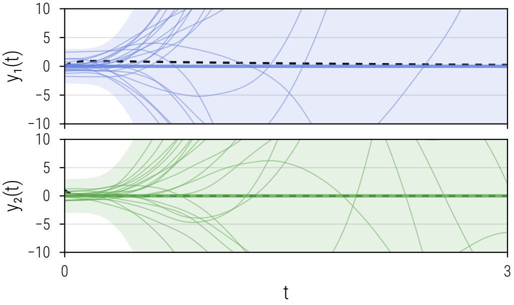
$$\mathrm{d}y^{(q)}(t) = \boxed{L \cdot y^{(q)}(t)}\,\mathrm{d}t + \mathrm{d}W(t)$$

Improving stability by adjusting the prior

$$\begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 20 \\ 0 & -20 \end{bmatrix} \cdot \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} + \begin{bmatrix} -0.5\sin(y_1(t)) \\ 0 \end{bmatrix}$$

q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

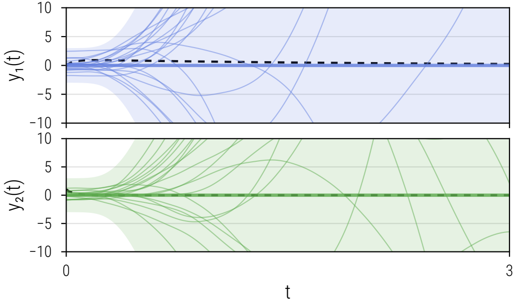
$$\mathrm{d}y^{(q)}(t) = \boxed{L \cdot y^{(q)}(t)}\,\mathrm{d}t + \mathrm{d}W(t)$$

Improving stability by adjusting the prior

$$\begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 20 \\ 0 & -20 \end{bmatrix} \cdot \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}}_{L \cdot y(t)} + \underbrace{\begin{bmatrix} -0.5 \sin(y_1(t)) \\ 0 \end{bmatrix}}_{N(y(t), t)}$$

q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

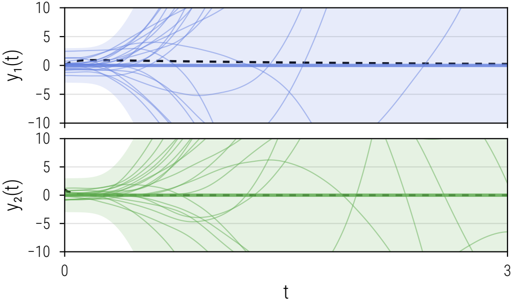
$$\mathrm{d}y^{(q)}(t) = L \cdot y^{(q)}(t)\,\mathrm{d}t + \mathrm{d}W(t)$$

Improving stability by adjusting the prior

$$\begin{bmatrix}\dot{y}_1(t)\\ \dot{y}_2(t)\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 20\\ 0 & -20\end{bmatrix}\cdot\begin{bmatrix}y_1(t)\\ y_2(t)\end{bmatrix}}_{L\cdot y(t)} + \underbrace{\begin{bmatrix}-0.5\sin(y_1(t))\\ 0\end{bmatrix}}_{N(y(t),t)}$$

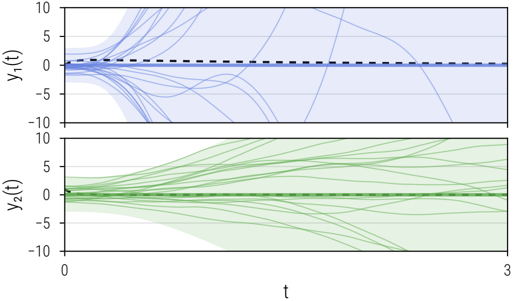
q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

$$\mathrm{d}y^{(q)}(t) = L\cdot y^{(q)}(t)\,\mathrm{d}t + \mathrm{d}W(t)$$

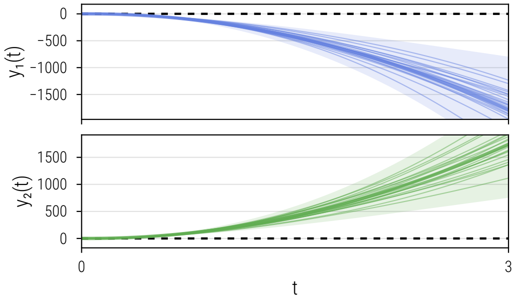


Improving stability by adjusting the prior

$$\begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 20 \\ 0 & -20 \end{bmatrix} \cdot \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}}_{L \cdot y(t)} + \underbrace{\begin{bmatrix} -0.5 \sin(y_1(t)) \\ 0 \end{bmatrix}}_{N(y(t),t)} \quad \text{with} \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

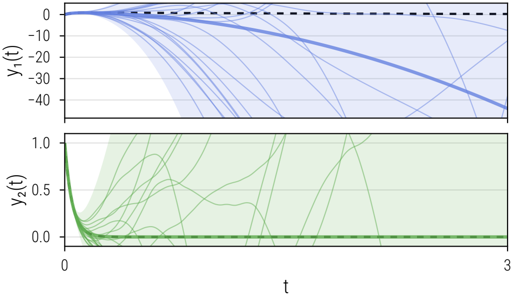
q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

$$\mathrm{d}y^{(q)}(t) = L \cdot y^{(q)}(t)\,\mathrm{d}t + \mathrm{d}W(t)$$

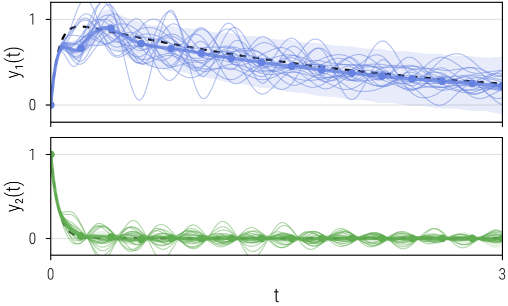


Improving stability by adjusting the prior

$$\begin{bmatrix}\dot{y}_1(t)\\ \dot{y}_2(t)\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 20\\ 0 & -20\end{bmatrix}\cdot\begin{bmatrix}y_1(t)\\ y_2(t)\end{bmatrix}}_{L\cdot y(t)} + \underbrace{\begin{bmatrix}-0.5\sin(y_1(t))\\ 0\end{bmatrix}}_{N(y(t),t)} \quad \text{with} \quad \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix}$$

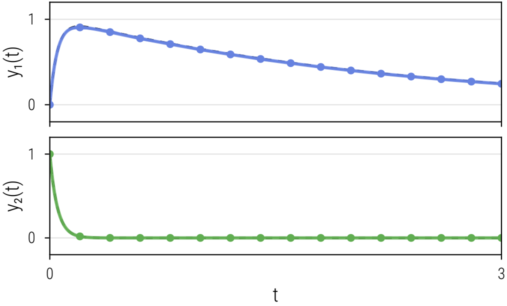
q-times integrated Wiener process:

$$\mathrm{d}y^{(q)}(t) = \mathrm{d}W(t)$$



q-times integrated Ornstein–Uhlenbeck process:

$$\mathrm{d}y^{(q)}(t) = L\cdot y^{(q)}(t)\,\mathrm{d}t + \mathrm{d}W(t)$$



Results on a stiff partial differential equation

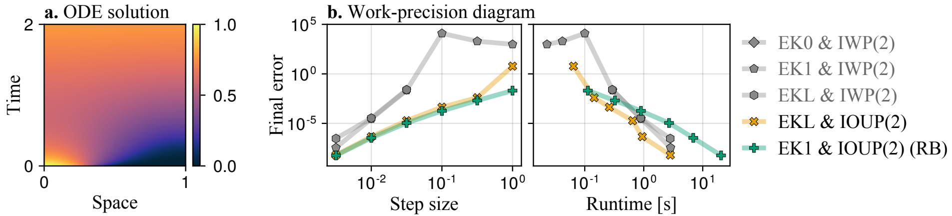


Figure: *Reaction-diffusion model.*

Results on a stiff partial differential equation

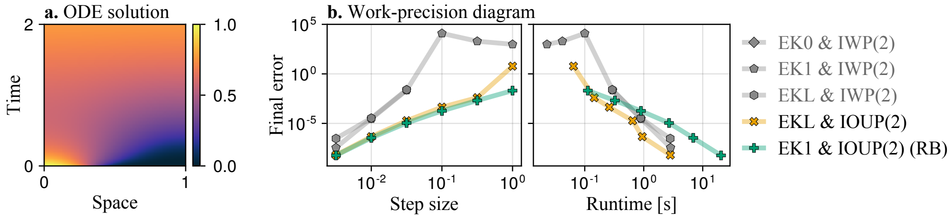


Figure: *Reaction-diffusion model.*

Linear dynamics can be incorporated into the *prior* to stabilize ODE filters.
⇒ Accurate simulation of stiff ODEs (and PDEs) at larger step sizes.

Prior

$y(t)$ is a Gauss–Markov process

3) **Probabilistic exponential integrators**
NB, Hennig, Tronarp
(NeurIPS'23)

$+$

Likelihood & Data

$z(t) = \dot{y}(t) - f(y(t), t)$

$z(t_i) \stackrel{!}{=} 0 \quad \forall i = 1:N$

1) **Pick-and-mix information operators for probabilistic ODE solvers**
NB, Tronarp, Hennig
(AISTATS'22)

$+$

Inference

Bayesian filtering and smoothing

2) **Parallel-in-Time Probabilistic Numerical ODE Solvers**
NB, Corenflos, Yaghoobi, Tronarp, Hennig, Särkkä
(JMLR'24)

Calibrated & adaptive solvers (AISTATS'21)

ODE filters in millions of dimensions (ICML'22)

Parameter Inference

$\dot{y}(t) = f_\theta(y(t), t)$
Find $p(\theta \mid y(t_{1:N}))$

Fenrir: Physics-enhanced regression (ICML'22)

Diffusion tempering (ICML'24)

Software

ProbNumDiffEq.jl (JOSS'24)

A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

Flexible

ODE filters consist of adjustable building blocks:

- **Prior:** Include linear dynamics for stability
- **Likelihood:** Customize to include nonlinear information or to match the given problem
- **Inference:** Use any suitable Bayesian filter / smoother

A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

Flexible

ODE filters consist of adjustable building blocks:

- **Prior:** Include linear dynamics for stability
- **Likelihood:** Customize to include nonlinear information or to match the given problem
- **Inference:** Use any suitable Bayesian filter / smoother

Efficient

- More accurate solutions for ODEs with conserved quantities and stiff ODEs
- Parallel-in-time inference on GPUs

A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

Flexible

ODE filters consist of adjustable building blocks:

- **Prior:** Include linear dynamics for stability
- **Likelihood:** Customize to include nonlinear information or to match the given problem
- **Inference:** Use any suitable Bayesian filter / smoother

Efficient

- More accurate solutions for ODEs with conserved quantities and stiff ODEs
- Parallel-in-time inference on GPUs

Accessible

- Open-source package: ProbNumDiffEq.jl

A Flexible and Efficient Framework for Probabilistic Numerical Simulation and Inference

Flexible

ODE filters consist of adjustable building blocks:

- **Prior:** Include linear dynamics for stability
- **Likelihood:** Customize to include nonlinear information or to match the given problem
- **Inference:** Use any suitable Bayesian filter / smoother

Efficient

- More accurate solutions for ODEs with conserved quantities and stiff ODEs
- Parallel-in-time inference on GPUs

Accessible

- Open-source package: ProbNumDiffEq.jl

Thank you all!