Fast probabilistic inference for ODEs with ProbNumDiffEq.jl

JuliaCon 2024

Nathanael Bosch

11. July 2024

Eberhard Karls Universität Tübingen

impрs-is

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

Background: **Ordinary Differential Equations and how to solve them**

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

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

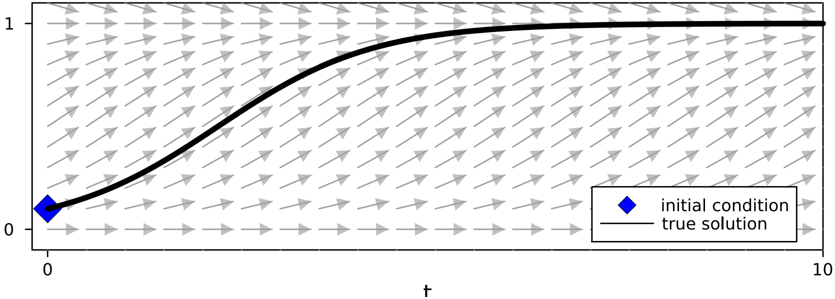
Numerical ODE solvers try to estimate an unknown function by evaluating the vector field

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

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

- **Simple example**: Logistic ODE

$$\dot{y}(t) = y(t)\left(1 - y(t)\right), \qquad t \in [0, 10], \qquad y(0) = 0.1.$$



Background: Ordinary Differential Equations and how to solve them

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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

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

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

Numerical ODE solvers:

- Forward Euler:

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

Background: Ordinary Differential Equations and how to solve them

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

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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

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

Numerical ODE solvers:

- Forward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t)$$
- Backward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t+h), t+h)$$

Background: Ordinary Differential Equations and how to solve them

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

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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

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

Numerical ODE solvers:

- Forward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t)$$
- Backward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t+h), t+h)$$
- Runge–Kutta:
  $$\hat{y}(t+h) = \hat{y}(t) + h \sum_{i=1}^{s} b_i f(\tilde{y}_i, t + c_i h)$$

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

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

Numerical ODE solvers:

- Forward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t)$$
- Backward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t+h), t+h)$$
- Runge–Kutta:
  $$\hat{y}(t+h) = \hat{y}(t) + h \sum_{i=1}^{s} b_i f(\tilde{y}_i, t + c_i h)$$
- Multistep:
  $$\hat{y}(t+h) = \hat{y}(t) + h \sum_{i=0}^{s-1} b_i f(\hat{y}(t - ih), t - ih)$$

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

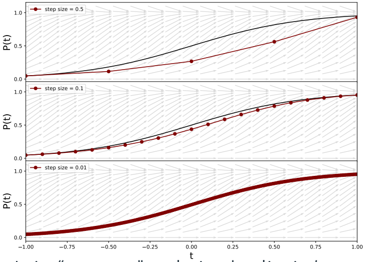
$$\dot{y}(t) = f(y(t), t)$$

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

Numerical ODE solvers:

- Forward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t)$$
- Backward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t+h), t+h)$$
- Runge–Kutta:
  $$\hat{y}(t+h) = \hat{y}(t) + h\sum_{i=1}^{s} b_i f(\tilde{y}_i, t + c_i h)$$
- Multistep:
  $$\hat{y}(t+h) = \hat{y}(t) + h\sum_{i=0}^{s-1} b_i f(\hat{y}(t - ih), t - ih)$$

Forward Euler for different step sizes:



$\Rightarrow$ It is "correct" only in the limit $h \to 0$!

Background: Ordinary Differential Equations and how to solve them

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

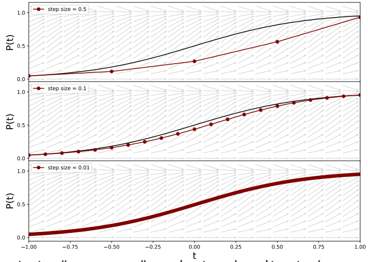
$$\dot{y}(t) = f(y(t), t)$$

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

Numerical ODE solvers:

- Forward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t), t)$$
- Backward Euler:
  $$\hat{y}(t+h) = \hat{y}(t) + hf(\hat{y}(t+h), t+h)$$
- Runge–Kutta:
  $$\hat{y}(t+h) = \hat{y}(t) + h\sum_{i=1}^{s} b_i f(\tilde{y}_i, t + c_i h)$$
- Multistep:
  $$\hat{y}(t+h) = \hat{y}(t) + h\sum_{i=0}^{s-1} b_i f(\hat{y}(t-ih), t-ih)$$

Forward Euler for different step sizes:



$\Rightarrow$ It is "correct" only in the limit $h \to 0$!

Numerical ODE solvers **estimate** $y(t)$ *by evaluating f on a discrete set of points.*

Probabilistic numerical ODE solvers

or “How to treat ODE solving as the Bayesian state estimation problem that it really is”

Probabilistic numerical ODE solvers

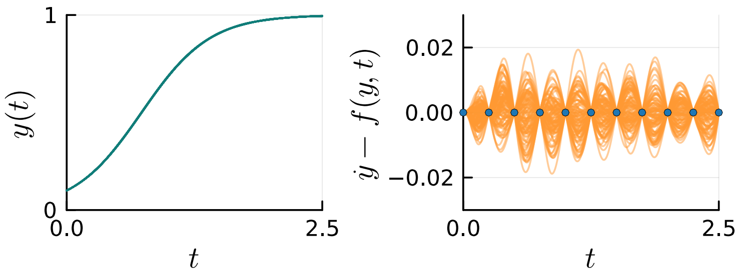
How to treat ODEs as the state estimation problem that they really are

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

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

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

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



Probabilistic numerical ODE solvers

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:**

Probabilistic numerical ODE solvers

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

Probabilistic numerical ODE solvers

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:** $y(t) \sim \mathcal{GP}$ a Gauss–Markov process
- **Likelihood:** (aka “observation model” or “information operator”)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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

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:** $y(t) \sim \mathcal{GP}$ a Gauss–Markov process
- **Likelihood:** (aka "observation model" or "information operator")

$$y(0) - y_0 = 0, \qquad \& \qquad \dot{y}(t_n) - f(y(t_n), t_n) = 0.$$

Probabilistic numerical ODE solvers

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

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

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

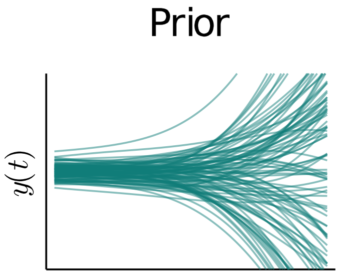
- **Prior:** $y(t) \sim \mathcal{GP}$ a Gauss–Markov process
- **Likelihood:** (aka "observation model" or "information operator")

$$y(0) - y_0 = 0, \qquad \& \qquad \dot{y}(t_n) - f(y(t_n), t_n) = 0.$$

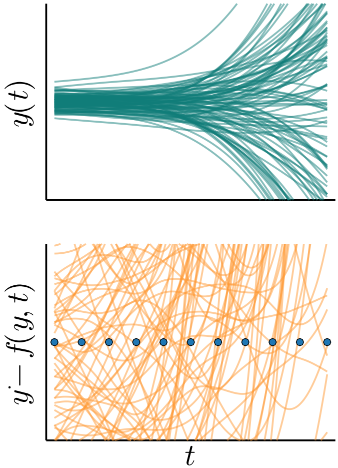
- **Inference:** Bayesian filtering and smoothing
  Kalman filter, extended Kalman filter, unscented Kalman filter, particle filters, ...

Probabilistic numerical ODE solvers in pictures

From the uninformed prior to the ODE solution posterior

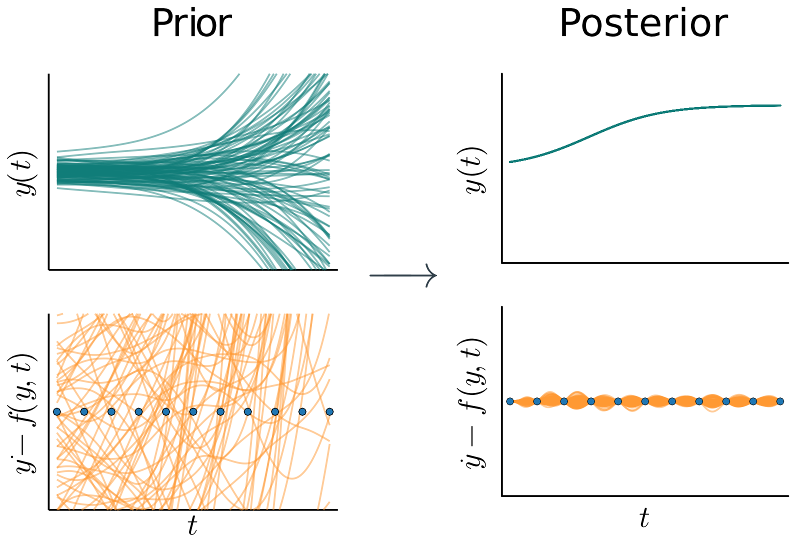


Prior



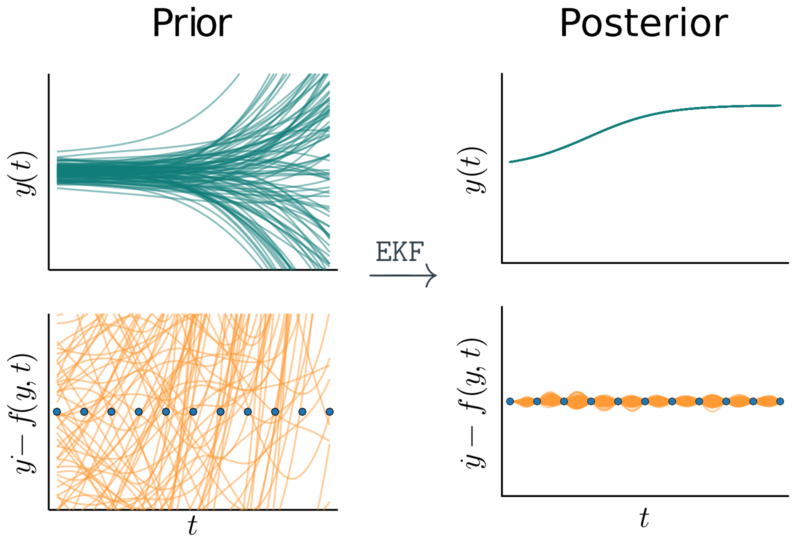
Probabilistic numerical ODE solvers in pictures

From the uninformed prior to the ODE solution posterior



Probabilistic numerical ODE solvers in pictures

From the uninformed prior to the ODE solution posterior



Probabilistic numerical ODE solvers in pseudo-code

We can solve ODEs with basically just an extended Kalman filter

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Algorithm The extended Kalman ODE filter

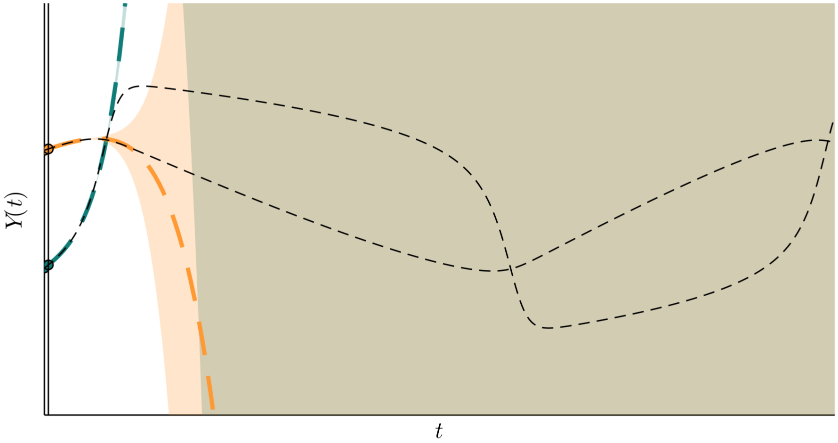
```
procedure EXTENDED KALMAN ODE FILTER((μ_0^-, Σ_0^-), (A, Q), (f, y_0), {t_i}_{i=1}^N)
    μ_0, Σ_0 ← KF_UPDATE(μ_0^-, Σ_0^-, E_0, 0_{d×d}, y_0)        // Initial update to fit the initial value
    for k ∈ {1, ..., N} do
        h_k ← t_k − t_{k−1}                                        // Step size
        μ_k^-, Σ_k^- ← KF_PREDICT(μ_{k−1}, Σ_{k−1}, A(h_k), Q(h_k)) // Kalman filter prediction
        m_k(x) := E_1 x − f(E_0 x, t_k)                             // Define the non-linear observation model
        μ_k, Σ_k ← EKF_UPDATE(μ_k^-, Σ_k^-, m_k, 0_{d×d}, 0⃗_d)      // Extended Kalman filter update
    end for
    return (μ_k, Σ_k)_{k=1}^N
end procedure
```

EXTENDED KALMAN ODE SMOOTHER: Just run a RTS smoother after the filter!

`https://github.com/nathanaelbosch/probnumspringschool2024-tutorial`

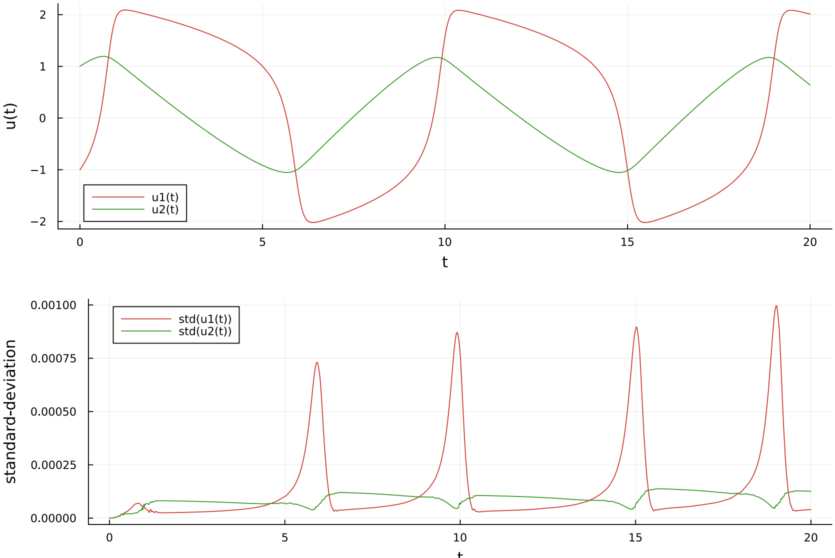
Probabilistic numerical ODE solvers in action

EBERHARD KARLS UNIVERSITÄT TÜBINGEN



Probabilistic numerical ODE solutions

The solution now contains error estimates!



The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]

The state of filtering-based probabilistic numerical ODE solvers

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]

The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]

The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]

The state of filtering-based probabilistic numerical ODE solvers

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]

The state of filtering-based probabilistic numerical ODE solvers

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]
  - Step-size adaptation [Bosch et al., 2021]

The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]
  - Step-size adaptation [Bosch et al., 2021]
  - Parallel-in-time formulation with $\mathcal{O}(\log(N))$ complexity [Bosch et al., 2023a]

The state of filtering-based probabilistic numerical ODE solvers

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]
  - Step-size adaptation [Bosch et al., 2021]
  - Parallel-in-time formulation with $\mathcal{O}(\log(N))$ complexity [Bosch et al., 2023a]
- More related differential equation problems:
  - Higher-order ODEs, DAEs, Hamiltonian systems [Bosch et al., 2022]
  - Boundary value problems (BVPs) [Krämer and Hennig, 2021]
  - Partial differential equations (PDEs) via method of lines [Krämer et al., 2022]

The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]
  - Step-size adaptation [Bosch et al., 2021]
  - Parallel-in-time formulation with $\mathcal{O}(\log(N))$ complexity [Bosch et al., 2023a]
- More related differential equation problems:
  - Higher-order ODEs, DAEs, Hamiltonian systems [Bosch et al., 2022]
  - Boundary value problems (BVPs) [Krämer and Hennig, 2021]
  - Partial differential equations (PDEs) via method of lines [Krämer et al., 2022]
- Inverse problems
  - Probabilistic numerics-based parameter inference in ODEs [Kersting et al., 2020a, Tronarp et al., 2022, Beck et al., 2024]
  - Efficient inference of time-varying latent forces [Schmidt et al., 2021]

The state of filtering-based probabilistic numerical ODE solvers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Properties and features:
  - Polynomial convergence rates [Kersting et al., 2020b, Tronarp et al., 2021]
  - A-stability [Tronarp et al., 2019]
  - L-stable probabilistic exponential integrators [Bosch et al., 2023b]
  - Connection to multi-step methods in Nordsieck form [Schober et al., 2019]
  - Complexity: $\mathcal{O}(d^3)$ for the A-stable semi-implicit method,
    $\mathcal{O}(d)$ for an explicit version with coarser covariances [Krämer et al., 2022]
  - Step-size adaptation [Bosch et al., 2021]
  - Parallel-in-time formulation with $\mathcal{O}(\log(N))$ complexity [Bosch et al., 2023a]
- More related differential equation problems:
  - Higher-order ODEs, DAEs, Hamiltonian systems [Bosch et al., 2022]
  - Boundary value problems (BVPs) [Krämer and Hennig, 2021]
  - Partial differential equations (PDEs) via method of lines [Krämer et al., 2022]
- Inverse problems
  - Probabilistic numerics-based parameter inference in ODEs [Kersting et al., 2020a, Tronarp et al., 2022, Beck et al., 2024]
  - Efficient inference of time-varying latent forces [Schmidt et al., 2021]

Probabilistic Numerics: Computation as Machine Learning

Philipp Hennig, Michael A. Osborne, Hans P. Kersting, 2022

ProbNumDiffEq.jl

Probabilistic numerical ODE solvers in Julia

How to use ProbNumDiffEq.jl

It's just like OrdinaryDiffEq.jl

OrdinaryDiffEq.jl

```julia
using OrdinaryDiffEq

function fitzhughnagumo(du, u, p, t)
    a, b, c = p
    x, y = u
    du[1] = c * (x - x^3 / 3 + y)
    du[2] = -(1/c) * (x - a - b * y)
end
u0 = [-1.0, 1.0]
tspan = (0.0, 20.0)
p = (0.2, 0.2, 3.0)
prob = ODEProblem(f, u0, tspan, p)

sol = solve(prob, Tsit5())
```

How to use ProbNumDiffEq.jl

It's just like OrdinaryDiffEq.jl

OrdinaryDiffEq.jl

```julia
using OrdinaryDiffEq

function fitzhughnagumo(du, u, p, t)
    a, b, c = p
    x, y = u
    du[1] = c * (x - x^3 / 3 + y)
    du[2] = -(1/c) * (x - a - b * y)
end
u0 = [-1.0, 1.0]
tspan = (0.0, 20.0)
p = (0.2, 0.2, 3.0)
prob = ODEProblem(f, u0, tspan, p)

sol = solve(prob, Tsit5())
```

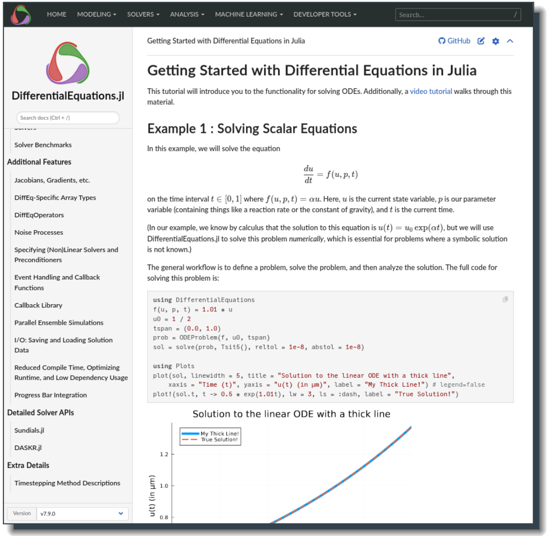
ProbNumDiffEq.jl

```julia
using ProbNumDiffEq

function fitzhughnagumo(du, u, p, t)
    a, b, c = p
    x, y = u
    du[1] = c * (x - x^3 / 3 + y)
    du[2] = -(1/c) * (x - a - b * y)
end
u0 = [-1.0, 1.0]
tspan = (0.0, 20.0)
p = (0.2, 0.2, 3.0)
prob = ODEProblem(f, u0, tspan, p)

sol = solve(prob, EK1())
```

Documentation


HOME MODELING SOLVERS ANALYSIS MACHINE LEARNING DEVELOPER TOOLS

DifferentialEquations.jl

Solver Benchmarks

Additional Features

Jacobians, Gradients, etc.

DiffEq-Specific Array Types

DiffEqOperators

Noise Processes

Specifying (Non)Linear Solvers and Preconditioners

Event Handling and Callback Functions

Callback Library

Parallel Ensemble Simulations

I/O: Saving and Loading Solution Data

Reduced Compile Time, Optimizing Runtime, and Low Dependency Usage

Progress Bar Integration

Detailed Solver APIs

Sundials.jl

DASKR.jl

Extra Details

Timestepping Method Descriptions

Version v7.9.0

Getting Started with Differential Equations in Julia

Getting Started with Differential Equations in Julia

This tutorial will introduce you to the functionality for solving ODEs. Additionally, a video tutorial walks through this material.

Example 1 : Solving Scalar Equations

In this example, we will solve the equation

$$\frac{du}{dt} = f(u,p,t)$$

on the time interval $t \in [0,1]$ where $f(u,p,t) = \alpha u$. Here, u is the current state variable, p is our parameter variable (containing things like a reaction rate or the constant of gravity), and t is the current time.

(In our example, we know by calculus that the solution to this equation is $u(t) = u_0 \exp(\alpha t)$, but we will use DifferentialEquations.jl to solve this problem *numerically*, which is essential for problems where a symbolic solution is not known.)

The general workflow is to define a problem, solve the problem, and then analyze the solution. The full code for solving this problem is:

```
using DifferentialEquations
f(u, p, t) = 1.01 * u
u0 = 1 / 2
tspan = (0.0, 1.0)
prob = ODEProblem(f, u0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-8, abstol = 1e-8)

using Plots
plot(sol, linewidth = 5, title = "Solution to the linear ODE with a thick line",
    xaxis = "Time (t)", yaxis = "u(t) (in μm)", label = "My Thick Line!") # legend=false
plot!(sol.t, t -> 0.5 * exp(1.01t), lw = 3, ls = :dash, label = "True Solution!")
```

Documentation

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

HOME MODELING ▾ SOLVERS ▾ ANALYSIS ▾ MACHINE LEARNING ▾ DEVELOPER TOOLS ▾ Search...

DifferentialEquations.jl

Search docs (Ctrl + /)

Version v7.9.0

Getting Started with Differential Equations in Julia GitHub

Getting Started with Differential Equations in Julia

This tutorial will introduce you to the functionality for solving ODEs. Additionally, a video tutorial walks through this material.

Example 1 : Solving Scalar Equations

In this example, we will solve the equation

$$\frac{du}{dt} = f(u,p,t)$$

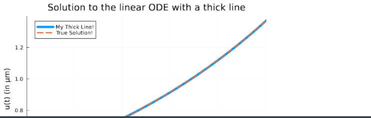
on the time interval $t \in [0,1]$ where $f(u,p,t) = \alpha u$. Here, u is the current state variable, p is our parameter variable (containing things like a reaction rate or the constant of gravity), and t is the current time.

(In our example, we know by calculus that the solution to this equation is $u(t) = u_0 \exp(\alpha t)$, but we will use DifferentialEquations.jl to solve this problem *numerically*, which is essential for problems where a symbolic solution is not known.)

The general workflow is to define a problem, solve the problem, and then analyze the solution. The full code for solving this problem is:

```
using DifferentialEquations
f(u, p, t) = 1.01 * u
u0 = 1 / 2
tspan = (0.0, 1.0)
prob = ODEProblem(f, u0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-8, abstol = 1e-8)

using Plots
plot(sol, linewidth = 5, title = "Solution to the linear ODE with a thick line",
    xaxis = "Time (t)", yaxis = "u(t) (in μm)", label = "My Thick Line!") # legend=false
plot!(sol.t, t -> 0.5 * exp(1.01t), lw = 3, ls = :dash, label = "True Solution!")
```



ProbNumDiffEq.jl

Search docs (Ctrl + /)

Version v0.16.0

Tutorials / Getting Started GitHub

Solving ODEs with Probabilistic Numerics

In this tutorial we solve a simple non-linear ordinary differential equation (ODE) with the *probabilistic numerical* ODE solvers implemented in this package.

Note

If you never used DifferentialEquations.jl, check out their "Getting Started with Differential Equations in Julia" tutorial. It explains how to define and solve ODE problems and how to analyze the solution, so it's a great starting point. Most of ProbNumDiffEq.jl works exactly as you would expect from DifferentialEquations.jl – just with some added uncertainties and related functionality on top!

In this tutorial, we consider a Fitzhugh-Nagumo model described by an ODE of the form

$$\dot{y}_1 = c\left(y_1 - \frac{y_1^3}{3} + y_2\right)$$
$$\dot{y}_2 = -\frac{1}{c}\left(y_1 - a - by_2\right)$$

on a time span $t \in [0, T]$, with initial value $y(0) = y_0$. In the following, we

1. define the problem with explicit choices of initial values, integration domains, and parameters,
2. solve the problem with our ODE filters, and
3. visualize the results and the corresponding uncertainties.

TL;DR: Just use DifferentialEquations.jl with the `EK1` algorithm

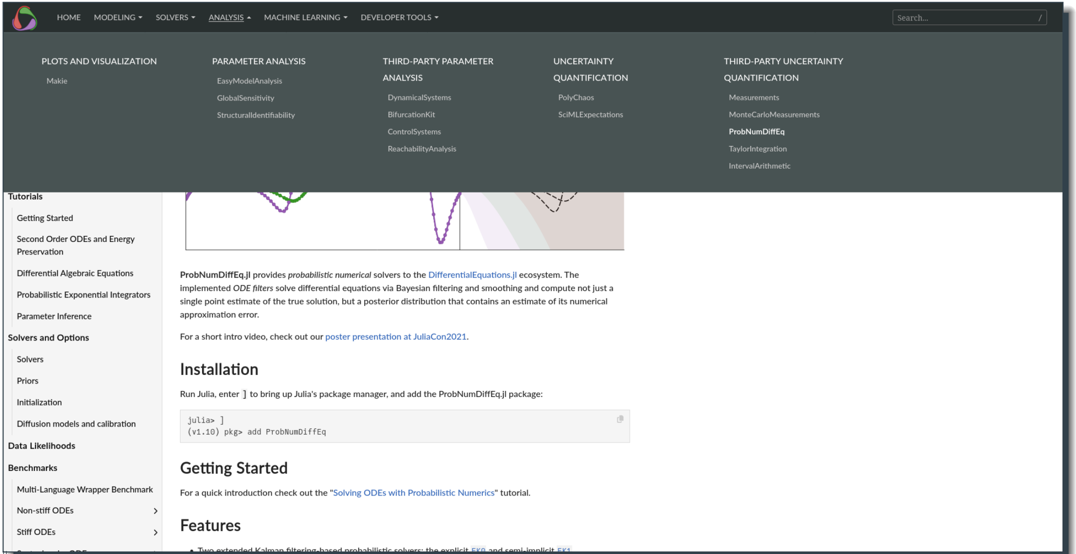
```
using ProbNumDiffEq, Plots

function fitz(du, u, p, t)
    a, b, c = p
    du[1] = c * (u[1] - u[1]^3 / 3 + u[2])
    du[2] = -(1 / c) * (u[1] - a - b * u[2])
end
u0 = [-1.0; 1.0]
tspan = (0.0, 20.0)
p = (0.2, 0.2, 3.0)
prob = ODEProblem(fitz, u0, tspan, p)

sol = solve(prob, EK1())
plot(sol)
```

Documentation

SciML's SEO score outperforms my own docs



HOME MODELING ▾ SOLVERS ▾ ANALYSIS ▴ MACHINE LEARNING ▾ DEVELOPER TOOLS ▾

Search... /

PLOTS AND VISUALIZATION
- Makie

PARAMETER ANALYSIS
- EasyModelAnalysis
- GlobalSensitivity
- StructuralIdentifiability

THIRD-PARTY PARAMETER ANALYSIS
- DynamicalSystems
- BifurcationKit
- ControlSystems
- ReachabilityAnalysis

UNCERTAINTY QUANTIFICATION
- PolyChaos
- SciMLExpectations

THIRD-PARTY UNCERTAINTY QUANTIFICATION
- Measurements
- MonteCarloMeasurements
- **ProbNumDiffEq**
- TaylorIntegration
- IntervalArithmetic

Tutorials
- Getting Started
- Second Order ODEs and Energy Preservation
- Differential Algebraic Equations
- Probabilistic Exponential Integrators
- Parameter Inference

Solvers and Options
- Solvers
- Priors
- Initialization
- Diffusion models and calibration

Data Likelihoods

Benchmarks
- Multi-Language Wrapper Benchmark
- Non-stiff ODEs ›
- Stiff ODEs ›

ProbNumDiffEq.jl provides *probabilistic numerical* solvers to the DifferentialEquations.jl ecosystem. The implemented *ODE filters* solve differential equations via Bayesian filtering and smoothing and compute not just a single point estimate of the true solution, but a posterior distribution that contains an estimate of its numerical approximation error.

For a short intro video, check out our poster presentation at JuliaCon2021.

Installation

Run Julia, enter] to bring up Julia's package manager, and add the ProbNumDiffEq.jl package:

```
julia> ]
(v1.10) pkg> add ProbNumDiffEq
```

Getting Started

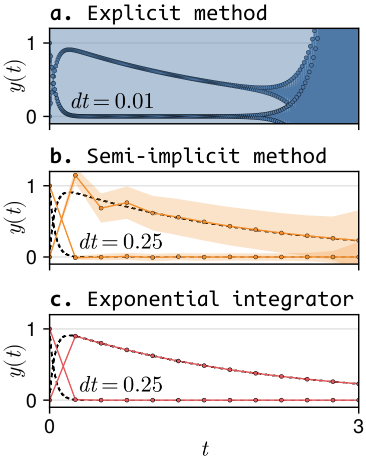
For a quick introduction check out the "Solving ODEs with Probabilistic Numerics" tutorial.

Features

Features of ProbNumDiffEq.jl

Standard ODE solver features

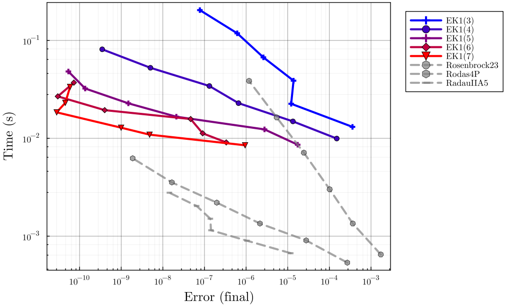
- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`



EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Standard ODE solver features

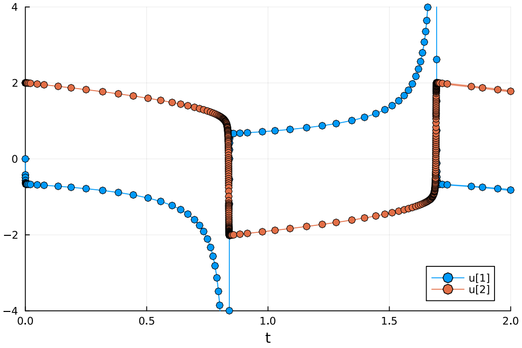
- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders:
  `EK0(1)`, `EK0(2)`, `EK0(3)`, …



Features of ProbNumDiffEq.jl

Standard ODE solver features

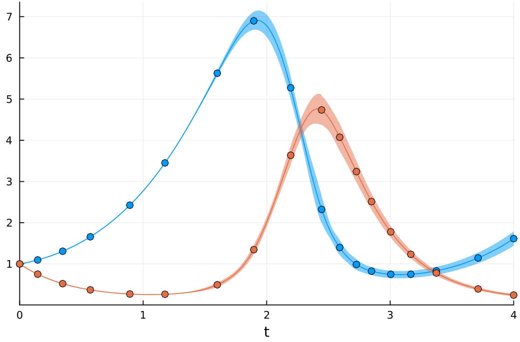
- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders:
  `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation:
  Same controllers as OrdinaryDiffEq.jl



Features of ProbNumDiffEq.jl

Standard ODE solver features

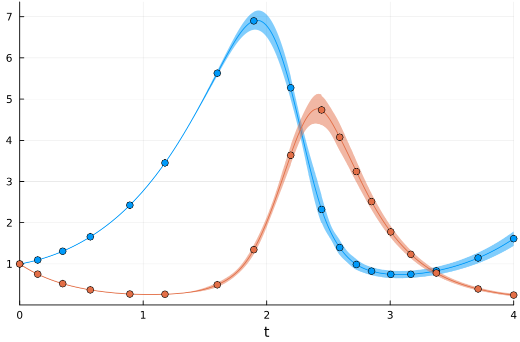
- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders:
  `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation:
  Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output



Features of ProbNumDiffEq.jl

Standard ODE solver features

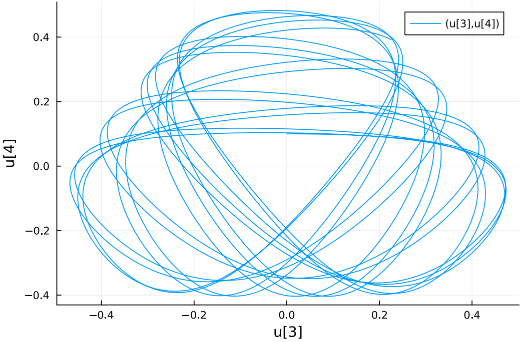
- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders:
  `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation:
  Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output
- ☒ Plot recipes



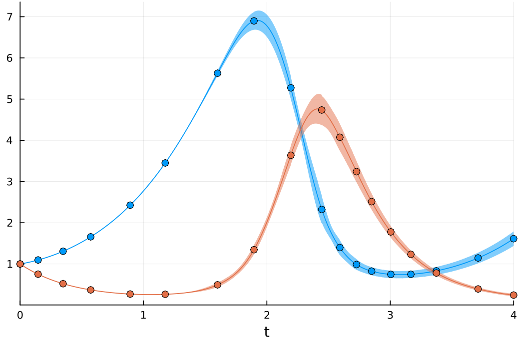
Features of ProbNumDiffEq.jl

Standard ODE solver features

- ☒ Explicit and implicit solvers: `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders: `EK0(1)`, `EK0(2)`, `EK0(3)`, ...
- ☒ Step-size adaptation: Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output
- ☒ Plot recipes
- ☒ Callbacks (including a custom `ManifoldUpdate` callback)



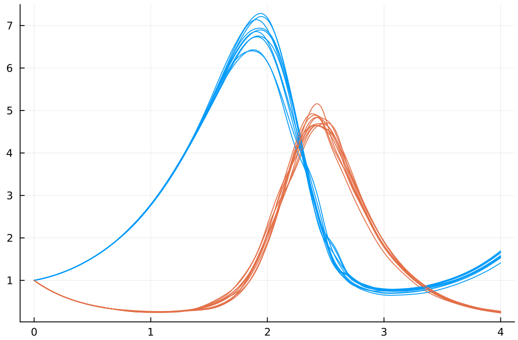
Features of ProbNumDiffEq.jl



Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)

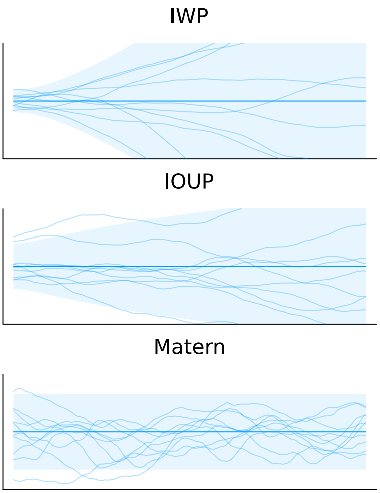
Features of ProbNumDiffEq.jl



Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)
- ☒ Sampling from the posterior

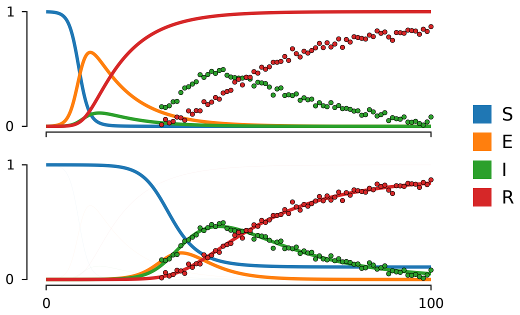
Features of ProbNumDiffEq.jl



Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)
- ☒ Sampling from the posterior
- ☒ Multiple different prior choices

Features of ProbNumDiffEq.jl



Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)
- ☒ Sampling from the posterior
- ☒ Multiple different prior choices
- ☒ Probabilistic data likelihoods (for parameter inference problems)

Features of ProbNumDiffEq.jl

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Standard ODE solver features

- ☒ Explicit and implicit solvers:
  `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders:
  `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation:
  Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output
- ☒ Plot recipes
- ☒ Callbacks (including a custom `ManifoldUpdate` callback)
- ☐ Support for `DAEProblem`
- ☐ Adjoint sensitivities

Probabilistic numerics-related features

- ☒ Numerical error estimates
  (shown by the plot recipe!)
- ☒ Sampling from the posterior
- ☒ Multiple different prior choices
- ☒ Probabilistic data likelihoods
  (for parameter inference problems)

Features of ProbNumDiffEq.jl

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Standard ODE solver features

- ☒ Explicit and implicit solvers: `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders: `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation: Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output
- ☒ Plot recipes
- ☒ Callbacks (including a custom `ManifoldUpdate` callback)
- ☐ Support for `DAEProblem`
- ☐ Adjoint sensitivities

Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)
- ☒ Sampling from the posterior
- ☒ Multiple different prior choices
- ☒ Probabilistic data likelihoods (for parameter inference problems)
- ☐ Other filtering algorithms: UKF, Cubature filters, particle filters…
- ☐ Custom prior interface
- ☐ Latent force inference
- ☐ Parallel-in-time solver (using the time-parallel iterated extended Kalman smoother)

Features of ProbNumDiffEq.jl

Standard ODE solver features

- ☒ Explicit and implicit solvers: `EK0`, `EK1`, `ExpEK`, `RosenbrockExpEK`
- ☒ Solvers of different orders: `EK0(1)`, `EK0(2)`, `EK0(3)`, …
- ☒ Step-size adaptation: Same controllers as OrdinaryDiffEq.jl
- ☒ Dense output
- ☒ Plot recipes
- ☒ Callbacks (including a custom `ManifoldUpdate` callback)
- ☐ Support for `DAEProblem`
- ☐ Adjoint sensitivities

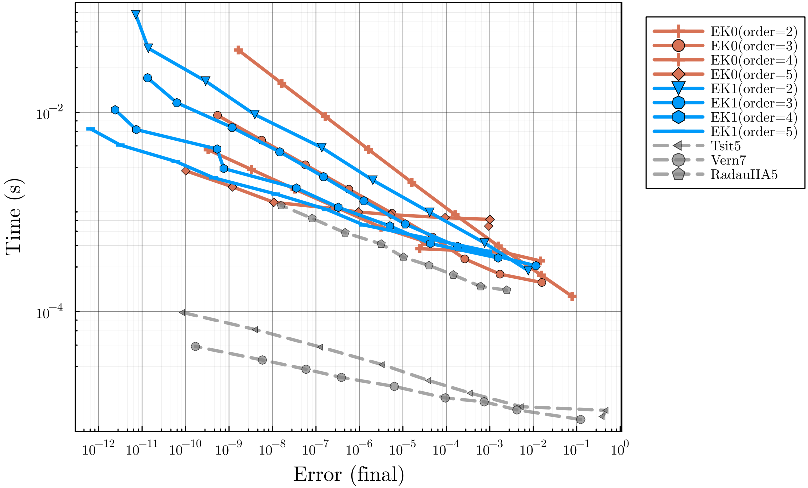
Probabilistic numerics-related features

- ☒ Numerical error estimates (shown by the plot recipe!)
- ☒ Sampling from the posterior
- ☒ Multiple different prior choices
- ☒ Probabilistic data likelihoods (for parameter inference problems)
- ☐ Other filtering algorithms: UKF, Cubature filters, particle filters…
- ☐ Custom prior interface
- ☐ Latent force inference
- ☐ Parallel-in-time solver (using the time-parallel iterated extended Kalman smoother)

Benchmarking ProbNumDiffEq.jl

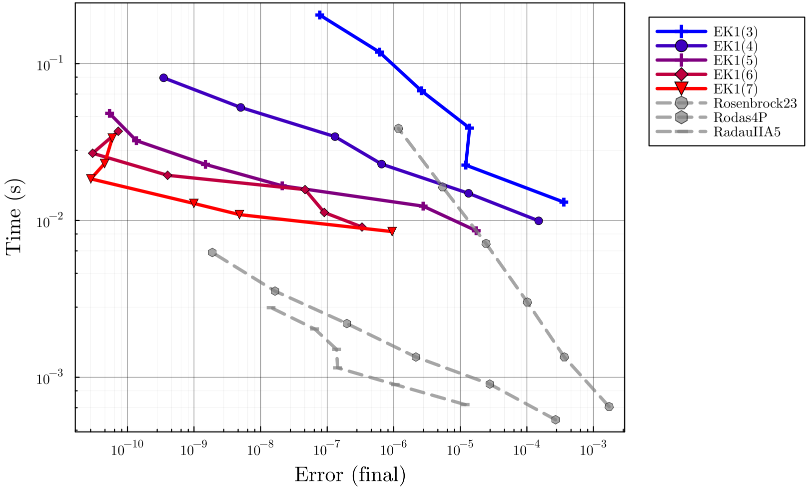
Benchmarks: Low-dimensional non-stiff ODE (Lotka-Volterra)

100x slower than Tsit5



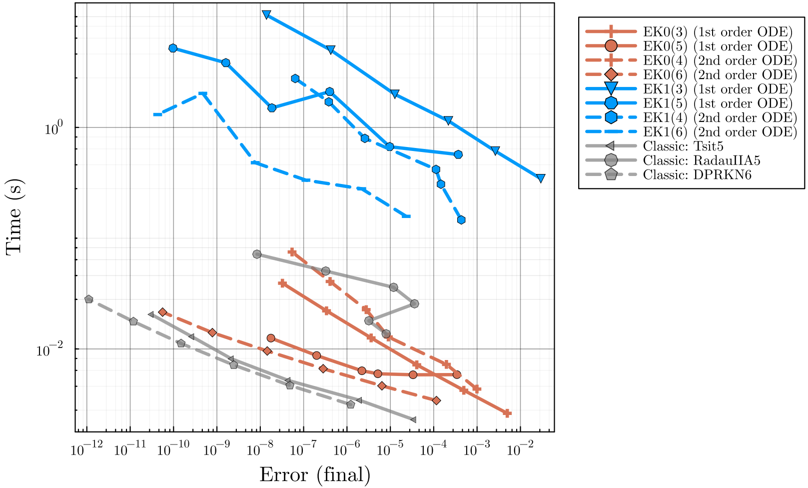
Benchmarks: Low-dimensional stiff ODE (Van-der-Pol)

10x slower than RadauIIA5



Benchmarks: Medium-dimensional non-stiff ODE (Pleiades)

Same ballpark as Tsit5 !

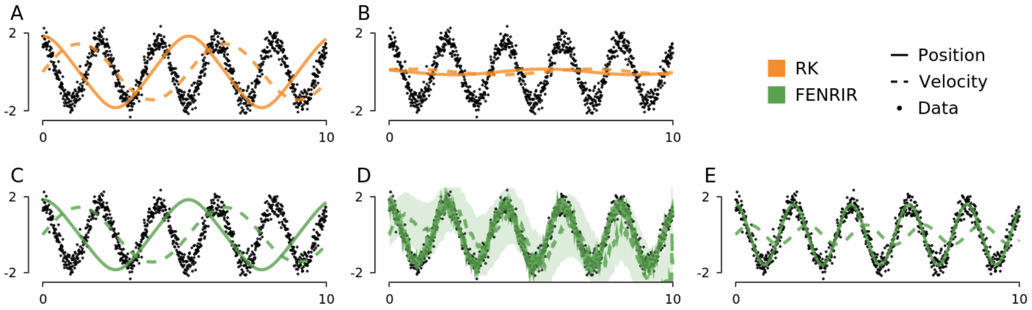


Beyond numerical uncertainty quantification

Probabilistic numerics for robust ODE parameter inference

Robust parameter inference in ODEs with ProbNumDiffEq.jl

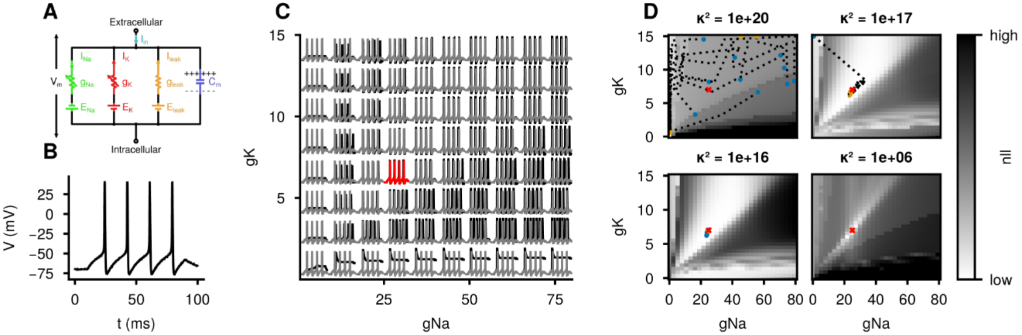
Filtering and smoothing often helps to escape local optima in oscillatory systems



[Tronarp et al., 2022]

Robust parameter inference in ODEs with ProbNumDiffEq.jl

Filtering and smoothing often helps to escape local optima in oscillatory systems



[Beck et al., 2024]

Robust parameter inference in ODEs with ProbNumDiffEq.jl

Filtering and smoothing often helps to escape local optima in oscillatory systems

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Parameter Inference with ProbNumDiffEq.jl

Let's assume we have an initial value problem (IVP)

$$\dot{y} = f_\theta(y, t), \qquad y(t_0) = y_0,$$

which we observe through a set $\mathcal{D} = \{u(t_n)\}_{n=1}^N$ of noisy data points

$$u(t_n) = Hy(t_n) + v_n, \qquad v_n \sim \mathcal{N}(0, R).$$

The question of interest is: How can we compute the marginal likelihood $p(\mathcal{D} \mid \theta)$? Short answer: We can't. It's intractable, because computing the true IVP solution exactly $y(t)$ is intractable. What we can do however is compute an approximate marginal likelihood. This is what `ProbNumDiffEq.DataLikelihoods` provides.

The specific problem, in code

Let's assume that the true underlying dynamics are given by a FitzHugh-Nagumo model

```
using ProbNumDiffEq, LinearAlgebra, OrdinaryDiffEq, Plots
Plots.theme(:default; markersize=2, markerstrokewidth=0.1)

function f(du, u, p, t)
    a, b, c = p
    du[1] = c*(u[1] - u[1]^3/3 + u[2])
    du[2] = -(1/c)*(u[1] -  a - b*u[2])
end
u0 = [-1.0, 1.0]
```

%

Summary

- *ODE solving is state estimation* $\Rightarrow$ treat initial value problems as state estimation problems
- ***Probablistic numerical ODE solvers* solve ODEs with Bayesian filtering and smoothing**

Summary

- *ODE solving is state estimation* $\Rightarrow$ treat initial value problems as state estimation problems
- ***Probablistic numerical ODE solvers*** **solve ODEs with Bayesian filtering and smoothing**

Try it out!

```
https://github.com/nathanaelbosch/ProbNumDiffEq.jl
]add ProbNumDiffEq
```

Summary

- *ODE solving is state estimation* ⇒ treat initial value problems as state estimation problems
- ***Probablistic numerical ODE solvers* solve ODEs with Bayesian filtering and smoothing**

Try it out!

`https://github.com/nathanaelbosch/ProbNumDiffEq.jl`

`]add ProbNumDiffEq`

Contribute!

- Try out the package and tell me how it goes!
- Open issues, report bugs, give feedback on the package design
- Help me improve performance / AD backend compatibility / GPU support / add features...
- Tell me about your usecase or show me an example!
- Design a logo!

Summary

- *ODE solving is state estimation* ⇒ treat initial value problems as state estimation problems
- ***Probablistic numerical ODE solvers* solve ODEs with Bayesian filtering and smoothing**

Try it out!

```
https://github.com/nathanaelbosch/ProbNumDiffEq.jl
]add ProbNumDiffEq
```

Contribute!

- Try out the package and tell me how it goes!
- Open issues, report bugs, give feedback on the package design
- Help me improve performance / AD backend compatibility / GPU support / add features...
- Tell me about your usecase or show me an example!
- Design a logo!

Thanks!

Bibliography I

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Beck, J., Bosch, N., Deistler, M., Kadhim, K. L., Macke, J. H., Hennig, P., and Berens, P. (2024). Diffusion tempering improves parameter estimation with probabilistic integrators for ordinary differential equations. In *Forty-first International Conference on Machine Learning*.
- Bosch, N., Corenflos, A., Yaghoobi, F., Tronarp, F., Hennig, P., and Särkkä, S. (2023a). Parallel-in-time probabilistic numerical ODE solvers.
- Bosch, N., Hennig, P., and Tronarp, F. (2021). Calibrated adaptive probabilistic ODE solvers. In Banerjee, A. and Fukumizu, K., editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3466–3474. PMLR.
- Bosch, N., Hennig, P., and Tronarp, F. (2023b). Probabilistic exponential integrators. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Bibliography II

- Bosch, N., Tronarp, F., and Hennig, P. (2022).
  Pick-and-mix information operators for probabilistic ODE solvers.
  In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10015–10027. PMLR.
- Kersting, H., Krämer, N., Schiegg, M., Daniel, C., Tiemann, M., and Hennig, P. (2020a).
  Differentiable likelihoods for fast inversion of 'Likelihood-free' dynamical systems.
  In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5198–5208. PMLR.
- Kersting, H., Sullivan, T. J., and Hennig, P. (2020b).
  Convergence rates of gaussian ode filters.
  Statistics and Computing, 30(6):1791–1816.

Bibliography III

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Krämer, N., Bosch, N., Schmidt, J., and Hennig, P. (2022).
  Probabilistic ODE solutions in millions of dimensions.
  In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11634–11649. PMLR.
- Krämer, N. and Hennig, P. (2021).
  Linear-time probabilistic solution of boundary value problems.
  In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11160–11171. Curran Associates, Inc.
- Krämer, N., Schmidt, J., and Hennig, P. (2022).
  Probabilistic numerical method of lines for time-dependent partial differential equations.
  In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 625–639. PMLR.

Bibliography IV

- Schmidt, J., Krämer, N., and Hennig, P. (2021).
  A probabilistic state space model for joint inference from differential equations and data.
  In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12374–12385. Curran Associates, Inc.
- Schober, M., Särkkä, S., and Hennig, P. (2019).
  A probabilistic model for the numerical solution of initial value problems.
  Statistics and Computing, 29(1):99–122.
- Tronarp, F., Bosch, N., and Hennig, P. (2022).
  Fenrir: Physics-enhanced regression for initial value problems.
  In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 21776–21794. PMLR.

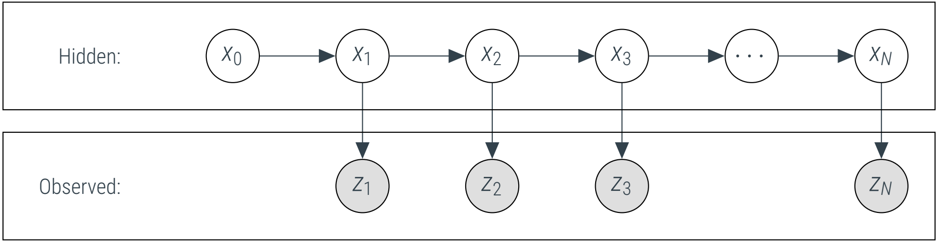
Bibliography V

- Tronarp, F., Kersting, H., Särkkä, S., and Hennig, P. (2019).
  Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: a new perspective.
  Statistics and Computing, 29(6):1297–1315.
- Tronarp, F., Särkkä, S., and Hennig, P. (2021).
  Bayesian ode solvers: the maximum a posteriori estimate.
  Statistics and Computing, 31(3):23.

BACKUP

Probabilistic numerical ODE solvers: The state-estimation problem

This is the actual state estimation problem that we solve



Initial distribution: $$x(0) \sim \mathcal{N}\left(x(0); \mu_0^-, \Sigma_0^-\right)$$

Prior / dynamics model: $$x(t+h) \mid x(t) \sim \mathcal{N}\left(x(t+h); A(h)x(t), Q(h)\right)$$

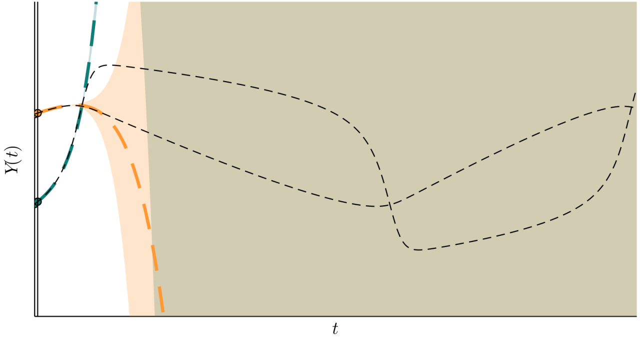
ODE likelihood: $$z(t_i) \mid x(t_i) \sim \delta\left(z(t_i); E_1 x(t_i) - f(E_0 x(t_i), t_i)\right), \qquad z_i \triangleq 0$$

Initial value likelihood: $$z^{\text{init}} \mid x(0) \sim \delta\left(z^{\text{init}}; E_0 x(0) - y_0\right), \qquad z^{\text{init}} \triangleq 0$$

$x(t)$ is the /state-space representation/ of $y(t)$; $E_0 x(t) \triangleq y(t)$, $E_1 x(t) \triangleq \dot{y}(t)$.

Local calibration and step-size adaptation

Fixed steps — the vanilla way as introduced so far

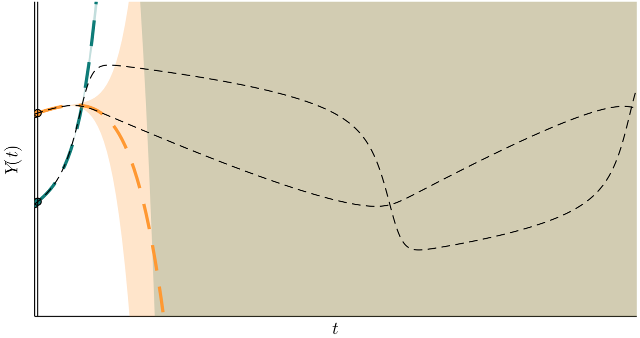


Local calibration and step-size adaptation

Fixed steps – the vanilla way as introduced so far

Calibration

- *Problem*: The Gauss–Markov prior has hyperparameters. How to choose them?
- Most notably: The *diffusion* σ (basically acts as an output scale)



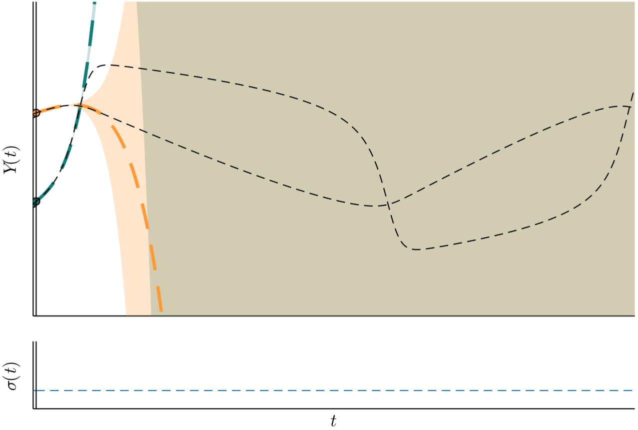
Local calibration and step-size adaptation

Local calibration by estimating a time-varying diffusion model $\sigma(t)$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Calibration

- *Problem*: The Gauss–Markov prior has hyperparameters. How to choose them?
- Most notably: The *diffusion* σ (basically acts as an output scale)
- *Solution*: (Quasi-)MLE (can be done in closed form here)



Local calibration and step-size adaptation

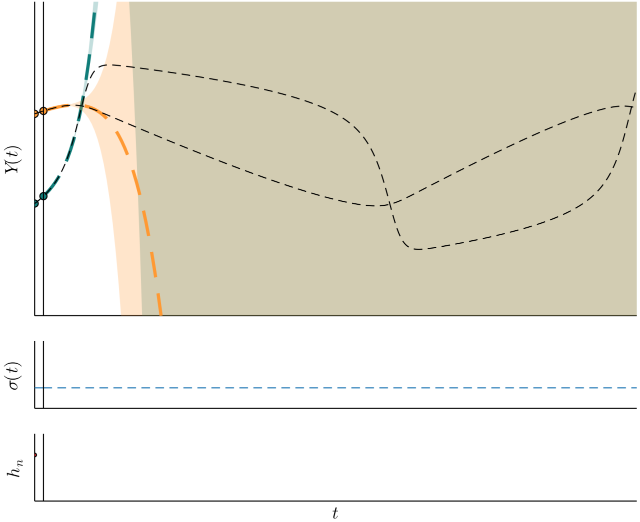
Adaptive step-size selection via local error estimation from the measurement residuals

Calibration

- *Problem*: The Gauss–Markov prior has hyperparameters. How to choose them?
- Most notably: The *diffusion* σ (basically acts as an output scale)
- *Solution*: (Quasi-)MLE (can be done in closed form here)

Step-size adaptation

- Local error estimates from measurement residuals
- Step-size selection with PI-control (similar as in classic solvers)



Prior: The ν-times integrated Wiener process

A very convenient prior with closed-form transition densities

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- **ν-times integrated Wiener process prior**: $x(t) \sim \mathrm{IWP}(q)$

$$
\begin{aligned}
\mathrm{d}x^{(i)}(t) &= x^{(i+1)}(t)\mathrm{d}t, \qquad i = 0, \dots, q-1, \\
\mathrm{d}x^{(q)}(t) &= \sigma \mathrm{d}W(t), \\
x(0) &\sim \mathcal{N}(\mu_0, \Sigma_0).
\end{aligned}
$$

- Corresponds to Taylor-polynomial + perturbation:

$$
x^{(0)}(t) = \sum_{m=0}^{q} x^{(m)}(0) \frac{t^m}{m!} + \sigma \int_0^t \frac{t-\tau}{q!} \mathrm{d}W(\tau)
$$

On linearization strategies and their influence on A-Stability

We can actually approximate the Jacobian in the EKF and still get sensible results / algorithms!

[Tronarp et al., 2019]

- Measurement model: $m(x(t), t) = x^{(1)}(t) - f(x^{(0)}(t), t)$
- A standard extended Kalman filter computes the Jacobian of the measurement mode: $J_m(\xi) = E_1 - J_f(E_0\xi, t)E_0 \setminus \Rightarrow$ This algorithm is often called `EK1`.
- Turns out the following also works: $J_f \approx 0$ and then $J_m(\xi) \approx E_1 \setminus \Rightarrow$ The resulting algorithm is often called `EK0`.

A comparison of `EK1` and `EK0`:

	Jacobian	type	A-stable	uncertainties	speed
`EK1`	$H = E_1 - J_f(E_0\mu^p)E_0$	semi-implicit	yes	more expressive	slower ($O(Nd^3q^3)$)
`EK0`	$H = E_1$	explicit	no	simpler	faster ($O(Ndq^3)$)