ProbNumDiffEq.jl: Fast and Practical ODE Filters in Julia

or “Building a PN library on existing non-PN code”

Nathanael Bosch

26.10.2021

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Max Planck Institute for Intelligent Systems

imprs-is

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

Key Challenges for Probabilistic Numerics

- speed up existing methods to be competitive with classic algorithms
- find *killer applications* of PN that goes beyod the functionality of classic methods

Key Challenges for Probabilistic Numerics

- **speed up existing methods to be competitive with classic algorithms**
- find *killer applications* of PN that goes beyod the functionality of classic methods

Key Challenges for Probabilistic Numerics

- **be competitive with classic algorithms**
  - speed
  - features
  - convenience
- find *killer applications* of PN that goes beyod the functionality of classic methods

Ordinary Differential Equations

- Problem setting: **Initial value problem**

$$\dot{y}(t) = f(y(t), t), \qquad t \in [t_{\mathsf{min}}, t_{\mathsf{max}}], \qquad y(t_{\mathsf{min}}) = y_0. \tag{1}$$

Goal: Approximate the ODE solution $\hat{y} \approx y(t)$.

- Problem setting: **Initial value problem**

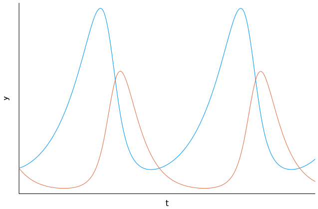
$$\dot{y}(t) = f(y(t), t), \qquad t \in [t_{\text{min}}, t_{\text{max}}], \qquad y(t_{\text{min}}) = y_0. \tag{1}$$

Goal: Approximate the ODE solution $\hat{y} \approx y(t)$.

Code Example: SciPy

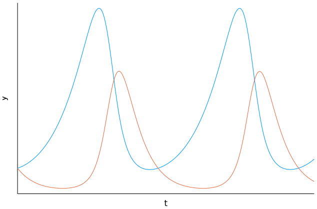
```python
import numpy as np
from scipy.integrate import solve_ivp

def lotkavolterra(t, y):
    y1 =  0.5 * y[0] - 0.05 * y[0] * y[1]
    y2 = -0.5 * y[1] + 0.05 * y[0] * y[1]
    return np.array([y1, y2])

tspan = [0.0, 20.0]
y0 = np.array([20, 20])
sol = solve_ivp(lotkavolterra, tspan, y0, method="RK45")
```

ODEs come in various forms

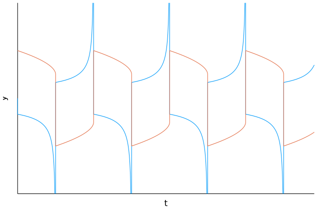


(a) Lotka-Volterra (non-stiff)

ODEs come in various forms

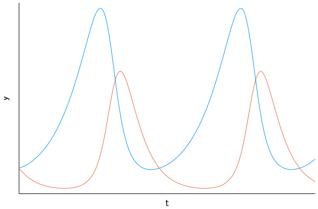


(a) Lotka-Volterra (non-stiff)

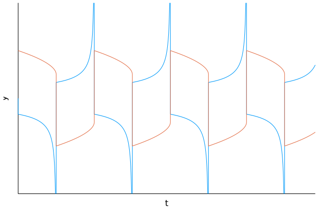


(b) Van der Pol (stiff)

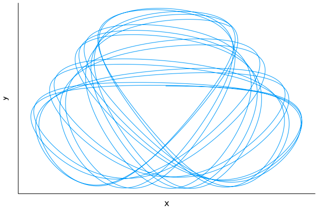
ODEs come in various forms



(a) Lotka-Volterra (non-stiff)



(b) Van der Pol (stiff)



(c) Henon-Heiles (second-order and energy preserving)

Solving ODEs in practice requires making choices

Algorithmic choices:

- Explicit or implicit solver? Runge–Kutta or multi-step? What order?
- Step-size adaptation or fixed steps? What accuracy?
- Higher-order ODE? Symplectic solver?

Solving ODEs in practice requires making choices

Algorithmic choices:

- Explicit or implicit solver? Runge–Kutta or multi-step? What order?
- Step-size adaptation or fixed steps? What accuracy?
- Higher-order ODE? Symplectic solver?

More features:

- Output control: Time-series or final value only? Dense output?
- Number type: Float32 or Float64? Arbitrary precision? Complex numbers?
- Taking derivatives: Discrete or continuous sensitivities? Forward or backward-mode?

Solving ODEs in practice requires making choices

Algorithmic choices:

- Explicit or implicit solver? Runge–Kutta or multi-step? What order?
- Step-size adaptation or fixed steps? What accuracy?
- Higher-order ODE? Symplectic solver?

More features:

- Output control: Time-series or final value only? Dense output?
- Number type: Float32 or Float64? Arbitrary precision? Complex numbers?
- Taking derivatives: Discrete or continuous sensitivities? Forward or backward-mode?

Existing software:

- SciPy
- MATLAB
- `deSolve` (R)
- Multiple Fortran libraries
- torchdiffeq
- jax
- DifferentialEquations.jl

Solving ODEs in practice requires making choices

Algorithmic choices:

- **Explicit or implicit solver?** Runge–Kutta or multi-step? **What order?**
- **Step-size adaptation or fixed steps? What accuracy?** [Bosch et al., 2021a]
- **Higher-order ODE? Symplectic solver?** [Bosch et al., 2021b]

More features:

- **Output control: Time-series or final value only? Dense output?**
- **Number type: Float32 or Float64? Arbitrary precision?** Complex numbers?
- Taking derivatives: **Discrete** or continuous sensitivities? **Forward** or backward-mode?

Existing software:

- SciPy
- MATLAB
- `deSolve` (R)
- Multiple Fortran libraries
- torchdiffeq
- jax
- DifferentialEquations.jl
- ProbNumDiffEq.jl

What is Julia?

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

julia

Download Documentation Blog Community Learn Research JSoC

Sponsor

The Julia Programming Language

Download Documentation

Star 36,756

Julia in a Nutshell

Fast

Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

Dynamic

Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.

Reproducible

Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable

Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General

Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

Open source

Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.

Why DifferentialEquations.jl?

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Comparison Of Differential Equation Solver Software

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK/Netlib /NAG	JitCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Fair	Excellent	Good	Fair	Good	Poor	Poor	Fair	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor****	Poor***	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	None	Good	Fair
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DASPK)	None	Poor	Good	None	None	Excellent	None

Why DifferentialEquations.jl?

DifferentialEquations.jl [Rackauckas and Nie, 2017]:

- $>$50 ($>$150?) available solvers (non-stiff, stiff, secondorder, exponential, symplectic, ...)
- ODEs, DAEs, SDEs, DDEs, BVPs, ...
- Wide range of (continuous & discrete) sensitivity analysis options [Rackauckas et al., 2018]
- Interacts well with other parts of the Julia ecosystem:
  - AD / Jacobians via ForwardDiff.jl, ReverseDiff.jl, Zygote.jl, Enzyme.jl, ...
  - NeuralODEs with Flux.jl
  - Probabilistic programming with Turing.jl

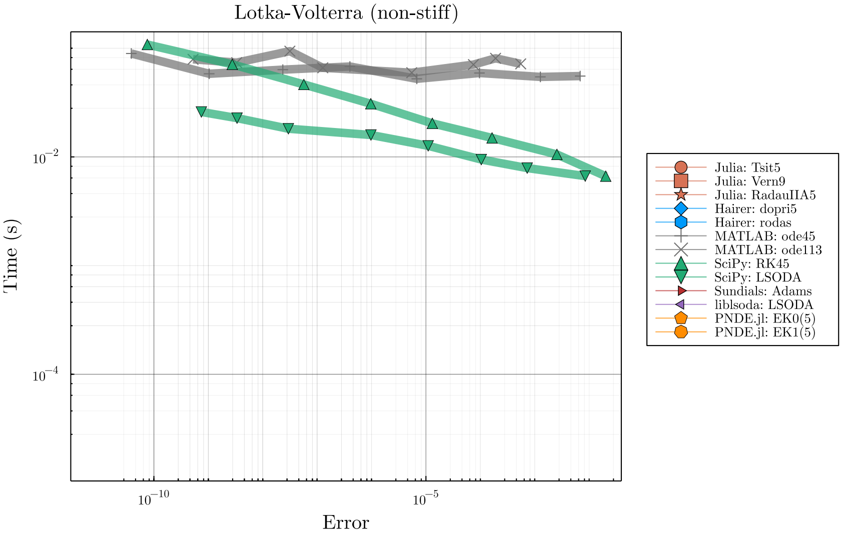
Why DifferentialEquations.jl?

DifferentialEquations.jl [Rackauckas and Nie, 2017]:

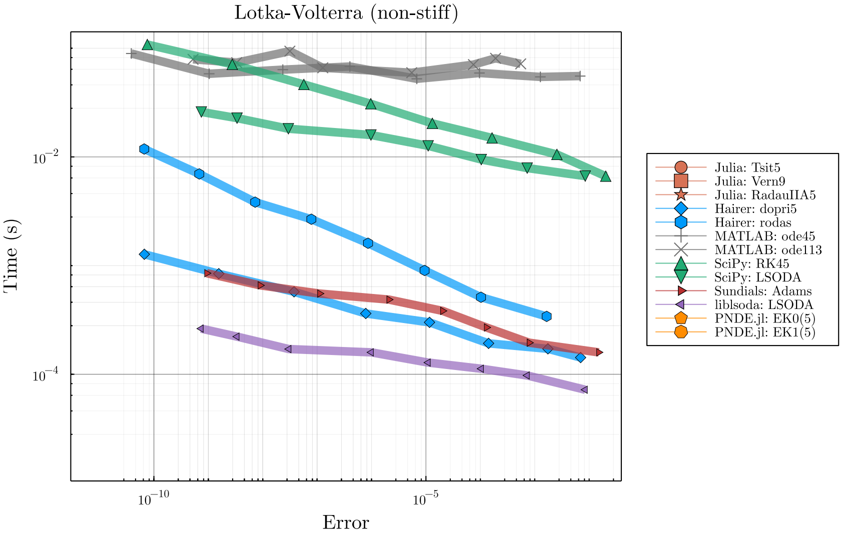
- $>$50 ($>$150?) available solvers (non-stiff, stiff, secondorder, exponential, symplectic, ...)
- ODEs, DAEs, SDEs, DDEs, BVPs, ...
- Wide range of (continuous & discrete) sensitivity analysis options [Rackauckas et al., 2018]
- Interacts well with other parts of the Julia ecosystem:
  - AD / Jacobians via ForwardDiff.jl, ReverseDiff.jl, Zygote.jl, Enzyme.jl, ...
  - NeuralODEs with Flux.jl
  - Probabilistic programming with Turing.jl
- Modular implementation and easy to extend
  - Core ODE solvers in OrdinaryDiffEq.jl
  - Specific solver contributions e.g. in GeometricIntegrators.jl or TaylorIntegration.jl
  - **ODE Filters: ProbNumDiffEq.jl**

Demo time

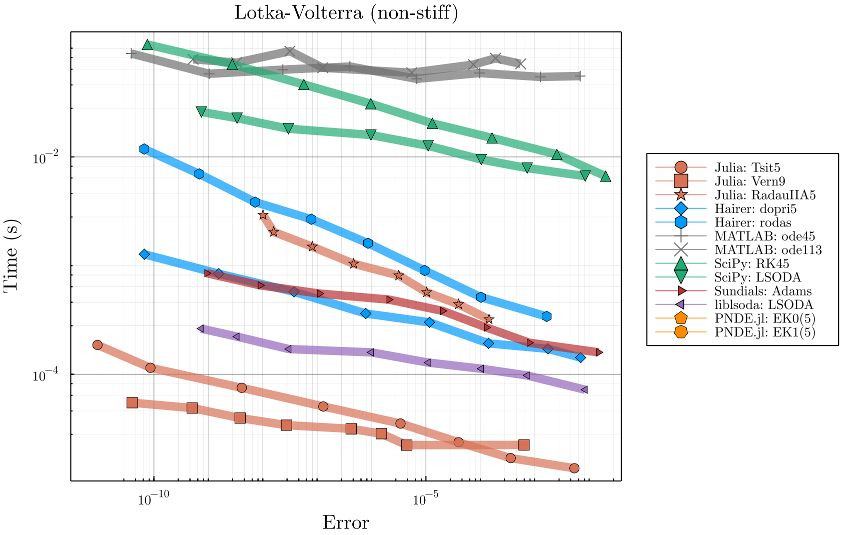
Benchmark (non-stiff)



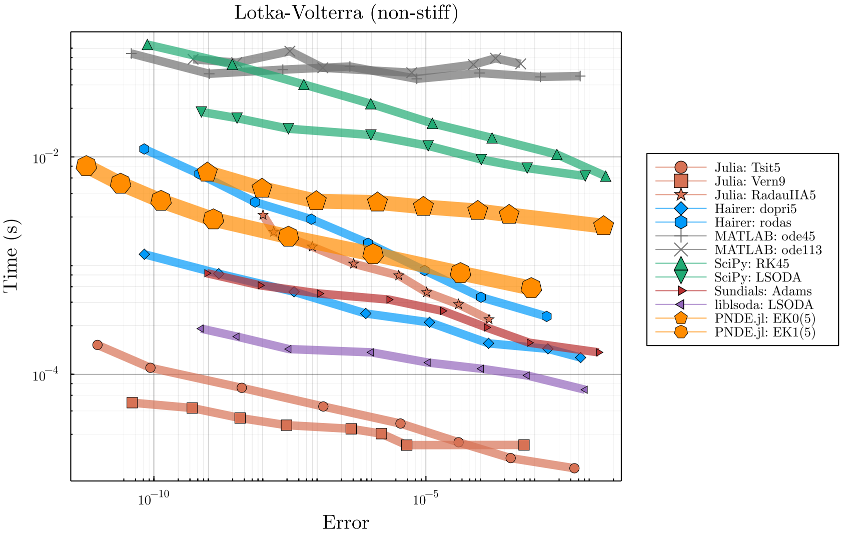
Benchmark (non-stiff)



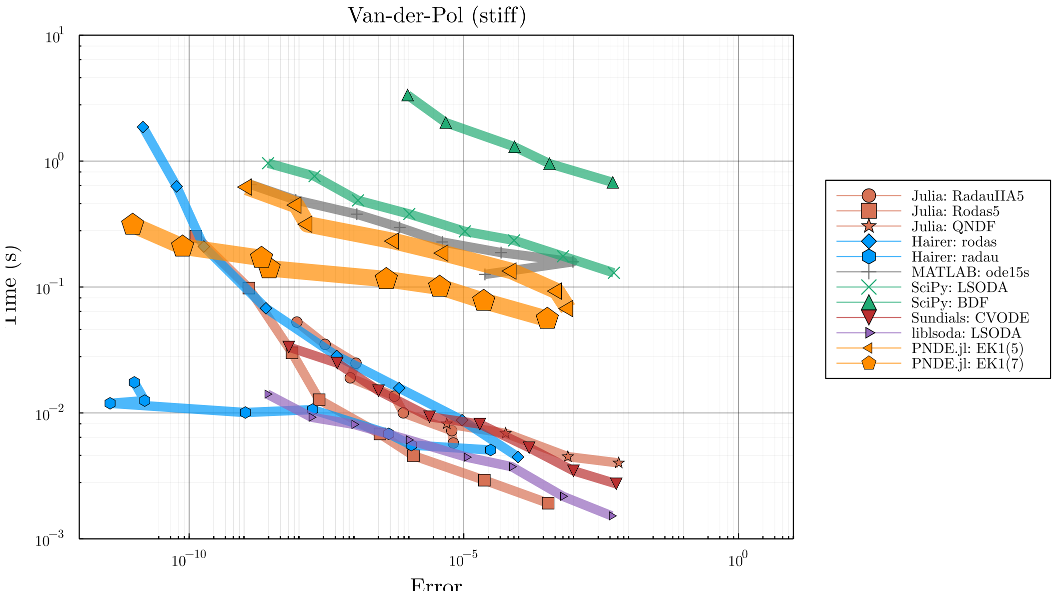
Benchmark (non-stiff)



Benchmark (non-stiff)



Benchmark (stiff)



Initial value problem

$$\ddot{y}(t) = f(\dot{y}(t), y(t), t), \qquad t \in [t_{\text{min}}, t_{\text{max}}], \qquad \dot{y}(t_{\text{min}}) = \dot{y}_0, \qquad y(t_{\text{min}}) = y_0. \tag{2}$$

ODE Filters in a nutshell:

- Prior: $y \sim$ Gauss-Markov
- Adjusted information operator:

$$\mathcal{Z}[y](t) = \ddot{y}(t) - f(\dot{y}(t), y(t), t) \equiv 0. \tag{3}$$

- Discretize and infer (with an extended Kalman filter)

$$p\left(y(t) \mid \{Z[y](t_i) = 0\}_{i=1}^{N}\right) \tag{4}$$

Second-order ODEs, energy preservation, additional derivatives, DAEs: [Bosch et al., 2021b]

Demo time

Acknowledgments

Thanks to all my collaborators:

- Philipp Hennig
- Filip Tronarp
- Nicholas Krämer
- Jonathan Schmidt

Bibliography I

- Bosch, N., Hennig, P., and Tronarp, F. (2021a).
  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., Tronarp, F., and Hennig, P. (2021b).
  Pick-and-mix information operators for probabilistic ode solvers.
  CoRR.
- Rackauckas, C., Ma, Y., Dixit, V., Guo, X., Innes, M., Revels, J., Nyberg, J., and Ivaturi, V. (2018).
  A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions.
  CoRR.

- Rackauckas, C. and Nie, Q. (2017).
  DifferentialEquations.jl a performant and feature-rich ecosystem for solving differential equations in julia.
  Journal of Open Research Software, 5(1).

Backup