EBERHARD KARLS UNIVERSITÄT TÜBINGEN

ProbNumDiffEq.jl – Probabilistic Numerics for ODEs

Nathanael Bosch, University of Tübingen

MAX-PLANCK-INSTITUT FÜR INTELLIGENTE SYSTEME

imprs-is

Summary

- **Numerical algorithms** compute an approximate solution to numerical problems, such as differential equations, linear algebra, integration, optimization, ...
- **Probabilistic numerical algorithms** return a posterior distribution over solutions, which includes a probabilistic quantification of their numerical approximation error.
- **ProbNumDiffEq.jl** implements probabilistic numerical solvers for ordinary differential equations (ODEs) in Julia, building on `OrdinaryDiffEq.jl` [7].

Probabilistic Numerical ODE Solvers

Consider an initial value problem

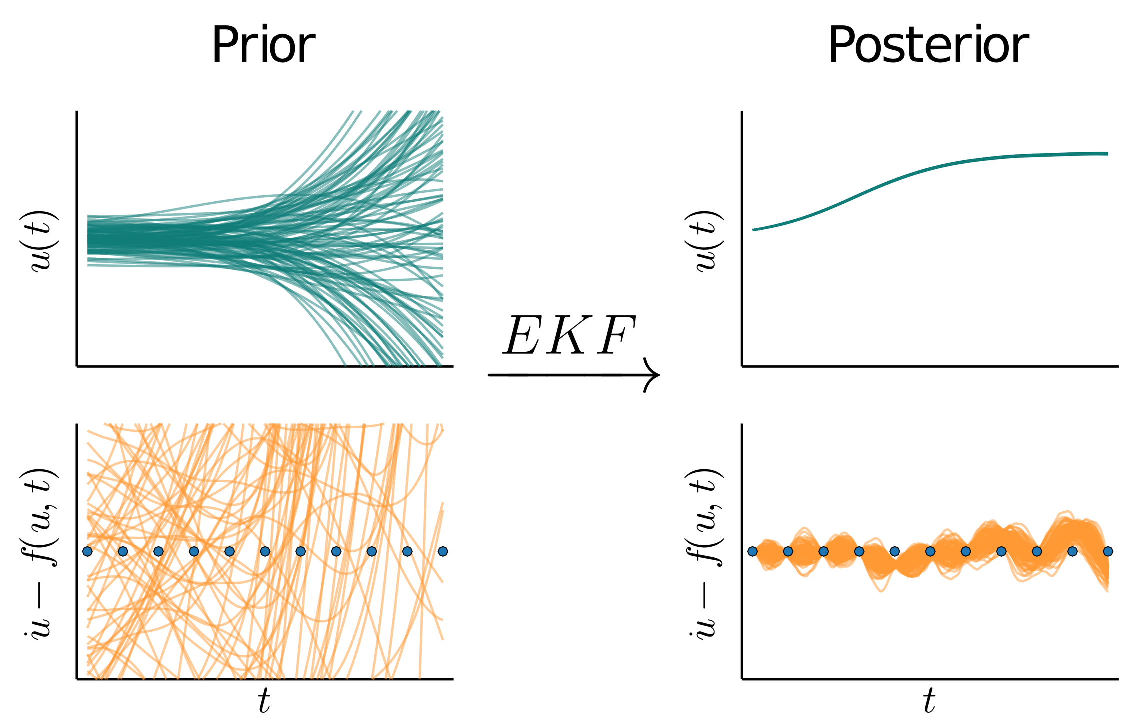
$$\dot{u}(t) = f\left(u(t), t\right), \qquad \forall t \in [t_0, T], \tag{1}$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ and initial value $u(t_0) = u_0 \in \mathbb{R}^d$.
To quantify the numerical error, we seek to compute posterior distributions

$$p\left(u(t) \;\middle|\; \{\dot{u}(t_n) = f(u(t_n), t_n)\}_{n=1}^N\right), \tag{2}$$

for some time discretization $\{t_n\}_{n=1}^N \subset [t_0, T]$.

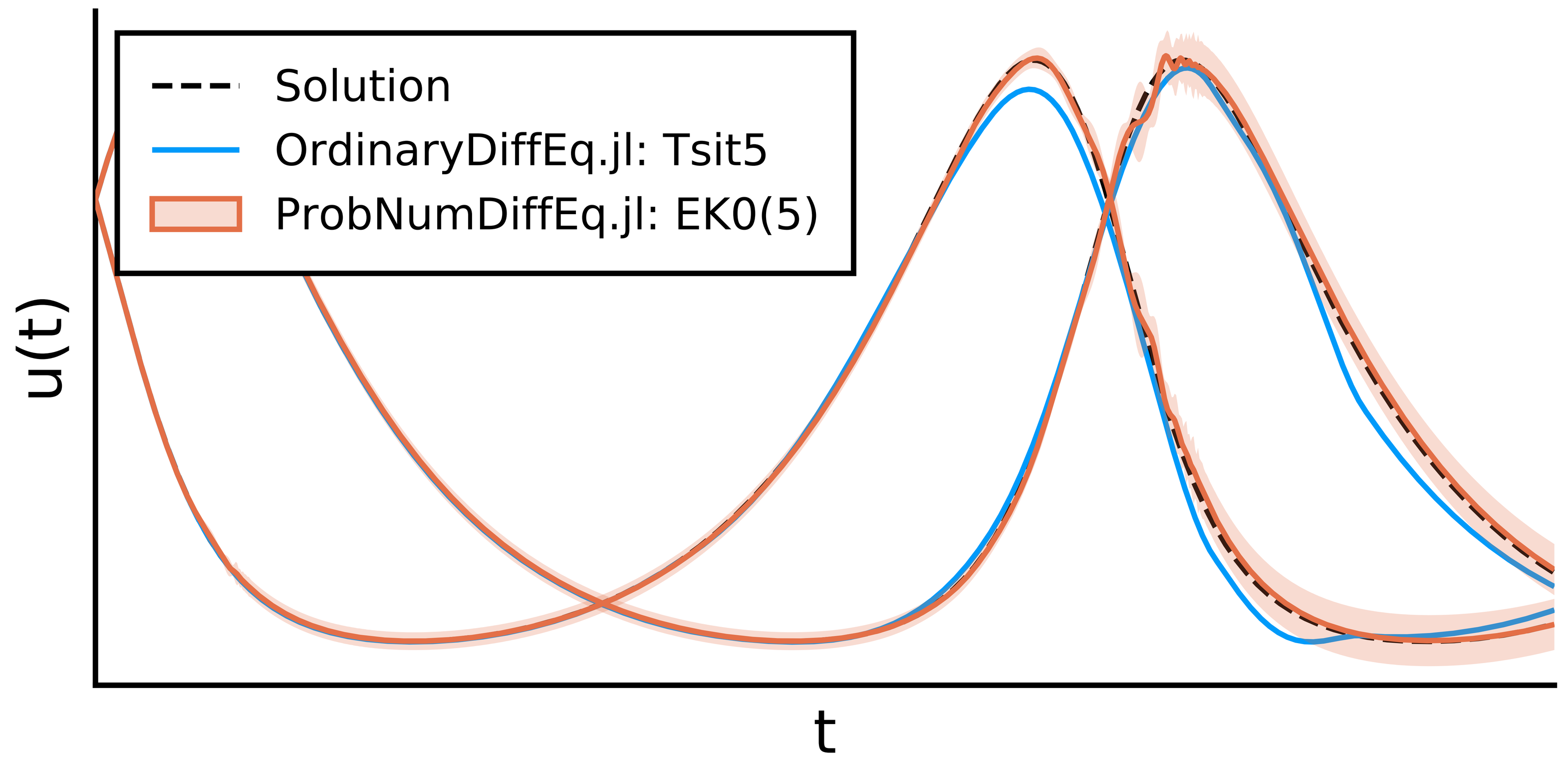
- **Prior:** Model $u(t)$ with a q-times integrated Wiener process (IWP) $U : [0, \infty) \to \mathbb{R}^{d(q+1)}$, $t \mapsto U(t) = \left[U^{(0)}(t), U^{(1)}(t), \dots, U^{(q)}(t)\right]$, with
$$\mathrm{d}U^{(i)}(t) = U^{(i+1)}(t)\,\mathrm{d}t, \quad i = 0, \dots, q-1, \tag{3a}$$
$$\mathrm{d}U^{(q)}(t) = \sigma I_d \,\mathrm{d}W(t), \tag{3b}$$
such that $U^{(i)}$ models the i-th derivative $\mathrm{d}^i u / \mathrm{d}t^i$.
- **Measurement process & Data:** $Z(t) := U^{(1)}(t) - f\left(U^{(0)}(t), t\right) \equiv 0$.
- **Approximate inference with Gaussian filtering / smoothing:** Approximate
$$p\left(U(t) \mid \{Z(t_i) = 0\}_{i=1}^N\right) \approx \mathcal{N}\left(\mu(t), \Sigma(t)\right) \tag{4}$$
efficiently (i.e. $\mathcal{O}(N)$) with extended Kalman filtering / smoothing [3, 5].
- Visual example for the logistic equation $\dot{u}(t) = u(t)(1 - u(t))$:



- This framework includes explicit (`EK0`), semi-implicit (`EK1`), and implicit methods (`IEKS`, currently only a prototype implementation) (see [4] for more information).
- The posterior distribution (Eq. (4)) naturally provides *dense output* and sampling.

Solving ODEs with `ProbNumDiffEq.jl`

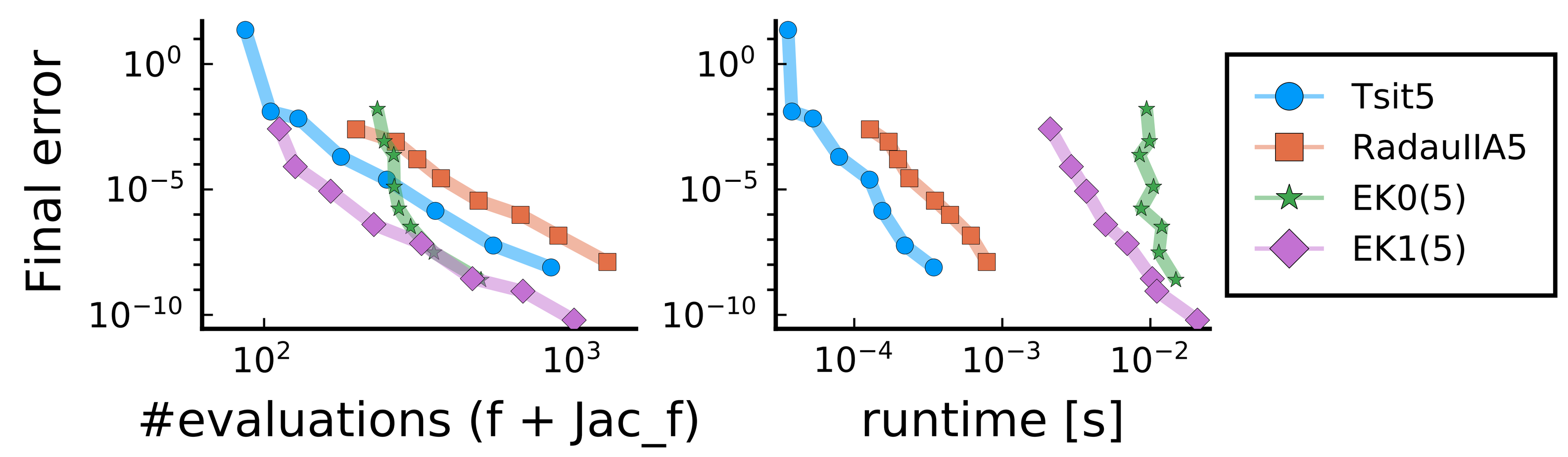
```julia
# ] add ProbNumDiffEq
using ProbNumDiffEq, OrdinaryDiffEq, Plots

# Problem definition as in DifferentialEquations.jl
lotkavolterra(u, p, t) = [0.5 * u[1] - 0.05 * u[1] * u[2]
                          -0.5 * u[2] + 0.05 * u[1] * u[2]]
u0 = [20.0; 20.0]
tspan = (0.0, 20.0)
prob = ODEProblem(lotkavolterra, u0, tspan)

# High-accuracy solve:
appxsol = solve(prob, Tsit5(), abstol=1e-12, reltol=1e-12)

# Low-accuracy solve with a non-probabilistic solver:
sol1 = solve(prob, Tsit5(), abstol=1e-1, reltol=1e-0)

# Low-accuracy solve with a probabilistic solver:
sol2 = solve(prob, EK0(order=5), abstol=1e-1, reltol=1e-0)

plot(appxsol, linestyle=:dash, color=:black, label="Solution")
plot!(sol1, color=1, label="OrdinaryDiffEq.jl: Tsit5")
plot!(sol2, color=2, label="ProbNumDiffEq.jl: EK0(5)")
```



Results:

- Both solutions are disturbed by a noticeable numerical approximation error.
- The probabilistic `EK0(5)` solver provides a posterior distribution and thereby estimates its own numerical uncertainty.
- For more inspection we could also generate samples of the posterior distribution.

Performance



- **Complexity** of extended Kalman filtering and smoothing: $\mathcal{O}\left(Nd^3q^3\right)$.
- **Many matrix-matrix operations**
  $\Rightarrow$ broadcast / `FastBroadcast.jl` not straight-forward.
- **Implementation details:** Currently the solver still allocates lots of memory for intermediate calculations which can be optimized away (WIP).

Additional Remarks

- `ProbNumDiffEq.jl` is compatible with many features from `OrdinaryDiffEq.jl` / `DifferentialEquations.jl` / Julia: Stepsize control, plotting functionality, callbacks, `ForwardDiff.jl`, ...
- But not everything works perfectly yet: Matrix-valued ODEs, GPU-arrays, `StaticArrays.jl`, backprop (e.g. with `Zygote.jl`), sensitivities, ...
- **Related approach:** The `ProbInts` method provided by `DiffEqUncertainty.jl` quantifies numerical uncertainty by repeatedly solving the ODE and disturbing the solution (see also [9]). In comparison, the filtering-based solvers of `ProbNumDiffEq.jl` require only a single solve to compute a probabilistic, Gaussian posterior.
- **For more probabilistic numerics check out** ProbNum, a feature-rich Python package for probabilistic numerics [6]. It includes probabilistic linear solvers, Bayesian quadrature, probabilistic ODE solvers, filtering and smoothing algorithms, and more.

References and related work:

[1] N. Bosch et al. "Calibrated Adaptive Probabilistic ODE Solvers". In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[2] S. Särkkä and A. Solin. "Applied Stochastic Differential Equations". In *Cambridge University Press*, 2019.

[3] F. Tronarp et al. "Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: a new perspective". In *Statistics and Computing*, 2019.

[4] F. Tronarp et al. "Bayesian Ode Solvers: the Maximum a Posteriori Estimate". In *Statistics and Computing*, 2021.

[5] S. Särkkä. "Bayesian Filtering and Smoothing". In *Cambridge University Press*, 2013.

[6] ProbNum, https://github.com/probabilistic-numerics/probnum.

[7] OrdinaryDiffEq.jl, https://github.com/SciML/OrdinaryDiffEq.jl.

[8] C. Rackauckas and Q. Nie. "DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in julia". In *Journal of Open Research Software*, 2017.

[9] P. Conrad et al. Girolami M, Särkkä S, Stuart A, Zygalakis K. "Statistical analysis of differential equations: introducing probability measures on numerical solutions" In *Statistics and Computing*, 2017.

nathanael.bosch@uni-tuebingen.de

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