Journal of Machine Learning Research 25 (2024) 1-29 Submitted 12/20; Revised 7/22; Published 4/24

Stable Implementation of Probabilistic ODE Solvers

Nicholas Krämer* NICHOLAS.KRAEMER@UNI-TUEBINGEN.DE
Tübingen AI Center, University of Tübingen
Maria-von-Linden-Straße 6, Tübingen, Germany

Philipp Hennig PHILIPP.HENNIG@UNI-TUEBINGEN.DE
Tübingen AI Center, University of Tübingen
Maria-von-Linden-Straße 6, Tübingen, Germany

Editor: Pradeep Ravikumar

Abstract

Probabilistic solvers for ordinary differential equations (ODEs) provide efficient quantification of numerical uncertainty associated with the simulation of dynamical systems. Their convergence rates have been established by a growing body of theoretical analysis. However, these algorithms suffer from numerical instability when run at high order or with small step sizes—that is, exactly in the regime in which they achieve the highest accuracy. The present work proposes and examines a solution to this problem. It involves three components: accurate initialisation, a coordinate change preconditioner that makes numerical stability concerns step-size-independent, and square-root implementation. Using all three techniques enables numerical computation of probabilistic solutions of ODEs with algorithms of order up to 11, as demonstrated on a set of challenging test problems. The resulting rapid convergence is shown to be competitive with high-order, state-of-the-art, classical methods. As a consequence, a barrier between analysing probabilistic ODE solvers and applying them to interesting machine learning problems is effectively removed.

Keywords: Probabilistic numerics, dynamical systems, ordinary differential equations, Gauss-Markov processes, state estimation

1. Introduction

Ordinary differential equations (ODEs) are a core concept of mechanistic modelling. Efficiently computing ODE solutions is thus important in a wide range of applications in the natural sciences. Recently, probabilistic solvers for ODEs have emerged (see the paper by Schober et al. (2014) and the references therein). These methods can not only return a single point estimate that represents an approximation of an ODE solution, but they also provide uncertainty quantification calibrated to be representative of the global error (Schober et al., 2019; Bosch et al., 2021). Like classical methods, they have linear complexity in the number of grid points, and they enjoy similar convergence properties: a solver that models the ODE solution as well as its first ν derivatives ($\nu \in \mathbb{N}$) can converge as fast as h^ν for step-size h (Kersting et al., 2020b; Tronarp et al., 2021). However, high-order algorithms in conjunction with small step-sizes suffer from numerical instabilities (see Table 1). Below in Section 3, we show why.

*. Now at the Technical University of Denmark. E-mail: PEKRA@DTU.DK. Work performed while in Tübingen.

	EK0 (traditional)	EK1 (traditional)	EK0 (stable)	EK1 (stable)
2	✓	✓	✓	✓
3	✓	✓	✓	✓
4	✓	✓	✓	✓
5	×	✓	✓	✓
6	×	×	✓	✓
7	×	×	✓	✓
8	×	×	✓	✓
9	×	×	✓	✓
10	×	×	✓	✓
11	×	×	✓	✓

Table 1: **Successful approximation of an ODE solution.** A probabilistic numerical approximation of the solution of the logistic ODE $\dot{x}(t) = 4x(t)(1 - x(t))$, $x(0) = 0.15$, $t \in [0., 2.]$, is computed with two solver variants (EK0 and EK1; explanations below), in the traditional and the stabilised version, for increasing regularity ν of the prior. A "✓" indicates that the solver successfully computed an adaptive solution with an error smaller than the prescribed tolerance 10^{-5}. A "×" indicates that numerical instability leads to failure. *Only in the proposed implementation, the probabilistic solvers reliably compute approximate solutions.*

Probabilistic ODE solvers cast the task of solving an ODE as a Gauss-Markov process regression problem with a non-linear observation model. This class of algorithms, at the core of probabilistic numerical methods (Hennig et al., 2015; Cockayne et al., 2019; Oates and Sullivan, 2019), builds on the large body of theory on Gaussian processes, stochastic differential equations, Bayesian filtering and smoothing as well as numerical analysis. This blend of ideas shows in the present work: the solution to the problem of numerically stable implementation draws on concepts related to Taylor-mode automatic differentiation, Nordsieck methods, and square-root Kalman filter implementations. It solves a problem in Gauss-Markov process regression and as such, has an impact on (probabilistic) numerics and possibly every chain of computation that benefits from cheap yet effective uncertainty quantification of numerical simulation of a dynamical system. In recent years, this has turned out to be an important challenge in machine learning.

Section 2 recalls the construction of probabilistic ODE solvers in the formulation as a problem of Bayesian state estimation (alternatives are provided by Chkrebtii et al. (2016), Conrad et al. (2017), or Abdulle and Garegnani (2020)). Section 3 explains and examines the tricks that are necessary to implement a high-order method. Therein, specific parts of the algorithm are isolated. A detailed, step-by-step guide is contained in Appendix A. Section 4 benchmarks the probabilistic ODE solver in the suggested implementation against high-order Runge-Kutta methods. The test environments that are used throughout the paper are the logistic equation, Lotka-Volterra, the restricted three-body problem, the SIR model, the Pleiades problem, and a stiff version of the van der Pol system.

Matrices are capitalised ($A \in \mathbb{R}^{d \times d}$), scalars are lower-case ($a \in \mathbb{R}$) and matrices with a Kronecker structure are capitalised and bold-faced ($\mathbf{A} = A \otimes I_3 \in \mathbb{R}^{3d \times 3d}$). Vectors

that are stacks of vectors are bold-faced ($\mathbf{x} = (x, y, z)^\top \in \mathbb{R}^{3d}$); generic vectors are not ($x \in \mathbb{R}^d$). Stochastic processes, e.g. the Wiener process ($w(t)$), are, with slight abuse of notation, written as functions ($x(t)$). We use zero-indexing when describing a matrix with its elements ($A = (a_{ij})_{i,j=0,\ldots,N}$). Condition numbers of matrices are computed with respect to the ℓ^2-norm.

2. Probabilistic ODE Solvers

The present work is concerned with numerically solving the d-dimensional, autonomous, first-order ordinary differential equation (ODE) initial value problem,

$$\begin{cases} \dot{x}(t) = f(x(t)), \text{ for } 0 \leq t \leq T < \infty, \\ x(0) = x_0. \end{cases}$$

This is no loss of generality: most ODEs are autonomous, but even non-autonomous or higher-order ODEs can be translated into autonomous, first-order ODEs. This would be done by writing the non-autonomous ODE as an autonomous ODE over the augmented state $\tilde{x} = (x(t), t)^\top$. The restriction to autonomous ODEs simplifies the notation with respect to Jacobians of f, which will be required frequently throughout the subsequent exposition; especially in Section 3.1.

In the machine learning literature, the scenario in which the ODE vector field is a neural network with weights θ, $f = \mathrm{NN}_\theta(x)$, has gained traction in recent years (Chen et al., 2018; Rackauckas et al., 2020). Numerically solving the initial value problem, in this case, corresponds to evaluating the neural network, in which the number of hidden layers corresponds to the number of evaluations that the solver takes. Probabilistic solvers have not yet been applied to neural ODEs. As shown in the present work (and in the related study by Bosch et al. (2021)), being able to faithfully integrate dynamical systems with algorithms of order $\nu = 5$ and larger implies that the approximation quality and convergence speed will not be a hindering factor for this endeavour any longer.

Other applications of ODE solvers consist of identifying mechanistic models from data. In the presence of a data set that is based on a dynamical system, the determination of such a system yields not only an understanding of the generative process responsible for the observations but also a compact representation of these dynamics as an ODE. Kersting et al. (2020a) show how probabilistic ODE solvers give rise to efficient algorithms that deal with this inverse problem.

Yet another application of computational ODE solutions lies within manifold learning: straight lines on manifolds, so-called geodesics, are computed by numerically solving a set of Euler-Lagrange ordinary differential equations. Geodesics are important for statistical analysis on manifolds because, among other things, they give rise to distance functions. Hennig and Hauberg (2014) and Arvanitidis et al. (2019) study the positive impact that a probabilistic ODE solver has on manifold learning.

Probabilistic ODE solvers are Gauss-Markov process regressors based on a non-linear observation model. Thus, Section 2.1 defines a prior distribution, Section 2.2 describes the observation model and Section 2.3 outlines common inference strategies. Section 2.4 discusses calibration and adaptive step-size selection.

2.1 Prior Distribution

This work considers Gauss-Markov priors $\mathbf{x} = \mathbf{x}(t)$ that are defined as solutions of linear, time-invariant stochastic differential equations (SDE) with Gaussian initial conditions,

$$\begin{cases} \mathrm{d}\mathbf{x}(t) = \mathbf{F}\,\mathbf{x}(t)\,\mathrm{d}t + \mathbf{L}\,\mathrm{d}\mathbf{w}(t), \text{ for } t \geq 0, \\ \mathbf{x}(0) \sim \mathcal{N}(\mathbf{m}_0, \mathbf{C}_0). \end{cases}$$

The vector $\mathbf{x}(t) = (x(t), \dot{x}(t), ..., x^{(\nu)}(t))^\top \in \mathbb{R}^{d(\nu+1)}$ models a stack of the ODE solution $x(t) \in \mathbb{R}^d$ and its derivatives up to order $\nu \in \mathbb{N}$. The dispersion matrix $\mathbf{L}$ is, in the cases that are of interest to us, always $\mathbf{L} = e_{\nu+1} \otimes I_d \in \mathbb{R}^{d(\nu+1)\times d}$. $\mathbf{w}$ is a d-dimensional Wiener process with constant diffusion $\Gamma > 0$. Choices of $\mathbf{F}$, $\mathbf{m}_0$ and $\mathbf{C}_0$ determine whether $\mathbf{x}(t)$ is, for instance, a ν-times integrated Wiener process (IWP(ν)), a ν-times integrated Ornstein-Uhlenbeck process, or a Matérn process of order $\nu + 1/2$.

Let $\mathbb{T} = \{t_0, ..., t_N\}$ be a grid on $[0, T]$. Without loss of generality assume $t_0 = 0$ and $t_N = T$. Define the step-size $h_n = t_{n+1} - t_n$. Restricted to $\mathbb{T}$, there is an alternative, discretised description of the prior process such that the distribution of the continuous process $\mathbf{x} = \mathbf{x}(t)$ restricted to $\mathbb{T}$ coincides with the distribution of the discrete process (Grewal and Andrews, 2014). Abbreviate $\mathbf{x}_n := \mathbf{x}(t_n)$; then $(\mathbf{x}_n)_{n=0,...,N}$ follows the distribution

$$\begin{cases} \mathbf{x}_{n+1} \mid \mathbf{x}_n \sim \mathcal{N}(\mathbf{A}_n\mathbf{x}_n, \mathbf{Q}_n), \text{ for } n = 0, ..., N, \\ \mathbf{x}_0 \sim \mathcal{N}(\mathbf{m}_0, \mathbf{C}_0), \end{cases}$$

with matrices $\mathbf{A}_n \in \mathbb{R}^{d(\nu+1)\times d(\nu+1)}$ and $\mathbf{Q}_n \in \mathbb{R}^{d(\nu+1)\times d(\nu+1)}$ given by (Särkkä and Solin, 2019, Section 6.1)

$$\mathbf{A}_n := \exp(\mathbf{F}h_n), \tag{3}$$

$$\mathbf{Q}_n := \int_0^{h_n} \exp(\mathbf{F}(h_n - \tau))\mathbf{L}\mathbf{L}^\top \exp(\mathbf{F}^\top(h_n - \tau))\,\mathrm{d}\tau. \tag{4}$$

In the following, we will sometimes refer to $\mathbf{Q}_n$ as "process noise covariance". Both $\mathbf{A}_n$ and $\mathbf{Q}_n$ can be computed efficiently with matrix fraction decomposition (Särkkä and Solin, 2019, Section 6.3). For the integrated Wiener process, there exist closed form solutions to Equations (3) and (4); we refer to Section 3.

2.2 Observation Model

Recall the abbreviation $\mathbf{x}_n := \mathbf{x}(t_n)$. Define the projection matrix $\mathbf{E}_i^\top = e_i^\top \otimes I_d \in \mathbb{R}^{d\times d(\nu+1)}$, where e_i is the ith canonical basis vector in $\mathbb{R}^{\nu+1}$, $i = 0, ..., \nu$. Loosely speaking, $\mathbf{E}_i^\top$ extracts the ith derivative from the stack of derivatives in $\mathbf{x}_n$. A probabilistic ODE solver computes a posterior distribution over $\mathbf{x}(t)$,

$$p\left(\mathbf{x}(t) \;\middle|\; \mathbf{E}_1^\top\mathbf{x}_n - f(\mathbf{E}_0^\top\mathbf{x}_n) = 0\right), \quad n = 0, ..., N,$$

based on the prior distribution from Section 2.1, likelihood function

$$\ell(\mathbf{x}_n) := \delta\left[\mathbf{E}_1^\top\mathbf{x}_n - f(\mathbf{E}_0^\top\mathbf{x}_n)\right], \quad n = 0, ..., N, \tag{5}$$

and (artificial) observations at each grid point, which are all equal to zero. This is a non-linear regression problem if f is non-linear (which it usually is). The data likelihood is a Dirac distribution composed with a non-linear function, and inference in this model is generally intractable. If $\mathbf{x}$ is assumed to be Gaussian, approximate Gaussian filtering and smoothing yields a tractable approximation of this likelihood and hence of the posterior distribution.

2.3 Approximate Gaussian Inference

The non-linear regression problem can be solved approximately by linearising f and applying Gaussian filtering and smoothing. Common choices are the extended Kalman filter, which linearises f with a Taylor approximation, and the unscented Kalman filter, which approximates the behaviour of f, as it acts on Gaussian random variables, with the unscented transform (Särkkä, 2013).

The following describes the extended Kalman filter as applied to the ODE problem (Tronarp et al., 2019). Let $\mathbf{x}_n \sim \mathcal{N}(\mathbf{m}_n, \mathbf{C}_n)$. The linearised observation model is

$$\ell(\mathbf{x}_n) \approx \delta\left[\mathbf{H}\,\mathbf{x}_n - \mathbf{b}\right].$$

$\mathbf{H}$ and $\mathbf{b}$ are derived using either a zeroth order Taylor approximation of $f(\mathbf{E}_0^\top \circ)$ at $\mathbf{m}_n$ (EK0),

$$\mathbf{H} = \mathbf{E}_1^\top, \quad \mathbf{b} = f(\mathbf{E}_0^\top \mathbf{m}_n), \tag{6}$$

or a first order Taylor approximation of $f(\mathbf{E}_0^\top \circ)$ at $\mathbf{m}_n$ (EK1),

$$\mathbf{H} = \mathbf{E}_1^\top - \nabla f(\mathbf{E}_0^\top \mathbf{m}_n)\mathbf{E}_0^\top, \quad \mathbf{b} = f(\mathbf{E}_0^\top \mathbf{m}_n) - \nabla f(\mathbf{E}_0^\top \mathbf{m}_n)\mathbf{E}_0^\top \mathbf{m}_n. \tag{7}$$

∇f is the Jacobian of f. If the ODE is not autonomous, the Jacobian of f with respect to x is used. Both choices, EK0 and EK1, enable Gaussian filtering and smoothing algorithms; see the implementation guide in Appendix A.

This work, like Bosch et al. (2021), only considers the extended Kalman filter. Everything explained herein applies to the unscented Kalman filter as well, but we do not use it for reasons of computational efficiency: Linearisation of the non-linear observation model (Equation (5)) with the unscented transform requires $d(\nu+1)$ evaluations of the ODE vector field f for a single ODE solver step—one evaluation for each of the so-called *sigma-points* that are used for the unscented transform. The costs of these evaluations should be judged in comparison to the costs of evaluating the Jacobian ∇f to form $\mathbf{H}$ in EK1. In EK0, neither is required.

The template for computing the step from grid-point t_n to t_{n+1} with the probabilistic ODE solver in its traditional implementation is in Algorithm 1. After initialising the iteration by setting the first component of $\mathbf{m}_n$ to x_0, and by setting the respective elements in $\mathbf{C}_n$ to zero, the ODE solver step in Algorithm 1 is repeated until the terminal point $t_N = T$ is reached. The output of a probabilistic ODE solver, including the posterior mean, confidence intervals, and samples is shown in Figure 1. Section 3 discusses which components need to be replaced with a more specialised implementation to gain improved numerical stability. Appendix A summarises the results that follow in Section 3 in an implementation guide, which will supersede Algorithm 1.

Algorithm 1 Template for a probabilistic ODE solver step from t_n to t_{n+1}.

Require: $\mathbf{x}_n \sim \mathcal{N}(\mathbf{m}_n, \mathbf{C}_n)$

Predict mean $\mathbf{m}^-_{n+1} \coloneqq \mathbf{A}_n \mathbf{m}_n$	▷ Equation (2)
Predict covariance $\mathbf{C}^-_{n+1} = \mathbf{A}_n \mathbf{C}_n \mathbf{A}_n^\top + \mathbf{Q}_n$	▷ Equation (2)
Linearise $\ell(\mathbf{x}) \approx \delta[\mathbf{H}\mathbf{x} - \mathbf{b}]$ at $\mathbf{m}^-_{n+1}$	▷ Equation (6) or Equation (7)
Compute observation mean $\mathbf{z} = \mathbf{H}\mathbf{m}^-_{n+1} - \mathbf{b}$	▷ (Tronarp et al., 2019, Eq. (12c))
Compute observation covariance $\mathbf{S} = \mathbf{H}\mathbf{C}^-_{n+1}\mathbf{H}^\top$	▷ (Tronarp et al., 2019, Eq. (12a))
Compute Kalman gain $\mathbf{K} = \mathbf{C}^-_{n+1}\mathbf{H}^\top\mathbf{S}^{-1}$	▷ (Tronarp et al., 2019, Eq. (12b))
Update mean $\mathbf{m}_{n+1} = \mathbf{m}^-_{n+1} - \mathbf{K}\mathbf{z}$	▷ (Tronarp et al., 2019, Eq. (12d))
Update covariance $\mathbf{C}_{n+1} = \mathbf{C}^-_{n+1} - \mathbf{K}\mathbf{S}\mathbf{K}^\top$	▷ (Tronarp et al., 2019, Eq. (12e))

The solution to the continuous-discrete state estimation problem posed by the probabilistic ODE solver is a posterior distribution over the continuous process $\mathbf{x}(t)$. It can be evaluated at any time t, that is, in between the grid points that were used by the ODE solver to approximate the solution to the initial value problem. In numerical analysis, this is called dense output (Hairer et al., 1993, Chapter II.6). In our setting, computing the dense output is realised by carrying out an additional, measurement-free smoothing step—and thus does not evaluate the ODE vector field f. We refer to the discussion surrounding Algorithm 10.27 in the book by Särkkä and Solin (2019).

2.4 Calibration and Step-Size Adaptation

Efficient ODE solvers use local error control and step-size adaptation. In probabilistic ODE solvers, the posterior covariance, which quantifies numerical uncertainty over the approximate ODE solution, benefits from the post-hoc calibration of the diffusion Γ of the driving Wiener process. Both tasks have recently been studied by Bosch et al. (2021). The authors present strategies for the calibration of Γ as a local quasi-maximum likelihood estimate. They extend the calibration techniques presented by Schober et al. (2019) and Tronarp et al. (2019) by evaluating the effect of time-varying diffusion (Schober et al., 2019) versus time-constant diffusion (Tronarp et al., 2019) on different variants of ODE solvers.

Uncertainty calibration can be combined efficiently with error control. Calibrated uncertainty estimates make error estimates more meaningful and improve the adaptive step-size selection (Bosch et al., 2021). In this work, we use the time-varying diffusion model together with on-the-fly calibration and the corresponding error estimate (Schober et al., 2019; Bosch et al., 2021).

At this point, we would like to emphasise that herein, uncertainty estimates are mostly ignored and only used to the extent that is required for error control. We benchmark the probabilistic ODE solver *as if it was a classical, deterministic method.* Compared to related work on probabilistic numerics, this is a rather drastic point of view and taken in order to demonstrate numerical stability and approximation quality of the algorithm. Both will be shown to be on the same level as well-established, high-order, classical numerical methods.

On top of these qualities, a probabilistic ODE solver provides uncertainty quantification in form of posterior covariances *without additional costs.* All of the run time comparisons

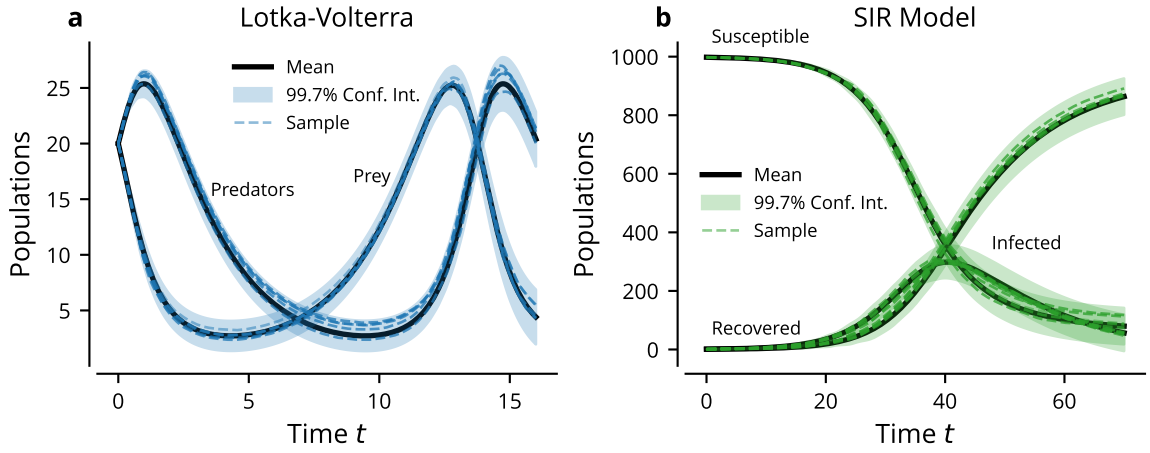

Figure 1: **Exemplary output of a probabilistic ODE solver.** Simulation of the Lotka–Volterra model (a) and the SIR model (b), in the parametrisations that are also used in the experiments in Section 4 below. Both problems use an EK1 based on a once-integrated Wiener process prior, a time-varying diffusion model, and fixed steps of size $h = 0.3$ (Lotka–Volterra), respectively $h = 3.5$ (SIR).

below already include the computation of posterior covariances. Readers interested in uncertainty calibration are referred to Bosch et al. (2021).

The EK1-solver is A-stable (Tronarp et al., 2019, Corollary 1); a definition of A-stability is provided by Dahlquist (1963). We demonstrate this stability together with the validity of the local error control scheme in Figure 2, by solving the van der Pol system of ordinary differential equations (Guckenheimer, 1980),

$$\ddot{x} = \mu((1 - x^2)\dot{x} - x) \tag{8}$$

from $t = 0$ to $t = 6.3$ with initial values $x(0) = 2$, and $\dot{x}(0) = 0$. We solve the ODE in Equation (8) after transforming it into a first-order problem. We replicate the parameterisation chosen by Bosch et al. (2021) and set $\mu = 10^6$. With such a μ, Equation (8) is a stiff ordinary differential equation. We solve this problem using EK1 with relative tolerance 10^{-6}, absolute tolerance 10^{-3}, order $\nu = 7$, adaptive step-selection with a PI-controller (Gustafsson et al., 1988), and the time-varying diffusion model originally proposed by Schober et al. (2019) and extended to EK1 by Bosch et al. (2021). We only rely on standard, 64-bit floating point arithmetic.

Without either of the implementation tricks presented below, computing the solution was impossible. The simulation took ≈ 3.5 seconds; a reference solution with a fifth-order Radau IIA solver, implemented in SciPy (Virtanen et al., 2020), on the same problem and with the same tolerance took ≈ 5 seconds. The van der Pol system with parameter $\mu = 10^3$ cannot be solved with EK0, likely because it does not possess the stability properties of EK1; the analysis by Tronarp et al. (2019) only applies to EK1 (and the unscented Kalman filter).

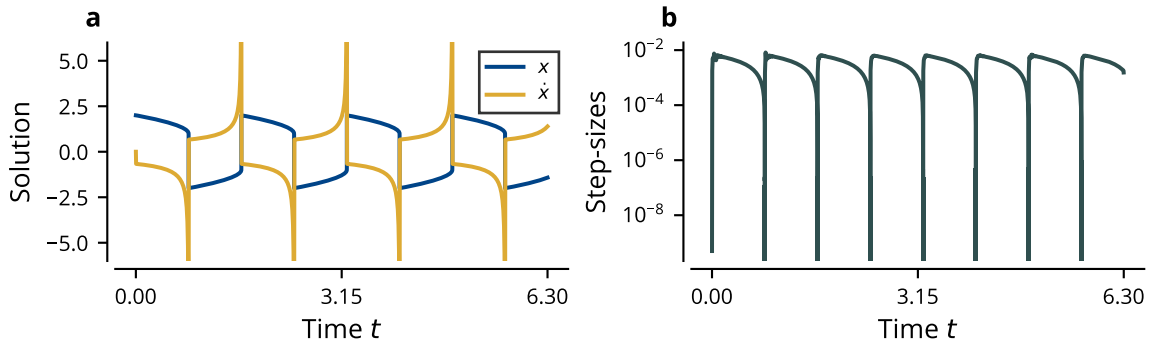

Figure 2: **Stiff van der Pol system.** The derivative of the solution of Eq. (8) exhibits extreme spikes (a). The y-axis is cropped at $(-6, 6)$, but $\dot{x}$ takes values much larger in magnitude. The adaptive error control selects extremely small step sizes (b), which shows both, the stiffness of the problem as well as the ability of the algorithm to detect and cope with it. The first step is immediately scaled down to $\approx 10^{-12}$ by the step-size controller.

From Figure 2 it is evident how small the step sizes must be in order for the integration to be successful.

3. Improved Numerical Stability

Exactly three components are important for the successful implementation of probabilistic ODE solvers: accurate initialisation (Section 3.1), a coordinate change in the state-space model that removes instabilities for small step-sizes or high orders (Section 3.2), and square-root implementation of the ODE solver (Section 3.3). Their implications on the overall computational complexity are discussed in Section 3.4.

3.1 Accurate Initialisation

It is important to initialise $\mathbf{m}_0$ and $\mathbf{C}_0$ as accurately as possible, for reasons of stability and approximation quality: if the initialisation is inaccurate, we report that in the best case, convergence rates do not hold and in the worst case, numerical over-/underflows happen after a few steps. (This was displayed in Table 1.)

Recall that $\mathbf{x}$ is a stack of the ODE solution x and its first ν derivatives. Ideally, the parameters of the initial distribution, $\mathbf{m}_0$ and $\mathbf{C}_0$, are chosen as

$$\mathbf{m}_0 = \begin{pmatrix} x(0) \\ \dot{x}(0) \\ \vdots \\ x^{(\nu)}(0) \end{pmatrix}, \quad \mathbf{C}_0 = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}.$$

It is non-trivial to compute those values efficiently—in the sequel we outline one option for doing this. Alternatives are discussed in Section 3.2 in the paper by Schober et al. (2019).

Applying Faà di Bruno's formula (Roman, 1980) to $y(t) := f(x(t))$ and substituting $\dot{x}(t) = f(x(t))$, computes higher order derivatives of x at 0. Let $0 \leq q < \nu$. The $(q+1)$th derivative of x, evaluated at $t = 0$ is obtained by following the recursion

$$\mathcal{F}_0(x) := f(x), \quad \mathcal{F}_{i+1}(x) := [\nabla \mathcal{F}_i](x) f(x), \quad i = 0, ..., q-1,$$

and evaluating at zero, $x^{(q+1)}(0) = \mathcal{F}_q(x_0)$. This approach can be implemented with automatic differentiation (AD). Care has to be taken with the choice of AD algorithm because if the recursive nature of the higher-order derivatives is not taken into account, the complexity of AD grows exponentially with respect to q (Kelly et al., 2020).

Taylor-mode automatic differentiation is an efficient way of computing higher order derivatives of a function. Loosely speaking, instead of tracking how to propagate directional derivatives (Jacobians), Taylor-mode AD tracks how to propagate truncated Taylor series. Let $\hat{x}$ be a νth order truncated Taylor series approximation of x at $t = 0$, and $\hat{y}$ and $\hat{f}$ be $(\nu-1)$th order truncated Taylor series approximations of $y(t) := f(x(t))$ and f, at $t = 0$ and $x = x_0$ respectively,

$$\hat{x}(t) = \sum_{q=0}^{\nu} x^{(q)}(0) t^q/(q!), \quad \hat{y}(t) = \sum_{q=0}^{\nu-1} y^{(q)}(0) t^q/(q!), \quad \hat{f}(x) = \sum_{|\rho|=0}^{\nu-1} f^{(\rho)}(x_0)(x-\xi)^\rho/(\rho!).$$

$\rho = (\rho_1, ..., \rho_d)$ is a multi-index, that is, $f^{(\rho)}$ means $f^{(\rho)}(x_0) = \left[\frac{\partial^{\rho_1}}{\partial x_1^{\rho_1}} \cdots \frac{\partial^{\rho_d}}{\partial x_d^{\rho_d}} f\right](x_0)$. Multi-index notation is necessary because the domain of f is multi-dimensional. The derivative coefficients $y^{(0)}, ..., y^{(\nu)}$ of $\hat{y}$ are computed by propagating $\hat{x}$ through $\hat{f}$.

Since x solves the ODE, $\dot{x} = y$ holds and higher-order Taylor coefficient terms of $\hat{x}$ can be computed from lower-order terms of $\hat{y}$, which themselves are computed from lower order derivatives of $\hat{x}$. More formally, the coefficients of $\hat{x}$ satisfy the recurrence relation $x^{(q+1)}(0) = y^{(q)}(0)/(q+1)$. The first ν derivatives of x can be read off exactly from the coefficients of $\hat{x}$, by definition of Taylor-series.

The computational complexity of this strategy grows quadratically, sometimes only almost linearly, in the order of the approximation (Griewank and Walther, 2008, Chapter 13). This is contrasted by the exponential growth in complexity in the order of the approximation of forward-mode AD. In our Python code, we use a Taylor-mode AD implementation in JAX (Bradbury et al., 2020; Bettencourt et al., 2019). Computing the νth coefficient of $\hat{x}$, which gives the value of the νth derivative at zero, requires a νth order Taylor approximation of f (everything else is computed with the iteration $x^{(q+1)}(0) = y^{(q)}(0)/(q+1)$. Computation of an nth order Taylor approximation of $f : \mathbb{R}^d \to \mathbb{R}^d$ requires storage $\binom{n+d}{d} \approx n^d/d!$ and propagation costs $\binom{2n+d}{d}$ (Griewank and Walther, 2008). The complexity with Taylor-mode AD is thus significantly lower than with forward-mode AD, even though both yield the exact same solution. Figure 3 displays how such a lower complexity makes the initialisation of the probabilistic solver feasible for large ν.

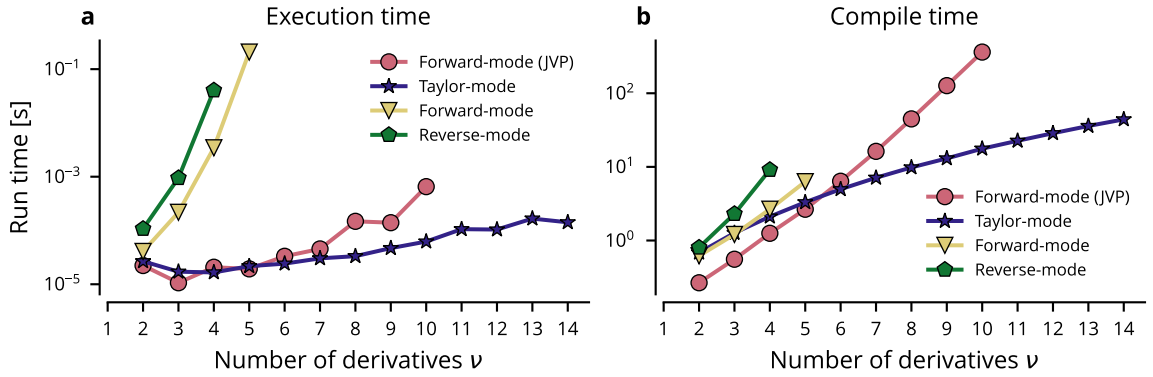

Figure 3: **Complexity of initialisation via automatic differentiation.** Execution time (a) and compile time (b) of automatic-differentiation-based initialisation strategies on the Pleiades problem (Hairer et al., 1993), a system of 28 first-order ODEs. Conventional forward- and reverse mode are too expensive for all ν. Jacobian-vector-product-based forward-mode differentiation (which avoids assembling full Jacobian matrices) is as efficient as Taylor-mode differentiation for $\nu < 5$, but becomes infeasible for $\nu > 5$. (Mind the different scales of the y-axes.) *Only Taylor-mode differentiation remains efficient for all ν.*

3.2 Rescaled Coordinates

The presentation in this section is restricted to the integrated Wiener process as a prior model. This seems to be a common choice, not only due to the intimate connection between integrated Wiener processes and polynomial splines (Wahba, 1978), but also because in the whole literature on probabilistic ODE solvers, only Magnani et al. (2017) and Kersting and Mahsereci (2020) have carried out experiments with a different prior. It is not clear whether the following coordinate change is optimal for prior models other than integrated Wiener processes.

This section deals with ill-conditioned matrices occurring in the filtering and smoothing iterations. Let $\mathbf{x}_n^F \sim \mathcal{N}(\mathbf{m}_n^F, \mathbf{C}_n^F)$ be the filter output at the nth step, which can be computed with Algorithm 1, and let $\mathbf{x}_{n+1}^S \sim \mathcal{N}(\mathbf{m}_{n+1}^S, \mathbf{C}_{n+1}^S)$ be the smoothing output at the $(n+1)$th step, that is, one step in the future. The smoothing distribution at the nth step, $\mathbf{x}_n^S \sim \mathcal{N}(\mathbf{m}_n^S, \mathbf{C}_n^S)$, is computed from these quantities as

$$
\begin{aligned}
\mathbf{m}_{n+1}^- &= \mathbf{A}_n \mathbf{m}_n^F \\
\mathbf{C}_{n+1}^- &= \mathbf{A}_n \mathbf{C}_n^F \mathbf{A}_n^\top + \mathbf{Q}_n \\
\mathbf{G}_n &= \mathbf{C}_n^F \mathbf{A}_n^\top (\mathbf{C}_{n+1}^-)^{-1} \\
\mathbf{m}_n^S &= \mathbf{m}_n^F - \mathbf{G}_n \left[\mathbf{m}_{n+1}^S - \mathbf{m}_{n+1}^-\right] \\
\mathbf{C}_n^S &= \mathbf{C}_n^F - \mathbf{G}_n \left[\mathbf{C}_{n+1}^S - \mathbf{C}_{n+1}^-\right] \mathbf{G}_n^\top,
\end{aligned}
\tag{9}
$$

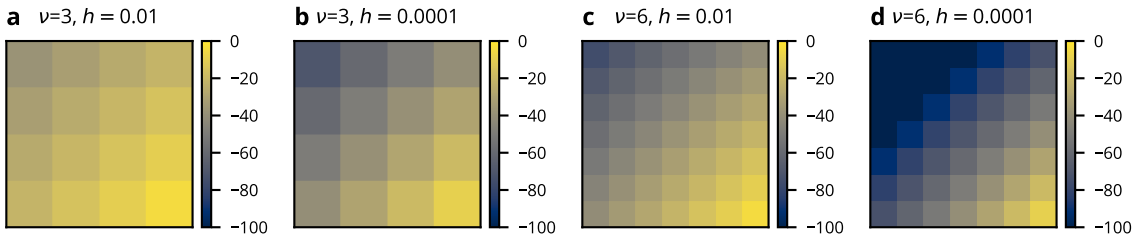

Figure 4: **The elements in Q_n decay rapidly from the bottom right to the top left element.** The figure displays the logarithm of the entries in Q, $[\log q_{ij}]_{i,j=0,\ldots,\nu}$, where the q_{ij} are the elements in Q_n (Equation 3.2), for different ν and h.

and thus depends on $(\mathbf{C}_{n+1}^-)^{-1}$. By going from $n = N-1$ to $n = 0$ backwards in time, the smoothing distributions can be computed sequentially and in complexity $\mathcal{O}(N)$.

$\mathbf{A}_n \mathbf{C}_n^F \mathbf{A}_n^\top$ is symmetric and positive semidefinite—after all, we want $\mathbf{C}_n^F$ to be zero, since we want the solution to be exact. The matrix $\mathbf{Q}_n$ is positive definite, but ill-conditioned; its representation for the IWP(ν) is

$$\mathbf{Q}_n = Q_n \otimes \Gamma, \quad Q_n = \left[\frac{h_n^{2\nu+1-i-j}}{(2\nu+1-i-j)(\nu-i)!(\nu-j)!} \right]_{i,j=0,\ldots,\nu},$$

and therefore, Q_n is a Hankel matrix whose entries decay rapidly from the bottom right element ($\approx h_n$) to the top left element ($\approx h_n^{2\nu+1}$) (see Figure 4). This means that the system responds in a highly anisotropic way to step-size h: the covariance increase for $x^{(\nu)}$ at each step is much larger than for, e.g., $x^{(0)}$, which is a problem for numerically stable computation of smoothing iterations. An example with precise values of the condition numbers of $\mathbf{Q}$ and $\mathbf{C}_{n+1}^-$ follows in the context of Table 2 and Figure 5 below.

In the following, we explain how to milden this ill-conditioning by a coordinate change in the state-space model (which we equivalently refer to as a preconditioner). Let $\mathbf{T} \in \mathbb{R}^{d(\nu+1)\times d(\nu+1)}$ be an invertible transformation matrix. The SDE for $\mathbf{x}^{\text{new}} := \mathbf{T}^{-1}\mathbf{x}$ is

$$\begin{cases} \mathrm{d}\mathbf{x}^{\text{new}}(t) = \mathbf{T}^{-1}\mathbf{F}\,\mathbf{T}\,\mathbf{x}^{\text{new}}(t)\,\mathrm{d}t + \mathbf{T}^{-1}\mathbf{L}\,\mathrm{d}\mathbf{w}(t), \text{ for } t \geq 0, \\ \mathbf{x}^{\text{new}}(0) \sim \mathcal{N}(\mathbf{T}^{-1}\mathbf{m}_0, \mathbf{T}^{-1}\mathbf{C}_0\mathbf{T}^{-\top}). \end{cases} \tag{10}$$

The optimal choice of $\mathbf{T}$ will depend on step-size h_n. Therefore we write $\mathbf{T}_n := \mathbf{T}(h_n)$ and use a different coordinate change at each filtering/smoothing step. The equivalent discretisation of the continuous model in Eq. (10) is

$$\begin{cases} \mathbf{x}_{n+1}^{\text{new}} \mid \mathbf{x}_n^{\text{new}} \sim \mathcal{N}(\mathbf{T}_n^{-1}\mathbf{A}_n\mathbf{T}_n\mathbf{x}_n, \mathbf{T}_n^{-1}\mathbf{Q}_n\mathbf{T}_n^{-\top}), \text{ for } n = 0, \ldots, N, \\ \mathbf{x}_0^{\text{new}} \sim \mathcal{N}(\mathbf{T}_0^{-1}\mathbf{m}_0, \mathbf{T}_0^{-1}\mathbf{C}_0\mathbf{T}_0^{-\top}). \end{cases} \tag{11}$$

The measurement model changes as well. It now reads

$$\ell(\mathbf{x}_n^{\text{new}}) \approx \delta \left[\overline{\mathbf{H}}\, \mathbf{T}_n\, \mathbf{x}_n^{\text{new}} - \overline{\mathbf{b}} \right], \tag{12}$$

which assumes that the state $\mathbf{x}_n^{\text{new}}$ "lives in the preconditioned space". $\overline{\mathbf{H}}$ and $\overline{\mathbf{b}}$ are derived by linearising $f(\mathbf{E}_0^\top \mathbf{T}_n \circ)$ with a zeroth or first order Taylor approximation at $\mathbf{m}_n$ (recall Equations (6) and (7)). The filtering and smoothing iterations are changed accordingly. A detailed implementation guide is in Appendix A.

Next, we propose such a coordinate change. For ν-times integrated Wiener processes, the transition matrix A_n has a closed-form representation (Schober et al., 2019),

$$\mathbf{A}_n = A_n \otimes I_d, \quad A_n = \left[\mathbb{I}_{i \leq j} \frac{h^{j-i}}{(j-i)!} \right]_{i,j=0,\ldots,\nu}$$

where $\mathbb{I}_{i \leq j} = 1$ if $i \leq j$ and $\mathbb{I}_{i \leq j} = 0$ otherwise. If in Equations (11) and (12), $\mathbf{T}_n$ is chosen as

$$\mathbf{T}_n := T_n \otimes I_d, \quad T_n := \sqrt{h_n}\, \text{diag}\left(\frac{h_n^\nu}{\nu!}, \frac{h_n^{\nu-1}}{(\nu-1)!}, \ldots, h_n, 1 \right), \tag{13}$$

the dependency of $\mathbf{A}_n$ and $\mathbf{Q}_n$ on h_n is removed, because those two matrices can be factorised as

$$\mathbf{A}_n = T_n\, \overline{A}\, T_n^{-1} \otimes I_d, \quad \mathbf{Q}_n = T_n\, \overline{Q}\, T_n^\top \otimes \Gamma. \tag{14}$$

Applying the coordinate change to $\mathbf{A}_n$ and $\mathbf{Q}_n$ leaves only $\overline{\mathbf{A}} = \overline{A} \otimes I_d$ and $\overline{\mathbf{Q}} = \overline{Q} \otimes \Gamma$, because $\mathbf{T}_n^{-1}$ and $\mathbf{T}_n$ cancel each other out (compare Eq. (11) to Eq. (14)). The upper triangular matrix $\overline{A}$ as well as the Hankel matrix $\overline{Q}$ are available in closed form. They are

$$\overline{A} := \left[\binom{\nu - i}{\nu - j} \right]_{i,j=0,\ldots,\nu} \quad \text{and} \quad \overline{Q} := \left[\frac{1}{2\nu + 1 - i - j} \right]_{i,j=0,\ldots,\nu},$$

where the elements of $\overline{A}$ are binomial coefficients. Removing the h_n-dependency from the discretisation is crucial, because (i) the elements—and hence, the condition number—of the process noise covariance are independent of the step-size and (ii) this transformation can be computed in closed form and applied to an $d(\nu+1) \times d(\nu+1)$ matrix in complexity $\mathcal{O}(d^2(\nu+1)^2)$, which is neglibile if viewed in the context of the matrix-matrix operations in each ODE solver step. The cheap application of the preconditioner is of utmost importance because ODE solver implementations need to be fast. We refer to Section 3.4 for a more thorough complexity analysis.

This transformation implies that, although we store the values $(x, \dot{x}, \ldots, x^{(\nu)})$, we work in the rescaled coordinates $(h^{-\nu} x\, \nu!, \ldots, x^{(\nu)})$. Even though the coordinate systems are different, this is conceptually related to and, in fact, inspired by the Nordsieck representation of a vector $(x, \ldots, h^\nu x^{(\nu)}/\nu!)$ (Nordsieck, 1962). Such a representation, which we refer to as Nordsieck coordinates, was used by Schober et al. (2019) to show that the zeroth-order linearisation filter (EK0) is a multi-step method with time-varying weights. It has therefore been proven useful to analyse the probabilistic ODE solver. The evaluation below will show that a variant of this change additionally solves problems of numerical stability.

Order, ν	$\log_{10}[\text{cond}(Q)]$			$\log_{10}[\rho]$			$\log_{10}[\min(\lambda_i(Q))]$		
	Prop.	Nord.	None	Prop.	Nord.	None	Prop.	Nord.	None
1	**1.3**	**1.3**	9.1	**0.5**	**0.5**	8.5	**-1.2**	-5.2	-13.1
3	**4.2**	4.3	28.9	**0.8**	1.3	26.4	**-4.0**	-9.1	↯
5	**7.2**	7.6	43.7	**1.0**	2.3	45.2	**-7.0**	-14.1	↯
7	**10.2**	11.0	57.3	**1.2**	3.4	64.6	**-10.0**	-19.8	↯
9	**13.2**	14.5	68.5	**1.3**	4.5	84.4	**-13.0**	-25.9	↯
11	**16.2**	17.4	79.9	**1.4**	5.6	104.6	**-16.0**	↯	↯

Table 2: **Conditioning of the preconditioned process noise covariance.** Evaluation of the condition number of the process noise covariance $\mathbf{Q}_h$ after preconditioning (left column), the ratio of largest element and the smallest element $\rho = \max_{ij}(\mathbf{Q}_h)_{ij} / \min_{ij}(\mathbf{Q}_h)_{ij}$ of this matrix (middle column; all elements are positive), and the smallest eigenvalue (right column). Evaluated are the proposed coordinated change ("Prop."), Nordsieck coordinates ("Nord.") and no preconditioning ("None"). The latter two are computed with $h = 10^{-4}$, which we argue to be a realistic scenario for an ODE solver; the former is step-size independent. Values are displayed in $\log_{10}$ basis and rounded to a single decimal. The "best" values (i.e. smallest in magnitude) are bold-faced—they all use the proposed coordinate change. NaN's are marked with a lightning (↯), which occurs if the logarithm of a negative number is taken—numerically, the matrix is not positive definite anymore.

The proposed change improves the condition number of the process noise covariance $\mathbf{Q}_n$ and the predictive covariance $\mathbf{C}_{n+1}^-$ more than Nordsieck coordinates do; see Table 2 (and Figure 5; more on this below). Without preconditioning, numerical instability is severe for $\nu > 1$; the competition between Nordsieck coordinates and the proposed change is close.

An evaluation of the effect of different coordinate changes on the condition numbers of the predictive covariances $(\mathbf{C}_{n+1}^-)_{n=0,\ldots,N}$ is displayed in Figure 5. From this experiment, two conclusions are evident: (i) non-preconditioned systems have large condition numbers; (ii) Nordsieck coordinates and the proposed preconditioner both remedy this problem, though Nordsieck coordinates perform worse than the transformation from Equation (13). This is aligned with the information in Table 2.

In summary: the ratio of the elements and eigenvalues in the preconditioned process noise covariance, as well as the condition number of the predictive covariance, speak in favour of using the proposed transformation $\mathbf{T}_n$ over Nordsieck coordinates, though both are better than no preconditioning. The condition numbers of the preconditioned process noise covariance matrix $\mathbf{Q}$ and predicted covariance matrix $\mathbf{C}_n^-$ are still large. But in practice, especially in combination with the square-root implementation that is detailed in the next section, the linear systems involving $\mathbf{C}_n^-$ are sufficiently well-conditioned such that the implementation scheme in Appendix A reliably computes approximate ODE solutions.

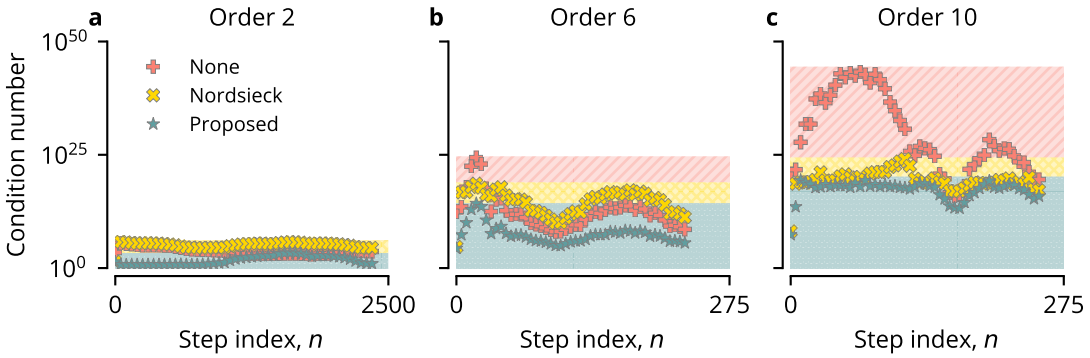

Figure 5: **Conditioning of the Cholesky factors of the predictive covariances on a test-problem.** Condition numbers of the Cholesky factors of the predictive covariances "as seen by the solver"; that is, an ODE solution is computed with adaptive step-sizes and tolerance 10^{-4} using the proposed coordinate change and at each step, the condition number of "what would have been predicted in Nordsieck/original coordinates" is computed. Evaluated for orders $\nu = 2$ (a), $\nu = 6$ (b), and $\nu = 10$ (c). Comparison of no preconditioning (red plus), Nordsieck coordinates (yellow cross) and the proposed change (blue star). The maximum condition number on this interval decides whether the smoothing iteration fails or succeeds. The discrepancy between the maximal condition numbers is shaded in red (None vs. Nordsieck), and yellow (Nordsieck vs. Proposed). The range between 0 and the maximum condition number of the proposed change is shaded blue. The underlying ODE is the Lotka-Volterra model in the parameterisation from the experiments below.

3.3 Square-Root Kalman Filter

Even with good initialisation and rescaled coordinates, numerical instability affects the implementation negatively. The reason is that the covariance matrices may have some negative eigenvalues due to round-off errors and finite precision arithmetic. Classical ODE solvers do not have this problem, because they do not provide the same uncertainty quantification. Probabilistic ODE solvers propagate uncertainty estimates in the form of covariance matrices $\mathbf{C}_0$, ..., $\mathbf{C}_N$, which need to be statistically valid, i.e. symmetric and positive (semi-)definite, even for small steps and high orders.

Symmetric, positive definite matrices allow Cholesky factorisations. Symmetric, positive semidefinite matrices do not allow Cholesky factorisations but matrix square-roots (e.g. computed with an LDL decomposition, a close relative of a Cholesky decomposition). Matrices that need to be inverted will be guaranteed to be positive definite (see Appendix A); for intermediate calculations, any matrix square root is sufficient.

If the filtering algorithm tracks matrix square-roots instead of full covariance matrices and applies all subsequent linear algebra operations to these square-roots only, positive (semi-)definiteness and symmetry are preserved throughout the entire iteration. This is the square-root Kalman filter. According to Grewal and Andrews (2014, p. 18), it dates back to Potter and Stern (1963), and is known to solve numerical instability issues (Grewal and Andrews, 2014, Chapter 7).

Let $\mathbf{C}_n = \mathbf{L}_C \mathbf{L}_C^\top$ be any matrix square-root factorisation of the covariance $\mathbf{C}_n$. The subscript n is omitted in $\mathbf{L}_C$ for readability reasons. Similarly, let $\mathbf{Q}_n = \mathbf{L}_Q \mathbf{L}_Q^\top$. Then, the right-hand side of Equation (9), which computes the predicted covariance, is the product of two $d(\nu+1) \times 2d(\nu+1)$ matrices

$$\mathbf{C}_{n+1}^- = \begin{pmatrix} \mathbf{A}_n \mathbf{L}_C & \mathbf{L}_Q \end{pmatrix} \begin{pmatrix} \mathbf{L}_C^\top \mathbf{A}_n^\top \\ \mathbf{L}_Q^\top \end{pmatrix}.$$

The QR decomposition factorises $(\mathbf{A}_n \mathbf{L}_C, \mathbf{L}_Q)^\top$ as

$$\begin{pmatrix} \mathbf{L}_C^\top \mathbf{A}_n^\top \\ \mathbf{L}_Q^\top \end{pmatrix} = \mathbf{X} \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix},$$

for an orthogonal matrix $\mathbf{X} \in \mathbb{R}^{d(\nu+1) \times d(\nu+1)}$ (the variable name "$\mathbf{Q}$" is already assigned to the process noise covariance) and an upper triangular matrix that stacks an upper triangular matrix $\mathbf{R}$ on top of zeros. $\mathbf{R}^\top$ is the Cholesky factor of $\mathbf{C}_{n+1}^-$,

$$\mathbf{C}_{n+1}^- = \begin{pmatrix} \mathbf{A}_n \mathbf{L}_C & \mathbf{L}_Q \end{pmatrix} \begin{pmatrix} \mathbf{L}_C^\top \mathbf{A}_n^\top \\ \mathbf{L}_Q^\top \end{pmatrix} = \begin{pmatrix} \mathbf{R}^\top & 0 \end{pmatrix} \mathbf{X}^\top \mathbf{X} \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix} = \mathbf{R}^\top \mathbf{R}.$$

The matrix $\mathbf{R}$ is unique up to multiplication with the matrix $\mathrm{diag}(\pm 1, ..., \pm 1)$. Multiplying such a matrix to $\mathbf{R}$ from the left and to $\mathbf{X}$ from the right ensures that the diagonal of $\mathbf{R}$ is always positive, which makes it a valid Cholesky factor, while preserving a valid QR decomposition, because the orthogonal matrix remains orthogonal.

The same trick can be applied to computing the Cholesky factor of the product of matrices $\mathbf{H}\mathbf{L}\mathbf{L}^\top\mathbf{H}^\top$ where $\mathbf{H} \in \mathbb{R}^{d(\nu+1)}$ is not quadratic. This is important for the update step of the EK0 or EK1. We refer to Appendix A below.

3.4 Computational Complexity

Assembly of $\mathbf{A}_n$ and $\mathbf{Q}_n$ at each step is replaced with pre-computation of $\overline{\mathbf{A}}$ and $\overline{\mathbf{Q}}$, which saves valuable computing time. $\mathbf{T}_n$ is diagonal and therefore, applying (and undoing) the preconditioner is cheap.

Inversion of covariance matrices, which is required for assembly of Kalman- and smoothing-gain, respectively, is expedited because of the readily computed Cholesky decomposition. The complexity of computing a QR decomposition of a matrix $M \in \mathbb{R}^{m \times n}$, $m \geq n$, is $\mathcal{O}(mn^2)$ (Higham, 2008, Table C.2), and thus in the same class as matrix-matrix multiplication. The latter is a prominent operation in the Kalman filter, so the asymptotical complexity of a single step of the ODE solver remains unaffected by the choice of square-root implementation over the "classical" implementation. In practice, the QR decomposition slightly increases the run time of the algorithm. Future work may consider implementing an efficient QR decomposition that exploits the sparsity pattern in e.g. $(\overline{\mathbf{A}}\mathbf{L}_n, \mathbf{L}_Q)^\top$, where the bottom half is triangular. In light of gaining numerical stability to the point where previously unfeasible algorithms can be implemented robustly, a small increase in computing time seems affordable.

3.5 Summary

This concludes the list of implementation tricks that are necessary to implement high-order probabilistic ODE solvers. Section 3.1 introduced accurate initialisation with Taylor-mode automatic differentiation, which is an automatic differentiation framework "tailored" to the propagation of truncated Taylor series; Section 3.2 explained that with a small twist on classical Nordsieck vector coordinate systems, numerical stability concerns in an ODE solver are step-size independent; Section 3.3 explained how to change the implementation of the filter step in order to track only the matrix square-roots of covariance matrices, which ensures positive semidefiniteness and symmetry throughout the iteration. A detailed, step-by-step implementation guide using all three proposed modifications is contained in Appendix A. Next, in Section 4, the effectiveness of the new scheme will be demonstrated.

4. Results

This section investigates how the proposed changes affect the computation of ODE solutions with high-order probabilistic solvers. At first, we show work-precision diagrams for the Lotka-Volterra system. This is a simple ODE problem, which can be computed to high precision with most ODE solvers. We hope to see rapid convergence for high-order methods, for both EK0 and EK1. We compare the probabilistic EK1 solver to SciPy implementations of Runge-Kutta methods. Afterwards, we repeat the same benchmarks on the three-body problem, which is a tougher ODE to solve than Lotka-Volterra, the SIR model, which is a non-periodic problem, and a stiff version of the van-der-Pol system. For all simulations, all three improvements from Section 3 were required for the solvers to compute the solutions reliably (recall Table 1).

We evaluate the final time error, which measures the discrepancy between the approximate ODE solution and a reference ODE solution at the final time point $t = T$. The final time error is the same error measure for both filtering and smoothing implementations;

thus, by considering this error we can relate to the convergence rates by Kersting et al. (2020b) who consider only filtering algorithms. We also approximate the root-mean-square error (RMSE) over time on an equidistant grid. This grid is different to the grid that is used for the computation of the solution (which uses adaptive step-size selection). The RMSE is an approximation of the L^2 distance and is chosen to show off the numerical stability of the smoothing steps. If both smoothing and dense output are numerically stable *and* convergence rates of roughly h^ν are matched, the implementation is sufficiently stable.

We compute reference solutions with LSODA (Hindmarsh and Petzold, 2005), and tolerance $\epsilon_{\text{abs}} = 10^{-14} = \epsilon_{\text{rel}} = 10^{-14}$. LSODA is chosen, because (i) it has automatic stiffness detection and switching, which copes well with the wide range of problems considered in the experiments, and (ii) because it is neither RK45 nor DOP853 and thus does not bias the work-precision diagrams.

4.1 Lotka-Volterra

We begin the experiments by numerically integrating the Lotka-Volterra predator-prey model (Lotka, 1978),

$$\begin{aligned}\dot{x}_1(t) &= 0.5x_1(t) - 0.05x_1(t)x_2(t)\\ \dot{x}_2(t) &= -0.05x_2(t) + 0.5x_1(t)x_2(t),\end{aligned}$$

from $t_0 = 0$ to $T = 20$, initialised at $x_1(0) = x_2(0) = 20$. x_1 is the number of prey and x_2 is the number of predators. The coefficients describe the interaction of the two species.

Convergence rates for EK0 and EK1 are shown in Figure 6, where the RMSE is plotted against the largest step, also known as fill distance. High-order convergence rates are visible for both EK0 and EK1 and all depicted orders—even for $\nu = 10$.

Strictly speaking, this demonstration of EK0 convergence does not fall into the setting of the convergence rates analysed by Kersting et al. (2020b), because we use a time-varying diffusion model. Nevertheless, the visible convergence rates of at least h^ν in Figure 6 confirm the numerical stability of the implementations and strengthen the conjecture by Kersting et al. (2020b) about the generalisability of their convergence rates from $\nu = 1$ to $\nu \gg 1$. EK1 is neither part of the analysis by Kersting et al. (2020b), which describes zeroth-order linearisation (EK0), nor part of the theory by Tronarp et al. (2021), which is concerned with the maximum-a-posteriori estimate (which can be computed by iterated extended Kalman smoothing). Though, arguably, one might speculate that similar convergence rates hold for EK1.

Next, we evaluate the performance of EK1 against Runge-Kutta implementations in SciPy. The results are depicted in Figure 7 and confirm the efficiency of the scheme. The probabilistic solver, based on EK1, exhibits a convergence rate of order 11 for $\nu = 10$—the work-precision curve of $\nu = 10$ in Figure 7 is almost vertical—which is beyond the capabilities of SciPy's ODE solver suite. The probabilistic solver by a factor ~ 10 faster than SciPy's Runge–Kutta methods of comparable orders, which can be attributed to just-in-time compilation.

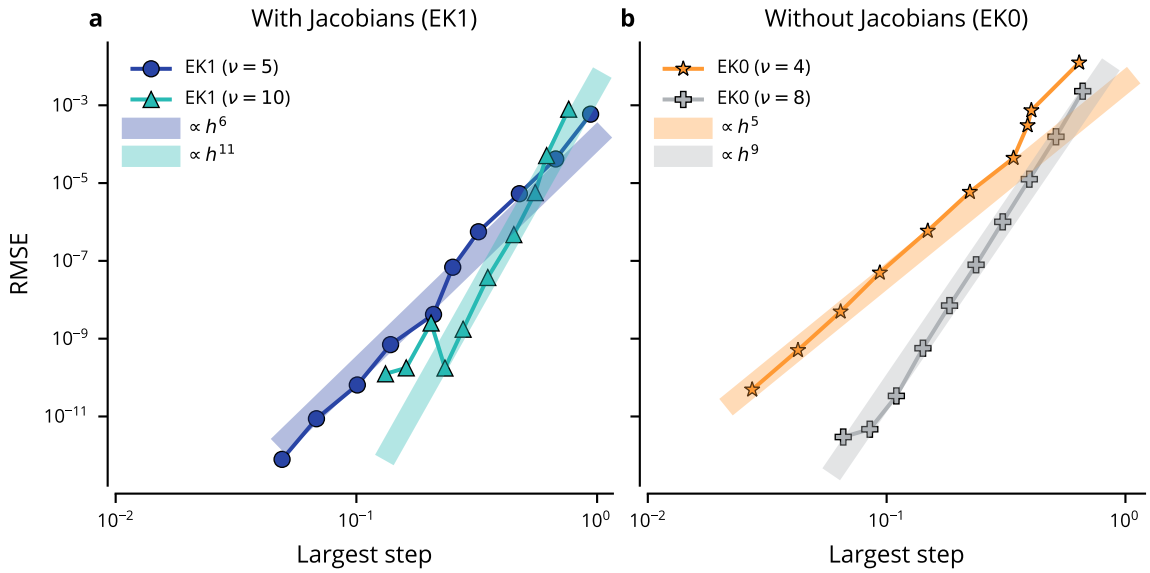

Figure 6: **EK0 and EK1 on Lotka-Volterra.** Convergence rates for the probabilistic ODE solver using EK1 and orders $\nu = 5, 11$ (a); and using EK0 and orders $\nu = 5, 8$. Convergence rates of at least h^ν hold, even for $\nu = 11$.

4.2 Three-Body

In the second example, we try the numerical solution of the restricted three-body problem as described by Hairer et al. (1993). It models the trajectory of a body in the gravitational system between the moon and earth. Let $\mu_1 = 0.012277471$ be the standardised moon-mass and $\mu_2 := 1 - \mu_1$. The solution of the ODE

$$\begin{aligned}
\ddot{x}_1(t) &= x_1(t) + 2\dot{x}_2(t) - \mu_2 \frac{x_1(t) + \mu_1}{D_1(t)} - \mu_1 \frac{x_1(t) - \mu_2}{D_2(t)} \\
\ddot{x}_2(t) &= x_2 - 2\dot{x}_1(t) - \mu_2 \frac{x_2(t)}{D_1(t)} - \mu_1 \frac{x_2(t)}{D_2(t)} \\
D_1(t) &= ((x_1(t) + \mu_1)^2 + x_2(t)^2)^{3/2} \\
D_2(t) &= ((x_1(t) - \mu_2)^2 + x_2(t)^2)^{3/2}
\end{aligned}$$

is periodic on $t \in [t_0, T] = [0, 17.065216560157962558891720624 9]$, if initialised with $x_1(0) = 0.994$, $x_2(0) = 0$, $\dot{x}_1(0) = 0$, and $\dot{x}_2(0) = -2.001585106379082522405378622 24$. Every decimal in $\dot{x}_2(0)$ and μ_1 respectively matters—if ignored, the solution is not periodic. This problem, although classified by Hairer et al. (1993) as non-stiff, is a much more challenging simulation than Lotka-Volterra. f has two singularities (at $(x_1, x_2) = (-\mu_1, 0)$ and at $(x_1, x_2) = (\mu_2, 0)$ respectively), and close to those singularities, much smaller steps are required to achieve given accuracy, than far away from the singularities.

Even on this comparably tough problem, high polynomial convergence rates seem to hold; see Figure 8. EK1 exhibits stable h^ν convergence even for $\nu = 11$. For EK0, the same

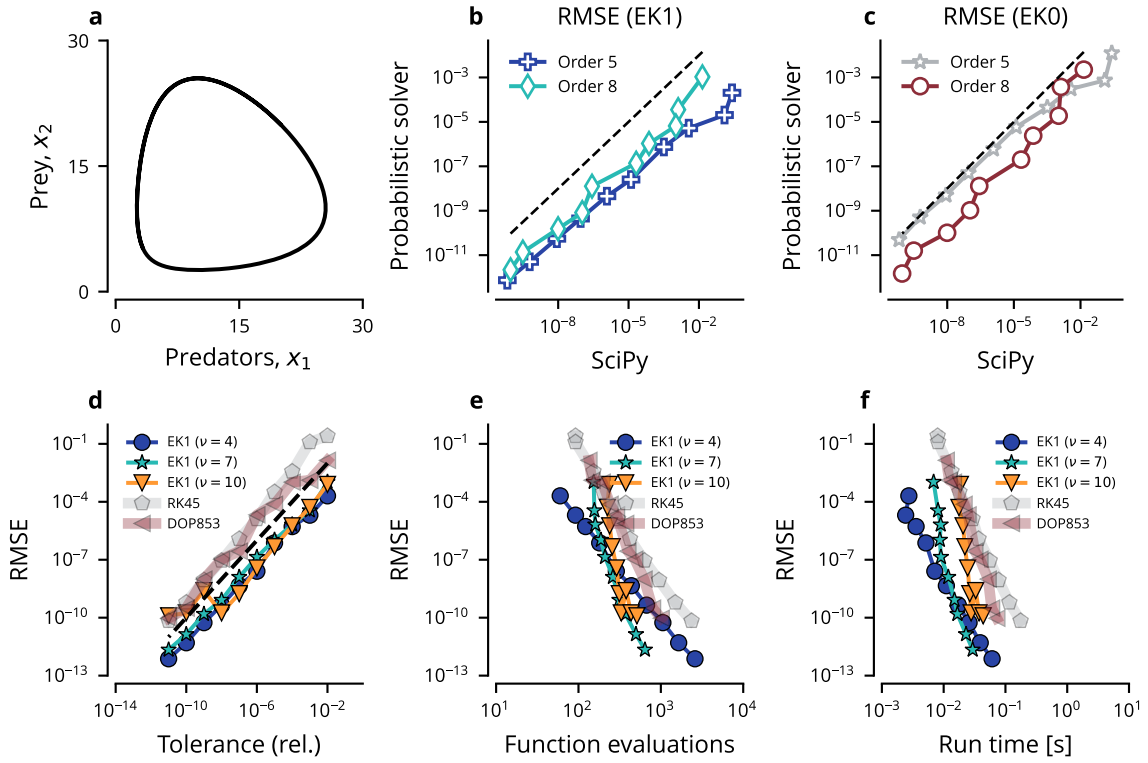

Figure 7: **Detailed results on Lotka-Volterra.** The reference solution is periodic (a). Both the EK0- and the EK1-solver converge to the reference solution as fast as SciPy's Runge-Kutta methods (b, c) The error responds well to a user-specified tolerance (d). In terms of function evaluations and time, *the solver converges fast, especially for high orders (e, f).*

rates are visible for $\nu = 4, 5$. Higher orders of EK0 still converged, but adaptive step-size selection was less efficient than for $\nu = 5$. For both solvers, we see *faster* convergence than h^ν; like in Figure 6 (Lotka-Volterra), we observe rate $h^{\nu+1}$. We do not investigate this faster-than-expected convergence further in this work.

Order $\nu = 11$ is the maximum order that was feasible in all the experiments, because the preconditioned process noise covariance matrix $\overline{Q}$ cannot be Cholesky-decomposed in standard double-precision due to ill-conditioning.

More detailed simulation results are depicted in Figure 9. We compare the runtime and accuracy of the probabilistic EK1-solver against reference Runge-Kutta implementations in SciPy. The performance of probabilistic solvers seems to be comparable to well-established, non-probabilistic solvers.

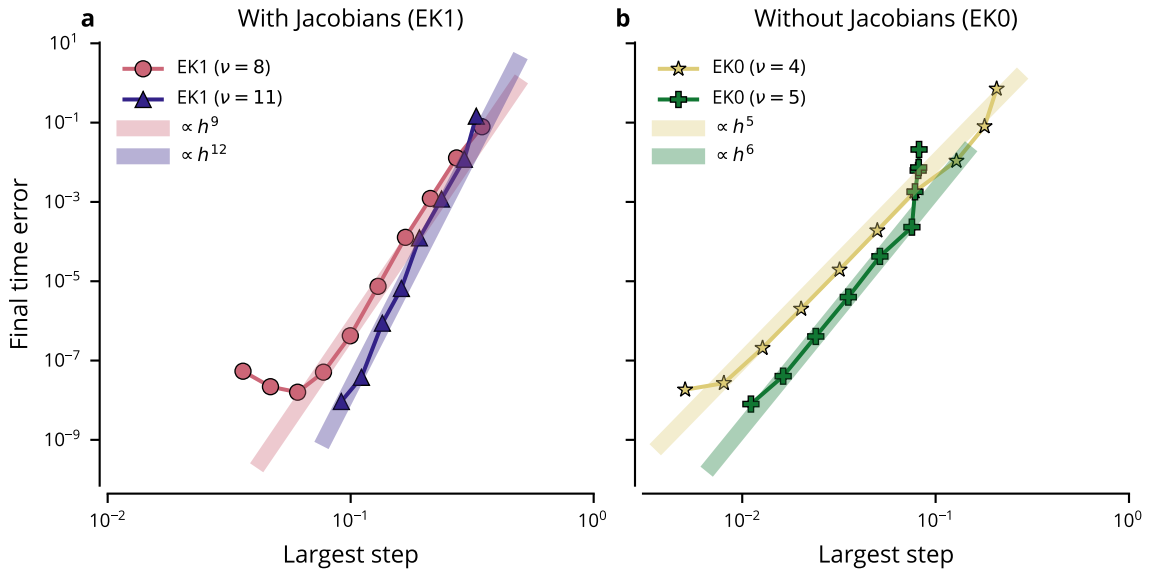

Figure 8: **EK0 and EK1 on the three-body problem.** Convergence rates for the probabilistic ODE solver using EK1 and orders $\nu = 8, 11$ (a); and using EK0 and orders $\nu = 4, 5$ (b). Again, convergence rates of at least h^ν hold, even for $\nu = 11$. This time, we evaluate the error at the final time point $t = T$. Both solvers exhibit reliable, fast convergence for all depicted orders.

4.3 SIR model

So far, we only considered systems of differential equations with periodic solutions. As a next example problem, we compute approximate solutions of the SIR model (Kermack and McKendrick, 1927),

$$\dot{S}(t) = -\beta S(t)I(t)/N, \quad \dot{I}(t) = \beta S(t)I(t)/N - \gamma I(t), \quad \dot{R}(t) = \gamma I(t),$$

for contact rate β, and incubation rate γ. The SIR model describes evolution of the number of susceptible, infected, and recovered people in a population of constant size $N = S(t) + I(t) + R(t)$ during an epidemic. In our experiments, we choose $\beta = 0.3$, $\gamma = 0.1$, $S(0) = 998$, $I(0) = 1$, and $R(0) = 1$, which implies $N = 1000$. We simulate from time 0 to time 200. The results are in Figure 10. Like on the previous two problems, high-order probabilistic solvers converge rapidly on the SIR model, even for a prior with $\nu = 11$ derivatives.

4.4 Stiff van der Pol

Finally, we return to the van-der-Pol problem from Equation (8), in order to benchmark the solver on a stiff ODE. Stiff ODEs are challenging problems and require stable ODE solvers. We choose $\mu = 10^5$, and leave the remaining parameters equal to those that lead to Figure 2. The results are in Figure 11. Altogether, the results confirm that the findings from the

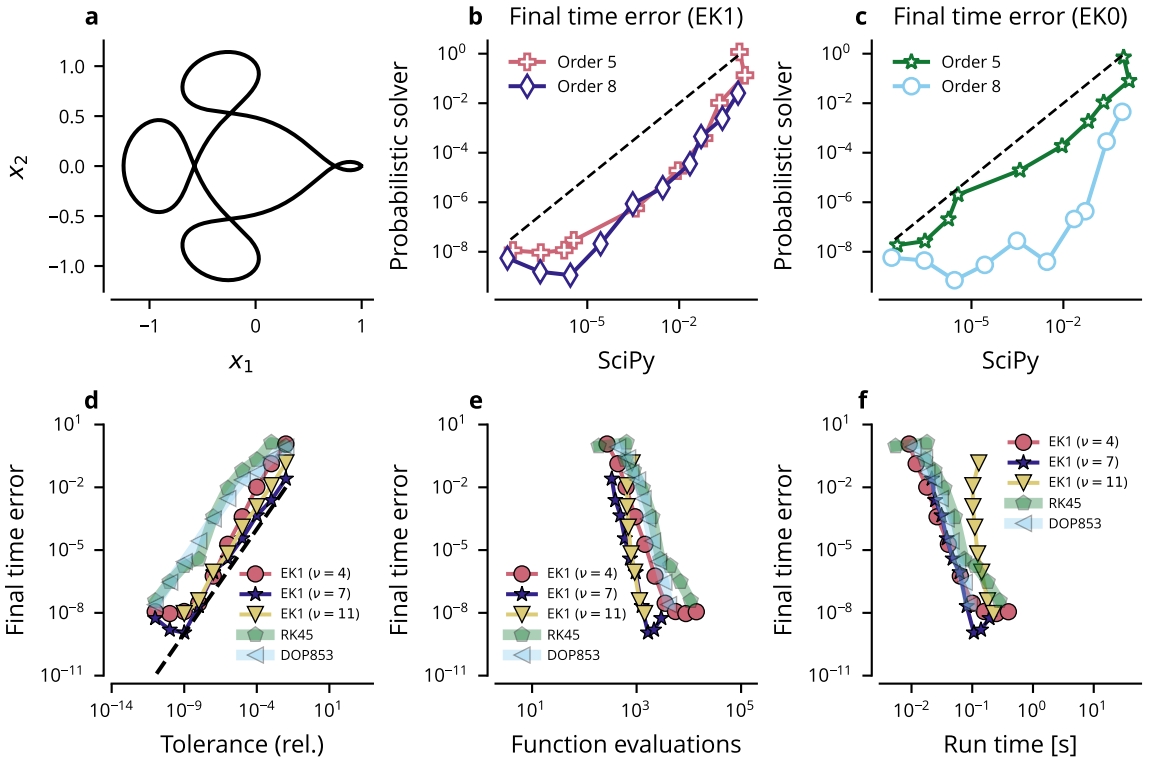

Figure 9: **Detailed results on the three-body problem.** A successful solution is periodic (a). The probabilistic EK1-solvers of orders $\nu = 5$ and $\nu = 8$ match the performance of SciPy implementations of Runge-Kutta methods of equal order exactly (RK45, DOP853; b, c), which is both measured in the root-mean-square error (RMSE; b) and in the error at final time $t = T$ (c). The RMSE improves proportionally to the user-specified tolerance (d), though it appears to lack behind with factor ~ 10. The same applies to Runge-Kutta solvers, whose lines are hardly visible in (d) because they have so much overlap with the markers of the probabilistic solvers. Solvers of order $\nu = 5, 8, 11$ converge fast (e, f). The runtime is proportional to the SciPy implementations of Runge-Kutta methods, up to factor ~ 10 (f).

previous experiments also hold for challenging, stiff problems. The EK0 cannot be used on this problem, likely because it does not possess the A-stability of the EK1.

4.5 Summary of the Experiments

Implementation was numerically stable for both EK0 and EK1 and all depicted orders $1 \leq \nu \leq 11$ in a way that (i) convergence rates are visible in work-precision diagrams even for orders up to $\nu = 11$ on the three-body problem and on a stiff version of the van-der-Pol

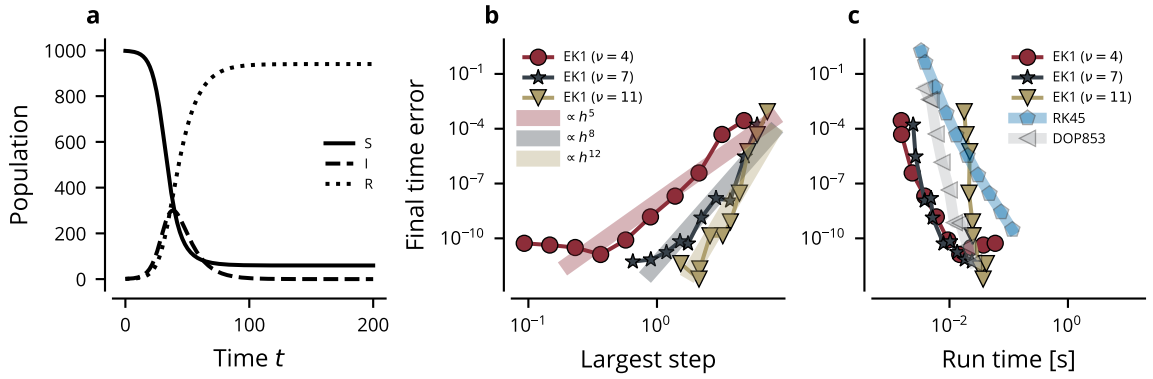

Figure 10: **Results on the SIR model.** Solution of the SIR model (a). High-order convergence rates are attained for solvers with order $\nu \in \{4, 7, 11\}$ (b), and the solver converges at least as fast as SciPy's solvers in terms of run time (wall-time; c).

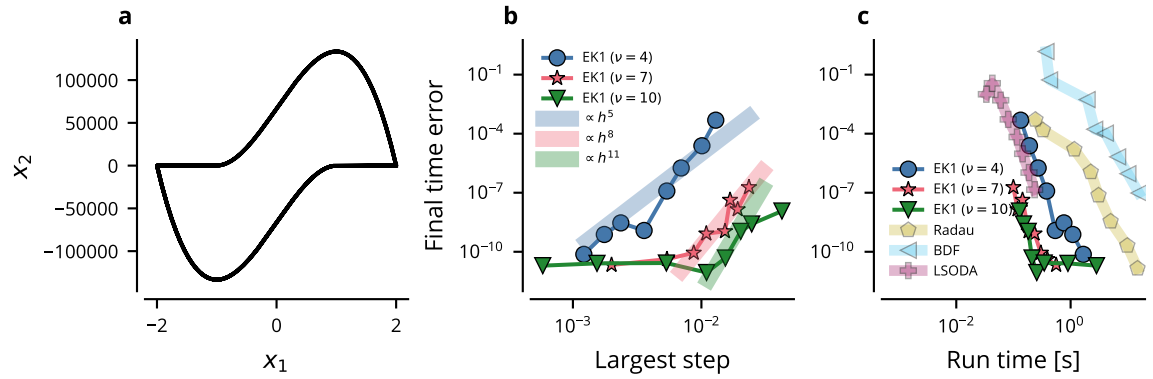

Figure 11: **Results on stiff van-der-Pol.** In the chosen parametrisation, van-der-Pol is stiff and has a periodic solution (a). But despite the stiffness, high convergence rates are attained (b), and the solver converges fast, especially compared to SciPy's implicit solvers (Radau, BDF; c).

system, and (ii) convergence is at least as fast as for Runge-Kutta methods of comparable order.

It is difficult to recommend an optimal choice of ν because this decision will likely be problem-specific. The following is based on the results in Section 4. In all scenarios, $\nu \geq 4$ was feasible. Since higher orders converge faster than lower orders, $\nu \geq 4$ seems to be an appropriate choice. The EK0 performed best with $\nu \leq 9$, so a good range for EK0 appears to be $4 \leq \nu \leq 8$. For a more involved problem, like the simulation of the three-body dynamics, $4 \leq \nu \leq 5$ was most efficient. EK1 showed rapid convergence on both non-stiff test problems for orders up to 11. Since the dimension of the state space is $d(\nu+1)$, choosing a high order comes at the price of computational complexity. This might explain why in the run-time benchmarks, orders $\nu = 5$ and $\nu = 8$ required less time to complete than order $\nu = 11$, even though the convergence rate is slower. It seems that high orders go well with low tolerances, i.e. high accuracy, but this conjecture requires further research. We summarise these findings in Table 3.

	Stiff	Non-Stiff
EK0	Do not use it.	$4 \leq \nu \leq 8$
EK1	$4 \leq \nu \leq 11$	$4 \leq \nu \leq 11$

Table 3: Which orders ν can be recommended based on the present experiments?

While the sole focus of the present investigation was showing that even if benchmarked "as a classical method" the probabilistic ODE solver is competitive with high-order Runge-Kutta methods, at this point we would like to recall that with the probabilistic algorithm, uncertainty quantification in the form of a posterior covariance comes for free—that is, computation of this quantity is *already* contained in the run-time analysis detailed above. The value of this uncertainty quantification for solving inverse problems has been demonstrated by Kersting et al. (2020a).

5. Discussion

The presented transformations evidently allow computation of ODE solutions with a probabilistic ODE filter/smoother and orders $\nu \gg 1$, which to the best of the authors' knowledge has not been possible before. Limits are given only by numerical (i.e. asymptotic, that is, A-, B- or L-) stability of the algorithm and computational efficiency for high-dimensional or stiff problems. These are questions that are not only important for the probabilistic solver but need theoretical analysis for many other methods, too. The presented guide enables empirical research on answering these questions.

The experiments show that Taylor-mode AD, a coordinate change in the state space, and square-root implementation of the filter are an improved implementation in terms of numerical stability, even over the Nordsieck-transformation that is mentioned by Schober et al. (2019), a variant of which has been used in `ProbNum`, a collection of probabilistic numerical algorithms in Python. The implementation tricks have been made available in `ProbNum` (Wenger et al., 2021) and have been adopted by related software implementations of probabilistic ODE solvers since.[1]

1. `https://github.com/pnkraemer/tornadox`, `https://github.com/nathanaelbosch/ProbNumDiffEq.jl`

In summary: the presented tricks effectively remove a barrier in computing probabilistic ODE solutions when it comes to high-order algorithms and small steps. This allows using probabilistic ODE solvers as a drop-in replacement for other, high-order, rapidly converging, classical algorithms, thereby enriching chains of statistical computation that involve numerical simulation of dynamical systems with cheap yet effective uncertainty quantification—all of which is now possible without losing out on speed or reliability of the simulation.

Acknowledgments

The authors gratefully acknowledge financial support by the European Research Council through ERC CoG Action 101123955 ANUBIS ; the DFG Cluster of Excellence "Machine Learning - New Perspectives for Science", EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); the DFG SPP 2298 (Project HE 7114/5-1), and the Carl Zeiss Foundation, (project "Certification and Foundations of Safe Machine Learning Systems in Healthcare"), as well as funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting N. Krämer.

They are grateful to Nathanael Bosch and Filip Tronarp for many valuable discussions. They further thank Hans Kersting, Jonathan Schmidt, Marius Hobbhahn, and Elizabeth Baker for helpful feedback on the manuscript.

Appendix A. Implementation Guide

The following explains detailed iteration schemes of the probabilistic ODE solver, including initialisation (A.1), prediction (A.2), update (A.3), and smoothing (A.4).

A.1 Initialisation

Choose an order ν (recommendations were made in the discussion in Section 5). Initialise the ODE solver with Taylor-mode automatic differentiation. The covariance has zeros, respectively. Before the first step, assemble $\overline{\mathbf{A}}$ and decompose $\overline{\mathbf{Q}}$ into its Cholesky factors, $\overline{\mathbf{Q}} = \mathbf{L}_Q \mathbf{L}_Q^\top$. For high orders, this remains a numerical bottleneck, because even $\overline{\mathbf{Q}}$ is ill-conditioned for large ν; recall Table 2. If not all derivatives are initialised accurately, set the respective entries of the initial covariance $\mathbf{C}_0$ to a non-zero value and decompose it into its Cholesky factors, $\mathbf{C}_0 = \mathbf{L}_0 \mathbf{L}_0^\top$ (using the LDL decomposition if necessary).

A.2 Prediction

Mean and covariance are stored in the original, non-transformed coordinates. Therefore, the update step consists of (i) applying the transformation

$$\overline{\mathbf{m}}_n = \mathbf{T}_n^{-1} \mathbf{m}_n, \quad \overline{\mathbf{L}}_n = \mathbf{T}_n^{-1} \mathbf{L}_n$$

and (ii) computing the prediction in the changed coordinate system,

$$\begin{aligned} \mathbf{m}_{n+1}^{-} &= \overline{\mathbf{A}}\,\overline{\mathbf{m}}_n, \\ \left(\overline{\mathbf{A}}\,\overline{\mathbf{L}}_n, \mathbf{L}_Q\right)^\top &= \mathbf{X}\,\mathbf{R}_{n+1} \\ \mathbf{L}_{n+1}^{-} &= (\mathbf{R}_{n+1})_{0:d(\nu+1)}^\top \end{aligned} \tag{17}$$

where Equation (17) is a QR decomposition. $\mathbf{X}$ is discarded. The notation $(\mathbf{R}_{n+1})_{0:d(\nu+1)}^\top$ implies that the top $d(\nu+1) \times d(\nu+1)$ block of $\mathbf{R}_{n+1}$ is extracted and transposed.

A.3 Update

The predicted mean and covariance "live in the preconditioned space". The update consists of a measurement step and a conditioning step. The measurement step starts with assembling either $\mathbf{H} = \mathbf{E}_1^\top \mathbf{T}_n$ (EK0) or $\mathbf{H} = \mathbf{E}_1^\top \mathbf{T}_n - \nabla f(\mathbf{E}_0^\top \mathbf{T}_n \mathbf{m}_{n+1}^{-}) \mathbf{E}_0^\top \mathbf{T}_n$ (EK1), and continues with computing

$$\begin{aligned} \mathbf{z}_{n+1} &= \mathbf{E}_1^\top \mathbf{T}_n \mathbf{m}_{n+1}^{-} - f(\mathbf{E}_0^\top \mathbf{T}_n \mathbf{m}_{n+1}^{-}) \\ (\mathbf{H}\mathbf{L}_{n+1}^{-})^\top &= \mathbf{X}\,\mathbf{R}_{n+1}^{S} \\ \mathbf{L}_S &= (\mathbf{R}_{n+1}^{S})_{0:d(\nu+1)}^\top \\ \mathbf{C}_{\text{cross}} &= \mathbf{L}_{n+1}^{-} (\mathbf{L}_{n+1}^{-})^\top \mathbf{H}^\top. \end{aligned}$$

Other than in Equation (11), $\mathbf{T}_n$ is part of $\mathbf{H}$ now. The conditioning step is

$$\begin{aligned} \mathbf{K}_{n+1} &= \mathbf{C}_{\text{cross}} \mathbf{L}_S^{-\top} \mathbf{L}_S^{-1} \\ \overline{\mathbf{m}}_{n+1} &= \mathbf{m}_{n+1}^{-} - \mathbf{K}_{n+1} \mathbf{z}_{n+1} \\ \overline{\mathbf{L}}_{n+1} &= (\mathbf{I} - \mathbf{K}_{n+1}\mathbf{H}) \mathbf{L}_{n+1}^{-}. \end{aligned}$$

Since each of $\mathbf{H}$, $\mathbf{T}_n$ and $\mathbf{L}_{n+1}^{-}$ are of full rank, $\mathbf{L}_S$ is invertible. Inversion of $\mathbf{L}_S \mathbf{L}_S^\top$ leverages the readily computed Cholesky-decomposition. After the respective update, $\overline{\mathbf{m}}_{n+1}$ and $\overline{\mathbf{L}}_{n+1}$ still "live in the preconditioned space". Therefore, they need to be transformed back to the original coordinates

$$\mathbf{m}_{n+1} = \mathbf{T}_n \overline{\mathbf{m}}_{n+1}, \quad \mathbf{L}_{n+1} = \mathbf{T}_n \overline{\mathbf{L}}_{n+1},$$

before storing them. $\mathbf{L}_{n+1}$ is not necessarily triangular or positive definite, but since it is a matrix square-root of $\mathbf{C}_{n+1}$, the posterior covariance is guaranteed to be symmetric and positive semidefinite.

A.4 Smoothing

First, all states are fetched into the "preconditioned coordinate system",

$$\begin{aligned} \overline{\mathbf{m}}_n^F &= \mathbf{T}_n^{-1} \mathbf{m}_n^F, & \overline{\mathbf{L}}_n^F &= \mathbf{T}_n^{-1} \mathbf{L}_n^F, \\ \overline{\mathbf{m}}_{n+1}^S &= \mathbf{T}_n^{-1} \mathbf{m}_{n+1}^S, & \overline{\mathbf{L}}_{n+1}^S &= \mathbf{T}_n^{-1} \mathbf{L}_{n+1}^S, \end{aligned}$$

after which the prediction step is repeated (it has to be repeated only on paper, implementations can reuse predictions from the filtering recursion),

$$\begin{aligned} \mathbf{m}_{n+1}^{-} &= \overline{\mathbf{A}}\,\overline{\mathbf{m}}_n^F \\ \left(\overline{\mathbf{A}}\,\overline{\mathbf{L}}_n^F, \mathbf{L}_Q\right)^\top &= \mathbf{X}\,\mathbf{R}_{n+1} \\ \mathbf{L}_{n+1}^{-} &= (\mathbf{R}_{n+1})_{0:d(\nu+1)}^\top, \end{aligned} \tag{21}$$

and again, Equation (21) is a QR decomposition. $\overline{\mathbf{A}}$ has full rank, $\overline{\mathbf{L}}_n^F$ is positive semidefinite, and $\mathbf{L}_Q$ is positive definite, therefore $\mathbf{L}_{n+1}^{-}$ is invertible. Second, the update is computed as

$$\begin{aligned} \mathbf{G}_{n+1} &= \overline{\mathbf{L}}_n^F \left(\overline{\mathbf{A}}\,\overline{\mathbf{L}}_n^F\right)^\top \left(\mathbf{L}_{n+1}^{-}\right)^{-\top} \left(\mathbf{L}_{n+1}^{-}\right)^{-1} \\ \overline{\mathbf{m}}_n^S &= \overline{\mathbf{m}}_n^F - \mathbf{G}_{n+1}\left[\overline{\mathbf{m}}_{n+1}^S - \mathbf{m}_{n+1}^{-}\right] \\ \left((\mathbf{I} - \mathbf{G}\overline{\mathbf{A}})\,\overline{\mathbf{L}}_n, \mathbf{G}\mathbf{L}_Q, \mathbf{G}\mathbf{L}_{n+1}^S\right)^\top &= \mathbf{X}\,\mathbf{R}_{n+1} \\ \overline{\mathbf{L}}_n^S &= (\mathbf{R}_{n+1})_{0:d(\nu+1)}^\top \end{aligned}$$

where the penultimate line is a QR decomposition that computes a Joseph-style update for the smoothing iteration; this is a counterpart to Equation (4.23) in the book by Grewal and Andrews (2014), applied to the smoothing step. Finally, before storing the values, the results are pushed back to the original coordinate system,

$$\mathbf{m}_n^S = \mathbf{T}_n \overline{\mathbf{m}}_n^S, \quad \mathbf{L}_n^S = \mathbf{T}_n \overline{\mathbf{L}}_n^S.$$

This concludes the smoothing step.

We emphasise that at least on paper, the outcome of these steps is identical to the outcome of ODE filters and smoothers in the standard implementation. In practice, the results may differ, though, because of the accumulation of round-off errors in the "classical" implementation.

References

Assyr Abdulle and Giacomo Garegnani. Random time step probabilistic methods for uncertainty quantification in chaotic and geometric numerical integration. *Statistics and Computing*, 2020.

Georgios Arvanitidis, Soren Hauberg, Philipp Hennig, and Michael Schober. Fast and robust shortest paths on manifolds learned from data. In *Proceedings of Machine Learning Research*, pages 1506–1515. PMLR, 2019.

Jesse Bettencourt, Matthew J Johnson, and David Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in JAX. 2019.

Nathanael Bosch, Philipp Hennig, and Filip Tronarp. Calibrated adaptive probabilistic ODE solvers. In *International Conference on Artificial Intelligence and Statistics*, pages 3466–3474. PMLR, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python + NumPy programs, 2018. *URL http://github. com/google/jax*, page 18, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

Oksana A Chkrebtii, David A Campbell, Ben Calderhead, and Mark A Girolami. Bayesian solution uncertainty quantification for differential equations. *Bayesian Analysis*, 11:1239–1267, 2016.

Jon Cockayne, Chris J Oates, TJ Sullivan, and Mark Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 2019.

Patrick R Conrad, Mark Girolami, Simo Särkkä, Andrew Stuart, and Konstantinos Zygalakis. Statistical analysis of differential equations: introducing probability measures on numerical solutions. *Statistics and Computing*, 27:1065–1082, 2017.

Germund G Dahlquist. A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3(1):27–43, 1963.

Mohinder S Grewal and Angus P Andrews. *Kalman filtering: Theory and Practice with MATLAB*. John Wiley & Sons, 2014.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

John Guckenheimer. Dynamics of the van der Pol equation. *IEEE Transactions on Circuits and Systems*, 27(11):983–989, 1980.

Kjell Gustafsson, Michael Lundh, and Gustaf Söderlind. A PI stepsize control for the numerical solution of ordinary differential equations. *BIT Numerical Mathematics*, 28(2): 270–287, 1988.

Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I – Nonstiff Problems*. Springer, 1993.

Philipp Hennig and Søren Hauberg. Probabilistic solutions to differential equations and their application to Riemannian statistics. In *Artificial Intelligence and Statistics*, pages 347–355, 2014.

Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, 2015.

Nicholas J Higham. *Functions of Matrices: Theory and Computation*. SIAM, 2008.

AC Hindmarsh and LR Petzold. LSODA: Ordinary differential equation solver for stiff or non-stiff system. 2005.

Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. In *Advances in Neural Information Processing Systems 33 Pre-Proceedings*, 2020.

William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115(772):700–721, 1927.

Hans Kersting and Maren Mahsereci. A Fourier state space model for Bayesian ODE filters. *Second workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models (ICML 2020), Virtual Conference*, 2020.

Hans Kersting, Nicholas Krämer, Martin Schiegg, Christian Daniel, Michael Tiemann, and Philipp Hennig. Differentiable likelihoods for fast inversion of 'likelihood-free' dynamical systems. *Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119*, 2020a.

Hans Kersting, Tim J Sullivan, and Philipp Hennig. Convergence rates of Gaussian ODE filters. *Statistics and Computing*, 30(6):1791–1816, 2020b.

Alfred J Lotka. The growth of mixed populations: two species competing for a common food supply. In *The Golden Age of Theoretical Ecology: 1923–1940*, pages 274–286. Springer, 1978.

Emilia Magnani, Hans Kersting, Michael Schober, and Philipp Hennig. Bayesian filtering for ODEs with bounded derivatives. *arXiv:1709.08471*, 2017.

Arnold Nordsieck. On numerical integration of ordinary differential equations. *Mathematics of Computation*, 16(77):22–49, 1962.

Chris J Oates and Tim J Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 29:1335–1351, 2019.

James Potter and Robert Stern. Statistical filtering of space navigation measurements. In *Guidance and Control Conference*, page 333, 1963.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv:2001.04385*, 2020.

Steven Roman. The formula of Faa di Bruno. *The American Mathematical Monthly*, 87 (10):805–809, 1980.

Simo Särkkä. *Bayesian Filtering and Smoothing*, volume 3. Cambridge University Press, 2013.

Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*, volume 10. Cambridge University Press, 2019.

Michael Schober, David Duvenaud, and Philipp Hennig. Probabilistic ODE solvers with Runge-Kutta means. In *Advances in Neural Information Processing Systems 27*, pages 739–747, 2014.

Michael Schober, Simo Särkkä, and Philipp Hennig. A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, 29:99–122, 2019.

Filip Tronarp, Hans Kersting, Simo Särkkä, and Philipp Hennig. Probabilistic solutions to ordinary differential equations as non-linear Bayesian filtering: A new perspective. *Statistics and Computing*, 29, 2019.

Filip Tronarp, Simo Särkkä, and Philipp Hennig. Bayesian ODE solvers: the maximum a posteriori estimate. *Statistics and Computing*, 31(3):1–18, 2021.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

Grace Wahba. Improper priors, spline smoothing and the problem of guarding against model errors in regression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(3):364–372, 1978.

Jonathan Wenger, Nicholas Krämer, Marvin Pförtner, Jonathan Schmidt, Nathanael Bosch, Nina Effenberger, Johannes Zenn, Alexandra Gessner, Toni Karvonen, François-Xavier Briol, et al. ProbNum: Probabilistic numerics in python. *arXiv:2112.02100*, 2021.