

Parallel-in-Time Probabilistic Numerical ODE Solvers

Nathanael Bosch

Tübingen AI Center, University of Tübingen

NATHANAEL.BOSCH@UNI-TUEBINGEN.DE

Adrien Corenflos

Department of Electrical Engineering and Automation, Aalto University

ADRIEN.CORENFLOS.STATS@GMAIL.COM

Fatemeh Yaghoobi

Department of Electrical Engineering and Automation, Aalto University

FATEMEH.YAGHOOBI@AALTO.FI

Filip Tronarp

Center for Mathematical Sciences, Lund University

FILIP.TRONARP@MATSTAT.LU.SE

Philipp Hennig

Tübingen AI Center, University of Tübingen

PHILIPP.HENNIG@UNI-TUEBINGEN.DE

Simo Särkkä

Department of Electrical Engineering and Automation, Aalto University

SIMO.SARKKA@AALTO.FI

Editor: Ryan Adams

Abstract

Probabilistic numerical solvers for ordinary differential equations (ODEs) treat the numerical simulation of dynamical systems as problems of Bayesian state estimation. Aside from producing posterior distributions over ODE solutions and thereby quantifying the numerical approximation error of the method itself, one less-often noted advantage of this formalism is the algorithmic flexibility gained by formulating numerical simulation in the framework of Bayesian filtering and smoothing. In this paper, we leverage this flexibility and build on the time-parallel formulation of iterated extended Kalman smoothers to formulate a *parallel-in-time* probabilistic numerical ODE solver. Instead of simulating the dynamical system sequentially in time, as done by current probabilistic solvers, the proposed method processes all time steps in parallel and thereby reduces the computational complexity from *linear* to *logarithmic* in the number of time steps. We demonstrate the effectiveness of our approach on a variety of ODEs and compare it to a range of both classic and probabilistic numerical ODE solvers.

Keywords: probabilistic numerics, ordinary differential equations, numerical analysis, parallel-in-time methods, Bayesian filtering and smoothing.

1. Introduction

Ordinary differential equations (ODEs) are used throughout the sciences to describe the evolution of dynamical systems over time. In machine learning, ODEs provide a continuous description of certain neural networks (Chen et al., 2018) and optimization procedures (Helmke et al., 2012; Su et al., 2016), and are used in generative modeling with normalizing flows (Papamakarios et al., 2021) and diffusion models (Song et al., 2021), among others. Unfortunately, all but the simplest ODEs are too complex to be solved analytically. Therefore, numerical methods are required to obtain a solution. While a multitude of numerical solvers

has been developed over the last century (Hairer et al., 1993; Deuffhard and Bornemann, 2012; Butcher, 2016), most commonly-used methods do not provide a quantification of their own inevitable numerical approximation error.

Probabilistic numerics provides a framework for treating classic numerical problems as problems of probabilistic inference (Hennig et al., 2015; Oates and Sullivan, 2019; Hennig et al., 2022). In the context of ODEs, methods based on Gaussian process regression (Skilling, 1992; Hennig and Hauberg, 2014) and in particular Gauss–Markov regression (Schober et al., 2019; Kersting et al., 2020; Tronarp et al., 2019) provide an efficient and flexible approach to compute posterior distributions over the solution of ODEs (Bosch et al., 2021; Krämer and Hennig, 2024), and even partial differential equations (Krämer et al., 2022; Bosch et al., 2023) and differential-algebraic equations (Bosch et al., 2022). These so-called *ODE filters* typically scale cubically in the ODE dimension (as do most *implicit* ODE solvers) and specific approximations enable linear scaling (shared by most *explicit* solvers) (Krämer et al., 2022). But to date, their linear scaling with the number of time steps remains.

For very large-scale simulations with very long time horizons, the sequential processing in time of most ODE solvers can become a bottleneck. This motivates the development of *parallel-in-time* methods: By leveraging the ever-increasing parallelization capabilities of modern computer hardware, parallel-in-time methods can achieve *sub-linear* scaling in the number of time steps (Gander, 2015). One well-known method of this kind is Parareal (Lions et al., 2001). It achieves temporal parallelism by combining an expensive, accurate solver with a cheap, coarse solver, in such a way that the fine solver is only ever applied to individual time slices in a parallel manner, leading to a square-root scaling (in ideal conditions). But, due to its sequential coarse-grid solve, Parareal still has only limited concurrency (Gander and Vandewalle, 2007), and while it has recently been extended probabilistically by Pentland et al. (2021, 2022) to improve its performance and convergence, these methods do not provide probabilistic solutions to ODEs per se.

In this paper, we leverage the time-parallel formulation of Gaussian filters and smoothers (Särkkä and García-Fernández, 2021; Yaghoobi et al., 2021, 2023) and develop a *parallel-in-time* probabilistic numerical ODE solver. The paper is structured as follows. Section 2 formulates numerical ODE solutions as Bayesian state estimation problems and presents the established, sequential, filtering-based probabilistic ODE solvers. Section 3 then presents our proposed parallel-in-time method; first as exact inference for affine ODEs, then as an iterative, approximate algorithm for general nonlinear ODEs. Section 4 then presents experiments on a variety of ODEs and compares the performance of our proposed method to that of existing (sequential) probabilistic and non-probabilistic ODE solvers. Finally, Section 5 concludes with a discussion of our results and an outlook on future work.

2. Numerical ODE Solutions as Bayesian State Estimation

Consider an initial value problem (IVP) given by a first-order ODE of the form

$$\dot{y}(t) = f(y(t), t), \quad t \in [0, T], \quad y(0) = y_0, \quad (1)$$

with vector field $f : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ and initial value $y_0 \in \mathbb{R}^d$. To capture the numerical error that arises from temporal discretization, the quantity of interest in probabilistic numerics

for ODEs is the *probabilistic numerical ODE solution*, defined as

$$p\left(y(t) \mid y(0) = y_0, \{\dot{y}(t_n) = f(y(t_n), t_n)\}_{n=1}^N\right), \quad (2)$$

for some prior $p(y(t))$ and with $\{t_n\}_{n=1}^N \subset [0, T]$ the chosen time-discretization.

In the following, we pose the probabilistic numerical ODE solution as a problem of Bayesian state estimation, and we define the prior, likelihood, data, and approximate inference scheme. For a more detailed description of the transformation of an IVP into a Gauss–Markov regression problem, refer to Tronarp et al. (2019).

2.1 Gauss–Markov Process Prior

We model the solution y of the IVP with a ν -times integrated Wiener process prior (IWP(ν)). More precisely, let $Y(t) = [Y^{(0)}(t), Y^{(1)}(t), \dots, Y^{(\nu)}(t)]$ be the solution of the following linear, time-invariant stochastic differential equation with Gaussian initial condition

$$dY^{(i)}(t) = Y^{(i+1)}(t) dt, \quad i = 0, \dots, \nu - 1, \quad (3a)$$

$$dY^{(\nu)}(t) = \sigma dW(t), \quad (3b)$$

$$Y(0) \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (3c)$$

with initial mean and covariance $\mu_0 \in \mathbb{R}^{d(\nu+1)}$, $\Sigma_0 \in \mathbb{R}^{d(\nu+1) \times d(\nu+1)}$, diffusion coefficient $\sigma \in \mathbb{R}_+$, and d -dimensional Wiener process $W : \mathbb{R} \rightarrow \mathbb{R}^d$. Then, $Y^{(i)}$ is chosen to model the i -th derivative of the IVP solution y . By construction, accessing the i -th derivative can be done by multiplying the state Y with a projection matrix $E_i := I_d \otimes e_i$, that is, $Y^{(i)}(t) = E_i Y(t)$.

This continuous-time prior satisfies discrete transition densities (Särkkä and Solin, 2019)

$$Y(t+h) \mid Y(t) \sim \mathcal{N}(\Phi(h)Y(t), \sigma^2 Q(h)), \quad (4)$$

with transition matrix and process noise covariance $\Phi(h), Q(h) \in \mathbb{R}^{d(\nu+1) \times d(\nu+1)}$ and step $h \in \mathbb{R}_+$. For the IWP(ν) these can be computed in closed form (Kersting et al., 2020), as

$$\Phi(h) = I_d \otimes \check{\Phi}(h), \quad \left[\check{\Phi}(h)\right]_{ij} = \mathbb{1}_{i=j} \frac{h^{i-j}}{(j-i)!}, \quad (5a)$$

$$Q(h) = I_d \otimes \check{Q}(h), \quad \left[\check{Q}(h)\right]_{ij} = \frac{h^{2\nu+1-i-j}}{(2\nu+1-i-j)(\nu-i)!(\nu-j)!}. \quad (5b)$$

Remark 1 (Alternative Gauss–Markov priors). *While ν -times integrated Wiener process priors have been the most common choice for filtering-based probabilistic ODE solvers in recent years, the methodology is not limited to this choice. Alternatives include the ν -times integrated Ornstein–Uhlenbeck process and the class of Matérn processes, both of which have a similar continuous-time SDE representation as well as Gaussian transition densities in discrete time. Refer to Bosch et al. (2023); Tronarp et al. (2021); Särkkä and Solin (2019).*

The initial distribution $\mathcal{N}(\mu_0, \Sigma_0)$ is chosen such that it encodes the initial condition $y(0) = y_0$. Furthermore, to improve the numerical stability and the quality of the posterior,

we initialize not only on the function value $Y^{(0)}(0) = y_0$, but also the higher order derivatives, that is, $Y^{(i)}(0) = \frac{d^i y}{dt^i}(0)$ for all $i \leq \nu$ (Krämer and Hennig, 2024). These terms can be efficiently computed via Taylor-mode automatic differentiation (Griewank, 2000; Bettencourt et al., 2019). As a result, we obtain an initial distribution with mean

$$\mu_0 = \left[y_0, \frac{dy}{dt}(0), \dots, \frac{d^\nu y}{dt^\nu}(0) \right]^T, \quad (6)$$

and zero covariance $\Sigma_0 = 0$, since the initial condition has to hold exactly.

2.2 Observation Model and Data

To relate the introduced Gauss–Markov prior to the IVP problem from Equation (1), we define an observation model in terms of the information operator

$$\mathcal{Z}[y](t) := \dot{y}(t) - f(y(t), t). \quad (7)$$

By construction, $\mathcal{Z}$ maps the true IVP solution y *exactly* to the zero function, that is, $\mathcal{Z}[y] \equiv 0$. In terms of the continuous process Y , the information operator can be expressed as

$$\mathcal{Z}[Y](t) = E_1 Y(t) - f(E_0 Y(t), t), \quad (8)$$

where E_0 and E_1 are the projection matrices introduced in Section 2.1 which select the zeroth and first derivative from the process Y , respectively. There again, if Y corresponds to the true IVP solution (and its true derivatives), then $\mathcal{Z}[Y] \equiv 0$.

Conversely, inferring the true IVP solution requires conditioning the process $Y(t)$ on $Z(t) = 0$ over the whole continuous interval $t \in [0, T]$. Since this is in general intractable, we instead condition $Y(t)$ only on discrete observations $Z(t_n) = 0$ on a grid $\mathbb{T} = \{t_n\}_{n=1}^N$. This leads to the Dirac likelihood model commonly used in ODE filtering (Tronarp et al., 2019):

$$Z(t_n) \mid Y(t_n) \sim \delta \left(Y^{(1)}(t_n) - f \left(Y^{(0)}(t_n), t_n \right) \right), \quad (9)$$

with zero-valued data $Z(t_n) = 0$ for all $t_n \in \mathbb{T}$.

Remark 2 (Information operators for other differential equation problems). *Similar information operators can be defined for other types of differential equations that are not exactly of the first-order form as given in Equation (1), such as higher-order differential equations, Hamiltonian dynamics, or differential-algebraic equations (Bosch et al., 2022).*

2.3 Discrete-Time Inference Problem

The combination of prior, likelihood, and data results in a Bayesian state estimation problem

$$Y(0) \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (10a)$$

$$Y(t_{n+1}) \mid Y(t_n) \sim \mathcal{N}(\Phi(t_{n+1} - t_n)Y(t_n), Q(t_{n+1} - t_n)), \quad (10b)$$

$$Z(t_n) \mid Y(t_n) \sim \delta \left(Y^{(1)}(t_n) - f \left(Y^{(0)}(t_n), t_n \right) \right), \quad (10c)$$

with zero data $Z(t_n) = 0$ for all $t_n \in \mathbb{T}$. The posterior distribution over $Y^{(0)}(t)$ then provides a probabilistic numerical ODE solution to the given IVP, as formulated in Equation (2).

This is a standard nonlinear Gauss–Markov regression problem, for which many approximate inference algorithms have previously been studied (Särkkä and Svensson, 2023). In the context of probabilistic ODE solvers, a popular approach for efficient approximate inference is Gaussian filtering and smoothing, where the solution is approximated with Gaussian distributions

$$p(Y(t) \mid \{Z(t_n) = 0\}_{n=1}^N) \approx \mathcal{N}(\mu(t), \Sigma(t)). \quad (11)$$

This is most commonly performed with extended Kalman filtering (EKF) and smoothing (EKS) (Schober et al., 2019; Tronarp et al., 2019; Kersting et al., 2020); though other methods have been proposed, for example based on numerical quadrature (Kersting and Hennig, 2016) or particle filtering (Tronarp et al., 2019). *Iterated* extended Kalman smoothing (e.g. Bell, 1994; Särkkä and Svensson, 2023) computes the “maximum a posteriori” estimate of the probabilistic numerical ODE solution (Tronarp et al., 2021). This will be the basis for the parallel-in-time ODE filter proposed in this work, explained in detail in Section 3.

2.4 Practical Considerations for Probabilistic Numerical ODE Solvers

While Bayesian state estimation methods such as the extended Kalman filter and smoother can, in principle, be directly applied to the formulated state estimation problem, there are a number of modifications and practical considerations that should be taken into account:

- *Square-root formulation:* Gaussian filters often suffer from numerical stability issues when applied to the ODE inference problem defined in Equation (10), in particular when using high orders and small steps, due to the ill-conditioning of the state transition covariance Q and numerical round-off error from finite precision arithmetic (Krämer and Hennig, 2024). To alleviate these issues, probabilistic numerical ODE solvers are typically formulated in square-root form (Krämer and Hennig, 2024); this is also the case for the proposed parallel-in-time method.
- *Preconditioned state transitions:* Krämer and Hennig (2024) suggest a coordinate change preconditioner to make the state transition matrices step-size independent and thereby improve the numerical stability of EKF-based probabilistic ODE solvers. This preconditioner is also used in this work.
- *Uncertainty calibration:* The Gauss–Markov prior as introduced in Section 2.1 has a free parameter, the diffusion σ , which directly influences the uncertainty estimates returned by the ODE filter, but not its mean estimates. In this paper, we compute a quasi-maximum likelihood estimate for the parameter σ post-hoc, as suggested by Tronarp et al. (2019).
- *Approximate linearization:* Variants of the standard EKF/EKS-based inference have been proposed in which the linearization of the vector-field is done only approximately. Approximating the Jacobian of the ODE vector field with a zero matrix enables inference with a complexity which scales only linearly with the ODE dimension (Krämer et al., 2022), while still providing polynomial convergence rates (Kersting

et al., 2020). A diagonal approximation of the Jacobian preserves the linear complexity, but improves the stability properties of the solver (Krämer et al., 2022). In this work, we only consider the exact first-order Taylor linearization.

- *Local error estimation and step-size adaptation:* Rather than predefining the time discretization grid, certain solvers employ an adaptive approach where the solver dynamically constructs the grid while controlling an internal estimate of the numerical error. Step-size adaptation based on *local* error estimates have been proposed for both classic (Hairer et al., 1993, Chapter II.4) and probabilistic ODE solvers (Schober et al., 2019; Bosch et al., 2021). On the other hand, *global* step-size selection is often employed in numerical boundary value problem (BVP) solvers (Ascher et al., 1995, Chapter 9), and has been extended to filtering-based probabilistic BVP solvers (Krämer and Hennig, 2021). For our purposes, we will focus on fixed grids.

3. Parallel-in-Time Probabilistic Numerical ODE Solvers

This section develops the main method proposed in this paper: a parallel-in-time probabilistic numerical ODE solver.

3.1 Time-Parallel Exact Inference in Affine Vector Fields

Let us first consider the simple case: An initial value problem with affine vector field

$$\dot{y}(t) = L(t)y(t) + d(t), \quad t \in [0, T], \quad y(0) = y_0. \quad (12)$$

The corresponding information model of the probabilistic solver is then also affine, with

$$Z(t) \mid Y(t) \sim \delta(H(t)Y(t) - d(t)), \quad (13a)$$

$$H(t) := E_1 - L(t)E_0. \quad (13b)$$

Let $\mathbb{T} = \{t_n\}_{n=1}^N \subset [0, T]$ be a discrete time grid. To simplify the notation in the following, we will denote a function evaluated at time t_n by a subscript n , that is $Y(t_n) =: Y_n$, except for the transition matrices where we will use $\Phi_n := \Phi(t_{n+1} - t_n)$ and $Q_n := Q(t_{n+1} - t_n)$. Then, the Bayesian state estimation problem from Equation (10) reduces to inference of $Y(t)$ in the model

$$Y_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (14a)$$

$$Y_{n+1} \mid Y_n \sim \mathcal{N}(\Phi_n Y_n, Q_n), \quad (14b)$$

$$Z_n \mid Y_n \sim \delta(H_n Y_n - d_n), \quad (14c)$$

with zero data $Z_n = 0$ for all $n = 1, \dots, N$. Since this is an affine Gaussian state estimation problem, it can be solved exactly with Gaussian filtering and smoothing (Kalman, 1960; Rauch et al., 1965; Särkkä and Svensson, 2023); see also (Tronarp et al., 2019, 2021) for explicit discussions of probabilistic numerical solvers for affine ODEs.

Recently, Särkkä and García-Fernández (2021) presented a time-parallel formulation of Bayesian filtering and smoothing, as well as a concrete algorithm for exact linear Gaussian filtering and smoothing—which could be directly applied to the problem formulation in

Equation (14). But as mentioned in Section 2.4, the resulting ODE solver might suffer from numerical instabilities. Therefore, we use the square-root formulation of the time-parallel linear Gaussian filter and smoother by Yaghoobi et al. (2023). In the following, we review the details of the algorithm.

3.1.1 TIME-PARALLEL GENERAL BAYESIAN FILTERING AND SMOOTHING

First, we follow the presentation of Särkkä and García-Fernández (2021) and formulate Bayesian filtering and smoothing as prefix sums. We define elements $a_n = (f_n, g_n)$ with

$$f_n(Y_n | Y_{n-1}) = p(Y_n | Z_n, Y_{n-1}), \quad (15a)$$

$$g_n(Y_{n-1}) = p(Z_n | Y_{n-1}), \quad (15b)$$

where for $n = 1$ we have $p(Y_1 | Z_1, Y_0) = p(Y_1 | Z_1)$ and $p(Z_1 | Y_0) = p(Z_1)$, together with a binary filtering operator $\otimes_f : (f_i, g_i) \otimes_f (f_j, g_j) \mapsto (f_{ij}, g_{ij})$ defined by

$$f_{ij}(x | z) := \frac{\int g_j(y) f_j(x | y) f_i(y | z) dy}{\int g_j(y) f_i(y | z) dy}, \quad (16a)$$

$$g_{ij}(z) := g_i(z) \int g_j(y) f_i(y | z) dy. \quad (16b)$$

Then, Särkkä and García-Fernández (2021, Theorem 3) show that $\otimes_f$ is associative and that

$$a_1 \otimes_f \cdots \otimes_f a_n = \begin{bmatrix} p(Y_n | Z_{1:n}) \\ p(Z_{1:n}) \end{bmatrix}, \quad (17)$$

that is, the filtering marginals and the marginal likelihood of the observations at step n are the results of a cumulative sum of the elements $a_{1:n}$ under $\otimes_f$. Since the filtering operator $\otimes_f$ is associative, this quantity can be computed in parallel with prefix-sum algorithms, such as the parallel scan algorithm by Blelloch (1989).

Remark 3 (On Prefix-Sums). *Prefix sums, also known as cumulative sums or inclusive scans, play an important role in parallel computing. Their computation can be efficiently parallelized and, if enough parallel resources are available, their (span) computational cost can be reduced from linear to logarithmic in the number of elements. One such algorithm is the well-known parallel scan algorithm by Blelloch (1989) which, given N elements and $N/2$ processors, computes the prefix sum in $2\lceil \log_2 N \rceil$ sequential steps with $2N - 2$ invocations of the binary operation. This algorithm is implemented in both tensorflow (Abadi et al., 2015) and JAX (Bradbury et al., 2018); the latter is used in this work.*

The time-parallel smoothing step can be constructed similarly: We define elements $b_n = p(Y_n | Z_{1:n}, Y_{n+1})$, with $b_N = p(Y_N | Z_{1:N})$, and a binary smoothing operator $b_i \otimes_s b_j = b_{ij}$, with

$$b_{ij}(x | z) = \int b_i(x | y) b_j(y | z) dy. \quad (18)$$

Then, $\otimes_s$ is associative and the smoothing marginal at time step n is the result of a reverse cumulative sum of the elements $b_{n:N}$ under $\otimes_s$ (Särkkä and García-Fernández, 2021):

$$b_n \otimes_s \cdots \otimes_s b_N = p(Y_n | Z_{1:N}). \quad (19)$$

Again, since the smoothing operator $\otimes_s$ is associative, this cumulative sum can be computed in parallel with a prefix-sum algorithm (Blelloch, 1989).

3.1.2 TIME-PARALLEL LINEAR GAUSSIAN FILTERING IN SQUARE-ROOT FORM

In the linear Gaussian case, the filtering elements $a_n = (f_n, g_n)$ can be parameterized by a set of parameters $\{A_n, b_n, C_n, \eta_n, J_n\}$ as follows:

$$f_n(Y_n | Y_{n-1}) = p(Y_n | Z_n, Y_{n-1}) = \mathcal{N}(Y_n; A_n Y_{n-1} + b_n, C_n), \quad (20a)$$

$$g_n(Y_{n-1}) = p(Z_n | Y_{n-1}) \propto \mathcal{N}_I(Y_{n-1}; \eta_n, J_n), \quad (20b)$$

where $\mathcal{N}_I$ denotes a Gaussian density parameterized in information form, that is, $\mathcal{N}_I(x; \eta, J) = \mathcal{N}(x; J^{-1}\eta, J^{-1})$. The parameters $\{A_n, b_n, C_n, \eta_n, J_n\}$ can be computed explicitly from the given state-space model (Särkkä and García-Fernández, 2021, Lemma 7). But since probabilistic numerical ODE solvers require a numerically stable implementation of the underlying filtering and smoothing algorithm (Krämer and Hennig, 2024), we formulate the time-parallel linear Gaussian filtering algorithm in square-root form, following Yaghoobi et al. (2023).

To this end, let $\sqrt{M}$ denote a left square-root of a positive semi-definite matrix M , that is, $\sqrt{M}\sqrt{M}^T = M$; the matrix $\sqrt{M}$ is sometimes also called a “generalized Cholesky factor” of M (S. Grewal and P. Andrews, 2014). To operate on square-root matrices, we also define the *triangularization* operator: Given a wide matrix $M \in \mathbb{R}^{n \times m}$, $m \geq n$, the triangularization operator $\text{tria}(M)$ first computes the QR decomposition of M^T , that is, $M^T = QR$, with wide orthonormal $Q \in \mathbb{R}^{m \times n}$ and square upper-triangular $R \in \mathbb{R}^{n \times n}$, and then returns R^T . This operator plays a central role in square-root filtering algorithms as it enables the numerically stable addition of covariance matrices, provided square-roots are available. Given two positive semi-definite matrices $A, B \in \mathbb{R}^{n \times n}$ with square-roots $\sqrt{A}, \sqrt{B}$, a square-root of the sum $A + B$ can be computed as

$$\sqrt{A + B} = \text{tria}([\sqrt{A} \quad \sqrt{B}]). \quad (21)$$

With these definitions in place, we briefly review the time-parallel linear Gaussian filtering algorithm in square-root form as provided by Yaghoobi et al. (2023) in the following.

Parameterization of the filtering elements. Let $m_0 = \mu_0$, $P_0 = \Sigma_0$, and $m_n = 0$, $P_n = 0$ for all $n \geq 1$, and define

$$m_n^- = \Phi_{n-1} m_{n-1}, \quad (22a)$$

$$\sqrt{P_n} = \text{tria}([\Phi_{n-1} \sqrt{P_{n-1}} \quad \sqrt{Q_{n-1}}]). \quad (22b)$$

Then, the square-root parameterization of the filtering elements a_n is given by

$$A_n = (I - K_n H_n) \Phi_{n-1}, \quad (23a)$$

$$b_n = m_n^- - K_n (H_n m_n^- - d_n), \quad (23b)$$

$$\sqrt{C_n} = \Psi_{22}, \quad (23c)$$

$$\eta_n = \sqrt{J_n} \sqrt{S_n}^{-1} d_n, \quad (23d)$$

$$\sqrt{J_n} = \Phi_{n-1}^T H_n^T \sqrt{S_n}^{-T}, \quad (23e)$$

where I is the identity matrix and Ψ_{22} , $\sqrt{S_n}$ and K_n are defined via

$$\begin{bmatrix} \Psi_{11} & 0 \\ \Psi_{21} & \Psi_{22} \end{bmatrix} = \text{tria} \left(\begin{bmatrix} H_n \sqrt{P_n^-} & \sqrt{R_n} \\ \sqrt{P_n^-} & 0 \end{bmatrix} \right), \quad (24a)$$

$$\sqrt{S_n} = \Psi_{11}, \quad (24b)$$

$$K_n = \Psi_{21} \Psi_{11}^{-1}. \quad (24c)$$

For generality the formula includes an observation noise covariance R_n , but note that in the context of probabilistic ODE solvers we have a noiseless measurement model with $\sqrt{R_n} = 0$.

Associative filtering operator. Let a_i, a_j be two filtering elements, parameterized in square-root form by $a_i = \{A_i, b_i, \sqrt{C_i}, \eta_i, \sqrt{J_i}\}$ and $a_j = \{A_j, b_j, \sqrt{C_j}, \eta_j, \sqrt{J_j}\}$. Then, the associative filtering operator $\otimes_f$ computes the filtering element $a_{ij} = a_i \otimes_f a_j$ as

$$A_{ij} = A_j A_i - A_j \sqrt{C_i} \Xi_{11}^{-\top} \Xi_{21}^\top A_i, \quad (25a)$$

$$b_{ij} = A_j \left(I - \sqrt{C_i} \Xi_{11}^{-\top} \Xi_{21}^\top \right) (b_i + \sqrt{C_i} \sqrt{C_i}^\top \eta_j) + b_j, \quad (25b)$$

$$\sqrt{C_{ij}} = \text{tria} \left(\begin{bmatrix} A_j \sqrt{C_i} \Xi_{11}^{-\top} & \sqrt{C_j} \end{bmatrix} \right), \quad (25c)$$

$$\eta_{ij} = A_i^\top \left(I - \Xi_{21} \Xi_{11}^{-1} \sqrt{C_i}^\top \right) (\eta_j - \sqrt{J_j} \sqrt{J_j}^\top b_i) + \eta_i, \quad (25d)$$

$$\sqrt{J_{ij}} = \text{tria} \left(\begin{bmatrix} A_i^\top \Xi_{22} & \sqrt{J_i} \end{bmatrix} \right), \quad (25e)$$

where Ξ_{11} , Ξ_{21} , Ξ_{22} are defined via

$$\begin{bmatrix} \Xi_{11} & 0 \\ \Xi_{21} & \Xi_{22} \end{bmatrix} = \text{tria} \left(\begin{bmatrix} \sqrt{C_i}^\top \sqrt{J_j} & I \\ \sqrt{J_j} & 0 \end{bmatrix} \right). \quad (26)$$

See Yaghoobi et al. (2023) for the detailed derivation.

The filtering marginals. By computing a cumulative sum of the elements $a_{1:N}$ with the binary operation $\otimes_f$, we obtain the filtering marginals at time $n = 1, \dots, N$ as

$$p(Y_n | Z_{1:n}) = \mathcal{N} \left(Y_n; m_n^f, P_n^f \right), \quad \text{with} \quad m_n^f := b_{1:n}, \quad \sqrt{P_n^f} := \sqrt{C_{1:n}}. \quad (27)$$

This concludes the time-parallel linear Gaussian square-root filter.

3.1.3 TIME-PARALLEL LINEAR GAUSSIAN SMOOTHING IN SQUARE-ROOT FORM

Similarly to the filtering equations, the linear Gaussian smoothing can also be formulated in terms of smoothing elements b_n and an associative operator $\otimes_s$, and the smoothing marginals can also be computed with a parallel prefix-sum algorithm.

Parameterization of the smoothing elements. The smoothing elements b_n can be described by a set of parameters $\{E_n, g_n, \sqrt{L_n}\}$, as

$$b_n = p(Y_n | Z_{1:n}, Y_{n+1}) = \mathcal{N} \left(Y_n; E_n Y_{n+1} + g_n, \sqrt{L_n} \sqrt{L_n}^\top \right). \quad (28)$$

The smoothing element parameters can be computed as

$$E_n = \Pi_{21}\Pi_{11}^{-1}, \quad (29a)$$

$$g_n = m_n^f - E_n\Phi_n m_n^f, \quad (29b)$$

$$\sqrt{L_n} = \Pi_{22}, \quad (29c)$$

where I is the identity matrix and the matrices Π_{11} , Π_{21} , Π_{22} are defined via

$$\begin{bmatrix} \Pi_{11} & 0 \\ \Pi_{21} & \Pi_{22} \end{bmatrix} = \text{tria} \left(\begin{bmatrix} \Phi_n \sqrt{P_n^f} & \sqrt{Q_n} \\ \sqrt{P_n^f} & 0 \end{bmatrix} \right). \quad (30)$$

Associative smoothing operator. Given two smoothing elements b_i, b_j , parameterized in square-root form by $b_i = \{E_i, g_i, \sqrt{L_i}\}$ and $b_j = \{E_j, g_j, \sqrt{L_j}\}$, the associative smoothing operator $\otimes_s$ computes the smoothing element $b_{ij} = b_i \otimes_s b_j$ as

$$E_{ij} = E_i E_j, \quad (31a)$$

$$g_{ij} = E_i g_j + g_i, \quad (31b)$$

$$\sqrt{L_{ij}} = \text{tria} \left(\begin{bmatrix} E_i \sqrt{L_j} & \sqrt{L_i} \end{bmatrix} \right). \quad (31c)$$

The smoothing marginals. The smoothing marginals can then be retrieved from the reverse cumulative sum of the smoothing elements as

$$p(Y_n | Z_{1:N}) = \mathcal{N}(Y_n; m_n^s, P_n^s), \quad (32a)$$

$$m_n^s = g_{n:N}, \quad (32b)$$

$$\sqrt{P_n^s} = \sqrt{L_{n:N}}. \quad (32c)$$

Refer to Yaghoobi et al. (2023) for a thorough derivation. This concludes the time-parallel Rauch–Tung–Striebel smoother, which can be used as a parallel-in-time probabilistic numerical ODE solver for affine ODEs. The full algorithm is summarized in Algorithm 1.

Algorithm 1 Time-parallel Rauch–Tung–Striebel Smoother (ParaRTS)

Input: Initial distribution (μ_0, Σ_0) , linear transition models $\{(\Phi_n, Q_n)\}_{n=0}^{N-1}$, affine observation models $\{(H_n, d_n)\}_{n=1}^N$, data $\{Z_n\}_{n=1}^N$.

Output: Smoothing marginals $p(Y_n | Z_{1:N}) = \mathcal{N}(Y_n; m_n^s, P_n^s)$

- 1: **function** PARARTS((μ_0, Σ_0) , $\{(\Phi_n, Q_n)\}_{n=0}^{N-1}$, $\{(H_n, d_n)\}_{n=1}^N$, $\{Z_n\}_{n=1}^N$)
 - 2: *Compute the filtering elements:*
 $a_n = (A_n, b_n, \sqrt{C_n}, \eta_n, \sqrt{J_n})$ for all $n = 1, \dots, N$ $\triangleright$ Eq. (23)
 - 3: *Run the time-parallel Kalman filter:*
 $\{(A_n, b_n, \sqrt{C_n}, \eta_n, \sqrt{J_n})\}_{n=1}^N \leftarrow \text{AssociativeScan}(\otimes_f, (a_n)_{n=1}^N)$ $\triangleright$ Eq. (25)
 $p(Y_n | Z_{1:N}) = \mathcal{N}(Y_n; m_n^f, P_n^f) \leftarrow \mathcal{N}(Y_n; b_n, C_n)$ $\triangleright$ Filtering marginals
 - 4: *Compute the smoothing elements:*
 $b_n = (E_n, g_n, \sqrt{L_n})$ for all $n = 0, \dots, N$ $\triangleright$ Eq. (28)
 - 5: *Run the time-parallel Rauch–Tung–Striebel smoother:*
 $\{(E_n, g_n, \sqrt{L_n})\}_{n=1}^N \leftarrow \text{ReverseAssociativeScan}(\otimes_s, (b_n)_{n=1}^N)$ $\triangleright$ Eq. (31)
return $\mathcal{N}(Y_n; g_n, L_n)$ for all $n = 1, \dots, N$ $\triangleright$ Smoothing marginals
-

3.2 Time-Parallel Approximate Inference in Nonlinear Vector Fields

Let us now consider the general case: An IVP with nonlinear vector field

$$\dot{y}(t) = f(y(t), t), \quad t \in [0, T], \quad y(0) = y_0. \quad (33)$$

As established in Section 2, the corresponding state estimation problem is

$$Y_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (34a)$$

$$Y_{n+1} | Y_n \sim \mathcal{N}(\Phi_n Y_n, Q_n), \quad (34b)$$

$$Z_n | Y_n \sim \delta(E_1 Y_n - f(E_0 Y_n, t_n)), \quad (34c)$$

with temporal discretization $\mathbb{T} = \{t_n\}_{n=1}^N \subset [0, T]$ and zero data $Z_n = 0$ for all $n = 1, \dots, N$. In this section, we describe a parallel-in-time algorithm for solving this state estimation problem: the *time-parallel iterated extended Kalman smoother*.

3.2.1 GLOBALLY LINEARIZING THE STATE-SPACE MODEL

To make inference tractable, we will linearize the whole state-space model along a reference trajectory. And since the observation model (specified in Equation (34c)) is the only nonlinear part of the state-space model, it is the only part that requires linearization. In this paper, we only consider linearization with a first-order Taylor expansion, but other methods are possible; see Remarks 4 and 5.

For any time-point $t_n \in \mathbb{T}$, we approximate the nonlinear observation model Equation (34c) with an affine observation model by performing a first-order Taylor series expansion around a linearization point $\eta_n \in \mathbb{R}^{d(\nu+1)}$. We obtain the affine model

$$Z_n | Y_n \sim \delta(H_n Y_n - d_n), \quad (35)$$

with H_n and d_n defined as

$$H_n := E_1 - F_y(E_0 \eta_n, t_n) E_0, \quad (36a)$$

$$d_n := f(E_0 \eta_n, t_n) - F_y(E_0 \eta_n, t_n) E_0 \eta_n, \quad (36b)$$

where F_y denotes the Jacobian of f with respect to y .

In iterated extended Kalman smoothing, this linearization is performed *globally* on all time steps simultaneously along a trajectory of linearization points $\{\eta_n\}_{n=1}^N \subset \mathbb{R}^{d(\nu+1)}$. We obtain the following linearized inference problem:

$$Y_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \quad (37a)$$

$$Y_{n+1} | Y_n \sim \mathcal{N}(\Phi_n Y_n, Q_n), \quad (37b)$$

$$Z_n | Y_n \sim \delta(H_n Y_n - d_n), \quad (37c)$$

with zero data $Z_n = 0$ for all $n = 1, \dots, N$. This is now a linear state-space model with linear Gaussian observations. It can therefore be solved exactly with the numerically stable, time-parallel Kalman filter and smoother presented in Section 3.1.

Remark 4 (Linearizing with approximate Jacobians (EKO & DiagonalEK1)). *To reduce the computational complexity with respect to the state dimension of the ODE, the vector field can also be linearized with an approximate Jacobian. Established choices include $F_y \approx 0$ and $F_y \approx \text{diag}(\nabla_y f)$, which result in probabilistic ODE solvers known as the EKO and DiagonalEK1, respectively. See Krämer et al. (2022) for more details.*

Remark 5 (Statistical linear regression). *Statistical linear regression (SLR) is a more general framework for approximating conditional distributions with affine Gaussian distributions, and many well-established filters can be understood as special cases of SLR. This includes notably the Taylor series expansion used in the EKF/EKS, but also sigma-point methods such as the unscented Kalman filter and smoother (Julier et al., 2000; Julier and Uhlmann, 2004; Särkkä, 2008), and more. For more information on SLR-based filters and smoothers refer to Särkkä and Svensson (2023, Chapter 9).*

3.2.2 ITERATED EXTENDED KALMAN SMOOTHING

The IEKS (Bell, 1994; Särkkä and Svensson, 2023) is an approximate Gaussian inference method for nonlinear state-space models, which iterates between linearizing the state-space model along the current best-guess trajectory and computing a new state trajectory estimate by solving the linearized model exactly. It can equivalently also be seen as an efficient implementation of the Gauss–Newton method, applied to maximizing the posterior density of the state trajectory (Bell, 1994). This also implies that the IEKS computes not just some Gaussian estimate, but the *maximum a posteriori* (MAP) estimate of the state trajectory. In the context of probabilistic numerical ODE solvers, the IEKS has been previously explored by Tronarp et al. (2021), and the resulting MAP estimate has been shown to satisfy polynomial convergence rates to the true ODE solution. Here, we formulate an IEKS-based probabilistic ODE solver in a parallel-in-time manner, by exploiting the time-parallel formulation of the Kalman filter and smoother from Section 3.1.

The IEKS is an iterative algorithm, which starts with an initial guess of the state trajectory and then iterates between the following two steps:

1. *Linearization step:* Linearize the state-space model along the current best-guess trajectory. This can be done independently for each time step and is therefore fully parallelizable.
2. *Linear smoothing step:* Solve the resulting linear state-space model exactly with the time-parallel Kalman filter and smoother from Section 3.1.

The algorithm terminates when a stopping criterion is met, for example when the change in the MAP estimate between two iterations is sufficiently small. A pseudo-code summary of the method is provided in Algorithm 2.

As with the sequential filtering-based probabilistic ODE solvers as presented in Section 2, the mean and covariance of the initial distribution $Y(0) \sim \mathcal{N}(\mu_0, \Sigma_0)$ are chosen such that μ_0 corresponds to the exact solution of the ODE and its derivatives and Σ_0 is set to zero; see also Krämer and Hennig (2024). The initial state trajectory estimate $\{\eta_n\}_{n=0}^N$ is chosen to be constant, that is, $\eta_n = \mu_0$ for all $n = 0, \dots, N$. Note that since only $E_0 \eta_n$ is required to perform the linearization, it could equivalently be set to $\eta_n = [y_0, 0, \dots, 0]$ for all n .

Algorithm 2 Parallel-in-Time Probabilistic Numerical ODE Solver (ParaIEKS)

Input: ODE-IVP (f, y_0) , prior transition model $\{(\Phi_n, Q_n)\}_{n=1}^N$, time grid $\{t_n\}_{n=0}^N \subset [0, T]$.

Output: Probabilistic numerical ODE solution

$$p\left(y(t_n) \mid y(0) = y_0, \{\dot{y}(t_m) = f(y(t_m), t_m)\}_{m=1}^N\right) \approx \mathcal{N}(\mu_n, \Sigma_n) \text{ for } n = 1, \dots, N.$$

```

1: function PARAIEKS( $(f, y_0)$ ,  $\{(\Phi_n, Q_n)\}_{n=1}^N$ ,  $\{t_n\}_{n=0}^N$ )
2:    $\mu_0, \Sigma_0 \leftarrow \text{ComputeExactInitialState}(f, y_0)$ , 0  $\triangleright$  With automatic differentiation
3:    $\eta_n \leftarrow \mu_0$  for all  $n = 0, \dots, N$   $\triangleright$  Constant initial guess of state trajectory
4:   while stopping criterion not met do
5:      $\triangleright$  Linearize the state-space model as in Sec. 3.2.1 (fully parallelizable):  $\triangleleft$ 
6:     for  $n = 1, \dots, N$  do
7:        $H_n, d_n \leftarrow \text{LinearizeObservationModel}(f, \eta_n, t_n)$ 
8:        $\triangleright$  Run the time-parallel RTS in the linearized model as in Sec. 3.1:  $\triangleleft$ 
9:        $\{\mu_n, \Sigma_n\}_{n=1}^N \leftarrow \text{ParaRTS}((\mu_0, \Sigma_0), \{(\Phi_n, Q_n)\}_{n=1}^N, \{(H_n, d_n)\}_{n=1}^N, \{0\}_{n=1}^N)$ 
10:       $\triangleright$  Choose the mean trajectory as the next linearization trajectory:  $\triangleleft$ 
11:       $\eta_n \leftarrow \mu_n$  for all  $n = 1, \dots, N$ 
12:   return  $y(t_n) \sim \mathcal{N}(E_0 \mu_n, E_0 \Sigma_n E_0^\top)$  for  $n = 1, \dots, N$ .
```

Remark 6 (Other choices for the initial state trajectory). *In the sequential IEKS the initial state trajectory is typically computed with a standard sequential extended Rauch–Tung–Striebel smoother (Tronarp et al., 2021). But in the context of the time-parallel IEKS, we can not afford to apply a sequential method first as this would break the parallel-in-time nature of the algorithm and increase the computational complexity from logistic back to linear in the number of time points. One possible alternative choice is compute an initial state trajectory with a sequential method on a coarse time grid with $\mathcal{O}(\log(N))$ time points, similar to classic parallel-in-time methods such as Parareal (Lions et al., 2001). But, this introduces additional hyperparameters and it is only beneficial if this trajectory is indeed better than a constant guess, which often requires a certain minimum number of time points. We therefore choose the constant initial guess for the state trajectory in this paper, and leave the exploration of more sophisticated initialization strategies for future work.*

Finally, the stopping criterion should be chosen such that the algorithm terminates when the MAP estimate of the state trajectory has converged. In our experiments, we chose a combination of two criteria: (i) the change in the state trajectory estimate between two iterations is sufficiently small, or (ii) the change in the *objective value* between two iterations is sufficiently small, where the objective value is defined as the negative log-density of the state trajectory under the prior:

$$\mathcal{V}(\eta_{0:N}) = \frac{1}{2} \sum_{n=1}^N \|\eta_n - \Phi_n \eta_{n-1}\|_{Q_n^{-1}}^2. \quad (38)$$

In our experiments, we use a relative tolerance of 10^{-13} for the first criterion and absolute and relative tolerances of 10^{-9} and 10^{-6} for the second criterion, respectively. Note that this procedure is compatible with the uncertainty calibration mentioned in Section 2.4 as the mean estimator is agnostic to the value of σ (and therefore Q).

Remark 7 (Convergence of the time-parallel IEKS). *The IEKS is equivalent to the Gauss–Newton method for computing the MAP estimate of the state trajectory (Bell, 1994), and the MAP estimate has been shown to satisfy polynomial convergence rates to the true ODE solution, (Tronarp et al., 2021). And under mild conditions on the Jacobian of the vector field, the MAP is a local optimum and the Gauss–Newton method is locally convergent (Tronarp et al., 2021; Knoth, 1989). While therefore the initial guess of the state trajectory must be sufficiently close to the true solution for the method to be guaranteed to converge, in our experiments we found that the constant initialization works well in practice; see Section 4 below. In addition, the stability of the method could be further improved by using a more sophisticated optimizer, such as the Levenberg–Marquardt method or Gauss–Newton with line search (Särkkä and Svensson, 2020), or the alternating method of multipliers (ADMM; Gao et al., 2019). We leave the exploration of these methods for future work.*

3.3 Computational Complexity

Let $C_{\text{KS-step}}^s$ be the summed costs of a predict, update, and smoothing step in the sequential Kalman smoother, and let $C_{\text{KS-step}}^p$ be the corresponding cost in the time-parallel formulation (which differs by a constant factor, i.e. $C_{\text{KS-step}}^p \propto C_{\text{KS-step}}^s$). Let $C_{\text{linearize}}$ be the cost of linearizing the vector field at a single time point, which requires both evaluating the vector field and computing its Jacobian. Then, on a grid with N time points, the cost of a standard sequential extended Kalman smoother is linear in N , with

$$C_{\text{EKS}}^s = N \cdot C_{\text{linearize}} + N \cdot C_{\text{KS-step}}^s. \quad (39)$$

The computational cost of the time-parallel IEKS differs in two ways: (i) the prefix-sum formulation of the Kalman smoother enables a time-parallel inference with logarithmic complexity, and (ii) the linearization is not done locally in a sequential manner but can be performed globally, fully in parallel. In the following we assume that we have at least N processors / threads / cores available (simply referred to as “cores” hereafter), which allow us to take perfect advantage of the temporal parallelization, and we disregard the effect of the number of cores on all other parts of the algorithm (such as the influence of the number of cores on the computation time of matrix multiplications). The computational cost of the time-parallel IEKS run for k iterations is then

$$C_{\text{IEKS}}^p = k \cdot \left(C_{\text{linearize}} + 2 \log_2(N) \cdot \left(C_{\text{KS-step}}^p \right) \right). \quad (40)$$

Thus, the runtime of the time-parallel IEKS is logarithmic in the number of time points, provided a sufficient number of cores are available. To make the comparison to the sequential EKS more explicit, we consider the *speedup* of the time-parallel IEKS over the sequential EKS, defined as $S = C_{\text{EKS}}^s / C_{\text{IEKS}}^p$. By simply inserting the computational costs from above and re-arranging the terms, it can be shown that the speedup is bounded by

$$S \leq \min \left\{ \frac{N}{k} \cdot \frac{C_{\text{linearize}} + C_{\text{KS-step}}^s}{C_{\text{linearize}}}, \frac{N}{2k \log_2(N)} \cdot \frac{C_{\text{linearize}} + C_{\text{KS-step}}^s}{C_{\text{KS-step}}^p} \right\}. \quad (41)$$

This term highlights two different regimes: If the cost of linearizing the vector field is small compared to the cost of the Kalman smoother, then the speedup is approximately bounded

by $S \lesssim \frac{N}{2k \log_2 N}$ (since the sequential and parallel Kalman smoother steps have similar cost). On the other hand, if the cost of linearizing the vector field is large compared to the cost of the Kalman smoother, then the benefit of parallelizing the linearization step dominates and the speedup is bounded by $S \lesssim \frac{N}{k}$.

4. Experiments

This section investigates the utility and performance of the proposed parallel-in-time ODE filter on a range of experiments. It is structured as follows: First, Section 4.1 compares the performance of the parallel-in-time probabilistic numerical ODE solver to its sequential counterpart on multiple test problems. Section 4.2 then investigates the parallel scaling of the proposed method and evaluates it on a range of problem sizes and for different GPUs. Finally, Section 4.3 benchmarks the proposed method against other well-established ODE solvers, including both classic and probabilistic numerical methods.

In the following, we refer to all filtering-based probabilistic numerical ODE solvers by their underlying filtering and smoothing algorithms: the proposed parallel-in-time method is referred to as **ParaIEKS**, the corresponding sequential method is simply **IEKS**, and the established sequential EKS-based method is referred to as **EKS**. And, whenever relevant, we write the smoothness of the chosen IWP prior into the method name, for example as **ParaIEKS**($\nu=1$) or **EKS**($\nu=2$).

Implementation All experiments are implemented in the Python programming language with the JAX software framework (Bradbury et al., 2018). Reference solutions are computed with SciPy (Virtanen et al., 2020) and Diffrax (Kidger, 2021). Unless specified otherwise, experiments are run on an NVIDIA V100 GPU. Code for the implementation and experiments is publicly available on GitHub.¹

4.1 The Parallel-in-time Solver Compared to its Sequential Version

We first compare the proposed parallel-in-time ODE solver **ParaIEKS** to its sequential counterpart: a probabilistic solver based on the sequential implementation **IEKS**. We evaluate the solvers on three test problems: The logistic ordinary differential equation

$$\dot{y}(t) = y(t) (1 - y(t)), \quad t \in [0, 10], \quad y(0) = 0.01, \quad (42)$$

an initial value problem based on the rigid body dynamics (Hairer et al., 1993)

$$\dot{y}(t) = \begin{bmatrix} -2y_2(t)y_3(t) \\ 1.25y_1(t)y_3(t) \\ -0.5y_1(t)y_2(t) \end{bmatrix}, \quad t \in [0, 20], \quad y(0) = \begin{bmatrix} 1 \\ 0 \\ 0.9 \end{bmatrix}, \quad (43)$$

and the Van der Pol oscillator (Van der Pol, 1920)

$$\dot{y}(t) = \begin{bmatrix} y_2(t) \\ \mu ((1 - y_1(t)^2) y_2(t) - y_1(t)) \end{bmatrix}, \quad t \in [0, 6.3], \quad y(0) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \quad (44)$$

here in a non-stiff version with parameter $\mu = 1$.

1. <https://github.com/nathanaelbosch/parallel-in-time-ode-filters>

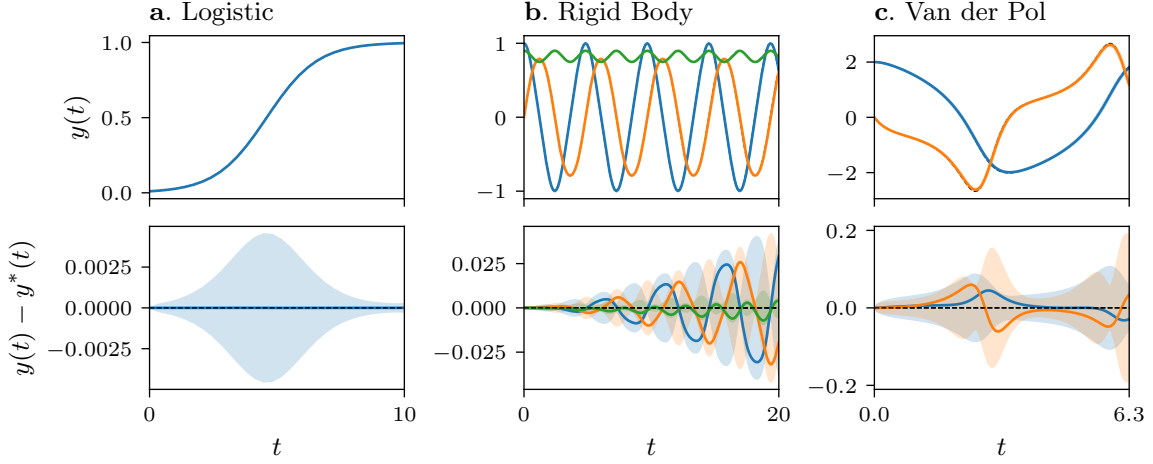


Figure 1: *Trajectories, errors, and error estimates computed by the parallel-in-time solver.* Top row: ODE solution trajectories. Visually, all three test problems seem to be solved accurately. Bottom row: Numerical errors (lines) and error estimates (shaded area). Ideally, for good calibration, the error should be of similar magnitude than the error estimate. The posterior appears underconfident on the logistic equation, and well-calibrated on the rigid body and Van der Pol problems.

We first solve the three problems with **ParaIEKS** on grids of sizes 30, 150, and 100, respectively for the logistic, rigid body, and Van der Pol problem, with a two-times integrated Wiener process prior. Reference solutions are computed with `diffax`’s **Kvaerno5** solver using adaptive steps and very low tolerances $\tau_{\{\text{abs}, \text{rel}\}} = 10^{-12}$ (Kidger, 2021; Kværnø, 2004). Figure 1 shows the resulting solution trajectories, together with numerical errors and error estimates. For these grid sizes, **ParaIEKS** computes accurate solutions on all three problems. Regarding calibration, the posterior appears underconfident for the logistic equation, as it overestimates the numerical error by more than one order of magnitude, but is reasonably confident on the rigid body and Van der Pol problems where the error estimate is of similar magnitude as the numerical error.

Next, we investigate the performance of **ParaIEKS** and compare it to its sequential implementation **IEKS**. We solve the three problems with both methods on a range of grid sizes, with both a one- and two-times integrated Wiener process prior. Reference solutions are computed with `diffax`’s **Kvaerno5** solver using adaptive steps and very low tolerances ($\tau_{\text{abs}} = 10^{-16}$, $\tau_{\text{rel}} = 10^{-13}$). Figure 2 shows the achieved root-mean-square errors (RMSE) for different grid sizes in a work-precision diagram. As expected, **ParaIEKS** and **IEKS** always achieve the same error for each problem and grid size, as both versions use the same initialization, compute the same quantities, and only differ in their filter and smoother implementation. However, the methods differ significantly in actual runtime, as shown in Figure 3. In our experiments on an NVIDIA V100 GPU, **ParaIEKS** is always strictly faster than **IEKS** across all problems, grid sizes, and priors, and we observe speed-ups of multiple orders of magnitude. Thus, when working with a GPU, **ParaIEKS** appears to be strictly superior to the sequential **IEKS**.

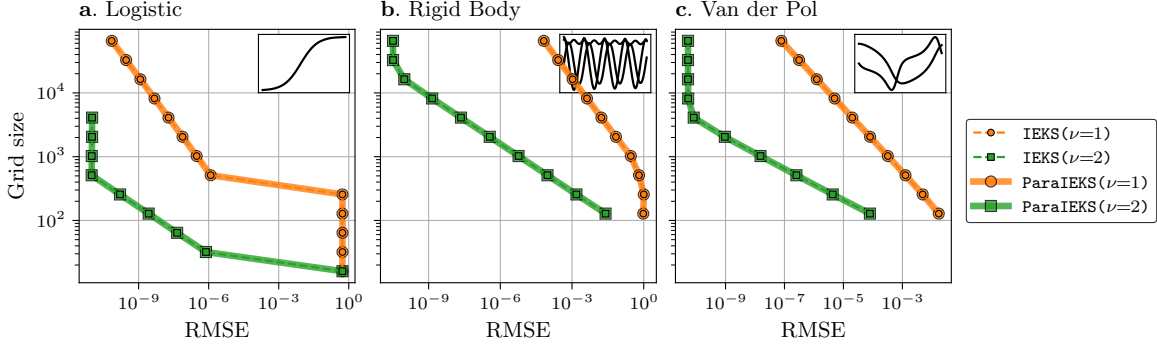


Figure 2: *The sequential and parallel IEKS compute numerically identical solutions.* For all three problems and all considered grid sizes, the sequential IEKS and the parallel ParaIEKS achieve (numerically) identical errors. This is expected, as both versions compute the same quantities and only differ in their implementation.

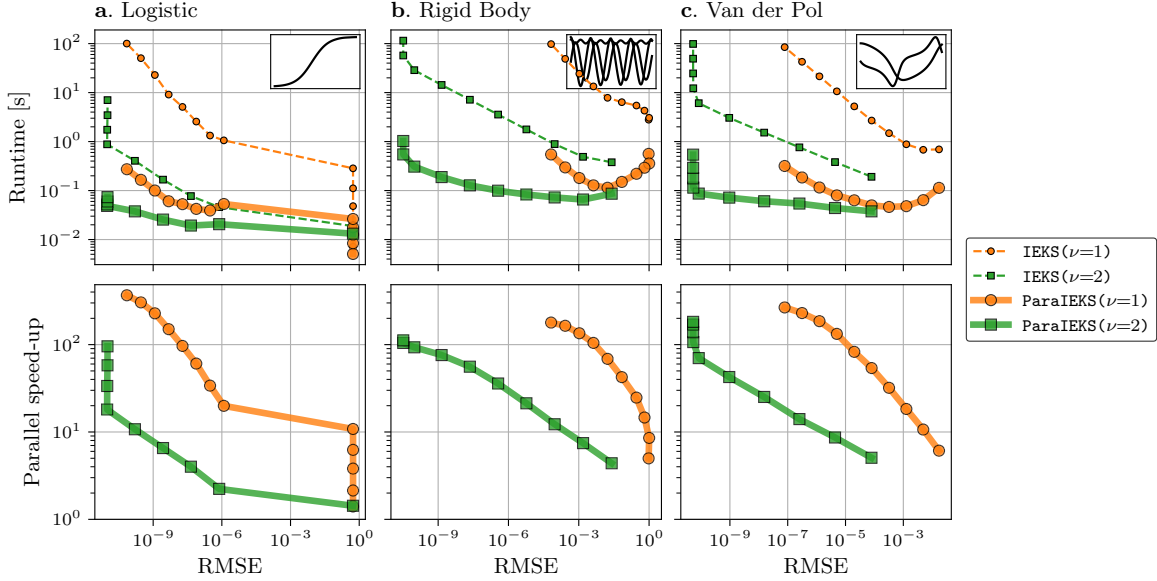


Figure 3: *Work-precision diagrams for the sequential and parallel IEKS-based ODE solver.* Top row: Runtime in seconds per error (lower-left is better). Bottom row: Speed-up of the parallel ParaIEKS over the sequential IEKS (higher is better). Across all problems, grid sizes, and priors, ParaIEKS outperforms the sequential IEKS.

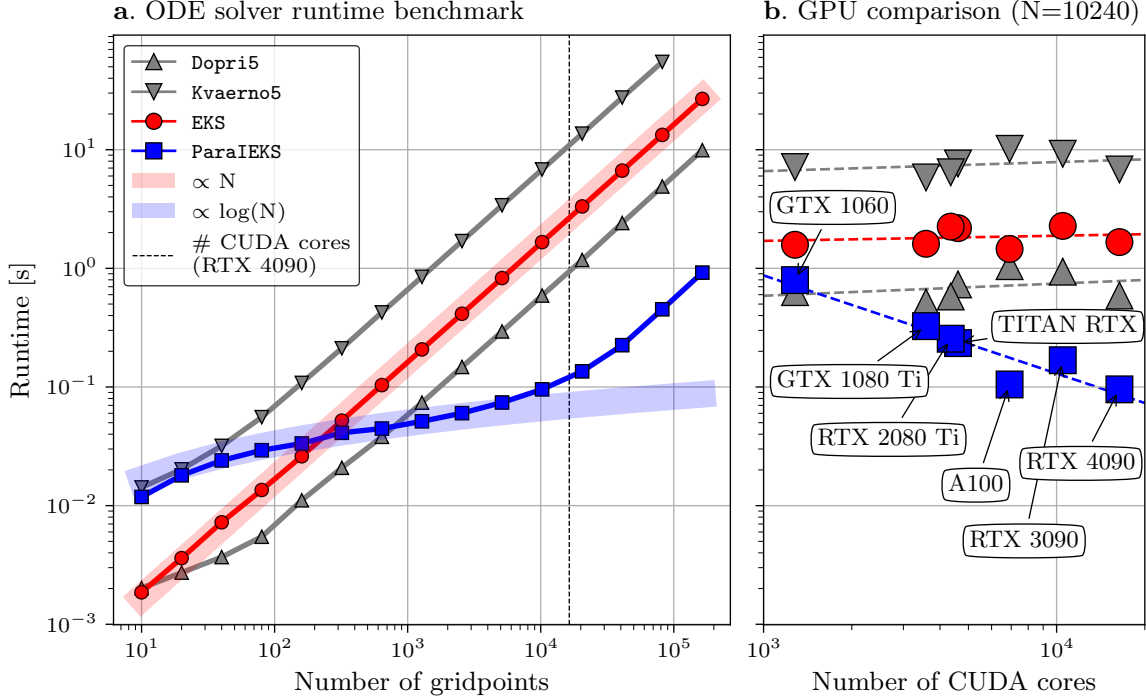


Figure 4: *The time-parallel **ParaIEKS** shows logarithmic scaling in the grid size and benefits from GPU improvements. In comparison, the sequential IEKS and the classic Dopri5 and Kvaerno5 solvers show the expected linear runtime complexity (left). The sequential methods also do not show relevant changes in runtime for GPUs with more CUDA cores, whereas **ParaIEKS**’s runtime improves (right).*

4.2 Runtimes and Parallel Scaling of the Parallel-in-Time Solver

Next we evaluate how the runtime of the proposed method scales with respect to increasing problem sizes, as well as increased numbers of available CUDA cores. We compare **ParaIEKS** to the established sequential **EKS**, as well as to two classic ODE solvers: the explicit Runge–Kutta method **Dopri5** (Dormand and Prince, 1980; Shampine, 1986), and the implicit Runge–Kutta method **Kvaerno5** (Kværnø, 2004); both provided by the **DiffraX** python package (Kidger, 2021). We apply all methods to the logistic ODE on a range of grid sizes, resulting from time discretizations with step sizes $h = 2^0, 2^{-1}, \dots, 2^{-14}$, as well as for multiple GPUs with varying numbers of CUDA cores. Figure 4 shows the results.

First, we observe the expected logarithmic scaling of **ParaIEKS** with respect to the grid size, for grids of size up to around $5 \cdot 10^3$ (Figure 4a). For larger grid sizes the runtime of **ParaIEKS** starts to grow linearly. This behavior is expected: The NVIDIA RTX 4090 used in this experiment has 16384 CUDA cores, so for larger grids the filter and smoother pass can not be fully parallelized anymore and additional grid points need to be processed sequentially. Note also that for very large grids, the 24GB memory of the GPU can become a limiting factor. Nevertheless, for large grid sizes $N > 10^3$, **ParaIEKS** achieves the lowest runtimes out of all the tested solvers, with up to an order of magnitude faster ODE solutions.

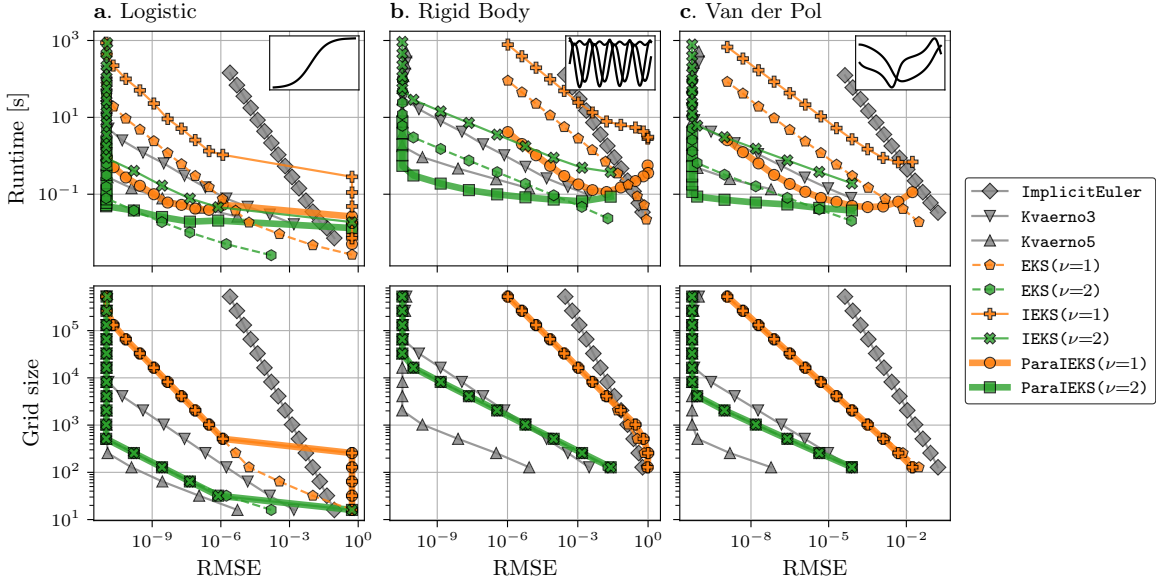


Figure 5: *Benchmarking ParaIEKS against other commonly used numerical ODE solvers.* Top row: Work-precision diagrams showing runtimes per error for a range of different ODE solvers (lower-left is better). Bottom row: Errors per specified grid size (lower-left is better). Per grid size, the closely related EKS, IEKS, and ParaIEKS solvers often coincide; Kvaerno5 achieves the lowest error per step as it has the highest order. In terms of runtime, ParaIEKS outperforms all other methods on medium-to-high accuracy settings due to its logarithmic time complexity.

Figure 4b. shows runtimes for different GPUs with varying numbers of CUDA cores for a grid of size $N = 10240$. We observe that the sequential IEKS, Dopri5 and Kvaerno5 solvers do not show a benefit from the improved GPU hardware, which is expected as the method does not leverage parallelization. On the other hand, the runtime of ParaIEKS decreases as the number of CUDA cores increases, and we observe speed-ups of up to an order of magnitude by using a newer GPU with a larger number of CUDA cores.

4.3 Benchmarking the Parallel-in-time Probabilistic Numerical ODE Solver

Finally, we compare ParaIEKS to a range of well-established implicit ODE solvers, including both classic and probabilistic numerical methods: we compare against the implicit Euler method (ImplicitEuler) and the Kvaerno3 and Kvaerno5 solvers (Kværnø, 2004) provided by DiffraX (Kidger, 2021), as well as the sequential EKS with local linearization, which is one of the currently most popular probabilistic numerical ODE solvers. We evaluate all solvers on all three test problems, namely the logistic ODE, the rigid body problem, and the Van der Pol oscillator, as introduced in Section 4.1. Reference solutions are computed with diffraX’s Kvaerno5 solver with adaptive steps and a very low error tolerance setting ($\tau_{\text{abs}} = 10^{-16}$, $\tau_{\text{rel}} = 10^{-13}$).

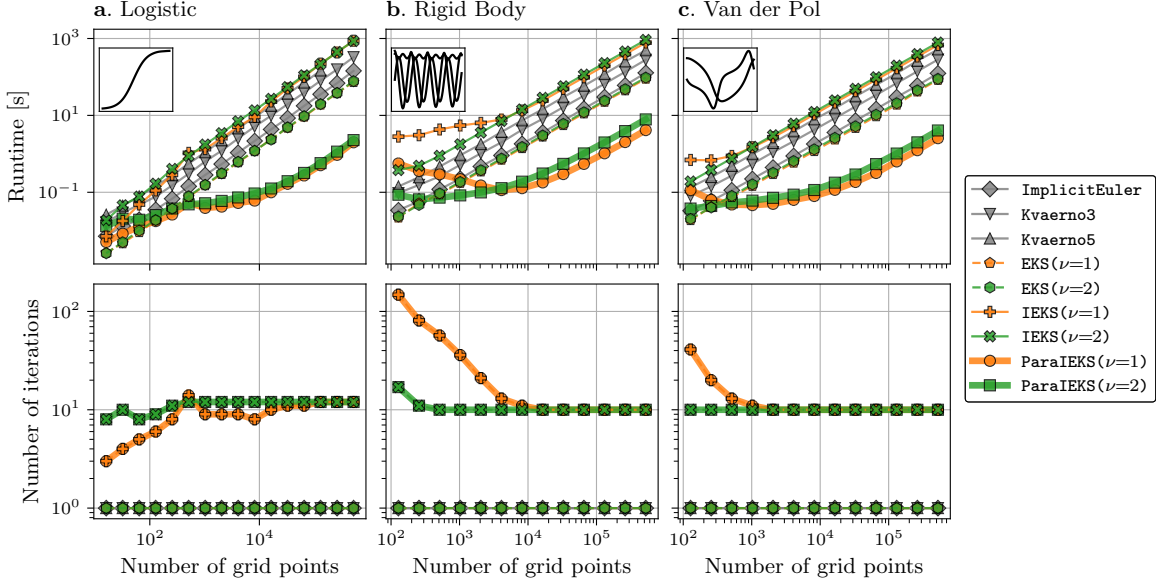


Figure 6: *Runtimes of the ODE solvers for each grid size, and number of IEKS iterations.* While all sequential solvers demonstrate linear scaling with the number of grid points, **ParaIEKS** shows sub-linear scaling up to a certain grid size (top). The number of IEKS iterations until convergence can vary with the grid size and the problem, but it seems that in many cases around 10 iterations suffice (bottom). The sequential methods solve the ODE in one sweep.

Figure 5 shows the results as work-precision diagrams. For small grid sizes (low accuracy), the logarithmic time complexity of **ParaIEKS** seems to not be very relevant and it is outperformed by the non-iterated **EKS**. In the particular case of the logistic ODE, it further seems that on coarse grids the MAP estimate computed by **ParaIEKS** (and by the sequential **IEKS**) differs significantly from the actual ODE solution, and thus the error of **ParaIEKS** on coarse grids is high (lower left figure). However, for larger grid sizes (medium-to-high accuracy), **ParaIEKS** outperforms both its sequential, non-iterated counterpart **EKS**, as well as the classic implicit methods. In particular, **ParaIEKS** with IWP(2) prior often shows runtimes lower than those of the classic **Kvaerno5** method for comparable errors, even though it has a lower order of convergence and it requires multiple iterations per solve. Figure 6 shows the runtimes per grid size of all methods, as well as the number of iterations performed by the **IEKS**. We again observe the initial logarithmic scaling of the runtime with increasing problem size, up to a threshold where the runtime starts scaling linearly, as previously shown in Section 4.2. Overall, the logarithmic time complexity of the proposed **ParaIEKS** appears to be very beneficial for high accuracy settings on GPUs and makes **ParaIEKS** a very competitive ODE solver in this comparison.

5. Conclusion

In this work, we have developed a *parallel-in-time* probabilistic numerical ODE solver. The method builds on iterated extended Kalman smoothing to compute the maximum a posteriori estimate of the probabilistic ODE solution, and by using the time-parallel formulation of the IEKS it is able to efficiently leverage modern parallel computer hardware such as GPUs to parallelize its computations. Given enough processors or cores, the proposed algorithm shares the logarithmic cost per time step of the parallel IEKS and the underlying parallel prefix-sum algorithm, as opposed to the linear time complexity of standard, sequentially-operating ODE solvers. We evaluated the performance of the proposed method in a number of experiments, and have seen that the proposed parallel-in-time solver can provide speed-ups of multiple orders of magnitude over the sequential IEKS-based solver. We also compared the proposed method to a range of well-established, both probabilistic and classical ODE solvers, and we have shown that the proposed parallel-in-time method is competitive with respect to the state-of-the-art in both accuracy and runtime.

This work opens up a number of interesting avenues for future research in the intersection of probabilistic numerics and parallel-in-time methods. Potential opportunities for improvement include the investigation of other optimization algorithms, such as Levenberg–Marquart or ADMM, or the usage of line search, all of which have been previously proposed for the sequential IEKS (Särkkä and Svensson, 2020; Gao et al., 2019). Furthermore, combining the solver with adaptive grid refinement approaches could also significantly improve its performance in practice. A different avenue would be to extend the proposed method to other related differential equation problems for which sequentially-operating probabilistic numerical methods already exist, such as higher-order ODEs, differential-algebraic equations, or boundary value problems (Bosch et al., 2022; Krämer and Hennig, 2021), or to apply the method to other types of probabilistic inference problems, such as parameter or latent-force inference in ODEs (Schmidt et al., 2021; Tronarp et al., 2022; Beck et al., 2024). Finally, the improved utilization of GPUs by our parallel-in-time method could be particularly beneficial to applications in the field of machine learning, where GPUs are often required to accelerate the computations of deep neural networks. In summary, the proposed parallel-in-time probabilistic numerical ODE solver not only advances the efficiency of probabilistic numerical ODE solvers, but also paves the way for a range of future research on parallel-in-time probabilistic numerical methods and their application across various scientific domains.

Acknowledgments

The authors gratefully acknowledge co-funding by the European Union (ERC, ANUBIS, 101123955. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them). Philipp Hennig is a member of the Machine Learning Cluster of Excellence, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC number 2064/1 – Project number 390727645; he also gratefully acknowledges the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research

and Arts of the State of Baden-Württemberg. The authors would like to thank Research Council of Finland for funding. Filip Tronarp was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Nathanael Bosch. The authors are grateful to Nicholas Krämer for many valuable discussion and to Jonathan Schmidt for feedback on the manuscript.

Individual Contributions

The original idea for this article came independently from SS and from discussions between FT and NB. The joint project was initiated and coordinated by SS and PH. The methodology was developed by NB in collaboration with AC, FT, PH, and SS. The implementation is primarily due to NB, with help from AC. The experimental evaluation was done by NB with support from FT and PH. The first version of the article was written by NB, after which all authors reviewed the manuscript.

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- U. M. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Society for Industrial and Applied Mathematics, 1995. doi: 10.1137/1.9781611971231.
- J. Beck, N. Bosch, M. Deistler, K. L. Kadhim, J. H. Macke, P. Hennig, and P. Berens. Diffusion tempering improves parameter estimation with probabilistic integrators for ordinary differential equations. *arXiv:2402.12231*, 2024.
- B. M. Bell. The iterated Kalman smoother as a Gauss–Newton method. *SIAM Journal on Optimization*, 4(3):626–636, 1994.
- J. Bettencourt, M. J. Johnson, and D. Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in JAX. In *Program Transformations for ML Workshop at NeurIPS 2019*, 2019.
- G. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, 1989. doi: 10.1109/12.42122.
- N. Bosch, P. Hennig, and F. Tronarp. Calibrated adaptive probabilistic ODE solvers. In A. Banerjee and K. Fukumizu, editors, *Proceedings of The 24th International Conference*

- on *Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3466–3474. PMLR, 2021.
- N. Bosch, F. Tronarp, and P. Hennig. Pick-and-mix information operators for probabilistic ODE solvers. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10015–10027. PMLR, 2022.
- N. Bosch, P. Hennig, and F. Tronarp. Probabilistic exponential integrators. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
- J. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2016. ISBN 9781119121503.
- R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- P. Deuffhard and F. Bornemann. *Scientific computing with ordinary differential equations*, volume 42. Springer Science & Business Media, 2012.
- J. R. Dormand and P. J. Prince. A family of embedded Runge–Kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26, 1980.
- M. J. Gander. 50 years of time parallel time integration. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113, Cham, 2015. Springer International Publishing. ISBN 978-3-319-23321-5.
- M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007. doi: 10.1137/05064607X.
- R. Gao, F. Tronarp, and S. Särkkä. Iterated extended kalman smoother-based variable splitting for l_1 -regularized state estimation. *IEEE Transactions on Signal Processing*, 67(19):5078–5092, 2019. doi: 10.1109/TSP.2019.2935868.
- A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in applied mathematics. Society for Industrial and Applied Mathematics, 2000. ISBN 9780898714517.
- E. Hairer, S. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8. Springer-Verlag, 1993. ISBN 978-3-540-56670-0. doi: 10.1007/978-3-540-78862-1.

- U. Helmke, R. Brockett, and J. Moore. *Optimization and Dynamical Systems*. Communications and Control Engineering. Springer London, 2012. ISBN 9781447134671.
- P. Hennig and S. Hauberg. Probabilistic solutions to differential equations and their application to riemannian statistics. In S. Kaski and J. Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 347–355. PMLR, 2014.
- P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings. Mathematical, physical, and engineering sciences*, 471, 2015.
- P. Hennig, M. A. Osborne, and H. P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022. doi: 10.1017/9781316681411.
- S. Julier and J. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. doi: 10.1109/JPROC.2003.823141.
- S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45(3):477–482, 2000. doi: 10.1109/9.847726.
- R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. ISSN 0021-9223. doi: 10.1115/1.3662552.
- H. Kersting and P. Hennig. Active uncertainty calibration in Bayesian ODE solvers. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 309–318, 2016.
- H. Kersting, T. J. Sullivan, and P. Hennig. Convergence rates of Gaussian ODE filters. *Statistics and computing*, 30(6):1791–1816, 2020.
- P. Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.
- O. Knuth. A globalization scheme for the generalized gauss-newton method. *Numerische Mathematik*, 56(6):591–607, Jun 1989. ISSN 0945-3245. doi: 10.1007/BF01396345.
- N. Krämer and P. Hennig. Linear-time probabilistic solution of boundary value problems. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- N. Krämer and P. Hennig. Stable implementation of probabilistic ode solvers. *Journal of Machine Learning Research*, 25(111):1–29, 2024.
- N. Krämer, N. Bosch, J. Schmidt, and P. Hennig. Probabilistic ODE solutions in millions of dimensions. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11634–11649. PMLR, 2022.

- N. Krämer, J. Schmidt, and P. Hennig. Probabilistic numerical method of lines for time-dependent partial differential equations. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 625–639. PMLR, 2022.
- A. Kværnø. Singly diagonally implicit Runge–Kutta methods with an explicit first stage. *BIT Numerical Mathematics*, 44(3):489–502, 2004.
- J.-L. Lions, Y. Maday, and G. Turinici. Résolution d’EDP par un schéma en temps “pararéel”. *Comptes Rendus de l’Académie des Sciences - Series I - Mathematics*, 332(7):661–668, 2001. ISSN 0764-4442.
- C. J. Oates and T. J. Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 29, 2019.
- G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(1), 2021. ISSN 1532-4435.
- K. Pentland, M. Tamborrino, D. Samaddar, and L. C. Appel. Stochastic parareal: An application of probabilistic methods to time-parallelization. *SIAM Journal on Scientific Computing*, 0(0):S82–S102, 2021. doi: 10.1137/21M1414231.
- K. Pentland, M. Tamborrino, T. J. Sullivan, J. Buchanan, and L. C. Appel. GParareal: a time-parallel ODE solver using Gaussian process emulation. *Statistics and Computing*, 33(1):23, 2022. ISSN 1573-1375. doi: 10.1007/s11222-022-10195-y.
- H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965. ISSN 1533-385X. doi: 10.2514/3.3166.
- M. S. Grewal and A. P. Andrews. Kalman filtering. 2014. doi: 10.1002/9781118984987.
- S. Särkkä and A. F. García-Fernández. Temporal parallelization of Bayesian smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, 2021. doi: 10.1109/TAC.2020.2976316.
- S. Särkkä and L. Svensson. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2023. ISBN 9781108912303.
- J. Schmidt, N. Krämer, and P. Hennig. A probabilistic state space model for joint inference from differential equations and data. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- M. Schober, S. Särkkä, and P. Hennig. A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, 29(1):99–122, 2019. ISSN 1573-1375. doi: 10.1007/s11222-017-9798-7.
- L. F. Shampine. Some practical Runge–Kutta formulas. *Mathematics of Computation*, 46(173):135–150, 1986. doi: <https://doi.org/10.2307/2008219>.

- J. Skilling. *Bayesian Solution of Ordinary Differential Equations*, pages 23–37. Springer, 1992. ISBN 978-94-017-2219-3. doi: 10.1007/978-94-017-2219-3_2.
- Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.
- W. Su, S. Boyd, and E. J. Candès. A differential equation for modeling Nesterov’s accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17(153): 1–43, 2016.
- S. Särkkä. Unscented Rauch–Tung–Striebel smoother. *IEEE Transactions on Automatic Control*, 53(3):845–849, 2008. doi: 10.1109/TAC.2008.919531.
- S. Särkkä and A. Solin. *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2019. doi: 10.1017/9781108186735.
- S. Särkkä and L. Svensson. Levenberg-Marquardt and line-search extended Kalman smoothers. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5875–5879, 2020. doi: 10.1109/ICASSP40776.2020.9054686.
- F. Tronarp, H. Kersting, S. Särkkä, and P. Hennig. Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: a new perspective. *Statistics and Computing*, 29(6):1297–1315, 2019.
- F. Tronarp, S. Särkkä, and P. Hennig. Bayesian ODE solvers: The maximum a posteriori estimate. *Statistics and Computing*, 31(3):1–18, 2021.
- F. Tronarp, N. Bosch, and P. Hennig. Fenrir: Physics-enhanced regression for initial value problems. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 21776–21794. PMLR, 2022.
- B. Van der Pol. Theory of the amplitude of free and forced triode vibrations. *Radio Review*, 1:701–710, 1920.
- P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.
- F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä. Parallel iterated extended and sigma-point Kalman smoothers. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5350–5354, 2021. doi: 10.1109/ICASSP39728.2021.9413364.

F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä. Parallel square-root statistical linear regression for inference in nonlinear state space models. *arXiv:2207.00426*, 2023.