

ProbNum: Probabilistic Numerics in Python

Jonathan Wenger

JONATHAN.WENGER@UNI-TUEBINGEN.DE

Nicholas Krämer

NICHOLAS.KRAEMER@UNI-TUEBINGEN.DE

Marvin Pfoertner

MARVIN.PFOERTNER@UNI-TUEBINGEN.DE

Jonathan Schmidt

JONATHAN.SCHMIDT@UNI-TUEBINGEN.DE

Nathanael Bosch

NATHANAEL.BOSCH@UNI-TUEBINGEN.DE

Nina Effenberger

NINA.EFFENBERGER@UNI-TUEBINGEN.DE

Johannes Zenn

JOHANNES.ZENN@STUDENT.UNI-TUEBINGEN.DE

Alexandra Gessner

ALEXANDRA.GESSNER@UNI-TUEBINGEN.DE

University of Tübingen, Tübingen, Germany

Toni Karvonen

TONI.KARVONEN@HELSINKI.FI

University of Helsinki, Helsinki, Finland

François-Xavier Briol

F.BRIOL@UCL.AC.UK

University College London, London, UK

Maren Mahsereci

MAREN.MAHSERECI@UNI-TUEBINGEN.DE

Philipp Hennig

PHILIPP.HENNIG@UNI-TUEBINGEN.DE

University of Tübingen, Tübingen, Germany

Abstract

Probabilistic numerical methods (PNMs) solve numerical problems via probabilistic inference. They have been developed for linear algebra, optimization, integration and differential equation simulation. PNMs naturally incorporate prior information about a problem and quantify uncertainty due to finite computational resources as well as stochastic input. In this paper, we present **ProbNum**: a Python library providing state-of-the-art probabilistic numerical solvers. **ProbNum** enables custom composition of PNMs for specific problem classes via a modular design as well as wrappers for off-the-shelf use. Tutorials, documentation, developer guides and benchmarks are available online at www.probnum.org.

Keywords: probabilistic numerics, machine learning, numerical analysis

1. Introduction

Mathematical models are used to explain and predict the behavior of complex systems in all fields of engineering and the physical sciences. In practice, their application relies heavily on numerical approximation. However, a finite computational budget or noise-corrupted inputs induce uncertainty over the quality of any computed approximation. For example, the accurate simulation of a tsunami model may take several hours [1], but operational needs require much faster approximate simulation [2]. Quantification of the induced error is crucial to any practitioner relying on the output of such a model.

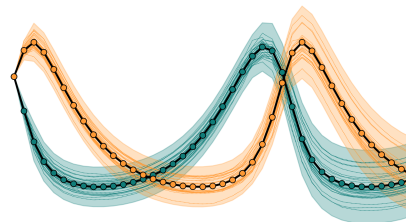
Probabilistic numerics (PN) [3–5] quantifies uncertainty in computation by solving problems from numerical analysis with statistical inference. The probabilistic viewpoint provides a principled way to encode structural knowledge about a problem into the algorithm and gives rise to new functionality beyond the scope of non-probabilistic methods [6].

```

import probnum as pn

belief = pn.linalg.problinsolve(A, b)
belief.x
# <Normal with shape=(n,), dtype=float64>

```



(a) Solving a linear system with ProbNum.

(b) Output belief of a PN method for ODEs.

Figure 1: ProbNum’s probabilistic numerical methods compute beliefs over the solution of a numerical problem. The example code in (a) shows how to solve a linear system, while (b) illustrates the uncertainty arising from approximating the solution of a differential equation.

Previously, implementations of PN algorithms have been either standalone, only available for a specific numerical problem, or not available at all. This lack of quality open-source software has inhibited the development of PN methods and their application. To promote the widespread adoption of PN in the scientific community, we introduce **ProbNum**, an open-source Python library implementing probabilistic numerical methods. Our library

- facilitates *rapid experimentation* with, and *application* of, state-of-the-art PN solvers;
- enables *custom composition of solvers* for specific problems from predefined components;
- simplifies and accelerates *development of new methods* via a unified API.

Finally, by implementing PN methods in a composable manner, **ProbNum** is an initial step towards the vision of PN of propagating uncertainty through chains of computations [3].

2. Core Functionality

ProbNum implements probabilistic numerical methods (PNMs) for solving linear systems, differential equations, and integration problems (see Figure 1 for an example). Table 1 lists all currently implemented solvers.

Table 1: Probabilistic numerical methods implemented in **ProbNum**.

Area	Problem	s.t.	QoI	Solver	Ref.
Linear Algebra	$Ax = b$	$A \in \mathbb{R}^{n \times n}$ A spd	x, A^{-1} x	Prob. linear solver BayesCG	[7, 8] [9, 10]
ODEs	$\dot{y}(t) = f(y(t), t)$	$y(t_0) = y_0$	$y(t)$	ODE filter Perturbed solver	[11–14] [15]
Integration	$F = \int_{\Omega} f(x) d\mu(x)$	$\Omega \subseteq \mathbb{R}^n$	F	Bayesian Monte Carlo Bayesian quadrature	[16] [17]

Linear Algebra PNMs for linear algebra are focused on solving linear systems. **ProbNum** implements iterative solvers, which either compute a belief over the (pseudo-)inverse of the matrix [7, 8] or the solution directly [9, 10]. While these perspectives differ conceptually, they are inherently connected [18].

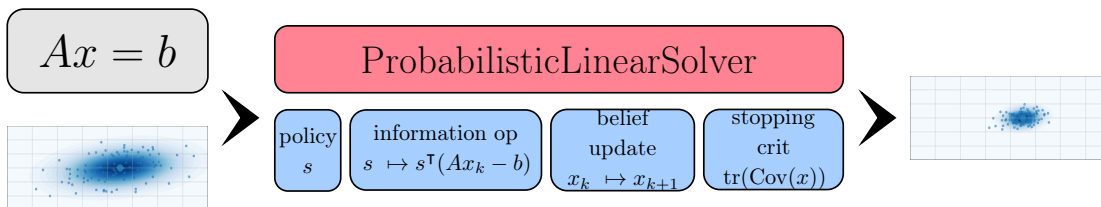


Figure 2: Probabilistic numerical methods in `ProbNum` are designed in a modular fashion. For example, a specialized linear solver (●) can be instantiated from (custom) modules (●). For a given problem $Ax = b$ and prior belief, the solver then outputs a belief about the solution.

Ordinary Differential Equations `ProbNum`'s stable implementation [11] of filtering-based ODE solvers and their variations [13, 14] enables uncertainty calibration, step-size adaptation, dense output, event handling and posterior sampling [11, 12]. `ProbNum` also provides perturbation-based ODE solvers [15].

Numerical Integration The Bayesian quadrature implementation in `ProbNum` comprises both Bayesian Monte Carlo [16] and Bayesian quadrature with fixed user-specified points [e.g. 17]. Currently, integrals can be estimated with respect to the Lebesgue and Gaussian measure.

3. Library Design

`ProbNum` mainly targets two groups of users. Those, that either want to

- (a) *use PNMs out-of-the-box* and explore their potential for their application of interest; or
- (b) *customize, develop and implement new PNMs* for specific problems.

This focus on two different user groups is realized with a corresponding design pattern which reflects `ProbNum`'s core principles: *usability*, *extensibility*, and *composability*.

Out-of-the-Box PN methods The widespread application of PN to scientific simulation problems necessitates that PN algorithms can be called without knowing the algorithmic details of each solver. Consequently, `ProbNum` provides user-friendly interface methods for standard PN algorithms. These take the numerical problem to be solved – and optionally known prior information – as input and return a belief over the quantities of interest. Interface methods often shadow the function signatures of their direct `NumPy` or `SciPy` equivalent, which makes them immediately *usable* inside more general simulation pipelines.

Plug-and-Play for Custom Problems Often, specific problems require tailored numerical methods. Therefore, `ProbNum` allows the construction of PN methods from individual components. In fact, `ProbNum`'s interface methods merely call these lower-level, customizable implementations of PN methods. Each PNM is constructed from a set of components, for example, policies, information operators, belief updates, hyperparameter optimization routines, or perturbation strategies. An illustrative example is shown in Figure 2. This compositional structure makes PN methods *extensible*: a user only needs to implement single components of PN algorithms in order to build a novel, directly usable PN implementation. Finally, since all PN algorithm components act on random variables or random processes, their implementations are naturally *composable* with each other.

Supporting Subpackages There are several `ProbNum` subpackages such as basic data structures given by random variables and processes, time- and memory-efficient matrix-free linear operators, as well as Bayesian filters and smoothers. These fundamentally enable and enhance the functionalities above, but may be of independent interest.

4. Project Development

`ProbNum` is an open-source, community-driven library hosted publicly on GitHub at



<https://github.com/probabilistic-numerics/probnum>,

which allows for the tracking of issues, bugs and feature requests, as well as code contributions via pull requests. `ProbNum` is distributed under the MIT license and available from the Python Packaging Index (PyPI) for all recent Python versions via `pip install probnum`. All contributors and community members are expected to adhere to the code of conduct.

Documentation `ProbNum`'s documentation can be found at www.probnum.org. A range of tutorials showcasing PN methods, their applications, and how to use the library are also available. A detailed development guide promotes contributions to the library.

Dependencies `ProbNum` fundamentally only depends on `NumPy` and `SciPy`, but includes the option to install additional packages to extend its functionality (such as automatic differentiation frameworks). All dependencies are kept up-to-date automatically via a bot.

Continuous Integration and Tests The project's documentation and tutorial notebooks are built for each commit to the main library and every pull request. Automatic linting preserves a high level of code quality and style. `ProbNum`'s extensive unit tests leverage `pytest`, and the test coverage is updated automatically whenever new changes occur. Finally, regular, automated benchmarks detect potential performance regressions.

5. Related Concepts and Libraries

`ProbNum`'s solvers may be used as a drop-in replacement for *classic numerical solvers* (such as those offered by `SciPy` [19]). While `ProbNum` does not implement any classical solvers, some PNMs fundamentally recover these without incurring significant computational overhead to compute uncertainty. The field of PN can be seen as a subset of the broad research area of *uncertainty quantification*, which “is the end-to-end study of the reliability of scientific inferences” [20]. However, existing software libraries for uncertainty quantification [21, 22] do not implement any PN algorithms. `ProbNum` provides (approximate) inference algorithms for numerical tasks by casting them as statistical inference problems. It is therefore not a *probabilistic programming* framework [23–25], which automate general probabilistic inference. Nevertheless, `ProbNum` may serve as a low-level building block inside probabilistic programs, e.g. to add computational uncertainty. Finally, there are special-purpose packages implementing some PNMs, e.g. for Bayesian optimization [26–28] and quadrature [27, 29].

6. Conclusion

We hope that `ProbNum` will increasingly be the home for algorithmic research in probabilistic numerics and grow to further be a practical tool for applied users who want to leverage

the functionality of probabilistic numerical methods. For developers and researchers alike, ProbNum offers the scaffold for efficient implementation of novel methods, as well as the repository of existing methods for comparisons. For applied users, ProbNum provides clean interfaces and a “one-stop-shop” for emerging new functionality.

Acknowledgments

We acknowledge contributions from Thomas Gläfle, Timothy Reid and others.¹ The authors thank the participants of the Dagstuhl seminar “Probabilistic Numerical Methods – From Theory to Implementation” for valuable discussions and feedback. JW, NK, MP, JS, NB, MM, AG, and PH gratefully acknowledge financial support by the European Research Council through ERC StG Action 757275 / PANAMA; 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) as well as project ADIMEM (FKZ: 01IS18052B); and 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 JW, NK, and NB. TK was supported by the Academy of Finland postdoctoral researcher grant 338567 “Scalable, adaptive and reliable probabilistic integration”.

References

- [1] J. Behrens and F. Dias. New computational methods in tsunami science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373 (2053), 2015.
- [2] Daniel Giles, Eugene Kashdan, Dimitra M. Salmanidou, Serge Guillas, and Frédéric Dias. Performance analysis of Volna-OP2 – massively parallel code for tsunami modelling. *Computers and Fluids*, 209, 2020.
- [3] Philipp Hennig, Mike A. Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 471(2179), 2015.
- [4] Jon Cockayne, Chris Oates, Tim J. Sullivan, and Mark Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 2019.
- [5] Chris J Oates and Tim J Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 29(6):1335–1351, 2019.
- [6] Jonathan Schmidt, Nicholas Krämer, and Philipp Hennig. A probabilistic state space model for joint inference from differential equations and data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
- [7] Philipp Hennig. Probabilistic interpretation of linear solvers. *SIAM Journal on Optimization*, 25(1):234–260, 2015.

1. Please see <https://github.com/probabilistic-numerics/probnum/graphs/contributors> for a complete and continuously updated list of contributors.

- [8] Jonathan Wenger and Philipp Hennig. Probabilistic linear solvers for machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [9] Jon Cockayne, Chris Oates, Ilse C. Ipsen, and Mark Girolami. A Bayesian conjugate gradient method. *Bayesian Analysis*, 14(3):937–1012, 2019.
- [10] Tim W. Reid, Ilse C. F. Ipsen, Jon Cockayne, and Chris J. Oates. BayesCG as an uncertainty aware version of CG. *arXiv preprint*, 2021. URL <http://arxiv.org/abs/2008.03225>.
- [11] Nicholas Krämer and Philipp Hennig. Stable implementation of probabilistic ODE solvers. *arXiv preprint*, 2020. URL <http://arxiv.org/abs/2012.10106>.
- [12] Nathanael Bosch, Philipp Hennig, and Filip Tronarp. Calibrated adaptive probabilistic ODE solvers. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.
- [13] Filip Tronarp, Hans Kersting, Simo Särkkä, and Philipp Hennig. Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: A new perspective. *Statistics and Computing*, 29(6):1297–1315, 2019.
- [14] Filip Tronarp, Simo Särkkä, and Philipp Hennig. Bayesian ODE solvers: The maximum a posteriori estimate. *Statistics and Computing*, 31(3):1–18, 2021.
- [15] Assyr Abdule and Giacomo Garegnani. Random time step probabilistic methods for uncertainty quantification in chaotic and geometric numerical integration. *Statistics and Computing*, 30: 1–26, 2020.
- [16] C. E. Rasmussen and Z. Ghahramani. Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2003.
- [17] A. O’Hagan. Bayes–Hermite quadrature. *Journal of Statistical Planning and Inference*, 29(3): 245–260, 1991.
- [18] Simon Bartels, Jon Cockayne, Ilse C. Ipsen, and Philipp Hennig. Probabilistic linear solvers: A unifying view. *Statistics and Computing*, 29(6):1249–1263, 2019.
- [19] 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.
- [20] U.S. Department of Energy. Scientific grand challenges in national security: The role of computing at the extreme scale. Technical report, 2009.
- [21] Bert Debuschere, Khachik Sargsyan, Cosmin Safta, and Kenny Chowdhary. *Uncertainty Quantification Toolkit (UQTK)*, pages 1807–1827. Springer International Publishing, 2017.
- [22] Audrey Olivier, Dimitrios Giovanis, B. S. Aakash, Mohit Chauhan, Lohit Vandanapu, and Michael D. Shields. UQpy: a general purpose Python package and development environment for uncertainty quantification. *Journal of Computational Science*, 47, 2020.
- [23] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2, 2016.

- [24] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv preprint*, 2019. URL <http://arxiv.org/abs/1912.11554>.
- [25] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research (JMLR)*, 20(1):973–978, 2019.
- [26] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL <https://github.com/fmfn/BayesianOptimization>.
- [27] Andrei Paleyes, Mark Pullin, Maren Mahsereci, Neil Lawrence, and Javier González. Emulation of physical processes with Emukit. In *Second Workshop on Machine Learning and the Physical Sciences (NeurIPS)*, 2019.
- [28] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, 2020.
- [29] S.-C. T. Choi, F. J. Hickernell, M. McCourt, and A. Sorokin. QMCPy: A quasi-Monte Carlo Python library, 2021. URL <http://arxiv.org/abs/2102.07833>.