The Journal of Open Source Software

PyNumDiff: Practical Numerical Differentiation for Noisy Data

Pavel Komarov [1¶], **Floris van Breugel** [2], **Maria Protogerou**[3], **and J. Nathan Kutz** [4]

1 Department of Electrical and Computer Engineering, University of Washington, USA **2** Department of Mechanical Engineering, University of Nevada, Reno, USA **3** Department of Applied Mathematics, University of Washington, USA **4** Autodesk Research, London, UK ¶ Corresponding author

DOI: 10.21105/joss.11172

Software

- Review
- Repository
- Archive

Editor: Prashant Jha

Reviewers:

- @alexzerg
- @sidxz

Submitted: 02 April 2026
Published: 02 November 2026

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Derivatives of measured data are vital across science and engineering, for identifying governing equations, designing controllers, and processing sensor streams alike. The textbook remedy, finite differencing, amplifies noise as $\frac{1}{\Delta t}$ and deteriorates rapidly as data grows noisier. Smoothing before differencing helps, but algorithm choice and settings substantially affect the result, and no single approach wins universally.

PyNumDiff is an open-source Python package consolidating a broad suite of numerical differentiation methods under a unified API, along with an optimization scheme to select hyperparameters even in the absence of ground truth. It implements seven algorithm families: (1) prefiltering followed by finite difference calculation, (2) iterated finite differencing, (3) polynomial fitting (Savitzky & Golay, 1964), (4) global (spectral) and local (wavelet, radial) basis function fitting, (5) total variation regularization (Chartrand, 2011), (6) Kalman smoothing (Kalman, 1960; Rauch et al., 1965), and (7) local approximation with linear models. All PyNumDiff methods return estimates of the true signal and its derivative as a matched pair, $(\hat{x}, \frac{d\hat{x}}{dt})$, and all have been mathematically refined to minimize computational complexity and make runtimes linear, log-linear, or near-linear.[1] This paper describes the package's second generation, through the v0.3 release series. A companion survey (Komarov et al., 2025, sec. 7) situates these methods in the literature, covers the theory behind each, benchmarks all methods across test signals, and guides selection for different application scenarios.

Statement of Need

Numerical differentiation has a diverse ecosystem of algorithms, each with different theoretical foundations and putative strengths. But there has been no consolidated home where methods can be easily swapped or tested head-to-head, and practitioners are left assembling solutions piecemeal from disparate packages (van Breugel et al., 2022).

PyNumDiff addresses this gap: Its unified interface lets users compare methods on the same data, exploit specialized capabilities (e.g. outlier robustness), and tune methods' hyperparameters against known ground truth or to a given signal bandlimit. Derivative estimation trades data fidelity against smoothness, so PyNumDiff uses dual-objective optimization, with a single scalar weight to steer the balance (van Breugel et al., 2020). A natural and growing application is SINDy (Brunton et al., 2016), which discovers governing equations by regressing measured derivatives, making quality of the derivative estimates a direct determinant of model accuracy.

[1] Convex-solver-based methods use CLARABEL (Goulart & Chen, 2024), an interior point method that employs heuristics which make it difficult to tightly bound analytically, although in practice it converges in 10–50 steps.

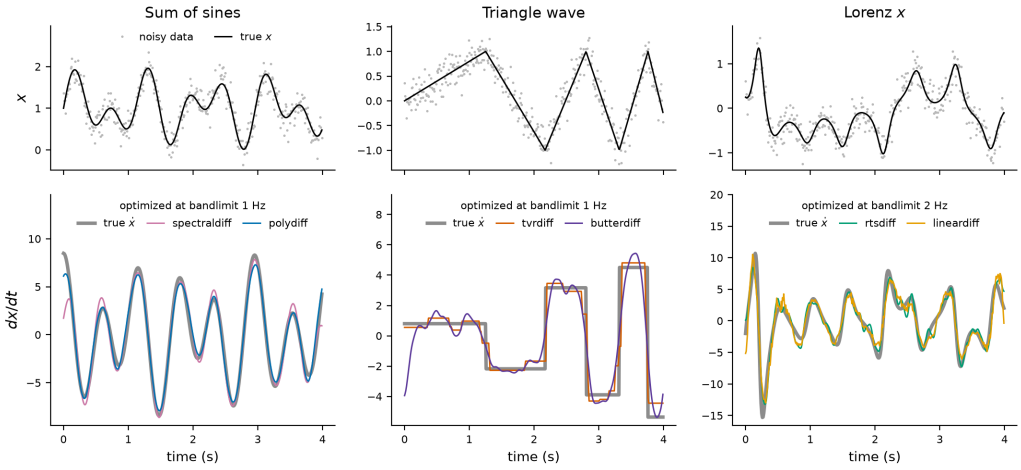


Figure 1: Three simulated signals with additive noise (top) and their derivatives (bottom). Each panel contrasts two methods from different families, six of the seven in total, with a fixed color per method. Hyperparameters come from `pynumdiff.optimize` rather than hand-tuning, at the bandlimit noted with each derivative panel: 1 Hz for the sines and triangle, 2 Hz for the faster Lorenz signal.

State of the Field

Relevant Python tools exist, but each covers only part of the space. `numpy.gradient` and `scipy.signal.savgol_filter` (Virtanen et al., 2020) are specific cases; `findiff` (Baer, 2018) offers high-order finite difference stencils suited to clean simulation data, but not to noisy measurements; `pykalman` (Duckworth & The pykalman developers, 2012) does not provide state-space models to turn the filter into a differentiator; and total variation regularization (Chartrand, 2011) was implemented with lagged-diffusivity iteration that approaches the convex minimizer rather than attaining it. Historically, practitioners have had to patch together packages, standalone scripts, and academic mathematics, with no shared API or principled way to compare results. The `derivative` package (Goldschmidt, 2021) attempts to bring more algorithms under one roof but lacks `NaN` handling and hyperparameter optimization. No existing package spans PyNumDiff's seven method families with a consistent interface.

The original PyNumDiff publication (van Breugel et al., 2022) established the core method set and optimization framework. This second generation rewrites and consolidates it from the ground up. `kerneldiff`, `rtsdiff`, and `tvrdiff` each cover by parameter what had been separate functions, so the package presents a simplified interface. `rbfdiff`, `waveletdiff`, and `robustdiff` are new, bringing the set to thirteen, all organized into seven modules and callable with consistent keyword-argument signatures. A single optimizer serves every one of them. This generation also adds support for multidimensional data, as well as irregular sample spacing, missing observations, and circular domains where the underlying mathematics allows.

Software Design

Package design. All differentiation methods share the call signature

```
x_hat, dxdt_hat = method(x, dt_or_t, **hyperparams)
```

where x is a NumPy array (Harris et al., 2020) of measurements, `dt_or_t` is either a scalar step size or an array of sample locations, and keyword arguments configure the method, making calls self-documenting. The v0.2 series preserves prior positional signatures behind deprecation warnings; v0.3 removes them, leaving only the keyword-argument interface.

Architecture. PyNumDiff is organized into seven differentiation modules plus shared `utils` and

optimize in a flat structure. Where strong alternatives exist, PyNumDiff delegates rather than reimplements: SciPy (Virtanen et al., 2020) provides spline fitting, Savitzky-Golay filtering, and signal processing routines; NumPy (Harris et al., 2020) provides the FFT; PyWavelets (Lee et al., 2019) provides the discrete wavelet transform for waveletdiff; CVXPY (Diamond & Boyd, 2016) handles convex optimization for robustdiff, tvrdiff, and lineardiff. The public kalman_filter and rts_smooth primitives let advanced users with known dynamics bypass rtsdiff's constant-derivative model to build their own bespoke differentiator. utils houses common subroutines like estimation of integration constants and data scales, calculation of evaluation metrics, and generation of simulated noisy data. optimize sets out a framework: per-method default search spaces, bounds with rounding rules, categorical handling, a caching layer, and a configurable objective.

Method capabilities. Every method handles multidimensional data via axis. Table 1 groups methods by their additional talents.

Capability	Methods
Variable step size	polydiff, splinediff, rbfdiff, rtsdiff, robustdiff, lineardiff
Missing data	polydiff, splinediff, rtsdiff, robustdiff, lineardiff
Outlier robustness	robustdiff, tvrdiff
Circular domain	rtsdiff

Table 1: Methods by auxiliary capability.

Irregular and incomplete sampling. Whether and how a method supports variable step size depends on the underlying model. For example, basis spline fits are indifferent to spacing, able to place knots with equal ease anywhere in the domain, but Kalman-based methods must compute a discrete-time transition by matrix exponential at each actual interval, while Butterworth filters inflexibly assume a constant sampling rate. Likewise, missing observations, given as NaN entries, can be excluded from fitting and imputed from the model in select cases, useful for real data where sensors may drop samples.

Outlier robustness. robustdiff replaces the quadratic Kalman cost with Huber loss (Huber, 1964) terms on both measurement and process residuals, following Aravkin et al. (2013), with CVXPY (Diamond & Boyd, 2016) as the optimization backend, operating over a sparse problem formulation so each solver iteration scales linearly with signal length. tvrdiff similarly applies a Huber loss on its data-fidelity term, while its total variation penalty on the derivative additionally promotes piecewise-smooth solutions.

Circular and wrapped domains. rtsdiff accepts circular=True for quantities like angles on a periodic domain. Innovation residuals are wrapped to $[-\pi, \pi]$ before each Kalman update via an innovation_fn hook, and x_hat is returned in the same range, avoiding the large spurious spikes naive smoothers produce when a signal crosses the $\pm\pi$ boundary.

Hyperparameter optimization. pynumdiff.optimize selects hyperparameters Φ by minimizing

$$L(\Phi) = \mathrm{RMSE}\big(\textstyle\int \hat{\dot{x}}(\Phi) + c,\ x\big) + \gamma\,\mathrm{TV}\big(\hat{\dot{x}}(\Phi)\big),$$

which requires no ground truth, because fidelity is measured by integrating the estimated derivative back against the measured signal (van Breugel et al., 2020). Root mean squared error in the first term can be (and is by default since v0.2.1) replaced with a Huber loss, so it is less prone to bias in the presence of outliers. The smoothness weight γ is calculated internally from the data's sampling rate and user-provided bandlimit, the highest frequency of meaningful signal. Default search spaces are defined at the top of the optimize module, each seeding Nelder-Mead runs from the Cartesian product of starting values. Minimization requires a continuous space, so discrete numbers like polynomial order and window size are handled by rounding, while truly categorical hyperparameters like kernel type are supported by pooling results across runs employing separate choices. The count of method evaluations

grows multiplicatively with search dimension, dominating cost far more than method runtime, so opportunities to collapse to fewer search dimensions have been thoroughly explored, and evaluations are cached to avoid redundant work across parallel searches. A single call to `suggest_method` runs optimization across nearly every method in the package and reports the best fit, a practical payoff of putting all behind one interface.

Testing and continuous integration. The test suite validates all methods against analytic functions with known derivatives, covering noiseless and noisy cases, scale equivariance, multidimensional application, and missing-data handling. Utilities and optimization are also tested for correct behavior, to verify functionality like robust estimation of data scale and the equivalence of parallel and serial search. The tests avoid tautology, never letting an implementation define its own expected result: Accuracy is asserted against empirically established error bounds, with the suite reporting any method that beats its bound by an order of magnitude so the bound can be ratcheted down. An optional `--plot` flag renders every case for visual inspection. GitHub Actions runs the suite on every push and pull request, and Coveralls tracks line coverage, currently 95%.

Research Impact Statement

The first generation (van Breugel et al., 2022) of PyNumDiff is cited in over twenty publications and has been applied in experimental biology (flight kinematics from motion capture) and control engineering (observer design). The PySINDy project (Silva et al., 2020) maintains its own differentiation submodule overlapping substantially with PyNumDiff's capabilities, and integration discussions are ongoing.

The companion taxonomy paper (Komarov et al., 2025, sec. 7) provides accuracy and bias results for every included method and derives recommendations, based on performance against simulated signals, using loss, search spaces, and other choices now encoded in PyNumDiff v0.3. The evidence is reproducible, with experiments runnable from Jupyter notebooks in PyNumDiff's repo, spanning sampling rate, noise distribution, noise scale, outlier contamination, and signal bandlimit. Additional tutorial notebooks cover basic usage, hyperparameter optimization, automatic method suggestion, and demonstrations of the outlier, multidimensional, and circular-domain capabilities described above. Full API documentation is published at pynumdiff.readthedocs.io.

AI Usage Disclosure

This paper was drafted with assistance from Claude Opus 5 (Anthropic), which also helped draft code revisions before and during review. All outputs were reviewed and heavily remolded by hand, and the authors take full responsibility for accuracy.

Acknowledgements

The authors thank Yuying Liu and Bingni W. Brunton for their contributions to the original PyNumDiff package (van Breugel et al., 2022), and Sasha Aravkin for discussions on convex optimization techniques that informed robust differentiation methods. This work was supported by the NSF AI Institute in Dynamic Systems (grant number 2112085).

References

Aravkin, A. Y., Burke, J. V., & Pillonetto, G. (2013). Optimization viewpoint on Kalman smoothing with applications to robust and sparse estimation. *Journal of Machine Learning Research*, *14*, 2513–2558. https://doi.org/10.1007/978-3-642-38398-4_8

Baer, M. (2018). *Findiff: Finite difference derivatives in Python*. https://github.com/maroba/findiff

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, *113*(15), 3932–3937. https://doi.org/10.1073/pnas.1517384113

Chartrand, R. (2011). Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, *2011*, 164564. https://doi.org/10.5402/2011/164564

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Duckworth, D., & The pykalman developers. (2012). *Pykalman: Kalman filters and smoothers for Python*. https://github.com/pykalman/pykalman

Goldschmidt, A. (2021). *Derivative: Numerical differentiation in Python*. https://github.com/andgoldschmidt/derivative

Goulart, P. J., & Chen, Y. (2024). *Clarabel: An interior-point solver for conic programs with quadratic objectives*. https://doi.org/10.48550/arXiv.2405.12762

Harris, C. R., Millman, K. J., van der Walt, S. J., & others. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, *35*(1), 73–101. https://doi.org/10.1214/aoms/1177703732

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, *82*(1), 35–45. https://doi.org/10.1115/1.3662552

Komarov, P., van Breugel, F., & Kutz, J. N. (2025). *A taxonomy of numerical differentiation methods*. https://arxiv.org/abs/2512.09090

Lee, G. R., Gommers, R., Wasilewski, F., Wohlfahrt, K., & O'Leary, A. (2019). PyWavelets: A Python package for wavelet analysis. *Journal of Open Source Software*, *4*(36), 1237. https://doi.org/10.21105/joss.01237

Rauch, H. E., Tung, F., & Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, *3*(8), 1445–1450. https://doi.org/10.2514/3.3166

Savitzky, A., & Golay, M. J. E. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, *36*(8), 1627–1639. https://doi.org/10.1021/ac60214a047

Silva, B. M. de, Champion, K., Quade, M., Loiseau, J.-C., Kutz, J. N., & Brunton, S. L. (2020). PySINDy: A Python package for the sparse identification of nonlinear dynamics from data. *Journal of Open Source Software*, *5*(49), 2104. https://doi.org/10.21105/joss.02104

van Breugel, F., Kutz, J. N., & Brunton, B. W. (2020). Numerical differentiation of noisy data: A unifying multi-objective optimization framework. *IEEE Access*, *8*, 196865–196877. https://doi.org/10.1109/ACCESS.2020.3034077

van Breugel, F., Liu, Y., Brunton, B. W., & Kutz, J. N. (2022). PyNumDiff: A Python package for numerical differentiation of noisy time-series data. *Journal of Open Source Software*, *7*(71), 4078. https://doi.org/10.21105/joss.04078

Virtanen, P., Gommers, R., Oliphant, T. E., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2