The Journal of Open Source Software

Peroxide: A Batteries-Included Numerical Computing Library for Rust

Tae-Geun Kim [1,2¶], **Giorgio Comitini** [3], **Jonas Grage** [4], **Benjamin Joens** [4], **Marc Schreiber** [4], **Soumya Sen** [4], **Russell R P Senthamarai** [4], **and Johanna Sörngård** [4]

1 Key Laboratory of Nuclear Physics and Ion-beam Application (MOE), Institute of Modern Physics, Fudan University, Shanghai 200433, China **2** RIKEN Center for Interdisciplinary Theoretical and Mathematical Sciences (iTHEMS), Wako, Saitama 351-0198, Japan **3** Università degli Studi di Catania, Catania, Italy **4** Independent Researcher ¶ Corresponding author

DOI: 10.21105/joss.10366

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @ariostas
- @jonaspleyer

Submitted: 21 March 2026
Published: 10 August 2026

Summary

Peroxide is a Rust numerical computing library that integrates eight domains in one crate: linear algebra, ordinary differential equation (ODE) solving, adaptive quadrature, spline interpolation with exact polynomial calculus, root-finding, forward-mode automatic differentiation (AD), statistics, and multi-format DataFrame I/O. We designed Peroxide for applied scientists, engineers, and students who need a cohesive numerical environment in Rust without assembling a multi-crate dependency graph. A dual-module API offers `prelude` defaults and a `fuga` module for explicit algorithm selection, addressing Rust's lack of default function arguments. Peroxide is released under the MIT/Apache-2.0 dual license and archived on Zenodo (Kim et al., 2026).

Statement of need

Scientific computing workflows rarely confine themselves to a single numerical domain. A typical pipeline may solve an ODE, interpolate the solution onto an irregular grid, compute a definite integral, find a root of a derived quantity, and serialize the results for statistical analysis. In Python, SciPy (Virtanen et al., 2020) and NumPy (Harris et al., 2020) consolidate these capabilities under one namespace. In Rust, they are scattered across specialized crates, each with its own type conventions and trait hierarchies.

Table 1 surveys the Rust numerical ecosystem. Specialist crates excel individually: diffsol (Robinson & Allmont, 2026) provides ODE/DAE solvers with sensitivity analysis; faer (Quinones, 2026) offers pure-Rust linear algebra competitive with LAPACK (Anderson et al., 1999); nalgebra (Crozet, 2025) provides matrix abstractions widely adopted across the ecosystem; argmin (Kroboth, 2025) supplies over 20 optimization algorithms; Polars (Vink, 2026) dominates tabular data processing. russell (Pedroso, 2025) covers the broadest range among alternatives (ODE solvers, quadrature, root-finding, and statistics) but lacks AD, spline calculus, and DataFrame support. No single crate spans all eight domains shown in Table 1. Peroxide fills this integration gap.

Table 1: Feature coverage of Rust numerical libraries. Each entry reflects the crate's own API, including optional feature flags; a dash indicates the feature is absent. Crates providing only N-dimensional array abstractions without numerical algorithms (e.g., ndarray (rust-ndarray developers, 2026)) are omitted.

	Peroxide	diffsol	russell	faer	nalgebra	argmin	num-dual	Polars
Linear algebra	✓	—	✓	✓	✓	—	—	—
ODE solvers	✓	✓	✓	—	—	—	—	—
Quadrature	✓	—	✓	—	—	—	—	—
Spline interp.	✓	—	—	—	—	—	—	—
Root-finding	✓	—	✓	—	—	✓	—	—
Automatic diff.	✓	✓	—	—	—	—	✓	—
Statistics	✓	—	✓	—	—	—	—	—
DataFrame I/O	✓	—	—	—	—	—	—	✓

We do not claim superiority over any specialist crate within its domain. Peroxide's contribution is the integration itself, enabled by design abstractions: a `ButcherTableau` trait unifies Runge-Kutta methods via compile-time constants, a `Calculus` trait supports exact differentiation and integration of piecewise polynomial splines, const generic typing constrains root-finding dimensions, and a `Real` trait carries AD-derived derivatives into optimization and root-finding without glue code.

State of the field

In automatic differentiation specifically, a review of available Rust AD crates on crates.io indicates that, to our knowledge, Peroxide is the only library combining const generic derivative order, normalized Taylor coefficient storage, and true Taylor-mode propagation with $O(N^2)$ cost per elementary operation. Most alternatives provide dual numbers up to second or third order with fixed type hierarchies (Rehner & Bauer, 2021) or nested generics with exponential cost at higher orders (Larionov, 2023). The ad-trait crate (Liang et al., 2025) offers both forward and reverse modes but is limited to first-order derivatives. Enzyme (Moses & Churavy, 2020) performs AD as an LLVM compiler pass and supports both forward and reverse modes. The two libraries are complementary: Peroxide provides high-order forward derivatives, while Enzyme adds reverse-mode AD.

For ODE solving, diffsol (Robinson & Allmont, 2026) is the most feature-rich Rust crate, providing BDF and SDIRK methods with sensitivity analysis. Peroxide's `ButcherTableau` trait-based architecture is a complementary approach: it encodes Runge-Kutta methods as compile-time constants, making it straightforward for users to add new methods by defining only coefficient arrays.

Software design

Architecture. We store matrices as a flat `Vec<f64>` with a `Shape` enum (`Row`/`Col`) that controls logical layout without copying data. This design trades the rich type-level dimensionality of nalgebra for a memory model that maps directly to both the pure-Rust `matrixmultiply` crate (Sverdrup, 2025) and, when the `O3` feature flag is enabled, to OpenBLAS (OpenBLAS developers, 2026), passing raw pointers with stride and transpose flags without intermediate type conversions. The trade-offs are that matrix dimensions are not enforced at compile time (unlike nalgebra's typed `Matrix<f64, R, C>`), and the layout does not extend to N-dimensional tensors as `ndarray` (rust-ndarray developers, 2026) does. A `Real` trait abstracts over `f64` and `AD` (= `Jet<2>`), so the same function can compute both values and derivatives.

Automatic differentiation. The const generic type `Jet<N>` stores the function value $c_0 = f(a)$ and N normalized Taylor coefficients $c_k = f^{(k)}(a)/k!$ (Griewank & Walther, 2008). Multiplication follows the Cauchy product of truncated power series:

$$c_n(f \cdot g) = \sum_{i=0}^{n} c_i(f)\, c_{n-i}(g)$$

Because the coefficients are pre-divided by $k!$, no binomial coefficients appear, yielding $O(N^2)$ Taylor-mode propagation that avoids factorial overflow at higher orders. Division and transcendental functions use analogous recurrence relations on normalized coefficients. The `#[ad_function]` procedural macro (from the companion `peroxide-ad` crate) transforms a plain `fn(f64) -> f64` into versions accepting `Jet<N>`, automatically generating first- and second-derivative functions.

ODE solvers. A `ButcherTableau` trait (Butcher, 2016) declares Runge-Kutta stage coefficients as compile-time constant arrays. A blanket implementation `impl<BU: ButcherTableau> ODEIntegrator for BU` provides the stepping logic, so adding a new method requires only four constant slices. We ship 10 integrators: four fixed-step explicit methods (RALS3, RK4, RALS4, RK5), five embedded adaptive pairs (BS23 (Bogacki & Shampine, 1989), RKF45 (Fehlberg, 1969), DP45 (Dormand & Prince, 1980), TSIT45 (Tsitouras, 2011), RKF78 (Fehlberg, 1969)), and one implicit Gauss-Legendre method (GL4).

Spline calculus. Cubic, cubic Hermite (Akima, 1970), and B-spline (Knott, 2000) interpolation are provided. The `PolynomialSpline` trait exposes `polynomial_at(x)`, returning the `Polynomial` governing each interval. The `Calculus` trait then operates symbolically: `derivative()` returns a new spline of analytic derivatives, and `integrate((a, b))` evaluates the exact definite integral by antidifferentiation, not finite differencing. To our knowledge, no other Rust spline crate (Sabadie, 2025; Usher, 2022) exposes piecewise polynomial objects for exact calculus.

Root-finding and quadrature. `RootFindingProblem<const I, const O, T>` enforces input/output dimensions at compile time; the Jacobian `[[f64; I]; O]` is stack-allocated. Five solvers are provided, including Broyden's method for multivariate systems. Gauss-Kronrod adaptive quadrature (Laurie, 1997; Piessens et al., 1983) is available in six orders (G7K15 through G30K61), each with a relative-error variant.

Statistics and I/O. Distributions (uniform, normal, gamma, beta, Student's t, and others) share a common trait with `sample`, `pdf`, and `cdf` methods. The `DataFrame` structure unifies CSV, NetCDF (NSF Unidata, 2026), and Parquet (Apache Software Foundation, 2026) serialization.

Example

The progressive disclosure API allows the same integration task to be expressed at two levels of control:

```
// With prelude: sensible default (Gauss-Kronrod G7K15R, tol = 1e-4)
use peroxide::prelude::*;
let area = integrate(|x| x.sin(), (0.0, std::f64::consts::PI));

// With fuga: explicit algorithm selection
use peroxide::fuga::*;
use std::f64::consts::PI;
let area = integrate(|x| x.sin(), (0.0, PI), GaussLegendre(15));
```

Research impact statement

Peroxide has been on crates.io since 2018, with over 1,250,000 downloads and more than 700 GitHub stars. Independent researchers have adopted Peroxide in peer-reviewed work: Comitini & Siringo (2025) used its Gauss-Kronrod quadrature for QCD screened massive expansion calculations, and Steuteville & Baker (2024) identified Peroxide as one of the three most widely used Rust ODE solver packages in a survey conducted at the U.S. National Renewable Energy Laboratory. The author's own work has used Peroxide's implicit Gauss-Legendre ODE solver for symplectic reference solutions (Kim & Park, 2024), for dataset generation (Kim, 2024), and for primordial black hole axion spectra via Gauss-Kronrod quadrature and cubic Hermite splines (Jho et al., 2026). The Zenodo archive (Kim et al., 2026) ensures long-term citability and reproducibility.

Validation

We maintain 22 integration test modules with 225 tests and 255 documentation tests; the AD module alone has 115 unit tests verifying `Jet<N>` at orders $N = 1$ through 10 against symbolic reference values. `Jet<N>` achieves machine-epsilon relative errors ($\sim 10^{-15}$) across all tested orders, while central finite differences degrade to $O(1)$ by order four due to the truncation/cancellation trade-off (Press et al., 2007). ODE integrators are tested against analytic solutions, with continuous integration on every push via GitHub Actions. The repository provides 42 standalone examples; API documentation (with KaTeX-rendered mathematics) is published on docs.rs/peroxide. Community guidelines are available in `CONTRIBUTING.md`.

Limitations

We do not claim best-in-class performance in any single domain: diffsol provides richer DAE support, faer delivers faster dense factorizations, and argmin offers a broader optimization catalog. The `Real` trait currently covers only `f64` and `Jet<2>`; extending it to arbitrary `Jet<N>` and generalizing the `ODEProblem` interface to generic types are planned. We support forward-mode AD only; reverse-mode differentiation and interoperability with Rust's experimental `std::autodiff` are longer-term goals. Sparse matrices, PDE solvers, and FFT lie outside the current scope.

AI usage disclosure

A generative AI tool (Claude Opus 4, Anthropic, March 2026) was used by the lead author (T.-G. Kim) for grammar checking and copy-editing of the manuscript text. All scientific content, software design decisions, technical claims, and code were produced entirely by the authors. The final manuscript was reviewed and approved by all authors.

Acknowledgements

We thank Victor Olowofeso for early-user feedback, all past and present Peroxide contributors, and the broader Rust scientific computing community for their code, issue reports, and discussions.

References

Akima, H. (1970). A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM*, *17*(4), 589–602. https://doi.org/10.1145/321607.321609

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK users' guide* (Third). Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898719604

Apache Software Foundation. (2026). *Apache Parquet*. https://parquet.apache.org/

Bogacki, P., & Shampine, L. F. (1989). A 3(2) pair of Runge-Kutta formulas. *Applied Mathematics Letters*, *2*(4), 321–325. https://doi.org/10.1016/0893-9659(89)90079-7

Butcher, J. C. (2016). *Numerical methods for ordinary differential equations* (3rd ed.). John Wiley & Sons. https://doi.org/10.1002/9781119121534

Comitini, G., & Siringo, F. (2025). Screened massive expansion of QCD at finite temperature: Two-loop approximation. *Physical Review D*, *111*, 054012. https://doi.org/10.1103/PhysRevD.111.054012

Crozet, S. (2025). *Nalgebra: General-purpose linear algebra library for Rust* (Version 0.34.1). https://github.com/dimforge/nalgebra

Dormand, J. R., & Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, *6*(1), 19–26. https://doi.org/10.1016/0771-050X(80)90013-3

Fehlberg, E. (1969). *Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems* (TR R-315). NASA.

Griewank, A., & Walther, A. (2008). *Evaluating derivatives: Principles and techniques of algorithmic differentiation* (2nd ed.). SIAM. https://doi.org/10.1137/1.9780898717761

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

Jho, Y., Kim, T.-G., Park, J.-C., Park, S. C., & Park, Y. (2026). Primordial black holes as a factory of axions: Extragalactic photons from axions. *Progress of Theoretical and Experimental Physics*. https://doi.org/10.1093/ptep/ptag011

Kim, T.-G. (2024). *HyperbolicLR: Epoch insensitive learning rate scheduler*. https://arxiv.org/abs/2407.15200

Kim, T.-G., Comitini, G., Grage, J., Joens, B., Schreiber, M., Sen, S., Senthamarai, R. R. P., & Sörngård, J. (2026). *Peroxide: A batteries-included numerical computing library for Rust* (Version 0.43.1). Zenodo. https://doi.org/10.5281/zenodo.21754256

Kim, T.-G., & Park, S. C. (2024). *Neural Hamilton: Can A.I. Understand Hamiltonian mechanics?* https://arxiv.org/abs/2410.20951

Knott, G. D. (2000). *Interpolating cubic splines*. Birkhäuser Boston. https://doi.org/10.1007/978-1-4612-1320-8

Kroboth, S. (2025). *Argmin: A pure Rust optimization framework* (Version 0.11.0). https://github.com/argmin-rs/argmin

Larionov, E. (2023). *Autodiff: Automatic differentiation via operator overloading for Rust*. https://github.com/elrnv/autodiff

Laurie, D. P. (1997). Calculation of Gauss-Kronrod quadrature rules. *Mathematics of Computation*, *66*(219), 1133–1145. https://doi.org/10.1090/S0025-5718-97-00861-2

Liang, C., Wang, Q., Xu, A., & Rakita, D. (2025). Ad-trait: A fast and flexible automatic differentiation library in Rust. *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://doi.org/10.1109/iros60139.2025.11245908

Moses, W. S., & Churavy, V. (2020). Instead of rewriting foreign code for machine learning,

automatically synthesize fast gradients. *Advances in Neural Information Processing Systems*, *33*, 12472–12485. https://arxiv.org/abs/2010.01709

NSF Unidata. (2026). *NetCDF: Network common data form [software]*. UCAR/NSF Unidata Program Center. https://doi.org/10.5065/D6H70CW6

OpenBLAS developers. (2026). *OpenBLAS: An optimized BLAS library* (Version 0.3.33). https://github.com/OpenMathLib/OpenBLAS

Pedroso, D. M. (2025). *russell: Rust scientific library* (Version 1.10.0). https://github.com/cpmech/russell

Piessens, R., Doncker-Kapenga, E. de, Überhuber, C. W., & Kahaner, D. K. (1983). *QUADPACK: A subroutine package for automatic integration*. Springer-Verlag. https://doi.org/10.1007/978-3-642-61786-7

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes: The art of scientific computing* (3rd ed.). Cambridge University Press. ISBN: 978-0-521-88068-8

Quinones, S. (2026). *Faer: Low-level linear algebra routines in pure Rust* (Version 0.24.0). https://github.com/sarah-quinones/faer-rs

Rehner, P., & Bauer, G. (2021). Application of generalized (hyper-) dual numbers in equation of state modeling. *Frontiers in Chemical Engineering*, *3*. https://doi.org/10.3389/fceng.2021.758090

Robinson, M., & Allmont, A. (2026). Diffsol: A Rust crate for solving differential equations. *Journal of Open Source Software*, *11*(117). https://doi.org/10.21105/joss.09384

rust-ndarray developers. (2026). *ndarray: An n-dimensional array for general elements and for numerics in Rust* (Version 0.17.2). https://github.com/rust-ndarray/ndarray

Sabadie, D. (2025). *Splines: Spline interpolation library for Rust* (Version 5.0.0). https://sr.ht/~hadronized/splines/

Steuteville, R., & Baker, C. (2024). Implementing ordinary differential equation solvers in Rust programming language for modeling vehicle powertrain systems. *SAE Technical Paper 2024-01-2148*. https://doi.org/10.4271/2024-01-2148

Sverdrup, U. (2025). *Matrixmultiply: General matrix multiplication of f32 and f64 in Rust* (Version 0.3.10). https://github.com/bluss/matrixmultiply

Tsitouras, Ch. (2011). Runge-Kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, *62*(2), 770–775. https://doi.org/10.1016/j.camwa.2011.06.002

Usher, W. (2022). *Bspline: B-spline interpolation for Rust*. https://github.com/Twinklebear/bspline

Vink, R. (2026). *Polars: Blazingly fast DataFrames in Rust, Python, Node.js, R and SQL* (Version 0.53.0). https://github.com/pola-rs/polars

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