

diffsol: Rust crate for solving differential equations

Martin Robinson   and Alex Allmont                    <img alt="ORCID

ODE solvers written in lower-level languages like C, Fortran, or Rust offer significant performance benefits. However, these solvers are often more difficult to wrap and use in higher-level languages like Python or MATLAB, primarily because users must supply their equations in the language of the solver. `diffsol` solves this issue by providing its own custom DiffSL DSL which is JIT compiled to efficient native code at run-time, meaning that users from a higher-level language like Python or R can specify their equations using a simple string-based format while still maintaining performance similar to that of pure Rust. Two other popular ODE solvers that take advantage of JIT compilation are `DifferentialEquations.jl` (Rackauckas & Nie, 2017) in Julia and `diffra` (Kidger, 2021) in Python. However, both these packages compile the entire solver as well as the equations, which is a significant amount of code. `diffSol` only compiles the equations, meaning that it has a significantly smaller “time-to-first-plot” for users. Another popular differential equations solver package utilising a DSL is `OpenModelica` (Fritzson et al., 2020). Wrappers to this package in higher-level languages like Python rely on messaging to a separate OpenModelica server, which can be slow and more complicated to set up. In contrast, `diffsol` can be integrated directly into higher-level languages using language bindings and linking to a single shared library, see for example, the `pydiffsol` Python bindings discussed below.

Features

The following solvers are available in `diffsol`:

1. A variable order Backwards Difference Formulae (BDF) solver, suitable for stiff problems and singular mass matrices. The basic algorithm is derived in (Byrne & Hindmarsh, 1975), however this particular implementation follows that implemented in the MATLAB routine `ode15s` (Shampine & Reichelt, 1997) and the SciPy implementation (Virtanen et al., 2020), which features the NDF formulas for improved stability.
2. A Singly Diagonally Implicit Runge-Kutta (SDIRK or ESDIRK) solver, suitable for moderately stiff problems and singular mass matrices. Two different butcher tableau are provided, TR-BDF2 (Bank et al., 1985; Hosea & Shampine, 1996) and ESDIRK34 (Jørgensen et al., 2018), or users can supply their own.
3. A variable order Explicit Runge-Kutta (ERK) solver, suitable for non-stiff problems. One butcher tableau is provided, the 4th order TSIT45 (Tsitouras, 2011), or users can supply their own.

All solvers feature:

- Linear algebra containers and linear solvers from the `nalgebra` or `faer` crates, including both dense and sparse matrix support.
- Adaptive step-size control to given relative and absolute tolerances. Tolerances can be set separately for the main equations, quadrature of the output function, and sensitivity analysis.
- Dense output, interpolating to times provided by the user.
- Event handling, stopping when a given condition $g_e(t, y, p)$ is met or at a specific time.
- Numerical quadrature of an optional output $g_o(t, y, p)$ function over time.
- Forward sensitivity analysis, calculating the gradient of an output function or the solver states y with respect to the parameters p .
- Adjoint sensitivity analysis, calculating the gradient of cost function $G(p)$ with respect to the parameters p . The cost function can be the integral of a continuous output function $g(t, y, p)$ or a sum of a set of discrete functions $g_i(t_i, y, p)$ at time points t_i .

Bindings in higher-level languages

`pydiffsol` provides Python bindings to `diffsol` using the PyO3 (PyO3 Project and Contributors, 2025) crate. It allows users to define ODEs in Python using the DiffSL DSL, and solve them

using the Rust `diffsol` library. `pydiffsol` aims to provide a simple and easy-to-use interface for solving ODEs in Python, while still maintaining the performance benefits of using Rust under the hood.

The goal is to develop further [bindings to other higher-level languages](#), including R, JAX, and JavaScript, exploiting the `DiffSL` DSL to provide high performance while maintaining ease of use and positioning the core `diffsol` library as a widely used cross-language, cross-platform, high-performance ODE solver.

Acknowledgements

We gratefully acknowledge the support of all the past and future contributors to the `diffsol` project, for their advice, enthusiasm, bug reports and code. In particular, we would like to thank the authors of the `pharmsol` crate, Julian Otalvaro and Markus Hovd.

References

Bank, R. E., Coughran, W. M., Fichtner, W., Grosse, E. H., Rose, D. J., & Smith, R. K. (1985). Transient simulation of silicon devices and circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(4), 436–451. <https://doi.org/10.1109/T-ED.1985.22232>

Byrne, G. D., & Hindmarsh, A. C. (1975). A polyalgorithm for the numerical solution of ordinary differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 1(1), 71–96. <https://doi.org/10.1145/355626.355636>

Bytecode Alliance. (2025). *Cranelift* (Version 0.115.1). <https://wasmtime.dev>

Dimforge. (2025). *Nalgebra* (Version 0.33.2). <https://github.com/dimforge/nalgebra>

Fritzson, P., Pop, A., Abdelhak, K., Ashgar, A., Bachmann, B., Braun, W., Bouskela, D., Braun, R., Buffoni, L., Casella, F., Castro, R., Franke, R., Fritzson, D., Gebremedhin, M., Heuermann, A., Lie, B., Mengist, A., Mikelsons, L., Moudgalya, K., ... Östlund, P. (2020). The OpenModelica integrated environment for modeling, simulation, and model-based development. *Modeling, Identification and Control*, 41(4), 241–295. <https://doi.org/10.4173/mic.2020.4.1>

Gardner, D. J., Reynolds, D. R., Woodward, C. S., & Balos, C. J. (2022). Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 48(3), 1–24. <https://doi.org/10.1145/3539801>

Hosea, M., & Shampine, L. (1996). Analysis and implementation of TR-BDF2. *Applied Numerical Mathematics*, 20(1-2), 21–37. [https://doi.org/10.1016/0168-9274\(95\)00115-8](https://doi.org/10.1016/0168-9274(95)00115-8)

Jørgensen, J. B., Kristensen, M. R., & Thomsen, P. G. (2018). A family of ESDIRK integration methods. *arXiv Preprint arXiv:1803.01613*. <https://doi.org/10.48550/arXiv.1803.01613>

Kazdadi, S. E. (2025). *Faer-rs* (Version 0.1.0). <https://faer-rs.github.io>

Kidger, P. (2021). *On neural differential equations* [PhD thesis, University of Oxford]. <https://doi.org/10.48550/arXiv.2202.02435>

Lattner, C., & Adve, V. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 75–86. <https://doi.org/10.1109/CGO.2004.1281665>

Moses, W. S., Narayanan, S. H. K., Paehler, L., Churavy, V., Schanen, M., Hückelheim, J., Doerfert, J., & Hovland, P. (2022). Scalable automatic differentiation of multiple

parallel paradigms through compiler augmentation. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. <https://doi.org/10.1109/SC41404.2022.00065>

PyO3 Project and Contributors. (2025). PyO3 (Version v0.27.2). <https://github.com/PyO3/pyo3>

Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. *The Journal of Open Research Software*, 5(1). <https://doi.org/10.5334/jors.151>

Shampine, L. F., & Reichelt, M. W. (1997). The MATLAB ODE suite. *SIAM Journal on Scientific Computing*, 18(1), 1–22. <https://doi.org/10.1137/s1064827594276424>

Tsitouras, C. (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>

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>