The Journal of Open Source Software

# NodePy: A package for the analysis of numerical ODE solvers

**David I. Ketcheson*[1], Hendrik Ranocha[1], Matteo Parsani[1], Umair bin Waheed[2], and Yiannis Hadjimichael[3]**

**1** King Abdullah University of Science and Technology **2** King Fahd University of Petroleum & Minerals **3** Eötvös Loránd Tudományegyetem

**DOI:** 10.21105/joss.02515

**Software**

- Review
- Repository
- Archive

**Editor:** Viviane Pons

**Reviewers:**

- @fruzsinaagocs
- @highlando

**Submitted:** 13 July 2020
**Published:** 17 November 2020



## Summary

Ordinary differential equations (ODEs) are used to model a vast range of physical and other phenomena. They also arise in the discretization of partial differential equations. In most cases, solutions of differential equations must be approximated by numerical methods. The study of the properties of numerical methods for ODEs comprises an important and large body of knowledge. NodePy (available from https://github.com/ketch/nodepy, with documentation at https://nodepy.readthedocs.io/en/latest/) is a software package for designing and studying the properties of numerical ODE solvers. For the most important classes of methods, NodePy can automatically assess their stability, accuracy, and many other properties. NodePy has also been used as a catalog of coefficients for time integration methods in PDE solver codes.

## Statement of need

There are many software packages that *implement* ODE solvers with the purpose of efficiently providing numerical solutions; in contrast, the purpose of NodePy is to facilitate understanding of the properties of the solver algorithms themselves. In this sense, it is a sort of meta-software, consisting of algorithms whose purpose is to compute properties of other algorithms. It also serves as a reference, providing precise definitions of many of the algorithms themselves.

NodePy is written entirely in Python and provides software implementations of many of the theoretical ideas contained for instance in reference texts on numerical analysis of ODEs (Hairer et al., 1993; Hairer & Wanner, 1996). It also contains implementations of many theoretical ideas from the numerical analysis literature. The implementation focuses on the two most important classes of methods; namely, Runge-Kutta and linear multistep methods, but includes some more exotic classes. NodePy provides a means for numerical analysts to quickly and easily determine the properties of existing methods or of new methods they may develop.

NodePy development has been motivated largely by research needs and it has been used in a number of papers (including some written by non-developers; e.g. Jin (2019) and Horváth (2019)) and also as a teaching tool for graduate-level numerical analysis courses. It relies on both SymPy (Meurer et al., 2017) and NumPy (Oliphant, 2006; Walt et al., 2011) in order to provide either exact or floating-point results based on the nature of the inputs provided. It makes use of Matplotlib (Hunter, 2007) for all graphical output.

*Corresponding author.

## Features

NodePy includes object-oriented representations of the following classes of numerical methods:

- Runge-Kutta methods
  - Explicit and implicit
  - Embedded pairs
  - Classes of low-storage methods
  - Dense output formulas
  - Perturbed/additive and downwind methods
- Linear multistep methods
- Two-step Runge-Kutta methods
- Additive (IMEX) linear multistep methods

The framework is designed to include general linear methods and even more exotic classes. Any method within these classes can be generated simply by entering its coefficients. Coefficients for many methods are catalogued or can be automatically generated within NodePy, including:

- Dozens of specific Runge-Kutta methods and pairs
- General extrapolation methods, of any order of accuracy, based on a variety of building-block schemes and optionally including an error estimator
- Deferred correction methods
- Optimal strong stability preserving (SSP) Runge-Kutta methods
- Adams-Bashforth, Adams-Moulton, and BDF methods of any order
- A number of other specialized families of methods

For all of these numerical schemes, NodePy provides methods and functions to compute many of their properties – too many to list here. The theory on which most of these properties are based is outlined in standard references (Hairer et al., 1993; Hairer & Wanner, 1996). Many other properties are based on recent research; usually the method docstring includes a reference to the relevant paper. Implementations of the methods themselves are also included as a convenience, though they are not the primary purpose and are not expected to be efficient since they are coded in pure Python. Additional intermediate objects, such as the absolute stability function of a method, are given their own software representation and corresponding methods.

Additional features are provided to facilitate the analysis and testing of these numerical methods. This includes a range of initial value problems for testing, such as the stiff and non-stiff DETEST suites, and a few simple PDE semi-discretizations. Also included is a library for dealing with rooted trees, which are a class of graphs that play a key role in the theory of Runge-Kutta methods.

NodePy is documented primarily through the Python docstrings for each method, most of which contain examples that also serve as tests ("doctests"). These tests are executed automatically for all new commits and pull requests using the Travis continuous integration service. Some higher-level documentation is also available at https://nodepy.readthedocs.io/en/latest/, but it is not intended to be comprehensive.

## Related research and software

We are not aware of other software packages with a similar purpose. NodePy development has proceeded in close connection to the RK-Opt package (David I. Ketcheson et al., 2020)

(https://github.com/ketch/RK-Opt). Whereas NodePy is focused in the analysis of numerical methods, RK-Opt is focused more on their design through the use of numerical optimization to search for optimal coefficients tailored to specific desired properties. A common workflow involves generating new methods with RK-Opt and then studying their properties in more detail using NodePy.

Some of the research projects that have made use of NodePy (most of which have led to its further development) include development of:

- Strong stability preserving (SSP) Runge-Kutta methods (Hadjimichael et al., 2013; David I. Ketcheson et al., 2009; David I. Ketcheson, 2008)
- SSP general linear methods (Bresten et al., 2017; David I. Ketcheson et al., 2011)
- Low-storage Runge-Kutta methods (David I. Ketcheson, 2010)
- Additive and downwind SSP Runge-Kutta methods (Higueras et al., 2018; David I. Ketcheson, 2011)
- High-order parallel extrapolation and deferred correction methods (David I. Ketcheson & bin Waheed, 2014)
- SSP linear multistep methods (Hadjimichael et al., 2016; Hadjimichael & Ketcheson, 2018)
- Dense output formulas for Runge-Kutta methods (David I. Ketcheson et al., 2017)
- Internal stability theory for Runge-Kutta methods (David I. Ketcheson et al., 2014)
- Embedded pairs for Runge-Kutta methods (Conde et al., 2018; Horváth, 2019)

Additional recent applications include (Jin, 2019; David I. Ketcheson, 2019; Norton, 2015; Nüßlein et al., 2020; Ranocha, 2019). As an example of a completely different kind of use, in the fluid dynamics code SpectralDNS, NodePy is used simply for convenience as a way to enable usage of a range of ODE solvers; here NodePy is used only for retrieving the coefficients and not for the time-stepping implementation. This facilitated the work in (David I. Ketcheson et al., 2020), for instance. As can be seen from this list, applications have mostly stemmed from the work of the main developer's research group, but have recently begun to expand beyond that.

## Acknowledgements

Much of the initial NodePy development was performed by D. Ketcheson while he was supported by a DOE Computational Science Graduate Fellowship. Development has also been supported by funding from King Abdullah University of Science and Technology. Additional minor contributions to the code have been provided by Mikael Mortensen, Alex Fikl, Sidafa Conde, John Sellers, Kevin Siswandi, and Colin Macdonald.

## References

Bresten, C., Gottlieb, S., Grant, Z., Higgs, D., Ketcheson, D. I., & Németh, A. (2017). Strong stability preserving multistep Runge-Kutta methods. *Math. Of Comp.*, *86*, 747–769. https://doi.org/10.1090/mcom/3115

Conde, S., Fekete, I., & Shadid, J. N. (2018). Embedded error estimation and adaptive step-size control for optimal explicit strong stability preserving Runge–Kutta methods. *arXiv Preprint arXiv:1806.08693*.

Hadjimichael, Y., & Ketcheson, D. I. (2018). Strong-stability-preserving additive linear multistep methods. *Math. Of Comp.*, *87*(313), 2295–2320. https://doi.org/10.1090/mcom/3296

Hadjimichael, Y., Ketcheson, D. I., Lóczi, L., & Németh, A. (2016). Strong stability preserving explicit linear multistep methods with variable step size. *Siam Journal On Numerical Analysis*, *54*(5), 2799–2832. https://doi.org/10.1137/15M101717X

Hadjimichael, Y., Macdonald, C. B., Ketcheson, D. I., & Verner, J. H. (2013). Strong stability preserving explicit Runge–Kutta methods of maximal effective order. *SIAM Journal on Numerical Analysis*, *51*(4), 2149–2165. https://doi.org/10.1137/120884201

Hairer, E., Nørsett, S. P., & Wanner, G. (1993). *Solving ordinary differential equations I: Nonstiff problems* (Second). Springer.

Hairer, E., & Wanner, G. (1996). *Solving ordinary differential equations II: Stiff and differential-algebraic problems* (Second, Vol. 14). Springer.

Higueras, I., Ketcheson, D. I., & Kocsis, T. A. (2018). Optimal monotonicity-preserving perturbations of a given Runge–Kutta method. *Journal of Scientific Computing*, *76*(3), 1337–1369. https://doi.org/10.1007/s10915-018-0664-3

Horváth, A. (2019). *Embedded pairs for optimal strong stability preserving Runge–Kutta methods* [Master's thesis]. Eötvös Loránd Tudományegyetem.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Jin, X. (2019). *Higher order stabilized time stepping in unfitted finite element method on moving domains* [PhD thesis]. Georg-August-Universität Göttingen.

Ketcheson, David I. (2010). Runge–Kutta methods with minimum storage implementations. *Journal of Computational Physics*, *229*(5), 1763–1773. https://doi.org/10.1016/j.jcp.2009.11.006

Ketcheson, David I. (2011). Step Sizes for Strong Stability Preservation with Downwind-Biased Operators. *SIAM Journal on Numerical Analysis*, *49*(4), 1649. https://doi.org/10.1137/100818674

Ketcheson, David I. (2008). Highly efficient strong stability preserving Runge-Kutta methods with low-storage implementations. *SIAM Journal on Scientific Computing*, *30*(4), 2113–2136. https://doi.org/10.1137/07070485X

Ketcheson, David I. (2019). Relaxation Runge-kutta methods: Conservation and stability for inner-product norms. *SIAM Journal on Numerical Analysis*, *57*(6), 2850–2870. https://doi.org/10.1137/19M1263662

Ketcheson, David I., & bin Waheed, U. (2014). A comparison of high order explicit Runge-Kutta, extrapolation, and deferred correction methods in serial and parallel. *CAMCoS*, *9*(2), 175–200. https://doi.org/10.2140/camcos.2014.9.175

Ketcheson, David I., Gottlieb, S., & Macdonald, C. B. (2011). Strong stability preserving two-step Runge–Kutta methods. *SIAM Journal on Numerical Analysis*, *49*(6), 2618–2639. https://doi.org/10.1137/10080960X

Ketcheson, David I., Lóczi, L., Jangabylova, A., & Kusmanov, A. (2017). Dense output for strong stability preserving Runge–Kutta methods. *Journal of Scientific Computing*, *71*(3), 944–958. https://doi.org/10.1007/s10915-016-0331-5

Ketcheson, David I., Lóczi, L., & Parsani, M. (2014). Internal error propagation in explicit Runge–Kutta methods. *SINUM*, *52*(5), 2227–2249. https://doi.org/10.1137/130936245

Ketcheson, David I., Macdonald, C. B., & Gottlieb, S. (2009). Optimal implicit strong stability preserving Runge–Kutta methods. *Applied Numerical Mathematics*, *59*(2), 373–392. https://doi.org/10.1016/j.apnum.2008.03.034

Ketcheson, David I., Mortensen, M., Parsani, M., & Schilling, N. (2020). More efficient time integration for fourier pseudospectral DNS of incompressible turbulence. *International Journal for Numerical Methods in Fluids*, *92*(2), 79–93. https://doi.org/10.1002/fld.4773

Ketcheson, David I., Parsani, M., Grant, Z. J., Ahmadia, A., & Ranocha, H. (2020). RK-Opt: A package for the design of numerical ODE solvers. *Journal of Open Source Software*, *5*(54), 2514. https://doi.org/10.21105/joss.02514

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., & others. (2017). SymPy: Symbolic computing in python. *PeerJ Computer Science*, *3*, e103. https://doi.org/10.7717/peerj-cs.103

Norton, T. J. T. (2015). *Structure-preserving general linear methods* [PhD thesis]. University of Bath.

Nüßlein, S., Ranocha, H., & Ketcheson, D. I. (2020). Positivity-preserving adaptive Runge-Kutta methods. *arXiv Preprint arXiv:2005.06268*.

Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA.

Ranocha, H. (2019). Some notes on summation by parts time integration methods. *Results in Applied Mathematics*, *1*, 100004. https://doi.org/10.1016/j.rinam.2019.100004

Walt, S. van der, Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22–30. https://doi.org/10.1109/MCSE.2011.37