The Journal of Open Source Software

Simframe: A Python Framework for Scientific Simulations

Sebastian M. Stammler[*1] **and Tilman Birnstiel**[1, 2]

1 University Observatory, Faculty of Physics, Ludwig-Maximilians-Universität München, Scheinerstr. 1, 81679 Munich, Germany **2** Exzellenzcluster ORIGINS, Boltzmannstr. 2, D-85748 Garching, Germany

DOI: 10.21105/joss.03882

Software

- Review
- Repository
- Archive

Editor: Bita Hasheminezhad

Reviewers:

- @schruste
- @lucaferranti

Submitted: 17 August 2021
Published: 11 January 2022

Summary

`Simframe` is a Python framework to facilitate scientific simulations. The scope of the software is to provide a framework which can hold data fields, which can be used to integrate differential equations, and which can read and write data files.

Conceptually, upon initialization `Simframe` is an empty frame that can be filled with `Fields` containing data. `Fields` are derived from `numpy.ndarrays` (Harris et al., 2020), but with extended functionality. The user can then specify differential equations to those data fields and can set up an integrator which is integrating those fields according the given differential equations. Therefore, `Simframe` can only work with data, that can be stored in `NumPy` arrays.

Data fields that should not be integrated themselves, but are still required for the model, can have an update function assigned to them, according to which they will be updated once per integration step.

`Simframe` contains a number of integration schemes of different orders, both for explicit and implicit integration. Furthermore, `Simframe` includes methods to read and write output files.

Due to its modular structure, `Simframe` can be extended at will, for example, by implementing new integration schemes and/or user-defined output formats.

Statement of need

Solving differential equations is part of the daily work of scientists. `Simframe` facilitates this by providing the infrastructure: Data structures, integration schemes, and methods to write and read output files.

On one hand, `Simframe` can be used to quickly solve small scientific problems, and, on the other hand, it can be easily extended to larger projects due to its versatility and modular structure.

Furthermore, `Simframe` is ideal for beginners without programming experience who are taking their first steps in solving differential equations. It can therefore be used to design lectures or practical courses at schools and universities, as it allows students to concentrate on the essentials without having to write larger programs on their own.

Plenty of ODE solver packages already exist for Python, like `solve_ivp` or `odeint` in `SciPy`'s `integrate` module, however, these do not provide data structures, nor input/output capabilities. `Simframe` offers a flexible framework to define, group, and describe data, define

*corresponding author

how it is updated, use existing integrators or define new ones, and to handle writing of data or serializing the entire simulation object, all in one modular package. Existing integrators like `solve_ivp` or `odeint` can be used within `Simframe` by simply adding them to an integration scheme.

Features

Data fields

The data fields of `Simframe` are subclassed `NumPy ndarrays`. The full `NumPy` functionality can therefore be used on `Simframe` data fields. The `ndarrays` have been extended to store additional information about differential equations or update functions and a string description of the field. The data fields can be arranged in groups to facilitate a clear structure within the data frame.

Integration schemes

`Simframe` includes a number of basic integration schemes by default (Hairer et al., 1993). All of the implemented schemes are Runge-Kutta methods of different orders. Some of the methods are adaptive, i.e., they are embedded Runge-Kutta methods, that return an optimal step size for the integration variable, such that the desired accuracy is achieved. The implicit methods require a matrix inversion that is either done directly by `NumPy` or by using the GMRES solver provided by `SciPy` (Virtanen et al., 2020).

Here is a list of all implemented integration schemes:

Order	Scheme			solver
1	Euler	explicit		
1	Euler	implicit		direct
1	Euler	implicit		GMRES
2	Fehlberg	explicit	adaptive	
2	Heun	explicit		
2	Heun-Euler	explicit	adaptive	
2	midpoint	explicit		
2	midpoint	implicit		direct
2	Ralston	explicit		
3	Bogacki-Shampine	explicit	adaptive	
3	Gottlieb-Shu	explicit	adaptive	
3	Heun	explicit		
3	Kutta	explicit		
3	Ralston	explicit		
3	Strong Stability Preserving	explicit		
4	3/8 rule	explicit		
4	Ralston	explicit		
4	Runge-Kutta	explicit		
5	Cash-Karp	explicit	adaptive	
5	Dormand-Prince	explicit	adaptive	

I/O

By default `Simframe` has two options for storing simulation results. One is by storing the data in a separate namespace within the `Simframe` object itself, useful for small simulations to access results without writing/reading data files. Another one is by storing the data in HDF5 data files using the `h5py` package (Collette, 2013).

If configured by the user, `Simframe` is writing dump files, from which the simulation can be resumed, in case the program crashed unexpectedly. These dump files are serialized `Simframe` objects using the `dill` package (McKerns et al., 2012).

Acknowledgements

The authors acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 714769 and funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants 361140270, 325594231, and Germany's Excellence Strategy - EXC-2094 - 390783311.

References

Collette, A. (2013). *Python and HDF5*. O'Reilly.

Hairer, E., Paul, N. S., & Wanner, G. (1993). *Solving ordinary differential equations i*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78862-1

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*. https://doi.org/10.1038/s41586-020-2649-2

McKerns, M. M., Strand, L., Sullivan, T., Fang, A., & Aivazis, M. A. G. (2012). *Building a framework for predictive science*. https://doi.org/10.25080/majora-ebaa42b7-00d

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1. 0 Contributors. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature*. https://doi.org/10.1038/s41592-019-0686-2