scikit-fem: A Python package for finite element assembly

Tom Gustafsson[1] **and G. D. McBain**[2]

1 Department of Mathematics and Systems Analysis, Aalto University **2** Memjet North Ryde Pty Ltd, Macquarie Park, NSW, Australia

DOI: 10.21105/joss.02369

Software

- Review
- Repository
- Archive

Editor: Marie E. Rognes

Reviewers:

- @thelfer
- @AnjaliSandip

Submitted: 06 June 2020
Published: 01 August 2020

Summary

Partial differential equations (PDEs)—such as the Navier–Stokes equations in fluid mechanics, the Maxwell equations in electromagnetism, and the Schrödinger equation in quantum mechanics—are the basic building blocks of modern physics and engineering. The finite element method (FEM) is a flexible computational technique for the discretization and solution of PDEs, especially in the case of complex spatial domains.

Conceptually, FEM transforms a time-independent (or temporally discretized) PDE into a system of linear equations $Ax = b$. `scikit-fem` is a lightweight Python library for the creation, or *assembly*, of the finite element matrix A and vector b. The user loads a computational mesh, picks suitable basis functions, and provides the PDE's weak formulation (Logg, Mardal, Wells, & others, 2012). This results in sparse matrices and vectors compatible with the SciPy (Virtanen et al., 2020) ecosystem.

Purpose and prior art

There exist several open source packages and frameworks that implement the finite element method. `scikit-fem` was developed as a simple and lightweight alternative to the existing Python packages with a focus on computational experimentation and custom PDE-based model development. We rely on pure interpreted Python code on top of the NumPy–SciPy base which makes `scikit-fem` easy to install and portable across multiple operating systems. The reliance on plain NumPy arrays and SciPy sparse matrices enables interoperability with various packages in the Python ecosystem such as meshio (Schlömer, 2020a), pacopy (Schlömer, 2020b), and pyamg (Olson & Schroder, 2018).

In contrast to NGSolve (Schöberl, 2014), FEniCS (Alnæs et al., 2015), Firedrake (Rathgeber et al., 2016), SfePy (Cimrman, Lukeš, & Rohan, 2019), and GetFEM (Renard & Poulios, 2020), `scikit-fem` contains no compiled code making the installation quick and straightforward. We specifically target finite element assembly instead of encapsulating the entire finite element analysis from pre- to postprocessing into a single framework. As a consequence, we cannot provide an end-to-end experience when it comes to, e.g., specific physical models or distributed computing. Our aim is to be generic in terms of PDEs and, hence, support a variety of finite element schemes. Currently `scikit-fem` includes basic support for H^1-, $H(\mathrm{div})$-, $H(\mathrm{curl})$-, and H^2-conforming problems as well as various nonconforming schemes.

`scikit-fem` accepts weak forms that depend on the values and the derivatives of the trial and the test functions, their high-order derivatives, the local mesh parameter, nonuniform material or coefficient fields defined at the quadrature points, or any existing finite element solutions. Iterations related to, e.g., nonlinear problems (Newton's method and the variants, parameter continuation) or adaptive mesh refinement (evaluation of functionals and the marking strategy) should be implemented by the user although we provide basic tools such as interpolation routines and conforming mesh refinement, and examples on using them. The same applies to boundary conditions: the linear system (A, b) is provided as such and eliminating or penalizing the correct degrees-of-freedom, implementing inhomogeneous or periodic boundary conditions should be done separately either by using the various helper routines of `scikit-fem` or by other means. `scikit-fem` has no explicit support for distributed computing although it could be used as a building block in parallel computations such as parameter sweeps or domain decomposition techniques.

Examples and enabled work

The documentation of `scikit-fem` contains over 30 examples that demonstrate the library and its use. The results of some of the examples are highlighted in Figure 1. Several publications already utilize computational results from `scikit-fem`, e.g., McBain, Mallinson, Brown, & Gustafsson (2018), Gustafsson, Stenberg, & Videman (2019), and Gustafsson, Stenberg, & Videman (2020). In addition, `scikit-fem` is used in a recently published Python package for battery modelling (Sulzer, Marquis, Timms, Robinson, & Chapman, 2020).

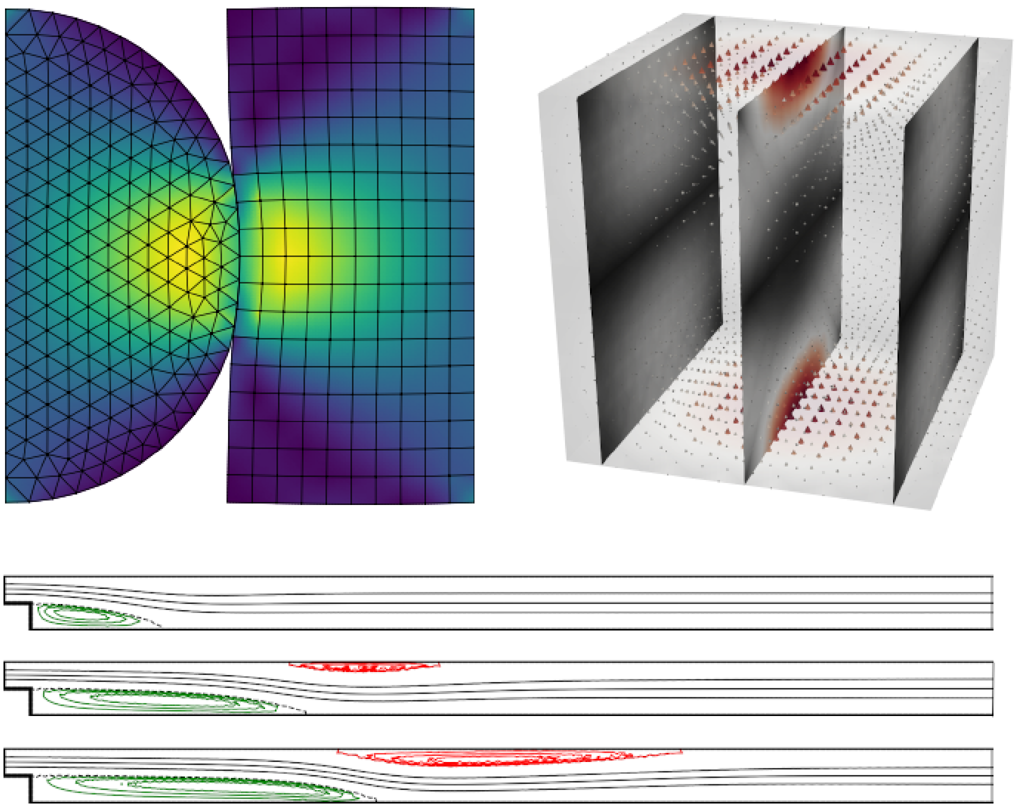
Figure 1: (Top left.) A combination of triangular and quadrilateral elements is used to solve the linear elastic contact problem. (Top right.) The lowest order tetrahedral Nédélec element is used to solve the $H(\text{curl})$-conforming model problem $\nabla \times \nabla \times E + E = f$. The color corresponds to the magnitude of the vector field E. (Bottom.) The Taylor–Hood element is used to solve the Navier–Stokes flow over a backward-facing step for different Reynolds numbers. The first and last two-dimensional figures were generated using `scikit-fem`'s wrapping of matplotlib (Hunter, 2007); three-dimensional postprocessing is more specialized and usually left to export of, e.g., VTK or XDMF formats, via meshio (Schlömer, 2020a) for subsequent rendering in, e.g., ParaView (Ahrens, Geveci, & Law, 2005) as done in the second figure.

Acknowledgements

Tom Gustafsson has received external funding from the Finnish Cultural Foundation and the Academy of Finland (decision nr. 324611) while working on the project.

References

Ahrens, J., Geveci, B., & Law, C. (2005). 36 - ParaView: An End-User Tool for Large-Data Visualization. In C. D. Hansen & C. R. Johnson (Eds.), *Visualization Handbook* (pp. 717–731). Burlington: Butterworth-Heinemann. doi:10.1016/B978-012387582-2/50038-1

Alnæs, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., et al. (2015). The FEniCS Project version 1.5. *Archive of Numerical Software*, *3*(100). doi:10.11588/ans.2015.100.20553

Cimrman, R., Lukeš, V., & Rohan, E. (2019). Multiscale finite element calculations in Python using SfePy. *Advances in Computational Mathematics*. doi:10.1007/s10444-019-09666-0

Gustafsson, T., Stenberg, R., & Videman, J. (2019). Nitsche's method for unilateral contact problems. *Portugaliae Mathematica*, *75*(3), 189–204. doi:10.4171/PM/2016

Gustafsson, T., Stenberg, R., & Videman, J. (2020). On Nitsche's method for elastic contact problems. *SIAM Journal on Scientific Computing*, *42*(2), B425–B446. doi:10.1137/19M1246869

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

Logg, A., Mardal, K.-A., Wells, G. N., & others. (2012). *Automated solution of differential equations by the finite element method*. (A. Logg, K.-A. Mardal, & G. N. Wells, Eds.). Springer. doi:10.1007/978-3-642-23099-8

McBain, G. D., Mallinson, S. G., Brown, B. R., & Gustafsson, T. (2018). Three ways to compute multiport inertance. *ANZIAM Journal*, *60*, 140–155. doi:10.21914/anziamj.v60i0.14058

Olson, L. N., & Schroder, J. B. (2018). PyAMG: Algebraic multigrid solvers in Python v4.0. Retrieved from https://github.com/pyamg/pyamg

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T. T., Bercea, G.-T., et al. (2016). Firedrake: Automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software*, *43*(3). doi:10.1145/2998441

Renard, Y., & Poulios, K. (2020, April). *GetFEM: Automated FE modeling of multiphysics problems based on a generic weak form language*. Retrieved from https://hal.archives-ouvertes.fr/hal-02532422

Schlömer, N. (2020a). nschloe/meshio: Input/output for many mesh formats. Zenodo. doi:10.5281/zenodo.1173115

Schlömer, N. (2020b). nschloe/pacopy: Numerical parameter continuation in Python. Retrieved from https://github.com/nschloe/pacopy

Schöberl, J. (2014). C++ 11 implementation of finite elements in NGSolve. *ASC Report 30/2014, Institute for analysis and scientific computing, Vienna University of Technology*.

Sulzer, V., Marquis, S., Timms, R., Robinson, M., & Chapman, S. (2020). Python Battery Mathematical Modelling (PyBaMM). doi:10.1149/osf.io/67ckj

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. doi:10.1038/s41592-019-0686-2