The Journal of Open Source Software

Bempp-cl: A fast Python based just-in-time compiling boundary element library.

Timo Betcke[1] **and Matthew W. Scroggs**[2]

1 Department of Mathematics, University College London **2** Department of Engineering, University of Cambridge

DOI: 10.21105/joss.02682

Software

- Review
- Repository
- Archive

Editor: Marie E. Rognes

Submitted: 14 September 2020
Published: 05 October 2020

Summary

The boundary element method (BEM) is a numerical method for approximating the solution of certain types of partial differential equations (PDEs) in homogeneous bounded or unbounded domains. The method finds the approximation by discretising a boundary integral equation that can be derived from the PDE. The mathematical background of BEM is covered in, for example, Steinbach (2008) or McLean (2000). Typical applications of BEM include electrostatic problems, and acoustic and electromagnetic scattering.

Bempp-cl is an open-source boundary element method library that can be used to assemble all the standard integral kernels for Laplace, Helmholtz, modified Helmholtz, and Maxwell problems. The library has a user-friendly Python interface that allows the user to use BEM to solve a variety of problems, including problems in electrostatics, acoustics and electromagnetics.

Bempp-cl began life as BEM++, and was a Python library with a C++ computational core. The ++ slowly changed into pp as functionality gradually moved from C++ to Python with only a few core routines remaining in C++. Bempp-cl is the culmination of efforts to fully move to Python, and is an almost complete rewrite of Bempp.

Statement of need

Bempp-cl provides a comprehensive collection of routines for the assembly of boundary integral operators to solve a wide range of relevant application problems. It contains an operator algebra that allows a straight-forward implementation of complex operator preconditioned systems of boundary integral equations (T. Betcke et al., 2020) and in particular implements everything that is required for Calder'on preconditioned Maxwell (Scroggs, Betcke, Burman, Śmigaj, & Wout, 2017) problems. Bempp-cl uses PyOpenCL (Klöckner et al., 2012) to just-in-time compile its computational kernels on a wide range of CPU and GPU devices and modern architectures. Alternatively, a fallback Numba implementation is provided.

An overview of Bempp features

Bempp-cl is divided into two parts: `bempp.api` and `bempp.core`. The user interface of the library is contained in `bempp.api`. The core assembly routines of the library are contained in `bempp.core`. The majority of users of Bempp-cl are unlikely to need to directly interact with the functionality in `bempp.core`.

There are five main steps that are commonly taken when solving a problem with BEM:

1. First a surface grid (or mesh) must be created on which to solve the problem.
2. Finite dimensional function spaces are defined on this grid.
3. Boundary integral operators acting on the function spaces are defined.
4. The operators are discretised and the resulting linear systems solved.
5. Domain potentials and far field operators can be used to evaluate the solution away from the boundary.

Bempp-cl provides routines that implement each of these steps.

Grid Interface

The submodule `bempp.api.shapes` contains the definitions of a number of shapes. From these, grids with various element sizes can be created internally using Gmsh (Geuzaine & Remacle, 2009). Alternatively, meshes can be imported from many formats using the meshio library (Schlömer & contributors, n.d.). Bempp-cl currently only supports flat triangle based surface meshes. Higher-order triangular meshes may be supported in the future.

Function Spaces

Bempp-cl provides piecewise constant and piecewise linear (continuous and discontinuous) function spaces for solving scalar problems. For Maxwell problems, Bempp-cl can create Rao–Wilton–Glisson (Rao, Wilton, & Glisson, 1982) div-conforming spaces and Nédélec (Nédélec, 1980) curl-conforming spaces. In addition to these, Bempp-cl can also generate constant and linear spaces on the barycentric dual grid as well as Buffa–Christiansen div-conforming spaces, as described in Buffa & Christiansen (2007). These spaces can all be created using the `bempp.api.function_space` command.

Boundary operators

Boundary operators for Laplace, Helmholtz, modified Helmholtz and Maxwell problems can be found in the `bempp.api.operators.boundary` submodule, as well as sparse identity operators. For Laplace and Helmholtz problems, Bempp-cl can create single layer, double layer, adjoint double layer and hypersingular operators. For Maxwell problems, both electric field and magnetic field operators can be used.

Discretisation and solvers

Operators are assembled using OpenCL or Numba based dense assembly, or via interface to fast multipole methods. Internally, Bempp-cl uses PyOpenCL (Klöckner et al., 2012) to just-in-time compile its operator assembly routines on a wide range of CPU and GPU compute devices. On systems without OpenCL support, Numba (Lam, Pitrou, & Seibert, 2015) is used to just-in-time compile Python-based assembly kernels, giving a slower but still viable alternative to OpenCL.

Bempp-cl provides an interface to the Exafmm-t library (Wang, Yokota, & Barba, 2020) for faster assembly of larger problems with lower memory requirements using the fast multipole method (FMM). The interface to Exafmm-t is writting in a generic way so that other FMM libraries or alternative matrix compression techniques could be used in future.

The submodule `bempp.api.linalg` contains wrapped versions of SciPy's (Jones, Oliphant, Peterson, & others, 2001) LU, CG, and GMRes solvers. By using SciPy's `LinearOperator` interface, Bempp-cl's boundary operators can easily be used with other iterative solvers.

Potential and far field operators

Potential and far fiels operators for the evaluation at points in the domain or the asymptotic behavior at infinity are included in the `bempp.api.operators.potential` and `bempp.api.operators.far_field` submodules.

Further information

Full documentation of the library, including a number of example Jupyter notebooks, can be found online at `bempp.com` and in in the in-development Bempp Handbook (T. Betcke & Scroggs, 2020).

Acknowledgements

We would like to thank the Exafmm team (Wang et al., 2020), and here in particular Lorena Barba and Tingyu Wang for their efforts to integrate Exafmm-t into Bempp-cl. We further thank the HyENA team (Messner, Urthaler, & Rammerstorfer, 2020) at Graz University of Technology who provided C++ definitions of core numerical quadrature rules, which were translated to Python as part of the development effort for Bempp-cl.

References

Betcke, T., & Scroggs, M. W. (2020). The Bempp handbook. Retrieved from https://bempp.com/handbook/

Betcke, T., Scroggs, M. W., & Śmigaj, W. (2020). Product algebras for Galerkin discretisations of boundary integral operators and their applications. *ACM Transactions on Mathematical Software*, *46*(1, 46), 4:1–4:22. doi:10.1145/3368618

Buffa, A., & Christiansen, S. H. (2007). A dual finite element complex on the barycentric refinement. *Mathematics of Computation*, *76*(260), 1743–1769. doi:10.1016/j.crma.2004.12.022

Geuzaine, C., & Remacle, J.-F. (2009). Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*(11)), 1309–1331. doi:10.1002/nme.2579

Jones, E., Oliphant, T., Peterson, P., & others. (2001). SciPy: Open source scientific tools for Python. Retrieved from http://www.scipy.org/

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., & Fasih, A. (2012). PyCUDA and PyOpenCL: A scripting-based approach to gpu run-time code generation. *Parallel Computing*, *38*(3), 157–174. doi:10.1016/j.parco.2011.09.001

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based Python JIT compiler. In *Proceedings of the second workshop on the llvm compiler infrastructure in hpc*, LLVM '15. Austin, Texas. doi:10.1145/2833157.2833162

McLean, W. (2000). *Strongly elliptic systems and boundary integral equations* (p. xiv+357). Cambridge University Press.

Messner, M., Urthaler, P., & Rammerstorfer, F. (2020). HyENA: Hyperbolic and elliptic numerical analysis. Retrieved from https://www.tugraz.at/en/institutes/am-bm/research/software/

Nédélec, J.-C. (1980). Mixed finite elements in $\mathbb{R}^3$. *Numerische Mathematik*, *35*(3), 315–341. doi:10.1007/BF01396415

Rao, S. M., Wilton, D. R., & Glisson, A. W. (1982). Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Transactions on antennas and propagation*, *30*(3), 409–418. doi:10.1109/TAP.1982.1142818

Schlömer, N., & contributors. (n.d.). Meshio: I/O for mesh files. Retrieved from https://github.com/nschloe/meshio

Scroggs, M. W., Betcke, T., Burman, E., Śmigaj, W., & Wout, E. van 't. (2017). Software frameworks for integral equations in electromagnetic scattering based on calderón identities. *Computers & Mathematics with Applications*, *74*(11), 2897–2914. doi:10.1016/j.camwa.2017.07.049

Steinbach, O. (2008). *Numerical approximation methods for elliptic boundary value problems* (p. xii+386). Springer-Verlag. doi:10.1007/978-0-387-68805-3

Wang, T., Yokota, R., & Barba, L. A. (2020). Exafmm-t. *Submitted to Journal of Open Source Software*.