The Journal of Open Source Software

Basix: a runtime finite element basis evaluation library

Matthew W. Scroggs [2], **Igor A. Baratta** [2], **Chris N. Richardson** [1], **and Garth N. Wells** [2]

1 BP Institute, University of Cambridge **2** Department of Engineering, University of Cambridge

DOI: 10.21105/joss.03982

Software

- Review
- Repository
- Archive

Editor: Jed Brown

Reviewers:

- @tisaac
- @wence-

Submitted: 02 November 2021
Published: 25 May 2022

Summary

The finite element method (FEM) (Ciarlet, 1978) is a widely used numerical method for approximating the solution of partial differential equations (PDEs). Solving a problem using FEM involves discretising the problem and searching for a solution in a finite dimensional space: these finite spaces are created by defining a finite element on each cell of a mesh.

Following Ciarlet (1978), a finite element is commonly defined by a triple $(R, \mathcal{V}, \mathcal{L})$, where:

- R is the reference cell, for example a triangle with vertices at (0,0), (1,0) and (0,1);
- $\mathcal{V}$ is a finite dimensional polynomial space, for example $\text{span}\{1, x, y, x^2, xy, y^2\}$;
- $\mathcal{L}$ is a basis of the dual space $\{f : \mathcal{V} \to \mathbb{R} \,|\, f \text{ is linear}\}$, for example the set of functionals that evaluate a function at the vertices of the triangle and at the midpoints of its edges.

The basis functions of the finite element are the polynomials in $\mathcal{V}$ such that one functional in $\mathcal{L}$ gives the value 1 for that function and all other functionals in $\mathcal{L}$ give 0. The examples given above define a degree 2 Lagrange space on a triangle; the basis functions of this space are shown in Figure 1.

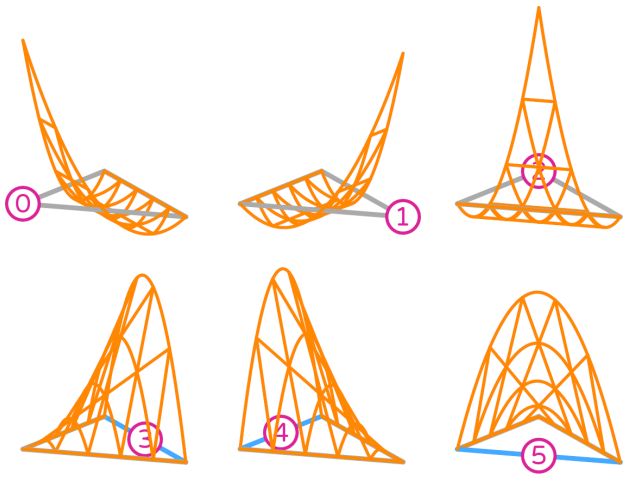

Figure 1: The six basis functions of a degree 2 Lagrange space on a triangle. The upper three functions arise from point evaluations at the vertices. The lower three arise from point evaluations at the midpoints of the edges. These diagrams are taken from DefElement (The DefElement contributors, 2021).

The functionals in $\mathcal{L}$ are each associated with a degree of freedom (DOF) of the finite element space. Each functional (or DOF) is additionally associated with a sub-entity of the reference cell. Ensuring that the same coefficients are assigned to the DOFs of neighbouring cells associated with a shared sub-entity gives the finite element space the desired continuity properties.

Basix is a C++ library that creates and tabulates a range of finite elements on triangles, tetrahedra, quadrilaterals, hexahedra, pyramids, and prisms. The majority of Basix's functionality can be used via its Python interface. A full list of currently supported elements is included below.

For many elements, the functionals in $\mathcal{L}$ are defined to be integrals on a sub-entity of the cell. Basix provides a range of quadrature rules to compute these integrals, including Gauss–Jacobi, Gauss–Lobatto–Legendre, and Xiao–Gimbutas (Xiao & Gimbutas, 2010). Internally, Basix uses xtensor (Mabille et al., 2021) for matrix and tensor storage and manipulation.

Basix forms part of FEniCSx alongside DOLFINx (Wells, Ballarin, et al., 2021), FFCx (Wells, Baratta, et al., 2021), and UFL (Alnæs et al., 2014). FEniCSx is the latest development version of FEniCS, a popular open source finite element project (Alnæs et al., 2015).

Statement of need

Basix allows users to:

- evaluate finite element basis functions and their derivatives at a set of points;
- access geometric and topological information about reference cells;
- apply push forward and pull back operations to map data between a reference cell and a physical cell;
- permute and transform DOFs to allow higher degree elements to be used on arbitrary meshes; and
- interpolate into a finite element space and between finite element spaces.

A full list of the built-in elements currently supported in Basix is included below. In addition to these element, users can create their own custom elements via either C++ or Python.

Basix's support for permuting and transforming DOFs is one of its key features. In an arbitrary mesh, neighbouring cells will not necessarily agree on the orientation of shared sub-entities. For higher degree elements, this will lead to a mismatch in the positioning and orientation of the DOFs associated with these sub-entities, as shown in Figure 2. Our use of permuting and transforming to rectify this issue is described in full detail in Scroggs et al. (2021).

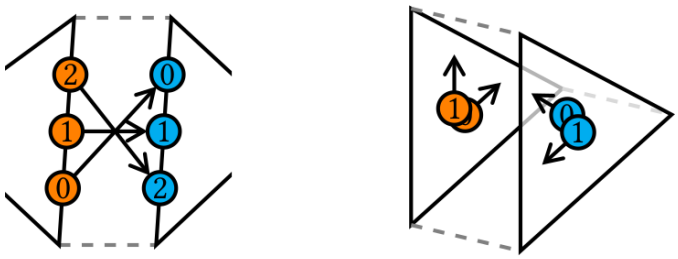

Figure 2: Left: In a degree 4 Lagrange space on a triangle, there are three DOFs associated with each edge. If two neighbouring triangles disagree on the orientation of a shared edge, then the positions of the DOFs will not match up. Right: In a degree 1 Nédélec first kind element on a tetrahedron, there are two DOFs associated with each triangular face. These DOFs will be associated with a vector tangential to the face of the triangles. If two neighbouring cells do not agree on the orientation of the triangle, then the directions of these vectors will not match.

In order to compute the basis functions of a finite element, each functional in $\mathcal{L}$ can be applied to the elements of a basis of the polynomial set $\mathcal{V}$ to compute the *dual matrix*. This dual matrix can then be inverted to obtain the coefficients that define the finite element basis functions in terms of the basis of $\mathcal{V}$. In NGSolve (Schöberl, 2014), the functionals for this approach are implemented in C++ using lambda functions. In MFEM (Anderson et al., 2021), the functionals are described in C++ using a set of points and weights: for point

evaluation functionals, there will be one point and the weight will be 1; for integral functionals, quadrature points and weights are used. In FIAT (Kirby, 2004) and Symfem (Scroggs, 2021), the functionals are implemented as Python objects. In Basix, we follow an approach similar to that taken by MFEM, and define the functionals in C++ using a set of points and weights.

In many other libraries—including Dune (Sander, 2020) and Bempp (Betcke & Scroggs, 2021; Śmigaj et al., 2015)—the basis functions of the finite element space are implemented explicitly.

In a large number of finite element libraries—including NGSolve and MFEM—the finite element definitions are included in the core of the library. In Dune, the finite element definitions are separated into the `dune-localfunctions` module. FIAT and Basix are the finite element definition and tabulations libraries used by legacy FEniCS and FEniCSx (respectively). Symfem is a standalone element definition library that computes basis functions symbolically. By separating the finite element definitions from the core of the code, users are able to tabulate elements and obtain information about them without having to interact with the other components of the library. In FEniCSx for example, this allows users who want to create custom integration kernels to get information about elements from Basix without having to extract information from the core of the full finite element library.

An additional advantage of this modular approach is that it allows us to adjust how elements are implemented and add new elements without needing to make changes to the other components of FEniCSx.

For some high degree finite elements on tensor product cells (quadrilaterals and hexahedra), elements can be more efficiently tabulated using sum factorisation: the elements can be represented as the product of elements defined on an interval, and so can be tabulated by combining copies of tabulated data on an interval rather than tabulating on the full cell. This tensor product evaluation is implemented in MFEM, FInAT (Homolya et al., 2017) and deal.ii (Bangerth et al., 2007). We have experimented with supporting these factorisations in Basix: in particular, details of the factorisations for Lagrange elements are currently provided by the library. We plan to support them more fully and for a wider range of elements in a future release.

The Python library FIAT (Kirby, 2004) (which is part of the legacy FEniCS library alongside UFL, FFC (Logg et al., 2012) and DOLFIN (Logg & Wells, 2010)) serves a similar purpose to Basix and can perform many of the same operations (with the exception of permutations and transformations) on triangles, tetrahedra, quadrilaterals, and hexahedra. As FIAT is written in Python, the FFC library would use the information from FIAT to generate code that could be used by the C++ finite element library DOLFIN.

An advantage of using Basix is the ability to call functions from C++ at runtime. This has allowed us to greatly reduce the amount of code generated in FFCx compared to FFC, as well as simplifying much of the implementation, while still allowing FFCx to access the information it needs using Basix's Python interface.

Supported elements

Interval

In Basix, the sub-entities of the reference interval are numbered as shown in Figure 3. The following elements are supported on an interval:

- Lagrange
- bubble
- serendipity (Arnold & Awanou, 2011)

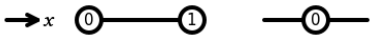

Figure 3: The numbering of a reference interval.

Triangle

In Basix, the sub-entities of the reference triangle are numbered as shown in Figure 4. The following elements are supported on a triangle:

- Lagrange
- Nédélec first kind (Nédélec, 1980)
- Raviart–Thomas (Raviart & Thomas, 1977)
- Nédélec second kind (Nédélec, 1986)
- Brezzi–Douglas–Marini (Brezzi et al., 1985)
- Regge (Christiansen, 2011; Regge, 1961)
- Hellan–Herrmann–Johnson (Arnold & Walker, 2020)
- Crouzeix–Raviart (Crouzeix & Raviart, 1973)
- bubble

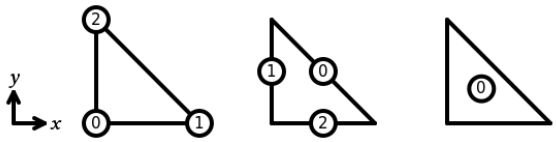

Figure 4: The numbering of a reference triangle.

Quadrilateral

In Basix, the sub-entities of the reference quadrilateral are numbered as shown in Figure 5. The following elements are supported on a quadrilateral:

- Lagrange
- Nédélec first kind
- Raviart–Thomas
- Nédélec second kind (Arnold & Awanou, 2014)
- Brezzi–Douglas–Marini (Arnold & Awanou, 2014)
- bubble
- DPC
- serendipity

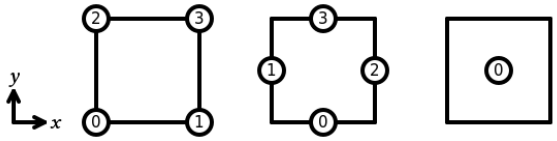

Figure 5: The numbering of a reference quadrilateral.

Tetrahedron

In Basix, the sub-entities of the reference tetrahedron are numbered as shown in Figure 6. The following elements are supported on a tetrahedron:

- Lagrange
- Nédélec first kind
- Raviart–Thomas

- Nédélec second kind
- Brezzi–Douglas–Marini
- Regge
- Crouzeix–Raviart
- bubble

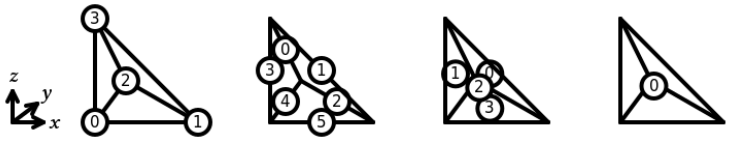

Figure 6: The numbering of a reference tetrahedron.

Hexahedron

In Basix, the sub-entities of the reference hexahedron are numbered as shown in Figure 7. The following elements are supported on a hexahedron:

- Lagrange
- Nédélec first kind
- Raviart–Thomas
- Nédélec second kind
- Brezzi–Douglas–Marini
- bubble
- DPC
- serendipity

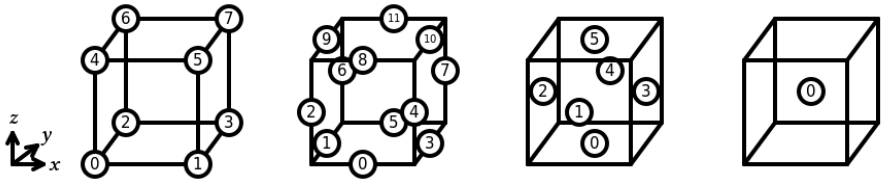

Figure 7: The numbering of a reference hexahedron.

Prism

In Basix, the sub-entities of the reference prism are numbered as shown in Figure 8. The following elements are supported on a prism:

- Lagrange

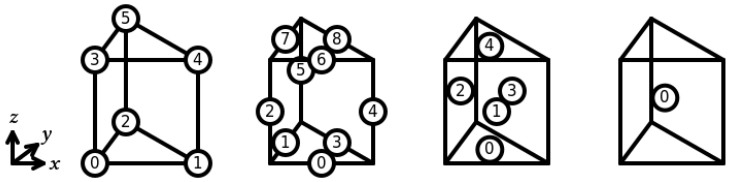

Figure 8: The numbering of a reference prism.

Pyramid

In Basix, the sub-entities of the reference pyramid are numbered as shown in Figure 9. The following elements are supported on a pyramid:

- Lagrange

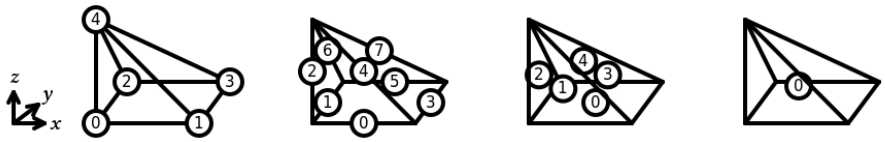

Figure 9: The numbering of a reference pyramid.

References

Alnæs, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C. N., Ring, J., Rognes, M. E., & Wells, G. N. (2015). The FEniCS project version 1.5. *Archive of Numerical Software*, *3*(100), 9–23. https://doi.org/10.11588/ans.2015.100.20553

Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., & Wells, G. N. (2014). Unified Form Language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*. https://doi.org/10.1145/2566630

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Cerveny, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., Pazner, W., Stowell, M., Tomov, V., Akkerman, I., Dahm, J., Medina, D., & Zampini, S. (2021). MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, *81*, 42–74. https://doi.org/10.1016/j.camwa.2020.06.009

Arnold, D. N., & Awanou, G. (2011). The serendipity family of finite elements. *Foundations of Computational Mathematics*, *11*(3), 337–344. https://doi.org/10.1007/s10208-011-9087-3

Arnold, D. N., & Awanou, G. (2014). Finite element differential forms on cubical meshes. *Mathematics of Computation*, *83*, 1551–5170. https://doi.org/10.1090/S0025-5718-2013-02783-4

Arnold, D. N., & Walker, S. W. (2020). The Hellan–Herrmann–Johnson method with curved elements. *SIAM Journal on Numberical Analysis*, *58*(5), 2829–2855. https://doi.org/10.1137/19M1288723

Bangerth, W., Hartmann, R., & Kanschat, G. (2007). Deal.II—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, *33*(4:24), 1–27. https://doi.org/10.1145/1268776.1268779

Betcke, T., & Scroggs, M. W. (2021). Bempp-cl: A fast Python based just-in-time compiling boundary element library. *Journal of Open Source Software*, *6*(59, 6), 2879. https://doi.org/10.21105/joss.02879

Brezzi, F., Douglas, J., & Marini, L. D. (1985). Two families of mixed finite elements for second order elliptic problems. *Numerische Mathematik*, *47*, 217–235. https://doi.org/10.1007/BF01389710

Christiansen, S. H. (2011). On the linearization of Regge calculus. *Numerische Mathematik*, *119*(4), 613–640. https://doi.org/10.1007/s00211-011-0394-z

Ciarlet, P. G. (1978). *The finite element method for elliptic problems*. North-Holland.

Crouzeix, M., & Raviart, P.-A. (1973). Conforming and nonconforming finite element methods for solving the stationary Stokes equations. *Revue Française d'Automatique, Informatique Et Recherche Opérationnelle*, *3*, 33–75. https://doi.org/10.1051/m2an/197307R300331

Homolya, M., Kirby, R. C., & Ham, D. A. (2017). *Exposing and exploiting structure: Optimal code generation for high-order finite element methods*. https://arxiv.org/abs/1711.02473

Kirby, R. C. (2004). Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, *30*(4), 502–516. https://doi.org/10.1145/1039813.1039820

Logg, A., Ølgaard, K. B., Rognes, M. E., & Wells, G. N. (2012). *FFC: The FEniCS Form Compiler*. 283–302. https://doi.org/10.1007/978-3-642-23099-8_11

Logg, A., & Wells, G. N. (2010). DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software*, *37*. https://doi.org/10.1145/1731022.1731030

Mabille, J., Beier, T., Brochart, D., Corlay, S., de Geus, T., Delsalle, A., Koethe, U., GitHub user kolibri91, Prouvost, A., Renou, M., GitHub user SoundDev, Vollprecht, W., & Zhu, J. (2021). *xtensor: C++ tensors with broadcasting and lazy computing*. https://github.com/xtensor-stack/xtensor

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

Nédélec, J.-C. (1986). A new family of mixed finite elements in $\mathbb{R}^3$. *Numerische Mathematik*, *50*(1), 57–81. https://doi.org/10.1007/BF01389668

Raviart, P.-A., & Thomas, J.-M. (1977). A mixed finite element method for 2nd order elliptic problems. In I. Galligani & E. Magenes (Eds.), *Mathematical aspects of finite element methods* (Vol. 606, pp. 292–315).

Regge, T. (1961). General relativity without coordinates. *Il Nuovo Cimento*, *19*(3), 558–571. https://doi.org/10.1007/BF02733251

Sander, O. (2020). Local finite elements and the dune-localfunctions module. In *DUNE — the distributed and unified numerics environment* (pp. 301–324). Springer International Publishing. https://doi.org/10.1007/978-3-030-59702-3_8

Schöberl, J. (2014). *C++11 implementation of finite elements in NGSolve*. Institute for Analysis; Scientific Computing, TU Wien. https://www.asc.tuwien.ac.at/~schoeberl/wiki/publications/ngs-cpp11.pdf

Scroggs, M. W. (2021). Symfem: A symbolic finite element definition library. *Journal of Open Source Software*, *6*(64, 6), 3556. https://doi.org/10.21105/joss.03556

Scroggs, M. W., Dokken, J. S., Richardson, C. N., & Wells, G. N. (2021). *Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes*. https://doi.org/10.1145/3524456

Śmigaj, W., Betcke, T., Arridge, S., Phillips, J., & Schweiger, M. (2015). Solving boundary integral problems with BEM++. *ACM Transactions on Mathematical Software*, *41*(2), 6:1–6:40. https://doi.org/10.1145/2590830

The DefElement contributors. (2021). *DefElement: An encyclopedia of finite element definitions*. https://defelement.com.

Wells, G. N., Ballarin, F., Baratta, I. A., Dean, J. P., Dokken, J. S., Hale, J. S., Habera, M., Richardson, C. N., Scroggs, M. W., & Sime, N. (2021). *DOLFINx: Next generation FEniCS problem solving environment*. https://github.com/FEniCS/dolfinx

Wells, G. N., Baratta, I. A., Habera, M., Hale, J. S., Richardson, C. N., & Scroggs, M. W. (2021). *FFCx: Next generation FEniCS form compiler*. https://github.com/FEniCS/ffcx

Xiao, H., & Gimbutas, Z. (2010). A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions. *Computers & Mathematics with Applications*, *59*(2), 663–676. https://doi.org/10.1016/j.camwa.2009.10.027