

Code generation for generally mapped finite elements

ROBERT C. KIRBY, Baylor University, USA

LAWRENCE MITCHELL, Durham University, UK

Many classical finite elements such as the Argyris and Bell elements have long been absent from high-level PDE software. Building on recent theoretical work, we describe how to implement very general finite element transformations in FInAT and hence into the Firedrake finite element system. Numerical results evaluate the new elements, comparing them to existing methods for classical problems. For a second order model problem, we find that new elements give smooth solutions at a mild increase in cost over standard Lagrange elements. For fourth-order problems, however, the newly-enabled methods significantly outperform interior penalty formulations. We also give some advanced use cases, solving the nonlinear Cahn-Hilliard equation and some biharmonic eigenvalue problems (including Chladni plates) using C^1 discretizations.

1 INTRODUCTION

The FIAT¹ project [Kirby 2004] has provided a new generation of finite element codes such as FEniCS [Logg et al. 2012] and Firedrake [Rathgeber et al. 2016] with a diverse set of finite element basis functions. Internally, FIAT represents the nodal bases for finite elements in terms of orthogonal polynomials and dense linear algebra, and it is capable of tabulating reference basis functions and their derivatives at any desired points. Wrapped behind a high-level declarative interface, many elements generally regarded as difficult to implement can be rapidly deployed in a flexible, performant environment. In the case of FEniCS and Firedrake, FIAT’s usage is hidden behind a *form compiler* [Homolya et al. 2018; Kirby and Logg 2006] that translates UFL [Alnæs et al. 2014], a domain-specific language for variational forms, into efficient lower-level code for constructing matrices and vectors. As an example of how this works, we refer to the code listing in Fig. 1.

```

from firedrake import *
mesh = UnitSquareMesh(10, 10)
V = FunctionSpace(mesh, "CG", 3)
u = TrialFunction(V)
v = TestFunction(V)
f = ...
a = (dot(grad(v), grad(u)) + v * u) * dx
L = f * v * dx
u0 = Function(V)
solve(a == L, u0)

```

Fig. 1. Sample code listing for a model problem in Firedrake using cubic Lagrange basis functions.

However, time and usage have revealed certain limitations of FIAT. Its Python implementation makes it most suitable to “run once” in a given simulation, tabulating basis functions at reference element quadrature points and then letting the rest of the finite element code map these to each cell in the mesh. Moreover, the dense tables of arrays that FIAT produces make it difficult to exploit tensor-product or other structure that might lead to more efficient algorithms. As a third issue, it is difficult for FIAT to communicate how bases should be mapped, leaving clients to handle their own pullbacks. This works naturally when reference and physical elements are equivalent under affine or Piola pullback,

¹The FInite element Automatic Tabulator

but implementing elements (see Fig. 2) such as Hermite [Ciarlet and Raviart 1972], Morley [Morley 1971], Bell [Bell 1969], and Argyris [Argyris et al. 1968] that require more complex transformations [Domínguez and Sayas 2008; Kirby 2018] easily leads to a proliferation of special cases. It also requires encoding basis tabulation and transformation in completely different code components. Just as the dense tabulation historically limited the ability of FEniCS and Firedrake to employ sum-factorization and other fast algorithms, lack of general transformations has limited their ability to address higher-order PDEs and other applications benefitting from smooth approximations.

Recent work on FInAT [Homolya et al. 2017] has addressed many of these issues. Unlike FIAT, FInAT is not a tabulator. Instead, it constructs abstract syntax using GEM [Homolya et al. 2018] for basis function tabulation. By emitting GEM, which is the tensor-based intermediate representation of the `tsfc` form compiler, FInAT provides symbolic, as well as numerical, tabulations. For example, as well as including syntax for table lookup from FIAT-generated values, FInAT also provides a kind of element calculus by which one may construct vector- or tensor-valued elements; use tensor-products of lower-dimensional bases to build structured bases on quadrilaterals, prisms, or hexahedra; or indicate that basis evaluation at particular points satisfies a Kronecker delta property. In Homolya et al. [2017], we describe these features of FInAT and extensions of `tsfc` to make use of them in generating efficient code. Our goal is for FInAT to become a “single source of truth” for finite element bases. This includes not only basis function evaluation, like FIAT, but also structural and algorithmic considerations as well as reference element transformations.

In this work, we present further developments in FInAT to enable the transformations of Argyris and other such elements described in Kirby [2018]. In particular, we equip FInAT with abstract syntax to provide basis transformations. FInAT’s routines for basis evaluation now assume that the caller will provide an object capable of providing (abstract syntax for) the required geometric quantities such as Jacobians, normals, and tangents required by the mapping. By extending `tsfc` to provide these, we have been able to deploy these extensions within Firedrake, making the utilization of such bases (nearly) seamless from the user perspective. Essentially, we enable users to replace the line `V = FunctionSpace(mesh, "CG", 3)` in Fig. 1 with `V = FunctionSpace(mesh, "Hermite", 3)` or any of the other elements such as those depicted in Fig. 2.

Although our examples are two-dimensional, our work extends in principle (and code infrastructure supports) tetrahedral elements as well. The code already has Hermite tetrahedra [Ciarlet and Raviart 1972], and implementing Morley-like elements [Wang and Xu 2013] would be straightforward. Three-dimensional C^1 analogs of the Argyris triangle, however, require very high polynomial degree. This high degree and the associated high cost of direct methods limits their practical use. An alternative is to use numerical approaches such as isogeometric analysis [Hughes et al. 2005] or maximum-entropy methods [Arroyo and Ortiz 2006] which naturally offer high order continuity, potentially at reduced cost. Of these methods, simplicial splines [Lai and Schumaker 2007] are of future interest. Such splines would require additional infrastructure in FInAT to reason about macro elements, but would bring Firedrake and other FInAT clients a step closer to the flexibility to change order and continuity available on (patches of) mapped logically uniform meshes in isogeometric analysis.

Among existing high-level PDE codes like FEniCS, Firedrake, deal.II [Bangerth et al. 2007], Sundance [Long 2003], and Feel++ [Prud’Homme et al. 2012] we have only found a general support for the kinds of elements we consider here in GetFEM++ [Fournié et al. 2010; Perduta 2013], which implements the Morley, and Argyris elements on triangles, and the Hermite element on triangles and tetrahedra. Except for FEniCS and Firedrake, these codes are all C++ libraries with varying degrees of abstraction in the problem description. While there is no impediment in such libraries to implementing the transformation theory described in Kirby [2018], we believe UFL affords a more succinct user interface,

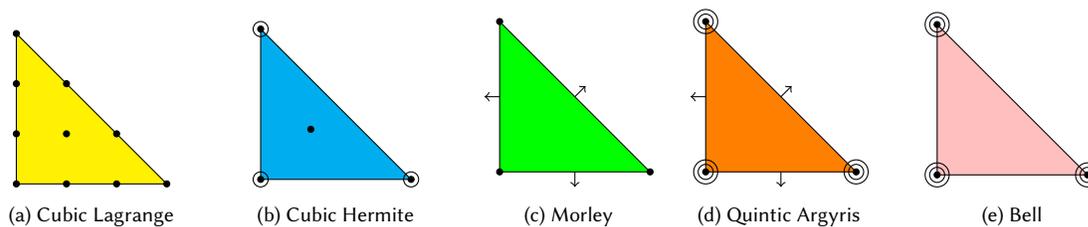


Fig. 2. Some famous triangular elements. Solid dots represent point value degrees of freedom, smaller circles represent gradients, and larger circles represent the collection of second derivatives. The arrows indicate directional derivatives evaluated at the tail of the arrow.

and code generation more opportunities for optimization [Homolya et al. 2017; Kirby et al. 2005, 2006; Luporini et al. 2017],

In the rest of the paper, we first review the theory developed in Kirby [2018] in section 2. In section 3 we describe the modifications to FInAT and the rest of the Firedrake code stack that are necessary to implement and automate the generalized mappings for these new elements. Strongly-enforced boundary conditions are the one limitation we face when implementing these elements. In section 3.2, we describe some of the mathematical issues that have to date prevented a general approach, so (like GetFEM++) we defer to weak enforcement of essential boundary conditions through Nitsche’s method [Nitsche 1971]. To test our implementation and the practical use of these elements, we present a series of numerical experiments in section 4. These include an attempt to quantify the practical effect in terms of the cost of our general transformations.

2 OVERVIEW OF TRANSFORMATION THEORY

Let \hat{K} be a reference domain with vertices $\{\hat{\mathbf{v}}_i\}_{i=0}^d$ and let K be a typical element with vertices $\{\mathbf{v}_i\}_{i=0}^d$. We let the affine mapping $F : K \rightarrow \hat{K}$ be as shown in Fig. 3. Given any function $\hat{\phi}$ defined on \hat{K} , we define its *pullback* by

$$\phi = F^*(\hat{\phi}) = \hat{\phi} \circ F. \quad (2.1)$$

To evaluate the new function ϕ at a point \mathbf{x} that is the image of some $\hat{\mathbf{x}}$ under F , we have

$$\phi(\mathbf{x}) = \hat{\phi}(\hat{\mathbf{x}}). \quad (2.2)$$

Similarly, if J is the Jacobian matrix of the mapping F , we can differentiate ϕ by the chain rule

$$\nabla \phi = J^T \hat{\nabla} \hat{\phi}, \quad (2.3)$$

where ∇ denotes differentiation with respect to the coordinate system on K and $\hat{\nabla}$ to that on \hat{K} .

We shall also need the *push forward* of linear functionals, which is defined simply by composition with the pullback. If P is a set of functions mapping K into \mathbb{R} (or some other vector space), then

$$F_*(n) = n \circ F^* \quad \forall n \in P. \quad (2.4)$$

Lagrange finite elements in simplicial domains form *affine-equivalent* families. One can find the Lagrange basis on \hat{K} and then obtain the Lagrange basis on any other domain K by means of the affine pullback. From eq. (2.2) and eq. (2.3), it is straightforward to tabulate the basis functions and their derivatives at reference domain quadrature points and map the results to any cell in the mesh. A similar property holds for vector-valued Raviart-Thomas [Raviart and

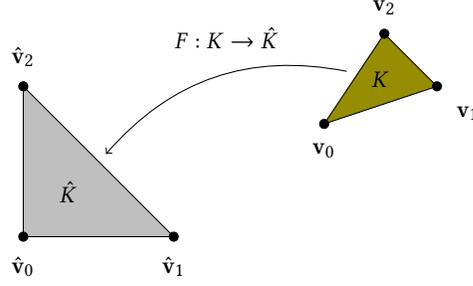


Fig. 3. Affine mapping to a reference cell \hat{K} from a typical cell K .

Thomas 1977] and Nédélec elements [Nédélec 1980], which can be defined on a reference domain and mapped via Piola transforms. However, other elements, such as scalar-valued elements with derivative degrees of freedom, do not typically satisfy affine equivalence, and this complicates the use of a reference element.

The cubic Hermite triangle [Ciarlet and Raviart 1972] provides a simple example. The ten degrees of freedom parametrizing a Hermite triangle are the function values and gradients at the vertices and the function value at the barycenter. We will specify the gradient in terms of the partial derivatives in the local Cartesian coordinate directions as shown in Fig. 4. Now, let $\hat{\psi}(\hat{x})$ be the reference Hermite polynomial having unit horizontal derivative at \hat{v}_0 , with all



Fig. 4. Reference and physical cubic Hermite elements with gradient degrees of freedom expressed in terms of local Cartesian directional derivatives.

other degrees of freedom vanishing. The chain rule tells us that

$$\nabla\psi(\mathbf{v}_0) = J^T \hat{\nabla}\hat{\psi}(\hat{\mathbf{v}}_0), \quad (2.5)$$

which is not equal to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ except in very special geometry. It is clear that $\psi(\mathbf{v}_i) = 0$ and that the gradient vanishes at the other two vertices.

As we show in Kirby [2018], we can take a simple linear combination of the pullbacks of the Hermite basis functions on \hat{K} to obtain the Hermite basis functions on K . Let $\{\psi_{3i}\}_{i=0}^2$ be the basis functions taking unit value at vertex \mathbf{v}_i , $\{\psi_{3i+1}\}_{i=0}^2$ having unit x -derivative at \mathbf{v}_i , $\{\psi_{3i+2}\}_{i=0}^2$ having unit y -derivative at \mathbf{v}_i , and ψ_9 having unit value at the

barycenter and the corresponding numbering of $\{\hat{\psi}_i\}_{i=0}^9$ on \hat{K} . Then, our theory gives that

$$\begin{bmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \psi_4 \\ \psi_5 \\ \psi_6 \\ \psi_7 \\ \psi_8 \\ \psi_9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial \hat{x}}{\partial x} & \frac{\partial \hat{x}}{\partial y} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial \hat{y}}{\partial x} & \frac{\partial \hat{y}}{\partial y} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial \hat{x}}{\partial x} & \frac{\partial \hat{x}}{\partial y} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial \hat{y}}{\partial x} & \frac{\partial \hat{y}}{\partial y} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial \hat{x}}{\partial x} & \frac{\partial \hat{x}}{\partial y} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial \hat{y}}{\partial x} & \frac{\partial \hat{y}}{\partial y} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F^*(\hat{\psi}_0) \\ F^*(\hat{\psi}_1) \\ F^*(\hat{\psi}_2) \\ F^*(\hat{\psi}_3) \\ F^*(\hat{\psi}_4) \\ F^*(\hat{\psi}_5) \\ F^*(\hat{\psi}_6) \\ F^*(\hat{\psi}_7) \\ F^*(\hat{\psi}_8) \\ F^*(\hat{\psi}_9) \end{bmatrix}. \quad (2.6)$$

Note that the gradient basis-functions are mapped in pairs, while the pull-back in fact maps the basis functions for point values correctly.

More generally, if F^* maps not just Hermite but some other finite element function space on \hat{K} onto that on K , then there exists a matrix M such that

$$\psi_i = \sum_{j=1}^N M_{ij} F^*(\hat{\psi}_j). \quad (2.7)$$

It is frequently much easier to construct $V = M^T$, which relates the *push-forward* of the physical finite element nodes to the reference element nodes.

While the assumption behind eq. (2.7) applies to Morley and Argyris elements, the corresponding M is more complicated because they do not form affine-interpolation equivalent families of elements. Equivalently [Brenner and Scott 2008], the spans of the reference nodes and pushed-forward physical nodes do not coincide. To see why this presents difficulty, consider Fig. 5, which shows the physical nodes pushed forward to the reference element. Since there is only a single directional derivative on each edge, pairs of basis function corresponding to a gradient (in some coordinates) cannot be adjusted by the Jacobian like in the Hermite case.

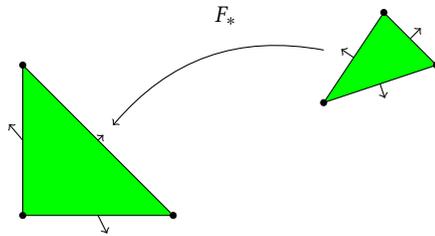


Fig. 5. Pushing forward the Morley derivative nodes in physical space does *not* produce the reference derivative nodes.

In Kirby [2018], we develop a three-step mapping technique that generalizes the approach in Domínguez and Sayas [2008]. First, one extends the reference and physical nodal sets with additional nodes such that their spans do coincide. The transformation between these sets can be constructed in a method similar to Hermite. Second, the action of the new nodes on the finite element space must be constructed in terms of the given ones. This can typically be done with interpolation theory. Finally, one extracts the nodes of the mapped finite element as a subset of the enriched set. Each

stage can be expressed as a matrix, and we have that

$$V = EV^C D. \quad (2.8)$$

Here, D , which has more rows than columns, maps the finite element nodes to the extended nodes. The square matrix V^C relates the push-forward of the extended nodes in physical space to the extended reference nodes. E has a single nonzero per row, selecting out the subset of extended reference nodes belonging to the reference finite element.

In the case of the Morley element, we extend the vertex values and normal derivatives at edge midpoints with the tangential derivatives at the same, as shown in Fig. 6. The tangential derivative of a quadratic at the edge midpoint

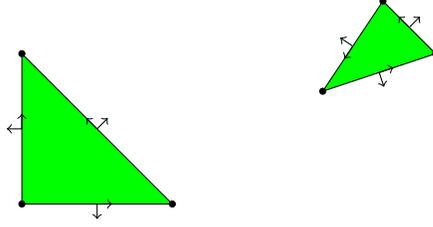


Fig. 6. Extended nodal sets for Morley reference (left) and typical (right) element are formed by including tangential derivatives along with normal derivatives at each edge midpoint.

can actually be computed by differencing the associated vertex values, so the D matrix is relatively easy to construct. The matrix V^C is similar to Hermite, although it is slightly more involved since it transforms gradients in normal and tangential rather than Cartesian coordinates. Let \mathbf{n}_i and \mathbf{t}_i denote the normal and tangent to edge i of K and $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{t}}_i$ those for \hat{K} . Then, we put $G_i = [\mathbf{n}_i \ \mathbf{t}_i]^T$ with an analogous definition for \hat{G}_i and hence define

$$B^i = \hat{G}_i J^{-T} G_i^T. \quad (2.9)$$

The matrix V^C is logically block-diagonal with unit values corresponding to the vertex degrees of freedom and B^i for the derivative nodes on edge i . Putting this together, we have

$$M^T = V = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{-B_{01}^0}{\ell_0} & \frac{B_{01}^0}{\ell_0} & B_{00}^0 & 0 & 0 \\ \frac{-B_{01}^1}{\ell_1} & 0 & \frac{B_{01}^1}{\ell_1} & 0 & B_{00}^1 & 0 \\ \frac{-B_{01}^2}{\ell_2} & \frac{B_{01}^2}{\ell_2} & 0 & 0 & 0 & B_{00}^2 \end{bmatrix}. \quad (2.10)$$

Owing to the higher degree and second-order derivatives, the Argyris triangle (Fig. 2d) transformation is more complicated, but follows the same kind of process. The Bell element (Fig. 2e), however, requires an extension of this theory. It is billed as a “reduced” or “simplified” element because it lacks the Argyris element’s normal derivatives at edge midpoints. However, these are removed by constraining the function space to have cubic normal derivatives rather than the quartic ones typical to a quintic polynomial. This means that the affine pullback does not preserve the function space. We deal with this by constructing and mapping a full quintic finite element, a subset of whose basis functions are the Bell basis functions. We refer the reader to Kirby [2018] for further details.

We address one further wrinkle in Kirby [2018]. In particular, global basis functions with a unit derivative value may differ in size considerably from those corresponding to a unit point value. The resulting scale separation between basis functions leads to a considerable increase in the condition number that degrades as $h \searrow 0$. To compensate for this, we can adjust M by composing with another linear transform that scales the derivative basis functions by a suitable, locally-defined, function of the mesh spacing h .

3 INCORPORATING TRANSFORMATIONS IN THE FIREDRAKE STACK

Making these elements work seamlessly from the end user perspective requires careful modification of several components of Firedrake. Throughout this process, we have been driven by two main design goals. First, we seek to provide all information related to basis function evaluation through FInAT (internally using FIAT for reference element tabulation as needed). Second, we seek to avoid making FInAT dependent on other Firedrake components so that it may be used by projects besides Firedrake. To address these two concerns we have developed an abstract interface, which a form compiler must implement, that allows FInAT to request expressions for the evaluation of geometric information on physical cells.

To facilitate these callbacks between the element library on the one hand, and the form compiler on the other, they need to agree on a common language for exchanging information. Where FIAT communicated with the form compiler through numerical arrays, FInAT communicates with the form compiler by exchanging GEM expressions [Homolya et al. 2018]. GEM is the intermediate language used in both `tsfc` and FInAT to describe tensor algebra. It is the natural choice for the exchange of geometric information, since any form compiler using FInAT must already understand GEM. Since it is a fairly basic language describing array indexing and arithmetic, it should be possible to convert from GEM to other representations such as pymbolic (<https://mathematician.de/software/pymbolic/>) or sympy [Joyner et al. 2012]. Our new elements then overload the `basis_evaluation` method to call FIAT for reference values and then provide abstract syntax for constructing and applying M to those values.

By isolating the transformations at the level of basis functions in this way, we required almost no changes in the rest of the Firedrake code base. In UFL, we made trivial changes to “register” the new elements: adding them to a list of available elements in the language. In `tsfc`, we bind the UFL elements to concrete FInAT classes and provide the necessary implementation of geometric mapping information. Most of the expressions are readily obtainable by translation of appropriate UFL, which has symbolic objects representing Jacobians, facet normals, and so forth. The mesh cell size at vertices, used for scaling of derivative basis functions, presents certain difficulties. The form compiler only makes reference to a single element, but the transformation theory requires a characteristic mesh size that is available at vertices and agreed upon by all cells sharing a given vertex. We therefore provide the mesh cell size as a normal coefficient field, and require that Firedrake provide it when assembling the local element tensor. This marginally changes the interface between `tsfc` and Firedrake: element kernels may now have one extra argument. However, this is handled internally and unbeknownst to the user.

3.1 Inserting the transformation matrix

Given the ability to construct the transformation M , it can be inserted into a finite element code in (at least) two distinct places. For one, the transformation can be applied to reference values at each quadrature point before the local integration is carried out on each cell. Alternatively, one could compute a local vector or matrix with the untransformed basis values and post-multiply by the resulting transformations as needed. Similarly, one can either evaluate members of

the function space by transforming the basis functions or by using the associative property to transform the expansion coefficients and then using the unmodified basis functions.

In the first, case, suppose that a bilinear form over some $\Omega \subset \mathbb{R}^2$ requires both basis values and gradients. If we have a basis with N_f members and a quadrature rule with N_q points, then transforming all the basis values and gradients will require $3 N_q$ matrix-vector products with M . Forming a load vector or matrix action will also require similar additional effort.

On the other hand, suppose that one forms an element matrix with untransformed basis functions and applies the transformation afterward. Essentially, one must compute the congruence transform

$$A = M\tilde{A}M^T, \quad (3.1)$$

where \tilde{A} is the ‘wrong’ element matrix. Beyond computing \tilde{A} , one must act on each row of \tilde{A} with M and then on each column of the result. This amounts to $2N_f$ matrix-vector products with M . Unless $N_q < \frac{2}{3}N_f$, using eq. (3.1) will be moderately less expensive. At any rate, as we point out in Kirby [2018], either are relatively small additional costs compared to the overall cost of forming the element matrix.

Although the cost of matrix formation is not significantly different with these two approaches, the difference is somewhat more substantial with load vectors or matrix actions. Considering the matrix action via eq. (3.1):

$$Au = M\tilde{A}M^T u, \quad (3.2)$$

where u is vector of degrees of freedom on an element, this can be factorized so that it requires only two applications of M by using the associative property in the correct way, reusing a kernel for computing the mapping $u \mapsto \tilde{A}u$.

In our implementation, we have opted simply to map the basis functions at each quadrature point. This required somewhat less invasive modifications to the form compiler and, as we see in section 4.2, does not seem to create any significant performance penalties. Future work could include additional refactoring of `tsfc` to support and compare either mode and select based on the particular use case or user-specified settings.

3.2 Boundary conditions

Strongly enforcing boundary conditions with most of these elements appears to be rather difficult. Instead, we use Nitsche-type penalty methods for essential boundary conditions. Similar techniques are used in GetFEM++ and have also been adopted in spline-based finite elements [Embar et al. 2010] for similar reasons. Fortunately, expressing the additional terms in the variational problem in UFL is not difficult. Here, we briefly demonstrate some of the issues in strongly constraining boundary values.

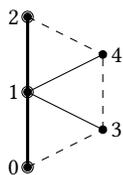
With Lagrange elements, (approximately) setting $u|_{\partial\Omega} = g$ for some continuous function g by interpolation is relatively straightforward, fixing u to agree with g at the boundary nodes. With Hermite elements, for example, this becomes more difficult. Since g lives only on $\partial\Omega$, one would have to provide a differentiable extension of g into Ω and, moreover, a code interface to obtaining the derivatives of g to enforce boundary conditions.

Further difficulties for Hermite elements are apparent, even just considering the simpler case of the homogeneous condition $u|_{\partial\Omega} = 0$. On a line segment, a cubic polynomial vanishes iff its endpoint values and derivatives do. So, on a triangle containing a boundary edge, we must constrain the correct values and tangential derivatives on the boundary vertices to force $u = 0$ on the edge.

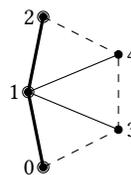
On one hand, consider a portion of a domain with a vertical boundary, as shown in Fig. 7a. To enforce $u = 0$ on the boundary segments connecting vertices 0 to 1 and 1 to 2, one can set the nodal values of u at vertices 0, 1, and 2 to

zero. Then, if one also sets the y derivative of u at these points to zero, u will vanish along both of these segments. This amounts to six constraints. In the more general case of colinear edges that are not aligned with a coordinate axis, one still has six constraints. However, these set linear combinations of the gradient components rather than particular degrees of freedom.

On the other hand, consider the case in Fig. 7b, where the boundary is not straight. Clearly, we must set $u = 0$ at vertices 0, 1, and 2. Moreover, we need to set the tangential derivatives in the directions running along the edges at vertices 0 and 1. Setting these tangential derivatives requires constraining a linear combination of the gradient components. We must set derivatives in two distinct directions at vertex 1 to force $u = 0$ along both of the edges. This, then, forces the entire gradient of u to vanish at vertex 1. Two immediate problems arise. First, the homogeneous Dirichlet condition on the boundary cannot be enforced without imposing an additional homogeneous Neumann condition at the vertices. Also, the number of constraints changes as soon as the boundary is not straight as we now require *seven* constraints: three function values, a linear combinations of derivatives at two vertices, and the full gradient at a third. As an additional technical problem, interpolation of inhomogeneous boundary conditions in spaces with derivative degrees of freedom requires higher regularity of the data.



(a) Setting $u = 0$ on the solid boundary requires setting vertex values and y - derivatives at nodes 0, 1, and 2 to be zero. No x - derivatives are modified in this case.



(b) Setting $u = 0$ on the solid boundary requires setting vertex values at nodes 0, 1, and 2 to be zero as well as the entire gradient at vertex 1 and linear combinations of the directional derivatives at vertices 0 and 2.

Fig. 7. Enforcing homogeneous Dirichlet boundary conditions with Hermite elements.

4 EXAMPLES

In this section, we turn to demonstrating and evaluating the new capabilities of Firedrake. In the first three subsections, we explore various computational aspects of these newly-enabled elements for some model problems. We consider the accuracy and computational costs of various discretizations of the Poisson and biharmonic problems. In particular, we can give per-element FLOP estimates for the generated code for various discretizations and also compare run time to build and solve the global sparse finite element matrices. Later, we also present some more advanced examples of usage. All timing results were obtained on a single core of an Intel E5-2640v3 (Haswell) processor. The measured single core STREAM triad [McCalpin 1995] bandwidth is 13 GB/s, and the peak floating point performance of a single core is 41.6 Gflop/s.

Dealing with high-order polynomials can lead to difficulties with linear solvers, especially for fourth-order problems such as the biharmonic equation. We do not explore optimal preconditioners or other aspects of iterative methods [Bramble and Zhang 1995; Brenner 1989; Xu 1996], instead using sparse LU factorization provided by PaStiX [Hénon et al. 2002], accessed through the PETSc library [Balay et al. 2018, 1997]. In all examples where we study the convergence,

we have also included one iteration of iterative refinement, which adds only a small extra cost, to improve the roundoff error that can become rather large at high degree or on fine meshes.

When using Lagrange elements for Poisson’s equation (or more generally, conforming discretizations of second-order elliptic operators), *static condensation* is also a reasonable technique to perform before sparse factorization. This is implemented through SLATE [Gibson et al. 2018], a domain-specific language for expressing localized linear algebra on finite element tensors. While the experiments in Gibson et al. [2018] demonstrate its effectiveness for hybridizing mixed and HDG-type methods, here we use its facilities for static condensation for continuous Galerkin methods. Since SLATE interfaces with the rest of Firedrake as a preconditioner acting on the implicit matrices described in Kirby and Mitchell [2018], we are able to factor either the original or condensed global matrix by a change of solver options. Except for Hermite, our newly-enabled elements have no internal degrees of freedom, so static condensation gives no assistance in these cases. Similarly, it does not work for interior penalty biharmonic discretizations since internal degrees of freedom couple across cell boundaries through jump terms on the derivatives.

Because scalable solvers are quite difficult and somewhat distinct from the innovations in code generation required for this work, we have focused on single-process performance. These new developments in no way clash with Firedrake’s existing parallel abilities, but absent effective preconditioners one may be limited to sparse direct parallel solvers such as PaStiX or MUMPS [Amestoy et al. 2000].

4.1 Convergence on model problems

As a first example, we demonstrate that we obtain theoretically-predicted convergence rates for some model problems on the unit square, labelled Ω . Our coarsest mesh is shown in Fig. 8, which has internal vertices of a regular 8×8 mesh sinusoidally perturbed to introduce geometric nonuniformity.

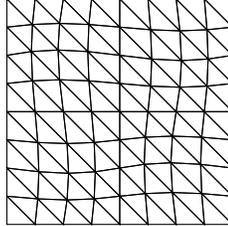


Fig. 8. Coarsest mesh for convergence studies.

First, we validate our new elements for Poisson’s equation

$$-\Delta u = f, \tag{4.1}$$

subject to Dirichlet boundary conditions $u = 0$ on $\partial\Omega$. One multiplies by a test function and integrates by parts in the typical way to obtain a variational formulation. In line with our observation in section 3.2, we will enforce the boundary condition through a consistent modification of the bilinear form, as introduced by Nitsche [1971].

$$a_h(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} (\nabla u \cdot n) v \, ds - \int_{\partial\Omega} u (\nabla v \cdot n) \, ds + \frac{\alpha}{h} \int_{\partial\Omega} uv \, ds. \tag{4.2}$$

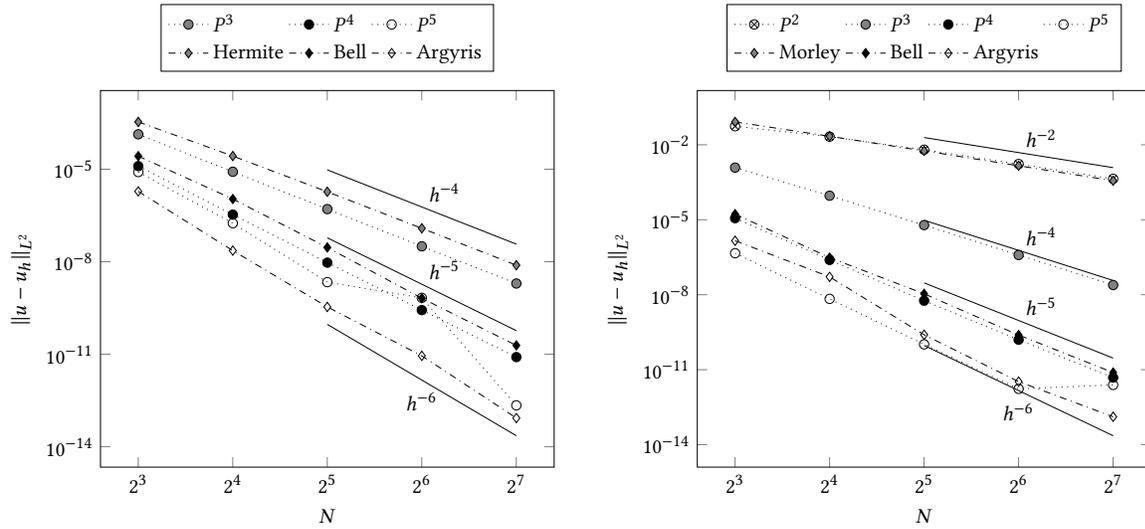
Here, α is a positive constant large enough to render the bilinear form coercive in a perturbed H^1 norm, and h is the characteristic mesh size. For $V_h \subset H^1$ consisting of one of the finite element spaces under consideration, we solve the

discrete variational problem

$$a_h(u_h, v_h) = (f, v_h) \equiv \int_{\Omega} f v_h \, dx \quad (4.3)$$

although additional terms would appear on the right-hand side with inhomogeneous Dirichlet boundary conditions. Implementing this in Firedrake requires straightforward modifications of the listing in Fig. 1.

Fig. 9a plots the L^2 error versus mesh refinement on uniform refinements of the mesh in Fig. 8. P^3 appears slightly more accurate, but of the same order, as cubic Hermite. Since P^3 is a strictly larger space than Hermite, its H^1 best approximation result (which affects the L^2 estimates through Aubin-Nitsche duality) should also be slightly better. The same comparison holds for P^4 and Bell, while we do not observe the same effect for P^5 and Argyris elements.



(a) L^2 error in solving Poisson's equation on a perturbed $N \times N$ mesh.

(b) L^2 error in solving biharmonic equation on a perturbed $N \times N$ mesh.

Fig. 9. L^2 errors solving prototypical second- and fourth-order PDEs using various higher-order finite elements.

While some situations could call for smooth approximations of second order equations, many of our new elements come into their own for the plate-bending biharmonic equation

$$\Delta^2 u = f \quad (4.4)$$

on Ω . In our examples, we consider clamped boundary conditions $u = \frac{\partial u}{\partial n} = 0$ on $\partial\Omega$. Following Brenner and Scott [2008], we employ the bilinear form

$$a(u, v) = \int_{\Omega} \Delta u \Delta v - (1 - \nu) (2u_{xx}v_{yy} + 2u_{yy}v_{xx} - 4u_{xy}v_{xy}) \, dx, \quad (4.5)$$

where $0 \leq \nu \leq \frac{1}{2}$ is the Poisson ratio for the plate. The terms multiplied by $(1 - \nu)$ may separately be integrated by parts to give $u_{xxyy}v$ times zero plus terms for incorporating strongly-supported boundary conditions. Subject to clamped boundary conditions, or on any subspace of H^2 not containing linear polynomials over Ω , the bilinear form a is coercive. Argyris and Bell elements give conforming, optimal-order approximations. Morley elements give a suitable, albeit suboptimal, nonconforming method.

As an alternative to discretizing the H^2 bilinear form with Morley, Bell, or Argyris elements, it is possible to adapt interior penalty techniques for C^0 elements [Engel et al. 2002; Wells et al. 2004]. This allows use of P^k elements for $k \geq 2$ by means of penalizing jumps in derivatives. These methods do not require the extra terms scaled by $(1 - \nu)$ in eq. (4.5), instead using the bilinear form

$$a_h(u, v) = \sum_{K \in \mathcal{T}} \int_K \Delta u \Delta v \, dx + \sum_{E \in \mathcal{E}_h^{\text{int}}} \left(\int_E \frac{\alpha}{h_E} \llbracket \nabla u \rrbracket \llbracket \nabla v \rrbracket \, ds - \int_E \{\Delta u\} \llbracket \nabla v \rrbracket \, ds - \int_E \llbracket \nabla u \rrbracket \{\Delta v\} \, ds \right). \quad (4.6)$$

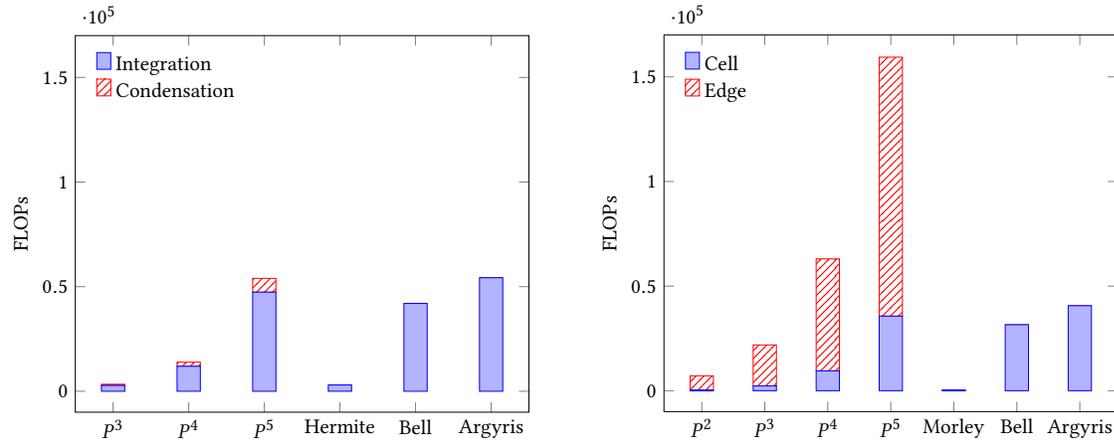
Here, K are the cells in a triangulation \mathcal{T} and $\mathcal{E}_h^{\text{int}}$ are the interior edges. The operators $\llbracket \cdot \rrbracket$ and $\{\cdot\}$ are the standard jump and average operators used in discontinuous Galerkin methods. This method is known to give optimal convergence rates in L^2 for $k > 2$ (but only second order for quadratics), and requires only Lagrange or other standard H^1 -conforming elements. In particular, it is frequently used with Lagrange elements. Although we will not include an exhaustive comparison, it is also worth mentioning other alternatives such as the Reissner-Mindlin formulation. This leads to a system of singularly-perturbed H^1 equations [Durán and Liberman 1992; Hale et al. 2018] and allows direct enforcement of Dirichlet boundary conditions for rotations at the cost of a mixed system. An alternative approach, using a mixed formulation is given by Li [Li 2018] using generalized Regge-type elements.

Fig. 9b plots the error versus mesh refinement on uniform refinements of the mesh in Fig. 8 for the two biharmonic models. Unlike the Poisson equation, we are not directly comparing the same discretization with different elements, but an H^2 discretization (nonconforming for Morley) versus an interior penalty method. Still, we see that P^2 with interior penalty gives slightly lower error than Morley elements, and P^4 is slightly more accurate than Bell until the finest meshes. On the other hand, Argyris is more accurate than P^5 on all the meshes, for which we observe suboptimal convergence rates, even after iterative refinement.

4.2 Cost of forming element matrices

While our newly-enabled elements provide effective solutions, we must also consider the computational cost associated with using them. Clearly, when the element transformation $M \neq \mathbb{I}$, extra work is required to perform element-level integrals as compared to Lagrange and other affine-equivalent elements. Here, we document the additional FLOPs required and the effect on time to assemble matrices for the problems considered above. Fig. 10a shows the tsfc-reported FLOP counts for building the element stiffness matrix for the Laplacian using Lagrange elements of degrees 3 through 5 as well as Hermite, Bell, and Argyris elements. These numbers include constructing the element Jacobian from the physical coordinates, transforming basis function gradients using M and the chain rule, and the loop over quadrature points and pairs of basis functions. They do not include any interaction with the global sparse matrix. In addition, we report the per-element cost of performing static condensation for Lagrange elements with a separate bar above that of forming the element matrices.

We notice that forming the element integral with Hermite elements requires slightly more operations than with P^3 Lagrange elements. Since the transformation eq. (2.6) requires only modifying some of the basis functions with the (already-computed) Jacobian, the actual up-tick in operation count is very small. On the other hand, the gap between P^5 Lagrange and Argyris is a bit larger. Two factors contribute to this. First, the Argyris transformation requires the evaluation of several geometric quantities (tangents, normals, edge lengths, Hessians) not otherwise required. Second, there are relatively more nonzeros in M for Argyris than Hermite. Comparing Bell elements to P^4 shows them to be quite a bit more expensive for the same order of approximation. Bell elements have nearly as many degrees of freedom as Argyris, require a complicated transformation matrix M , and also require a more accurate quadrature rule owing to



(a) Laplace operator of eq. (4.2). Stacked above P^k elements are the additional FLOPs required to form the local Schur complement in static condensation

(b) Biharmonic operator. The last three entries relate to the bilinear form in eq. (4.5) and include no edge terms, while the P^k entries are from the DG form eq. (4.6). The per-cell cost of the edge integrals seems to drive the overall FLOP count for interior penalty methods.

Fig. 10. Per-element FLOP counts for assembling Laplace and biharmonic operators using various elements.

the presence of quintic terms. Fig. 10a shows that the per-element cost is indeed closer to (but less than) the Argyris element than it is to P^4 . Beyond this, static condensation adds a small per-element cost to P^k elements, making P^3 and Hermite roughly comparable as well as P^5 and Argyris.

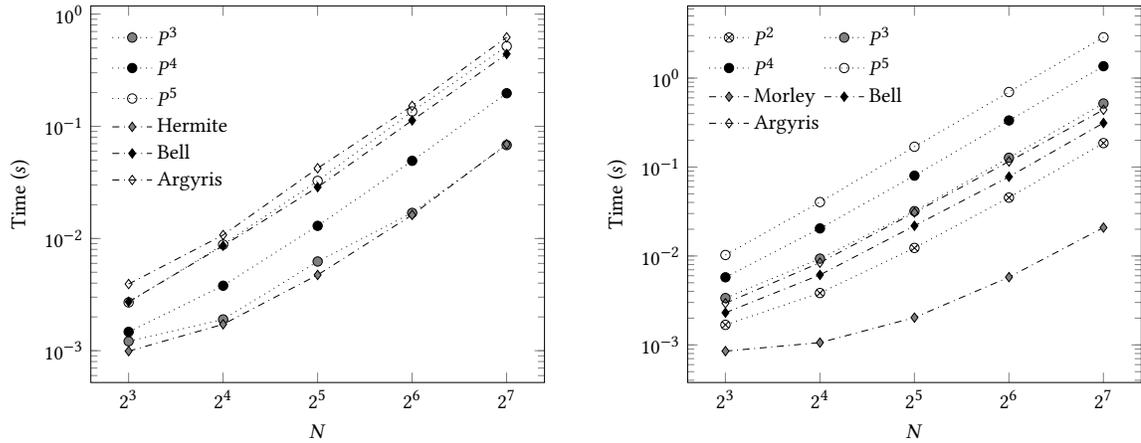
We now turn to the biharmonic operator. Although the bilinear form eq. (4.5) has a more complex cell integral and requires the transformation M , the cost of the edge terms in the bilinear form eq. (4.6) is quite large. Since there are about 1.5 times as many edges as triangles in our meshes, we report the FLOP count of the kernel to perform the element integral plus 1.5 times the per-facet FLOP count. Observing Fig. 10b, the H^2 methods have considerably lower per-element operation counts.

4.3 Building and solving global systems

Per-element FLOP counts are not the only factor affecting the overall cost of assembling linear systems. In fact, the concentration of degrees of freedom at vertices for Hermite, Bell, and Argyris elements leads to overall smaller systems with markedly different costs of assembly and sparsity patterns than Lagrange elements.

Fig. 11a gives the time to build the discrete Poisson system on the meshes considered in our convergence study above. We observe that Hermite and P^3 , and Argyris and P^5 elements require very similar assembly time, with Bell somewhat higher than P^4 . By sharing vertex degrees of freedom between many elements, we may be achieving slightly better insertion patterns when summing element matrices into the global system for H^2 elements. We speculate that this, combined with the overall smaller system size help offset the higher per-element FLOP counts. Fig. 11b shows a clear win for H^2 elements. With fewer global degrees of freedom and less expensive element-level computation, we see a much lower cost to assemble than for the interior penalty method with P^k elements.

As a further illustration, we present the global sparsity pattern on an coarse 8×8 mesh for each of our discretizations for Poisson and the biharmonic operator in Figs. 12 and 13. For Poisson, we compare P^k elements with their smaller and



(a) Laplace assembly time. We see that the higher FLOP counts for C^1 elements (c.f. Fig. 10a) only have a noticeable effect on assembly time for Bell elements (relative to P^4).

(b) Biharmonic assembly time. The lower FLOP counts for C^1 discretisations translate into significantly lower assembly times relative to the appropriate C^0 interior penalty scheme.

Fig. 11. Time to assemble the Laplace and biharmonic operators on a perturbed $N \times N$ mesh. The timings include preallocation of the matrix sparsity (which is amortizable over solves), as well as building element tensors, and matrix insertion.

slightly denser statically condensed counterparts in the first row. The H^2 element with comparable accuracy appears in below each pair of P^k patterns, demonstrating the smaller, but denser system in each case. For the biharmonic operator, edge terms in the interior penalty formulation prevent static condensation for P^k elements, and H^2 elements have few, if any, interior degrees of freedom to eliminate, so we just consider the global stiffness matrix. We see that the H^2 elements in fact give smaller and *sparser* systems than P^k elements, owing to the absence of edge terms. Since the sparsity pattern is driven by the elements and mesh connectivity rather than bilinear forms, we note that these trends in sparsity patterns are rather generic for second- and fourth-order scalar problems, respectively.

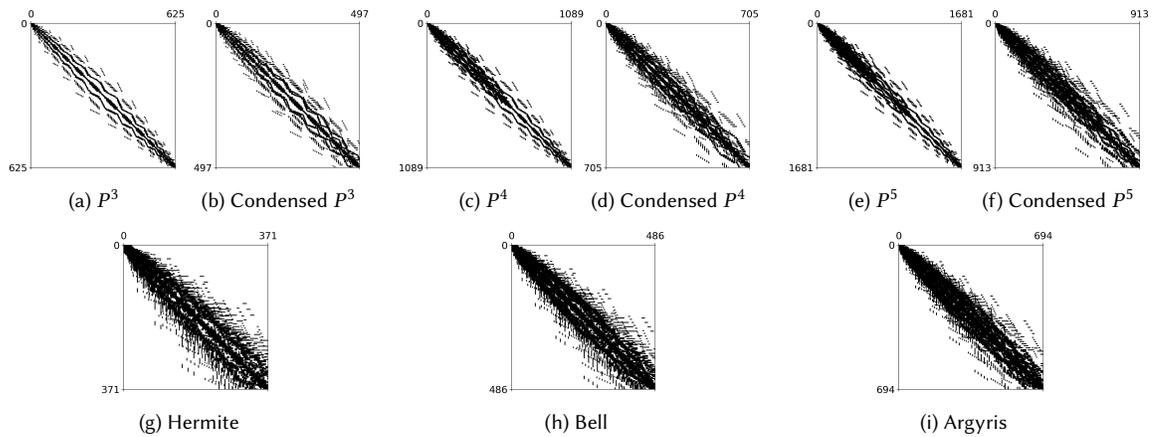


Fig. 12. Sparsity patterns of the discrete Laplacian on a regular 8×8 mesh using H^1 elements (top row) and H^2 elements (bottom row).

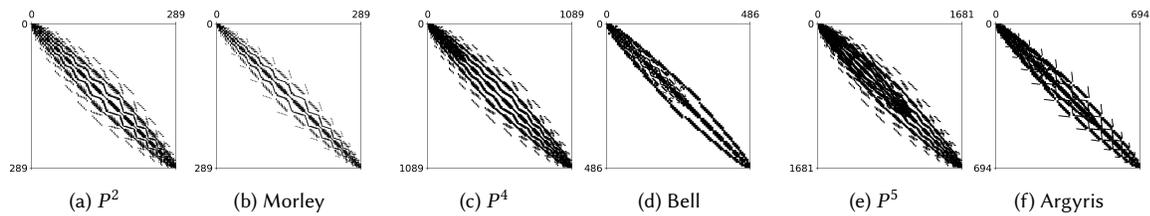
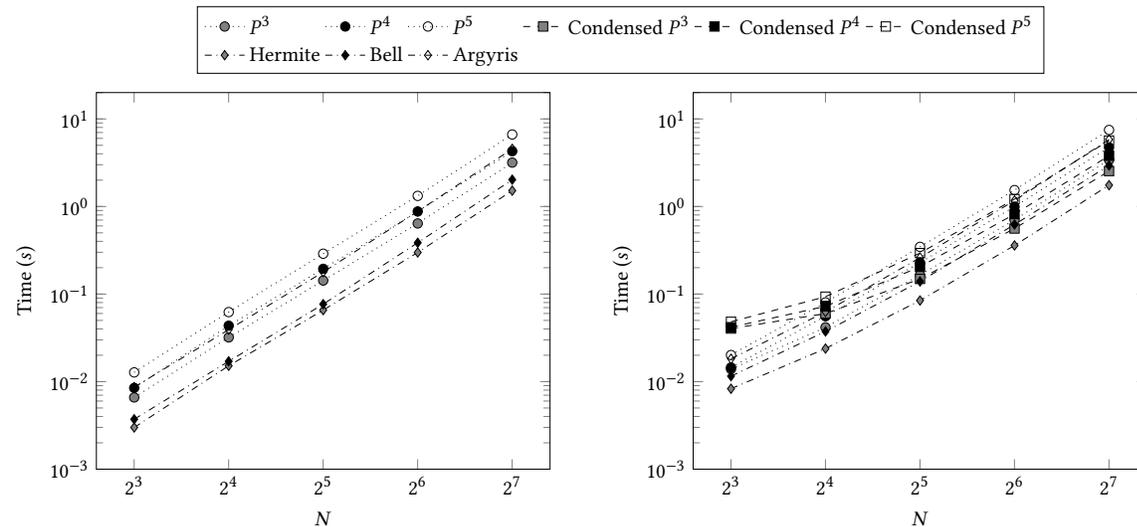


Fig. 13. Sparsity patterns of the discrete biharmonic operator on a regular 8×8 mesh for interior penalty discretizations (a, c, e) and using H^2 discretizations (b, d, f).

Having considered the cost of assembling the matrices and the resulting sparsity patterns, we now present timings for solving the systems with sparse LU factorization provided by PaStiX, plus iterative refinement. These times are shown in Fig. 14a. For this solver, it appears that Hermite and Bell systems are actually cheaper to solve than any of the P^k systems, and Argyris appears to have a comparable cost to P^4 rather than P^5 . We note that these observations only on one sparse direct solver with “out-of-the-box” options and leave open the possibility that reordering or other options for this or other sparse factorizations could lead to different rankings among the solvers. We also consider the



(a) Time to factor and solve the sparse linear system for the discrete Laplacian on $N \times N$ mesh, including one iteration of iterative refinement.

(b) Total solver time for Poisson's equation on $N \times N$ mesh. This includes element integration (and condensation), sparse matrix preallocation and assembly, and factorization.

Fig. 14. Time to solution for the Poisson equation using different discretisations. We see that static condensation does not gain us anything for small problems, due to the overhead involved in forming the condensed system.

effects of static condensation. In this case, the full global stiffness matrix is never formed. In Fig. 14b, we compare the overall solver time for statically condensed systems to the combined times of building and solving the uncondensed systems. It appears that condensation improves the total run-time for P^k elements on the finer meshes, and we would expect this trend to continue under additional mesh refinement. In addition to the visualized sparsity patterns, we have also computed the average number of nonzeros per row and condition number on a coarse (8×8) mesh for the

Poisson operator, shown in Table 1a. Notice how the conditioning of the H^2 elements is significantly worse. In contrast, when applied to the biharmonic operator, the picture is reversed (Table 1b). The H^2 discretizations have both better conditioning (Bell apart) and lower numbers of nonzeros per row than an interior penalty discretization using H^1 elements.

Element	Total DoFs	Nonzeros/row	κ
P^3	625	16.1	$4.49 \cdot 10^2$
P_c^3	497	15.3	$2.77 \cdot 10^2$
Hermite	371	18.6	$3.13 \cdot 10^3$
P^4	1,089	22.5	$1.29 \cdot 10^3$
P_c^4	705	20.0	$6.36 \cdot 10^2$
Bell	486	36.8	$1.56 \cdot 10^7$
P^5	1,681	29.8	$3.98 \cdot 10^3$
P_c^5	913	24.7	$2.99 \cdot 10^3$
Argyris	694	41.0	$9.58 \cdot 10^6$

Element	Total DoFs	Nonzeros/row	κ
P^2	289	21.6	$9.23 \cdot 10^3$
Morley	289	10.6	$4.82 \cdot 10^2$
P^4	1,089	54.8	$3.87 \cdot 10^5$
Bell	486	36.8	$1.19 \cdot 10^6$
P^5	1,681	77.0	$2.53 \cdot 10^6$
Argyris	694	41.0	$6.27 \cdot 10^5$

(a) Laplace operator, P_c^k denotes statically condensed P^k .

(b) Biharmonic operator.

Table 1. Sparsity and conditioning information for H^1 - and H^2 -conforming discretizations of the Laplace and biharmonic operators on a coarse 8×8 mesh.

For the biharmonic operator, we have seen several factors (lower local FLOP count, smaller system, lower bandwidth) that should contribute to making sparse direct methods far more efficient for the H^2 discretizations than for interior penalty methods. Fig. 15a confirms this hypothesis, with solution of the Argyris element typically being cheaper than even the P^3 interior penalty method, not to mention P^5 .

4.4 Some advanced examples

4.4.1 The Cahn-Hilliard equation. In this section, we demonstrate the use of these elements on further problems. First, we consider the two-dimensional Cahn-Hilliard equation, a model of phase separation in binary fluids. Here, the chemical concentration c satisfies the fourth-order time-dependent equation

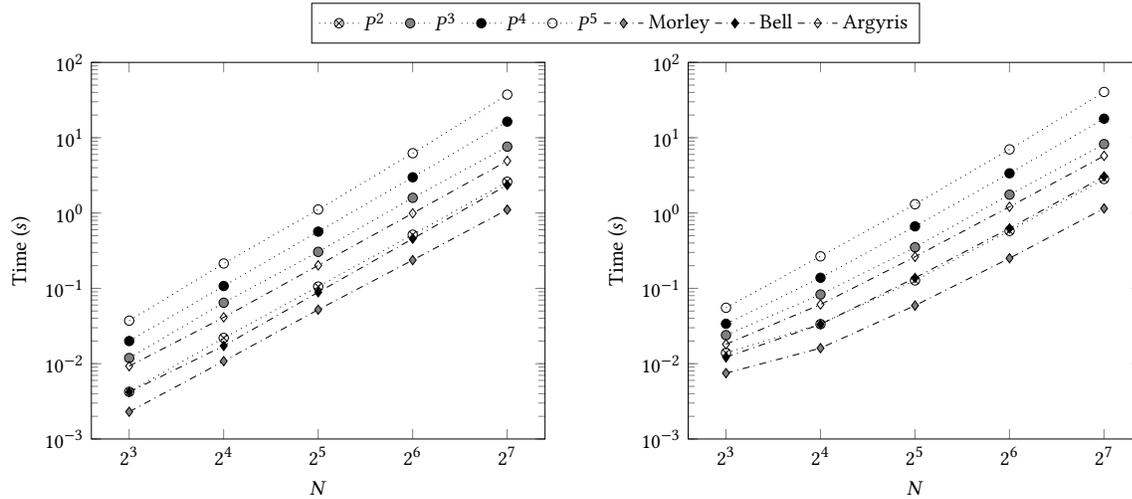
$$\frac{\partial c}{\partial t} - \nabla \cdot M \left(\nabla \left(\frac{df}{dc} - \lambda \nabla^2 c \right) \right) = 0 \quad \text{in } \Omega. \quad (4.7)$$

f is some typically non-convex function (in our case, we take $f(c) = 100c^2(1-c)^2$, and λ and M are scalar parameters controlling rates. Although $M = M(c)$ (the so-called degenerate mobility case) is possible, we have considered just constant M in our examples. The system is closed with the boundary conditions

$$M \left(\nabla \left(\frac{df}{dc} - \lambda \nabla^2 c \right) \right) \cdot n = 0 \quad \text{on } \partial\Omega, \quad (4.8)$$

$$M\lambda \nabla c \cdot n = 0 \quad \text{on } \partial\Omega. \quad (4.9)$$

The Cahn-Hilliard equation is often re-written into a system of two second-order equations [Barrett and Blowey 1999]. This eliminates the need for a C^1 discretization, but it also increases the number of unknowns and introduces a saddle point into the linear system. Alternatively, one maintains a primal form via an interior penalty method [Wells



(a) Time to factor and solve the sparse linear system for the discrete biharmonic operator on perturbed $N \times N$ mesh, including one iteration of iterative refinement.

(b) Total solver time for the biharmonic equation on $N \times N$ mesh. This includes element integration, sparse matrix preallocation and assembly, and factorization.

Fig. 15. Time to solution for the biharmonic equation. We see that the smaller, sparser H^2 discretizations give a significant advantage over the equivalent H^1 interior penalty scheme.

et al. 2006], although we have seen above that this leads to larger and less favorable linear systems for fourth-order operators.

In our simulations, we use a 16×16 mesh, but with higher-order elements, this leads to reasonable resolution. In our example, we use Bell elements, but it is simple to change to Argyris or the lower-order nonconforming Morley element – one merely changes the requested element in the `FunctionSpace` constructor. The complete code for simulating this problem, with the exception of output routines, is shown in Fig. 16. Using Crank-Nicolson time stepping with $\Delta t = 5 \times 10^{-6}$, we obtain the final state depicted in Fig. 17b. We remark that visualization for higher-order polynomials is performed by interpolating the solution onto piecewise linears on a refined mesh.

4.4.2 Chladni plates. As a final example, we consider the Chladni plate problem, which dates back to the eighteenth century. Chladni, a musician and physicist, discovered beautiful patterns appearing when a metal plate covered with dust or sand was excited with a violin bow. These patterns, now called *Chladni figures*, changed dramatically as a function of pitch. After work by Germain [1821, 1826], Kirchoff [1850] showed Chladni's patterns were in fact eigenpairs of the biharmonic operator under free boundary conditions and was able to give a solution on circular plates where symmetries offer simplification. Realizing this connection and effectively computing the figures in other geometries were historically different matters, and it was not until 1909 when Ritz [1909] was able to give the first computation of Chladni figures for square plates. For many more details, a beautiful historical overview of the computation of Chladni figures is given by Gander and Wanner [2012].

To compute the Chladni figures, we use Firedrake to assemble the discrete biharmonic eq. (4.5) and mass matrices using Argyris elements on a 64×64 mesh of the square domain $\Omega = [-1, 1]^2$. These are then fed into SLEPc [Hernandez et al. 2005]. The eigenpairs of the generalized eigenproblem are then constructed via a shift-and-invert strategy (the

```

from firedrake import *
import numpy
mesh = UnitSquareMesh(16, 16)
# Set parameters
lmbda = 1e-2
delta_t = 5e-6
theta = 0.5
M = 1
beta = 250
# Pick function space
V = FunctionSpace(mesh, "Bell", 5)
c = Function(V)
c0 = Function(V)
# Initial conditions
c0.vector()[::6] = 0.63 + 0.2*(0.5 - numpy.random.random(c0.vector().local_size() // 6))
c.assign(c0)

c_theta = theta*c + (1 - theta)*c0
dfdc = 200*(c_theta*(1 - c_theta)**2 - c_theta**2*(1 - c_theta))
n = FacetNormal(mesh)
h = CellSize(mesh)
# Nonlinear residual
v = TestFunction(V)
F = (inner(c - c0, v)*dx +
     delta_t*(inner(M*grad(dfdc), grad(v))*dx +
              inner(M*lmbda*div(grad(c_theta)), div(grad(v)))*dx -
              inner(M*lmbda*div(grad(c_theta)), dot(grad(v), n))*ds -
              inner(M*lmbda*dot(grad(c_theta), n), div(grad(v)))*ds +
              inner((beta/h)*M*lmbda*dot(grad(c_theta), n), dot(grad(v), n))*ds))
problem = NonlinearVariationalProblem(F, c)
solver = NonlinearVariationalSolver(problem, solver_parameters={"ksp_type": "preonly",
                                                             "pc_type": "lu"})

t = 0
T = 0.0025
while t < T:
    solver.solve()
    c0.assign(c)
    t += delta_t

```

Fig. 16. Code for solving the Cahn-Hilliard equation with Bell elements.

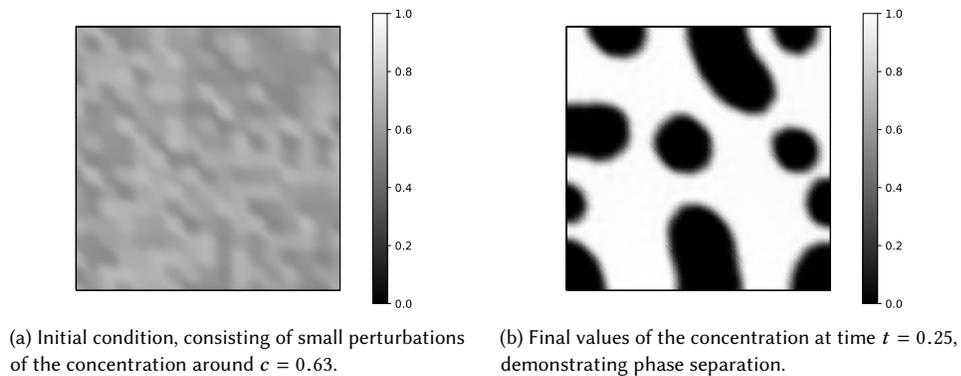


Fig. 17. Snapshots of the concentration field for the Cahn-Hilliard simulation, using the code of Fig. 16.

options are `-eps_nev 30 -eps_tol 1e-11 -eps_target 10 -st_type sinvert`). We plot the zero contours of a number of eigenmodes in Fig. 18.

While our extensions now enable succinct solution of a challenging classical problem, Firedrake also enables us to perform similar analysis in more complex geometric settings beyond the scope of analytic techniques. For example, consider the guitar-shaped domain illustrated in Fig. 19. We again compute eigenmodes of the plate-bending

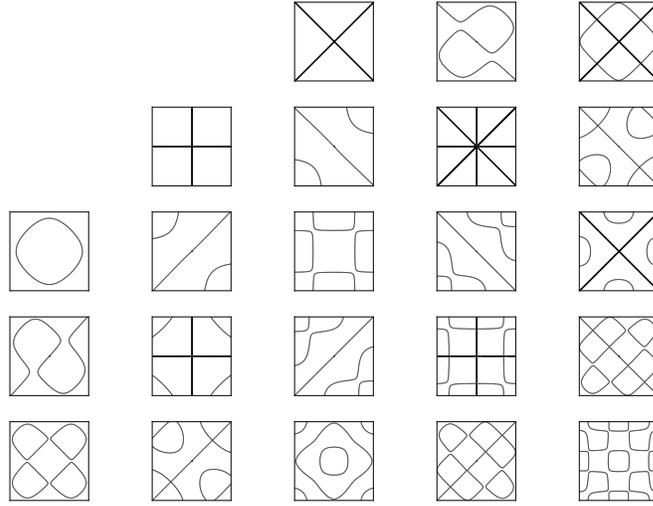


Fig. 18. Several Chladni figures computed with Argyris elements on a 64×64 mesh. These are a subset of those computed in [Gander and Wanner \[2012\]](#). The empty top-left figures correspond to the null modes of the operator under free boundary conditions. Figures for repeated eigenvalues may differ from those presented in other works since there is not a unique orthonormal basis for the eigenspace.

operator eq. (4.5) on this domain using Morley triangles on a mesh of 17277 vertices and 34536 elements, generated with gmsh [\[Geuzaine and Remacle 2009\]](#). In this case, we only leave the boundary free on the sound hole, and use clamped conditions on the exterior. We remark that studying eigenvalue computation with Morley elements is well-established [\[Rannacher 1979\]](#) but still receiving research attention [\[Gallistl 2014; Zhang et al. 2018\]](#). The additional Nitsche terms are more involved than in the case of eq. (4.2), and the full bilinear form is

$$\begin{aligned}
 a_h(u, v) = & \int_{\Omega} \Delta u \Delta v - (1 - \nu) (2u_{xx}v_{yy} + 2u_{yy}v_{xx} - 4u_{xy}v_{xy}) \, dx \\
 & + \frac{\beta_1}{h^2} \int_{\partial\Omega} uv \, ds + \frac{\beta_2}{h} \int_{\partial\Omega} \Delta u \Delta v \, ds \\
 & + \int_{\partial\Omega} ((\Delta u)_n - 2(1 - \nu)u_{ntt}) v \, ds + \int_{\partial\Omega} u ((\Delta v)_n - 2(1 - \nu)v_{ntt}) \, ds \\
 & + \int_{\partial\Omega} (\Delta u - 2(1 - \nu)u_{tt}) v_n \, ds + \int_{\partial\Omega} u_n (\Delta v - 2(1 - \nu)v_{tt}) \, ds,
 \end{aligned} \tag{4.10}$$

where h is again a suitable measure of the mesh cell size, β_1 and β_2 are positive constants, and $\bullet_n := (\nabla \bullet) \cdot n$, where n is the outward pointing unit normal on a facet, similarly for \bullet_t , only using the tangent to the facet, obtained by a counter-clockwise rotation of n . The first twenty-eight eigenmodes of the operator eq. (4.10) are shown Fig. 19. Of course, this is a very crude approximation of the actual eigenmodes supported by the top plate of a guitar. To guide the design of an instrument we would, at the very least, need to incorporate the anisotropic properties of wood into the model, as well as thinking about structural supports such as bass bars and ribbing. Some of these modelling issues are discussed, in the context of violins, in [Gough \[2007\]](#), and a simple multiphysics guitar model in [Tapia \[2002\]](#). Here, however, we leave more involved simulations as future work.

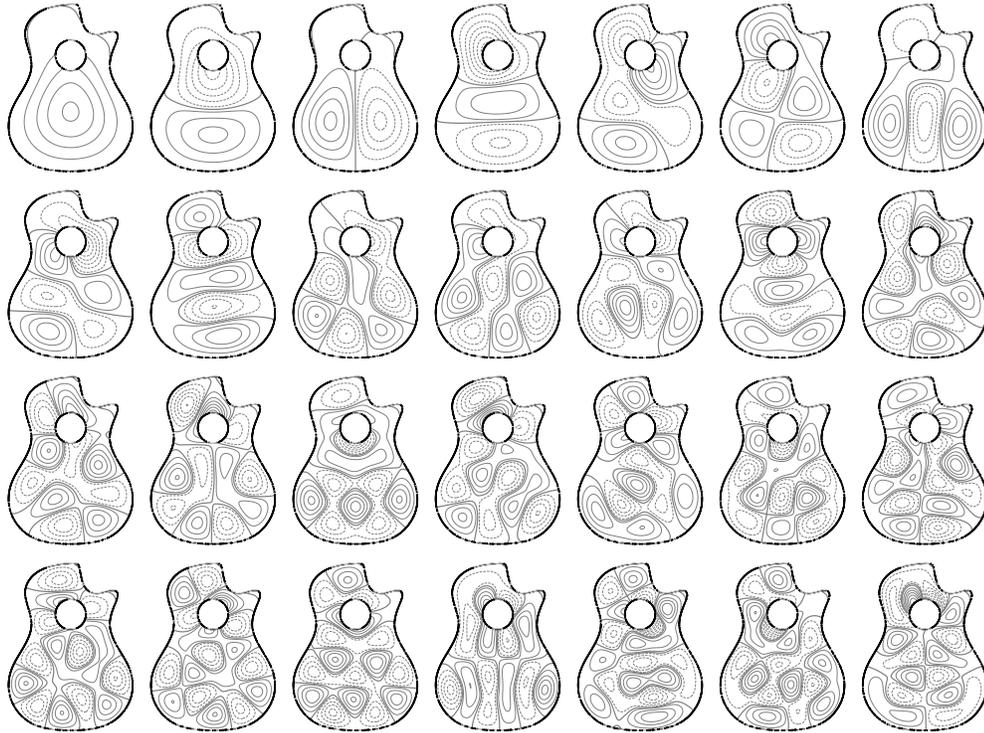


Fig. 19. The first twenty-eight eigenmodes, arranged in order of increasing frequency, of the biharmonic operator eq. (4.10) with clamped boundary conditions on the exterior boundary of the domain, and free conditions on the sound hole. Contour lines are equally spaced, with negative contours dashed and the zero-contour slightly thicker.

5 CONCLUSIONS AND FUTURE WORK

The key idea of FInAT, that an element library can provide structure and not just numerical values, has made significant progress possible within the Firedrake code stack. While earlier work on FInAT focused on vector and tensor product structure, our current work has enabled a wide range of useful finite elements previously inaccessible to automated code generation systems. Our numerical results indicate that these elements are indeed viable for many interesting problems. Automation now makes Argyris and other “difficult” finite elements of comparable cost to deploy as any Lagrange elements.

In the future, we hope to extend this work in many directions. In particular, we need to extend the transformations and hence code infrastructure to support embedded manifolds [Rognes et al. 2013]. Also, we believe that the extensions we have developed in FInAT will provide a starting point to consider macro-element techniques for splines and other more complex elements [Lai and Schumaker 2007].

CODE AVAILABILITY

For reproducibility, we cite archives of the exact software versions used to produce the results in this paper. All major Firedrake components have been archived on Zenodo [zenodo/Firedrake-20190830.0 2019]. This record collates DOIs for the components, as well as containing the data and scripts used to produce the results in this paper. An installation

of Firedrake with components matching those used to produce the results can be obtained following the instructions at www.firedrakeproject.org/download.html with

```
export PETSC_CONFIGURE_OPTIONS="--download-pastix --download-ptscotch"
python3 firedrake-install --doi 10.5281/zenodo.3381901 --slepc --install \
  git+https://github.com/thomasgibson/scpc.git@3a1173ebb3610dcdf5090088294a542274bbb73a
```

A README file in the archived record contains more detailed information.

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/L000407/1]; and the National Science Foundation, award number 1525697. Figs. 2 to 6 are adapted from Kirby [2018], licensed under CC-BY-NC-ND.

REFERENCES

- Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. 2014. Unified Form Language: a domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Software* 40, 2 (2014), 9:1–9:37. <https://doi.org/10.1145/2566630> arXiv:1211.4047
- Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. 2000. MUMPS: a general purpose distributed memory sparse solver. In *International Workshop on Applied Parallel Computing*. Springer, 121–130.
- J. H. Argyris, I. Fried, and D. W. Scharpf. 1968. The TUBA family of plate elements for the matrix displacement method. *Aeronautical Journal* 72 (1968), 701–709. <https://doi.org/10.1017/S000192400008489X>
- Marino Arroyo and Michael Ortiz. 2006. Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. *International journal for numerical methods in engineering* 65, 13 (2006), 2167–2202.
- Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2018. *PETSc Users Manual*. Technical Report ANL-95/11 - Revision 3.9. Argonne National Laboratory.
- Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. 1997. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen (Eds.). Birkhäuser Press, 163–202. https://doi.org/10.1007/978-1-4612-1986-6_8
- Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. 2007. deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Trans. Math. Software* 33, 4 (2007), 24:1–24:27. <https://doi.org/10.1145/1268776.1268779>
- John Barrett and James Blowey. 1999. Finite element approximation of the Cahn-Hilliard equation with concentration dependent mobility. *Math. Comp.* 68, 226 (1999), 487–517. <https://doi.org/10.1090/S0025-5718-99-01015-7>
- Kolbein Bell. 1969. A refined triangular plate bending finite element. *Internat. J. Numer. Methods Engrg.* 1, 1 (1969), 101–122. <https://doi.org/10.1002/nme.1620010108>
- James H. Bramble and Xuejun Zhang. 1995. Multigrid methods for the biharmonic problem discretized by conforming C^1 finite elements on nonnested meshes. *Numerical Functional Analysis and Optimization* 16, 7-8 (1995), 835–846. <https://doi.org/10.1080/01630569508816649>
- Susanne C. Brenner. 1989. An optimal-order nonconforming multigrid method for the biharmonic equation. *SIAM J. Numer. Anal.* 26, 5 (1989), 1124–1138. <https://doi.org/10.1137/0726062>
- Susanne C. Brenner and L. Ridgway Scott. 2008. *The mathematical theory of finite element methods* (third ed.). Texts in Applied Mathematics, Vol. 15. Springer, New York.
- Philippe G. Ciarlet and Pierre-Arnaud Raviart. 1972. General Lagrange and Hermite interpolation in \mathbb{R}^n with applications to finite element methods. *Archive for Rational Mechanics and Analysis* 46, 3 (1972), 177–199. <https://doi.org/10.1007/BF00252458>
- Victor Domínguez and Francisco-Javier Sayas. 2008. Algorithm 884: A simple Matlab implementation of the Argyris element. *ACM Trans. Math. Software* 35, 2 (2008), 16. <https://doi.org/10.1145/1377612.1377620>
- Ricardo Durán and Elsa Liberman. 1992. On mixed finite element methods for the Reissner-Mindlin plate model. *Mathematics of computation* 58, 198 (1992), 561–573.
- Anand Embar, John Dolbow, and Isaac Harari. 2010. Imposing Dirichlet boundary conditions with Nitsche's method and spline-based finite elements. *Internat. J. Numer. Methods Engrg.* 83, 7 (2010), 877–898. <https://doi.org/10.1002/nme.2863>
- G. Engel, K. Garikipati, T. J. R. Hughes, M. G. Larson, L. Mazzei, and R. L. Taylor. 2002. Continuous/discontinuous finite element approximations of fourth-order elliptic problems in structural and continuum mechanics with applications to thin beams and plates, and strain gradient elasticity. *Computer Methods in Applied Mechanics and Engineering* 191, 34 (2002), 3669–3750. [https://doi.org/10.1016/S0045-7825\(02\)00286-4](https://doi.org/10.1016/S0045-7825(02)00286-4)

- Michel Fournié, Nicolas Renon, Yves Renard, and Daniel Ruiz. 2010. CFD parallel simulation using GetFEM++ and MUMPS. In *European Conference on Parallel Processing*. Springer, 77–88. https://doi.org/10.1007/978-3-642-15291-7_9
- Dietmar Gallistl. 2014. Morley finite element method for the eigenvalues of the biharmonic operator. *IMA J. Numer. Anal.* 35, 4 (2014), 1779–1811. <https://doi.org/10.1093/imanum/dru054>
- Martin J. Gander and Gerhard Wanner. 2012. From Euler, Ritz, and Galerkin to Modern Computing. *SIAM Rev.* 54, 4 (2012), 627–666. <https://doi.org/10.1137/100804036>
- Sophie Germain. 1821. *Recherches sur la théorie des surfaces élastiques*. V. Courcier.
- Sophie Germain. 1826. *Remarques sur la nature, les bornes et l'étendue de la question des surfaces élastiques, et équation générale de ces surfaces*. V. Courcier.
- Christophe Geuzaine and Jean-François Remacle. 2009. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331. <https://doi.org/10.1002/nme.2579>
- Thomas H. Gibson, Lawrence Mitchell, David A. Ham, and Colin J Cotter. 2018. A domain-specific language for the hybridization and static condensation of finite element methods. arXiv:1802.00303
- C. Gough. 2007. The violin: Chladni patterns, plates, shells, and sounds. *The European Physical Journal Special Topics* 145, 1 (2007), 77–101. <https://doi.org/10.1140/epjst/e2007-00149-0>
- Jack S. Hale, Matteo Brunetti, Stéphane P. A. Bordas, and Corrado Maurini. 2018. Simple and extensible plate and shell finite element models through automatic code generation tools. *Computers & Structures* 209 (2018), 163–181.
- P. Hénon, P. Ramet, and J. Roman. 2002. PaStiX: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Comput.* 28, 2 (2002), 301–321. [https://doi.org/10.1016/S0167-8191\(01\)00141-7](https://doi.org/10.1016/S0167-8191(01)00141-7)
- Vicente Hernandez, Jose E. Roman, and Vicente Vidal. 2005. SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software* 31, 3 (2005), 351–362. <https://doi.org/10.1145/1089014.1089019>
- Miklós Homolya, Robert C. Kirby, and David A. Ham. 2017. Exposing and exploiting structure: optimal code generation for high-order finite element methods. arXiv:1711.02473
- Miklós Homolya, Lawrence Mitchell, Fabio Luporini, and David A. Ham. 2018. TSFC: a structure-preserving form compiler. *SIAM Journal on Scientific Computing* 40, 3 (2018), C401–C428. <https://doi.org/10.1137/17M1130642>
- Thomas J. R. Hughes, John A. Cottrell, and Yuri Bazilevs. 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering* 194, 39–41 (2005), 4135–4195.
- David Joyner, Ondřej Čertík, Aaron Meurer, and Brian E. Granger. 2012. Open source computer algebra systems: SymPy. *ACM Communications in Computer Algebra* 45, 3/4 (2012), 225–234. <https://doi.org/10.1145/2110170.2110185>
- Robert C. Kirby. 2004. Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Software* 30, 4 (2004), 502–516. <https://doi.org/10.1145/1039813.1039820>
- Robert C. Kirby. 2018. A general approach to transforming finite elements. *SMAI Journal of Computational Mathematics* 4 (2018), 197–224. <https://doi.org/10.5802/smai-jcm.33>
- Robert C. Kirby, Matthew G. Knepley, Anders Logg, and L. Ridgway Scott. 2005. Optimizing the evaluation of finite element matrices. *SIAM Journal on Scientific Computing* 27, 3 (2005), 741–758. <https://doi.org/10.1137/040607824>
- Robert C. Kirby and Anders Logg. 2006. A compiler for variational forms. *ACM Trans. Math. Software* 32, 3 (2006), 417–444. <https://doi.org/10.1145/1163641.1163644> arXiv:1112.0402
- Robert C. Kirby, Anders Logg, L. Ridgway Scott, and Andy R. Terrel. 2006. Topological optimization of the evaluation of finite element matrices. *SIAM Journal on Scientific Computing* 28, 1 (2006), 224–240. <https://doi.org/10.1137/050635547>
- Robert C. Kirby and Lawrence Mitchell. 2018. Solver composition across the PDE/linear algebra barrier. *SIAM Journal on Scientific Computing* 40, 1 (2018), C76–C98. <https://doi.org/10.1137/17M1133208>
- Gustav Kirchoff. 1850. Über das Gleichgewicht und die Bewegung einer elastischen Scheibe. *Journal für die reine und angewandte Mathematik* 40 (1850), 51–88. <https://doi.org/10.1515/crll.1850.40.51>
- Ming-Jun Lai and Larry L. Schumaker. 2007. *Spline functions on triangulations*. Encyclopedia of Mathematics and its Applications, Vol. 110. Cambridge University Press, Cambridge.
- Lizao Li. 2018. *Regge finite elements with applications in solid mechanics and relativity*. Ph.D. Dissertation. University of Minnesota.
- Anders Logg, Kent-Andre Mardal, and Garth N. Wells (Eds.). 2012. *Automated solution of differential equations by the finite element method: the FEniCS book*. Vol. 84. Springer. <https://doi.org/10.1007/978-3-642-23099-8>
- Kevin R. Long. 2003. Sundance rapid prototyping tool for parallel PDE optimization. In *Large-Scale PDE-Constrained Optimization*, Lorenz T. Biegler, Matthias Heinkenschloss, Omar Ghattas, and Bart van Bloemen Waanders (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 331–341. https://doi.org/10.1007/978-3-642-55508-4_20
- Fabio Luporini, David A. Ham, and Paul H. J. Kelly. 2017. An algorithm for the optimization of finite element integration loops. *ACM Trans. Math. Software* 44, 1 (2017), 3:1–3:26. <https://doi.org/10.1145/3054944> arXiv:1604.05872
- John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture Newsletter* (1995), 19–25.
- L. S. D. Morley. 1971. The constant-moment plate-bending element. *The Journal of Strain Analysis for Engineering Design* 6, 1 (1971), 20–24. <https://doi.org/10.1243/03093247V061020>

- Jean-Claude Nédélec. 1980. Mixed finite elements in \mathbb{R}^3 . *Numer. Math.* 35, 3 (1980), 315–341. <https://doi.org/10.1007/BF01396415>
- Joachim Nitsche. 1971. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 36 (1971), 9–15.
- Anna Perduta. 2013. Enhancing MESH adaptation capabilities of GetFEM++ “FEM engine” with MAdLib library. *Mechanics and Control* 32, 4 (2013), 143. <https://doi.org/10.7494/mech.2013.32.4.143>
- Christophe Prud’Homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samake, and Gonçalo Pena. 2012. Feel++: A computational framework for galerkin methods and advanced numerical methods. In *ESAIM: Proceedings*, Vol. 38. EDP Sciences, 429–455.
- Rolf Rannacher. 1979. Nonconforming finite element methods for eigenvalue problems in linear plate theory. *Numer. Math.* 33, 1 (1979), 23–42. <https://doi.org/10.1007/BF01396493>
- Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. 2016. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Software* 43, 3 (2016), 24:1–24:27. <https://doi.org/10.1145/2998441> arXiv:1501.01809
- Pierre-Arnaud Raviart and Jean-Marie Thomas. 1977. A mixed finite element method for 2nd order elliptic problems. In *Mathematical aspects of finite element methods*. 292–315. <https://doi.org/10.1007/BFb0064470>
- Walter Ritz. 1909. Theorie der Transversalschwingungen einer quadratischen Platte mit freien Rändern. *Annalen der Physik* 333, 4 (1909), 737–786. <https://doi.org/10.1002/andp.19093330403>
- Marie E. Rognes, David A. Ham, Colin J. Cotter, and Andrew T. T. McRae. 2013. Automating the solution of PDEs on the sphere and other manifolds in FEniCS 1.2. *Geoscientific Model Development* 6, 6 (2013), 2099–2119. <https://doi.org/10.5194/gmd-6-2099-2013>
- Eduardo Rodrigo Gonzalez Tapia. 2002. *Simulating a Classical Acoustic Guitar, Finite Elements and Multiphysics Modelling*. Ph.D. Dissertation. KTH Royal Institute of Technology.
- Ming Wang and Jinchao Xu. 2013. Minimal finite element spaces for $2m^{\text{th}}$ -order partial differential equations in \mathbb{R}^n . *Math. Comp.* 82, 281 (2013), 25–43.
- Garth N. Wells, Krishna Garikipati, and Luisa Molari. 2004. A discontinuous Galerkin formulation for a strain gradient-dependent damage model. *Computer Methods in Applied Mechanics and Engineering* 193, 33–35 (2004), 3633–3645. <https://doi.org/10.1016/j.cma.2004.01.020>
- Garth N. Wells, Ellen Kuhl, and Krishna Garikipati. 2006. A discontinuous Galerkin method for the Cahn–Hilliard equation. *J. Comput. Phys.* 218, 2 (2006), 860–877. <https://doi.org/10.1016/j.jcp.2006.03.010>
- Jinchao Xu. 1996. The auxiliary space method and optimal multigrid preconditioning techniques for unstructured grids. *Computing* 56, 3 (1996), 215–235. <https://doi.org/10.1007/BF02238513>
- zenodo/Firedrake-20190830.0 2019. Software used in ‘Code generation for generally mapped finite elements’. <https://doi.org/10.5281/zenodo.3381901>
- Shuo Zhang, Yingxia Xi, and Xia Ji. 2018. A multi-level mixed element method for the eigenvalue problem of biharmonic equation. *Journal of Scientific Computing* 75 (2018), 1415–1444. <https://doi.org/10.1007/s10915-017-0592-7>