

# Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake

JUSTIN CRUM, University of Arizona, USA  
 CYRUS CHENG, Imperial College London, UK  
 DAVID A. HAM, Imperial College London, UK  
 LAWRENCE MITCHELL, Durham University, UK  
 ROBERT C. KIRBY, Baylor University, USA  
 JOSHUA A. LEVINE, University of Arizona, USA  
 ANDREW GILLETTE, University of Arizona, USA

We present an implementation of the trimmed serendipity finite element family, using the open source finite element package Firedrake. The new elements can be used seamlessly within the software suite for problems requiring  $H^1$ ,  $H(\text{curl})$ , or  $H(\text{div})$ -conforming elements on meshes of squares or cubes. To test how well trimmed serendipity elements perform in comparison to traditional tensor product elements, we perform a sequence of numerical experiments including the primal Poisson, mixed Poisson, and Maxwell cavity eigenvalue problems. Overall, we find that the trimmed serendipity elements converge, as expected, at the same rate as the respective tensor product elements while being able to offer significant savings in the time or memory required to solve certain problems.

## 1 INTRODUCTION

In addition to the four families of finite elements present on the Periodic Table of Finite Elements [Arnold and Logg 2014], recent research has examined a fifth family, called the *trimmed serendipity finite elements*. Similar to their tensor product and “regular” serendipity counterparts, trimmed serendipity elements provide another finite element method for approximating the solution to partial differential equations on square or cubical meshes. While potential computational advantages of these new elements have been described in previous works, they have never been realized or tested in a modern finite element package. In particular, the serendipity and trimmed serendipity elements should attain the same rate of convergence as tensor product elements while requiring fewer degrees of freedom (DOFs) to do so. By implementing the trimmed serendipity and tensor product elements in the same software package, we seek to make these elements available to the broader finite element community while assessing their computational advantages in a common and practical setting for the first time.

In this paper, we present a complete implementation of the trimmed serendipity element family in 2D and 3D within the Firedrake finite element package [Rathgeber et al. 2016], accompanied by a suite of numerical experiments demonstrating their approximation accuracy and potential benefits on benchmark partial differential equation problems. Firedrake is a Python package for computing the solution to PDEs by using the finite element method. By leveraging the Unified Form Language [Alnæs et al. 2014; Logg et al. 2012], it is able to provide a flexible and high level interface for the solution of PDEs that is reminiscent of their mathematical formulation.

---

Authors’ addresses: Justin Crum, jcrum@math.arizona.edu, University of Arizona, Department of Mathematics, Tucson, AZ, USA; Cyrus Cheng, cyrus.cheng15@alumni.imperial.ac.uk, Imperial College London, Department of Mathematics, London, SW7 2AZ, UK; David A. Ham, Imperial College London, Department of Mathematics, London, SW7 2AZ, UK, david.ham@imperial.ac.uk; Lawrence Mitchell, Durham University, Department of Computer Science, Upper Mountjoy, Durham, DH1 3LE, UK, lawrence.mitchell@durham.ac.uk; Robert C. Kirby, Baylor University, Department of Mathematics, 1410 S. 4th Street, Waco, TX, USA, robert\_kirby@baylor.edu; Joshua A. Levine, University of Arizona, Department of Computer Science, Tucson, AZ, USA, josh@email.arizona.edu; Andrew Gillette, University of Arizona, Department of Mathematics, Tucson, AZ, USA, agillette@math.arizona.edu.

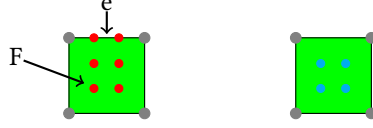


Fig. 1. The degree 2 RCTF tensor product element [Raviart and Thomas 1977] (left) used in Listing 1 where we indicate DOFs on the face and one edge as an example. The RCTF element is an example of a tensor product element in 2D, which depending upon the orientation of the DOFs on the edges, can be used for either  $H(\text{div})$  or  $H(\text{curl})$  problems. The right displays a similar example for the DQ element, which is an  $L^2$ -conforming tensor product element in 2D, and is necessary to form the stable pair for the mixed Poisson problem.

An example of writing a discretized PDE and computing its approximate solution in Firedrake can be found in Listing 1. To do this, we first write the mixed Poisson equation on the domain  $\Omega := [0, 1] \times [0, 1]$  with boundary  $\Gamma$  in the (continuous) weak formulation. We assume homogeneous Dirichlet boundary conditions so that the integral over  $\Gamma$  is 0. The problem statement is then: find  $\sigma \in \Sigma := H(\text{div})$  and  $u \in V := L^2$  such that

$$\begin{aligned} \int_{\Omega} (\sigma \cdot \tau + \nabla \cdot \tau u) \, dx &= \int_{\Gamma} \tau \cdot nu \, dx \quad \forall \tau \in \Sigma, \\ \int_{\Omega} \nabla \cdot \sigma v \, dx &= - \int_{\Omega} f v \, dx \quad \forall v \in V. \end{aligned} \quad (1)$$

Solving the discretized version of this requires choosing a suitable pair of finite element spaces to create a stable method. On a mesh of squares, the typical stable pair of finite elements would be RCTF and DQ for the  $H(\text{div})$  and  $L^2$  elements as in Fig. 1. Later we will use the trimmed serendipity pair  $S_{\text{minusDiv}}$  and DPC as an alternative stable pair. We demonstrate this tensor product pairing in Listing 1, where the order of the vector and scalar elements are offset by 1 in accordance with the theory for optimal convergence rates.

Listing 1. Basic Firedrake implementation of the mixed Poisson problem showcasing where to choose the elements that are used and how to create the equations in Firedrake's notation.

---

```

1  polyDegree = 2
2  numberOfCells = 2**5
3  mesh = UnitSquareMesh(numberOfCells, numberOfCells, quadrilateral=True)
4  hDivSpace = FunctionSpace(mesh, "RCTF", polyDegree)
5  l2Space = FunctionSpace(mesh, "DQ", polyDegree - 1)
6  mixedSpace = hDivSpace * l2Space
7
8  sigma, u = TrialFunctions(mixedSpace)
9  tau, v = TestFunctions(mixedSpace)
10
11 x, y = SpatialCoordinate(mesh)
12 uex = sin(pi*x)*sin(pi*y)
13
14 f = -div(grad(uex))
15 a = (dot(sigma, tau) + div(tau)*u + div(sigma)*v)*dx
16 l = -f*v*dx
17 w = Function(mixedSpace)
18 solve(a == l, w)

```

---

An important strength of Firedrake is its modular structure for both users and developers. For the user, swapping to trimmed serendipity elements to solve the mixed Poisson problem is now only a matter of modifying lines 4–5 in Listing 1 to the appropriate identifiers, `SminusDiv` and `DPC` inside the `FunctionSpace` calls that define `hDivSpace` and `l2Space`. For developers, implementing a new element type – such as trimmed serendipity – is simply a matter of defining a suitable computational basis and connecting it to the intermediate interfaces in the included libraries. We implement the basis functions from Gillette et al. [2019] (that are already associated to portions of the square or cube geometry) within FIAT, and then Firedrake’s form compiler generates the code for the variational forms.

Having these available in Firedrake allows users to test the theoretical convergence rates and approximation power that the trimmed serendipity elements are trying to attain. At low orders in both 2D and 3D, the trimmed serendipity elements have slightly fewer DOFs than tensor product elements. However as the degree grows, the gap in DOFs between these two types of elements becomes more significant.

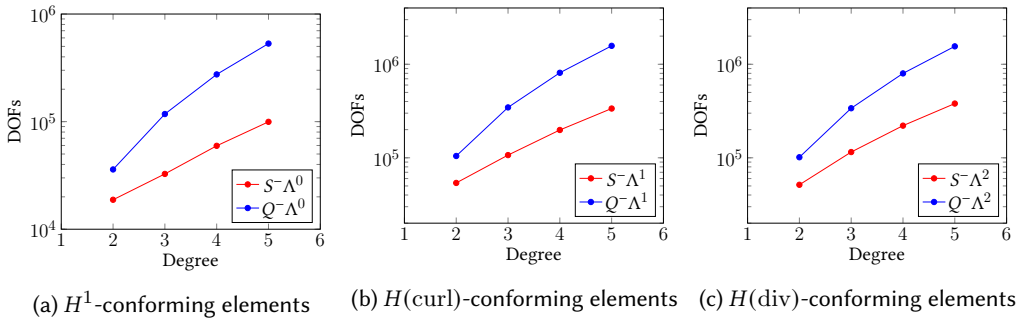


Fig. 2. Comparison of the DOFs required for a trimmed serendipity element vs a tensor product element. The DOFs were calculated on a  $[0, 1]^3$  mesh with a total of  $16^3$  cubes. The red trendlines represent the trimmed serendipity elements while the blue trendlines represent the tensor product elements.

While the formulae for the dimensions of each space are somewhat involved, a simple count of DOFs in a standard case highlights the potential gains. To implement a standard mixed method for Darcy flow with quadratic order error decay in the velocity estimate, the divergence of the velocity and the pressure, on a single cube the pair of tensor product elements used have a total of 44 DOFs. For the same setup, the appropriate pair of regular serendipity elements have 43 DOFs, but the pair of trimmed serendipity elements have only 25 DOFs. This reduces the number of DOFs by 42% (from tensor product to trimmed serendipity), with no theoretical loss in the order of accuracy.

Further, the number of DOFs shared between adjacent faces in this context is reduced by 25% – from 4 per face to 3 per face – which has a dramatic impact on the global degrees of freedom for the problem. Such reductions increase with the approximation degree, meaning the computational savings accrued by trimmed serendipity should be most dramatic at large degree. This pattern is seen in Fig. 2, where the gap in DOFs is larger at order  $r = 5$  than at order  $r = 2$ . Seeing how such back-of-the-envelope calculations might bear out in a practical context requires the thorough implementation provided in this paper.

Finally, we are interested in different ways to compare tensor product and trimmed serendipity elements. The first way we discuss is a direct comparison between tensor product and trimmed serendipity elements of the same order. This will yield a comparison where the DOFs required for trimmed serendipity are lower than the DOFs required for tensor product elements, the rates of

convergence are the same, but the constant for trimmed serendipity elements is worse. The second way we do this analysis is to focus on trimmed serendipity elements of one order higher than the tensor product elements. This compares elements with similar numbers of DOFs, but the trimmed serendipity method has a higher rate of convergence.

Summarizing, the contributions of this paper are that we:

- (1) show how the Firedrake software environment can be used to implement trimmed serendipity elements and allow a user to work with these elements with only small changes to their existing code,
- (2) illustrate trimmed serendipity finite elements achieving the theoretical bounds that are predicted in terms of convergence rates, and
- (3) examine the costs and benefits of using trimmed serendipity elements within the confines of typical test problems for numerical analysis by comparing them to tensor product elements.

## 2 BACKGROUND AND NOTATION FOR TRIMMED SERENDIPITY ELEMENTS

Over the last 15 years, conforming finite element methods for Hodge-Laplace type problems on simplicial and cubical meshes have been categorized using the language of Finite Element Exterior Calculus (FEEC) [Arnold et al. 2015, 2006, 2010]. Among its many advantages, FEEC gives an easy, unified way to notate different element types. The four best known families of elements are denoted  $\mathcal{P}_r^- \Lambda^k$ ,  $\mathcal{P}_r \Lambda^k$ ,  $\mathcal{Q}_r^- \Lambda^k$ , and  $\mathcal{S}_r \Lambda^k$ , which are, respectively, the trimmed polynomial, polynomial, tensor product, and serendipity elements of order  $r$  using  $k$ -forms. The  $\mathcal{P}$  and  $\mathcal{P}^-$  spaces are defined over meshes of simplices (triangles, tetrahedra, etc) while the  $\mathcal{Q}^-$  and  $\mathcal{S}$  spaces are defined over meshes of hypercubes (squares, cubes, etc). The optional notation  $(\square_n)$  specifies that the space is constructed over the  $n$ -dimensional cube in  $\mathbb{R}^n$ , but we will frequently omit this addition if  $n$  is clear from context.

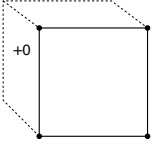
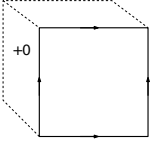
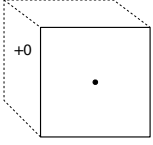
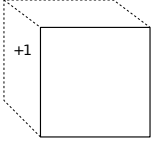
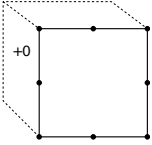
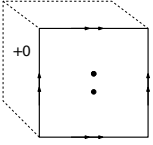
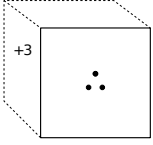
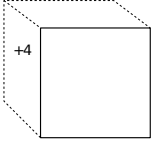
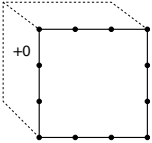
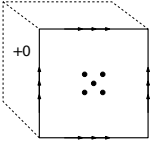
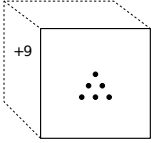
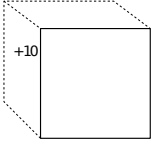
The mathematical results from FEEC regarding these four families synthesize decades of research into what constitutes a stable finite element, i.e. a numerical method that can be proven to converge to the correct solution of certain PDEs in certain norms at a prescribed rate, indicated by the subscript  $r$ . In any dimension, the 0-form spaces provide scalar-valued,  $H^1$ -conforming elements. In 2D, 1-forms can represent both  $H(\text{curl})$  and  $H(\text{div})$  elements, depending on the orientation of the DOFs defined on the edges of a mesh. In 3D, 1-forms correspond to  $H(\text{curl})$  elements while 2-forms correspond to  $H(\text{div})$  elements. Notably, the serendipity family  $\mathcal{S}_r \Lambda^k$  is the youngest, least implemented, and hence least understood among all these families. In particular, the 1-form and 2-form (regular) serendipity elements in 3D were only characterized in 2014 by Arnold and Awanou [2014], whereas the equivalent tensor product elements were first described more than 30 years earlier in Nédélec [1980, 1986].

### 2.1 Trimmed Serendipity Elements

The trimmed serendipity family is an even newer addition to this collection that attains a key optimality condition arising from the FEEC framework. Christiansen and Gillette [2016] computed the minimal possible dimensions for an exact sequence of conforming finite element spaces on cubes that contained  $\mathcal{P}_r \Lambda^k$  for each  $k$ . Gillette and Kloefkorn [2019] identified polynomial differential form spaces with these prescribed dimensions and approximation power, denoting them trimmed serendipity elements with the notation  $\mathcal{S}_r^- \Lambda^k$ . The trimmed serendipity spaces share the same interleaving containment structure with the serendipity spaces as the polynomial and trimmed polynomial subspaces have. That is, for any fixed dimension  $n$ , any form order  $0 \leq k \leq n$ , and any approximation order  $r \geq 1$ , we have both

$$\mathcal{S}_r \Lambda^k \subset \mathcal{S}_{r+1}^- \Lambda^k \subset \mathcal{S}_{r+1} \Lambda^k$$

Table 1. Trimmed serendipity elements on a reference cube in 3D, akin to the Periodic Table of the Finite Elements. Columns show increasing form order from  $k = 0$  to  $k = 3$ , corresponding to  $H^1$ ,  $H(\text{curl})$ ,  $H(\text{div})$ , and  $L^2$  conformity, respectively. Rows show increasing order of approximation  $r = 1, 2, 3$ . On the front face of each element, dots indicate the number of DOFs associated to each vertex, edge, or face of the cubical element. The number of DOFs associated to the interior of the cube is indicated with “+X.” The total degree of freedom count for each element is shown to its right.

	$\mathcal{S}_r^-\Lambda^0(\square_3)$	$\mathcal{S}_r^-\Lambda^1(\square_3)$	$\mathcal{S}_r^-\Lambda^2(\square_3)$	$\mathcal{S}_r^-\Lambda^3(\square_3)$
$r = 1$	 8	 12	 6	 1
$r = 2$	 20	 36	 21	 4
$r = 3$	 32	 66	 45	 10

and

$$\mathcal{P}_r\Lambda^k \subset \mathcal{P}_{r+1}^-\Lambda^k \subset \mathcal{P}_{r+1}\Lambda^k.$$

These containments express the fact that typically a finite element practitioner will get the most computational efficiency for a desired approximation order by choosing an appropriate  $\mathcal{P}^-$  space instead of a  $\mathcal{P}$  space. An equivalent claim is implied about serendipity and trimmed serendipity spaces, with additional comparison to the larger dimensioned tensor product spaces,  $\mathcal{Q}^-$ , possible as well.

As a step toward testing such ideas, Gillette, Kloefkorn and Sanders gave a systematic way to build the computational basis for each of these elements for dimensions  $n = 2, 3$ ,  $k = 0, 1, 2, 3$ -forms, and arbitrary order  $r \geq 1$  [Gillette et al. 2019]. These “computational bases” are well-suited for implementation since each basis function is associated to a unique mesh identity - i.e. a specific vertex, edge, face (for cubes) or element interior. The required geometric localization of DOFs is visualized for low orders in 3D in Table 1, arranged equivalently to the Periodic Table of the Finite Elements. We note that neither the particular bases defined in Gillette et al. [2019] nor any other general implementation of trimmed serendipity elements has been attempted prior to this paper.

**2.1.1 Scalar Trimmed Serendipity Elements.** The scalar-valued trimmed serendipity elements that are represented by 0-forms are used as the shape functions for an  $H^1$ -conforming finite element space. These are denoted by  $\mathcal{S}_r^-\Lambda^0$  and are identical to the scalar-valued serendipity elements from the Periodic Table of Finite Elements, i.e.  $\mathcal{S}_r^-\Lambda^0(\square_n) = \mathcal{S}_r\Lambda^0(\square_n)$  for any  $n$ . Arnold and Awanou

provided a simple description of the functions in  $S_r\Lambda^0$  as the span of all monomials of “superlinear degree” less than or equal to  $r$  [Arnold and Awanou 2011].

Likewise, the scalar-valued trimmed serendipity elements that are represented by  $n$ -forms create  $L^2$ -conforming finite element spaces. These are denoted by  $S_r^-\Lambda^n(\square_n)$ , and here we have the equality  $S_r^-\Lambda^n(\square_n) = S_{r+1}\Lambda^n(\square_n)$ . In terms of the Periodic Table of Finite Elements, these are the  $\mathbf{dPc}_r$  spaces. The shape functions for these spaces are simply the space of order  $r$  polynomials. Since no inter-element continuity is needed for  $L^2$ -conformity, we have the additional equivalence  $S_r^-\Lambda^n(\square_n) = \mathcal{P}_{r+1}\Lambda^n(\square_n)$ .

**2.1.2 Vector-valued Trimmed Serendipity Elements.** The trimmed serendipity elements are truly distinct from regular serendipity spaces for  $k$  values  $0 < k < n$ . Here, we will only consider dimensions  $n = 2$  and  $n = 3$ , where the  $k$ -form spaces can be identified as vector-valued finite elements. In 2D, the vector-valued spaces  $S_r^-\Lambda^1(\square_2)$  bear close relation to the Arbogast-Correa elements [Arbogast and Correa 2016], as explained in Gillette and Kloforn [2019, Prop 2.2]. In 3D, the vector valued spaces  $S_r^-\Lambda^1(\square_3)$  and  $S_r^-\Lambda^2(\square_3)$  were also characterized by Cockburn and Fu [2017], as explained in Gillette and Kloforn [2019, Prop 2.3].

A major reason that the vector trimmed serendipity elements have only recently been considered in the mathematical literature is that their DOF per element count is complicated. As evidenced by Table 1, *certain* DOFs grow in predictable patterns with  $r$ . For instance, elements in the 1-form family,  $S_r^-\Lambda^1(\square_3)$ , have exactly  $r$  DOFs per edge of the cube (corresponding to “order  $r$ ” approximation on edges) and elements in the 2-form family,  $S_r^-\Lambda^2(\square_3)$ , have  $\binom{r+1}{2}$  DOFs per face (corresponding to “order  $r$ ” approximation on faces). However, the 2-form family has the more obscure quantity of  $(r^3 - 2r^2 + 3r)/2$  DOFs associated to the interior of an element (for  $r > 1$ ). This growth pattern is recognized as sequence A064808 by the On-line Encyclopedia of Integer Sequences [Sloane, editor 2021] and is in agreement with the general formulae presented in Gillette and Kloforn [2019], but a natural geometric interpretation remains elusive. Equally unexpected patterns are evident in the growth of DOFs on faces and interiors of the  $S_r^-\Lambda^1(\square_3)$  family. As we will discuss in the next section, Firedrake makes the creation and incorporation of such unusual DOF growth patterns simple for the developer, and thus opens these elements to numerical testing for the first time.

### 3 BUILDING CAPACITY FOR SERENDIPITY ELEMENT TYPES IN FIREDRAKE

Firedrake uses FIAT [Kirby 2004, 2012] to provide finite element basis functions on reference elements. To implement a new element in FIAT, we must provide both rules to tabulate basis functions and their derivatives at reference element points and a data structure that assigns basis functions to particular reference element facets. Said element is then made available in Firedrake by providing a symbolic name (in UFL [Alnæs et al. 2014]) and a translation from symbolic name to concrete implementation in the form compiler TSFC [Homolya et al. 2018]. While FIAT initially considered a very wide range of finite elements [Kirby et al. 2012], it would seek to express their bases as linear combinations of orthogonal polynomials. However, for some elements, it is easier to directly describe the basis functions. We follow the construction of Gillette et al. [2019] which provides explicit formulae for the basis functions for each of the trimmed serendipity elements and directly implement tabulation of the basis functions. To provide tabulations of derivatives, we implement the basis functions symbolically with SymPy [Meurer et al. 2017] and compute derivatives symbolically.

We use the decompositions from Gillette et al. [2019] to group the basis functions according to the geometric portions of a reference mesh element (vertices, edges, faces, or cell interiors). For instance, a basis for  $S_r^-\Lambda^1(\square_3)$  – the trimmed serendipity  $H(\text{curl})$ -conforming element in 3D – can

be decomposed as

$$\mathcal{S}_r^-\Lambda^1(\square_3) = \underbrace{\left[ \bigoplus_{i=0}^{r-1} E_i\Lambda^1 \right]}_{\text{edge functions}} \oplus \underbrace{\left[ \bigoplus_{i=2}^{r-1} F_i\Lambda^1 \right]}_{\text{face functions}} \oplus [\tilde{F}_r\Lambda^1] \oplus \underbrace{\left[ \bigoplus_{i=4}^{r-1} I_i\Lambda^1 \right]}_{\text{interior functions}} \oplus [\tilde{I}_r\Lambda^1]. \quad (2)$$

Subsets in these decompositions denoted with an  $E$ ,  $F$ , or  $I$  are defined on edges, faces, and interior, respectively, of the cubical cell in 3D.

To see how this plays out in the Firedrake implementation, consider the  $H(\text{curl})$  elements for trimmed serendipity at order  $r = 2$  in  $n = 3$  dimensions, indicating the space  $\mathcal{S}_2^-\Lambda^1$  in 3D. The space  $\tilde{I}_r\Lambda^1$  is defined to be empty for  $r < 4$ , so there are only two sets of functions to include in this case: one set associated to edges of the reference element (the  $E_i\Lambda^1$  sum) and one set for the faces of the reference element (the  $\tilde{F}_2\Lambda^1$  functions). According to Eq. (2), the basis functions can be decomposed as

$$\mathcal{S}_2^-\Lambda^1(\square_3) = \left[ \bigoplus_{i=0}^1 E_i\Lambda^1 \right] \oplus [\tilde{F}_2\Lambda^1] \oplus [\tilde{I}_2\Lambda^1]. \quad (3)$$

We then implement these in Firedrake as follows. The first step is to determine the number of DOFs we will need on the reference element where we will define the basis functions. To do this, we count the DOFs for each mesh entity on the reference element (vertex, edge, face, and interior). For example,  $\mathcal{S}_2^-\Lambda^1(\square_3)$  should have no DOFs at the vertices (these only come into play for the  $H^1$ -conforming elements), two DOFs on each edge, two DOFs on each face, and no DOFs on the interior. This agrees with Eq. (3), where  $E_i\Lambda^1$  supplies one DOF to each edge for each  $i = 0, 1$ , and then  $\tilde{F}_2\Lambda^1$  supplies two DOFs to each face. An illustration of this can be seen in the second column, second row of Table 1.

With the number of DOFs assigned to each mesh entity in the reference element, we can then define the basis functions. The order of definition is important so that basis functions are matched to the proper mesh entity. Note that Eq. (3) doesn't explicitly give a way to order the basis functions. Instead, we need to use the properties of the basis functions to determine the correct ordering. This is best illustrated by an example. Two of the "edge" basis functions that are contained in the sum of the  $E_i\Lambda^1$  sets are  $(y+1)(z+1)dx$  and  $x(y+1)(z+1)dx$ . Notice that these functions have no  $dy$  or  $dz$  portions. Therefore, these functions vanish on any edge *not* parallel to the  $x$  axis. Further, the polynomial coefficients of these forms indicate that they also vanish on the planes  $y = -1$  and  $z = -1$ . Thus, the only edge of the cube on which these functions do *not* vanish is the edge contained in the line  $\{y = 1\} \cap \{z = 1\}$ . This edge is shown in blue and labeled with an "e" in Fig. 3.

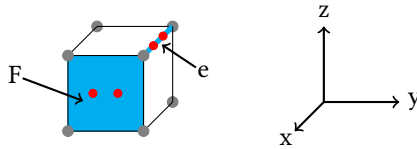


Fig. 3. The reference cube  $[0, 1]^3$  is shown, with coordinate axes indicated. The edge  $e$  lies in the intersection of the planes  $y = 1$  and  $z = 1$ . To ensure proper continuity, basis functions associated to  $e$  must vanish on all edges of the cube *except*  $e$ . Examples of such functions for  $e$  are described in detail in the text. Additional examples of functions associated to the face  $F$  are also given.

This process is what determines the ordering for the basis functions. If, in the first step described above where we are assigning DOFs to mesh entities, we assign the first two DOFs to be on the



edge of the reference element where  $y = 1$  and  $z = 1$ , then we must define the basis functions  $(y + 1)(z + 1)dx$  and  $x(y + 1)(z + 1)dx$  as our first two basis functions.

While the formulas above are taken from [Gillette et al. \[2019\]](#), these monomials have poor conditioning at higher orders. Therefore, we use Legendre polynomials obtained symbolically from SymPy via the `leg` function. Then we write the differential forms in vector notation. For our examples of  $(y + 1)(z + 1)dx$  and  $x(y + 1)(z + 1)dx$ , we get `tuple([(leg(j, x_mid)*dz[1]*dy[1], 0, 0)])`, where `leg(j, x_mid)` is used for the 1 or  $x$  coefficient, and the `dy[1]` and `dz[1]` are used for the values of  $(y + 1)$  and  $(z + 1)$ , respectively. After repeating this process for each of the edges and associated basis functions, we then also do a similar process for the face functions in the set  $\tilde{F}_2\Lambda^1$ .

This process changes only slightly if we instead had considered the the 2-forms  $\mathcal{S}_2^-\Lambda^2(\square_3)$ . In this scenario, we would have the basis functions given by the equation

$$\mathcal{S}_2^-\Lambda^2(\square_3) = \left[ \bigoplus_{i=0}^1 F_i\Lambda^2 \right] \oplus [\tilde{I}_2\Lambda^2].$$

In this case, one of the basis functions is  $y^j z^k (x + 1)dydz$ . The  $dydz$  represents a 2-form, which vanishes on any face *not* parallel to the  $yz$  plane. This leaves only the faces contained in the planes  $x = 1$  or  $x = -1$  as possibilities for association. As in the 1-form example above, the polynomial coefficient of the form indicates that this function will vanish on an additional mesh entity, namely, the face contained in the plane  $x = -1$ , in this case. Hence, we associate this function with the face contained in  $x = 1$  (labeled with an “F” in [Fig. 3](#)). The rest of the process for defining the 2-form basis functions is similar to the process for the 1-form basis functions.

Table 2. A translation between FEEC and Firedrake usage names for tensor product and trimmed serendipity elements. In 2D, the 0-forms are  $H^1$  conforming spaces, the 1-forms are  $H(\text{curl})$  and  $H(\text{div})$  conforming spaces (dependent upon orientation of the DOFs), and the 2-forms are  $L^2$  conforming spaces. For 3D, the 0-forms are  $H^1$  conforming spaces, the 1-forms are  $H(\text{curl})$  conforming spaces, 2-forms are  $H(\text{div})$  conforming spaces, and 3-forms are  $L^2$  conforming spaces.

FEEC	UFL name (2D)	UFL name (3D)
$Q_r^-\Lambda^0$	Lagrange	Lagrange
$Q_r^-\Lambda^1$	RTCE or RTCF	NCE
$Q_r^-\Lambda^2$	DQ	NCF
$Q_r^-\Lambda^3$	-	DQ
$S_r^-\Lambda^0$	S	S
$S_r^-\Lambda^1$	SminusCurl or SminusDiv	SminusCurl
$S_r^-\Lambda^2$	DPC	SminusDiv
$S_r^-\Lambda^3$	-	DPC

The newly supported elements, mapping FEEC spaces onto names in UFL are shown in the lower half of [Table 2](#). Modifying the code from [Listing 1](#) to utilise trimmed serendipity spaces rather than tensor product spaces is then simply a case of replacing the element names in the `FunctionSpace` definitions with appropriate trimmed space names. Concretely, the new `FunctionSpace` definitions are shown in [Listing 2](#), the rest of the code remains unchanged.

## 4 EXPERIMENTS

The following experiments show the benefits and costs of using trimmed serendipity elements in comparison to traditional tensor product elements. We first present a basic projection example as a



Listing 2. Setting up Firedrake to use the trimmed serendipity elements in a mixed Poisson problem in 3D.

---

```

3  ...
4  hDivSpace = FunctionSpace(mesh, "SminusDiv", polyDegree)
5  l2Space = FunctionSpace(mesh, "DPC", polyDegree - 1)
6  ...

```

---

means of confirming approximation properties of our elements. Next, we present results on a primal Poisson problem (to test  $H^1$  elements), a mixed Poisson problem (to test  $H(\text{div})$  and  $L^2$  elements), and a cavity resonator problem (to test  $H(\text{curl})$  elements). Since the  $H(\text{curl})$  and  $H(\text{div})$  elements are only a rotation of the DOFs on the edges of elements in 2D, we do not give an experiment using  $H(\text{curl})$  elements in 2D.

The experiments were performed either on a single cluster compute node with 2x AMD EPYC 7642 48-core (Rome) processors (2.4GHz) and 512GB of memory running CentOS 7 or a similar node with 3TB of memory. The 2D experiments were all done using the 512GB node, while in 3D, the 4th order experiments were run on the 3TB node. Each job was run by submitting a SLURM script that requested one node in isolation to ensure no other jobs were running at the same time. We utilized on-node parallelism by requesting a full node and executing the jobs with `mpirun` set to use 24 processes, with a few exceptions for smaller cases that will be pointed out later. Timing data was collected after first performing a dry run of the code to warm the cache and then taking the minimum time over three subsequent runs. The timing results presented here depend upon the solver choice, and changing that may give different results illustrating the relative efficiency between  $Q^-$  and  $S^-$ .

For simplicity, our numerical experiments all use a sparse direct solver. We expect the multigrid theory in Arnold et al. [2000, 2006] to carry over from existing  $Q^-$  spaces to  $S^-$  spaces. Optimal smoothers require aggregating degrees of freedom for vertex patches, and we anticipate that the reduction in local dimensionality that trimmed serendipity spaces offer will be beneficial in these contexts as well.

#### 4.1 Projection

We solve an  $L^2$  projection problem to give a baseline accuracy test for the elements. Given either the unit square or unit cube as our domain of integration on definite integrals, we compute the projection of  $f$  into the function space  $V$  by using a discretization of the problem: find  $u \in V$  such that

$$\int_{\Omega} u \cdot v \, dx = \int_{\Omega} g \cdot v \, dx, \text{ where } g = \nabla f$$

for all  $v \in V$ . For our experiments, we choose  $V$  to be  $H(\text{curl})$  spaces for the proper dimension (see Table 2) and  $f = \sin(\pi x)\sin(\pi y)$  or  $f = \sin(\pi x)\sin(\pi y)\sin(\pi z)$  for two or three dimensions respectively. Recall that the trimmed serendipity elements are denoted with  $S_r^-$  and the tensor product elements with  $Q_r^-$ . In Firedrake, the trimmed serendipity elements use the label `SminusCurl` or `SminusDiv` for 1-forms in 2D, depending upon the orientation of the edge DOFs, and in 3D, they represent the 1- and 2-forms respectively. The tensor product elements use the labels `RTCE` (or `RTCF`) and `NCE` for 1-forms in 2D and 3D, while the 2-forms in 3D are `NCF`. To solve the projection problem, we use a Galerkin  $L^2$  projection into  $V$ .

The goal of the projection problem is to establish that the elements attain the mathematically expected rates of convergence as mesh size decreases, as well as comparing relative efficiencies of  $S^-$  and  $Q^-$ . For this, we create a mesh on  $[0, 1]^n$  of uniformly sized squares or cubes, where we refine the mesh from  $N = 4$  squares (or cubes) in each row and column to  $N = 128$  squares in

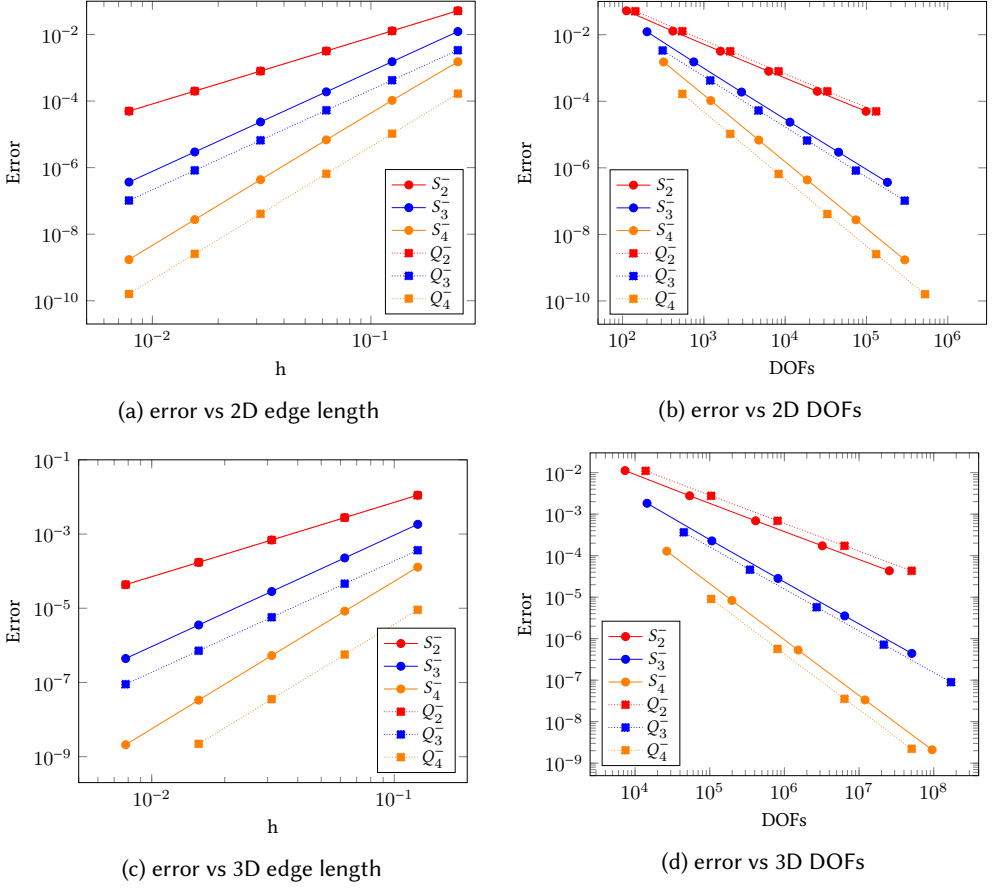


Fig. 4. Results of solving an  $L^2$  projection problem using trimmed serendipity and tensor product  $H(\text{curl})$  elements in both 2D (top row) and 3D (bottom row). Experiments were ran for  $k$ -forms for  $k = 0, 1, 2, 3$  in 2D and 3D, however only the 1-forms in 3D are displayed here. In every case, the trimmed serendipity element trendline has the same slope as its tensor product counterpart, as expected by the theory. In most cases, the tensor product elements perform better by these metrics, in the sense that their trendlines lie below the trimmed serendipity trendlines. The notable exception is the quadratic case,  $r = 2$ , where trimmed serendipity perform slightly better in terms of DOFs, in both 2D and 3D.

each row and column (or  $N = 64$  cubes). This results in a mesh with  $N^2$  or  $N^3$  squares or cubes respectively. For the following results, we will use  $h = \frac{1}{N}$  since the mesh elements are all uniformly sized. We employ both trimmed serendipity elements and tensor product elements on each mesh and record the  $L^2$  error in each case. In Fig. 4, we report the  $L^2$  error in terms of the classical measure of maximum edge length ( $h$ ) as well as total number of DOFs.

The expectation is that  $\mathcal{S}_r^-$  and  $\mathcal{Q}_r^-$  converge at the same rate with respect to  $h$ , which is confirmed in the plots by parallel trendlines. These parallel trendlines can be seen in each of the projection plots. For  $r = 2$ , we see the trendline for  $\mathcal{S}_r^-$  below the trendline for  $\mathcal{Q}_r^-$ , indicating that the trimmed serendipity elements achieve a better accuracy for the amount of DOFs required than the tensor product elements.

For the  $r = 2$  case, the trimmed serendipity elements achieve a better accuracy while requiring fewer DOFs. Therefore in this low order case, trimmed serendipity elements would be beneficial to use. In the case of  $r = 3, 4$  the trendlines for  $S_r^-$  are above the trendlines for  $Q_r^-$ . However, considering the elements  $Q_3^-$  and  $S_4^-$ , we see that  $Q_3^-$  requires approximately  $1.71 \times 10^8$  DOFs and  $S_4^-$  at the same mesh refinement requires  $0.95 \times 10^8$  DOFs. While  $Q_3^-$  attains an error of  $8.96 \times 10^{-8}$ , the  $S_4^-$  elements attain an error of  $2.1 \times 10^{-9}$ .

## 4.2 The Poisson Problem

In this section we discuss results for both the primal Poisson problem and the mixed Poisson problem. We solve the primal Poisson problem described below on a unit square domain  $\Omega$  for  $u \in U$ :

$$\begin{aligned} -\Delta u &= f \\ u|_{\partial\Omega} &= 0 \end{aligned}$$

where  $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$ , yielding the solution  $u(x, y) = \sin(\pi x) \sin(\pi y)$ . In 3D, we can extend this to  $f(x, y, z) = 3\pi^2 \sin(\pi x) \sin(\pi y) \sin(\pi z)$  with the solution  $u(x, y, z) = \sin(\pi x) \sin(\pi y) \sin(\pi z)$  on the unit cube. The primal Poisson equation is as follows: find  $u \in V := H^1(\Omega)$  such that:

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx = \int_{\Omega} f v \, dx, \quad \text{for all } v \in V.$$

Accordingly, the primal Poisson problem employs  $H^1$  elements, using  $S$  for  $S_r^- \Lambda^0$  and Lagrange for  $Q_r^- \Lambda^0$ .

The mixed Poisson problem introduces an intermediate variable,  $\sigma$ , which is solved for simultaneously. Formally, this is: find  $\sigma \in H(\text{div})$  and  $u \in L^2$  such that:

$$\begin{aligned} \sigma - \nabla u &= 0 \\ \nabla \cdot \sigma &= -f \\ u|_{\partial\Omega} &= 0 \end{aligned}$$

In a similar fashion as for the primal Poisson problem, we can create the weak formulation of the mixed Poisson problem that we saw in Eq. (1).

These equations are discretized and solved using a suitable *pair* of finite elements – one of  $H(\text{div})$  type and one of  $L^2$  type. We use  $(Q_r^- \Lambda^{n-1}, Q_r^- \Lambda^n)$  and  $(S_r^- \Lambda^{n-1}, S_r^- \Lambda^n)$  for dimensions  $n = 2$  and  $n = 3$ . Note that the mathematical notation here calls for  $Q_r^- \Lambda^{n-1}$  to be paired with  $Q_r^- \Lambda^n$ , but the code notation will require setting the degree of the  $L^2$  element one below the degree of the  $H(\text{div})$  element, and similar with the trimmed serendipity elements. For both the primal Poisson and mixed Poisson problems, we solve the system using MUMPS [Amestoy et al. 2001, 2006] with a sparse direct LU factorization. The exact set of solver parameters used for the mixed Poisson problem are shown in Listing 3.

The empirical convergence results for the primal Poisson and mixed Poisson problems can be seen in Figs. 5a to 5d. In each of the subfigures of Fig. 5, we see that independent of the performance of each element, the  $S_r^-$  and  $Q_r^-$  have parallel trendlines, indicating that they have the same overall convergence rate. For the primal Poisson problem in 2D and 3D, the trimmed serendipity elements perform as well or better than the tensor product elements for orders  $r = 2, 3$ . Furthermore, comparing the elements via DOFs as in the projection problem yields another instance where we see that using  $S_3^-$  instead of  $Q_2^-$  will attain a higher accuracy for essential the same number of DOFs.

In Fig. 6 we analyze the timing data for computing the solutions to the primal and mixed Poisson problems using trimmed serendipity and tensor product elements. As in the error vs DOFs graphs,

Listing 3. An example of some solver parameters that we can use for the mixed Poisson problem. The options presented here solve the algebraic system with a simplified Newton method where the Jacobian is held constant at the first iterate. Therefore it is factored at the beginning and triangular solves are applied to it at each subsequent iteration. This has the effect of performing iterative refinement [Moler 1967; Wilkinson 1994] and yields an increased algebraic accuracy on fine meshes.

---

```

1  ...
2  params = {"snes_type": "newtonls",
3           "snes_linesearch_type": "basic",
4           "snes_monitor": None,
5           "snes_converged_reason": None,
6           "mat_type": "aij",
7           "snes_max_it": 10,
8           "snes_lag_jacobian": -2,
9           "snes_lag_preconditioner": -2,
10          "ksp_type": "preonly",
11          "ksp_converged_reason": None,
12          "ksp_monitor_true_residual": None,
13          "pc_type": "lu",
14          "snes_rtol": 1e-12,
15          "snes_atol": 1e-20,
16          "pc_factor_mat_solver_type": "mumps",
17          "mat_mumps_icntl_14": "200",
18          "mat_mumps_icntl_11": "2"}
19  ...

```

---

in Figs. 6a and 6b, we see good evidence that serendipity elements are able to produce better results at a faster rate. Specifically, in Fig. 6a, we can see that serendipity elements are able to do a larger number of DOFs in the same amount of time. In Fig. 6b we see that for a given error level, trimmed serendipity elements require less time. Overall Fig. 6c shows that the time scaling is dependent upon the number of DOFs regardless of the element that is used. Further evidence of this is seen in Fig. 6d. Similar to the previous analysis of DOFs vs error for the mixed Poisson problem, the timing data here illustrates that attaining the extra accuracy from using  $\mathcal{S}_3^-$  instead of  $\mathcal{Q}_2^-$  does not invoke a larger time requirement.

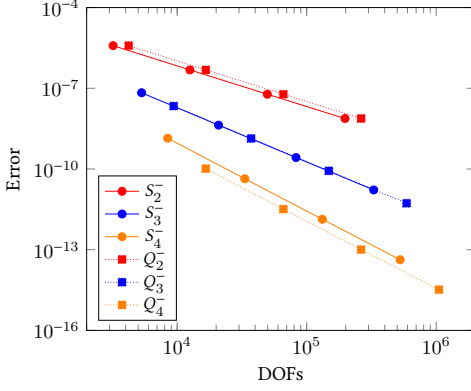
### 4.3 Cavity Resonator

The last numerical experiment that we give here is the cavity resonator problem, making use of the  $H(\text{curl})$  elements in 3D. We pose a Maxwell eigenvalue problem on the domain  $\Omega = [0, 1]^3$  with perfectly conducting boundary conditions, yielding an eigenvalue problem where  $\lambda$  represents a quantity proportional to the frequency squared of the time-harmonic electric field (i.e. eigenvalues) and  $E$  represents the electric field (i.e. eigenfunctions):

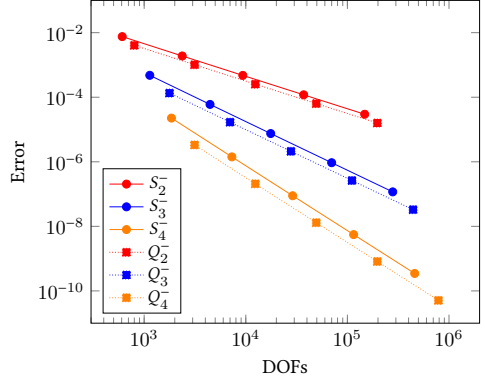
$$\begin{aligned}\nabla \cdot E &= 0 \text{ in } \Omega \\ \nabla \times \nabla \times E &= \lambda E \text{ in } \Omega \\ E \times n &= 0 \text{ on } \partial\Omega.\end{aligned}$$

We consider the weak formulation of this problem (similar to Fumio [1987]), where  $\omega$  represents the resonances (i.e. eigenvalues) and  $E$  represents the electric field (i.e. eigenfunctions):

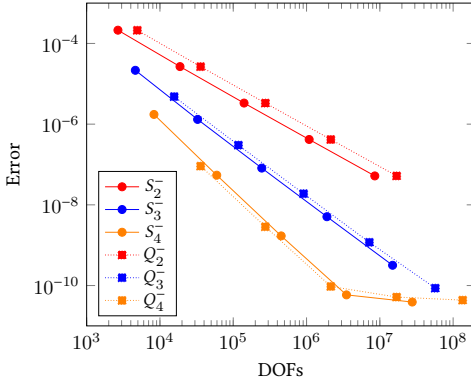
$$\int_{\Omega} (\nabla \times F) \cdot (\nabla \times E) \, dx = \omega^2 \int_{\Omega} F \cdot E \, dx \text{ for all } F \in H_0(\text{curl}).$$



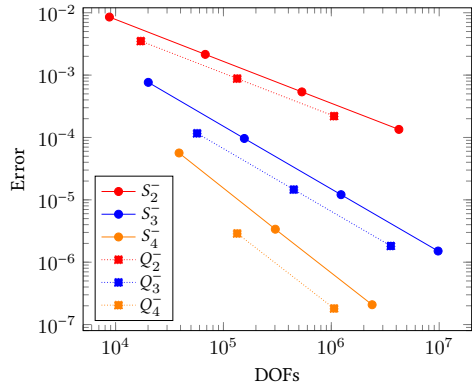
(a) 2D primal Poisson convergence analysis.



(b) 2D mixed Poisson convergence analysis.



(c) 3D primal Poisson convergence analysis.



(d) 3D mixed Poisson convergence analysis.

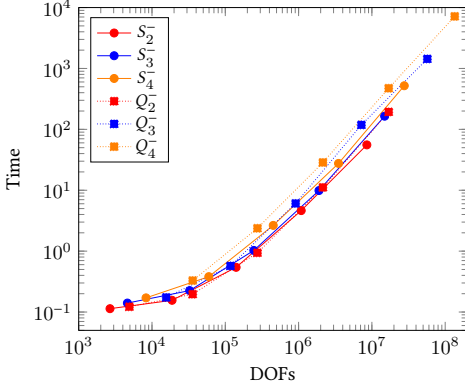
Fig. 5. A convergence analysis of the primal and mixed Poisson problems in 2D and 3D. Error here is calculated as the  $L^2$  error between the exact solution and the approximate using the corresponding finite element space.

The exact eigenvalues follow the formula

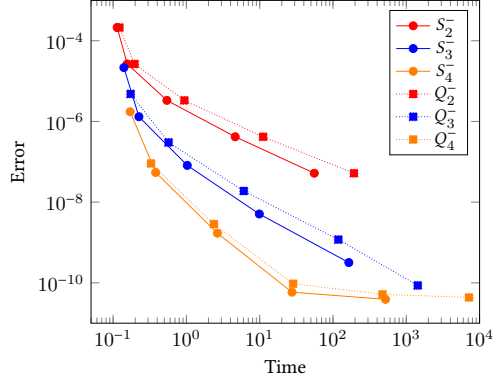
$$\omega^2 = m_1^2 + m_2^2 + m_3^2$$

where  $m_i \in \mathbb{N} \cup 0$  and no more than one of  $m_1, m_2, m_3$  may be equal to 0 at a time [Rognes et al. 2010].

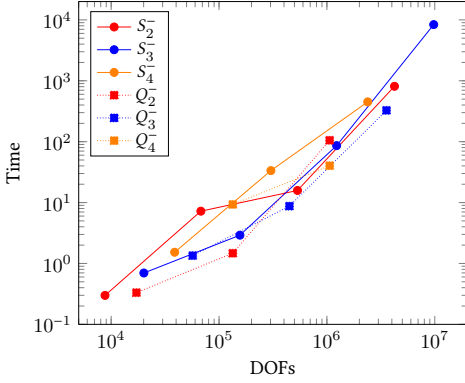
In Table 3, we display the convergence rates of different eigenvalues when computing the eigenvalues with tensor product and trimmed serendipity elements in 3D. The table is split into halves, the top half showing values from using  $Q^- H(\text{curl})$  elements while the bottom half shows values from using the corresponding  $S^-$  elements. Each half of the table has a row giving the DOFs in the mesh for each refinement level  $N$  and a row giving the time per iteration that the solver required.



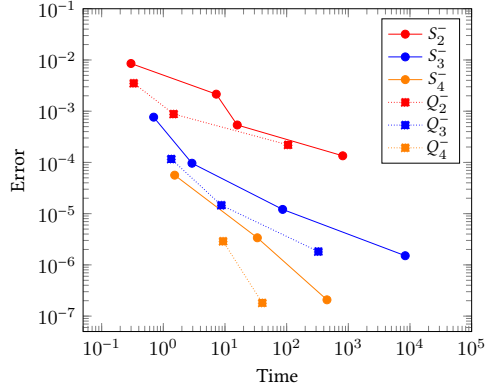
(a) 3D primal Poisson Time vs DOFs



(b) 3D primal Poisson Error vs Time



(c) 3D mixed Poisson Time vs DOFs



(d) 3D Mixed Poisson Error vs Time

Fig. 6. Analyzing timing data for primal and mixed Poisson problems using trimmed serendipity and tensor product elements. Error here is calculated as the  $L^2$  error between the exact solution and the approximate solution found using the corresponding finite element space.

Note that the convergence rates are computed by

$$r = \frac{\log\left(\frac{\tilde{\lambda}_{i,N} - \lambda_{i,N}}{\lambda_{i,N+1} - \lambda_{i,N+1}}\right)}{\log\left(\frac{h_N}{h_{N+1}}\right)}.$$

Based off earlier eigenvalue works [Boffi 2010], we expect the rate of convergence to be double the order of the finite element used to solve the problem. This is reflected in the table well for both  $S^-$  and  $Q^-$  elements. The “-” entries in the table indicate the eigenvalue solver did not find that specific eigenvalue in the allowed number of iterations; we set the solver to iterate a sufficient number of times to find the first 15 eigenvalue-eigenvector pairs.

The experiment was done by using Firedrake to create the mass and stiffness matrices as petsc4py objects [Balay et al. 2021, 1997; Dalcin et al. 2011], then using slepc4py [Hernandez et al. 2005; Roman et al. 2020] to do the eigenvalue analysis. The eigenvalue analysis was done by computing

Table 3. A comparison of how  $\mathcal{Q}_2^-$  and  $\mathcal{S}_2^-$  finite elements solve the Maxwell cavity resonator eigenvalue problem,  $\langle \text{curl}(F), \text{curl}(E) \rangle = \omega^2 \langle F, E \rangle$ . An eigenvalue found with the same error multiple times was condensed to a single row. Numbers in parentheses next to the actual eigenvalue are the number of times we found an approximation of the actual eigenvalue. The columns labeled  $N = 4, 8, 16, 32$  are giving the approximate eigenvalues found on a mesh of size  $N \times N \times N$ . The values in parentheses in these columns indicates the rate of convergence for that approximate eigenvalue.

$\mathcal{Q}_2^- H(\text{curl})$ Elements				
Actual (Count)	N = 4	N = 8	N = 16	N = 32
2 (3)	2.001024	2.000066 (3.96)	2.000004 (4.04)	2.0000003 (4.00)
3 (2)	3.001536	3.000098 (3.97)	3.000006 (4.03)	3.0000004 (4.02)
5 (4)	5.030601	5.002081 (3.88)	5.000133 (3.97)	5.000008 (4.06)
6 (3)	6.031114	6.002114 (3.88)	6.000135 (3.97)	6.000008 (4.08)
8 (0)	—	—	—	—
DOF	1944	13872	104544	811200
EPS solve time per iteration	0.01565225	0.0743845	1.0484236	7.6186526

$\mathcal{S}_2^- H(\text{curl})$ Elements				
Actual (Count)	N = 4	N = 8	N = 16	N = 32
2 (3)	2.001092	2.000066 (4.05)	2.000004 (4.04)	2.000000 (4.00)
3 (2)	3.009018	3.000586 (3.94)	3.000037 (3.99)	3.000002 (4.21)
5 (3)	5.032027	5.002097 (3.93)	5.000133 (3.98)	5.000008 (4.06)
5 (1)	5.032027	5.002097 (3.93)	5.000133 (3.98)	—
6 (1)	6.072012	6.004976 (3.86)	6.000319 (3.96)	6.000020 (4.00)
6 (1)	6.072012	6.004976 (3.86)	6.000319 (3.96)	6.000024 (3.73)
6 (1)	—	—	6.00038	6.000024 (3.98)
8 (1)	—	—	—	8.000017
DOF	1080	7344	53856	411840
EPS solve time per iteration	0.01288725	0.0309768	0.401663	4.1996873

an inverted shift to a target of 3.0, then solving for 15 eigenvalue-eigenvector pairs. The SLEPc solve was done using the default (Krylov-Schur) solver with a tolerance level of  $10^{-7}$ . We note that the eigensolver finds a varying number of spurious eigenvalues with value 1. These exist because Firedrake enforces strong boundary conditions by placing a 1 on the diagonal and zeroing out the rows and columns, not due to the elements that we use or the SLEPc solver that is called. We do not report these eigenvalues.

Since both elements attain the expected convergence rate, we focus on the rest of the results in the table. Investigating the error in the eigenvalues in the chart compared to the exact values, we see that tensor product elements are able to get results that are up to an order of magnitude better near the target eigenvalue. On the other hand, this loss of accuracy from using trimmed serendipity elements is offset by a reduction in required time to solve for the requested eigenvalues. At every mesh refinement level, trimmed serendipity elements have nearly half the DOFs of tensor product elements, and correspondingly, require approximately half the time per iteration to solve for the



eigenvalues (outside of the case  $N = 4$ ). At higher orders, we expect that this will be even more exaggerated.

Continuing the eigenvalue example, we used Firedrake and SLEPc to compute two eigenvalues,  $\lambda = 3$  and  $\lambda = 5$ . We computed the eigenvalues at different orders of the elements, from  $r = 2$  to  $r = 5$ , and kept the mesh constant at  $16 \times 16 \times 16$ . The timing data was then collected by choosing the largest time required for any of the multiplicities of 3 or 5 that the solver found.

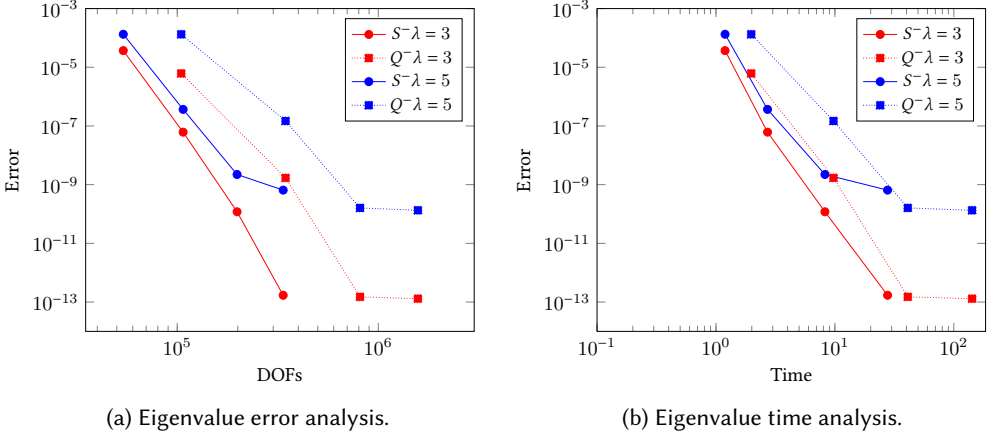


Fig. 7. Results for solving for  $\lambda = 3$  and  $\lambda = 5$  using Firedrake and SLEPc by increasing the order from 2 to 5. Error is calculated as the absolute value of the error between the actual eigenvalue and the approximated eigenvalue.

The error results shown in Fig. 7a for the eigenvalue problem indicate that trimmed serendipity elements yield less error in the eigenvalues for the number DOFs required to compute them than the tensor product elements. This is a change from the mixed Poisson problem where the DOFs vs Error trendline for trimmed serendipity was generally above the trendline for tensor product elements. The timing results in Fig. 7b for the trimmed serendipity elements were similarly better for the eigenvalue problem in comparison to the tensor product elements. Overall, our results indicate that small eigenvalues are computed more efficiently with trimmed serendipity elements when one considers error vs total DOF count or run-time.

## 5 DISCUSSION

This implementation of trimmed serendipity elements gives a new method for computing the solution to a discretized PDE and has been tested on meshes of squares and cubes. Completing the implementation of these elements within Firedrake by using the basis functions defined in Gillette et al. [2019] is an illustration of Firedrake’s modular capabilities for implementing new and unusual finite elements.

The convergence studies done in each of the numerical experiments show that the trimmed serendipity elements can attain the theoretical rates of convergence that they were predicted to achieve. While we only illustrate orders 2, 3, and 4 in 2D and 3D, our implementation of trimmed serendipity elements in Firedrake is designed to work in both 2D and 3D for arbitrary orders  $r$ .

In comparison to tensor product elements  $Q_r^-$ , the we make a choice when using trimmed serendipity elements  $S_r^-$  to lower accuracy in return for less computation, both in terms of DOFs and time required. At low orders the choice to use trimmed serendipity elements could actually

reduce the error per DOF, as we saw in the primal Poisson problem, where the trendlines for DOFs vs Error for trimmed serendipity elements were below the trendlines for the tensor product elements. In the mixed Poisson case however, the opposite was true, and the trendlines for the tensor product elements were below the trendlines for the trimmed serendipity elements.

Rather than comparing in terms of order, it can also be beneficial to compare the two elements based off of the DOFs that they require. Consider the 3D mixed Poisson problem while focusing on DOFs vs Error given in Fig. 5d. The tensor product elements  $Q_2^-$  required a similar number of DOFs as the trimmed serendipity elements  $S_3^-$ . Compared this way, the trimmed serendipity elements provide an extra order of magnitude of accuracy over the tensor product element. Furthermore, in Fig. 6d the time required for  $S_3^-$  and  $Q_2^-$  was also approximately equal. Thus while it is helpful to compare  $Q_r^-$  and  $S_r^-$  to see that the trimmed serendipity elements have the expected convergence behavior, a more practical computational comparison is between  $Q_r^-$  and  $S_{r+1}^-$ .

The eigenvalue problem yields another example of comparing the tensor product and trimmed serendipity elements, where instead of refining the mesh, we refined the order of the element used. Just as in the mixed Poisson problem, we again see that Fig. 7a shows  $S_2^-$  has a higher error for  $\lambda = 3$  than  $Q_2^-$ . However comparing against where the DOFs are approximately equal leads to a comparison between  $S_3^-$  and  $Q_2^-$ . In this scenario, we had that  $Q_2^-$  required 104544 DOFs yielding an error of  $1.33 \times 10^{-4}$  for  $\lambda = 5$  while  $S_3^-$  required 106896 and achieved an error of  $3.67 \times 10^{-7}$  for  $\lambda = 5$ . In this case, we note that the time required for  $S_3^-$  did require more time to solve, using about 2.71 seconds while the  $Q_2^-$  required 1.98 seconds.

Our computational findings suggest that trimmed serendipity elements could be particularly beneficial at improving accuracy for compute-bounded applications. For any application, there is eventually a mesh resolution and element order for which refining the mesh or increasing the tensor product order is computationally infeasible. In this instance, keeping the mesh but switching to a trimmed serendipity method of one order higher presents a new option to the practitioner that still provides an increase in accuracy without a significant increase to computational cost.

## CODE AVAILABILITY

All major Firedrake components, as well as the code for the numerical experiments in the paper have been archived on [Zenodo](#) [2021].

## ACKNOWLEDGMENTS

This work was supported by National Science Foundation (NSF) Collaborative Research Awards DMS-1913094 and DMS-1912653 and by U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number DE-SC-0019039.

## 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), 1–37. <https://doi.org/10.1145/2566630>
- Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.* 23, 1 (2001), 15–41. <https://doi.org/10.1137/S0895479899358194>
- Patrick R. Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stéphane Pralet. 2006. Hybrid scheduling for the parallel solution of linear systems. *Parallel Comput.* 32, 2 (2006), 136–156. <https://doi.org/10.1016/j.parco.2005.07.004>
- Todd Arbogast and Maicon R. Correa. 2016. Two families of  $H(\text{div})$  mixed finite elements on quadrilaterals of minimal dimension. *SIAM J. Numer. Anal.* 54, 6 (2016), 3332–3356. <https://doi.org/10.1137/15M1013705>
- Douglas N. Arnold and Gerard Awanou. 2011. The serendipity family of finite elements. *Foundations of Computational Mathematics* 11, 3 (2011), 337–344. <https://doi.org/10.1007/s10208-011-9087-3>
- Douglas N. Arnold and Gerard Awanou. 2014. Finite element differential forms on cubical meshes. *Math. Comp.* 83, 288 (2014), 1551–1570. <https://doi.org/10.1090/S0025-5718-2013-02783-4>

- Douglas N. Arnold, Daniele Boffi, and Francesca Bonizzoni. 2015. Finite element differential forms on curvilinear cubic meshes and their approximation properties. *Numer. Math.* 129, 1 (2015), 1–20. <https://doi.org/10.1007/s00211-014-0631-3>
- Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2000. Multigrid in  $H(\text{div})$  and  $H(\text{curl})$ . *Numer. Math.* 85 (2000), 197–217. <https://doi.org/10.1007/PL00005386>
- Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2006. Finite element exterior calculus, homological techniques, and applications. *Acta Numerica* 15 (2006), 1–155. <https://doi.org/10.1017/S0962492906210018>
- Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2010. Finite element exterior calculus: from Hodge theory to numerical stability. *Bull. Amer. Math. Soc.* 47, 2 (2010), 281–354. <https://doi.org/10.1090/S0273-0979-10-01278-4>
- Douglas N. Arnold and Anders Logg. 2014. Periodic table of the finite elements. *SIAM News* 47, 9 (2014), 212. <https://sinews.siam.org/Details-Page/periodic-table-of-the-finite-elements>
- Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, 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. 2021. *PETSc Users Manual*. Technical Report ANL-95/11 - Revision 3.15. Argonne National Laboratory. <https://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf>
- 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, Boston, MA, 163–202. [https://doi.org/10.1007/978-1-4612-1986-6\\_8](https://doi.org/10.1007/978-1-4612-1986-6_8)
- Daniele Boffi. 2010. Finite element approximation of eigenvalue problems. *Acta Numerica* 19 (2010), 1–120. <https://doi.org/10.1017/S0962492910000012>
- Snorre H. Christiansen and Andrew Gillette. 2016. Constructions of some minimal finite element systems. *ESAIM: Mathematical Modelling and Numerical Analysis* 50, 3 (2016), 833–850. <https://doi.org/10.1051/m2an/2015089>
- Bernardo Cockburn and Guosheng Fu. 2017. A systematic construction of finite element commuting exact sequences. *SIAM J. Numer. Anal.* 55, 4 (2017), 1650–1688. <https://doi.org/10.1137/16M1073352>
- Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. 2011. Parallel distributed computing using Python. *Advances in Water Resources* 34, 9 (2011), 1124–1139. <https://doi.org/10.1016/j.advwatres.2011.04.013> New Computational Methods and Software Tools.
- Kikuchi Fumio. 1987. Mixed and penalty formulations for finite element analysis of an eigenvalue problem in electromagnetism. *Computer Methods in Applied Mechanics and Engineering* 64, 1-3 (1987), 509–521. [https://doi.org/10.1016/0045-7825\(87\)90053-3](https://doi.org/10.1016/0045-7825(87)90053-3)
- Andrew Gillette and Tyler Kloefkorn. 2019. Trimmed serendipity finite element differential forms. *Math. Comp.* 88, 316 (2019), 583–606. <https://doi.org/10.1090/mcom/3354>
- Andrew Gillette, Tyler Kloefkorn, and Victoria Sanders. 2019. Computational serendipity and tensor product finite element differential forms. *The SMAI Journal of Computational Mathematics* 5 (2019), 1–21. <https://doi.org/10.5802/smai-jcm.41>
- 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, 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>
- 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. 2012. FIAT: numerical construction of finite element basis functions. See [Logg et al. 2012], 247–255. [https://doi.org/10.1007/978-3-642-23099-8\\_13](https://doi.org/10.1007/978-3-642-23099-8_13)
- Robert C Kirby, Anders Logg, Marie E Rognes, and Andy R Terrel. 2012. Common and unusual finite elements. See [Logg et al. 2012], 95–119. [https://doi.org/10.1007/978-3-642-23099-8\\_3](https://doi.org/10.1007/978-3-642-23099-8_3)
- Anders Logg, Kent-Andre Mardal, and Garth N. Wells (Eds.). 2012. *Automated Solution of Differential Equations by the Finite Element Method: the FEniCS Book*. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-23099-8>
- Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. <https://doi.org/10.7717/peerj-cs.103>
- Cleve B. Moler. 1967. Iterative refinement in floating point. *J. ACM* 14, 2 (1967), 316–321. <https://doi.org/10.1145/321386.321394>
- J.-C. Nédélec. 1980. Mixed finite elements in  $\mathbb{R}^3$ . *Numer. Math.* 35, 3 (1980), 315–341. <https://doi.org/10.1007/BF01396415>
- J.-C. Nédélec. 1986. A new family of mixed finite elements in  $\mathbb{R}^3$ . *Numer. Math.* 50, 1 (1986), 57–81. <https://doi.org/10.1007/BF01389668>

- 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), 1–27. <https://doi.org/10.1145/2998441>
- P. A. Raviart and J. M. Thomas. 1977. A mixed finite element method for 2nd order elliptic problems. In *Mathematical aspects of finite element methods*. Springer, Berlin, Heidelberg, 292–315. <https://doi.org/10.1007/BFb0064470>
- Marie E. Rognes, Robert C. Kirby, and Anders Logg. 2010. Efficient assembly of  $H(\text{div})$  and  $H(\text{curl})$  conforming finite elements. *SIAM Journal on Scientific Computing* 31, 6 (2010), 4130–4151. <https://doi.org/10.1137/08073901X>
- Jose E. Roman, Carmen Campos, Lisandro Dalcin, Eloy Romero, and Andrés Tomás. 2020. *SLEPc Users Manual*. Technical Report DSIC-II/24/02 - Revision 3.15. D. Sistemes Informàtics i Computació, Universitat Politècnica de València. <https://slepc.upv.es/documentation/slepc.pdf>
- N. J. A. Sloane, editor. 2021. The On-Line Encyclopedia of Integer Sequences. <https://oeis.org>
- James Hardy Wilkinson. 1994. *Rounding errors in algebraic processes*. Courier Corporation, Mineola, NY.
- Zenodo 2021. Software used in 'Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake'. <https://doi.org/10.5281/zenodo.4701708>