



# Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake

JUSTIN CRUM, University of Arizona, USA

CYRUS CHENG and DAVID A. HAM, Imperial College London, UK

LAWRENCE MITCHELL, Durham University, UK

ROBERT C. KIRBY, Baylor University, USA

JOSHUA A. LEVINE and 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.

CCS Concepts: • **Mathematics of computing** → **Mathematical software performance; Solvers; Partial differential equations; Discretization;**

Additional Key Words and Phrases: FEM, finite element method, trimmed serendipity, PDEs, firedrake

## ACM Reference format:

Justin Crum, Cyrus Cheng, David A. Ham, Lawrence Mitchell, Robert C. Kirby, Joshua A. Levine, and Andrew Gillette. 2022. Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake. *ACM Trans. Math. Softw.* 48, 1, Article 8 (February 2022), 19 pages.  
<https://doi.org/10.1145/3490485>

---

## 1 INTRODUCTION

Over the past 15 years, conforming finite element methods for Hodge-Laplace-type problems on simplicial and cubical meshes have been analyzed and categorized using the mathematical theory

---

This work was supported by National Science Foundation (NSF) Collaborative Research Awards No. DMS-1913094 and No. DMS-1912653 and by U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award No. DE-SC-0019039.

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



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2022 Copyright held by the owner/author(s).

0098-3500/2022/02-ART8

<https://doi.org/10.1145/3490485>

of **Finite Element Exterior Calculus (FEEC)** [Arnold et al. 2015, 2006, 2010]. While this effort was initially focused on placing related theoretical results under a single, unified mathematical framework, it also exposed the frontier of knowledge and spawned a wealth of spin-off projects that improved awareness, understanding, and implementation of the methods described. Among these follow-on projects was a revisiting of the notion of “serendipity” finite elements, a particular type of finite element method dating back to the 1970s. Named for its seemingly too-good-to-be-true computational benefit, a serendipity finite element method converged to the correct solution of a **partial differential equation (PDE)** at an equivalent rate but with fewer degrees of freedom than the corresponding tensor product method. The simplest and most well-known serendipity method replaces the quadratic 9-node square element with a “serendipity” 8-node element that has no interior degree of freedom, but still converges at a quadratic rate in the appropriate sense.

The core idea of serendipity elements, i.e., reducing degrees of freedom from tensor product methods without reducing the accuracy of the method, floated around computational engineering communities until it found a resurgence under the FEEC framework. In a seminal work by Arnold and Awanou [Arnold and Awanou 2011] the scalar-valued serendipity elements were formalized as providing approximation on a space of polynomials  $\mathcal{S}_r$  that nests between maximum-degree  $r$  polynomials  $\mathcal{P}_r$  and tensor product degree  $r$  polynomials  $\mathcal{Q}_r$ . In a subsequent work [Arnold and Awanou 2014], this notion was extended using FEEC to introduce “serendipity vector elements” to the literature—analogs of the famous Nédélec elements [Nédélec 1980, 1986]—which led to the widely circulated Periodic Table of Finite Elements [Arnold and Logg 2014].

Despite excitement at these developments, implementing serendipity elements and realizing any potential computational benefits proved to be a significant challenge. While creating data structures for “all polynomials up to degree  $r$ ” is straightforward, simply trying to write down the approximation spaces used by serendipity elements—especially the vector-valued elements—requires a substantial amount of mathematical notation and explanation. As a consequence, the serendipity elements have never been implemented or rigorously studied beyond a few special use cases.

Recent work on a closely related family of methods—called *trimmed serendipity elements* [Gillette and Kloefkorn 2019]—has opened the door to potential widespread usage and benefit from the notions of serendipity theory just described. The trimmed serendipity spaces are identical to the serendipity spaces in the scalar-valued cases but are distinct in the vector-valued cases where, notably, they have fewer degrees of freedom than the serendipity elements of the same order. Since the computational motivation for serendipity methods is entirely about reducing degree of freedom count while preserving approximation power, the smaller dimensionality of the trimmed serendipity spaces obviates the need to implement the vector-valued “non-trimmed” serendipity elements.

Implementation of trimmed serendipity spaces is feasible by merging two independent efforts: the systematic definition of computational basis functions for these methods from work by Gillette, Kloefkorn and Sanders [Gillette et al. 2019] and the easily extensible open-source finite element software package Firedrake [Rathgeber et al. 2016]. The Unified Form Language [Alnæs et al. 2014; Logg et al. 2012], inspired by FEEC, provides a common backbone for translating the basis functions from their formal statement into a code structure using a well-established, high level interface. We explain the idea of the implementation by walking through the process of discretizing a PDE and selecting a finite element method for approximating its solution.

*Motivating example.* Listing 1 provides a snippet of code that a user could write to approximate a solution to the mixed Poisson equation on the domain  $\Omega := [0, 1] \times [0, 1]$  with boundary  $\Gamma$ . The



Fig. 1. The degree 2 RTCF tensor product element (Raviart-Thomas  $H(\text{curl})$  elements on quadrilaterals, [Raviart and Thomas 1977]) (left) used in Listing 1. The RTCF 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.

formal problem statement of the continuous weak form in this case is: find  $\sigma \in \Sigma := H(\text{div})$  and  $u \in V := L^2$  such that

$$\begin{aligned} \int_{\Omega} (\sigma \cdot \tau + \nabla \cdot u \tau) \, dx &= 0 & \forall \tau \in \Sigma, \\ \int_{\Omega} \nabla \cdot \sigma v \, dx &= - \int_{\Omega} f v \, dx & \forall v \in V. \end{aligned} \quad (1)$$

We assume homogeneous Dirichlet boundary conditions so that the Dirichlet data in  $u$  vanishes on  $\Gamma$ . Solving the discretized version of Equation (1) requires choosing a suitable pair of finite element spaces to create a stable method. On a mesh of squares, the typical stable pair of tensor product finite elements would be RTCF and DQ for  $H(\text{div})$  and  $L^2$ , respectively. These elements are visualized in Figure 1 and are part of the tensor product family of elements. We demonstrate the 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, "RTCF", 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 and 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

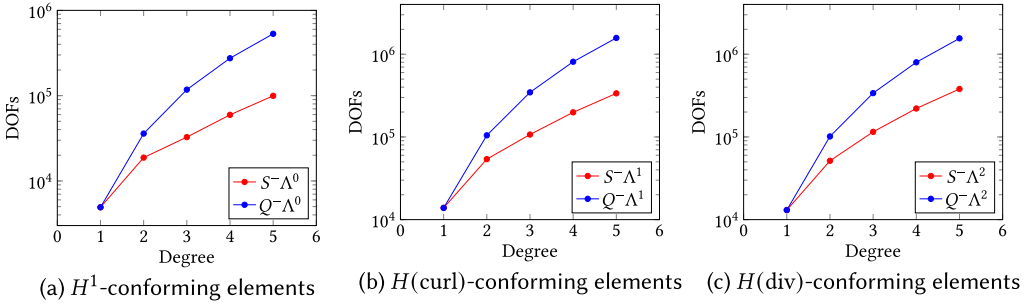


Fig. 2. Comparison of the degrees of freedom (DOFs) required for a trimmed serendipity element  $S^-\Lambda^k$  vs a tensor product element  $Q^-\Lambda^k$ , as calculated on a  $[0, 1]^3$  mesh with a total of  $16^3$  cubes. After order 1, where the element types coincide, the trimmed serendipity elements have strictly fewer degrees of freedom than the corresponding tensor product elements, with a gap that increases as the degree of polynomial approximation order increases.

suitable computational basis and connecting it to the intermediate interfaces in the included libraries.

Accordingly, we have implemented the basis functions from Gillette et al. [2019] using Firedrake’s internal coding conventions and then carried out tests of various use cases in a reliable computational framework. We show in Figure 2 how the number of **degrees of freedom (DOFs)** grow in tensor product spaces ( $Q^-$ ) versus trimmed serendipity spaces ( $S^-$ ) in various element types, for increasing degree of polynomial approximation order. Seeing how such quick back-of-the-envelope calculations might translate into significant computational savings requires the thorough implementation provided in this article.

The main contributions of this article are as follows:

- (1) We explain the implementation of trimmed serendipity elements within the Firedrake software environment and describe how users can easily employ the elements in their own code,
- (2) We validate the implementation of the trimmed serendipity finite elements by confirming they attain the theoretical bounds that are predicted in terms of convergence rates, and
- (3) We 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

Among its many advantages, FEEC [Arnold et al. 2015, 2006, 2010] 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  $\mathcal{S}_r^-\Lambda^k$ . Thus, trimmed serendipity elements represent the cheapest possible way to get an order  $r$  conforming finite element method, if cost is only measured in terms of number of degrees of freedom.

To test if this benefit translated to computational speedups, 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 article.

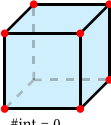
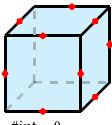
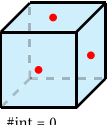
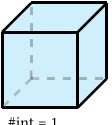
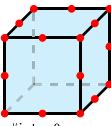
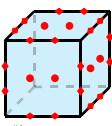
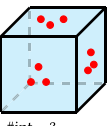
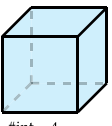
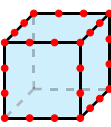
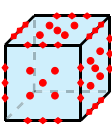
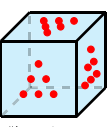
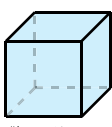
**2.1.1 Scalar Trimmed Serendipity Elements.** The scalar-valued trimmed serendipity elements that are represented by 0-forms are used as the shape functions for an  $H^1$ -conforming finite element space. These are denoted by  $\mathcal{S}_r^-\Lambda^0$  and are identical to the scalar-valued serendipity 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  $\mathcal{S}_r\Lambda^0$  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  $\mathcal{S}_r^-\Lambda^n(\square_n)$ , and here we have the equality  $\mathcal{S}_r^-\Lambda^n(\square_n) = \mathcal{S}_{r+1}\Lambda^n(\square_n)$ . 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  $\mathcal{S}_r^-\Lambda^n(\square_n) = \mathcal{P}_{r+1}\Lambda^n(\square_n)$ .

**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  $\mathcal{S}_r^-\Lambda^1(\square_2)$  bear close relation to the Arbogast-Correa elements [Arbogast and Correa 2016], as explained in Gillette and Kloefkorn [2019, Prop 2.2]. In 3D, the vector valued spaces  $\mathcal{S}_r^-\Lambda^1(\square_3)$  and  $\mathcal{S}_r^-\Lambda^2(\square_3)$  were also characterized by Cockburn and Fu [2017], as explained in Gillette and Kloefkorn [2019, Property 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

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

	$S_r^- \Lambda^0(\square_3)$	$S_r^- \Lambda^1(\square_3)$	$S_r^- \Lambda^2(\square_3)$	$S_r^- \Lambda^3(\square_3)$
$r = 1$	 #int = 0 <span style="float: right;">8</span>	 #int = 0 <span style="float: right;">12</span>	 #int = 0 <span style="float: right;">6</span>	 #int = 1 <span style="float: right;">1</span>
$r = 2$	 #int = 0 <span style="float: right;">20</span>	 #int = 0 <span style="float: right;">36</span>	 #int = 3 <span style="float: right;">21</span>	 #int = 4 <span style="float: right;">4</span>
$r = 3$	 #int = 0 <span style="float: right;">32</span>	 #int = 0 <span style="float: right;">66</span>	 #int = 9 <span style="float: right;">45</span>	 #int = 10 <span style="float: right;">10</span>

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 “# int =”. The total degree of freedom count for each element is shown to its right.

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 2018] and is in agreement with the general formulae presented in Gillette and Kloefkorn [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  $\mathcal{S}_r^-\Lambda^1(\square_3)$ —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. The subsets  $\tilde{F}$  and  $\tilde{I}$  are extra sets of basis functions defined on the faces or the interior that follow a different pattern in their definitions than those of the functions in  $E$ ,  $F$ , or  $I$ .

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 Equation (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 Equation (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 Equation (3) does not 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 Figure 3.

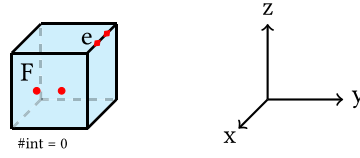


Fig. 3. The reference cube  $[0, 1]^3$  is shown, with coordinate axes indicated. The edge  $e$  lies in the intersection of the planes  $y = 1$  and  $z = 1$ . To ensure proper continuity, basis functions associated to  $e$  must vanish on all edges of the cube *except*  $e$ . Examples of such functions 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 denoted `leg`. 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 computed at the midpoint between two vertices and 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 Figure 3). The rest of the process for defining the 2-form basis functions is similar to the process for the 1-form basis functions.

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 utilize 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 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  $H(\text{div})$



Table 2. A Translation between FEEC and Firedrake Usage Names for Tensor Product and Trimmed Serendipity Elements

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

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.

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.

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 ...

```

The experiments were performed either on a single cluster compute node with 2x AMD EPYC 7642 48-core (Rome) processors (2.4 GHz) and 512 GB of memory running CentOS 7 or a similar node with 3 TB of memory. The 2D experiments were all done using the 512 GB node, while in 3D, the fourth-order experiments were run on the 3 TB 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  $u \in V \subset H(\text{curl})$  such that

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

for all  $v \in V$ . For our experiments, we choose  $V$  to be  $H(\text{curl})$  spaces for the proper dimension (one of the spaces RTCE, NCE, or  $S_{\text{minusCur1}}$ ) 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  $S_{\text{minusCur1}}$  or  $S_{\text{minusDiv}}$  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  $L^2$  rates of approximation as mesh size decreases, as well as comparing relative efficiencies of  $S_r^-$  and  $Q_r^-$ . 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 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 Figure 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  $S_r^-$  and  $Q_r^-$  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. The overall results from the projection problem show that tensor product and trimmed serendipity elements give similar levels of error.

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 formulation and the mixed formulation of the Poisson problem. We solve the primal weak formulation 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 weak formulation of the 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 formulation employs  $H^1$  elements, using  $S$  for  $S_r^- \Lambda^0$  and Lagrange for  $Q_r^- \Lambda^0$ .

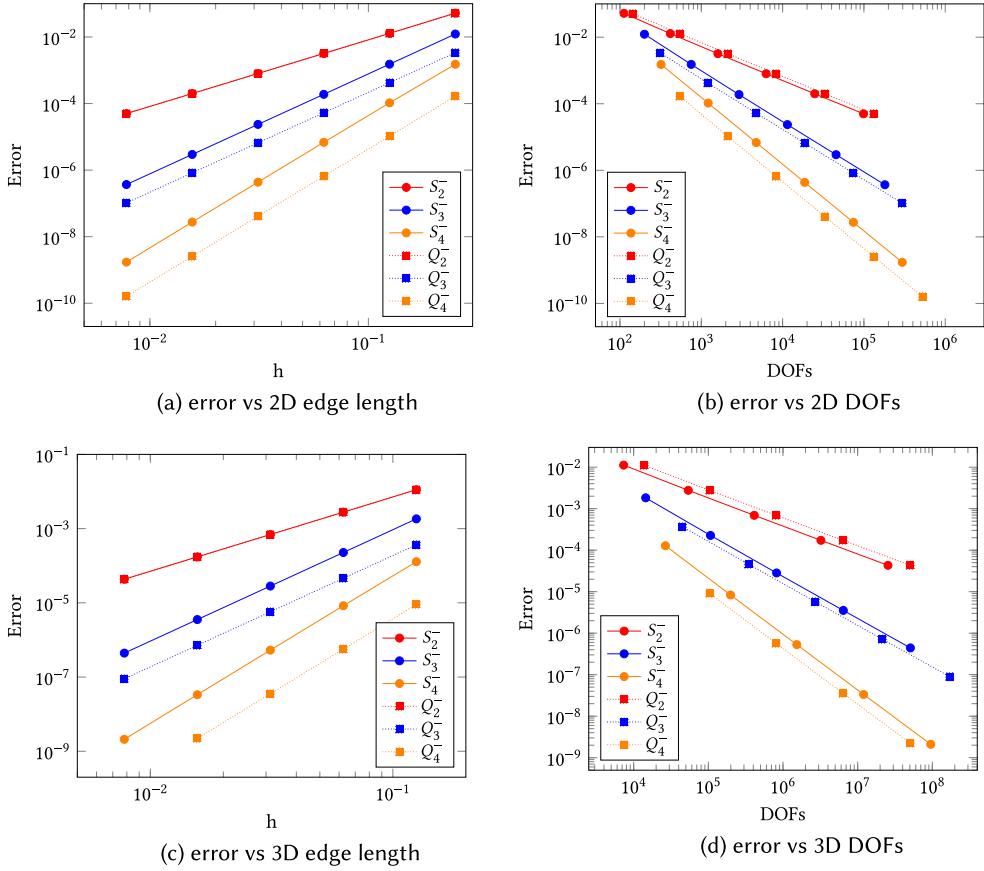


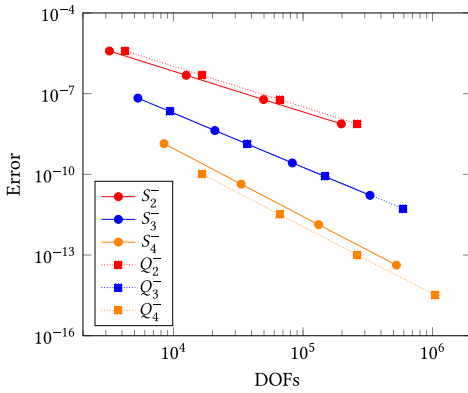
Fig. 4. Results of solving an  $L^2$  projection problem using trimmed serendipity and tensor product  $H(\text{curl})$  elements in both 2D (top row) and 3D (bottom row). Experiments were ran for  $k$ -forms for  $k = 0, 1, 2, 3$  in 2D and 3D, however, only the 1-forms in 3D are displayed here. In every case, the trimmed serendipity element trendline has the same slope as its tensor product counterpart, as expected by the theory. In panels (a) and (c), the order 2 elements only show one trendline as they differ only slightly in error values and have the same edge lengths.

The mixed formulation of the 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

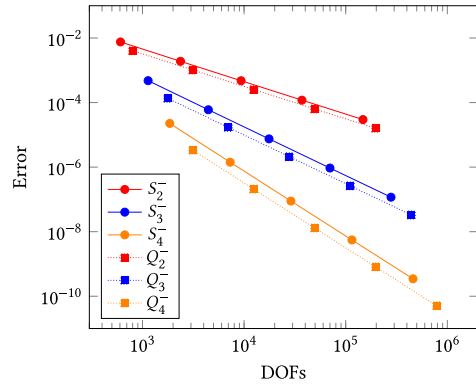
$$\begin{aligned}\sigma - \nabla u &= 0, \\ \nabla \cdot \sigma &= -f, \\ u|_{\partial\Omega} &= 0.\end{aligned}$$

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

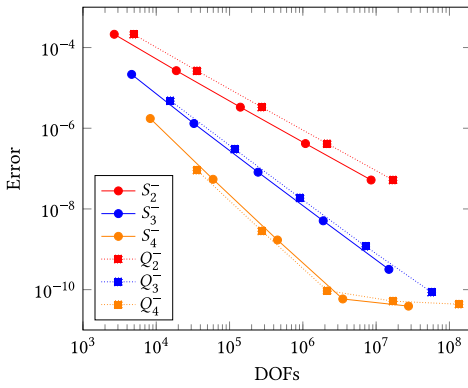
These equations are discretized and solved using a suitable *pair* of finite elements—one of  $H(\text{div})$  type and one of  $L^2$  type. We use  $(Q_r^-\Lambda^{n-1}, Q_r^-\Lambda^n)$  and  $(S_r^-\Lambda^{n-1}, S_r^-\Lambda^n)$  for dimensions  $n = 2$  and  $n = 3$ . Note that the mathematical 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



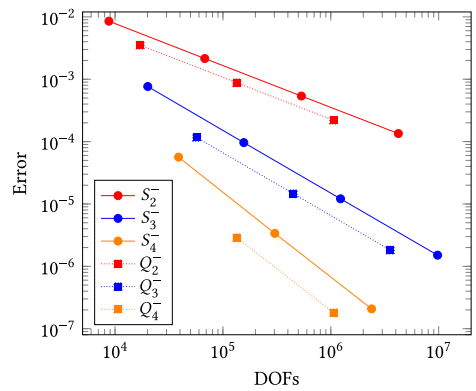
(a) 2D primal Poisson convergence analysis.



(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.

sparse direct LU factorization using iterative refinement to attain high accuracy in the solvers and allow us to focus on analyzing the elements instead of confounding variables. At high degree and fine mesh resolutions, we noticed that both the tensor product and trimmed serendipity elements would hit a floor in error values that was unexpected. At lower degrees or coarser meshes, this was unnecessary, but we chose to keep the solver parameters the same to be consistent throughout the experiments. The exact details of the MUMPS configuration can be found both in the zenodo link and in the appendix in order for results to be reproducible.

The empirical convergence results for the primal and mixed formulations of the Poisson problems can be seen in Figures 5(a)–5(d). In each of the subfigures of Figure 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 formulation in 2D and 3D, the trimmed serendipity elements perform similar to 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 essentially the same number of DOFs.

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

Table 3. Expressing the Number of Nonzero Entries in the Matrices Used to Compute Solutions to the Primal and Mixed Formulations of the Poisson Problem

$\mathcal{Q}_4^-$ Elements			
Primal $n = 2$	Primal $n = 3$	Mixed $n = 2$	Mixed $n = 3$
381,825	143,992,308	2,096,704	989,178,624
$5.51 \times 10^{-6}$	$4.99 \times 10^{-7}$	$3.38 \times 10^{-6}$	$2.18 \times 10^{-7}$
$\mathcal{S}_4^-$ Elements			
Primal $n = 2$	Primal $n = 3$	Mixed $n = 2$	Mixed $n = 3$
156,625	17,148,900	848,624	107,771,596
$8.97 \times 10^{-6}$	$1.39 \times 10^{-6}$	$4.01 \times 10^{-6}$	$2.98 \times 10^{-7}$

The data shown here represents order 4 elements, where the meshes are either  $128^2$  or  $64^3$  depending on the dimension of the space. The first row of each half indicates the number of nonzero entries, while the second row of each half indicates the proportion of the number of nonzero entries.

graphs, we see good evidence in Figures 6(a) and 6(b), that the trimmed serendipity and tensor product elements compute solutions at a similar speed based on the number of DOFs. In Figure 6(b), we see that for a given error level, trimmed serendipity elements require less time. The overall time required being dependent upon on the number of DOFs rather than the element type is seen again in Figure 6(c). Further evidence of this is seen in Figure 6(d). Similar to the previous analysis of DOFs vs error for the mixed formulation, 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. The sparsity of the matrices involved in the order 4 elements can be seen in Table 3.

### 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):

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

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].

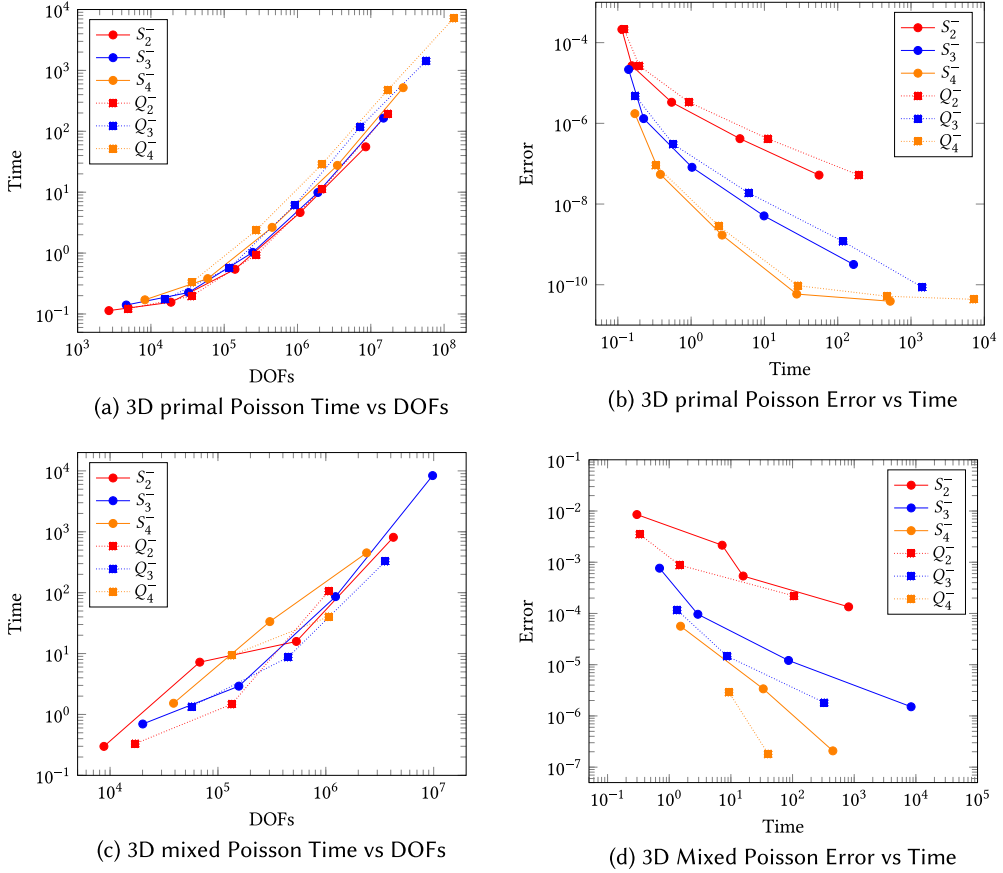


Fig. 6. Analyzing timing data for primal and mixed Poisson problems using trimmed serendipity and tensor product elements. Error here is calculated as the  $L^2$  error between the exact solution and the approximate solution found using the corresponding finite element space.

In Table 4, 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.

Note that the convergence rates are computed by

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

Based on 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.



Table 4. A Comparison of How  $Q_2^-$  and  $S_2^-$  Finite Elements Solve the Maxwell Cavity Resonator Eigenvalue Problem,  $\langle \text{curl}(F), \text{curl}(E) \rangle = \omega^2 \langle F, E \rangle$ 

$Q_2^- H(\text{curl})$ 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	1,944	13,872	104,544	811,200
EPS solve time per iteration	0.01565225	0.0743845	1.0484236	7.6186526
$S_2^- H(\text{curl})$ 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	1,080	7,344	53,856	411,840
EPS solve time per iteration	0.01288725	0.0309768	0.401663	4.1996873

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.

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 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. However, 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

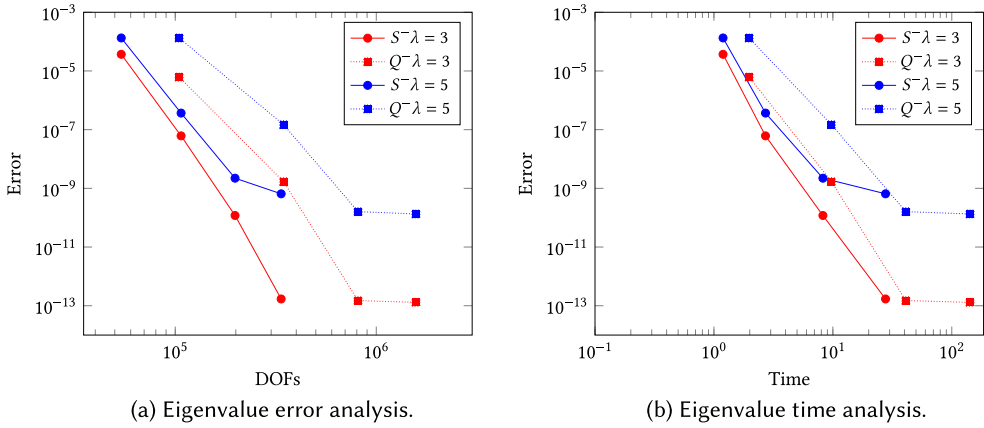


Fig. 7. Results for solving for  $\lambda = 3$  and  $\lambda = 5$  using Firedrake and SLEPc by increasing the order from 2 to 5. Error is calculated as the absolute value of the error between the actual eigenvalue and the approximated eigenvalue.

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.

The error results shown in Figure 7(a) 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 formulation of the Poisson problem where the DOFs vs Error trendline for trimmed serendipity was generally above the trendline for tensor product elements. The timing results in Figure 7(b) showed that the timing requirements for both trimmed serendipity and tensor product elements were similar, with trimmed serendipity generally requiring a little bit less time for a given error value.

## 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  $Q_r^-$ , 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 formulation of the 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 formulation 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 approximation order, it can also be beneficial to compare the two elements based on the DOFs that they require. Consider the 3D mixed formulation of the Poisson problem while focusing on DOFs vs Error given in Figure 5(d). 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 Figure 6(d) 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 Figure 7(a) 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 104,544 DOFs yielding an error of  $1.33 \times 10^{-4}$  for  $\lambda = 5$ , while  $S_3^-$  required 106,896 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 s while the  $Q_2^-$  required 1.98 s.

Our computational findings suggest that trimmed serendipity elements could be particularly beneficial at improving accuracy for compute-bound 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 article have been archived on Zenodo [2021].

## A APPENDIX: SOLVER CONFIGURATIONS

The solver configurations for the primal and mixed Poisson formulations can be found below.

Listing 3. An example of some solver parameters that we can use for the 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  ...

```

---

## 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. Softw.* 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. *Found. Comput. Math.* 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. Retrieved from <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: Math. Model. Numer. Anal.* 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. *Adv. Water Res.* 34, 9 (2011), 1124–1139. <https://doi.org/10.1016/j.advwatres.2011.04.013>
- Kikuchi Fumio. 1987. Mixed and penalty formulations for finite element analysis of an eigenvalue problem in electromagnetism. *Comput. Methods Appl. Mech. Eng.* 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. *SMAI J. Comput. Math.* 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 J. Sci. Comput.* 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. <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 Cimrman, and Anthony Scopatz. 2017. SymPy: Symbolic computing in python. *PeerJ Comput. Sci.* 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, 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 J. Sci. Comput.* 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. Sistemas Informàtics i Computació, Universitat Politècnica de València. Retrieved from <https://slep.upv.es/documentation/slep.pdf>.
- Neil J. A. Sloane. 2018. The on-line encyclopedia of integer sequences. *Notices Amer. Math. Soc.* 65, 09 (Oct. 2018), 1. <https://doi.org/10.1090/noti1734>
- 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://zenodo.org/record/4701708#.YbOR473MKUk>.

Received April 2021; revised September 2021; accepted October 2021