Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake

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

An abstract about Firedrake and FEM here.

ACM Reference Format:
Cyrus Cheng, Justin Crum, Andrew Gillette, David Ham, Robert Kirby, Joshua A. Levine, and Lawrence Mitchell. 2018. Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake. 1, 1 (January 2018), 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

2 BACKGROUND ON SERENDIPITY AND TRIMMED SERENDIPITY ELEMENTS

2.1 2D Elements

(1) Scalar (classical = Arnold-Awanou = $\mathcal{S}_r\Lambda^0(\mathbb{R}^2)$)
(2) Vector Serendipity (BDM = Arnold-Awanou = $\mathcal{S}_r\Lambda^1(\mathbb{R}^2)$)
(3) Vector Trimmed Serendipity (Arbogast-Correa = Gillette-Kloefkorn = $\mathcal{S}_r^-\Lambda^1(\mathbb{R}^2)$)
(4) Direct (Arbogast-Tao / Arbogast-Correa)

2.1.1 *Scalar (classical = Arnold-Awanou = $\mathcal{S}_r\Lambda^0(\mathbb{R}^2)$).*

2.1.2 *Vector Serendipity (BDM = Arnold-Awanou = $\mathcal{S}_r\Lambda^1(\mathbb{R}^2)$).*

2.1.3 *Vector Trimmed Serendipity (Arbogast-Correa = Gillette-Kloefkorn = $\mathcal{S}_r^-\Lambda^1(\mathbb{R}^2)$).*

2.1.4 *Direct (Arbogast-Tao / Arbogast-Correa).*

Authors' addresses: Cyrus Cheng, cyrus.cheng15@imperial.ac.uk, Imperial College, London, United Kingdom; Justin Crum, jcrum@math.arizona.edu, University of Arizona, Tucson, Arizona; Andrew Gillette, University of Arizona, Tucson, Arizona, agillette@math.arizona.edu; David Ham, Imperial College, London, United Kingdom, david.ham@imperial.ac.uk; Robert Kirby, Baylor University, Waco, Texas, robert_kirby@baylor.edu; Joshua A. Levine, University of Arizona, josh@email.arizona.edu; Lawrence Mitchell, Durham University, Durham, United Kingdom, wence@gmx.li.

Manuscript submitted to ACM

2.2 3D Elements

(1) Scalar (classical = Arnold-Awanou = $\mathcal{S}_r\Lambda^0(\mathbb{R}^3)$)
(2) Vector serendipity (Arnold-Awanou = $\mathcal{S}_r\Lambda^1(\mathbb{R}^3)$ and $\mathcal{S}_r\Lambda^2(\mathbb{R}^3)$
(3) Vector trimmed serendipity (Gillette-Kloefkorn = $\mathcal{S}_r^-\Lambda^1(\mathbb{R}^3)$ and $\mathcal{S}_r^-\Lambda^2(\mathbb{R}^3)$)

3 BUILDING CAPACITY FOR SERENDIPITY ELEMENT TYPES IN FIREDRAKE

Firedrake is a python package that is made to solve finite element problems. It does this by tying together many packages in a cohesive manner. For implementing a new finite element, we can focus on a few specific packages, namely FIAT, FInAT, UFL, and TSFC.

First, we use the UFL package. This is a symbolic math package that implemented the Unified Form Language in Python. Specifically, UFL allows Firedrake to specify a PDE in weak form that we then want to solve. Next, FIAT (FInite element Automatic Tabulator) allows a simple Ciarlet implementation of finite elements that makes the process of adding in new types of elements a relatively simple task. On the other hand, FInAT (FInAT Is not A Tabulator) is a way to pass symbolic for evaluation of finite elements. What FInAT gives out can then be handled by a form compiler. The form compiler that Firedrake uses is TSFC (Two Stage Form Compiler), and is used to take the high level, weak form PDE and turn it into low level code that can be solved.

Implementing a new element, like the trimmed Serendipity elements in 2 and 3D, then becomes a matter of adding in a new element code to FIAT, telling UFL what to expect when we call the new element (i.e., what type of shapes does it work on, what dimensions does it work in, what type of element it is), and connecting the dots in FInAT and TSFC. Starting off in the FIAT portion of the code, we create a new element by making a call to the FiniteElement class. In this class, we give the relevant information for describing a reference element of the type we're choosing. For example, we need to give it the number of degrees of freedom on each edge and face, and then we need to build and assign basis functions to each of the degrees of freedom. Once done, the rest of the code is designed for passing around the new element from library to library. We import the fiat element to FInAT so that it knows of the existence of the new element. Then we tell TSFC where to find the new element, and give it a name that a Firedrake user can give when calling the element. Finally, we give UFL the directions of what sort of elements we are allowed to use the new finite element on.

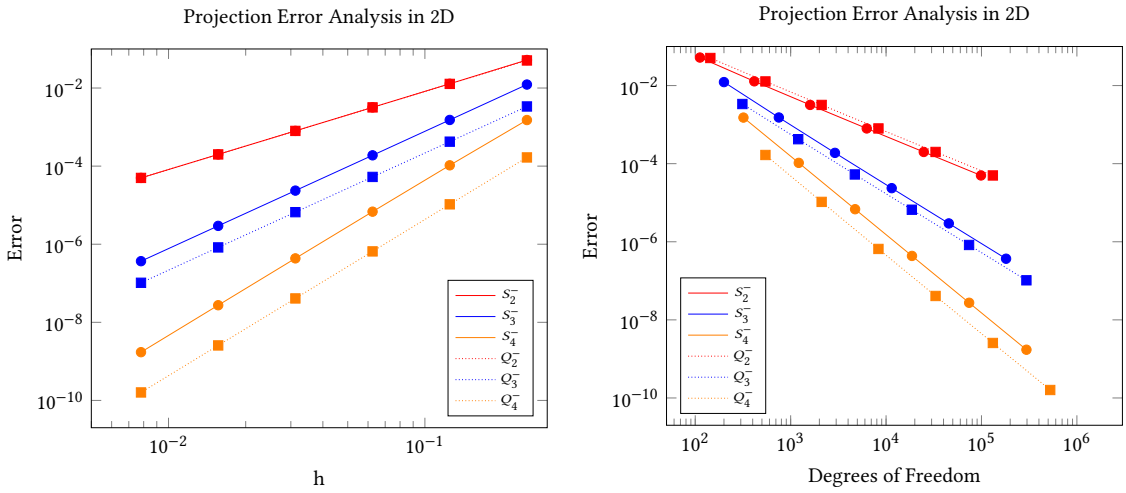
4 EXPERIMENTS

The following experiments were designed to show the benefits and costs to using trimmed Serendipity elements in comparison to traditional tensor product elements. We used a basic projection example to check implementation, as well as a primal Poisson problem (to test scalar elements), a mixed Poisson problem (to test H(div) elements), and a cavity resonator problem (to test H(curl) elements). Note that the cavity resonator problem was done only in 3D, as in 2D, the H(curl) elements are just a rotation of the H(div) elements.

4.1 Projection

We solve the projection problem to give a baseline of expectations for the elements.

GET THE EXACT PROBLEM THAT THE PROJECTION PROBLEM IS SOLVING.



(a) 2D edge length vs error

(b) 2D degrees of freedom vs error

(c) 3D edge length vs error

(d) 3D degrees of freedom vs error

Fig. 1. Results of solving projection problem using 2D and 3D S^- curl elements and tensor product curl elements.

First, we graphed the errors from computing projections using trimmed Serendipity and tensor product elements in 2D, shown in Figure 1a. Since we expect that the order of convergence for both of these elements should be the same (only off by a constant factor), we look for the lines representing trimmed Serendipity and tensor product elements at a given order to be parallel.

After checking that it is converging at the right rate, we then want to analyze the data from a standpoint of memory usage. To do this, we plot degrees of freedom vs error, as seen in Figure 1b. Overall, the error given by trimmed Serendipity elements is higher than the error from using tensor product elements of the same order. We see similar results in Figures 1d and 1c.

4.2 The Poisson Problem

In this section we discuss results for both the primal Poisson problem as well as the mixed Poisson problem. We solve the primal Poisson problem described below on a unit square domain Ω:

$$-\nabla^2 u = f \tag{1}$$

$$u|_{\partial\Omega} = 0 \tag{2}$$

$$\nabla u \cdot n = 0 \text{ on } \partial\Omega \tag{3}$$

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 $f(x, y, z) = 2\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z)$ with the solution being $u(x, y, z) = \sin(\pi x)\sin(\pi y)\sin(\pi z)$.

The primal Poisson problem allows us to test scalar Serendipity elements against Lagrange elements. Then we wish to also test H(div) elements. To do this, we solve a discretization of the mixed Poisson problem:

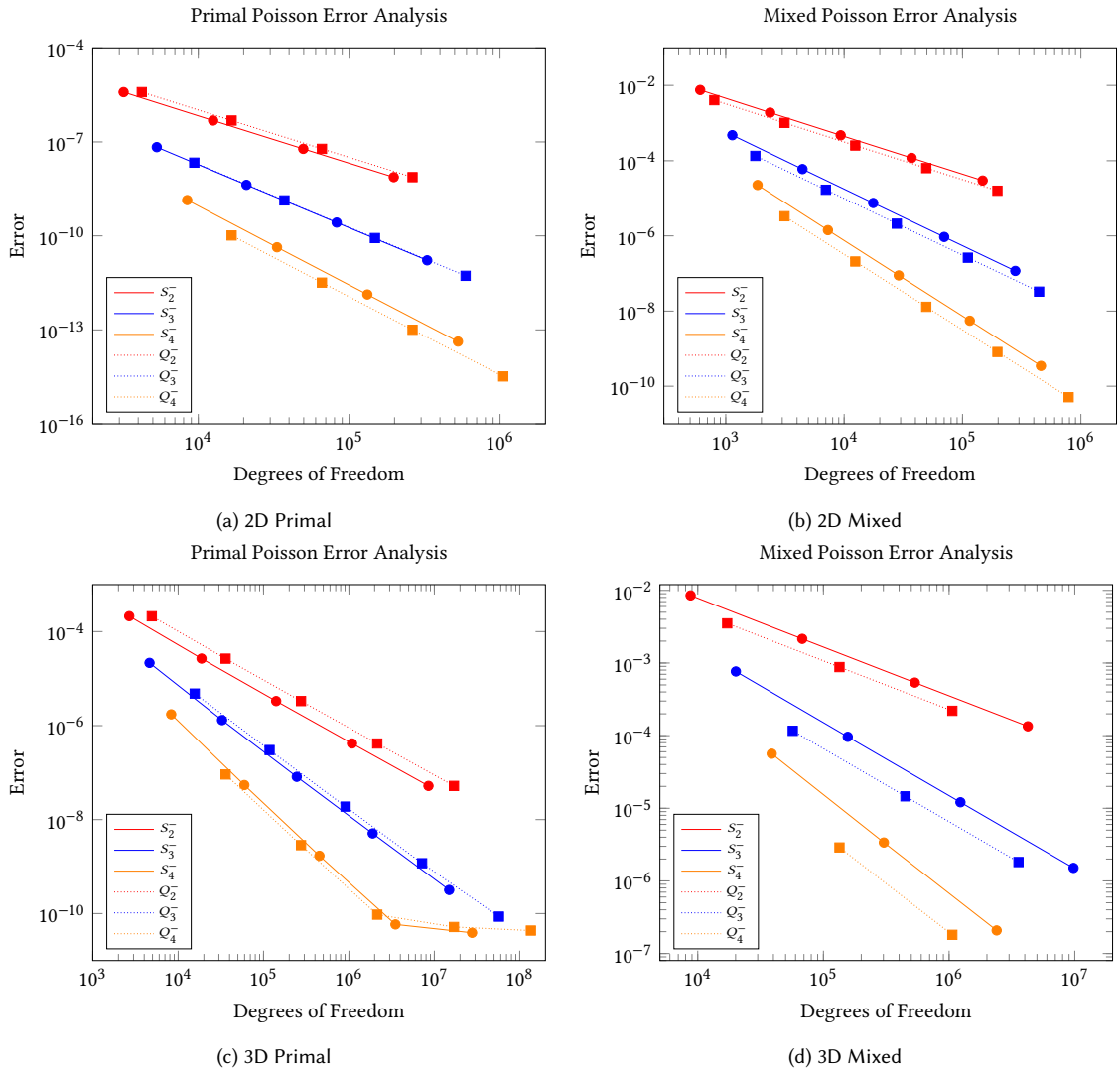
$$\begin{aligned} \sigma - \nabla u &= 0 \\ \nabla \cdot \sigma &= -f \\ u|_{\partial\Omega} &= 0 \end{aligned} \tag{4}$$

Figures 2a, 2b, 2c, and 2d give us some interesting data points to focus on. Before analyzing the results, note again that Serendipity and trimmed Serendipity elements are the same in the primal Poisson case. While these elements differ in the H(div) and H(curl) cases, our results here indicate that we likely will not see Serendiptiy elements outperforming trimmed Serendipity elements in 3D, so their implementation has been left for a future work.

In each of the figures in 2, we see that regardless of which element is better, they follow similar trajectories again, indicating that they have the same overall convergence rate. The primal Poisson problem shows us that there are scenarios where using Serendipity elements can be beneficial in terms of error depending on the number of degrees of freedom. However, in the mixed Poisson problem, we don't immediately see this benefit in either the 2D case or the 3D case.

However, we would also like to analyze timing for tensor product elements and trimmed Serendipity elements. We do this in the subfigures of 3. In 3a and 3b, we see good evidence that Serendipity elements are able to to produce better results at a faster rate. Specifically, in 3a, we can see that Serendipity elements are able to do a larger number of degrees of freedom in the same amount of time. In fig 3b we see that for a given error level, Serendipity elements require less time.

However the picture that is painted in 3c and 3d is much less clear. Overall, trimmed Serendipity elements seem to do a bit worse than the corresponding tensor product elements for the mixed Poisson problem, but it is promising that in 3c, the points tend to stay in an overall linear fashion bunched together.



(a) 2D Primal (b) 2D Mixed

(c) 3D Primal (d) 3D Mixed

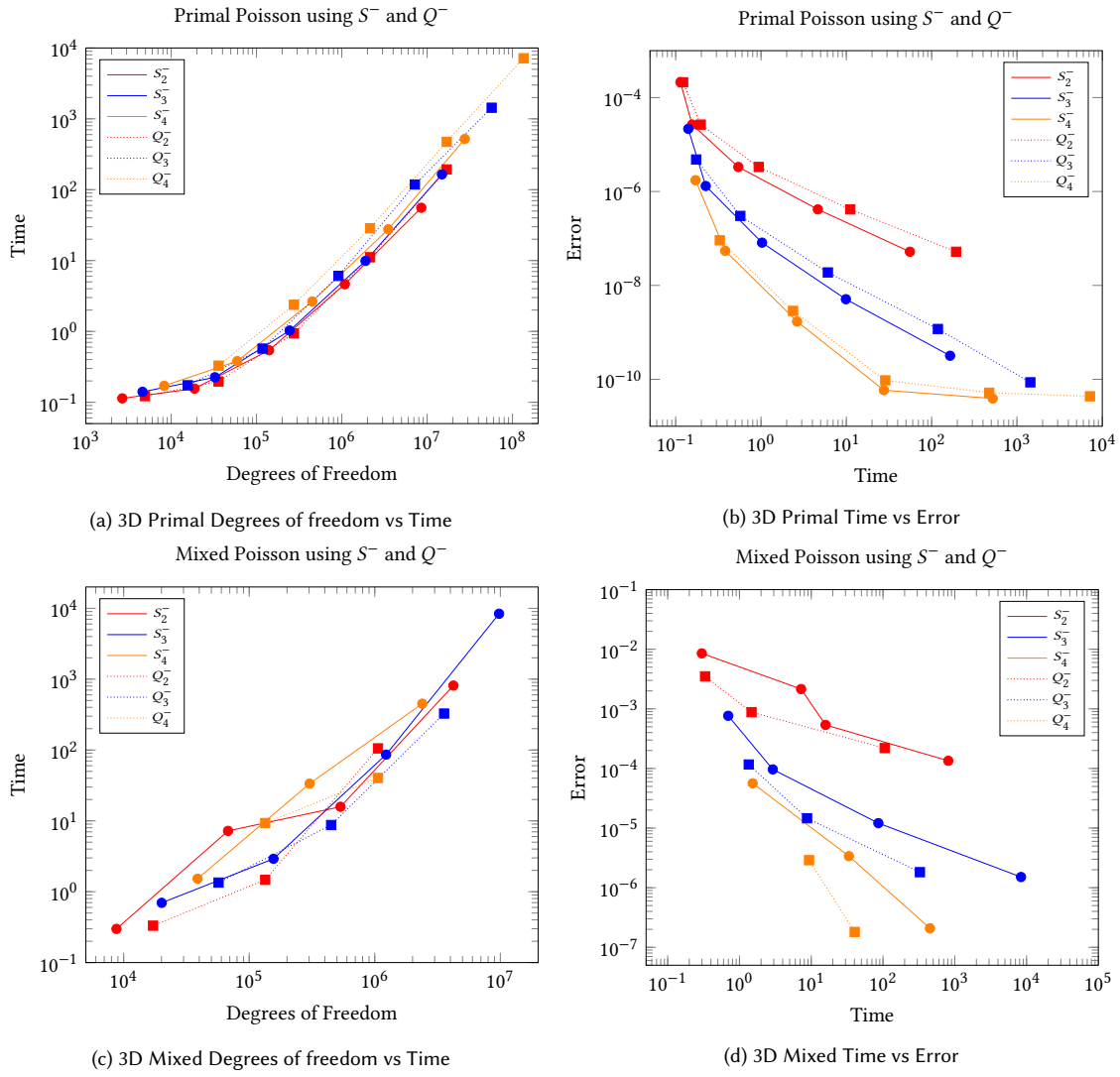
Fig. 2. An error analysis of the primal and mixed Poisson problems in 2D and 3D.

4.3 Cavity Resonator

For testing the H(curl) elements in 3D, we use the cavity resonator problem. Here, the time-harmonic Maxwell equations are applied to a domain $\Omega = [0, 1]^3$ with perfectly conducting boundary conditions, yielding an eigenvalue problem where ω represents the resonances (i.e. eigenvalues) and E represents the electric field (i.e. eigenfunctions):

$$\langle \text{curl}(F), \text{curl}(E) \rangle = \omega^2 \langle F, E \rangle \text{ for all } F \in H_0(\text{curl}). \tag{5}$$

The exact eigenvalues should follow the formula



(a) 3D Primal Degrees of freedom vs Time

(b) 3D Primal Time vs Error

(c) 3D Mixed Degrees of freedom vs Time

(d) 3D Mixed Time vs Error

Fig. 3. Analyzing timing data for primal and mixed Poisson problems using trimmed Serendipity and tensor product elements.

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

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 [2].

In Table 1, we look at the convergence rates of different eigenvalues based off solving the problem with tensor product (NCE) elements and trimmed Serendipity (S^-) elements in 3D. The table is split into two mirrored halves, the top half giving values from using NCE elements while the bottom half gives values from using S^- elements. The column labeled "Actual" represents the theoretical eigenvalue that eigenvalues in that row are converging towards. Each of the

NCE Elements				
Actual	N = 4	N = 8	N = 16	N = 32
2	2.001024	2.000066 (3.96)	2.000004 (4.04)	2.0000003 (4.00)
2	2.001024	2.000066 (3.96)	2.000004 (4.04)	2.0000003 (4.00)
2	2.001024	2.000066 (3.96)	2.000004 (4.04)	2.0000003 (4.00)
3	3.001536	3.000098 (3.97)	3.000006 (4.03)	3.0000004 (4.02)
3	3.001536	3.000098 (3.97)	3.000006 (4.03)	3.0000004 (4.02)
5	5.030601	5.002081 (3.88)	5.000133 (3.97)	5.000008 (4.06)
5	5.030601	5.002081 (3.88)	5.000133 (3.97)	5.000008 (4.06)
5	5.030601	5.002081 (3.88)	5.000133 (3.97)	5.000008 (4.06)
5	5.030601	5.0002081 (3.88)	5.000133 (3.97)	5.000008 (4.06)
6	6.031114	6.002114 (3.88)	6.000135 (3.97)	6.000008 (4.08)
6	6.031114	6.002114 (3.88)	6.000135 (3.97)	6.000008 (4.08)
6	6.031114	6.002114 (3.88)	6.000135 (3.97)	6.000008 (4.08)
8	-	-	-	-
Iterations	4	4	3	5
DOF	1944	13872	104544	811200
EPS Solve Time (seconds)	0.062609	0.297538	3.145271	38.093263
S^- H(curl) Elements				
Actual	N = 4	N = 8	N = 16	N = 32
2	2.001092	2.000066 (4.05)	2.000004 (4.04)	2.000000 (4.00)
2	2.001092	2.000066 (4.05)	2.000004 (4.04)	2.000000 (4.00)
2	2.001092	2.000066 (4.05)	2.000004 (4.04)	2.000000 (4.00)
3	3.009018	3.000586 (3.94)	3.000037 (3.99)	3.000002 (4.21)
3	3.009018	3.000586 (3.94)	3.000037 (3.99)	3.000002 (4.21)
5	5.032027	5.002097 (3.93)	5.000133 (3.98)	5.000008 (4.06)
5	5.032027	5.002097 (3.93)	5.000133 (3.98)	5.000008 (4.06)
5	5.032027	5.002097 (3.93)	5.000133 (3.98)	5.000008 (4.06)
5	5.032027	5.002097 (3.93)	5.000133 (3.98)	-
6	6.072012	6.004976 (3.86)	6.000319 (3.96)	6.000020 (4.00)
6	6.072012	6.004976 (3.86)	6.000319 (3.96)	6.000024 (3.73)
6	-	-	6.00038	6.000024 (3.98)
8	-	-	-	8.000017
Iterations	4	5	4	3
DOF	1080	7344	53856	411840
EPS Solve Time (seconds)	0.051549	0.154884	1.606652	12.599062

Table 1. A comparison of how order 2 NCE and S^- finite elements solve the Maxwell cavity resonator eigenvalue problem, $\langle \mathrm{curl}(F), \mathrm{curl}(E) \rangle = \omega^2 \langle F, E \rangle$.

$N = 4, N = 8, N = 16, N = 32$ columns represents the approximate eigenvalues calculated on a mesh of size $NxNxN$. The number in parenthesis next to approximate eigenvalues is the rate of convergence of that eigenvalue. Finally, each half has a row giving the overall degrees of freedom in the mesh at each given mesh size and another row that gives the time that the eigenvalue solver needed to find the requested number of eigenvalues.

Note that the convergence rates are computed by doing

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

and are indicated in the chart by using parentheses. We use H(curl) to solve the problems, corresponding with edge elements in 3D. Based off earlier eigenvalue works [1], we expect that the rate of convergence be double the order of the finite element used to solve the problem. This is reflected in the table relatively well for both S^- and NCE elements.

Any eigenvalue that has a − spot is to be interpreted as the eigenvalue solver did not find that specific eigenvalue in the number of iterations it required to find the first 15 requested eigenvalue-eigenvector pairs.

The experiment was done by using SLEPc in Firedrake, computing an inverted shift to a target of 3.0, then asking SLEPc for 15 eigenvalue-eigenvector pairs. SLEPc was then give a tolerance level of 1e − 7, and a couple of specific mumps parameters (icntl 14 set to 200 and icntl 13 set to 1). We ignored the eigenvalues of 1, as they correspond only to the boundary conditions.

Knowing that both elements are solving this problem in a fashion that is expected theoretically, we can analyze the rest of the results shown in this table. Investigating the error in the eigenvalues in the chart compared to the exact values, we see that NCE elements are able to get results that are up to a magnitude better near the target eigenvalue. On the other hand, this loss of accuracy from using trimmed Serendipity elements results in a significant reduction in required time to solve for the requested eigenvalues. At every mesh refinement level, trimmed Serendipity elements have nearly half the DOFs of NCE elements, and correspondingly, require about half the time to solve for the eigenvalues. At higher orders, we expect that this will be even more exaggerated.

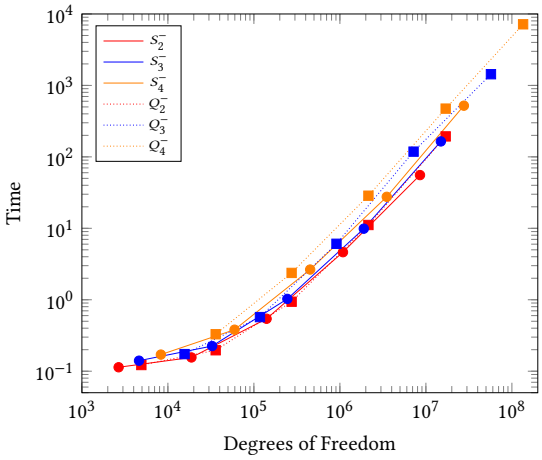
5 CONCLUSION

Each finite element has a time and place where it could be considered beneficial to use. We explored the numerical properties of trimmed Serendipity elements to refine our understanding of how they act. From the theory, we knew that trimmed Serendipity elements should be able to converge at the same rate as tensor product and Serendipity elements, while using fewer degrees of freedom overall. The plots here demonstrate that the rate of convergence is consistent with what we expect at many orders for H(curl), H(div), and L^2 elements in both 2 and 3D.

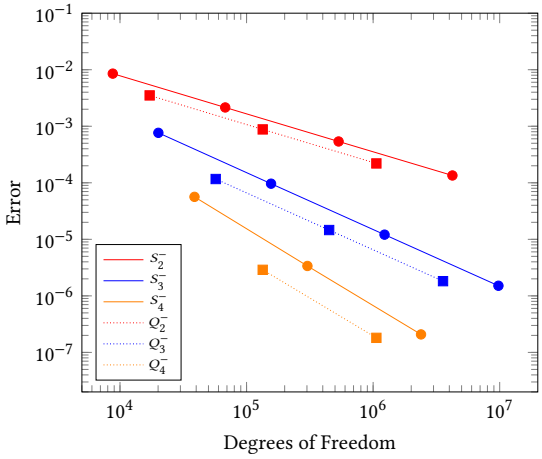
Beyond the convergence rates hitting what we expect, we were able to analyze memory usage on these problems by studying the degrees of freedom required. Trimmed Serendipity, while generally have a worse error at a given order k, also uses significantly fewer degrees of freedom. This is illustrated well in the Maxwell Cavity Eigenvalue problem in 1, where the degrees of freedom required were nearly half of what the tensor product elements used.

Another example of this is the 3D mixed Poisson problem, where we see that the trimmed Serendipity elements are able to be used at more refined meshes while the tensor product elements would need to be allotted more time on a high memory computer to be able to get results on the same sized mesh.

In general, it is clear from these results that trimmed Serendipity is not always a better choice compared to tensor product elements. However, these examples illustrates the benefit of trimmed Serendipity elements–in a setting where the mesh is fixed, the option to use a trimmed Serendipity element might give an extra way to refine a problem to increase the accuracy of a solution.



Mixed Poisson Error Analysis



REFERENCES

[1] Daniele Boffi. 2010. Finite element approximation of eigenvalue problems. *Acta Numer.* 19 (2010), 1–120.

[2] Marie E Rognes, Robert C Kirby, and Anders Logg. 2010. Efficient assembly of H(div) and H(curl) conforming finite elements. *SIAM Journal on Scientific Computing* 31, 6 (2010), 4130–4151.