# Firedrake: A complete package for solving partial differential equations using finite element methods

Justin Crum

May 19, 2020

## 1 Introduction

Firedrake is a python package solving partial differential equations through finite element methods. To this end, the Firedrake developers have attempted to implement the major methods that have been covered in the Periodic Table of Finite Elements.

Firedrake works by interfacing with a few other packages. Beyond just typical math packages for solving numerical problems like Eigen and petsc, Firedrake also makes use of packages UFL, TSFC, FIAT, and FInAT to get all the math done that it needs to do.

In this paper, we discuss the implementation of Serendipity and Trimmed Serendipity finite elements. Serendipity elements had previously been implemented in two dimensions, but not in three dimensions entirely. Trimmed Serendipity elements had not previously implemented in any dimension before this work began (within the Firedrake framework).

## 2 Sections

Probably need some words here about different things. Like what are Serendipity and/or Trimmed Serendipity elements, previous work that was done, and possibly a bit about Firedrake.

## 3 Work Done

We have done the following:

1. Implement Trimmed Serendipity elements in two dimensions.
2. Tests have started on Trimmed Serendipity elements in two dimensions with promising results.
3. In two dimensions, tests for Trimmed Serendipity have been conducted on both square/regular quad meshes as well as "arbitrary" quad meshes. The test arbitrary quad mesh that we have made use of is a trapezoidal mesh, created by taking the configuration of the square mesh and moving select vertices up or down.

4. Development of 3d Trimmed Serendipity elements is in progress.

5. Test on Trimmed Serendipity elements in three dimensions seem to indicate that this is still in the bug-testing phase. Something is definitely either wrong with the PDE being solved (less likely, but possible) or wrong with the implementation (more likely).

6. The 3d Trimmed Serendipity elements do in fact map basis functions to the proper entity (edge, face). We do see the correct number of basis functions regardless of dimension.

# 4 What to do next?

1. We would like to continue tests on 2d Trimmed Serendipity. This includes a couple of factors:
   (a) Check to make sure the correct functions are getting the correct entity (face/edge) assignment. In two dimensions, this really only matters for edge assignment.
   (b) Test 2d Trimmed Serendipity elements on an analytic test problem using numpy (somehow). Goal here is to make sure that we can recreate polynomials exactly when we expect to.
   (c) Check that the higher order portions of the 2d Trimmed Serendipity are implemented correctly–that basis functions are going to the right spot, that there are enough basis functions (this should be correct otherwise the code should break), and that the basis functions themselves are correctly implemented.

2. Continue bug testing on 3d Trimmed Serendipity elements. This includes:
   (a) Check that basis functions are being implemented properly.
   (b) Look closer at the PDE being used as a test PDE and make sure it is fitting for the type of problem we are doing.
   (c) Test 3d Trimmed Serendipity on an analytic test problem and see if we recreate the polynomials we expect it to.
   (d) If error is still bad.... Ask for a meeting with Andrew, Josh, and Rob all in the same zoom chat and figure out what else could be going wrong.

3. Implement the 3d Serendipity elements for H(div) and H(curl). These will be a little tricky to do, but will generally follow the same methodology used to implement the trimmed Serendipity elements.

4. Continue remembering that H(div) is the column that is second to last and H(curl) is the column that comes second. That is–H(div) elements mean that if I take a divergence of a vector element, I get a scalar element, whereas H(curl) elements mean that if I take a curl of a vector element, I get the next vector element.

5. Replace SminusE and SminusF in 2d fully with SminusCurl and SminusDiv fully. Currently SminusDiv is in development. The 2d portion should be able to replace SminusF entirely. The 3d portion needs more work to see why the error is bad.

# 5 Results

Here we want to give some results from tests that we have done. The following figures all depict a certain number of degrees of freedom vs error in the function and are based off a tides PDE problem where the mesh is a trapezoidal mesh.

Specifically, I have been running experiments on the tides PDE (described below) using RTC and $S^-$ elements on both square and trapezoidal meshes. The RTC spaces get paired with DQ spaces from the same row of the periodic table. The $S^-$ spaces get paired with DPC spaces from the same row of the table that Andrew has. The figures below show the results of these simulations on the uniform square meshes.

The code can be found at:

`https://github.com/Justincrum/MyFiredrakeWork/tree/master/TrimmedSerendipityTestCodes`

In the figures that follow, we should different types of degrees of freedom counts vs error. The degrees of freedom that we count are either

1. Shared degrees of freedom. These are degrees of freedom that live on edges that are shared by more than one cell of the mesh.

2. Total degrees of freedom. This is just the total number of degrees of freedom in the mesh.

3. Face degrees of freedom. These are the degrees of freedom that are attached to a single face. They are not shared by more than once cell of the mesh and do not live on the boundary.

The results that we get here are actually as expected. At higher degree polynomials, we see that the trimmed Serendipity elements do outperform the tensor product elements as the mesh gets finer. In the case of a coarser mesh, the trimmed Serendipity elements tend to get outperformed just slightly by the tensor product elements, but overall the rates of convergence are very similar.

Furthermore, in the high order polynomial setting, the mesh getting finer has the effect of the solver not being able to handle the tensor product elements well. This seems to indicate that the trimmed Serendipity elements are more robust to what type of solver is used to do these computations rather than having to tailor the solver to the granularity of the mesh and the element type.

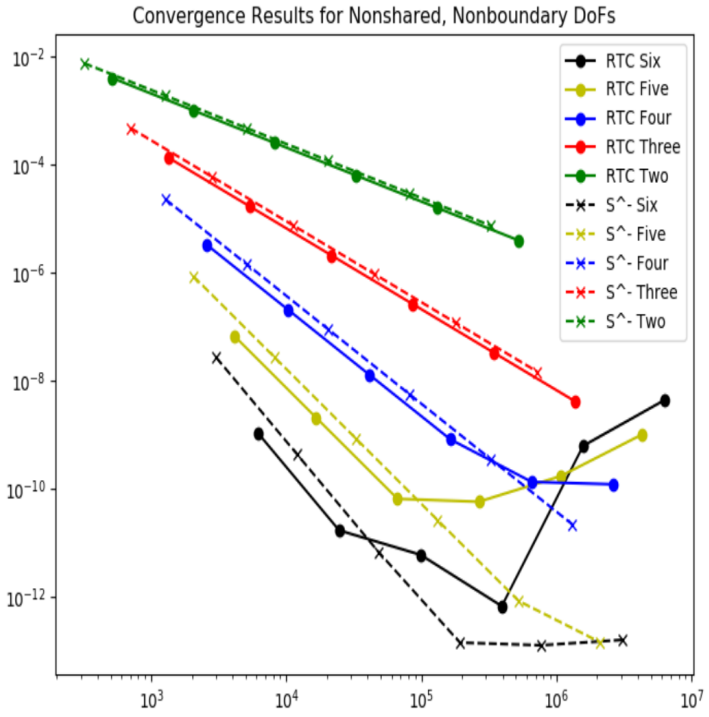


Figure 1: Showing the comparison of non-shared, non-boundary degrees of freedom vs error.

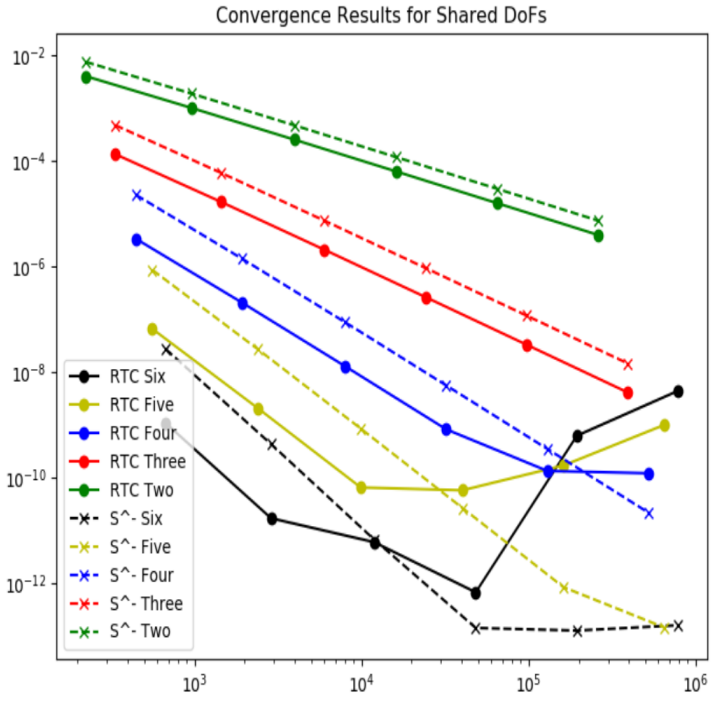


Figure 2: Showing the comparison of shared degrees of freedom vs error.

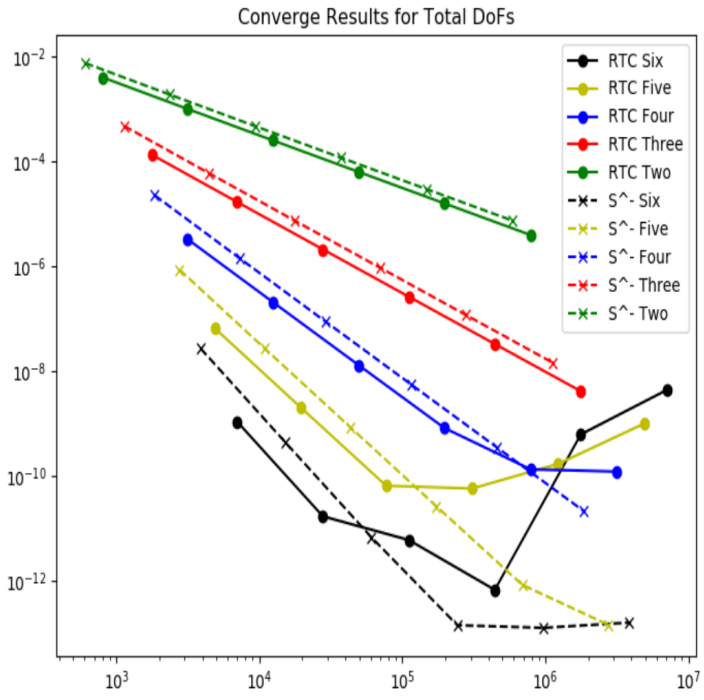


Figure 3: Comparison of total degrees of freedom vs error.