IMPLEMENTING AND TESTING EXTERIOR CALCULUS DISCRETIZATION TECHNIQUES FOR PDES

By

Justin Crum

A Dissertation Submitted to the Faculty of the

GRADUATE INTERDISCIPLINARY PROGRAM IN APPLIED MATHEMATICS

In Partial Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2022

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by: Justin Crum
titled: Implementing and Testing Exterior Calculus Discretization Techniques for PDEs

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Joshua A. Levine — Date: Feb 9, 2022

Andrew Gillette — Date: Feb 9, 2022

Moysey Brio — Date: Feb 9, 2022

Lise-Marie Imbert-Gerard — Date: Feb 10, 2022

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

We hereby certify that we have read this dissertation prepared under our direction and recommend that it be accepted as fulfilling the dissertation requirement.

Joshua A. Levine — Date: Feb 9, 2022
Department of Computer Science

Andrew Gillette — Date: Feb 9, 2022
Designated Campus Colleague, Associate (Math)

Acknowledgements

I have done the work on this dissertation as a student in the Graduate Interdisciplinary Program in Applied Mathematics under the direction of Dr. Andrew Gillette and Dr. Josh Levine. It has been supported financially by the Department of Mathematics and National Science Foundation (NSF) Collaborative Research Awards DMS-1913094 and DMS-1912653.

I would like to thank my advisors, Dr. Andrew Gillette and Dr. Josh Levine. Dr. Gillette spent 5 years working with me, teaching me, helping me find job opportunities, and he put me in a position to succeed at each step. Dr. Gillette's ideas and questions that he focuses on have been a great inspiration to me as a researcher and formed a large part of how I view applied mathematics research. Even after leaving the university, Dr. Gillette continued to mentor me, and the time he sacrificed from his family to do so will always be appreciated. Dr. Levine spent nearly as long as my advisor, and gave me the perspective of someone that uses mathematics in many exciting research areas. He made sure that I grew as a mathematician and researcher until I had the qualities he believed were deserving of a PhD. Alongside the research meetings, Dr. Levine was always ready to give guidance and mentorship when I needed it. His advice and encouragement came at moments in my degree that were crucial to my success.

Beyond my advisors, I collaborated with some very intelligent and helpful people on the Firedrake team such as Dr. Robert Kirby, Dr. Lawrence Mitchell, Dr. David Ham, Cyrus Cheng, and Dr. Thomas Gibson. In particular Dr. Kirby has been instrumental in my success on the work using trimmed serendipity finite elements in Firedrake. He was a fantastic resource any time I was stuck, and always had helpful comments for me to debug code or teach me more about finite element methods.

Additionally, I would like to thank my other committee members, Dr. Lise-Marie Imbert-Gerard and Dr. Moysey Brio. Both of these people have sacrificed time and energy to help me finish my degree and give me advice on how to make my dissertation the best it can be.

Dr. Moysey Brio believed in me enough to give me a spot in this program, and I will always be thankful for that.

Finally, a thank you to my family and friends. My wife, Taryn, has kept me moving forward and supported me more than anything I could have asked for. My parents, Linda and Kevin, have always been ready to help out in any way that they can, whether it was a meal or groceries in the pandemic, or planning something fun for a holiday. And to all my friends both in and out of the mathematics world (Alyssa, Jessica, Nick, Elsa, Chase, Jared, Aaron, and Ashlee), thank you all for helping me through hard math problems or taking my mind off of them when I needed it.

Table of Contents

List of Figures

List of Tables

ABSTRACT

Finite element exterior calculus and discrete exterior calculus have been actively used and developed in the last two decades. In this dissertation we first work in the context of finite element exterior calculus, studying the "trimmed serendipity" finite element family by building an implementation of them that allows for analyzing their properties in practice. Then, we apply the trimmed serendipity elements to the monodomain equation, an application problem of interest in cardiac modeling. Finally, we swap to the discrete exterior calculus context where we define and apply a fractional discrete exterior derivative.

By using a Python package called Firedrake, we illustrate how to implement the trimmed serendipity finite elements. Using that code, we then give a rigorous analysis of how the trimmed serendipity finite elements compare against the tensor product finite elements on a variety of toy problems. These include a projection problem, a primal formulation of the Poisson problem, a mixed formulation of the Poisson problem, and the Maxwell cavity eigenvalue problem.

After testing the elements on these toy problems, we move to applying the elements to approximating the solution of the mondomain equation. This particular equation can be changed in different ways to model how electricity flows through the heart. Depending on the exact choice of model used, it can range from a simple model that shows only the macro-scale dynamics all the way to a complicated model that incorporates micro-scale dynamics

of different ionic compounds. Using this model, we test the trimmed serendipity elements in conjunction with a recent implementation of Runge-Kutta methods in Firedrake.

Finally, we end with studying how fractional derivatives can be discretized in the setting of discrete exterior calculus. While the discrete exterior derivative and many other operators are well defined and commonly used in discrete exterior calculus, the fractional derivative operator is a little more difficult. The nonlocal nature of a fractional operator leads to issues of how to define distances between non-adjacent simplices in a mesh, which discrete exterior calculus does not normally do. After giving a definition for a fractional discrete exterior derivative, we analyze its properties and give a discussion on some of these difficulties.

CHAPTER 1

Introduction

1.1. Problem Statement

Engineers and scientists model real world phenomena using partial differential equations (PDEs) and fractional differential equations (FDEs). Most of these equations lack analytic solutions. Instead, to solve the equations involved in many of these models, researchers will use powerful computers and steadily advancing numerical methods to obtain highly accurate approximations. However, the cost of using these methods cannot be ignored, and understanding how different numerical techniques compare to one another can potentially save large amounts of computation time and power.

In this dissertation, we use two different exterior calculus discretization approaches: one to analyze a recent finite element method and the second to provide a new definition of a discrete fractional derivative. The first approach, finite element exterior calculus (FEEC), provides a framework for finite element methods in which relatively new finite elements have been defined, such as the trimmed serendipity family, and allows for a rigorous practical comparison between the trimmed serendipity elements and their predecessors. The second approach, discrete exterior calculus (DEC), provides a way of thinking about the conversion of continuous operators to discrete counterparts that we leverage for defining a new fractional derivative operator.

1.1.1. FEEC and Trimmed Serendipity Elements

The implementation and use of trimmed serendipity finite elements is led by a desire to compute approximate solutions to PDEs at a reduced cost without losing the rate of convergence

that a more computationally expensive element would provide. The cost of a finite element method can be attributed to many different factors such as the order of the element, the sparsity of matrices involved, and the number of degrees of freedom (DOFs) in the mesh. We focus on how the number of DOFs in a mesh affects the convergence rate and time required for the computer to find approximate solutions.

One important way to quantify the difference in finite element families is the use of reference elements. A reference element is one that defines the basic geometry the element is used for and the number of DOFs on the element. The family of 2D tensor product finite elements, denoted as $\mathcal{Q}_2^-\Lambda^k$, have reference elements as depicted in Fig. 1.1, and the notation will be described more in-depth in Chapter 2.

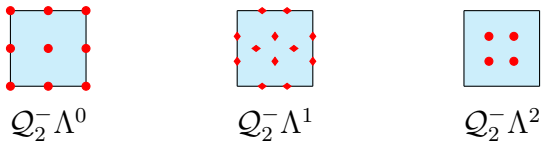


Figure 1.1. The reference elements for the second order tensor product family of finite elements, $\mathcal{Q}_2^-\Lambda^k$. Each red dot indicates a degree of freedom associated to the element.

If we remove the inner DOF from the face of the $\mathcal{Q}_2^-\Lambda^0$ element, the rate of convergence for the corresponding method does not change. The rigorous method for doing this will also reduce the number of DOFs on the inner face of $\mathcal{Q}_2^-\Lambda^1$, and results in a finite element method that theoretically converges at the same rate as the corresponding tensor product method. The resulting finite element family is called the serendipity finite elements, and they are denoted $\mathcal{S}_r\Lambda^k$. However, while the reference elements makes it clear that using $\mathcal{S}_2\Lambda^0$ instead of $\mathcal{Q}_2^-\Lambda^0$ will result in fewer DOFs, we see an increase in the total DOFs on the reference elements for $\mathcal{S}_2\Lambda^1$ and $\mathcal{S}_2\Lambda^2$. Fortunately at higher orders, the elements $\mathcal{S}_r\Lambda^1$ and $\mathcal{S}_r\Lambda^2$ will yield reductions in the total number of DOFs required, just as the $\mathcal{S}_2\Lambda^0$ elements do.

The key difference between how the serendipity elements and the tensor product elements are defined depends on how we associate DOFs to functions that we can use in computation.

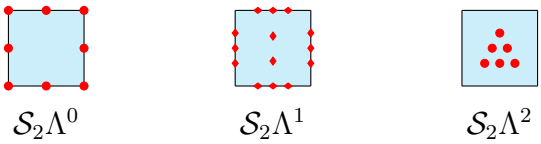


Figure 1.2. The reference elements for the second order serendipity family of finite elements, $\mathcal{S}_2\Lambda^k$.

In the case of finite elements, the set of DOFs on a reference element can be bijectively mapped to a set of polynomial functions. For example, $\mathcal{Q}_2^-\Lambda^0$ uses polynomials similar to $\{1, x, y, xy, x^2, x^2y, xy^2, y^2, x^2y^2\}$ (biquadratic polynomials).

The serendipity elements were created [10] with the idea that the elements $\mathcal{S}_r\Lambda^0$ should be contained in $\mathcal{Q}_r^-\Lambda^0$. Thus, the polynomial functions that the serendipity element associates its DOFs to must be some subset of polynomials such that any factor of x or y is at most degree r. The definition does this by applying the idea of *superlinearity*. A polynomial's superlinear degree is the largest degree of any term when only counting variables that enter the term with an exponent greater than 1. For example, xy^2 has a superlinear degree of 2, while x^2y^2 has a superlinear degree of 4. Therefore, the family $\mathcal{S}_2\Lambda^0$ does not have a DOF that corresponds with the function x^2y^2. The concept of superlinearity is applied when defining the sets of polynomial functions that are associated with the DOFs for each element type of the form $\mathcal{S}_r\Lambda^k$ for $r \geq 0$ and $0 \leq k \leq n$.

While superlinearity helps define the serendipity elements, the trimmed serendipity elements mentioned above were designed with the goal of achieving a minimal number of DOFs for each space $\mathcal{S}_r^-\Lambda^k$ such that computing an approximate solution to a PDE using the elements would still yield the same rate of convergence. Defining the sets of functions that are associated with the DOFs of the trimmed serendipity elements requires use of the Koszul operator, which we will describe in more detail in Chapter 3. Observe that with this definition, the trimmed serendipity elements have the property that $\mathcal{S}_r^-\Lambda^k \subseteq \mathcal{S}_r\Lambda^k$, with equality in the case of $k = 0$, and the elements $\mathcal{S}_r^-\Lambda^k$ can be seen in Fig. 1.3.

The comparisons we present will focus on the tensor product elements and the trimmed serendipity elements, since the (regular) serendipity elements will generally fall in the middle of the other two families. Previous work had focused on the theoretical properties of the trimmed serendipity elements, but had largely left the practical implementation and experimentation out. Thus the first goal of our work is to illustrate our method of implementing the trimmed serendipity elements in a software package. Using that implementation, we then demonstrate how the trimmed serendipity elements perform in practice on a variety of test problems by analyzing different metrics such as the convergence rate, the error, and the number of DOFs.

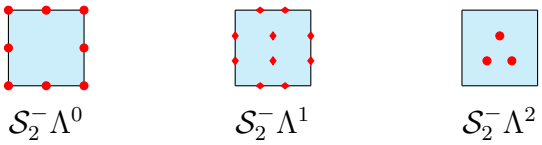


Figure 1.3. The reference elements for the second order trimmed serendipity family of finite elements, $\mathcal{S}_2^-\Lambda^k$.

1.1.2. The Monodomain Equation

Using the trimmed serendipity elements on toy test problems is the first way we verify and analyze how effective they are as a finite element family in practice. However, the monodomain equation is a setting that is of more interest for real world modeling. One particular application of the monodomain equation is studying how electricity flows through the human heart [75].

The monodomain equation is given by

$$\chi\big(C_m\frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) = \nabla\cdot\sigma\nabla u \tag{1.1}$$

where χ is the surface area to volume ratio, C_m is the specific capacitance of the cell membrane, u is the electric potential, I_{ion} is current due to flows of ions, and σ is a conductivity tensor. The function I_{ion} can be defined in many ways, from simplistic cubic functions of the potential that are useful for proof of concepts to complicated ODE systems that try to accurately represent the micro-scale dynamics in the heart.

The monodomain equation also evolves in time, unlike the toy problems we use to verify the accuracy and theoretical properties of the trimmed serendipity elements. To handle this, we use Irksome, a recent introduction to Firedrake. Irksome gives Firedrake the ability to solve ODE systems using implicit Runge-Kutta methods. Finally, due to the complexity of this problem, we require more sophisticated methods for setting up the monodomain equation in Firedrake to ensure that we can solve the resulting algebraic systems quickly enough. To do this, we use the so called Rana preconditioner [60].

When analysing the results for the monodomain equation, we focus on both numerical properties and some of the physical properties that cardiac researchers might be interested in. The numerical properties, such as the number of iterations required to solve and how long the computations take are typical. We compute the "activation time" of the solutions, which is a quantity of interest to cardiac researchers. In general, cardiac researchers refer to activation time as the amount of time it takes for tissue to return to resting potential after it has been stimulated. Thus, we will give both the numerical results of interest to mathematicians, and the practical results of interest to the cardiac modeling community.

1.1.3. DEC and Fractional Calculus

In DEC [30], we traditionally perform computations on meshes that satisfy being a "simplicial complex". A simplicial complex is a mesh made up of only p-dimensional simplices, where p ranges between 0 and n (the overall dimension of the space) such that the simplices

satisfy some nice properties. For example, 0-simplices are vertices, 1-simplices are edges, 2-simplices are triangular faces, and 3-simplices are tetrahedra. We give the formal definition of a simplicial complex later on, in Chapter 2.

Computations in DEC depend on using operators that take values on these simplices as inputs. The simplest example of a function defined on the simplicial complex would be assigning a (scalar) value to each vertex in the mesh. For DEC, this would then be represented by a vector of length equal to the number of vertices in the mesh. Similarly, we could represent a function by values assigned to edges, or to faces, or to any single type of p-simplex in the mesh.

Moving between the different types of functions that we can associate with the different values of p is where the first interesting set of DEC operators comes into play. The discrete exterior derivative allows movement of a function on p-simplices to $(p+1)$-simplices. That is, a discrete exterior derivative could map a function on vertices to a function on edges, or a function on edges to a function on faces. These operators will have different shapes, but will share common properties and are derived in similar fashions. In this case, the operator taking a function on vertices to a function on edges would be a rectangular matrix with the number of rows equal to the number of vertices, and the number of columns equal to the number of edges.

In general, operators in DEC are discretized with the philosophy that these operators should preserve one or more important properties that the smooth version of the operator possesses. In the case of the discrete exterior derivative and its smooth counterpart, one such property is that taking the discrete exterior derivative twice in a row should result in a 0 function. We are interested in making use of the discrete exterior derivative operators and extending them to a fractional setting. Therefore, we want to use the smooth version of the operator and discretize it while preserving important properties.

The 1D Caputo fractional derivative is given by

$$D^s_{[a,x]} f(x) = \begin{cases} \dfrac{1}{\Gamma(n-s)} \displaystyle\int_a^x \frac{f^{(n)}(\tau) d\tau}{(x-\tau)^{s+1-n}} & s \notin \mathbb{N} \\ \\ f^{(s)}(x) & s \in \mathbb{N} \end{cases} \tag{1.2}$$

where $s \geq 0$ is the order of the fractional derivative, $[a, x]$ is the interval of interest, $\Gamma(z)$ is the gamma function and $n = \lceil s \rceil$. As can be seen above, a fractional derivative operator is designed so that it uses non-local information in either space or time. For example, a fractional derivative of a time variable would introduce a process to the model that uses the memory of the state of the model at previous time steps to inform the model as to how it should behave at the current time step. For a fractional derivative in space, this equates to using information from an arbitrarily large neighborhood to inform the model's behavior at a specific point.

Discretizing this operator following the philosophy of DEC requires a few steps, some of which come with different challenges along the way. The derivative operator $f^{(n)}$ translates well in the DEC setting for $s \in (0, 1)$, but the next step is to convert the quantity $\frac{1}{(x-\tau)^{s+1-n}}$. In multiple dimensions, this changes to a measure of distance, which must be defined for *all* pairs of points, edges, faces or tetrahedron in the mesh because of the integration. Unfortunately, in DEC, the metric that is established is a local one, so we must give a way to define these distances for elements of a mesh that are not directly connected.

Finally, both the Caputo derivative and the discrete exterior derivative have properties that are important to their definitions. For example, the Caputo derivative converges to the normal derivative as the order $s \to 1$. Thus when we discretize this operator using DEC as the foundation, we would like to be able to replicate this property.

These different settings lead to exploring the problem of how one defines a fractional derivative, which is a nonlocal operator, in the setting of DEC, which is a discretization

technique that makes use of local information whenever possible. We will give one way to do this based off the Caputo derivative in the smooth setting and discuss the properties that the operator should or should not preserve.

1.1.4. Problem Summaries

Overall, we focus on three separate, though related, problem areas. The first is how to implement the trimmed serendipity elements in a software package. The second is testing those elements against common benchmark problems as well as the more sophisticated monodomain model. Finally, we give a definition of a fractional discrete exterior derivative and analysis of it, while also discussing some of the difficulties within using the fractional derivative in the local setting of DEC.

1.2. Related Works and Context in the Literature

1.2.1. Finite Element Exterior Calculus and Software

The ideas central to FEEC include work from the 1900's and early 2000's [9]. Following that, there was large interest in how exterior calculus techniques could be used to help describe and define new numerical techniques, and this resulted in the formal definition of FEEC [15]. Applying these techniques to describe finite element methods in this context led to the definition of the serendipity elements [10, 11] and some of the results were summarized in the periodic table of finite elements [12].

The serendipity and trimmed serendipity elements [39] have the property that $\mathcal{S}_{r-1}\Lambda^k \subseteq \mathcal{S}_r^-\Lambda^k \subseteq \mathcal{S}_r\Lambda^k$. This indicates that analyzing either one of these families of finite elements will make it possible to infer how well the other family does. Since the trimmed serendipity elements at order r will have less DOFs than the serendipity elements at order r, we will focus on using the trimmed serendipity elements. Using the computational basis functions

for the trimmed serendipity elements [40], we are able to analyse convergence, sparsity, and time required to finish various computations.

Elements similar to the trimmed serendipity family were also explored in other non-FEEC contexts around the same time [7], [8] [24]. Each of these results created finite elements that are similar to the trimmed serendipity family, but none of them were done in a unified manner such that the elements were defined for any arbitrary dimension and order. Since the trimmed serendipity family was defined using FEEC, it naturally led to a unified approach for each of the elements in the family. Thus in this work we focus on the trimmed serendipity family of elements rather than their similar counterparts.

During the same overall time period, there were many different software packages created for computing approximate solutions to PDEs. Many of these are still in active development, a few examples of which are FEniCS [2], MFEM [6], Firedrake [69], and deal.II [18]. While any of these packages would work well for approximating solutions to PDEs using finite element methods, we chose to use Firedrake. Firedrake has a large library of elements, an active development community interested in making the package better, and is very intuitive to use. For example, mathematical operators translate directly into words ($\nabla u \to \text{grad}(u)$, $\nabla \cdot u \to \text{div}(u)$). Furthermore, Firedrake can interface directly with PETSc [17, 16] through Python dictionaries to create solver parameters for PDEs in order to compute approximate solutions in an efficient manner.

1.2.2. The Monodomain Equation

The monodomain equation as an application problem [47] in the cardiac field started receiving attention when they first described a basic ionic transport function. Following this, there was the Fitzhugh model [36], and later connected to the work done on electric engineering [65]. This resulted in what is now referred to as the Fitzhugh-Nagumo model, and is the simplified model that we will use for the ionic transport function.

To compute an approximate solution to the monodomain equation, we make use of Firedrake and the Irksome package [35]. The Irksome package allows a user to compute approximate solutions to systems of time-dependent ODEs in Firedrake using implicit Runge-Kutta methods. We use Irksome and some recent work on preconditioners [60] in order to solve the resulting systems. Similar work was done using other PDE solvers [42], where they focused on the numerical effectiveness of computing approximations to the monodomain equation using Runge-Kutta methods.

Other previous work on this topic has included efforts to model the micro-scale dynamics more accurately. This led to the development of the state-of-the-art TenTuscher model [78, 77]. Since its inception, this model has been studied extensively [66], and while there are models that can potentially model the flow of electricity through heart tissue better, they are also more complex than the 18 equation TenTuscher model.

1.2.3. Discrete Exterior Calculus and Fractional Derivatives

The founding work on DEC [46, 30] was also completed in the early 2000's. Over the years, it has found many application areas such as such as texture mapping [31], direction field design [79], fluid simulation [33] and [51], and meshing [64]. Theoretical and numerical calculations to understand how well the operators in DEC can work have made progress in recent years [62, 63], and its limitations have also been discussed [55]. The main software for DEC is PyDEC [19].

Meanwhile, FDEs have been studied much longer. Fractional differential equations can be studied through the use of a Caputo fractional derivative [22], a Riemann-Liouville fractional derivative [41], or a number of other different ways of defining a fractional derivative. Some examples of how these get used in practice are "anomalous diffusion," [72] simulated cloth movement underwater [67], electric currents through a heart tissue affected by ischaemia [34], and shape analysis [68].

There have also been a number of theoretical advances in fractional operators over the years that have extended 1D operators to appropriate non-local versions in multiple dimensions [21], [43], [76] and [29]. This dissertation makes use of the DEC setting to describe another way of computing fractional derivatives in multiple spatial dimensions. Because DEC is able to provide nice definitions for how to discretize common operators in PDEs, such as the Laplacian operator, we believe that being able to interpret the individual pieces in terms of fractional versions of the operators would allow for a similar method of creating discrete fractional operators that preserve properties important to the smooth setting.

1.3. Summary of Results

(1) Finite Element Computational Comparisons

(Chapter 3) This chapter is devoted to a rigorous practical analysis of how trimmed serendipity elements compare to tensor product elements. While both of these elements have been theoretically studied intensively, the new basis functions that were defined for the trimmed serendipity elements give a clear path for implementing them. Our contribution [27] includes comparisons on different metrics such as the number of nonzero entries in matrices, L^2 convergence plots, and timing data. For implementation, we chose to use Firedrake [81].

(2) The Monodomain Equation

(Chapter 4) We then illustrate an example problem that is of practical interest to researchers in the cardiac electrophysiology setting. For them, the monodomain equation represents a simplification of the bidomain model that is still capable of reproducing the behavior of electric waves across the heart. Doing this requires precise set up of the monodomain PDE together with an ODE system that describes the micro-dynamics of the heart. We give results showing how the trimmed serendipity elements are able to compute the solution to these equations using IRKSOME, a module within Firedrake for solving ODE systems using Runge-Kutta methods. In unpublished work, we solve a benchmark study

using trimmed serendipity elements and compute relevant quantities such as activation time that are of interest to cardiac researchers.

(3) Fractional Discrete Exterior Derivative

(Chapter 5) The fractional derivative is a difficult quantity to compute, and doing it in multiple dimensions can lead to many different definitions. We present one definition [26] of a discrete fractional derivative operator that is derived for a DEC-framework and follows the ideology of replicating behavior that the smooth operator has. This leads to a definition for the fractional discrete exterior derivative that we then are able to implement and test on some basic functions.

CHAPTER 2

Background Information and Notation

In each of the following sections, we give background information pertinent to FEEC, DEC, or fractional calculus. This is not a complete covering of these topics, but instead a focused attempt at giving just enough information to a novice in these topics to understand the content given in Chapters 3, 4, and 5. We start with a discussion on finite element methods.

2.1. Finite Element Methods

Finite element methods are a customizable way to compute approximate solutions to PDEs, allowing the user to make choices about what type of geometry to use to discretize the domain, what order of elements (p) to use, and how large should the elements be (h). In 2D, elements can be chosen between squares and triangles (or sometimes, more general polygons), while in 3D the choice comes down to hexagons, prisms or tetrahedra. All of this customization comes with a cost, however: if the user does not understand enough about the method or the problem, they may make an incorrect choice, like using an $H(\text{curl})$ element for an $H(\text{div})$ conforming problem.

Choices like the geometry and order of an element are some of the key pieces to what defines a specific finite element method. For example, one may choose to use square discretization cells with biquadratic polynomials for approximating the solution on the cells. We give the formal definition of a finite element below.

Definition 1 (Ciarlet Finite Element)**.** *Let:*

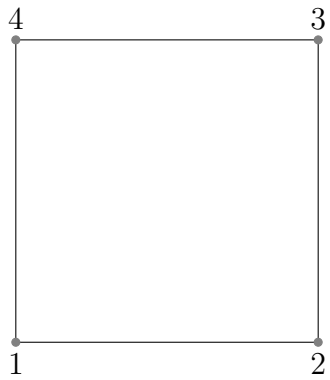


Figure 2.1. Basic reference element for a square Lagrange finite element.

- $\mathcal{K} \subseteq \mathbf{R}^n$ *be a bounded closed set with nonempty interior and piecewise smooth boundary*
- $\mathcal{P}$ *be a finite-dimensional space of functions on* $\mathcal{K}$
- $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ *be a basis for* $\mathcal{P}'$*, the dual space of* $\mathcal{P}$

Then the triple $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ *is called a finite element.*

In the above definition, the set $\mathcal{K}$ contains the element domain, the set $\mathcal{P}$ contains the shape functions, and the set $\mathcal{N}$ contains the nodal variables. Note that a basis for $\mathcal{P}$ that is dual to the set $\mathcal{N}$ is called the nodal basis of $\mathcal{P}$. There are many examples of finite element methods where the choice of the triple is made carefully for properties that a user might want. A couple of examples of these are given below.

Lagrange Elements. Consider the space $\mathcal{P} = \mathcal{Q}_1(x, y)$, the space of bilinear polynomials. Take the element domain $\mathcal{K}$ to be squares and let $K = [0, 1]^2 \in \mathcal{K}$, as depicted in Fig. 2.1. Then set $\mathcal{N} = \{N_1, N_2, N_3, N_4\}$ as the nodal variables, and choose a basis of $\phi_1 = (1-x)(1-y)$, $\phi_2 = x(1-y)$, $\phi_3 = xy$, and $\phi_4 = (1-x)y$ for the bilinear polynomials. This choice will force $N_i(\phi_j) = \delta_{ij}$, where δ_{ij} is the kronecker delta function, so that the set $\{\phi_1, \phi_2, \phi_3, \phi_4\}$ defines the nodal basis.

The Lagrange element above is also known as a tensor product element, and is one example of an element that becomes part of the wider framework that finite element exterior

calculus will provide. If we instead chose the set $\mathcal{N}$ to include 9 nodal variables, with 1 on each vertex, 1 on each edge, and 1 in the center of the face, then we would also choose a nodal basis that is made up of the biquadratic polynomial functions. These choices as to the exact order of the elements and the basis that follows are what lead us to being able to define the entire family of scalar tensor product elements. Beyond just defining the elements, however, we also care about how we would use them to approximate a solution to a PDE.

Motivating Example. Consider the mixed formulation of the Poisson problem on a domain $\Omega = [0,1] \times [0,1]$ with boundary Γ. Find functions $\sigma \in \Sigma := H(\text{div})$ and $u \in U := L^2$ such that

$$\begin{aligned} \sigma - \nabla u &= 0 \text{ on } \Omega \\ \nabla \cdot \sigma &= -f \text{ on } \Omega \end{aligned} \tag{2.1}$$

when applying homogeneous Dirichlet boundary conditions. To solve this using a finite element method the equation must be written in a weak form. Doing this yields

$$\begin{aligned} \int_\Omega (\sigma \cdot \tau + \nabla \cdot u\tau) \, \mathrm{d}x &= 0 \qquad && \forall\ \tau \in \Sigma, \\ \int_\Omega \nabla \cdot \sigma v \, \mathrm{d}x &= -\int_\Omega f v \, \mathrm{d}x \qquad && \forall\ v \in V. \end{aligned} \tag{2.2}$$

After that, the user chooses a suitable finite element pairing for the spaces Σ and U in accordance with finite element theory.

2.2. Exterior Calculus

Before we connect the basic finite element theory with FEEC, we first give some key definitions in exterior calculus. For a more rigorous discussion of topology and geometry, see any of the useful textbooks on the matter [56], [74].

To start, we need a domain that we can discretize. Choosing the domain to be a manifold will be beneficial later.

Definition 2. *A manifold M is a set with three main properties:*

(1) *M is a Hausdorff space.*

(2) *M is second-countable.*

(3) *M is locally Euclidean of dimension n.*

In order for M to be a smooth manifold, we require that M has a smooth structure paired with M. Some examples of smooth manifolds include T^2, S^n, and all finite dimension vector spaces. However, we want to be able to think about the discrete version of a manifold. The key to that is being able to discretize the manifold in some way and think of it still as living in $\mathbb{R}^n$.

Definition 3. *Let F be a smooth function mapping from M to N, where both M and N are smooth manifolds. Then F has constant rank r if dF_p as a matrix has rank r for all $p \in M$. If F has constant rank r and $r = dim(M)$, then F is said to be a smooth immersion. Note that this means that dF_p is an injective mapping for any $p \in M$. Finally, a smooth embedding is a smooth immersion that also has the property that F is a homeomorphism onto its image $F(M) \subseteq N$.*

The *Strong Whitney Embedding Theorem* tells us that every smooth manifold can actually be smoothly embedded into $\mathbb{R}^{2n}$. For our work, this is more of a theoretical result than a practical one, as we will often not be concerned with how to do this embedding.

Definition 4. *A vector field X based at p on M is a section of the map $\pi : TM \to M$. We think of the vector field X with base point p as $X^i(p)\frac{\partial}{\partial x^i}|_p$, where the indexing on i is used as in Einstein summation notation to mean the sum over all relevant coordinates.*

Vector fields in $\mathbb{R}^n$ have a familiar form. For example $x\frac{\partial}{\partial x} + y^2\frac{\partial}{\partial y}$, is a vector field in $\mathbb{R}^2$. If we wanted to actually compute the vectors at each point for this vector field, we would treat $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ as standard basis vectors, and x, y^2 as the weights for each point.

To fully encapsulate a differential equation using the language of differential geometry, we do need more than vector fields. We start with covector fields and then general k-forms.

Definition 5. *A covector on a vector space V is a linear functional on V. A covector field or 1-form is a linear functional on the space of vectors fields of a manifold M.*

A general 1-form on M (if we choose coordinate charts) will be described in terms of a basis dx^i, which is dual to the basis for vector fields $\frac{\partial}{\partial x^i}$.

Example 1. *Let $X = 2x\frac{\partial}{\partial x} + 3xy^4\frac{\partial}{\partial y}$ be a vector field on $\mathbb{R}^2$ and $\alpha = 9ydx + sin(x)ydy$. Then we have that $\alpha(X) = 18xy + 3xy^5 sin(x)$.*

A smooth 0-form is just a smooth function on M. Finally, we define differential k-forms. We create differential k-forms by combining together 1-forms using the wedge operator $\wedge$.

Definition 6. *A differential k-form ω is a map $\omega : \Omega^k(M) \to \mathbb{R}$.*

Thus one example of a differential 2-form would be $\alpha = 5xydx \wedge dy$. Note that this definition is simplified from the technical definition given in textbooks [57]. The natural question arising from differential k-forms is how to move between them. Moving between the differential forms is important when it comes to being able to discretize these operators using DEC.

Definition 7. *The exterior derivative d is a map from $\Omega^k(M) \to \Omega^{k+1}(M)$ (k-forms to $k+1$-forms) such that*

(1) $d(\alpha + t\beta) = d\alpha + td\beta$

(2) $d(\omega \wedge \eta) = d\omega \wedge \eta + (-1)^l \omega \wedge d\eta$ *for* $\omega \in \Omega^k(M)$ *and* $\eta \in \Omega^l(M)$

Hodge Star	$*$	k-forms to $(n-k)$-forms
Flat	$\flat$	vector fields to 1-forms
Sharp	$\sharp$	1-forms to vector fields

Table 2.1. A few of the other operators that are commonly used in exterior calculus when considering PDEs.

(3) $d \circ d = 0$.

As an example, the exterior derivative d maps a 1-form to a 2-form by computing

$$d(\omega_j dx) = \sum_{i<j} \big(\frac{\partial \omega_j}{\partial x^i} - \frac{\partial \omega_i}{\partial x^j}\big) dx^i \wedge dx^j.$$

This definition [57] extends to higher order forms, but we leave the rigorous discussion of this out. When using the exterior derivative, we can note that going from 0-forms to 1-forms looks similar to the gradient operation. For example, if we let $f = f(x, y, z)$ be a 0-form, then we have that

$$df(x, y, z) = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial z} dz.$$

Considering the example of $\mathbb{R}^3$, $d : \Omega^1(M) \to \Omega^2(M)$ is analogous to the curl operator, while $d : \Omega^2(M) \to \Omega^3(M)$ looks like the divergence operator. There are other operators that we do not discuss in-depth that allow one to move from the vector calculus domain to the exterior calculus domain or vice versa, such as $*$, $\flat$, and $\sharp$. Each of these operators is useful in being able to solve PDEs through an exterior calculus framework. These are listed in Table 2.1 with their names and domain and range.

2.2.1. Finite Element Exterior Calculus

By using the exterior calculus ideas above, and a few more specific ones we give below, we can start to apply exterior calculus to get the finite element exterior calculus. The framework that FEEC sets up for finite element methods helps make the choice of which finite element

to use. One of the key ways of doing this is by writing down a *de Rham complex*, which can inform a finite element user as to the properties that the elements must satisfy.

Definition 8. *Let $\mathcal{R}$ be a commutative ring, and suppose we have a sequence of spaces A^n and a sequence of $\mathcal{R}$-linear maps f. Then the sequence*

$$\dots A^k \xrightarrow{f} A^{k+1} \xrightarrow{f} A^{k+2} \dots$$

is called a complex if $f \circ f = 0$ for any starting k.

Then a de Rham complex is a complex where the maps are the exterior derivative and the spaces are the differential k-forms. One example of the de Rham complex for a subset $\Omega \subset \mathbb{R}^3$ would be

$$0 \to H^1(\Omega) \xrightarrow{\text{grad}} H(\text{curl}, \Omega; \mathbb{R}^3) \xrightarrow{\text{curl}} H(\text{div}, \Omega; \mathbb{R}^3) \xrightarrow{\text{div}} L^2(\Omega) \to 0. \tag{2.3}$$

This de Rham complex can be thought of as mapping between differential forms by associating it with

$$0 \to H^1(\Omega) \xrightarrow{d_0} H(\text{curl}, \Omega; \mathbb{R}^3) \xrightarrow{d_1} H(\text{div}, \Omega; \mathbb{R}^3) \xrightarrow{d_2} L^2(\Omega) \xrightarrow{d_3} 0.$$

Similarly, in an $\mathbb{R}^2$ setting, we end up with

$$0 \to C^\infty(\Omega) \xrightarrow{\text{curl}} C^\infty(\Omega, \mathbb{R}^2) \xrightarrow{\text{div}} C^\infty(\Omega) \to 0$$

and

$$0 \to C^\infty(\Omega) \xrightarrow{d_0} C^\infty(\Omega, \mathbb{R}^2) \xrightarrow{d_1} C^\infty(\Omega) \xrightarrow{d_2} 0.$$

Now, recall the example of the mixed Poisson problem where we want to solve for a vector function σ and a scalar function u. Then it is natural that we want a space of finite

elements associated with the $H(\text{div})$ space and with the L^2 space because we know that $\nabla \cdot \sigma$ must exist and that $u \in L^2$. Using the 3D version of the de Rham complex, we associate the $H(\text{div})$ space with 2-forms and the L^2 space with 3-forms.

Using all the different pieces, we get the notation that is used in FEEC when referring to spaces. A set of elements would be denoted as $\mathcal{E}_r\Lambda^k(\Omega^n)$, where $\mathcal{E}$ is replaced with the element letter, such as $\mathcal{Q}$ for tensor product elements and $\mathcal{S}$ for serendipity elements. There is sometimes a $^-$ to denote the "trimmed" version of elements, and it will follow the element letter. The value of r is the order of the polynomials used for the element, k is the degree of the differential forms involved, Ω is the domain and n is the dimension of the domain. For example, the spaces involved in the mixed Poisson problem are $\mathcal{Q}_r^-\Lambda^2(\Omega^3)$ and $\mathcal{Q}_r^-\Lambda^3(\Omega^3)$.

The tensor product elements and serendipity elements are a couple of examples of how this notation can be used. Other elements from traditional finite element methods have also been converted into similar differential form notations, such as the spaces $\mathcal{P}\Lambda^k$ and $\mathcal{P}^-\Lambda^k$. The compact description of many different scalar and vector finite elements in this way encompasses traditional finite elements within the $\mathcal{P}$ and $\mathcal{P}^-$ families, such as Lagrange, discontinous Galerkin, Raviart-Thomas, Brezzi-Douglas-Marini, and Nédélec elements.

2.2.2. Discrete Exterior Calculus

In contrast to the way FEEC uses de Rham complexes to inform decisions for building families of finite elements, DEC focuses on discretizing the operators we are interested in.

To use DEC, we build up a simplicial complex K of dimension N by using p-dimensional simplices σ^p, where $p \in \{0, 1, \ldots, N\}$. These simplices are defined by $\sigma^p = [v_0, v_1 \ldots, v_p]$, where the order that we give $v_0, v_1, \ldots v_p$ specifies the orientation of the p-simplex.

In general, p-simplices can be thought of as a p-dimensional analogue of a tetrahedron. At low dimensions, we have the following p-simplices:

- 0-simplices are vertices

- 1-simplices are edges
- 2-simplices are (triangular) faces
- 3-simplices are tetrahedron.

On each p-simplex, we can attribute some value, and this represents a discrete p-form, also called a cochain. This assignment is a way of thinking of the cochains as functionals on the simplicial complex.

Similar to the exterior calculus setting, we can move between p simplices and $(p-1)$ simplices. To do that, we make use of the boundary operator ∂_p acting on a simplex σ^p

$$\partial_p \sigma^p = \partial_p([v_0, v_1, \ldots, v_p]) = \sum_{i=0}^{p} (-1)^i [v_0, \ldots, \hat{v_i}, \ldots, v_p].$$

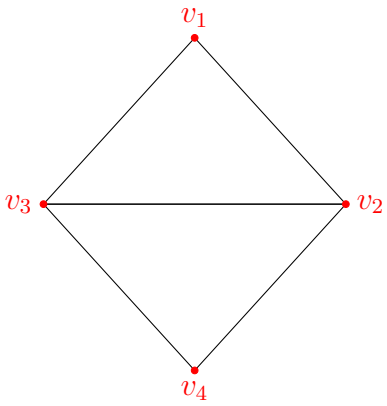
This has the property that $\partial_p \circ \partial_{p+1} = 0$. The coboundary operator then should map cochains defined on the simplices, and instead of going down in order, it should go up.

This leaves us with the coboundary operator δ^p being defined as

$$\langle \delta^p \alpha^p, c_{p+1} \rangle = \langle \alpha^p, \partial_{p+1} c_{p+1} \rangle.$$

The coboundary operator gets defined as the discrete exterior derivative, and we notate the discrete exterior derivative as $\mathbb{D}_p$. This is a rectangular matrix that has a number of columns equal to the number of p-simplices in the complex and a number of rows equal to the number of $(p+1)$-simplices in the complex. Specifically, $\mathbb{D}_p$ is the transpose of the boundary operator matrix, and it allows movement from discrete p-forms to discrete $(p+1)$-forms.

Consider the mesh in Fig. 2.2 as an example of a small simplicial complex. Beside it, we have an example of a 0-form $\hat{f}$, which is a mapping between the vertices of the simplicial complex and $\mathbb{R}$. After that, we show the result of applying the discrete exterior derivative to the 0-form $\hat{f}$.



$$\hat{f} = \begin{bmatrix} 3 \\ -7 \\ -6 \\ 2 \end{bmatrix} \qquad \mathbb{D}_0\hat{f} = \begin{bmatrix} -10 \\ 9 \\ 1 \\ 9 \\ 8 \end{bmatrix}$$

Figure 2.2. We show a small simplicial complex made up of 4 vertices that are the 0-simplices, 5 edges that are the 1-simplices, and 2 faces that are the 2-simplices. The vectors $\hat{f}$ and $\mathbb{D}_0\hat{f}$ show examples of a 0-form and its derivative.

The discrete exterior derivative is a local operator that does not make use of any metric information. The matrix $\mathbb{D}_p$ only has nonzero entries of ± 1 for simplices that are adjacent. However, other DEC operators (such as the Hodge star) make use of metric information that is given by a local metric $d(v_0, v_1)$ where v_0, v_1 are part of an edge $[v_0, v_1] \in K$ (see [46] a discussion on this). These values give information about the distances between vertices on the primal mesh. As we will see later, this will become important when extending the discrete exterior derivative to a fractional setting.

2.3. Fractional Differential Equations

Fractional calculus has many different definitions of a fractional derivative (e.g. Riemann-Liouville, Caputo, Grunwald-Letnikov [45]). Central to all of them is the idea that the "sth" derivative of a function for $s \in [0, 1]$ should give back a function in between the 0th (identity) and 1st derivative that varies continuously with s, such as in Fig. 2.3. In our work, we will focus on the Caputo derivative. Before we give the definition of the Caputo derivative itself, first we give some background information.

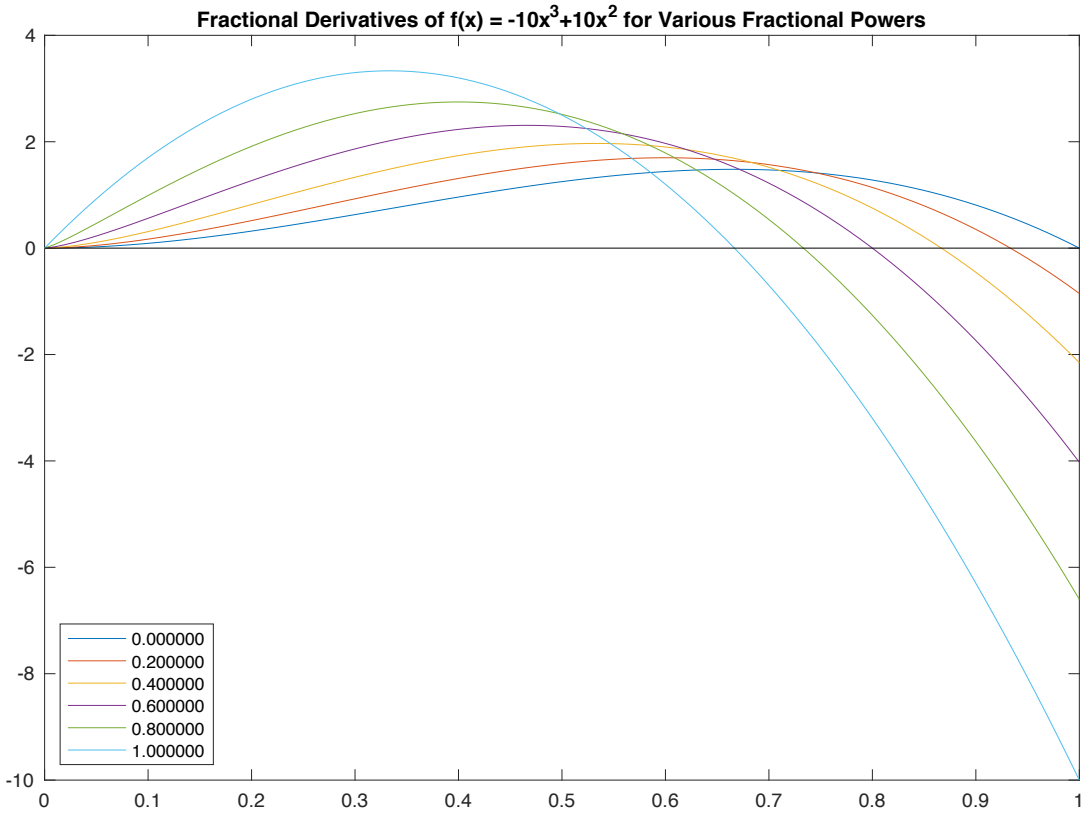


Figure 2.3. Fractional powers of order $s = 0,\ 0.2,\ 0.4,\ 0.6,\ 0.8,\ 1$, for the left-sided Caputo derivative of $f(x) = -10x^3 + 10x^2$.

Definition 9. *A function f is said to be* n-times continuously differentiable *if we can take n derivatives of f and $f^{(n)}$ is a continuous function. In this case, we denote it as $f \in C^n[a, b]$.*

Definition 10. *The* Gamma function *is given by*

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt \tag{2.4}$$

where $Re(z) > 0$.

For natural numbers, the Gamma function acts like a factorial. That is, $\Gamma(n+1) = n!$ whenever $n \in \mathbb{N}$. In terms of fractional calculus, the Gamma function will typically just be a constant that we can compute when needed. Now we have the fractional Caputo derivative.

Definition 11. *The* left-sided fractional Caputo derivative *of order $s \in \mathbb{R}^+$ of a function $f \in C^n[a,b]$ is*

$$D^s_{[a,x]} f(x) = \begin{cases} \dfrac{1}{\Gamma(n-s)} \displaystyle\int_a^x \frac{f^{(n)}(\tau)d\tau}{(x-\tau)^{s+1-n}} & s \notin \mathbb{N} \\ f^{(s)}(x) & s \in \mathbb{N} \end{cases} \tag{2.5}$$

where $n = \lceil s \rceil$ [22]. Note that often in the fractional calculus literature, the Caputo derivative is denoted as

$$\overset{C}{a} D^s_x f(x).$$

We choose to not use this notation because the majority of this work will focus only on the Caputo derivative, so we can drop the ^{C} and choose the initial notation. The only time we deviate from this is in Section **??**, where we will briefly discuss the Riemann-Liouville derivative. In that case, we will make sure it is clear that we are discussing a fractional derivative operator that is not the Caputo derivative.

The subscript $[a,x]$ serves two purposes. The first is that it defines the domain of integration for the fractional derivative, and the second is that it is used to denote which variable this is with respect to. We will also make use of a 2-sided definition of a fractional Caputo derivative.

Definition 12. *A two-sided fractional Caputo derivative of order $s \in \mathbb{R}^+ \backslash \mathbb{Z}$ of a function $f \in C^n[a,b]$ is*

$$D^s f(x) = \frac{1}{\Gamma(n-s)} \left(\int_a^x \frac{f^{(n)}(\tau)}{(x-\tau)^{s+1-n}} d\tau + \int_x^b \frac{f^{(n)}(\tau)}{(\tau - x)^{s+1-n}} d\tau \right). \tag{2.6}$$

This can be written in a more compact form as

$$D^s f(x) = \frac{1}{\Gamma(n-s)} \int_{[a,b]\setminus\{x\}} \frac{f^{(n)}(\tau)}{|x-\tau|^{s+1-n}} d\tau. \tag{2.7}$$

To understand the behavior of these operators, we have some examples in the continuous setting of the left-sided fractional Caputo derivative.

Example 2. *Due to taking a derivative before integrating, any constant function $f(x) = c$ on $[0, 1]$ has the property that*

$$D^s_{[a,x]} f(x) = 0 \tag{2.8}$$

for any $s \in \mathbb{R}$.

Example 3. *Let $f(x) = x^q$, $q > 0$ and $s \in (0, 1)$. Then we have that*

$$D^s_{[0,x]} f(x) = \frac{\Gamma(q+1) x^{q-s}}{\Gamma(q+1-s)}. \tag{2.9}$$

This result mimics what we would expect in the case of $s = 1$, i.e. $D^1_{[0,x]} x^q = q x^{q-1}$. We see some results for the left-sided fractional Caputo derivative for $f(x) = -10x^3 + 10x^2$ in Fig. 2.3 for a few different powers of s.

By changing the subscript notation to $[a_x, x]$ (or more generally $[a_i, x_i]$), we can extend the one-variable definition of the Caputo derivative to functions of multiple variables by writing

$$\frac{\partial^s}{(\partial x)^s}\bigg|_{(a_x,x)} f(x,y) = \frac{1}{\Gamma(1-s)} \int_{a_x}^{x} \frac{f_x(\tau,y)d\tau}{(x-\tau)^s}$$

if $s \in (0,1)$.

Regardless of how many variables we choose to work in, part of the difficulty with FDEs becomes obvious from the definitions: the operators that we are using are non-local. Since these derivatives require the integration over some domain $[a,x]$, a small neighborhood of information at a point is not enough to give the value of the fractional derivative at that point. This leads to some numerical difficulties.

Primarily, a major issue is that numeric computations are more challenging. For spatial FDEs, examining a large neighborhood manifests as more computational work to gather and process the surrounding information. In general, for every fractional exponent, we may need to examine the values defined on the entire domain to accurately compute the derivative. For fractional derivatives that involve time, the computation necessitates storing historical information for all previous values. This represents a process with memory, but there is a tradeoff in that storing all history in practice would be computationally infeasible.

In this work we focus on spatial FDEs, but needing more spatial information to compute the fractional derivative can be challenging. Consider using a matrix to encode the fractional derivative operator. Due to the non-local nature of a fractional derivative, such matrices lose a lot of the sparsity and structure that they would normally have. For example, [72], investigated the use of a finite element scheme to solve the fractional advection-dispersion equation. In order to use quadratic basis elements on a 64×64 mesh, Roop had to store a full 16641×16641 stiffness matrix with no special structure.

Spatial FDEs are a primary factor in choosing to use a 2-sided fractional Caputo operator. Using a left-sided operator in a temporal setting can be interpreted as meaning the process has a memory effect. A left-sided operator in a spatial setting would ignore non-local effects that occur on the right side with no geometrical justification. Thus the 2-sided definition

we have given will be used in our discretization for defining a fractional discrete exterior derivative.

CHAPTER 3

Trimmed Serendipity Elements

Using the finite element exterior calculus notation and definitions from the previous chapter, we discuss the trimmed serendipity finite element family. Previous to this work, trimmed serendipity elements had a solid theoretical foundation that showed that they should be able to approximate solutions to PDEs and attain convergence rates equal to that of tensor product elements. Thus in this work, we focus on showing these results in practice, which requires first a discussion on the implementation of these elements into a software package, followed by an analysis of the elements in practice when compared to the tensor product elements.

3.1. Background

Rigorously defining serendipity and trimmed serendipity elements has been a relatively recent advancement in finite element theory. The serendipity scalar elements [10] were first introduced in the context of FEEC before the serendipity family as a whole, including the vector elements, were defined formally [11]. These elements are denoted using the FEEC notation as $\mathcal{S}_r\Lambda^k(\Omega^n)$.

As discussed in the introduction, the trimmed serendipity elements [39] followed the serendipity elements, and they had the design goal of minimizing the number of DOFs used in any particular trimmed serendipity element. In particular, they wanted an element on n-dimensional cubes with a basis that contained all of the functions in $\mathcal{P}_{r-1}\Lambda^k$. The optimality condition for the minimal number of DOFs satisfying the property of containing the $r-1$ degree polynomials had been previously worked out [23]. The trimmed serendipity elements are denoted as $\mathcal{S}_r^-\Lambda^k(\Omega^n)$.

The method for creating these elements was to use the Koszul operator κ.

Definition 13. *The Koszul operator is a map on differential forms,* $\kappa : \Lambda^k(\Omega^n) \to \Lambda^{k-1}(\Omega^n)$ *such that the action on a k-form with a monomial coefficient is given by*

$$\kappa(x^\alpha dx_\sigma) = \sum_{i=1}^{k} \left((-1)^{i+1} x^\alpha x_{\sigma(i)} \right) dx_{\sigma(1)} \wedge \ldots \wedge \widehat{dx_{\sigma(i)}} \wedge \ldots \wedge dx_{\sigma(k)}$$

where the $\widehat{dx_{\sigma(i)}}$ *denotes a skipped factor.*

By applying the Koszul operator to the (regular) serendipity elements, we can define the trimmed serendipity elements. The trimmed serendipity elements are given by

$$\mathcal{S}_r^- \Lambda^k := \mathcal{S}_{r-1}\Lambda^k + \kappa \mathcal{S}_{r-1}\Lambda^{k+1}.$$

Therefore, we get two inclusions, the first of which is obvious

$$\mathcal{S}_{r-1} \subseteq \mathcal{S}_r^- \Lambda^k \subseteq \mathcal{S}_r \Lambda^k.$$

The second inclusion is due to the multiplication of x^α and $x_{\sigma(i)}$ that results in an r^{th} degree polynomial if x^α was degree $r-1$. These inclusions are analogous to the case of the polynomial and trimmed polynomial spaces on simplices, $\mathcal{P}_r^- \Lambda^k$ and $\mathcal{P}_r \Lambda^k$.

3.1.1. Scalar and Vector 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, 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 [10].

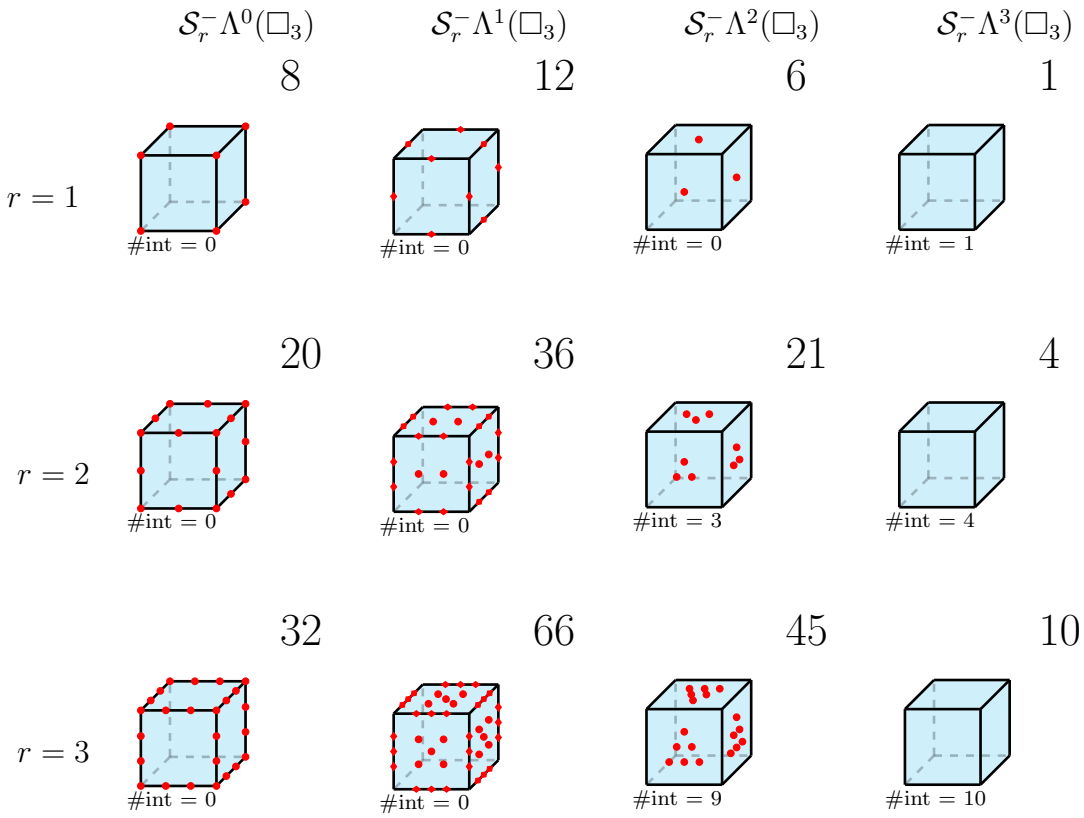
Likewise, the scalar-valued trimmed serendipity elements that are represented by n-forms create L^2-conforming finite element spaces. These are denoted by $\mathcal{S}_r^-\Lambda^n(\square_n)$, and here we have the equality $\mathcal{S}_r^-\Lambda^n(\square_n) = \mathcal{S}_{r+1}\Lambda^n(\square_n)$. 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 $\mathcal{S}_r^-\Lambda^n(\square_n) = \mathcal{P}_{r+1}\Lambda^n(\square_n)$. 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.

A major reason that the vector trimmed serendipity elements have only recently been considered in the mathematical literature is that their DOF per element count is complicated. As evidenced by Table 3.1, *certain* DOFs grow in predictable patterns with r. For instance, elements in the 1-form family, $\mathcal{S}_r^-\Lambda^1(\square_3)$, have exactly r DOFs per edge of the cube (corresponding to "order r" approximation on edges) and elements in the 2-form family, $\mathcal{S}_r^-\Lambda^2(\square_3)$, 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 [73] and is in agreement with the general formulae presented in the previous work on trimmed serendipity elements [39], but a natural geometric interpretation remains elusive. Equally unexpected patterns are evident in the growth of DOFs on faces and interiors of the $\mathcal{S}_r^-\Lambda^1(\square_3)$ family.

3.2. Implementation

To implement the trimmed serendipity elements, we chose to use a python library called Firedrake. Firedrake is built on top of a variety of other python and C++ libraries to achieve an efficient and robust way of using finite element methods. One of the libraries that

Table 3.1. Trimmed serendipity elements on a reference cube in 3D, akin to the Periodic Table of the Finite Elements. Columns show increasing form order from $k = 0$ to $k = 3$, corresponding to H^1, $H(\text{curl})$, $H(\text{div})$, and L^2 conformity, respectively. Rows show increasing order of approximation $r = 1, 2, 3$. On the front face of each element, dots indicate the number of DOFs associated to each vertex, edge, or face of the cubical element. The number of DOFs associated to the interior of the cube is indicated with "# int =." The total degree of freedom count for each element is shown to its right.



Firedrake uses is FIAT [52, 53] to provide finite element basis functions on reference elements. The FIAT library is the first step in making a new finite element usable in Firedrake.

We briefly describe the rest of the process that Firedrake uses when creating new elements in FIAT. First, we must provide both the 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 [3]) and a translation from symbolic name to concrete implementation in the form compiler TSFC [48].

While FIAT initially considered a very wide range of finite elements [54], 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 the computational trimmed serendipity bases [40] 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 [61] and compute derivatives symbolically.

We use the trimmed serendipity computational decompositions 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

$$\mathcal{S}_r^-\Lambda^1(\square_3) = \underbrace{\left[\bigoplus_{i=0}^{r-1} E_i\Lambda^1\right]}_{\text{edge functions}} \oplus \underbrace{\left[\bigoplus_{i=2}^{r-1} F_i\Lambda^1\right] \oplus \left[\tilde{F}_r\Lambda^1\right]}_{\text{face functions}} \oplus \underbrace{\left[\bigoplus_{i=4}^{r-1} I_i\Lambda^1\right] \oplus \left[\tilde{I}_r\Lambda^1\right]}_{\text{interior functions}}. \tag{3.1}$$

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 $S_2^-\Lambda^1$ in 3D. The space $\tilde{I}_r\Lambda^1$ is defined to be empty for $r < 4$, so there are only two sets of functions to include in this case: one set associated to edges of the reference element (the $E_i\Lambda^1$ sum) and one set for the faces of the reference element (the $\tilde{F}_2\Lambda^1$ functions). According to Eq. (3.1),

the basis functions can be decomposed as

$$\mathcal{S}_2^-\Lambda^1(\square_3) = \left[\bigoplus_{i=0}^{1} E_i\Lambda^1\right] \oplus \left[\tilde{F}_2\Lambda^1\right] \oplus \left[\tilde{I}_2\Lambda^1\right]. \tag{3.2}$$

We then implement these in Firedrake as follows. The first step is to determine the number of DOFs we will need on the reference element where we will define the basis functions. To do this, we count the DOFs for each mesh entity on the reference element (vertex, edge, face, and interior). For example, $\mathcal{S}_2^-\Lambda^1(\square_3)$ should have no DOFs at the vertices (these only come into play for the H^1-conforming elements), two DOFs on each edge, two DOFs on each face, and no DOFs on the interior. This agrees with Eq. (3.2), 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 3.1.

With the number of DOFs assigned to each mesh entity in the reference element, we can then define the basis functions. The order of definition is important so that basis functions are matched to the proper mesh entity. Note that Eq. (3.2) doesn't explicitly give a way to order the basis functions. Instead, we need to use the properties of the basis functions to determine the correct ordering. This is best illustrated by an example. Two of the "edge" basis functions that are contained in the sum of the $E_i\Lambda^1$ sets are $(y+1)(z+1)dx$ and $x(y+1)(z+1)dx$. Notice that these functions have no dy or dz portions. Therefore, these function vanish on any edge *not* parallel to the x axis. Further, the polynomial coefficients of these forms indicate that they also vanish on the planes $y = -1$ and $z = -1$. Thus, the only edge of the cube on which these functions do *not* vanish is the edge contained in the line $\{y = 1\} \cap \{z = 1\}$. This edge is shown in blue and labeled with an "e" in Fig. 3.1.

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.

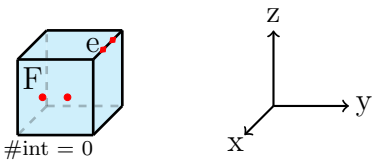


Figure 3.1. The reference cube $[0,1]^3$ is shown, with coordinate axes indicated. The edge e lies in the intersection of the planes $y = 1$ and $z = 1$. To ensure proper continuity, basis functions associated to e must vanish on all edges of the cube *except* e. Examples of such functions for e are described in detail in the text. Additional examples of functions associated to the face F are also given.

While the formulas above [40] are technically correct, these monomials have poor conditioning at higher orders. Therefore, we use Legendre polynomials obtained symbolically from SymPy via the `legendre` 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

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

In this case, one of the basis functions is $y^j z^k(x+1)dydz$. The $dydz$ represents a 2-form, which vanishes on any face *not* parallel to the yz plane. This leaves only the faces contained in the planes $x = 1$ or $x = -1$ as possibilities for association. As in the 1-form example above, the polynomial coefficient of the form indicates that this function will vanish on an additional mesh entity, namely, the face contained in the plane $x = -1$, in this case. Hence, we associate this function with the face contained in $x = 1$ (labeled with an "F" in Fig. 3.1).

The rest of the process for defining the 2-form basis functions is similar to the process for the 1-form basis functions.

Table 3.2. A translation between FEEC and Firedrake usage names for tensor product and trimmed serendipity elements. In 2D, the 0-forms are H^1 conforming spaces, the 1-forms are $H(\text{curl})$ and $H(\text{div})$ conforming spaces (dependent upon oreintation of the DOFs), and the 2-forms are L^2 conforming spaces. For 3D, the 0-forms are H^1 conforming spaces, the 1-forms are $H(\text{curl})$ conforming spaces, 2-forms are $H(\text{div})$ conforming spaces, and 3-forms are L^2 conforming spaces.

FEEC	UFL name (2D)	UFL name (3D)
$\mathcal{Q}_r^-\Lambda^0$	`Lagrange`	`Lagrange`
$\mathcal{Q}_r^-\Lambda^1$	`RTCE` or `RTCF`	`NCE`
$\mathcal{Q}_r^-\Lambda^2$	`DQ`	`NCF`
$\mathcal{Q}_r^-\Lambda^3$	-	`DQ`
$\mathcal{S}_r^-\Lambda^0$	`S`	`S`
$\mathcal{S}_r^-\Lambda^1$	`SminusCurl` or `SminusDiv`	`SminusCurl`
$\mathcal{S}_r^-\Lambda^2$	`DPC`	`SminusDiv`
$\mathcal{S}_r^-\Lambda^3$	-	`DPC`

The newly supported elements, mapping FEEC spaces onto names in UFL are shown in the lower half of Table 3.2. Modifying the code from **??** 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 3.1, the rest of the code remains unchanged.

Listing 3.1. Setting up Firedrake to use the trimmed serendipity elements in a mixed Poisson problem in 3D.

```
...
hDivSpace = FunctionSpace(mesh, "SminusDiv", polyDegree)
l2Space = FunctionSpace(mesh, "DPC", polyDegree - 1)
...
```

3.3. Comparison and Results

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 $H(\text{div})$ elements are only a rotation of the DOFs on the edges of elements in 2D, we do not give an experiment using $H(\text{curl})$ elements in 2D.

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

For simplicity, our numerical experiments all use a sparse direct solver. We expect the multigrid theory [13, 14] to carry over from existing $\mathcal{Q}^-$ spaces to $\mathcal{S}^-$ 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.

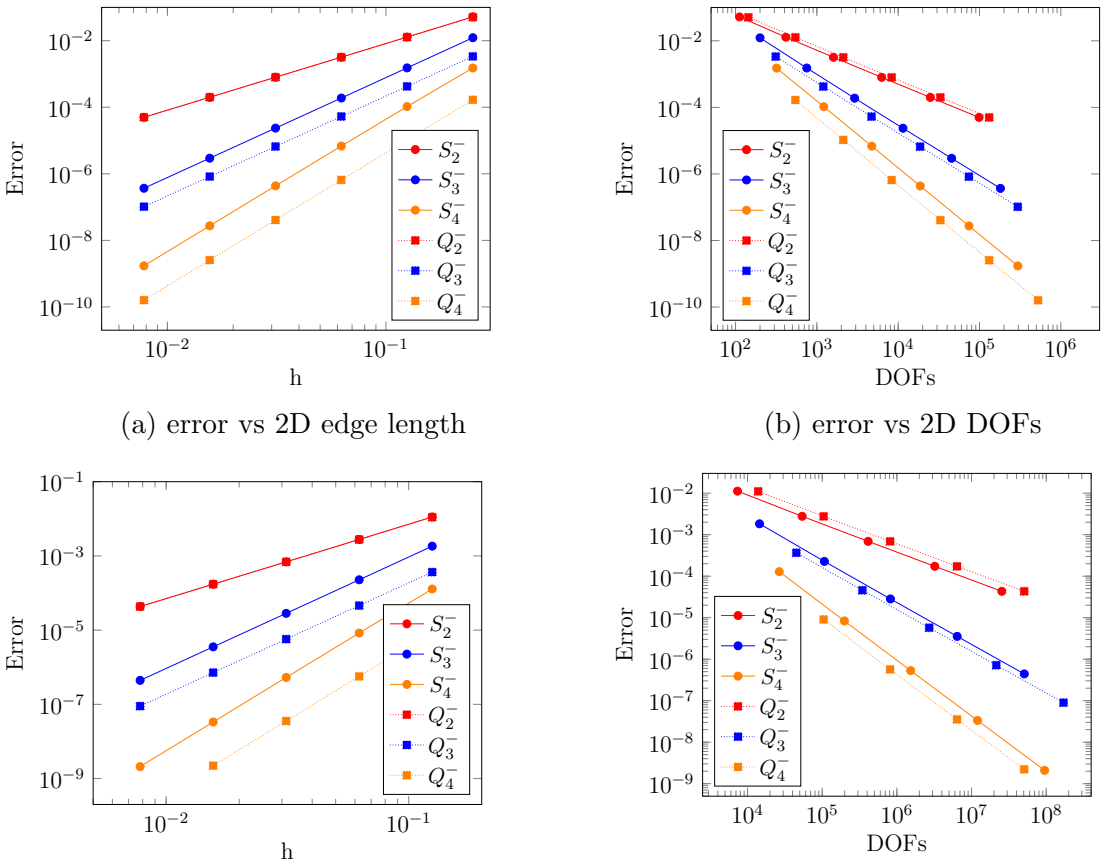
3.3.1. Projection

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

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

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 `SminusCurl`) 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 $\mathcal{S}_r^-$ and the tensor product elements with $\mathcal{Q}_r^-$. In Firedrake, the trimmed serendipity elements use the label `SminusCurl` or `SminusDiv` for 1-forms in 2D, depending upon the orientation of the edge DOFs, and in 3D, they represent the 1- and 2-forms respectively. The tensor product elements use the labels `RTCE` (or `RTCF`) and `NCE` for 1-forms in 2D and 3D, while the 2-forms in 3D are `NCF`. To solve the projection problem, we use a Galerkin L^2 projection into V.

The goal of the projection problem is to establish that the elements attain the mathematically expected L^2 rates of approximation as mesh size decreases, as well as comparing relative efficiencies of $\mathcal{S}^-$ and $\mathcal{Q}^-$. For this, we create a mesh on $[0,1]^n$ of uniformly sized squares or cubes, where we refine the mesh from $N = 4$ squares (or cubes) in each row and column to $N = 128$ squares in 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



(a) error vs 2D edge length

(b) error vs 2D DOFs

(c) error vs 3D edge length

(d) error vs 3D DOFs

Figure 3.2. 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 (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.

Fig. 3.2, we report the L^2 error in terms of the classical measure of maximum edge length (h) as well as total number of DOFs.

The expectation is that $\mathcal{S}_r^-$ and $\mathcal{Q}_r^-$ converge at the same rate with respect to h, which is confirmed in the plots by parallel trendlines. These parallel trendlines can be seen in each of the projection plots. 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 $\mathcal{S}_r^-$ are above the trendlines for $\mathcal{Q}_r^-$. However, considering the elements $\mathcal{Q}_3^-$ and $\mathcal{S}_4^-$, we see that $\mathcal{Q}_3^-$ requires approximately 1.71×10^8 DOFs and $\mathcal{S}_4^-$ at the same mesh refinement requires 0.95×10^8 DOFs. While $\mathcal{Q}_3^-$ attains an error of 8.96×10^{-8}, the $\mathcal{S}_4^-$ elements attain an error of 2.1×10^{-9}.

3.3.2. 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 Ω for $u \in U$:

$$-\Delta u = f$$

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

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:

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x, \qquad \text{for all } v \in V.$$

Accordingly, the primal formulation employs H^1 elements, using `S` for $\mathcal{S}_r^-\Lambda^0$ and `Lagrange` for $\mathcal{Q}_r^-\Lambda^0$.

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

$$\sigma - \nabla u = 0$$

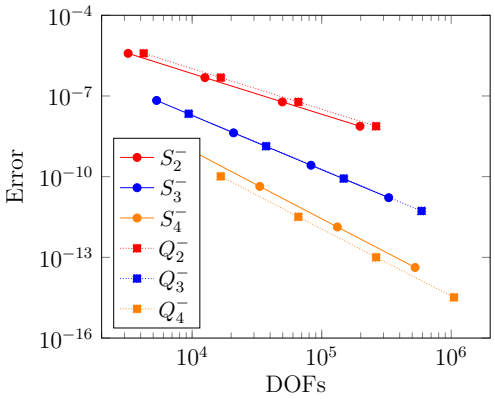
$$\nabla \cdot \sigma = -f$$

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

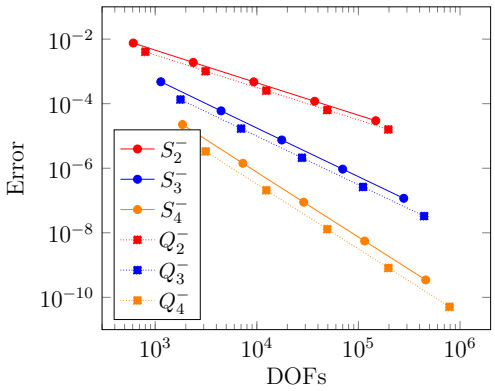
In a similar fashion as for the primal formulation, we can create the mixed formulation of the Poisson problem that we saw in Eq. (2.2).

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 $(\mathcal{Q}_r^- \Lambda^{n-1}, \mathcal{Q}_r^- \Lambda^n)$ and $(\mathcal{S}_r^- \Lambda^{n-1}, \mathcal{S}_r^- \Lambda^n)$ for dimensions $n = 2$ and $n = 3$. Note that the mathematical notation here calls for $\mathcal{Q}_r^- \Lambda^{n-1}$ to be paired with $\mathcal{Q}_r^- \Lambda^n$, 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 [5, 4] with a sparse direct LU factorization using iterative refinement in order 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.

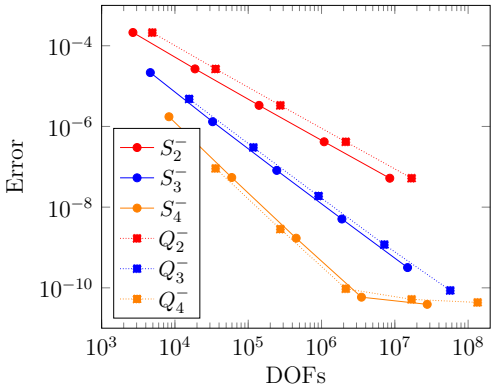
The empirical convergence results for the primal and mixed formulations of the Poisson problems can be seen in Figs. 3.3a to 3.3d. In each of the subfigures of Fig. 3.3, we see that independent of the performance of each element, the $\mathcal{S}_r^-$ and $\mathcal{Q}_r^-$ have parallel trendlines,



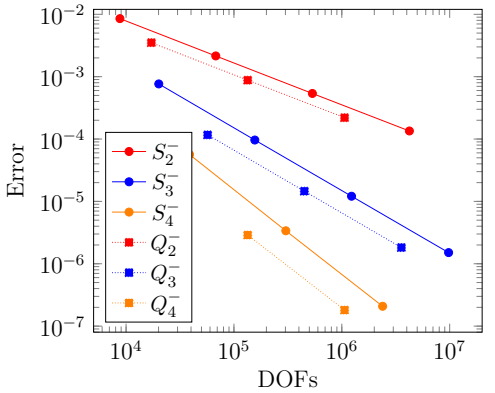
(a) 2D primal Poisson convergence analysis.



(b) 2D mixed Poisson convergence analysis.



(c) 3D primal Poisson convergence analysis.



(d) 3D mixed Poisson convergence analysis.

Figure 3.3. 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.

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 $\mathcal{S}_3^-$ instead of $\mathcal{Q}_2^-$ will attain a higher accuracy for essentially the same number of DOFs.

In Fig. 3.4 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 graphs, we see good evidence in Figs. 3.4a and 3.4b that the trimmed serendipity and tensor product elements compute solutions at a similar speed based on the number of

DOFs. In Fig. 3.4b 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 Fig. 3.4c. Further evidence of this is seen in Fig. 3.4d. 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 3.3.

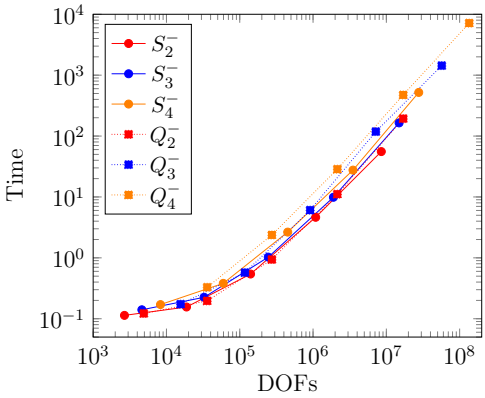
Table 3.3. Expressing the number of nonzero entries in the matrices used to compute solutions to the primal and mixed formulations of the Poisson problem. The data shown here represents order 4 elements, where the meshes are either 128^2 or 64^3 depending on the dimension of the space. The first row 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.

$\mathcal{Q}_4^-$ Elements			
Primal $n = 2$	Primal $n = 3$	Mixed $n = 2$	Mixed $n = 3$
381825	143992308	2096704	989178624
5.5130876e-06	4.9973590e-07	3.3813064e-06	2.1836045e-07

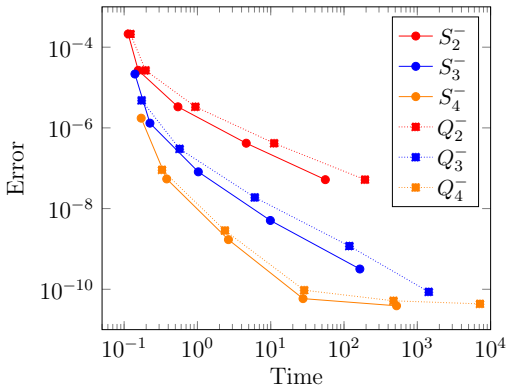
$\mathcal{S}_4^-$ Elements			
Primal $n = 2$	Primal $n = 3$	Mixed $n = 2$	Mixed $n = 3$
156625	17148900	848624	107771596
8.9758414e-06	1.3943711e-06	4.0144191e-06	2.9862278e-07

3.3.3. Cavity Resonator

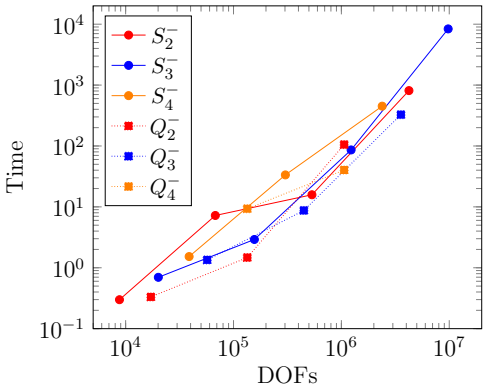
The last numerical experiment that we give here is the cavity resonator problem, making use of the $H(\text{curl})$ elements in 3D. We a pose a Maxwell eigenvalue problem on the domain $\Omega = [0, 1]^3$ with perfectly conducting boundary conditions, yielding an eigenvalue problem where λ represents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



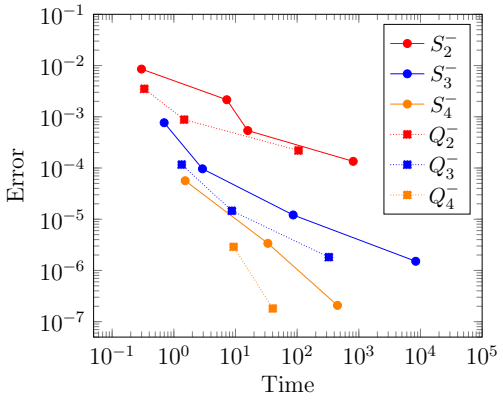
(a) 3D primal Poisson Time vs DOFs



(b) 3D primal Poisson Error vs Time



(c) 3D mixed Poisson Time vs DOFs



(d) 3D Mixed Poisson Error vs Time

Figure 3.4. 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.

electric field (i.e. eigenvalues) and E represents the electric field (i.e. eigenfunctions):

$$\nabla \cdot E = 0 \text{ in } \Omega$$

$$\nabla \times \nabla \times E = \lambda E \text{ in } \Omega$$

$$E \times n = 0 \text{ on } \partial\Omega.$$

We consider the weak formulation of this problem [37], where ω represents the resonances (i.e. eigenvalues) and E represents the electric field (i.e. eigenfunctions):

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

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 [70].

In Table 3.4, 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 $\mathcal{Q}^-$ $H(\text{curl})$ elements while the bottom half shows values from using the corresponding $\mathcal{S}^-$ 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 off earlier eigenvalue works [20], 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 $\mathcal{S}^-$ and $\mathcal{Q}^-$ elements. The "–" entries in the table indicate the eigenvalue solver did not find that specific eigenvalue in the allowed number of iterations; we set the solver to iterate a sufficient number of times to find the first 15 eigenvalue-eigenvector pairs.

The experiment was done by using Firedrake to create the mass and stiffness matrices as petsc4py objects [17, 16, 28], then using slepc4py [44, 71] to do the eigenvalue analysis.

Table 3.4. A comparison of how $\mathcal{Q}_2^-$ and $\mathcal{S}_2^-$ finite elements solve the Maxwell cavity resonator eigenvalue problem, $\langle \text{curl}(F), \text{curl}(E)\rangle = \omega^2\langle F, E\rangle$. An eigenvalue found with the same error multiple times was condensed to a single row. Numbers in parentheses next to the actual eigenvalue are the number of times we found an approximation of the actual eigenvalue. The columns labeled $N = 4, 8, 16, 32$ are giving the approximate eigenvalues found on a mesh of size $N \times N \times N$. The values in parentheses in these columns indicates the rate of convergence for that approximate eigenvalue.

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

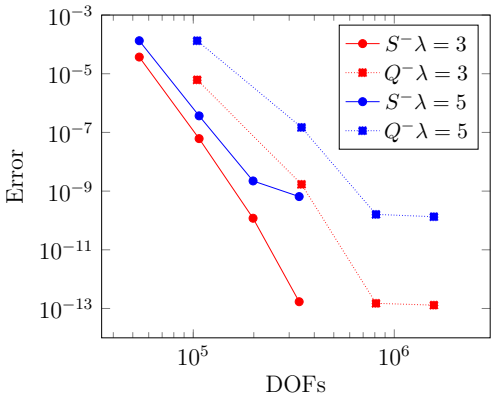
$\mathcal{S}_2^-$ $H(\text{curl})$ Elements				
Actual (Count)	N = 4	N = 8	N = 16	N = 32
2 (3)	2.001092	2.000066 (4.05)	2.000004 (4.04)	2.000000 (4.00)
3 (2)	3.009018	3.000586 (3.94)	3.000037 (3.99)	3.000002 (4.21)
5 (3)	5.032027	5.002097 (3.93)	5.000133 (3.98)	5.000008 (4.06)
5 (1)	5.032027	5.002097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

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

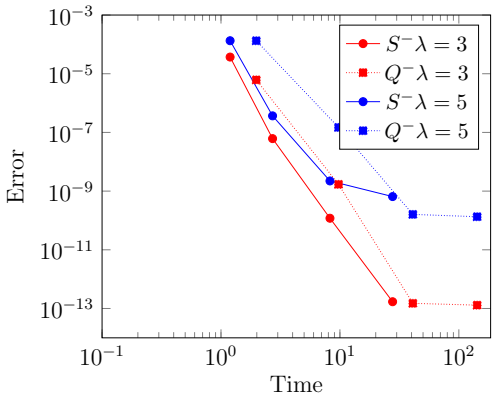
columns, not due to the elements that we use or the SLEPc solver that is called. We do not report these eigenvalues.

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. On the other hand, this loss of accuracy from using trimmed serendipity elements is offset by a reduction in required time to solve for the requested eigenvalues. At every mesh refinement level, trimmed serendipity elements have nearly half the DOFs of tensor product elements, and correspondingly, require approximately half the time per iteration to solve for the eigenvalues (outside of the case $N = 4$). At higher orders, we expect that this will be even more exaggerated.

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



(a) Eigenvalue error analysis.



(b) Eigenvalue time analysis.

Figure 3.5. Results for solving for $\lambda = 3$ and $\lambda = 5$ using Firedrake and SLEPc by increasing the order from 2 to 5. Error is calculated as the absolute value of the error between the actual eigenvalue and the approximated eigenvalue.

The error results shown in Fig. 3.5a 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 Fig. 3.5b 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.

3.4. 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 previously [40] 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 $\mathcal{Q}_r^-$, we make a choice when using trimmed serendipity elements $\mathcal{S}_r^-$ 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 off of the DOFs that they require. Consider the 3D mixed formulation of the Poisson problem while focusing on DOFs vs Error given in Fig. 3.3d. The tensor product elements $\mathcal{Q}_2^-$ required a similar number of DOFs as the trimmed serendipity elements $\mathcal{S}_3^-$. Compared this way, the trimmed serendipity elements provide an extra order of magnitude of accuracy over the tensor product element. Furthermore, in Fig. 3.4d the time required for $\mathcal{S}_3^-$ and $\mathcal{Q}_2^-$ was also approximately equal. Thus while it is helpful to compare $\mathcal{Q}_r^-$ and $\mathcal{S}_r^-$ to see that the trimmed serendipity elements have the expected convergence behavior, a more practical computational comparison is between $\mathcal{Q}_r^-$ and $\mathcal{S}_{r+1}^-$.

The eigenvalue problem yields another example of comparing the tensor product and trimmed serendipity elements, where instead of refining the mesh, we refined the order of the element used. Just as in the mixed Poisson problem, we again see that Fig. 3.5a shows $\mathcal{S}_2^-$ has a higher error for $\lambda = 3$ than $\mathcal{Q}_2^-$. However comparing against where the DOFs are approximately equal leads to a comparison between $\mathcal{S}_3^-$ and $\mathcal{Q}_2^-$. In this scenario, we had that $\mathcal{Q}_2^-$ required 104544 DOFs yielding an error of 1.33×10^{-4} for $\lambda = 5$ while $\mathcal{S}_3^-$ required 106896 and achieved an error of 3.67×10^{-7} for $\lambda = 5$. In this case, we note that the time required for $\mathcal{S}_3^-$ did require more time to solve, using about 2.71 seconds while the $\mathcal{Q}_2^-$ required 1.98 seconds.

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.

CHAPTER 4

Application of Trimmed Serendipity Elements: The Monodomain Equation

While the previous results demonstrated the numerical effectiveness of applying trimmed serendipity elements to common test problems, we are also interested in what trimmed serendipity elements can do for speeding up computations in a practical setting. The monodomain equation solved in [80] is one such model that can be used to simulate electric flow through a human heart. The variable of interest u is the membrane potential, σ is the conductivity tensor (a $n \times n$ diagonal matrix representing conductivity in each direction), I_{ion} is the current due to flows of ions through channels in the cell membranes (determined as a function of the potential depedent upon what cellular model we use), C_m is the specific capacitance of the cell membrane, and χ is the surface area to volume ratio. The strong form of the equation is:

$$\chi\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) = \nabla \cdot \sigma \nabla u.$$

We derive a weak formulation of this as follows:

$$\begin{aligned}
\chi\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) &= \nabla \cdot \sigma \nabla u \\
\chi v\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) &= v \nabla \cdot \sigma \nabla u \\
\int_\Omega \chi v\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) dx &= \int_\Omega v \nabla \cdot \sigma \nabla u dx \\
\int_\Omega \chi v\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big) dx &= \int_\Omega \nabla \cdot (v \sigma \nabla u) dx - \int_\Omega \nabla v \cdot \sigma \nabla u dx
\end{aligned}$$

$$\int_\Omega \chi v\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big)dx = \int_{\partial\Omega} v\sigma\nabla u \cdot n ds - \int_\Omega \nabla v \cdot \sigma \nabla u dx$$

$$\int_\Omega \chi v\big(C_m \frac{\partial u}{\partial t} + I_{\text{ion}}(u)\big)dx = -\int_\Omega \nabla v \cdot \sigma \nabla u dx \tag{4.1}$$

Note that in the third line to the fourth line, we use the identity $f(\nabla\cdot g) = \nabla\cdot(fg) - \nabla f \cdot g$, substituting $f = v$ and $g = \sigma\nabla u$. From the fourth line to the fifth we apply the generalized Stokes' Theorem (Gauss Divergence Theorem in multivariable calculus), $\int_\Omega dF = \int_{\partial\Omega} F$. Finally, from the fifth line to the sixth, we require one of two conditions. Either that v is chosen from a test function space such that $v|_{\partial\Omega} = 0$, where $\partial\Omega$ is the boundary of Ω, or that $\nabla u \cdot \mathbf{n} = 0$, where $\mathbf{n}$ is the outward unit normal vector. We choose to apply a no-flux boundary condition such that $\nabla u \cdot \mathbf{n} = 0$.

When solving the weak form of the monodomain equation in Firedrake, we could do it in two ways. The first is by applying an operator split and using a backward Euler method on the time derivative, and the second is applying a Firedrake implementation of Runge-Kutta solvers called Irksome. In the first scenario, this means we solve the discretized equation

$$\int_\Omega v u^{(n+1)} dx + \frac{\Delta t}{C_m}\int_\Omega I_{\text{ion}}(u) v dx = \int_\Omega v u^{(n)} dx - \frac{\Delta t}{\chi C_m}\int_\Omega \nabla v \cdot \sigma \nabla u dx. \tag{4.2}$$

Otherwise, we will solve the weak form of the monodomain Eq. (4.1) directly by setting up the proper Runge-Kutta system.

4.1. Cell Models and Ionic Flow

In the monodomain equation, the remaining term that needs to be defined is the function $I_{\text{ion}}(u)$, which is the function that describes the sum of all of the transmembrane ionic currents. This ionic flow function is normally a solution to a set of ODEs, though the exact

number and definition of those ODEs can change based on how simple or complex the user wants to make the model. In the simplest scenario, one can set the cell model to be a simple cubic polynomial such as

$$I_{\text{ion}}(u) = A^2(u - u_{\text{rest}})(u - u_{\text{th}})(u - u_{\text{peak}}) \tag{4.3}$$

where A is the upstroke velocity, v_{rest} is the resting potential, v_{th} is the threshold potential, and v_{max} is the maximum potential. This model [75] is described as a model accounting for the macroscopic behavior that is seen in the heart tissue. While it is simple to implement, it does not allow for repolarization (the process of the transmembrane potentials going back down to the resting potential). Thus, as a first pass, this model is acceptable to use for proof of concept, but will not suffice as a long term way to reproduce results or to actually simulate how electric current flows through the heart.

The second model that we describe is called the FitzHugh-Nagumo (FHN) model [36]. The original FHN model had some issues with repolarizing, but it has been modified since 1961 to be more accurate to the dynamics that are seen in cardiac electrophysiology modeling. The version of the model that we give as an example [49, 42] sets up using an I_{ion} function that requires a concentration variable.

In this case, we define our concentration variable as c_1 and it is controlled by

$$\frac{\partial c_1}{\partial t} = \epsilon(u + \beta - \gamma c_1).$$

Then the function I_{ion} is determined by

$$I_{\text{ion}} = \frac{1}{\epsilon}\left(u - \frac{u^3}{3} - c_1\right).$$

In this example, we take $\epsilon = 0.1$, $\beta = 1.0$, $\gamma = 0.5$, σ is a 2×2 identity matrix, $\chi = 1.0$, and $C_m = 1.0$. To use this model, we could solve using an operator splitting scheme where we first solve for c_1 at a given time step, then use the results from c_1 in I_{ion}, which is plugged into the monodomain equation. The other way that we could solve this, which will be the method that we give results for, is using Irksome within Firedrake to set up a full system of differential equations and solve using a Runge-Kutta method.

To set up the system for Firedrake to plug into Irksome, we must write the PDE for c_1 in a weak form, then we add the weak form of c_1 with the weak form for the monodomain equation. We show an example of what this looks like in Firedrake in Listing 4.1, where lines 24 – 29 illustrate how to set up a weak form for the entire PDE system that will be solved.

The next model that we have for the I_{ion} function is to model it using the Hodgkin-Huxley model. This was first described in 1952, and was used to describe a large nerve cell found in a squid. While not directly useful in modeling electric potential in the human heart, it gives a good simplified version of the state of the art model.

In the Hodgkin-Huxley model, they describe 3 ionic currents, the sum of which determines the I_{ion} function. These currents are potassium, sodium, and a leakage. As we'll see later when discussing the Ten Tusscher model, the potassium and sodium currents in the Hodgkin-Huxley model are gated, while the leakage current is time independent and given by a simple linear function. These currents are given by

$$I_{\text{Na}} = \overline{G}_{\text{Na}} m^3 h (\nu - \nu_{\text{Na}}) \tag{4.4}$$

$$I_{\text{K}} = \overline{G}_{\text{K}} n^4 (\nu - \nu_{\text{K}}) \tag{4.5}$$

$$I_{\text{L}} = G_{\text{L}} (\nu - \nu_{\text{L}}). \tag{4.6}$$

Listing 4.1. Firedrake implementation of a monodomain equation solver using the FHN model.

```
mesh = RectangleMesh(100, 100, 70, 70, quadrilateral=True)
polyOrder = 1

V = FunctionSpace(mesh, "Q", polyOrder)
Z = V * V
x, y = SpatialCoordinate(mesh)

InitialPotential = conditional(x < 3.5, Constant(2.0), Constant(-1.28791))
InitialCell = conditional(And(And( 31<= x, x < 39),
                And(0 <= y, y < 35)), Constant(2.0), Constant(-0.5758))

#Constants ...

butcher_tableau = RadauIIA(1)
ns = butcher_tableau.num_stages

uu = Function(Z)
vu, vc = TestFunctions(Z)
uu.sub(0).interpolate(InitialPotential)
uu.sub(1).interpolate(InitialCell)
(u, c1) = split(uu)

Fu = inner(chi * capacitance * Dt(u), vu)*dx + inner(grad(u),
    sigma * grad(vu))*dx +
    inner((chi/eps) * (-u + (u**3 / 3) + c1), vu)*dx
Fc = inner(Dt(c1), vc)*dx - inner(eps * u, vc)*dx -
    inner(beta * eps, vc)*dx + inner(gamma * eps * c1, vc)*dx
F = Fu + Fc

stepper = TimeStepper(F, butcher_tableau, t, dt, uu,
                  solver_parameters=params)

for j in range(300):
    stepper.advance()
    t.assign(float(t) + float(dt))
```

In this model, we have that $I_{\text{ion}} = I_{\text{Na}} + I_{\text{K}} + I_{\text{L}}$, $\nu = v - v_{\text{eq}}$, and similarly $\nu_{\text{Na}} = v_{\text{Na}} - v_{\text{eq}}, \nu_{\text{K}} = v_{\text{K}} - v_{\text{eq}}$, and $\nu_L = v_{\text{L}} - v_{\text{eq}}$. The values of $v_{\text{Na}}, v_{\text{K}}$, and v_{L} are the true equilibrium potentials for those currents. The values of $\overline{G}_{\text{Na}}, \overline{G}_{\text{K}}$, and G_{L} are the maximum conductances for each current, and m, n, and h are gate variables that we describe below.

The channels that ions flow through in cells can be controlled by gates. Each gate in a channel must be open in order for the ions to flow through cells, and each channel may have multiple types of gates or multiple gates of the same type.

Consider the following scenario: denote g as the probability that a channel is open and make a steady-state assumption (the validity of which is left aside). If α is the rate at which the gates open and β is the rate at which gates close (generally functions of the transmembrane potential u of the cell), then write $g_\infty = \frac{\alpha}{\alpha+\beta}$. Then at a given time, the probability that a channel is completely open is g_∞. If the channel consists of n similar subunits, then the probability O that the channel is open is g_∞^n.

This can be extended to having subunits in a channel with different behaviors, each with different probabilities of being open. If we have two types of subunits, n of one type with probability g of being open and m of the second type with probability h of being open, then $O = g^n h^m$. Finally, the current through the membrane is given by the product of the maximum current $G_{\max}$ and the proportion of all open channels O. This looks like

$$I = G_{\max} O(v - v_{\text{eq}})$$

where O are products like described above, v_{eq} is the equilibrium potential for the specific ion, and $G_{\max}$ is the maximum conductance (the conductance if all channels were open).

The evolution of channels in time when not assuming a steady-state is more complicated. If we assume that $\alpha(u)$ is the rate that a gate opens with respect to the potential and that $\beta(u)$ is the rate that a gate closes, then we may write an arbitrary gating equation as

$$\frac{dg}{dt} = \alpha(u)(1-g) - \beta(u)g. \tag{4.7}$$

In the case where α and β do not depend on the potential, then we can turn this differential equation into

$$\frac{dg}{dt} = \frac{g_\infty - g}{\tau_g}$$

yielding a solution

$$g(t) = g_\infty + (g_0 - g_\infty)e^{-t/\tau_g}.$$

Using these gating variables, we now define the Ten Tusscher cellular model, which is the model that recent benchmark studies use when computing approximate solutions to the monodomain equation. This model underwent two different iterations [78, 77], receiving modifications the second time around to more accurately deal with the calcium concentrations. According to one benchmark study [66], some appropriate test values for the constants are given in Table 4.1.

Table 4.1. The values for the surface area to volume ratio, capacitance, and various conductivities in each direction, where σ_{il} is the intra-longitudinal direction, σ_{it} is in the intra-tranversal direction, σ_{el} is the extra-longitudinal direction, and σ_{et} is the extra-transversal direction.

χ	140
C_m	0.01
σ_{il}	0.17
σ_{it}	0.019
σ_{el}	0.62
σ_{et}	0.24

The conductivity tensor σ is created by applying the formula $\sigma = \sigma_i\sigma_e(\sigma_e+\sigma_i)^{-1}$ for both the transversal and longitudinal directions. We apply an initial stimulation of 2 ms at $50,000\mu$A cm^{-3}.

In the original work, the I_{ion} function was given as a sum of all of the transmembrane currents as

$$I_{\text{ion}} = I_{\text{Na}} + I_{\text{K1}} + I_{\text{to}} + I_{\text{Kr}} + I_{\text{Ks}} + I_{\text{CaL}} + I_{\text{NaCa}} + I_{\text{NaK}} + I_{\text{pCa}} + I_{\text{pK}} + I_{\text{bCa}} + I_{\text{bNa}}.$$

Each of these are defined in Chapter 7, with some definitions coming from the original work in 2004 and others coming from the updated definitions given in 2006.

The ionic flows that depend on the gating variables will vary based on the exact cellular model that is chosen. Since the gating variables are all determined by ODEs, this implies that the ionic flow function will require a method for solving the ODEs before we can solve the monodomain equation at any given time step. Though this model is the current state of the art model, we do not give any results using it here. We discuss this model more in the discussion section at the end of the chapter as a possible avenue forward.

4.2. Results

The FHN model is a good starting model for behavior of a wave dictated by the monodomain equation. The specific set up for the FHN model [42] uses a domain of 70×70 with time steps between 10^{-3} and 10^{-1}. For 2D results, we discretize this domain with uniformly sized 100 cells in each direction. In 3D, we extend to a $70 \times 70 \times 70$ domain, again with uniformly sized 100 cells in each direction.

We compute approximate solutions to the FHN model of the monodomain equation using the same setup for computing as used in the previous chapter. The only modification we make is that 3D calculations are done using 47 processors instead of the 24 used in the 2D calculations. The experiments we present here focus on the analysis of a relatively new preconditioner and the difference in computation time between the serendipity and tensor product elements. Recall that the trimmed serendipity elements and serendipity elements for H^1 elements are equal, and thus we will simply refer to them as serendipity elements in this chapter.

We use the 2D FHN model as a numerical experiment to ensure that we are attaining the results that we expect. For this problem, we assign initial conditions for both the potential variable u and the concentration variable c as

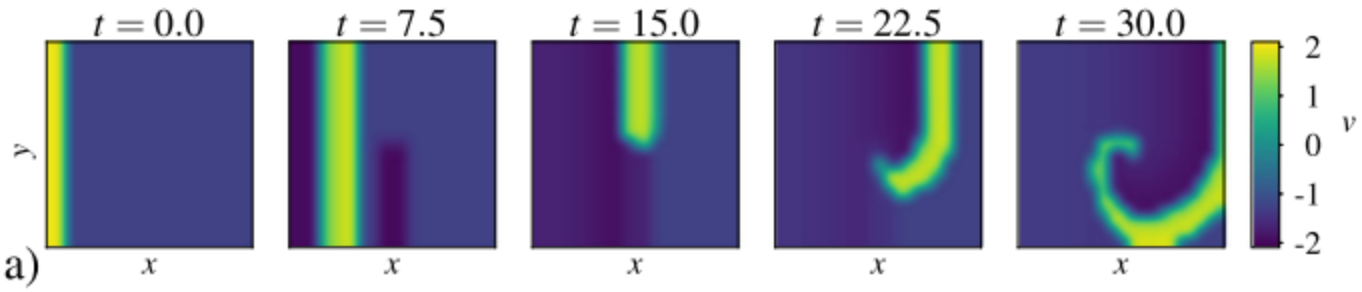


Figure 4.1. Reference solution for the FHN model. [42]

$$u(0,x) = \begin{cases} 2 & x < 3.5 \\ -1.28791 & \text{otherwise} \end{cases}$$

$$c(0,x) = \begin{cases} 2 & x \in [31,39), y < 35 \\ -0.5758 & \text{otherwise} \end{cases}.$$

In this experiment, we see a spiral wave form during the first 30 seconds of simulated time. The expected solution is shown in Fig. 4.1 while our computed approximation is given in Fig. 4.2. A visual inspection shows small differences that we attribute to differences in the size of mesh elements, but overall that the solver in Firedrake is attaining the expected behavior.

As we see in Table 4.2, the serendipity elements are able to finish the computations either at an approximately equal speed in the order 1 case, or at a faster rate as the order increases. The order 1 case is the expected result since $\mathcal{S}_1 = \mathcal{Q}_1^-$. We also find that as the order increases, the serendipity elements continue to reduce the computation time in comparison to the tensor product elements.

The preconditioner that we chose to use was the recently defined Rana preconditioner [60]. The goal when using this preconditioner was to keep the inner iterations of the solve to a relatively stable value independent of the number of Runge-Kutta stages used to solve the

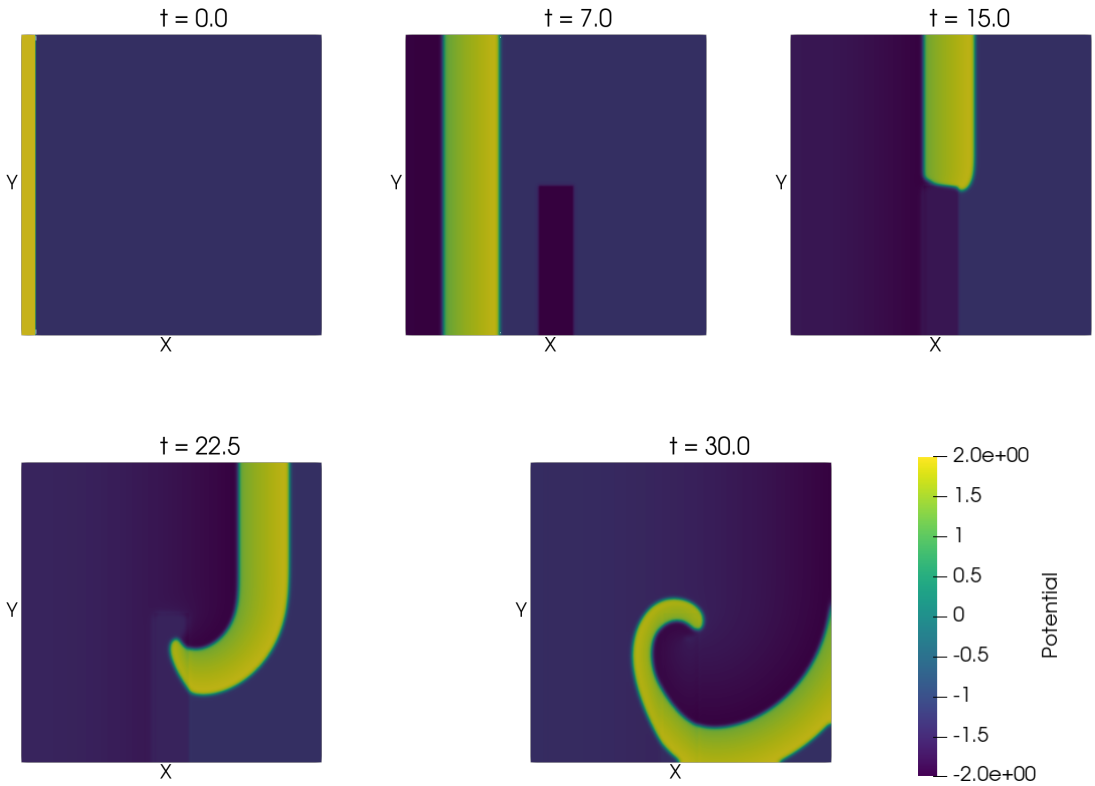


Figure 4.2. The approximate computed solution to the monodomain equation with the FHN model using order 2 Serendipity elements in Firedrake.

Table 4.2. Computation time requirements for solving the 2D monodomain equation using the FHN model. Results are obtained using a RadauIIA Runge-Kutta method, and shown for both 2 and 3 stage methods.

2 Stage Solve			3 Stage Solve		
	$\mathcal{Q}^-$	$\mathcal{S}$		$\mathcal{Q}^-$	$\mathcal{S}$
Order	Time	Time	Order	Time	Time
1	155.9335	157.4953	1	243.4092	245.5694
2	251.1965	230.1182	2	375.8359	359.3904
3	548.1152	392.2462	3	858.7056	669.3561

ODE system. The results in Table 4.3 show that for both order 1 and 2 elements, regardless of which element type, the iteration counts between 2 and 3 stage methods are very similar. For order 3 methods, we do start to see a slightly larger, though still small, variation in the number of iterations required for the geometric multigrid portion of the solver.

Table 4.3. Iteration analysis of the preconditioner used to compute the approximate solution to the FHN model. Results are obtained using the RadauIIA Runge-Kutta method, and shown for both 2 and 3 stage methods.

2 Stage Solve					3 Stage Solve				
	$\mathcal{Q}^-$		$\mathcal{S}$			$\mathcal{Q}^-$		$\mathcal{S}$	
Order	Newton	GMres	Newton	GMres	Order	Newton	GMres	Newton	GMres
1	3.9136	8.382	3.9136	8.382	1	3.8803	8.515	3.8803	8.515
2	4.0066	8.164	4.0066	8.5588	2	3.777	8.346	3.7807	9.15
3	4.0033	9.392	4.0066	9.6558	3	4	10.107	4.0033	10.244

Finally, we map the activation time of the domain. The activation time is defined to be the time at which a location first hits 0 potential. As the wave transitions from a solid wavefront to a spiral, we can see in Fig. 4.3 the activation times in the domain stop being uniform in the y-direction. In particular, we can see the spiral wave's progression through the domain. We are also able to find a few uncolored zones on the right half of the domain that represent areas where the domain never achieved a potential greater than 0.

Figure 4.3. Activation time plots for the 2D monodomain equation using the FHN model. The left plot shows the entire domain while the right plot shows the zoomed central portion to investigate the spiral. In the left plot, the initial condition starts activated on the left side of the domain, while all other white areas represent zones that are never activated.

To experiment in 3D, we start with the following as the initial conditions that act as a straightforward extension of the 2D initial conditions to 3D:

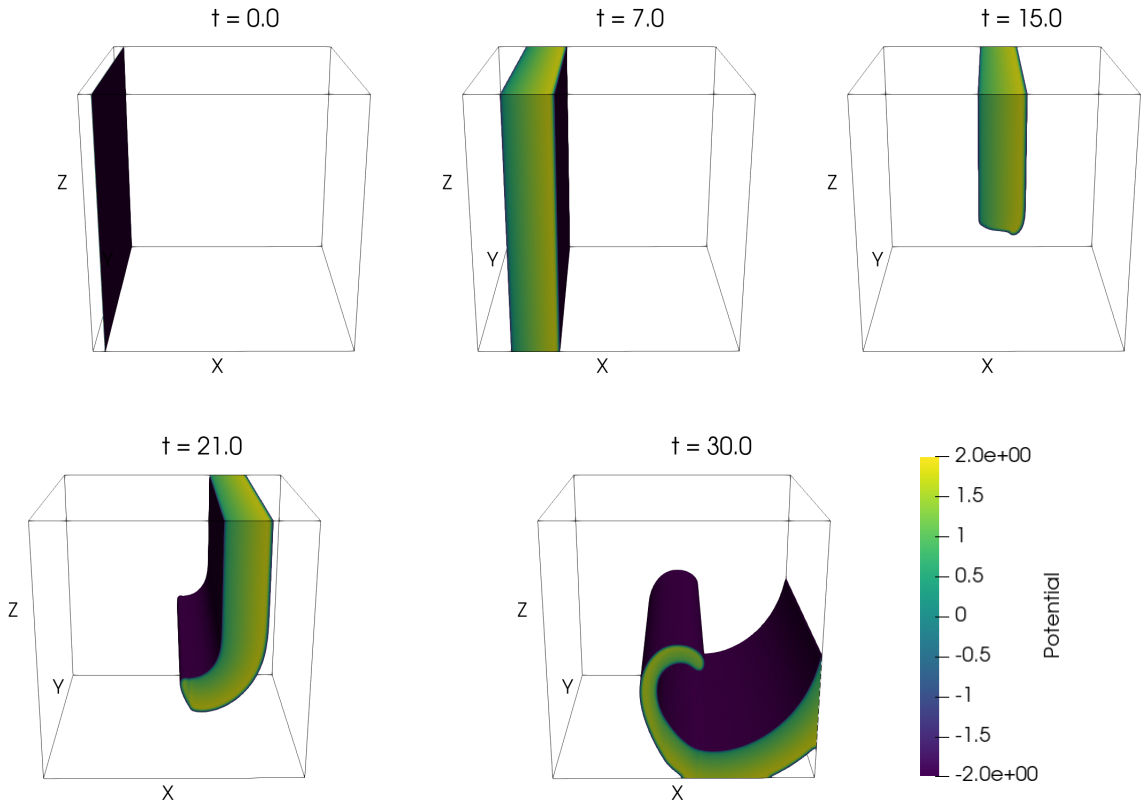


Figure 4.4. Evolution of wave in 3D for the straightforward extension of initial conditions.

.

$$u(0,x) = \begin{cases} 2 & x < 3.5 \\ -1.28791 & \text{otherwise} \end{cases}$$

$$c(0,x) = \begin{cases} 2 & x \in [31,39), z \in [0,35) \\ -0.5758 & \text{otherwise} \end{cases} .$$

This extension of the 2D case yields similar looking results in 3D. In particular, taking the $x-z$ slices of the domain will result in the same sequence of images as we showed above in the 2D experiment. To illustrate the evolution of the wave in the 3D cube, we give a sequence of isovolume plots, shown in Fig. 4.4.

As we saw in the previous chapter, the (trimmed) serendipity elements should be able to gain some significant time savings in comparison to the tensor product elements. The timing results for the 3D model are shown in Table 4.4. While we see that the order 2 elements behave as expected with the serendipity elements requiring less time to run, the order 1 elements do not have the approximately equal behavior that is expected.

Table 4.4. Computation time requirements for solving the 3D monodomain equation using the FHN model. Results are obtained using a RadauIIA Runge-Kutta method. The 3 stage method for order 2 becomes very time intensive for the tensor product elements, thus we omit it here.

2 Stage Solve

	$\mathcal{Q}^-$	$\mathcal{S}$
Order	Time	Time
1	4246.2523	6511.608
2	84698.416	54986.531

3 Stage Solve

	$\mathcal{Q}^-$	$\mathcal{S}$
Order	Time	Time
1	7287.874	11397.718

However, while the timing results in order 1 require more investigation, the Runge-Kutta results in Table 4.5 for the 3D model are similar to the 2D model. Specifically, we see that going from a 2 stage method to a 3 stage method results in very little change in either the newton iterations or the geometric multigrid iterations that the solver needs. These results indicate that the Rana preconditioner is a good preconditioner for the monodomain equation with the FHN model in both 2D and 3D.

Table 4.5. Iteration analysis of the preconditioner used to compute the approximate solution to the 3D FHN model. Results are obtained using the RadauIIA Runge-Kutta method, and shown for both 2 and 3 stage methods for element order and stage pairings that did not require prohibitively long computation times.

2 Stage Solve

	$\mathcal{Q}^-$		$\mathcal{S}$	
Order	Newton	GMres	Newton	GMres
1	3.9136	8.6723	3.9136	8.6723
2	4.0066	8.5505	4.0066	9.3922

3 Stage Solve

	$\mathcal{Q}^-$		$\mathcal{S}$	
Order	Newton	GMres	Newton	GMres
1	3.88	8.708	3.88	8.708

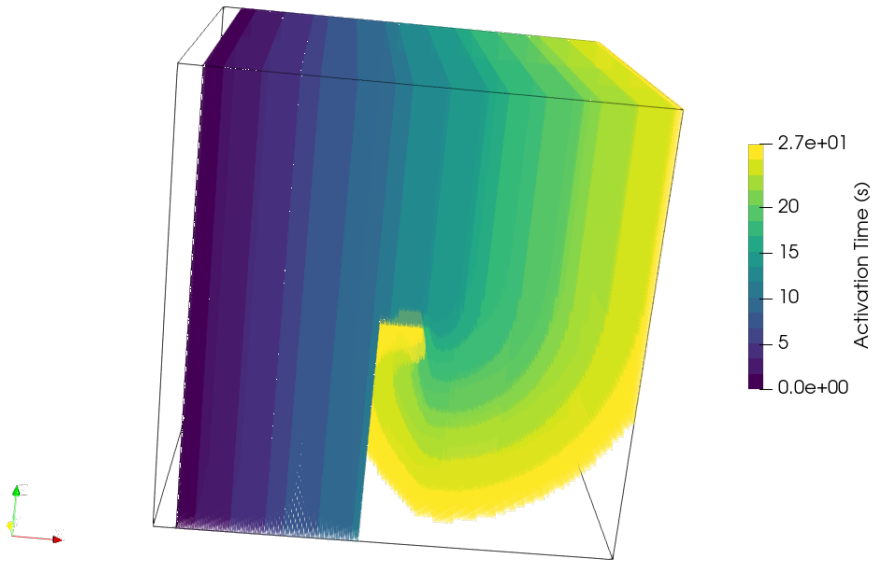


Figure 4.5. The activation time plot for the 3D experiment. Times go from 1 to 27 simulated seconds. The left slice is the initial condition that is automatically activated, while the other uncolored zones represent areas that never achieve positive potential.

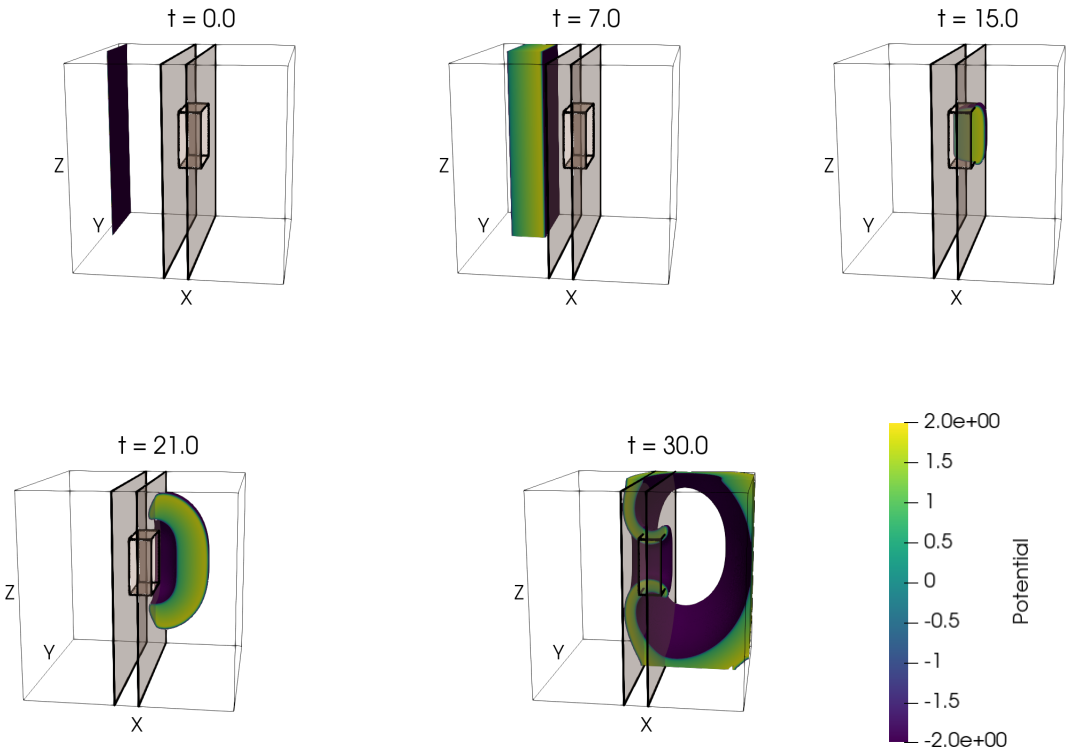
Finally for the basic extension to 3D, we show activation time results. The activation time here is a good initial start at considering what activation times might look like in 3D for more interesting meshes using the Ten Tusscher model.

We are also interested another qualitative experiment with slightly different initial conditions. Consider the initial conditions as

$$u(0,x) = \begin{cases} 2 & x < 3.5 \\ -1.28791 & \text{otherwise} \end{cases}$$

$$c(0,x) = \begin{cases} 2 & x \in [31,39) \backslash \mathrm{B} \\ -0.5758 & \text{otherwise} \end{cases}$$

Figure 4.6. Evolution of wave in 3D with the modified initial conditions.



where $B = x \in [31, 39) \cap y \in [30, 50) \cap z \in [30, 50)$. In this case, we see a wave of potential flow through the left half of the domain. When the wave hits the resistance of the concentration variable, it goes through the hole that is slightly off-center. After that, we see the wave hitting the end of the cube at $x = 70$ causing it to curve back in on itself. This causes a similar sort of spiraling to what we see in 2D but in multiple directions, and it is not uniform in each direction due to the hole in the concentration variable being off-center.

4.3. Discussion

The monodomain equation is one practical example of a PDE that can use the serendipity elements. We can see that it affords the user an ability to speed up computation time, and

based off the analysis in Chapter 3, that it comes with a small increase in error when compared to tensor product elements. This problem in particular is one that cardiac researchers are interested in being able to solve as quickly as possible, so being able to complete computations at 65 percent of the computation time may be significant.

The numerical analyses shown in this chapter contribute to testing a recently derived preconditioner. They show that this preconditioner may be a good choice for solving the monodomain equation with the FHN model without seeing a large increase in the number of iterations involved, regardless of the number of Runge-Kutta stages that are used. Furthermore, we gave illustrations of the physical results of activation time for this particular model.

In the future, this problem has two clear next steps. The first is to understand why the implementation of the first order serendipity elements in Firedrake are causing a large computation time increase in comparison to the first order tensor product elements. As mentioned previously, these elements are mathematically the same, and that is illustrated by the fact that the newton and geometric multigrid iterations within the preconditioner yield the exact same average iteration counts. The most likely cause is some sort of extra overhead in the serendipity element code that the overall computation time doesn't manage to recover. It is also possible that the H^1 serendipity elements in Firedrake have some other type of issue in the code, but that is unlikely due to the fact that the elements give the same mathematical results as the tensor product elements do.

The second step is to extend the preconditioner to be applicable to the full Ten Tusscher model. While the Ten Tusscher model is the state-of-the-art model used within the monodomain equation, it has more complexity within it. With order 2 serendipity elements using 2 Runge-Kutta stages already using more than two-thirds of a day to run for the FHN model, increasing the computational complexity of the system for a time-sensitive problem

would be difficult to sell to practitioners. Therefore, even just using the current preconditioner extended to each of the ODEs in the Ten Tusscher model would likely not be what cardiac researchers would want to use. However, it is important to be able to use the Ten Tusscher model in order to achieve an accurate model of electric flow through the heart, as otherwise, the micro-scale dynamics are not fully captured.

CHAPTER 5

Fractional Differential Equations using Discrete Exterior Calculus

Combining fractional calculus and DEC comes with its own set of challenges. Part of the design of DEC is that it is a set of tools that work in a local setting, with no need to understand global distances. Alternatively, as described earlier, fractional differential equations inherently require the use of nonlocal information. This section will present one possible solution to the problem of how to compute a fractional derivative in the setting of DEC.

5.1. The Fractional Discrete Exterior Derivative

Our goal is to create a definition that imitates the fractional Caputo derivative. While doing this, our definition for fractional discrete exterior derivative should also act in a similar fashion to the normal discrete exterior derivative map. Thus we have some properties that we seek to preserve:

(1) That the operator $\mathbb{D}_0^s$ maps a discrete 0-form f with values at the vertices of a mesh to a discrete 1-form $\mathbb{D}_0^s f$, taking values at the edges of the mesh. Similarly, $\mathbb{D}_1^s$ should map a discrete 1-form to a discrete 2-form taking values at the faces.

(2) $\mathbb{D}_0^s$ maps a constant 0-form to 0 at each edge of the mesh.

(3) As the fractional order goes to 1, we recover $\mathbb{D}_k$.

Note: the Caputo derivative does not yield the identity operator in the limit $s \to 0$, so it is not a property that we try to emulate.

We mention this here before illustrating the design of our definition. In choosing to use DEC to tackle fractional differential equations, we must make certain design choices that are going to lead to complicating portions of DEC that are normally straightforward. We

believe that the extra complications are worth it to make use of the framework that DEC gives us in solving differential equations. In particular, the ability to computing distances between p-simplices is a non-trivial requirement for the computation.

5.1.1. Defining the Fractional Discrete Exterior Derivative

As we build our definition, we will start with the Caputo derivative definition and replace smooth operators with DEC operators wherever possible. For convenience, we restrict ourselves to the setting where the two-sided fractional Caputo derivative is:

$$D^s f(x) = \frac{1}{\Gamma(1-s)} \int_{[a,b]\setminus\{x\}} \frac{f'(\tau)d\tau}{|x-\tau|^s}.$$

whenever $s \in (0,1)$.

Before we begin discretizing, notice that the Caputo derivative operator can be decomposed into a sequence of standard operations working inside out. First we differentiate f, then we integrate a weighted version of the derivative. Thus we will end up discretizing this in the same sequence of steps.

To discretize, we first apply a discrete exterior derivative to a 0-form α, yielding a discrete 1-form $\mathbb{D}_0\alpha$. The discrete 1-form should take values at each edge, so for the remaining steps we will focus on what the fractional operator computes at a specific edge x.

Next, we discretize the integral with a kernel of $\frac{1}{|x-t|^s}$. This is similar to discretizing a convolution of $\mathbb{D}_0\alpha$ against the convolution kernel of $\frac{1}{|x-t|^s}$. This translates to computing a weighted sum

$$\sum_{i=0}^{|E|-1} \frac{1}{||x-t_i||^s} \mathbb{D}_0\alpha$$

where t_i runs over each of the edges in the mesh, x is the edge of interest, and $||x-t_i||$ represents the distance between the barycenters of the two edges.

However, this is undefined when $t_i = x$. To fix this, we will denote a weighting vector

$$(w_x)_i = \frac{1}{||x - t_i||^s} \tag{5.1}$$

whenever $x \neq t_i$ and

$$(w_x)_i = C_s \max_{j \neq k} \left(\frac{1}{||t_k - t_j||^s} \right) \tag{5.2}$$

when $x = t_i$. Note that the constant C_s is used to give more weight to the current edge than the other edges in the mesh. Choosing how to properly define this entry in the vector w_x is an interesting problem by itself, and our definition is sensitive to the choice of this value. We found in our numerical experiments that that $C_s = \frac{2s}{1-s}$ is a reasonable choice. Finally, we weight by $\frac{1}{\Gamma(1-s)}$. This yields the value of $(\mathbb{D}_0^s \alpha)(x)$.

$$\left(\mathbb{D}_0^s \alpha\right)(x) = \frac{1}{\Gamma(1-s)} w_x \cdot \mathbb{D}_0 \alpha.$$

We can generalize this definition as follows:

Definition 14 (Fractional Discrete Exterior Derivative)**.** *Let α be a discrete p-form on a connected simplicial complex M. Then the* (p, s)-fractional discrete derivative of α *at a $(p+1)$ simplex σ^{p+1} is*

$$\left(\mathbb{D}_p^s \alpha\right)(\sigma^{p+1}) = \frac{1}{\Gamma(1-s)} w_{\sigma^{p+1}} \cdot \mathbb{D}_p \alpha \tag{5.3}$$

where σ^{p+1} is a specific $p+1$ simplex, $w_{\sigma^{p+1}}$ is the weighting vector from above, replacing edges x_k and t_i with $(p+1)$-simplices as necessary, and $\mathbb{D}_p \alpha$ is the typical p^{th} order discrete exterior derivative of α.

Before we get to testing the definition on some examples, we proceed with a brief discussion on the properties of the above Definition 14. In this discussion, $|P|$ will represent the number of p-simplices in the mesh.

(1) This definition does enact the property that $\mathbb{D}_p^s$ correctly maps from p-forms to $(p+1)$-forms. To see this explicitly, see the equivalent matrix form of the definition that we give below.

(2) We know from DEC that the dot product

$$w_{\sigma^{p+1}} \cdot \mathbb{D}_p \alpha$$

will result in $\mathbb{D}_p^s \alpha = 0$ whenever α takes a constant value for all p-simplices in the mesh.

(3) Currently, this definition does *not* yield a recovery of $\mathbb{D}_p$ when we take $s \to 1$. This can be fixed, and we will show what this may look like below.

Note that we can rewrite the definition of the fractional discrete exterior derivative operator using matrices. To do this, let W be the matrix made out of using the vectors w_{x_k} as the columns. This creates a $|P| \times |P|$ matrix that we can use to say

$$\mathbb{D}_p^s \alpha = \frac{1}{\Gamma(1-s)} W \mathbb{D}_n \alpha. \tag{5.4}$$

The choice to start this way was made to make it easier to see the action at a specific simplex. However, this form makes it clear that $\mathbb{D}_p^s : \Omega_d^p \to \Omega_d^{p+1}$.

The natural way that we can force the property of recovering $\mathbb{D}_p$ in the limit of $s \to 1$ is to piecewise define w_{x_k} based off the fractional order. If $s \in (0,1)$ (or more generally, $s \in \mathbb{R}^+ \backslash \mathbb{N}$), then define w_{x_k} as in Eq. (5.1), and in the case that $s = 1$ (or $s \in \mathbb{N}$), define

$$(w_{x_k})_i = \delta_{ik}, \tag{5.5}$$

where δ_{ik} is the Kronecker delta function, taking a value of 0 if $i \neq k$ and 1 when $i = k$. One reason for doing this piecewise definition is that $\Gamma(0) = 0$. Therefore both the fractional Caputo derivative and the fractional discrete exterior derivative cannot be defined here by the usual definition or the new definition, respectively. Furthermore, in order for $\mathbb{D}_p^s$ to recover $\mathbb{D}_p$ in the limit $s \to 1$, we would need that $W \to \Gamma(1-s) I$. In the next section, we give some evidence that as $s \to 1$, our approximation gets better. This indicates that the W matrix is indeed becoming diagonally dominant so that the fractional contribution gets less as we get close to $s = 1$. Hence we believe that we do achieve property 3.

There are other properties that we chose not to enforce that may have been natural given the setting of DEC. For example, the composition property $\mathbb{D}_{p+1} \circ \mathbb{D}_p = 0$. However, our definition does not satisfy an extension to $\mathbb{D}_{p+1}^s \circ \mathbb{D}_p^s = 0$. To see this, let W_k be the corresponding weighting matrix to $\mathbb{D}_k^s$ and notice that:

$$\mathbb{D}_{p+1}^s \mathbb{D}_p^s \alpha = \frac{1}{\Gamma(1-s)} \left(W_{p+1} \mathbb{D}_{p+1}\right) \mathbb{D}_p^s \alpha \tag{5.6}$$

$$= \frac{1}{\Gamma(1-s)} W_{p+1} \mathbb{D}_{p+1} W_p \mathbb{D}_p \alpha. \tag{5.7}$$

In general, W_p and $\mathbb{D}_p$ cannot commute, and the distances that are involved in W_k will not force 0 from any specific entries.

Part of the expectation for $\mathbb{D}_{p+1} \circ \mathbb{D}_p = 0$ comes from the smooth setting, where we have $d^{p+1} \circ d^p = 0$. In other words, all exact forms in the smooth setting are closed. While a similar property would be good to have, we do not have a clear reason for expecting it to be true after the weighting process occurs. Additional formal discussion on the smooth theory of fractional exterior calculus is left out(chapter 12 of [76]).

Finally, we must address the use of a norm $||\cdot||$ in the definition. The DEC framework only assumes the existence of a local metric [46], which can be used to measure distance only

between adjacent simplices. We next discuss how to extend the local metric to be able to compute the distance between two arbitrary p-simplices.

To do this, we first need a well-defined way to find the distance between two arbitrary vertices on our mesh. We make use of the fact that between any adjacent vertices u and v, we know the distance $d(u, v)$ where d is the local metric on the mesh. Then we can apply an all pairs-shortest path algorithm to find the minimum distance between each pair of vertices on the path. This gives a distance between any pair of vertices, not just adjacent ones, which we denote $d_m(u, v)$, where the subscript m is used to differentiate the minimum distance between any pair of vertices on the mesh from the local metric d.

We extend distance between 0-simplices to build a distance between two p-simplices. Let σ^p, η^p be two simplices, $\{u_k\}$ and $\{v_k\}$ be the set of vertices attached to σ^p and η^p respectively, with $k = 0, 1, \ldots, p$. Then we define the distance

$$||\sigma^p - \eta^p||$$

computing

$$\min_{i,j} d_m(v_i, y_j) + l(\sigma^p) + l(\eta^p)$$

for all pairs of i, j with $i, j = 0, 1, \ldots, p$, where $l(\sigma^p)$ and $l(\eta^p)$ is the length between the barycenter and the boundary of each of the respective simplices.

As an aside, if we know that the embedding of the simplicial complex K uses a metric that is induced by Euclidean metric of $\mathbb{R}^N$, then it is easier to implement the weighting matrix by using the Euclidean distance between circumcenters or barycenters of the two simplices. That is, if we know that there exists a global metric that correctly embeds the complex, we can use the global metric.

5.1.2. Testing the Fractional Discrete Exterior Derivative

The next step is to test the fractional discrete exterior derivative definition. First though, we give a brief outline for how one would go about implementing the fractional discrete exterior derivative as defined above, assuming we already have a discrete p-form α.

(1) Create $\mathbb{D}_p$ matrix.

(2) Create matrix W.

(3) Apply the product of W and $\mathbb{D}_p$ to α.

(4) Multiply by the scaling factor $\dfrac{1}{\Gamma(1-s)}$.

In step 2 we will have to do a computation that finds the distance between any two given $(p+1)$-simplices before W can be created.

Now we do an example. Recall from Eq. (2.9) the form of the Caputo derivative acting on power functions

$$D^s_{[0,x]}x^q = \frac{\Gamma(q+1)x^{q-s}}{\Gamma(q+1-s)}.$$

We used the left-sided fractional Caputo derivative to find this result, and assumed that $s \in (0,1)$ with a domain of interest of $[0,1]$. However, we based our fractional discrete exterior derivative on the 2-sided fractional Caputo derivative. In the following examples, we follow the convention that $D^s_{[0,x]}$ represents a left-sided Caputo derivative, while the notation D^s represents a two-sided Caputo derivative.

Example 4. *As a specific example, consider x^3 on $[0,1]$ and $s = .5$. Then we have that*

$$D^{.5}x^3 = \frac{\Gamma(4)x^{2.5}}{\Gamma(3.5)} - \frac{3}{\Gamma(3.5)}(1-x)^{\frac{1}{2}}(.75 + x + 2x^2) \tag{5.8}$$

as the analytic solution.

As we see in this example, according to Fig. 5.1 using our definition yields a similar values. The plots of the weighted discrete 1-forms that result from the fractional discrete

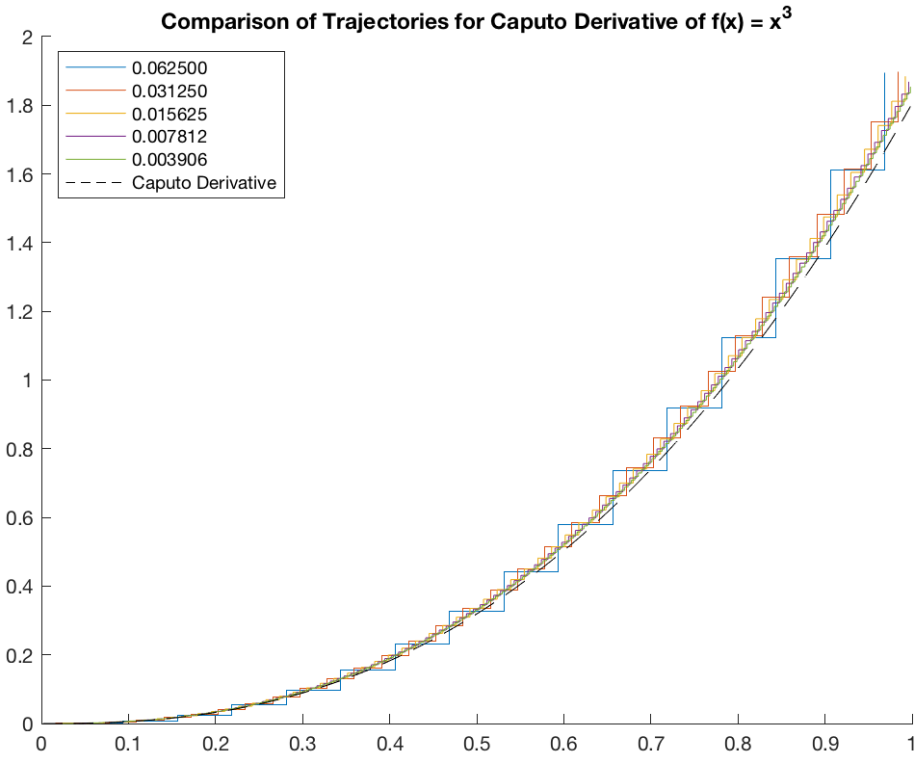


Figure 5.1. The comparison of $\mathbb{D}_0^{.5}x^3$ with $D^{.5}x^3$, for uniform meshes of various element widths. The element widths tested are given in the legend.

exterior derivative applied to x^3 on the mesh are obtained using the stairs plot in Matlab. This plotting method results in a staircase function where a given step starts at a barycenter of an edge and lasts until the next step. For each of these steps, we use the computed value in the vector $\mathbb{D}_0^s x^3$, and plot it as a constant value for the rest of the edge.

Example 5. *Let $s = 0.5$, $[a, b] = [0, 1]$, and consider the function*

$$f(x) = -10x^3 + 10x^2,$$

with a fractional Caputo derivative of

$$D^{.5}f(x) = -\frac{10\Gamma(4)}{\Gamma\left(\frac{7}{2}\right)}x^{\frac{5}{2}} + \frac{10\Gamma(3)}{\Gamma\left(\frac{5}{2}\right)}x^{\frac{3}{2}} - \frac{30}{\Gamma\left(\frac{7}{2}\right)}(1-x)^{\frac{1}{2}}\left(\frac{3}{4} + x + 2x^2\right) + \frac{20}{\Gamma\left(\frac{5}{2}\right)}(1-x)^{\frac{1}{2}}(x+\frac{1}{2}). \tag{5.9}$$

Since the function we are working with has an easily calculated closed form for its fractional Caputo derivative and a discrete 1-form can be thought of as piecewise constant on edges, we are able to symbolically integrate to get our error values. The results of the error analysis are given in the table below.

Edges	L_2 error
2	1.5619
4	0.9933
8	.6778
16	.4759
32	.3363
64	.2378
128	.1681
256	.1188
512	.0839
1024	.0593

The error results here give the indication that our error is going to 0 as the step size decreases. In Fig. 5.2, we compare the closed-form version against another piecewise linear estimate that we get as the result of applying the fractional discrete exterior derivative. In the case of different step sizes, we see that we do get closer to the true trajectory of the fractional derivative.

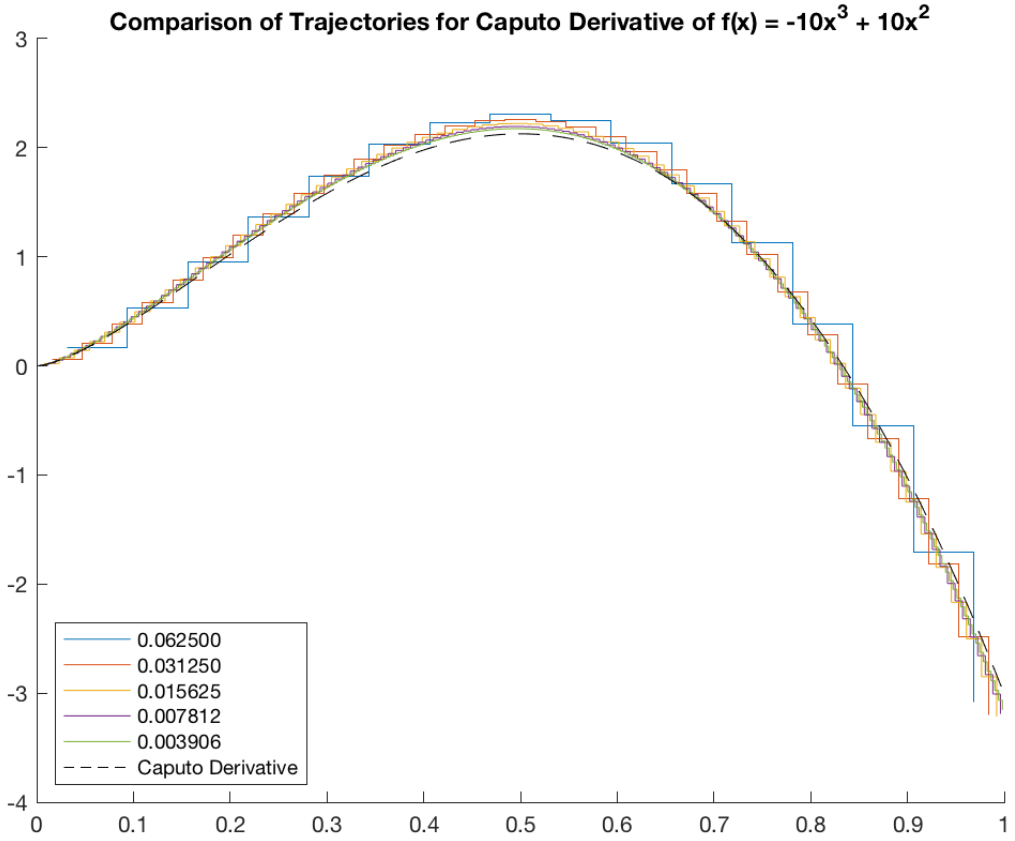


Figure 5.2. The comparison of $\mathbb{D}_0^{.5}\left(-10x^3+10x^2\right)$ with $D^{.5}\left(-10x^3+10x^2\right)$. We use varying step sizes denoted in the legend.

Since this example has a simple analytic solution to the fractional Caputo derivative we can test the effect of the value of $s \in (0,1)$.

In FDEs, it is commonplace to test the error against the fractional powers. Some applications have a clear reasoning, whether experimental or physical, for desiring a fractional power to be some specific value. Other applications treat the fractional power as free parameter. Our results in Fig. 5.3 show that while we can change the fractional power as we desire in $(0,1)$, our error can fluctuate within this range. Part of this error is expected because of the $\frac{1}{\Gamma(1-s)}$ constant based on s.

Finally, we want to check how our definition compares to the left-sided fractional Caputo derivative of e^x. To that end, we give the following definition and then a brief discussion of

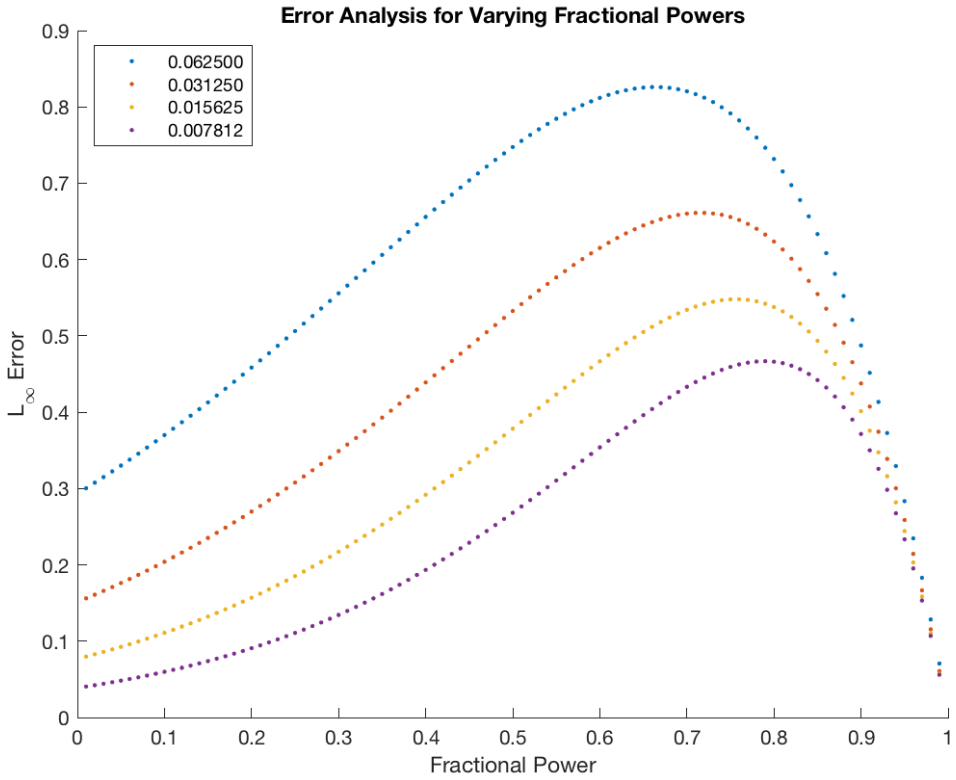


Figure 5.3. We compare the fractional exponent s against the L_∞ error for the function $f(x) = -10x^3 + 10x^2$. We use a step size varying step sizes given in the legend.

the modification of W to account for a left-sided fractional derivative instead of a 2-sided fractional derivative.

Definition 15. *The Mittag-Leffler two parameter function is*

$$E_{a,b}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(ak+b)} \tag{5.10}$$

where $a, b \in \mathbb{R}^+$ *and* $z \in \mathbb{C}$.

In general, we can think of this function as being a generalization of the exponential function. In fact, for the values of $a = 1, b = 1$, we recover the exponential function exactly. The Mittag-Leffler function can be used to give a nice looking form for the result of the fractional Caputo derivative of e^x. Explicitly, we get

$$D^s_{[0,x]} e^x = x^{1-s} E_{1,2-s}(x) \tag{5.11}$$

whenever $s \in (0,1)$ [50].

Our discretization of the fractional discrete exterior derivative is based off the two-sided fractional Caputo derivative. However, we want to be able to compare against $D^s_{[0,x]} e^x$. In the left-sided fractional Caputo derivative, we have an integration domain of $[0, x]$. To account for this, we need to modify the weighting matrix W. Specifically, each entry of W represents the weight assigned to one simplex in relation to another simplex, say σ_i and σ_j. Then, in the 1-d case, to determine if the entry should be zero to account for the left-sided version of the derivative we compare the x-coordinates of the barycenters of σ_i and σ_j. If $x_i < x_j$, we leave $W_{i,j}$ as it is and otherwise we set $W_{i,j} = 0$. The results of this comparison are given in Fig. 5.4.

In Fig. 5.4, we see the difference in trajectories for the Caputo derivative of e^x. The code used to generate the Mittag-Leffler function is from [38]. Our definition undershoots the value of the Caputo derivative, but does seem to exhibit the correct behavior. To do an accurate error analysis in this case, we choose the L_∞ norm. We see the results of this analysis in Fig. 5.5.

The following examples are efforts in testing the fractional discrete exterior derivative in 2-d. We start with a basic function

Example 6. *Let $f(x, y) = -x^2 + y^2$. Then the 2-sided fractional gradient field on the region $M = [0, 1] \times [0, 1]$ of order $s \in (0, 1)$ is*

$$\left(\frac{-2x^{\frac{3}{2}}}{\Gamma\left(\frac{5}{2}\right)} - \frac{2(1-x)^{\frac{1}{2}}(x+\frac{1}{2})}{\Gamma\left(\frac{5}{2}\right)}, \frac{2y^{\frac{3}{2}}}{\Gamma\left(\frac{5}{2}\right)} + \frac{2(1-y)^{\frac{1}{2}}(y+\frac{1}{2})}{\Gamma\left(\frac{5}{2}\right)} \right). \tag{5.12}$$

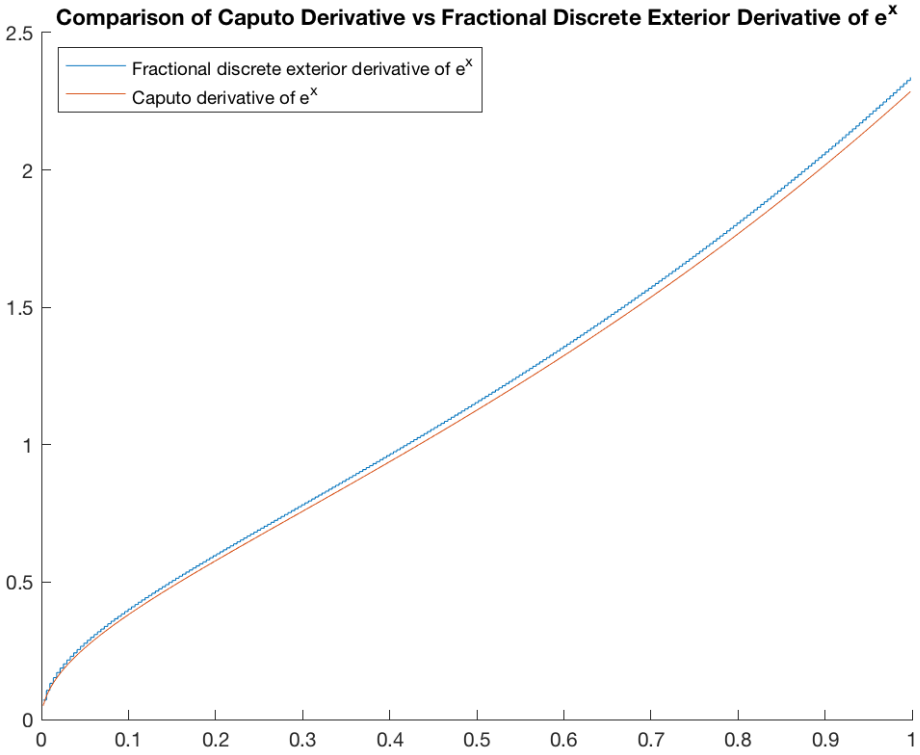


Figure 5.4. Comparison of the Caputo derivative of e^x versus $\mathbb{D}_0^{.5}e^x$.

Using this, we consider a triangulated plot of M with the gradient vector field drawn at the barycenters of the triangles. This field is plotted in Fig. 5.6.

To test the fractional discrete exterior derivative on this example and visualize a vector field requires some extra work. We first create the 0-form vector α that represents the value of f at the vertices of the mesh. After applying the fractional discrete exterior derivative, we get a discrete 1-form $\mathbb{D}^{.5}\alpha$, i.e. a cochain storing a value for each edge of the mesh with edge orientation implied by the global vertex ordering. Visualizing and measuring the error of this data type requires some care.

We first construct the Whitney map for 1-cochains, based on [46, Definition 3.3.4]. The construction is as follows: compute the barycentric coordinates λ_1, λ_2, λ_3 locally on each triangle, use these to compute the three Whitney 1-forms $(\lambda_i \nabla \lambda_j - \lambda_j \nabla \lambda_i)$ associated to

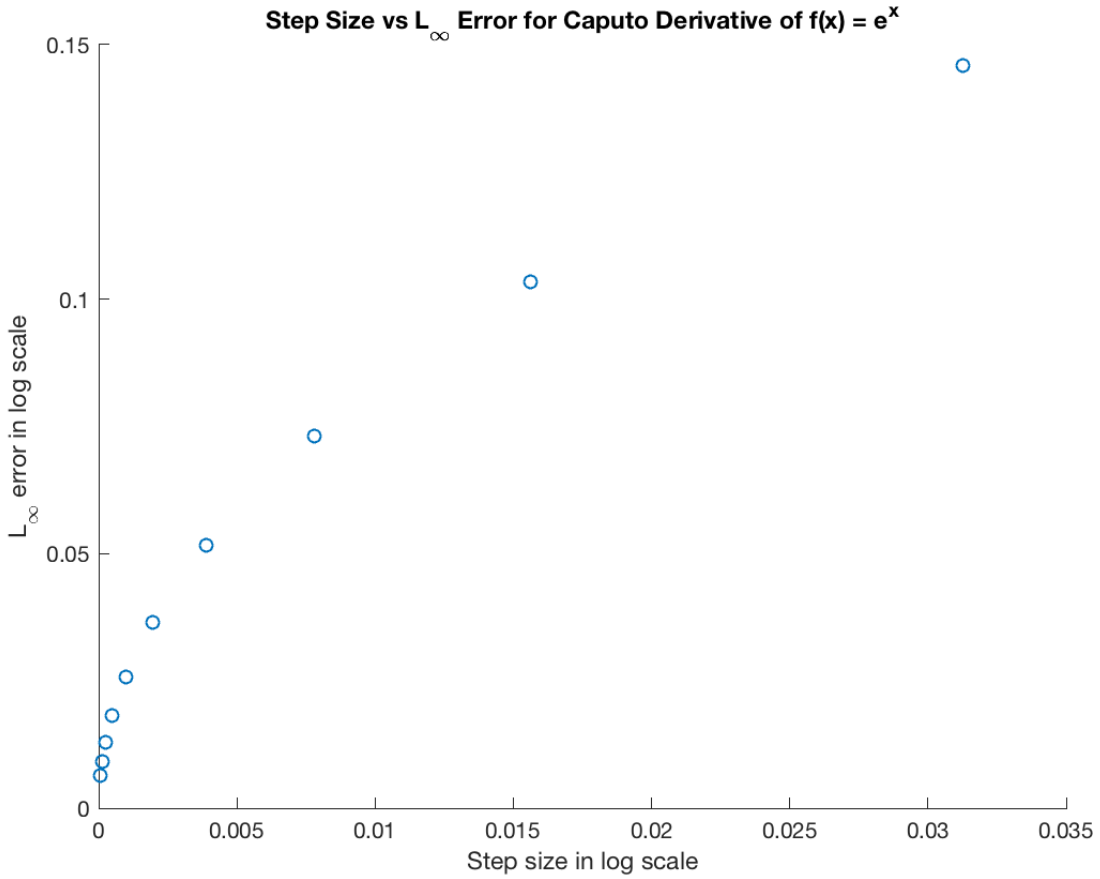


Figure 5.5. Error analysis for $\mathbb{D}_0^s e^x$. We use the L_∞ norm to find max errors at varying step sizes in comparison to $x^{\cdot 5} E_{1,1.5}(x)$, the closed form of the left-sided Caputo derivative of e^x.

each edge, then weight each Whitney 1-form by the cochain value on the corresponding edge. The result is a vector field, defined piecewise over the entire mesh, that recovers the cochain values on each edge (when projected onto an edge). We evaluate this global vector field at the barycenter of each triangle for both qualitative and quantitative error analysis.

To give a sense of what the fractional Caputo gradient field on M looks like, see Fig. 5.6. In Fig. 5.7, we look at relative error. Notice that originally, the function has a saddle type critical point (i.e. the gradient of the function is zero) at the origin. This critical point, along with some boundary effects from our definition, lead to a high relative error near the origin.

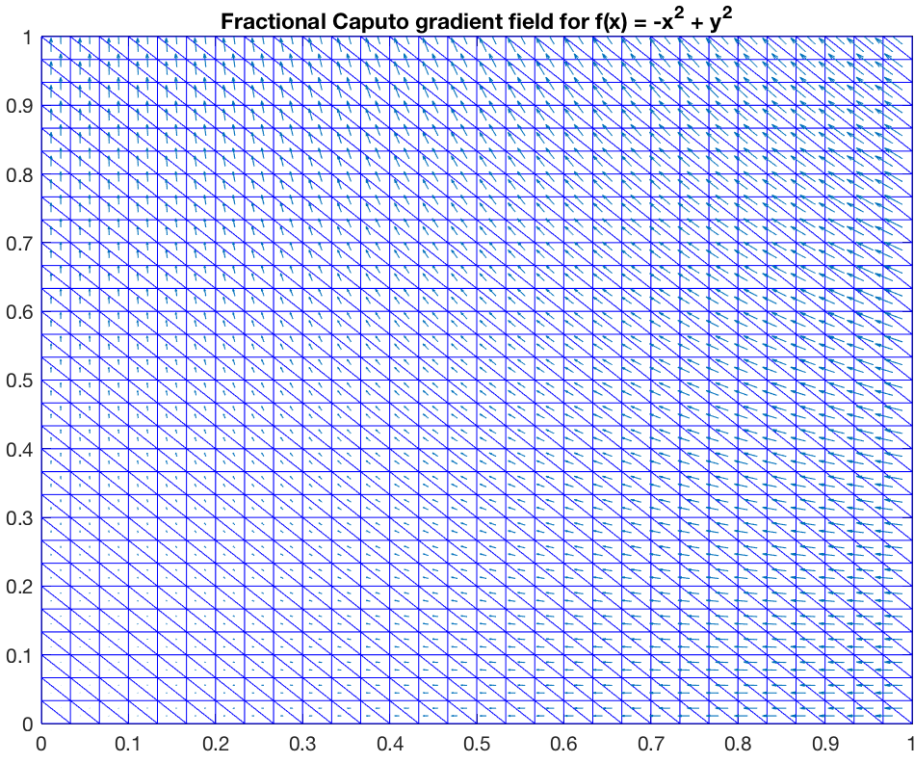


Figure 5.6. The 2-sided fractional Caputo gradient field for the function $-x^2 + y^2$.

Example 7. *We do one last example. Let*

$$f(x, y) = (x - .1)^2 + (y - .1)^2 \tag{5.13}$$

be defined on $M = [0, 1] \times [0, 1]$. *In this case, we know that* $f(x, y)$ *has a minimum at* $(.1, .1)$. *The 2-sided fractional Caputo gradient field of* f *is*

$$D^{\frac{1}{2}} f = \left(\frac{2x^{\frac{3}{2}} - (1-x)^{\frac{1}{2}}(\frac{1}{2} - 2x)}{\Gamma\left(\frac{5}{2}\right)} - \frac{x^{\frac{1}{2}} - (1-x)^{\frac{1}{2}}}{5\Gamma\left(\frac{3}{2}\right)}, \right. \tag{5.14}$$
$$\left. \frac{2y^{\frac{3}{2}} - (1-y)^{\frac{1}{2}}(\frac{1}{2} - 2y)}{\Gamma\left(\frac{5}{2}\right)} - \frac{y^{\frac{1}{2}} - (1-y)^{\frac{1}{2}}}{5\Gamma\left(\frac{3}{2}\right)} \right)$$

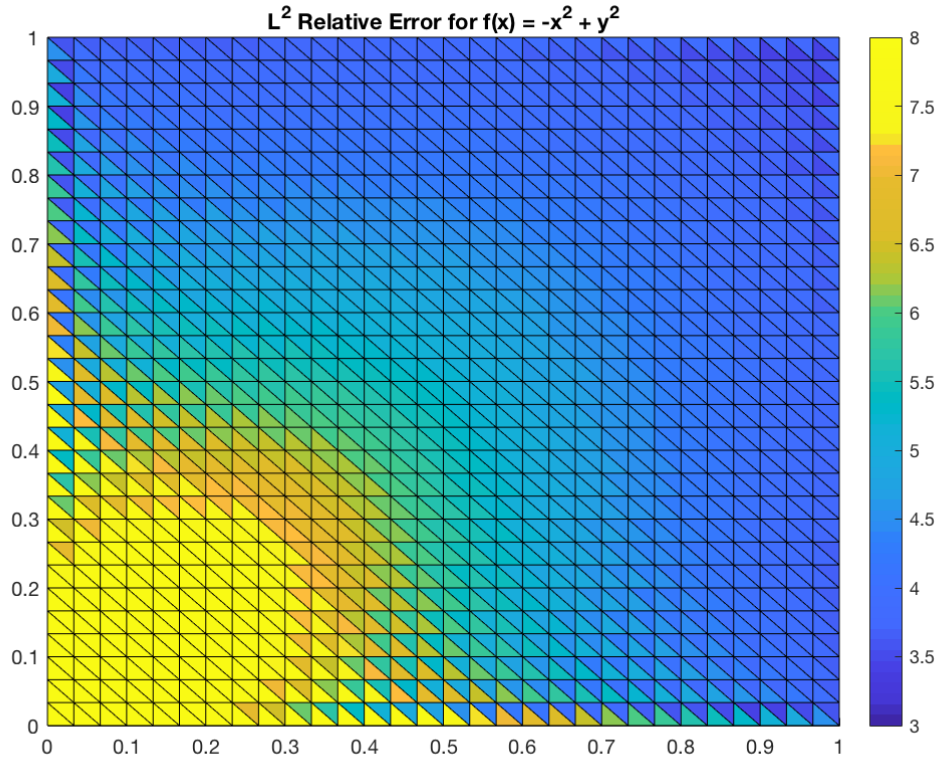


Figure 5.7. The triangles are colored to represent the relative L^2 error for $f(x, y) = -x^2 + y^2$. We get error values ranging between .0627 and 4.4227, with an average of 1.2126.

This fractional gradient field is plotted in Fig. 5.8. We also give a plot for relative error for f in Fig. 5.9. In this case, we again have a critical point that affects our results. For this function, the critical point exists in the domain M. One point of interest is that the fractional Caputo derivative does not preserve the location of critical points, which is what causes a large amount of the error in Fig. 5.9. In either setting, understanding the behavior of the fractional discrete exterior derivative on functions of multiple variables with critical points is a topic that we intend to investigate further in the future.

5.2. Discussion

In this section we gave a new definition of the fractional discrete exterior derivative that satisfied three key properties. Our definition was based on the fractional Caputo derivative,

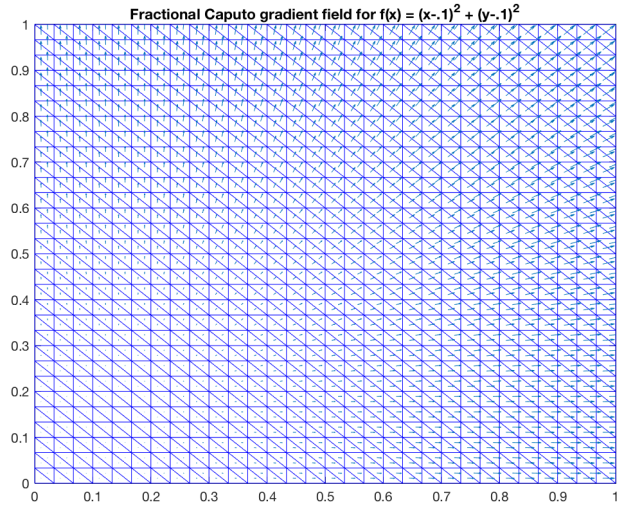


Figure 5.8. The 2-sided fractional Caputo gradient field for the function $f(x, y) = (x - .1)^2 + (y - .1)^2$.

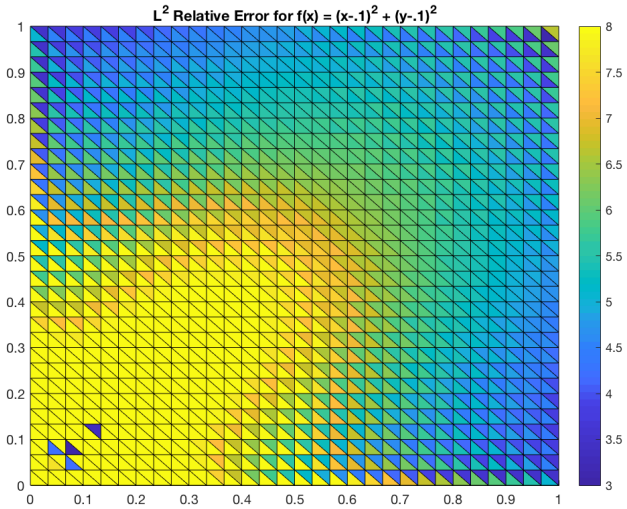


Figure 5.9. The triangles are colored to represent the relative L^2 error for $f(x) = (x - .1)^2 + (y - .1)^2$. Errors here range from .2369 to 2.7504, with an average error of 1.3561.

and we tested on some 1- and 2-d examples. Our definition requires an interpretation of distance between two arbitrary p-simplices on the mesh.

There are some limitations in the fractional discrete exterior derivative definition. First, it complicates the process of taking a discrete exterior derivative, which is normally represented

by a single matrix with a nice structure. While this does mean that the process of taking a fractional discrete exterior derivative is more involved, this is to be expected in comparison to a normal discrete exterior derivative. Specifically, the non-local nature of the fractional derivative means that we have to dedicate time in the computation to determining distances between simplices that are far apart. The other limitation is that our definition does not act on critical points the same way that the fractional Caputo derivative does. This means that near critical points, we end up with a much higher error term than we would like.

For future work, we have many avenues to investigate. First, we would like to study the way that the fractional discrete exterior derivative responds to tweaking the diagonal of the matrix W. Furthermore, we would like to be able to understand the behavior better at the critical points of a function.

There is also a connection that we have not yet discussed. In the past couple of decades, there has been work on defining an extension of exterior calculus to a fractional exterior calculus setting, e.g. [76, 25]. This includes a definition of a fractional exterior derivative, with the form

$$d^s = \sum_{i=1}^{n} \frac{\partial^s}{\partial x^{i^s}} [dx^i]^s.$$

We intentionally chose not to make use of definition this when defining our operator. Directly discretizing this fractional exterior derivative would not fit in the "discrete first" approach that DEC takes, and discretizing it in a way analogous to how the discrete exterior derivative is discretized would require a notion of a fractional boundary. One possible strength of discretizing with this formula is the possibility of ensuring $d^s \circ d^s = 0$, as is shown in [25]. Still, fractional exterior calculus as a whole is a relatively new field, and it is not clear which properties are essential to preserve when considering their discretization.

Beyond just investigating more of the properties of the definition, we want to extend into a more DEC related setting. The results on an interval and a rectangular region subdivided

into triangles are promising, however one strength of DEC is its ability to work on manifold-like simplicial complexes. With a definition of the fractional discrete exterior derivative, we should be able to extend current work using DEC to FDEs on simplicial complexes that represent more intricate geometric objects.

In the vein of solving FDEs, one of the prominent operators in FDEs is the fractional Laplacian, $(-\Delta)^s$. The Laplacian operator has a nice definition in DEC where $\Delta = *d*d$. Hence studying the fractional Laplacian in the DEC setting may reduce to studying $\star\mathbb{D}^s_{p-1}\star\mathbb{D}^s_0$, with $\star$ being some sort of fractional discrete Hodge star operator. A good overview of the work that has been done on the fractional Laplacian was written by Lischke et. al in [58].

As discussed in the introduction, there was also a shape analysis application by Paquet and Viktor in [68] using the fractional de Rham operator. In their approach, they take a fractional power by using an eigenvalue decomposition, and raising the eigenvalues to the fractional power. In the fractional Laplacian literature, this is referred to as the spectral method for computing the fractional power of the operator. This would be interesting to test against when we are ready to use our method to try discretizing the fractional Laplacian. In general, we cannot use the spectral method for the fractional discrete exterior derivative because it is a non-square matrix.

We would also be interested in exploring what the other definitions of fractional derivatives yield when discretized through DEC. For example, the Riemann-Liouville fractional derivative is given by

$$ {}^{RL}D^s_{[a,x]}f(x) = \frac{1}{\Gamma(n-s)}\frac{d^n}{dx^n}\int_a^x \frac{f(t)}{(x-t)^{n+s-1}}dt, \tag{5.15}$$

where $[a,b]$ is some domain of interest, $s \in \mathbb{R}^+$ and $n = \lceil s \rceil$.

The main difference between this definition and the Caputo definition is the order of the differentiation and integration. Because the integration occurs before taking any derivatives, the class of functions we act on is slightly less restrictive. We suspect that the Riemann-Liouville definition leads to a fractional discrete exterior derivative operator that has the form

$$\frac{1}{\Gamma(1-s)}\mathbb{D}_p W\alpha \tag{5.16}$$

where W is a weighting matrix similar to before, but instead of acting on discrete $(p+1)$-forms, it acts on discrete p-forms.

Having a working version of the fractional discrete exterior derivative derived from the Riemann-Liouville derivative will be useful. Many applications of FDEs require either the Caputo derivative or the Riemann-Liouville derivative, and they in general do not yield the same results. Enabling the computation of both forms will allow for more tests that can be used to compare and evaluate fractional derivatives.

We are also interested in using these operators to study fractional equivalents to problems that have been previously investigated with DEC. For example, recent work has been done on spatial Darcy flow problems to numerically compare results against experimental data.

CHAPTER 6

Conclusion

The work here is a collection of results testing and extending aspects of discretization techniques of finite element exterior calculus and discrete exterior calculus. Within the domain of FEEC, we provided an implementation of the trimmed serendipity elements in Firedrake. Using this implementation, we then gave an analysis of how the trimmed serendipity elements compare to the tensor product elements for a few different small PDE toy problems. Finally, we applied these elements to the monodomain equation. In terms of DEC, we defined and tested a new definition for how we would calculate a fractional discrete exterior derivative.

6.1. Future Directions

Our research with trimmed serendipity elements has some natural directions to progress in the future. Currently, the analysis we have given of the trimmed serendipity elements is solely on square and cubical meshes. Applying these elements to meshes with transformed squares or cubes and comparing their capabilities to tensor product elements would further illustrate their usefulness in practical settings.

Another avenue of research with the trimmed serendipity elements would be investigating the structure of the basis functions. In the tensor product case, there are ways to speed up computation time when approximating solutions to PDEs, such as static condensation. Unfortunately, the current method for implementing the trimmed serendipity elements does not allow for this sort of computational trick. Devising methods for how to do this with trimmed serendipity elements would continue to demonstrate their viability as a family of finite element methods that are useful to practitioners.

Another interesting application area for the trimmed serendipity elements would be using them to approximate solutions to FDEs. Since FDEs require non-local information, the overall reduction in degrees of freedom between the tensor product elements and the trimmed serendipity elements may have an increased significance. In recent years, many works have been focused on using finite element methods to solve various FDEs [1, 32, 82]. Typical finite elements become computationally difficult and work has to be done to address the large number of DOFs involved [72].

Solving FDEs is also relevant to continuing the work on extending DEC. The fractional discrete exterior derivative gives a starting point for solving FDEs in the setting of DEC. However, the current testing of the definition here does not get into actually implementing the fractional discrete exterior derivative into a DEC software package. Furthermore, converting FDEs to the notation of DEC may require fractional analogues of other common operators which we have not yet defined. Doing this would allow for further testing of how well the fractional discrete exterior derivative works in practice.

There are many other ways that we could have chosen to define the fractional discrete exterior derivative. Among them include discretizing the Riemann-Liouville fractional derivative or any number of other fractional derivative definitions. The literature for fractional derivatives includes many definitions, and it is possible that other fractional derivative definitions would also lead to viable fractional discrete exterior derivative operators.

Finally, taking a parallel route to DEC is also a future consideration. The traditional discrete exterior calculus was created by taking the operators from exterior calculus and discretizing them in a natural way. Using the theory of fractional exterior calculus [76], one may be able to achieve a fully fractional discrete exterior calculus, independent of the original setting of DEC [46].

CHAPTER 7

Appendix A: Ten Tusscher Model Equations

First, we give the various currents [78] that remain unchanged in the later rendition of the model. The first current to be described is the fast Na^+ current. This uses the following equations:

$$E_{\mathrm{Na}} = \frac{RT}{zF}\log\frac{\mathrm{Na}_o}{\mathrm{Na}_i}$$

$$I_{\mathrm{Na}} = G_{\mathrm{Na}} m^3 hj(V - E_{\mathrm{Na}})$$

$$m_\infty = \frac{1}{[1 + e^{(-56.86-V)/9.03}]^2}$$

$$\alpha_m = \frac{1}{1 + e^{(-60-V)/5}}$$

$$\beta_m = \frac{0.1}{1 + e^{(V+35)/5}} + \frac{0.1}{1 + e^{(V-50)/200}}$$

$$\tau_m = \alpha_m \beta_m$$

$$h_\infty = \frac{1}{[1 + e^{(V+71.55)/7.43}]^2}$$

$$\alpha_h = 0 \text{ if } V \geq -40$$

$$\alpha_h = 0.057e^{-(V+80)/6.8} \text{ otherwise}$$

$$\beta_h = \frac{0.77}{0.13[1 + e^{-(V+10.66)/11.1}]} \text{ if } V \geq -40$$

$$\beta_h = 2.7e^{0.079V} + 3.1 \times 10^5 e^{0.3485V} \text{ otherwise}$$

$$\tau_h = \frac{1}{\alpha_h + \beta_h}$$

$$j_\infty = \frac{1}{[1 + e^{(V+71.55)/7.43}]^2}$$

$$\alpha_j = 0 \text{ if } V \geq -40$$

$$\alpha_j = \frac{(-2.5428 \times 10^4 e^{0.2444V} - 6.948 \times 10^6 e^{-0.04391V})(V + 37.78)}{1 + e^{0.311(V+79.23)}} \text{ otherwise}$$

$$\beta_j = \frac{0.6e^{0.057V}}{1 + e^{-0.1(V+32)}} \text{ if } V \geq -40$$

$$\beta_j = \frac{0.02424e^{-0.01052V}}{1 + e^{-0.1378(V+40.14)}} \text{ otherwise}$$

$$\tau_j = \frac{1}{\alpha_j + \beta_j}$$

Next we give the equations for the current I_{to}, the transient outward current.

$$I_{\text{to}} = G_{\text{to}} rs(V - E_k)$$

$$r_\infty = \frac{1}{1 + e^{(20-V)/6}}$$

$$s_\infty = \frac{1}{1 + e^{(V+20)/5}}$$

$$\tau_r = 9.5e^{-(V+40)^2/1800} + 0.8$$

$$\tau_s = 85e^{-(V+45)^2/320} + \frac{5}{1 + e^{(V-20)/5}} + 3$$

After that, we have the rapid delayed rectifier current.

$$I_{\text{Kr}} = G_{\text{Kr}} \sqrt{\frac{K_O}{5.4}} x_{r1} x_{r2} (V - E_k)$$

$$x_{r1\infty} = \frac{1}{1 + e^{(-26-V)/7)}}$$

$$\alpha_{xr1} = \frac{450}{1 + e^{(-45-V)/10}}$$

$$\beta_{xr1} = \frac{6}{1 + e^{(V+30)/11.5}}$$

$$\tau_{xr1} = \alpha_{xr1}\beta_{xr1}$$

$$x_{r2\infty} = \frac{1}{1+e^{(V+88)/24}}$$

$$\alpha_{xr2} = \frac{3}{1+e^{(-60-V)/20}}$$

$$\beta_{xr2} = \frac{1.12}{1+e^{(V-60)/20}}$$

$$\tau_{xr2} = \alpha_{xr2}\beta_{xr2}$$

Then there is the inward rectifier current K^+.

$$I_{K1} = G_{\mathrm{K1}}\sqrt{\frac{K_o}{5.4}}x_{K1\infty}(V - E_K)$$

$$\alpha_{K1} = \frac{0.1}{1+e^{0.06(V-E_K-200)}}$$

$$\beta_{K1} = \frac{3e^{0.0002(V-E_K+100)} + e^{0.1(V-E_K-10)}}{1+e^{-0.5(V-E_K)}}$$

$$x_{K1\infty} = \frac{\alpha_{K1}}{\alpha_{K1}+\beta_{K1}}$$

Following that, the Na^+/Ca^{2+} exchanger current is given.

$$I_{\mathrm{NaCa}} = \frac{e^{\gamma VF/RT}\mathrm{Na}_i^3\mathrm{Ca}_o - e^{(\gamma-1)VF/RT}\mathrm{Na}_o^3\mathrm{Ca}_i\alpha}{(K_{\mathrm{mNai}}^3 + \mathrm{Na}_o^3)(K_{\mathrm{mCa}} + \mathrm{Ca}_o)(1 + k_{\mathrm{sat}}e^{(\gamma-1)VF/RT})}$$

The Na^+/Ca^{2+} pump current.

$$I_{\mathrm{NaK}} = \frac{P_{\mathrm{NaK}}\mathrm{K}_o\mathrm{Na}_i}{(K_o + K_{mK})(\mathrm{Na}_i + K_{\mathrm{mNa}})(1 + 0.1245e^{-0.1VF/RT} + 0.0353e^{-VF/RT})}$$

The current I_{pCa}.

$$I_{\mathrm{pCa}} = G_{\mathrm{pCa}} \frac{\mathrm{Ca}_i}{K_{\mathrm{pCa}} + \mathrm{Ca}_i}$$

The current I_{pK}.

$$I_{\mathrm{pK}} = G_{\mathrm{pK}} \frac{V - E_{\mathrm{K}}}{1 + e^{(25-V)/5.98}}$$

The background currents I_{bNa} and I_{bCa}. Note that E_{Na} was given earlier.

$$I_{\mathrm{bNa}} = G_{\mathrm{bNa}}(V - E_{\mathrm{Na}})$$

$$I_{\mathrm{bCa}} = G_{\mathrm{bCa}}(V - E_{\mathrm{Ca}})$$

$$E_{\mathrm{Ca}} = \frac{RT}{zF} \log \frac{\mathrm{Ca}_o}{\mathrm{Ca}_i}$$

The next two currents, I_{Ks} and I_{CaL} are currents that were introduced in 2004 and then modified in 2006.

The L-Type Ca^{2+} current.

$$I_{\mathrm{CaL}} = 4 G_{\mathrm{CaL}} d f f_2 f_{\mathrm{cass}} \frac{(V-15)F^2}{RT} \frac{0.25 \mathrm{Ca}_{\mathrm{SS}} e^{2(V-15)F/RT} - \mathrm{Ca}_{\mathrm{O}}}{e^{2(V-15)F/RT} - 1}$$

$$d_\infty = \frac{1}{1 + e^{(-8-V)/7.5}}$$

$$\alpha_d = \frac{1.4}{1 + e^{(-35-V)/13}} + 0.25$$

$$\beta_d = \frac{1.4}{1 + e^{(V+5)/5}}$$

$$\gamma_d = \frac{1}{1 + e^{(50-V)/20}}$$

$$\tau_d = \alpha_d * \beta_d + \gamma_d$$

$$f_\infty = \frac{1}{1 + e^{(V+20)/7}}$$

$$\alpha_f = 1102.5 e^{-\left(\frac{V+27}{15}\right)^2}$$

$$\beta_f = \frac{200}{1 + e^{(13-V)/10}}$$

$$\gamma_f = \frac{180}{1 + e^{(V+30)/10}} + 20$$

$$\tau_f = \alpha_f + \beta_f + \gamma_f$$

$$f_{2\infty} = \frac{0.67}{1 + e^{(V+35)/7}} + 0.33$$

$$\alpha_{f2} = 600 e^{-\frac{(V+25)^2}{170}}$$

$$\beta_{f2} = \frac{31}{1 + e^{(25-V)/10}}$$

$$\gamma_{f2} = \frac{16}{1 + e^{(V+30)/10}}$$

$$\tau_{f2} = \alpha_{f2} + \beta_{f2} + \gamma_{f2}$$

$$f_{\text{cass}\infty} = \frac{0.6}{1 + \left(\frac{\text{Ca}_{\text{SS}}}{0.05}\right)^2} + 0.4$$

$$\tau_{f\text{cass}} = \frac{80}{1 + \left(\frac{\text{Ca}_{\text{SS}}}{0.05}\right)^2} + 2$$

Next is the slow delayed rectifier current I_{Ks}.

$$I_{\text{Ks}} = G_{\text{Ks}} x_s^2 (V - E_{Ks})$$

$$x_{s\infty} = \frac{1}{1 + e^{(-5-V)/14}}$$

$$\alpha_{xs} = \frac{1400}{\sqrt{1 + e^{(5-V)/6}}}$$

$$\beta_{xs} = \frac{1}{1 + e^{(V-35)/15}}$$

$$\tau_{xs} = \alpha_{xs}\beta_{xs} + 80$$

Finally, we give the equations that dictate the calcium dynamics, introduced in the second work [77].

$$I_{\text{leak}} = V_{\text{leak}}(\text{Ca}_{\text{SR}} - \text{Ca}_{\text{i}})$$

$$I_{\text{up}} = \frac{V_{\text{maxup}}}{1 + K_{\text{up}}^2/\text{Ca}_{\text{i}}^2}$$

$$I_{\text{rel}} = V_{\text{rel}}O(\text{Ca}_{\text{SR}} - \text{Ca}_{\text{SS}})$$

$$I_{\text{xfer}} = V_{\text{xfer}}(\text{Ca}_{\text{SS}} - \text{Ca}_{\text{i}})$$

$$O = \frac{k_1\text{Ca}_{\text{SS}}^2\overline{R}}{k_3 + k_1\text{Ca}_{\text{SS}}^2}$$

$$k_1 = \frac{k_{1'}}{k_{\text{casr}}}$$

$$k_2 = k_{2'}k_{\text{casr}}$$

$$k_{\text{casr}} = \text{max}_{\text{sr}} - \frac{\text{max}_{\text{sr}} - \text{min}_{\text{sr}}}{1 + (\text{EC}/\text{Ca}_{\text{SR}})^2}$$

$$\text{Ca}_{\text{ibufc}} = \frac{\text{Ca}_{\text{i}} \times \text{Buf}_{\text{c}}}{\text{Ca}_{\text{i}} + K_{\text{bufc}}}$$

$$\text{Ca}_{\text{srbufsr}} = \frac{\text{Ca}_{\text{sr}} \times \text{Buf}_{\text{sr}}}{\text{Ca}_{\text{sr}} + K_{\text{bufsr}}}$$

$$\text{Ca}_{\text{ssbufss}} = \frac{\text{Ca}_{\text{ss}} \times \text{Buf}_{\text{ss}}}{\text{Ca}_{\text{ss}} + K_{\text{bufss}}}$$

Finally, with all the above equations given, we describe the system of differential equations that need to be solved in order to determine the function I_{ion}.

$$\begin{aligned}
\frac{d\overline{R}}{dt} &= -k_2 \mathrm{Ca}_{\mathrm{SS}} \overline{R} + k_4(1-\overline{R}) \\
\frac{d\mathrm{Ca}_{\mathrm{itotal}}}{dt} &= -\frac{I_{\mathrm{bCa}} + I_{\mathrm{pCa}} - 2I_{\mathrm{NaCa}}}{2V_{\mathrm{c}}F} + \frac{V_{\mathrm{sr}}}{V_{\mathrm{c}}}(I_{\mathrm{leak}} - I_{\mathrm{up}}) + I_{\mathrm{xfer}} \\
\frac{d\mathrm{Ca}_{\mathrm{SRtotal}}}{dt} &= I_{\mathrm{up}} - I_{\mathrm{leak}} - I_{\mathrm{rel}} \\
\frac{d\mathrm{Ca}_{\mathrm{SStotal}}}{dt} &= -\frac{I_{\mathrm{CaL}}}{2V_{\mathrm{SS}}F} + \frac{V_{\mathrm{SR}}}{V_{\mathrm{SS}}} I_{\mathrm{rel}} - \frac{V_{\mathrm{c}}}{V_{\mathrm{SS}}} I_{\mathrm{xfer}} \\
\frac{d\mathrm{Na}_{\mathrm{i}}}{dt} &= -\frac{I_{\mathrm{Na}} + I_{\mathrm{bNa}} + 3I_{\mathrm{NaK}} + 3I_{\mathrm{NaCa}}}{V_{\mathrm{C}}F} \\
\frac{d\mathrm{K}_{\mathrm{i}}}{dt} &= -\frac{I_{\mathrm{K1}} + I_{\mathrm{to}} + I_{\mathrm{Kr}} + I_{\mathrm{Ks}} - 2I_{\mathrm{NaK}} + I_{\mathrm{pK}} + I_{\mathrm{stim}} - I_{\mathrm{ax}}}{V_{\mathrm{C}}F}
\end{aligned}$$

Then for the gating variables $d, f, f_2, f_{\mathrm{cass}}, x_s, m, h, j, r, s, x_{r1}, x_{r2}$ we get a Hodgkin-Huxley type ODE (described above and repeated here using an arbitrary gate variable g

$$\frac{dg}{dt} = \alpha(V)(1-g) - \beta(V)g.$$

Note that the current I_{stim} is the externally applied stimulus, and the current I_{ax} is the axial current flow.

References

[1] Om P Agrawal. A general finite element formulation for fractional variational problems. *Journal of Mathematical Analysis and Applications*, 337(1):1–12, 2008.

[2] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[3] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified Form Language: a domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, 40(2):1–37, 2014. doi: 10.1145/2566630.

[4] Patrick R. Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stéphane Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2): 136–156, 2006. doi: 10.1016/j.parco.2005.07.004.

[5] Patrrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001. doi: 10.1137/S0895479899358194.

[6] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021.

[7] Todd Arbogast and Maicon R. Correa. Two families of $H(\mathrm{div})$ mixed finite elements on quadrilaterals of minimal dimension. *SIAM Journal on Numerical Analysis*, 54(6): 3332–3356, 2016. doi: 10.1137/15M1013705.

[8] Todd Arbogast and Zhen Tao. Direct serendipity and mixed finite elements on convex quadrilaterals. *arXiv preprint arXiv:1809.02192*, 2018.

[9] Douglas N Arnold. Differential complexes and numerical stability. *arXiv preprint math/0212391*, 2002.

[10] Douglas N. Arnold and Gerard Awanou. The serendipity family of finite elements. *Foundations of Computational Mathematics*, 11(3):337–344, 2011. doi: 10.1007/s10208-011-9087-3.

[11] Douglas N. Arnold and Gerard Awanou. Finite element differential forms on cubical meshes. *Mathematics of Computation*, 83(288):1551–1570, 2014. doi: 10.1090/S0025-5718-2013-02783-4.

[12] Douglas N. Arnold and Anders Logg. Periodic table of the finite elements. *SIAM News*, 47(9):212, 2014. URL `https://sinews.siam.org/Details-Page/periodic-table-of-the-finite-elements`.

[13] Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$. *Numerische Mathematik*, 85:197–217, 2000. doi: 10.1007/PL00005386.

[14] Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. Finite element exterior calculus, homological techniques, and applications. *Acta Numerica*, 15:1–155, 2006. doi: 10.1017/S0962492906210018.

[15] Douglas N Arnold, Richard S Falk, and Ragnar Winther. Finite element exterior calculus, homological techniques, and applications. *Acta numerica*, 15:1–155, 2006.

[16] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific*

Computing, pages 163–202, Boston, MA, 1997. Birkhäuser Press. ISBN 978-1-4612-1986-6. doi: 10.1007/978-1-4612-1986-6_8.

[17] 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. PETSc users manual. Technical Report ANL-95/11 - Revision 3.15, Argonne National Laboratory, 2021. URL `https://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf`.

[18] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. deal. ii—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):24–es, 2007.

[19] Nathan Bell and Anil N Hirani. Pydec: software and algorithms for discretization of exterior calculus. *ACM Transactions on Mathematical Software (TOMS)*, 39(1):1–41, 2012.

[20] Daniele Boffi. Finite element approximation of eigenvalue problems. *Acta Numerica*, 19:1–120, 2010. doi: 10.1017/S0962492910000012.

[21] Kevin Burrage, Nicholas Hale, and David Kay. An efficient implicit fem scheme for fractional-in-space reaction-diffusion equations. *SIAM Journal on Scientific Computing*, 34(4):A2145–A2172, 2012.

[22] Michele Caputo. Linear models of dissipation whose Q is almost frequency independent-II. *Geophysical Journal International*, 13(5):529–539, 1967.

[23] Snorre H. Christiansen and Andrew Gillette. Constructions of some minimal finite element systems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 50(3):833–850, 2016. doi: 10.1051/m2an/2015089.

[24] Bernardo Cockburn and Guosheng Fu. A systematic construction of finite element commuting exact sequences. *SIAM Journal on Numerical Analysis*, 55(4):1650–1688, 2017.

[25] Kathleen Cottrill-Shepherd and Mark Naber. Fractional differential forms. *Journal of Mathematical Physics*, 42(5):2203–2212, 2001. ISSN 00222488. doi: 10.1063/1.1364688.

[26] Justin Crum, Joshua A Levine, and Andrew Gillette. Extending discrete exterior calculus to a fractional derivative. *Computer-Aided Design*, 114:64–72, 2019.

[27] Justin Crum, Cyrus Cheng, David A Ham, Lawrence Mitchell, Robert C Kirby, Joshua A Levine, and Andrew Gillette. Bringing trimmed serendipity methods to computational practice in firedrake. *arXiv preprint arXiv:2104.12986*, 2021.

[28] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011. doi: 10.1016/j.advwatres.2011.04.013. New Computational Methods and Software Tools.

[29] Marta D'Elia, Mamikon Gulian, Hayley Olson, and George Em Karniadakis. A unified theory of fractional, nonlocal, and weighted nonlocal vector calculus. *arXiv preprint arXiv:2005.07686*, 2020.

[30] Mathieu Desbrun, Anil N Hirani, Melvin Leok, and Jerrold E Marsden. Discrete exterior calculus. *arXiv preprint math/0508341*, 2005.

[31] Ayelet Dominitz and Allen Tannenbaum. Texture mapping via optimal mass transport. *IEEE transactions on visualization and computer graphics*, 16(3):419–433, 2010.

[32] Beiping Duan, Bangti Jin, Raytcho Lazarov, Joseph Pasciak, and Zhi Zhou. Space-time petrov–galerkin fem for fractional diffusion problems. *Computational Methods in Applied Mathematics*, 18(1):1–20, 2018.

[33] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics (TOG)*, 26(1):

4, 2007.

[34] Megan E. Farquhar, Timothy J. Moroney, Qianqian Yang, Ian W. Turner, and Kevin Burrage. Computational modelling of cardiac ischaemia using a variable-order fractional Laplacian. pages 1–21, 2018. URL `http://arxiv.org/abs/1809.07936`. arXiv:1809.07936.

[35] Patrick E Farrell, Robert C Kirby, and Jorge Marchena-Menendez. Irksome: Automating runge–kutta time-stepping for finite element methods. *ACM Transactions on Mathematical Software (TOMS)*, 47(4):1–26, 2021.

[36] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[37] Kikuchi Fumio. Mixed and penalty formulations for finite element analysis of an eigenvalue problem in electromagnetism. *Computer Methods in Applied Mechanics and Engineering*, 64(1-3):509–521, 1987. doi: 10.1016/0045-7825(87)90053-3.

[38] Roberto Garrappa. Numerical evaluation of two and three parameter Mittag-Leffler functions. 2015. doi: 10.1137/140971191.

[39] Andrew Gillette and Tyler Kloefkorn. Trimmed serendipity finite element differential forms. *Mathematics of Computation*, 88(316):583–606, 2019. doi: 10.1090/mcom/3354.

[40] Andrew Gillette, Tyler Kloefkorn, and Victoria Sanders. Computational serendipity and tensor product finite element differential forms. *The SMAI Journal of Computational Mathematics*, 5:1–21, 2019. doi: 10.5802/smai-jcm.41.

[41] Rudolf Gorenflo and Francesco Mainardi. Fractional calculus. In *Fractals and fractional calculus in continuum mechanics*, pages 223–276. Springer, 1997.

[42] Kevin R Green and Raymond J Spiteri. Gating-enhanced imex splitting methods for cardiac monodomain simulation. *Numerical Algorithms*, 81(4):1443–1457, 2019.

[43] Max Gunzburger and Richard B Lehoucq. A nonlocal vector calculus with application to nonlocal boundary value problems. *Multiscale Modeling & Simulation*, 8(5):1581–1598,

2010.

[44] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3): 351–362, 2005,. doi: 10.1145/1089014.1089019.

[45] Richard Herrmann. *Fractional Calculus An Introduction for Physicists*. 2010.

[46] Anil N Hirani. *Discrete Exterior Calculus*. PhD thesis, Caltech, 2003.

[47] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[48] Miklós Homolya, Lawrence Mitchell, Fabio Luporini, and David A. Ham. TSFC: a structure-preserving form compiler. *SIAM Journal on Scientific Computing*, 40(3): C401–C428, 2018. doi: 10.1137/17M1130642.

[49] N Hooke, CS Henriquez, P Lanzkron, and D Rose. Linear algebraic transformations of the bidomain equations: implications for numerical methods. *Mathematical biosciences*, 120(2):127–145, 1994.

[50] Mariya Kamenova Ishteva. Properties and Applications of the Caputo Fractional Operator, 2005.

[51] Pankaj Jagad, Abdullah Abukhwejah, Mamdouh Mohamed, and Ravi Samtaney. A primitive variable discrete exterior calculus discretization of incompressible navier–stokes equations over surface simplicial meshes. *Physics of Fluids*, 33(1):017114, 2021.

[52] Robert C. Kirby. Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, 2004. doi: 10.1145/1039813.1039820.

[53] Robert C. Kirby. FIAT: numerical construction of finite element basis functions. In Logg et al. [59], pages 247–255. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8_13.

[54] Robert C Kirby, Anders Logg, Marie E Rognes, and Andy R Terrel. Common and unusual finite elements. In Logg et al. [59], pages 95–119. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8_3.

[55] P Robert Kotiuga. Theoretical limitations of discrete exterior calculus in the context of computational electromagnetics. *IEEE Transactions on Magnetics*, 44(6):1162–1165, 2008.

[56] John Lee. *Introduction to topological manifolds*, volume 202. Springer Science & Business Media, 2010.

[57] John M Lee. Smooth manifolds. In *Introduction to Smooth Manifolds*, pages 1–31. Springer, 2013.

[58] Anna Lischke, Guofei Pang, Mamikon Gulian, Fangying Song, Christian Glusa, Xiaoning Zheng, Zhiping Mao, Wei Cai, Mark M. Meerschaert, Mark Ainsworth, and George Em Karniadakis. What Is the Fractional Laplacian? pages 1–80, 2018. URL `http://arxiv.org/abs/1801.09767`.

[59] Anders Logg, Kent-Andre Mardal, and Garth N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method: the FEniCS Book*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8.

[60] Md Masud Rana, Victoria E Howle, Katharine Long, Ashley Meek, and William Milestone. A new block preconditioner for implicit runge–kutta methods for parabolic pde problems. *SIAM Journal on Scientific Computing*, 43(5):S475–S495, 2021.

[61] 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. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, January 2017. doi: 10.7717/peerj-cs.103.

[62] Mamdouh S Mohamed, Anil N Hirani, and Ravi Samtaney. Comparison of discrete hodge star operators for surfaces. *Computer-Aided Design*, 78:118–125, 2016.

[63] Mamdouh S Mohamed, Anil N Hirani, and Ravi Samtaney. Numerical convergence of discrete exterior calculus on arbitrary surface meshes. *International Journal for Computational Methods in Engineering Science and Mechanics*, 19(3):194–206, 2018.

[64] Patrick Mullen, Pooran Memari, Fernando de Goes, and Mathieu Desbrun. HOT: Hodge-optimized triangulations. In *ACM Transactions on Graphics (TOG)*, volume 30, page 103. ACM, 2011.

[65] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

[66] Steven A Niederer, Eric Kerfoot, Alan P Benson, Miguel O Bernabeu, Olivier Bernus, Chris Bradley, Elizabeth M Cherry, Richard Clayton, Flavio H Fenton, Alan Garny, et al. Verification of cardiac tissue electrophysiology simulators using an n-version benchmark. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1954):4331–4351, 2011.

[67] Oktar Ozgen, Marcelo Kallmann, Lynnette Es Ramirez, and Carlos Fm Coimbra. Underwater cloth simulation with fractional derivatives. *ACM Transactions on Graphics (TOG)*, 29(3):1–9, 2010. ISSN 0730-0301. URL `http://graphics.ucmerced.edu/papers/10-tog-fracdef.pdf`.

[68] Eric Paquet and Herna Lydia Viktor. Isometrically invariant description of deformable objects based on the fractional heat equation. In Richard Wilson, Edwin Hancock, Adrian Bors, and William Smith, editors, *Computer Analysis of Images and Patterns*, pages 135–143, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40246-3.

[69] 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. Firedrake: automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software*, 43(3):1–27, 2016. doi: 10.1145/2998441.

[70] Marie E. Rognes, Robert C. Kirby, and Anders Logg. Efficient assembly of $h(\div)$ and $h(\mathrm{curl})$ conforming finite elements. *SIAM Journal on Scientific Computing*, 31(6):4130–4151, 2010. doi: 10.1137/08073901X.

[71] Jose E. Roman, Carmen Campos, Lisandro Dalcin, Eloy Romero, and Andrés Tomás. SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 3.15, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2020. URL `https://slepc.upv.es/documentation/slepc.pdf`.

[72] John Paul Roop. Computational aspects of FEM approximation of fractional advection dispersion equations on bounded domains in R2. *Journal of Computational and Applied Mathematics*, 193(1):243–268, 2006. ISSN 03770427. doi: 10.1016/j.cam.2005.06.005.

[73] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. *Notices of the American Mathematical Society*, 65(09):1, October 2018. doi: 10.1090/noti1734. URL `https://doi.org/10.1090/noti1734`.

[74] Michael Spivak. *A comprehensive introduction to differential geometry*, volume 5. Publish or Perish, Incorporated, 1975.

[75] Joakim Sundnes, Glenn Terje Lines, Xing Cai, Bjørn Frederik Nielsen, Kent-Andre Mardal, and Aslak Tveito. *Computing the electrical activity in the heart*, volume 1. Springer Science & Business Media, 2007.

[76] Vasily E Tarasov. *Fractional dynamics: applications of fractional calculus to dynamics of particles, fields and media*. Springer Science & Business Media, 2011.

[77] Kirsten HWJ Ten Tusscher and Alexander V Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology-Heart and Circulatory*

Physiology, 291(3):H1088–H1100, 2006.

[78] Kirsten HWJ ten Tusscher, Denis Noble, Peter-John Noble, and Alexander V Panfilov. A model for human ventricular tissue. *American Journal of Physiology-Heart and Circulatory Physiology*, 286(4):H1573–H1589, 2004.

[79] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. Directional field synthesis, design, and processing. In *Computer Graphics Forum*, volume 35, pages 545–572. Wiley Online Library, 2016.

[80] Kevin P Vincent, Matthew J Gonzales, Andrew K Gillette, Christopher T Villongco, Simone Pezzuto, Jeffrey H Omens, Michael J Holst, and Andrew D McCulloch. High-order finite element methods for cardiac monodomain simulations. *Frontiers in physiology*, 6: 217, 2015.

[81] Zenodo. Software used in 'Bringing Trimmed Serendipity Methods to Computational Practice in Firedrake', apr 2021. URL `https://doi.org/10.5281/zenodo.4701708`.

[82] Yunying Zheng, Changpin Li, and Zhengang Zhao. A note on the finite element method for the space-fractional advection diffusion equation. *Computers & Mathematics with Applications*, 59(5):1718–1726, 2010.