The Pennsylvania State University

The Graduate School

College of Engineering

LITHIUM ION BATTERY MODELING USING ORTHOGONAL PROJECTIONS AND DESCRIPTOR FORM

A Thesis in

Mechanical Engineering

by

Michael D. Beeney

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

August 2013

The thesis of Michael D. Beeney was reviewed and approved* by the following:

Hosam Fathy
Assistant Professor of Mechanical Engineering
Thesis Advisor

Chris Rahn
Professor of Mechanical Engineering

Karen A. Thole
Professor of Mechanical Engineering
Department Head of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School.

Abstract

This thesis focuses on computationally efficient methods to solve the equations of the Doyle Fuller Newman electrochemical battery model. The two methods used in this work are orthogonal projections to solve partial differential equations and the descriptor form to solve a system of differential algebraic equations. An orthonormal set of polynomials is required for the orthogonal projection method. Two sets of Legendre Polynomials are used, one that is orthonormal over the thickness of the battery, and another over the radius of the solid electrode particles. The orthogonal projection method is then benchmarked against the finite difference method to assess the benefits and drawbacks. It is shown that the orthogonal projection method is computationally faster than the finite difference method with very little loss in accuracy. It can also be shown that the descriptor form not only simplifies the inclusion of the boundary conditions, but also increases the computational speed of the model.

Based on the knowledge that there is a computational benefit in using orthogonal projections in descriptor form, all five components of the Doyle Fuller Newman model are solved using orthogonal projections. The nonlinear components of the model are linearized at each time step. The battery model is capable of running faster than real time while providing results similar to the manufacturer's data [1].

Table of Contents

List of Figures

List of Tables

List of Symbols

Table 1. Orthogonal Projection Expressions

Symbol	Description	Units
$c_{1,n}(r,x,t)$	Solid Phase Concentration (Negative)	$mol\ m^{-3}$
$c_{1,p}(r,x,t)$	Solid Phase Concentration (Positive)	$mol\ m^{-3}$
$c_{2,n}(x,t)$	Solution Phase Concentration (Negative)	$mol\ m^{-3}$
$c_{2,s}(x,t)$	Solution Phase Concentration (Separator)	$mol\ m^{-3}$
$c_{2,p}(x,t)$	Solution Phase Concentration (Positive)	$mol\ m^{-3}$
$P_{1,n}(x,t)$	Solid Phase Potential (Negative)	V
$P_{1,p}(x,t)$	Solid Phase Potential (Positive)	V
$P_{2,n}(x,t)$	Solution Phase Potential (Negative)	V
$P_{2,s}(x,t)$	Solution Phase Potential (Separator)	V
$P_{2,p}(x,t)$	Solution Phase Potential (Positive)	V
$J_n(x,t)$	Intercalation Current (Negative)	$A\ m^{-2}$
$J_p(x,t)$	Intercalation Current (Positive)	$A\ m^{-2}$

Table 2. Legendre Polynomials

Symbol	Description	Units
$\phi_n(x)$	Cartesian Legendre Polynomial (Negative)	–
$\phi_s(x)$	Cartesian Legendre Polynomial (Separator)	–
$\phi_p(x)$	Cartesian Legendre Polynomial (Positive)	–
$\omega_n(r)$	Radial Legendre Polynomial (Negative)	–
$\omega_p(r)$	Radial Legendre Polynomial (Positive)	–

Table 3. Battery Parameters

Symbol	Description	Units
$d_{1,n}$	Solid Diffusivity (Negative)	$m^2 s^{-1}$
a_n	Specific Interfacial Area (Negative)	m^{-1}
R_n	Particle Radius (Negative)	m
$d_{1,p}$	Solid Diffusivity (Positive)	$m^2 s^{-1}$
a_p	Specific Interfacial Area (Positive)	m^{-1}
R_p	Particle Radius (Positive)	m
F	Faraday Constant	$C\ mol^{-1}$
ϵ_n	Volume Fraction (Negative)	–
$d_{2,n}$	Solution Diffusivity (Negative)	$m^2 s^{-1}$
t_n^+	Transference Number (Negative)	–
ϵ_s	Volume Fraction (Separator)	–
$d_{2,s}$	Solution Diffusivity (Separator)	$m^2 s^{-1}$
ϵ_p	Volume Fraction (Positive)	–
$d_{2,p}$	Solution Diffusivity (Positive)	$m^2 s^{-1}$
t_p^+	Transference Number (Positive)	–
σ_n^{eff}	Effective Conductivity (Negative)	$m^{-1}\ \Omega^{-1}$
σ_p^{eff}	Effective Conductivity (Positive)	$m^{-1}\ \Omega^{-1}$
i_{app}	Applied Current	A
$Area$	Current Collector Area	m^2
κ_n^{eff}	Electrolyte Conductivity (Negative)	$m^{-1}\ \Omega^{-1}$
$\kappa_{D,n}$	Diffusion Constant (Negative)	$A\ m^2\ mol^{-1}$
κ_s^{eff}	Electrolyte Conductivity (Separator)	$m^{-1}\ \Omega^{-1}$
$\kappa_{D,s}$	Diffusion Constant (Separator)	$A\ m^2\ mol^{-1}$
κ_p^{eff}	Electrolyte Conductivity (Positive)	$m^{-1}\ \Omega^{-1}$
$\kappa_{D,p}$	Diffusion Constant (Positive)	$A\ m^2\ mol^{-1}$
R	Universal Gas Constant	$J\ K^{-1}\ mol^{-1}$
T	Temperature	K
$\alpha_{a,n}$	Apparent Exchange Coefficient (Negative)	–
$\alpha_{c,n}$	Apparent Exchange Coefficient (Negative)	–
k_n	Reaction Rate (Negative)	$(Am^{-2})(mol\ m^3)^{1+\alpha}$
u_{nref}	Equilibrium Potential (Negative)	V
$\alpha_{a,p}$	Apparent Exchange Coefficient (Positive)	–
$\alpha_{c,p}$	Apparent Exchange Coefficient (Positive)	–
k_p	Reaction Rate (Positive)	$(Am^{-2})(mol\ m^3)^{1+\alpha}$
u_{pref}	Equilibrium Potential (Positive)	V

Acknowledgments

I would like to first and foremost thank my advisor, Dr. Hosam Fathy for his guidance in creating my thesis, along with his insight and wisdom in making me a better researcher, scholar, and thinker. For all of this I am truly grateful.

I would also like to thank Michelle Kehs for her help in creating the battery model.

I would be remiss not to thank the past and present members of the Control Optimization Laboratory for their thoughts, ideas, help, words of encouragement, baked goods, interesting conversations, white board drawings, and reminders to always check. You guys are the *literally* the COOLest research group, ever. But seriously... thanks!

Also, I have to thank my parents for their love and support throughout my life (starting with trips to Penn State Altoona as a baby). You were always there for me in the good and bad times, and for that I am forever appreciative.

Last but most certainly not least, I have to thank Lindsey for her love, patience and support of my six years of college.

Dedication

To the 42% I received on my first college exam. The motivation it provided is priceless.

Chapter 1

Introduction

This thesis studies the tradeoffs between computational speed and accuracy of an electrochemical battery model solved using orthogonal projections in descriptor form. The orthogonal projection method is a computationally efficient method of solving partial differential equations (PDEs), and the descriptor form is an excellent method for solving differential algebraic equations (DAEs). Other authors have used orthogonal projections, however, none have addressed several fundamental questions, such as the benefits and drawbacks of using orthogonal projections over other methods, and how many orthogonal projections are necessary for accurate results. To the best of the author's knowledge, this thesis answers both these questions for the first time.

This thesis also contains an electrochemical battery model solved exclusively with orthogonal projections in descriptor form. The model is capable of running faster than real time while giving important battery data that is difficult to measure physically.

1.1 Lithium Ion Battery History

Since the commercialization of the rechargeable lithium-ion battery by Sony in 1991 [16], the use of lithium ion batteries has increased dramatically, thanks to their high energy and power density, low internal impedance, and long cycle life compared to lead acid and nickel based battery cells [2]. Today, lithium ion cells are used in applications ranging from cell phones and laptops to hybrid and electric cars and aircraft. With this growing demand, there is a need to control batteries to maximize their performance and their lifetime. To do this, control oriented battery models that are capable of running in real time onboard a system are required. There are many types of battery models, only some of which are suitable for control-oriented applications. Battery models range from the simplest equivalent circuit model, all the way to Kinetic Monte Carlo models that require the use of supercomputers [14]. In the next section I focus on three common control-oriented models; Empirical Models, Single Particle Model, and Pseudo 2 Dimensional Model.

1.2 Comparison of Control Oriented Battery Models

Broadly speaking, there are three types of control oriented lithium ion battery models. The simplest is empirical based equivalent circuit models (ECM). These models use basic elements from electrical circuits, such as capacitors, resistors, and inductors, connected in series and parallel to replicate the dynamics of a battery. These models are advantageous because they are very simple, and can be created by fitting experimental data to electric circuits. Disadvantages include the need for accurate battery data, and a lack of first principle modeling to understand electrochemical battery behavior.

The second type of control oriented battery model is the single particle model (SPM). The SPM represents the solid phase of the cathode and the anode each as one single particle, where lithium ions diffuse in and out of the particle radially. This model is based on electrochemical fundamentals, but because the solution phase is not modeled, the SPM is only valid at low current values.

The final type of battery model is the pseudo 2 dimensional model (P2DM), pioneered by Doyle et al. [5]. The name comes from the solid concentration being modeled using a radial diffusion equation while the solution concentration is modeled using a diffusion equation in the Cartesian direction. This model is the most complicated of the three control-oriented models discussed, but it is the most accurate and allows for high current rates. The Doyle Fuller Newman (DFN) model is discussed in more detail in the following section.

1.3 Doyle Fuller Newman Model

The DFN model, developed in 1993 [5] and expanded in 1994 [8], models a generic lithium ion battery as a pseudo 2 dimensional rocking chair, where lithium ions are inserted and extracted in the positive and negative electrodes. The electrodes themselves are comprised of active particles, electrolyte, and a binding material that does not react during operation of the battery. The separator is made of an inert material that allows for lithium ions to diffuse through the medium, but does not allow electrons to travel through the separator.

For the present work, the particular battery chemistry is lithium iron phosphate ($LiFePO_4$). The DFN model was adapted for this particular chemistry by Forman et al. [7] for use in genetic identification and Fisher identifiability studies.

For this particular chemistry, there are five main components that need to be modeled. The first component of the DFN model governs the concentration in the solid phase c_1. The concentration is governed by the following partial differential equation:

$$\frac{\partial c_1}{\partial t} = \frac{d_1}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial c_1}{\partial r}) \tag{1.1}$$

Two boundary conditions are needed. The first is located at the center of the sphere while the second is at the surface of the spherical particle. These two boundary conditions are:

$$\frac{\partial c_1}{\partial r} = 0, \quad r = 0 \tag{1.2}$$

$$\frac{\partial c_1}{\partial r} = \frac{-J}{d_1 aF}, \quad r = R \tag{1.3}$$

The second part of the DFN model governs the concentration in the solution phase c_2. The concentration is driven by a diffusion phenomenon over three domains; the negative electrode, the separator, and the positive electrode. The partial differential equation for c_2 is:

$$\epsilon_2\frac{\partial c_2}{\partial t} = \frac{\partial}{\partial x}(d_2{}^{eff}\frac{\partial c_2}{\partial x}) + \frac{1-t^+}{F}J \tag{1.4}$$

The third piece of the DFN model governs the potential in the solid phase, P_1. This is an ODE dictating P_1 as a function of the x direction at every instant in time. The solid phase potential has the form:

$$\frac{\partial}{\partial x}(\sigma^{eff}\frac{\partial P_1}{\partial x}) - J = 0 \tag{1.5}$$

The fourth part of the DFN model governs the potential in the solution phase P_2. Similar to P_1 this is an ODE dictating that spatial distribution of potential, however there is a nonlinear component that depends on the solution concentration c_2. The equation for the solution phase potential is shown below in Equation 1.6.

$$\frac{\partial}{\partial x}(\kappa^{eff}\frac{\partial P_2}{\partial x}) + J + \frac{\partial}{\partial x}(\kappa_D\frac{\partial}{\partial x}\ln(c_2)) = 0 \tag{1.6}$$

The last component of the DFN model governs the charge transfer kinetics between the solid

and solution phase. This is the Butler-Volmer equation:

$$\begin{aligned} J &= ai_0[\exp(\frac{\alpha_a F}{RT}\eta) - \exp(-\frac{\alpha_c F}{RT}\eta)] \\ i_0 &= k_j({c_1}^{max} - {c_1}^s)^{\alpha_a}({c_1}^s)^{\alpha_c}(c_2)^{\alpha_a} \\ \eta &= P_1 - P_2 - u_{ref} \end{aligned} \tag{1.7}$$

Equations 1.1 - 1.7 are discussed in more detail in Chapter 4.

1.4 Model Reduction Techniques

Previous research has looked at various ways to solve the equations of the DFN model in an accurate and computationally efficient manner. The equations of the DFN model are complicated to solve due to the PDEs that appear in the solid and solution phase concentrations, along with the algebraic constraints governed by the charge transfer kinetics. To solve the PDEs in the DFN model, there are many methods such as Finite Element, Finite Difference, etc. [13]. These methods are simple to construct, however they are computationally expensive. Model reduction is necessary for the DFN model to be computationally efficient.

Smith et al. [15] transforms the solid phase diffusion PDE into an infinite pole transcendental transfer function. From there, the authors reduce the order of the model by truncating high frequency pole/residue pairs, and grouping similar poles into bins for further reduction. For the solution phase, the authors use the finite element method to solve the solution concentration in the negative electrode - separator - positive electrode system. The original forty-fifth order model is reduced to just three states by grouping similar poles.

Guo et al. [9] use the eigenfunction expansion method to transform the solid phase diffusion partial differential equation into a series of ordinary differential equations for a single particle model. Their results show that 10 eigenfuctions produced nearly identical results as 2000 eigenfunctions. It should be noted that these results are only valid at low current rates because the solution phase concentration is negelected.

Forman et al. [6] use a set of Padé approximations to reduce the number of state equations for the solid phase portion of the model. They compare the Bode plots of the exact solution, the Padé approximation and the finite difference method. It is shown that the Padé approximation is very accurate at lower frequencies and is more accurate than a finite difference for a given order.

In addition, Forman et al. quasi-linearize the algebraic components of the Doyle Fuller Newman model. This allows for the non-linear Butler-Volmer equation to be linearized at each time step.

Cai and White [3] use Proper Orthogonal Decomposition (POD) to reduce the DFN model. The full electrochemical model contained 14,580 equations, which is reduced to just 50 using POD. The authors find only a 1.3% error between the full and reduced models at a current of 10C. The primary downside of this method is that either experimental data or a high order model needs to be obtained to develop a reduced order model using POD.

Cai and White also use orthogonal collocation on finite elements (OCFE) as a model reduction technique [4]. In this paper, the authors use finite elements to discretize the spherical particles. On the inner points within each element, the collocation points are located at the roots of the Jacobi polynomial. They compare a reduced order model using OCFE that contains 1,608 equations to a full order model using 11,270 finite elements and found similar results, with the reduced order model being 30 times faster than the original.

Subramanian et al. approximate the spherical diffusion process as a two or three parameter model that has the form [17]:

$$c(r,t) = a(t) + b(t)(\frac{r^2}{R^2}) + d(t)(\frac{r^4}{R^4}) \tag{1.8}$$

By selecting even powers of r/R, the authors are able to guarantee that the center Neumann boundary condition is automatically satisfied. The authors create a twenty node finite difference model of spherical diffusion to compare to the two parameter model and the three parameter model. They state that the two parameter model did well in some cases, but the three parameter model matches the finite difference model very well for all cases the authors tested.

Northrop et al. [12] reformulates the solution phase concentration component of the Doyle Fuller Newman model by using the orthogonal collocation method. This method, similar to the one in the present work, separates variables that are functions of time and space by the following form:

$$u(X,t) = F(X,t) + \sum_{k=0}^{N} B_k(t)T_k(X) \tag{1.9}$$

Where $u(X,t)$ is the time and spatial dependent variable, $T_k(X)$ are the trial functions with homogeneous boundary conditions, $F(X,t)$ satisfies the boundary conditions, and lastly $B_k(t)$

are the time varying weights on the trial functions. The trial functions themselves must be linearly independent. The authors choose the trial functions to be cosine functions of the form $cos(k\pi X)$. The collocation points are points within the domain where the governing differential equations are exactly satisfied. After reduction, the model is reduced to a set of 24 differential algebraic equations for the solution phase concentration.

It is important to note that these methods can be used independently for each component of the DFN model. For instance, Li and Choe [11] use [17] to reduce the solid phase concentration while using the work of [15] to reduce the solution phase concentration.

Chapter 2

Legendre Polynomials

The Orthogonal Projection Method utilizes a set of functions that are orthonormal to each other, over the domain 0 to L for the Cartesian components of the DFN model and $-R$ to R for the radial components. For the current work, Legendre Polynomials are used. The Legendre Polynomials are usually written in the domain $[-1, 1]$ and are orthogonal to each other, but typically not normalized. The non-normalized, non-shifted Legendre Polynomials obey the following integral [10]:

$$\int_{-1}^{1} \phi_m(x)\phi_n(x)\mathrm{d}x = \frac{2\delta_{mn}}{2n+1}$$
$$\delta_{mn} = \begin{cases} 1 & m = n \\ 0 & m \neq n \end{cases} \tag{2.1}$$

Table 2.1 shows the first six non-normalized, non-shifted Legendre Polynomials. Figure 2.1 shows the first six non-normalized, non-shifted Legendre Polynomials. Appendix A.1 contains the code to plot Figure 2.1.

Table 2.1. First six non-normalized, non-shifted Legendre Polynomials [10]

Order n	Legendre Polynomial $\phi_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2 - 1)$
3	$\frac{1}{2}(5x^3 - 3x)$
4	$\frac{1}{8}(35x^4 - 30x^2 + 3)$
5	$\frac{1}{8}(65x^5 - 70x^3 + 15x)$

2.1 Shifting and Normalizing the Cartesian Legendre Polynomials

For the components of the Doyle Fuller Newman model that are a function of the x direction, the Legendre Polynomials must be orthonormal in the interval $[0, L]$. To shift and normalize the Legendre Polynomials, the Legendre Polynomials must obey the integral shown in equation 2.2.

$$\int_{0}^{L} \phi_m(x)\phi_n(x)\mathrm{d}x = \delta_{mn} \tag{2.2}$$

The process to determine the shifted, normalized Legendre Polynomials starts by assuming

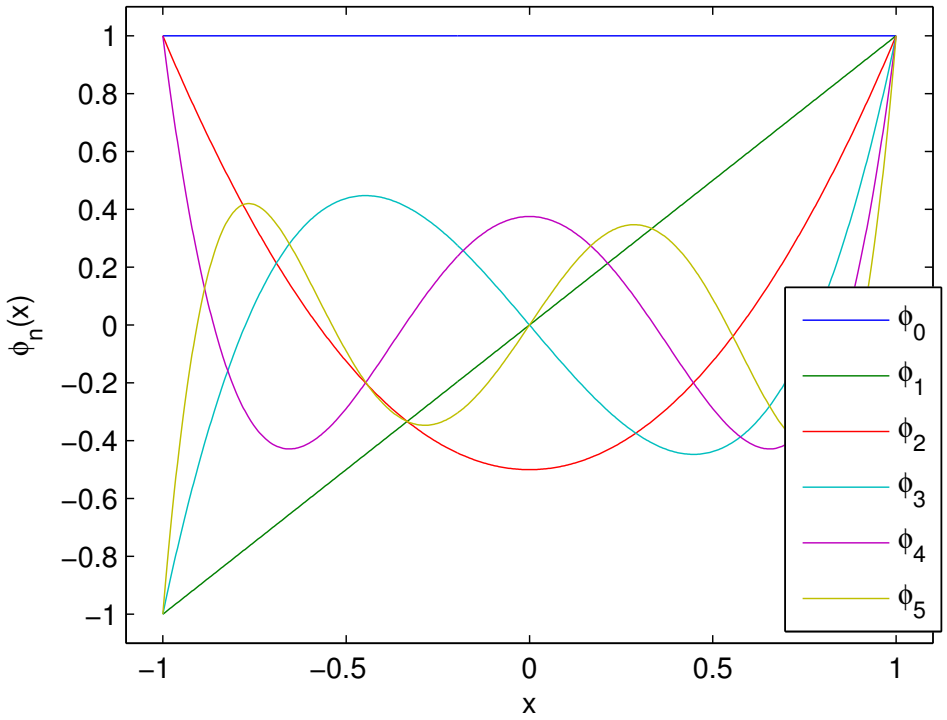

Figure 2.1. First six non-normalized, non-shifted Legendre Polynomials

the Legendre Polynomials have the form:

$$\begin{aligned} \phi_0(x) &= a_0 \\ \phi_1(x) &= a_1 x + a_2 \\ \phi_2(x) &= a_3 x^2 + a_4 x + a_5 \\ \phi_3(x) &= a_6 x^3 + a_7 x^2 + a_8 x + a_9 \\ &\vdots \end{aligned} \tag{2.3}$$

After this, the zeroth order Legendre Polynomial is solved by using equation 2.2. Only one equation is needed to solve for the single unknown coefficient. Once the unknown coefficient is found, then two equations are generated to solve the two unknowns in $\phi_1(x)$. These equations come by multiplying and integrating $\phi_0(x)$ and $\phi_1(x)$ together, and by squaring $\phi_1(x)$ and integrating. This process, known as the method of undetermined coefficients, continues until a sufficient number of Legendre Polynomials have been found. In Appendix A.2 the first three transformed Legendre Polynomial are solved. In Table 2.2 the first six shifted, normalized Legendre Polynomials are listed. In Figures 2.2 and 2.3, the polynomials are plotted for $L = 1$ and

$L = 3$ respectively.

Table 2.2. First six normalized, shifted Legendre Polynomials

Order n	Legendre Polynomial $\phi_n(x)$
0	$\sqrt{\frac{1}{L}}$
1	$\sqrt{\frac{3}{L}}[\frac{2x}{L} - 1]$
2	$\frac{1}{2}\sqrt{\frac{5}{L}}[3(\frac{2x}{L} - 1)^2 - 1]$
3	$\frac{1}{2}\sqrt{\frac{7}{L}}[5(\frac{2x}{L} - 1)^3 - 3(\frac{2x}{L} - 1)]$
4	$\frac{1}{8}\sqrt{\frac{9}{L}}[35(\frac{2x}{L} - 1)^4 - 30(\frac{2x}{L} - 1)^2 + 3]$
5	$\frac{1}{8}\sqrt{\frac{11}{L}}[63(\frac{2x}{L} - 1)^5 - 70(\frac{2x}{L} - 1)^3 + 15(\frac{2x}{L} - 1)]$

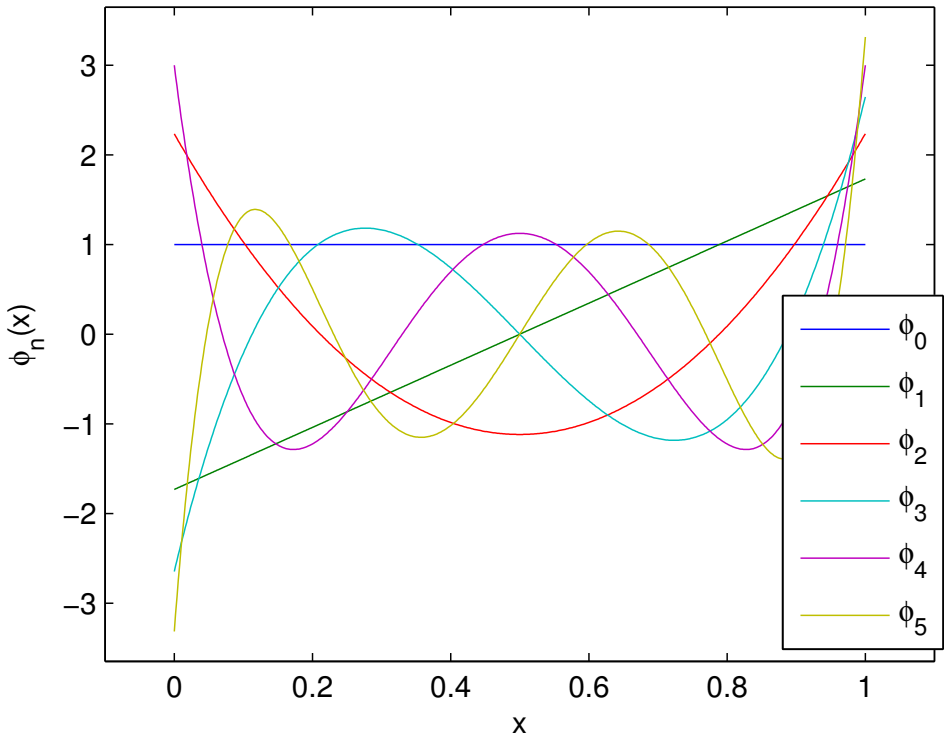

Figure 2.2. Normalized, Shifted Legendre Polynomials $L = 1$

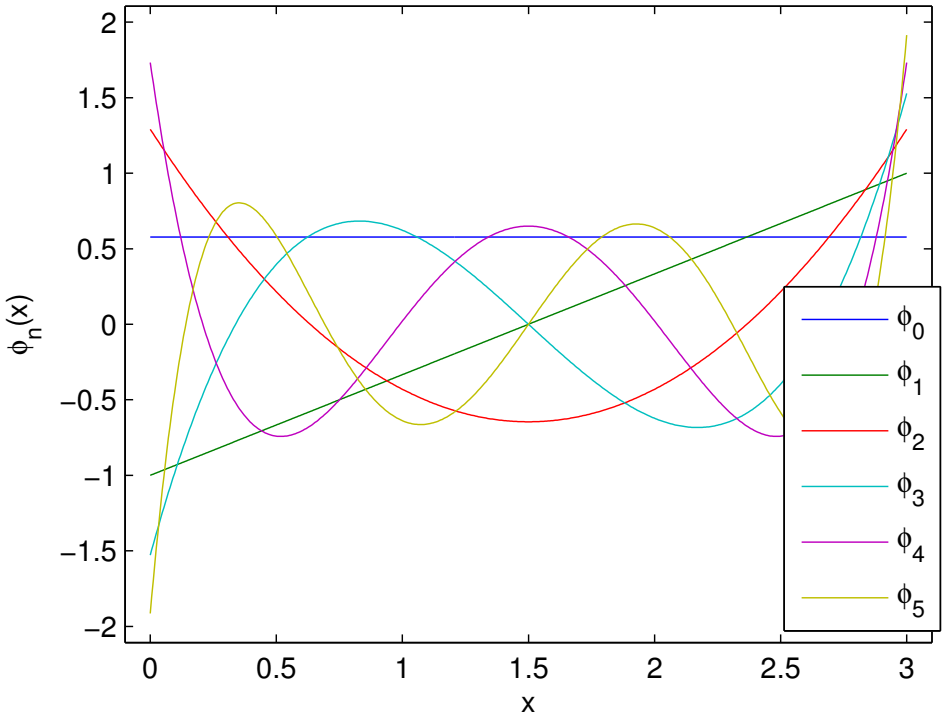

Figure 2.3. Normalized, Shifted Legendre Polynomials $L = 3$

2.2 Normalizing the Radial Legendre Polynomials

To use orthogonal projections for the radial components of the Doyle Fuller Newman model, the Legendre Polynomials must have a zero slope at $r = 0$ and be orthonormal over the interval $[0, R]$. These two conditions yield the following two integrals:

$$\int_{-R}^{R} \phi_m(r)\phi_n(r)\mathrm{d}r = 0, m \neq n \tag{2.4}$$

$$\int_{0}^{R} (\phi_n(r))^2\mathrm{d}r = 1 \tag{2.5}$$

Similar to the Cartesian case, it is assumed that the radial Legendre Polynomials take the form:

$$\begin{aligned} \phi_0(r) &= b_0 \\ \phi_1(r) &= b_1 r + b_2 \\ \phi_2(r) &= b_3 r^2 + b_4 r + b_5 \\ \phi_3(r) &= b_6 r^3 + b_7 r^2 + b_8 r + b_9 \\ &\vdots \end{aligned} \tag{2.6}$$

Equation 2.5 is used to determine b_0 and subsequently $\phi_0(r)$. Once this is completed, b_1 and b_2 are found using equations 2.4 and 2.5 in conjunction with $\phi_0(r)$. This process continues until a sufficent number of radial Legendre Polynomials are found. In Appendix A.4 the first three radial Legendre Polynomials are calculated. Table 2.3 shows the first seven radial Legendre Polynomials, while Figures 2.4 and 2.5 plots the polynomials when $R = 1$ and $R = 3$.

Table 2.3. First seven normalized Radial Legendre Polynomials

Order n	Legendre Polynomial $\phi_n(r)$
0	$\sqrt{\frac{1}{R}}$
1	$\sqrt{\frac{3}{R}}[\frac{r}{R}]$
2	$\frac{1}{2}\sqrt{\frac{5}{R}}[3(\frac{r}{R})^2 - 1]$
3	$\frac{1}{2}\sqrt{\frac{7}{R}}[5(\frac{r}{R})^3 - 3(\frac{r}{R})]$
4	$\frac{1}{8}\sqrt{\frac{9}{R}}[35(\frac{r}{R})^4 - 30(\frac{r}{R})^2 + 3]$
5	$\frac{1}{8}\sqrt{\frac{11}{R}}[63(\frac{r}{R})^5 - 70(\frac{r}{R})^3 + 15(\frac{r}{R})]$
6	$\frac{1}{16}\sqrt{\frac{13}{R}}[231(\frac{r}{R})^6 - 315(\frac{r}{R})^4 + 105(\frac{r}{R})^2 - 5]$

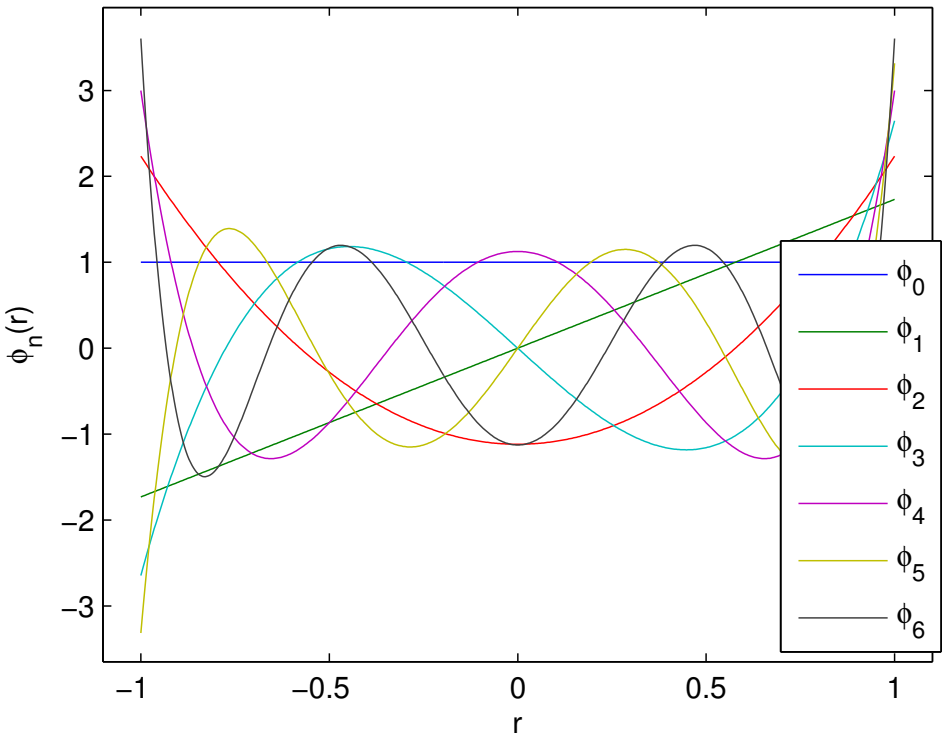

Figure 2.4. Normalized, Radial Legendre Polynomials $R = 1$

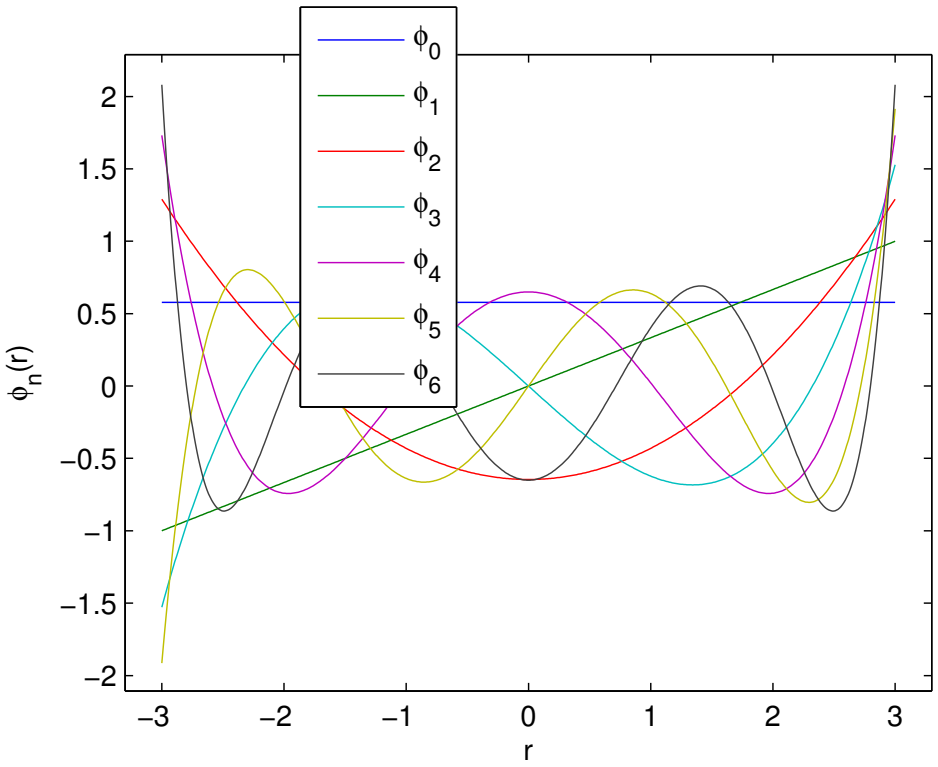

Figure 2.5. Normalized Radial Legendre Polynomials $R = 3$

Chapter 3

Orthogonal Projection Method

3.1 Comparison of Finite Difference Method to Orthogonal Projection Method

To benchmark the speed and accuracy of the Orthogonal Projection Method, (OPM) a Cartesian one domain diffusion system, shown in Figure 3.1, is created and solved using three variations of the OPM and the Finite Difference Method (FDM) in MATLAB. The partial differential equation governing this this system is:

$$\epsilon \frac{\partial c(x,t)}{\partial t} = D \frac{\partial^2 c(x,t)}{\partial x^2} - \frac{1-t^+}{F} J(x,t) \tag{3.1}$$

The two boundary conditions on each end of the domain are:

$$\left. \frac{\partial c(x,t)}{\partial x} \right|_{x=0,1} = 0 \tag{3.2}$$

It should be noted for this code, it is assumed $J(x,t)$ is an input into the one domain system, and the constants used in the simulation are not representative of real battery values.

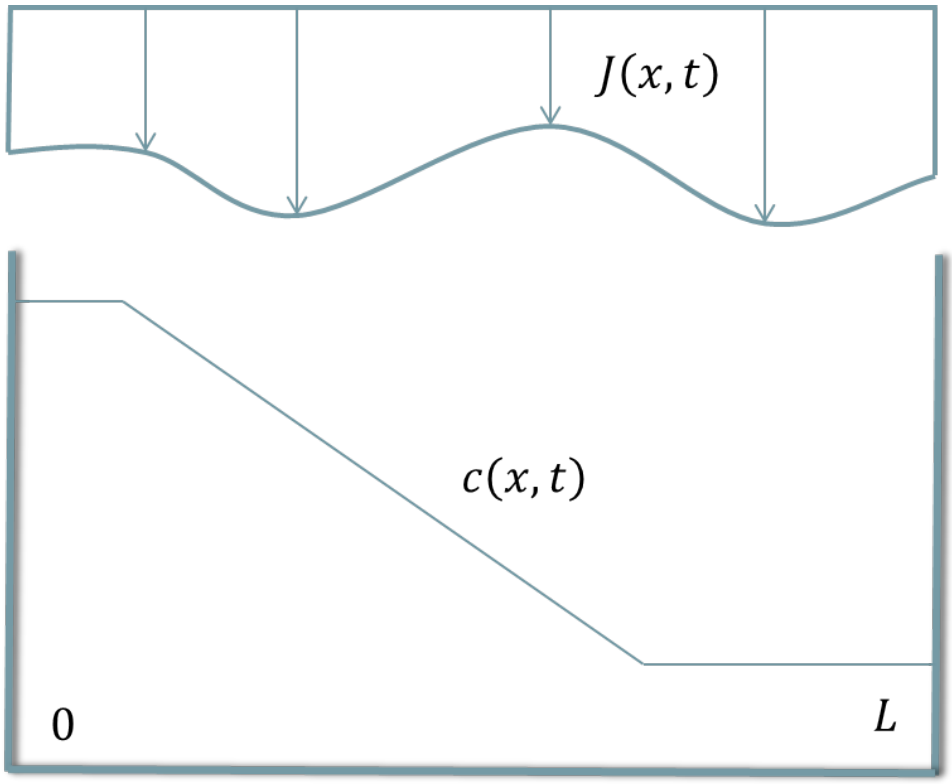

Figure 3.1. Cartesian One Domain Diffusion System

3.1.1 Finite Difference Method

The partial differential governing equation is converted into a large number of ordinary differential equations by discretizing the x dimension into equally spaced nodes. The MATLAB command lsim is used to solve the resulting ODEs. The partial derivatives with respect to space in the governing equation and boundary conditions are approximated by:

$$\left.\frac{\partial c(x,t)}{\partial x}\right|_{x=0} \approx \frac{-c(1,j)+c(2,j)}{\Delta x} \tag{3.3}$$

$$\frac{\partial^2 c(x,t)}{\partial x^2} \approx \frac{c(i-1,j)-2c(i,j)+c(i+1,j)}{\Delta x^2} \tag{3.4}$$

$$\left.\frac{\partial c(x,t)}{\partial x}\right|_{x=1} \approx \frac{c(N-1,j)-c(N,j)}{\Delta x} \tag{3.5}$$

The FDM can then be cast in state space form:

$$[\dot{\vec{c}}] = [A][\vec{c}] + [B][\vec{J}] \tag{3.6}$$

The resulting A and B state space matrices are:

$$A = \begin{bmatrix} \frac{-D}{\epsilon\Delta x^2} & \frac{D}{\epsilon\Delta x^2} & 0 & \cdots & 0 & 0 & 0 \\ \frac{D}{\epsilon\Delta x^2} & \frac{-2D}{\epsilon\Delta x^2} & \frac{D}{\epsilon\Delta x^2} & \cdots & 0 & 0 & 0 \\ 0 & \frac{D}{\epsilon\Delta x^2} & \frac{-2D}{\epsilon\Delta x^2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{-2D}{\epsilon\Delta x^2} & \frac{D}{\epsilon\Delta x^2} & 0 \\ 0 & 0 & 0 & \cdots & \frac{D}{\epsilon\Delta x^2} & \frac{-2D}{\epsilon\Delta x^2} & \frac{D}{\epsilon\Delta x^2} \\ 0 & 0 & 0 & \cdots & 0 & \frac{D}{\epsilon\Delta x^2} & \frac{-D}{\epsilon\Delta x^2} \end{bmatrix} \tag{3.7}$$

$$B = \begin{bmatrix} \frac{1-t^+}{\epsilon F} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1-t^+}{\epsilon F} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{1-t^+}{\epsilon F} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1-t^+}{\epsilon F} & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & \frac{1-t^+}{\epsilon F} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & \frac{1-t^+}{\epsilon F} \end{bmatrix} \tag{3.8}$$

$J(x,t)$ is also discretized into equally spaced nodes spatially, and discretized into nodes temporally. The MATLAB code for the FDM is found in Appendix B.1.

3.1.2 Orthogonal Projection Method (Fourth Order)

The Orthogonal Projection Method takes variables that are functions of time and space, and separates them into unknown functions of time and known functions of space. For the present work, the known functions of space are the Cartesian Legendre Polynomials discussed in Section 2.1. For the Cartesian, one domain diffusion problem, the concentration $c(x,t)$ and input $J(x,t)$

are approximated using fourth order Orthogonal Projections:

$$c(x,t) = \beta_0(t)\phi_0(x) + \beta_1(t)\phi_1(x) + \beta_2(t)\phi_2(x) + \beta_3(t)\phi_3(x) \tag{3.9}$$

$$J(x,t) = \gamma_0(t)\phi_0(x) + \gamma_1(t)\phi_1(x) + \gamma_2(t)\phi_2(x) + \gamma_3(t)\phi_3(x) \tag{3.10}$$

Substituting equations 3.9 and 3.10 into equation 3.1:

$$\begin{aligned} &\begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & \phi_3 \end{bmatrix} \begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix} \\ &= \frac{D}{\epsilon} \begin{bmatrix} \phi_0'' & \phi_1'' & \phi_2'' & \phi_3'' \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \frac{1-t^+}{F\epsilon} \begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & \phi_3 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \end{aligned} \tag{3.11}$$

To separate the β_n terms from each other, it is important to take advantage of the orthonormal property of the Legendre Polynomials. By applying Galerkin Projections ($\int_0^L \phi_n(x)\mathrm{d}x$) to equation 3.11, the governing equation then becomes:

$$\begin{aligned} &\begin{bmatrix} \int_0^L \phi_0\phi_0\mathrm{d}x & \int_0^L \phi_1\phi_0\mathrm{d}x & \int_0^L \phi_2\phi_0\mathrm{d}x & \int_0^L \phi_3\phi_0\mathrm{d}x \\ \int_0^L \phi_0\phi_1\mathrm{d}x & \int_0^L \phi_1\phi_1\mathrm{d}x & \int_0^L \phi_2\phi_1\mathrm{d}x & \int_0^L \phi_3\phi_1\mathrm{d}x \\ \int_0^L \phi_0\phi_2\mathrm{d}x & \int_0^L \phi_1\phi_2\mathrm{d}x & \int_0^L \phi_2\phi_2\mathrm{d}x & \int_0^L \phi_3\phi_2\mathrm{d}x \\ \int_0^L \phi_0\phi_3\mathrm{d}x & \int_0^L \phi_1\phi_3\mathrm{d}x & \int_0^L \phi_2\phi_3\mathrm{d}x & \int_0^L \phi_3\phi_3\mathrm{d}x \end{bmatrix} \begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix} \\ &= \frac{D}{\epsilon} \begin{bmatrix} \int_0^L \phi_0''\phi_0\mathrm{d}x & \int_0^L \phi_1''\phi_0\mathrm{d}x & \int_0^L \phi_2''\phi_0\mathrm{d}x & \int_0^L \phi_3''\phi_0\mathrm{d}x \\ \int_0^L \phi_0''\phi_1\mathrm{d}x & \int_0^L \phi_1''\phi_1\mathrm{d}x & \int_0^L \phi_2''\phi_1\mathrm{d}x & \int_0^L \phi_3''\phi_1\mathrm{d}x \\ \int_0^L \phi_0''\phi_2\mathrm{d}x & \int_0^L \phi_1''\phi_2\mathrm{d}x & \int_0^L \phi_2''\phi_2\mathrm{d}x & \int_0^L \phi_3''\phi_2\mathrm{d}x \\ \int_0^L \phi_0''\phi_3\mathrm{d}x & \int_0^L \phi_1''\phi_3\mathrm{d}x & \int_0^L \phi_2''\phi_3\mathrm{d}x & \int_0^L \phi_3''\phi_3\mathrm{d}x \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \\ &+ \frac{1-t^+}{F\epsilon} \begin{bmatrix} \int_0^L \phi_0\phi_0\mathrm{d}x & \int_0^L \phi_1\phi_0\mathrm{d}x & \int_0^L \phi_2\phi_0\mathrm{d}x & \int_0^L \phi_3\phi_0\mathrm{d}x \\ \int_0^L \phi_0\phi_1\mathrm{d}x & \int_0^L \phi_1\phi_1\mathrm{d}x & \int_0^L \phi_2\phi_1\mathrm{d}x & \int_0^L \phi_3\phi_1\mathrm{d}x \\ \int_0^L \phi_0\phi_2\mathrm{d}x & \int_0^L \phi_1\phi_2\mathrm{d}x & \int_0^L \phi_2\phi_2\mathrm{d}x & \int_0^L \phi_3\phi_2\mathrm{d}x \\ \int_0^L \phi_0\phi_3\mathrm{d}x & \int_0^L \phi_1\phi_3\mathrm{d}x & \int_0^L \phi_2\phi_3\mathrm{d}x & \int_0^L \phi_3\phi_3\mathrm{d}x \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \end{aligned} \tag{3.12}$$

The orthonormality of the Legendre Polynomials simplifies the equation to a state space representation of the form:

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix} = \frac{D}{\epsilon} \begin{bmatrix} 0 & 0 & \frac{12\sqrt{5}}{L^2} & 0 \\ 0 & 0 & 0 & \frac{20\sqrt{21}}{L^2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \frac{1-t^+}{F\epsilon} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \tag{3.13}$$

Boundary conditions need to be imposed. To do this, the expression for c(x,t) is substituted into the left and right side boundary conditions. This yields:

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L^{3/2}} & \frac{-6\sqrt{5}}{L^{3/2}} & \frac{12\sqrt{7}}{L^{3/2}} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = 0 \tag{3.14}$$

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L^{3/2}} & \frac{+6\sqrt{5}}{L^{3/2}} & \frac{12\sqrt{7}}{L^{3/2}} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = 0 \tag{3.15}$$

The two boundary conditions specify that $\beta_2 = 0$ and $\beta_3 = \frac{-1}{2\sqrt{21}}\beta_1$ for all time. This gives a system of two dynamic equations and two algebraic constraints shown in Equation 3.16:

$$\begin{aligned} \dot{\beta_0} &= +\tfrac{1-t^+}{F\epsilon}\gamma_0 \\ \dot{\beta_1} &= -\tfrac{D}{\epsilon}\tfrac{10}{L^2}\beta_1 + \tfrac{1-t^+}{F\epsilon}\gamma_1 \\ \beta_2 &= 0 \\ \beta_3 &= \tfrac{-1}{2\sqrt{21}}\beta_1 \end{aligned} \tag{3.16}$$

Initial conditions ($\beta_n(t=0)$) and inputs ($\gamma_n(t)$) need to be specified for the Cartesian one domain diffusion system using the orthogonal projection method. This is done by applying Galerkin Projections to Equations 3.9 and 3.10. The initial β values and input γ values are:

$$\beta_n(0) = \int_0^L c(x,0)\phi_n(x)\mathrm{d}x, n = 0 \cdots 3 \tag{3.17}$$

$$\gamma_n(t) = \int_0^L J(x,t)\phi_n(x)\mathrm{d}x, n = 0 \cdots 3 \tag{3.18}$$

The MATLAB code for the fourth order Orthogonal Projection Method (OPM-4) is found in Appendix B.2.

3.1.3 Comparison

With the orthogonal projection method, pre-processing and post-processing are required, but the fourth order OPM runs faster than the FDM with moderate to large number of spatial nodes, seen in Tables 3.1 and 3.2. For the initial conditions shown in Figure 3.2, the FDM and OPM-4 solutions differ at $t = 0$, however they converge as time increases, shown in Figure 3.3. The initial conditions selected are a worse-case scenario, where the initial concentration is only c^0 continuous, while the Legendre Polynomials are smooth functions.

Table 3.1. Comparison of OPM-4 and FDM: 1,001 Time Steps

Number of Space Nodes	Run Time: OPM-4 [Seconds]	Run Time: FDM [Seconds]
101	0.31	0.53
201	0.33	0.72
501	0.36	3.99
1001	0.42	25.01

Table 3.2. Comparison of OPM-4 and FDM: 10,001 Time Steps

Number of Space Nodes	Run Time: OPM-4 [Seconds]	Run Time: FDM [Seconds]
101	2.48	0.72
201	2.59	1.10
501	2.81	7.70
1001	3.21	39.00

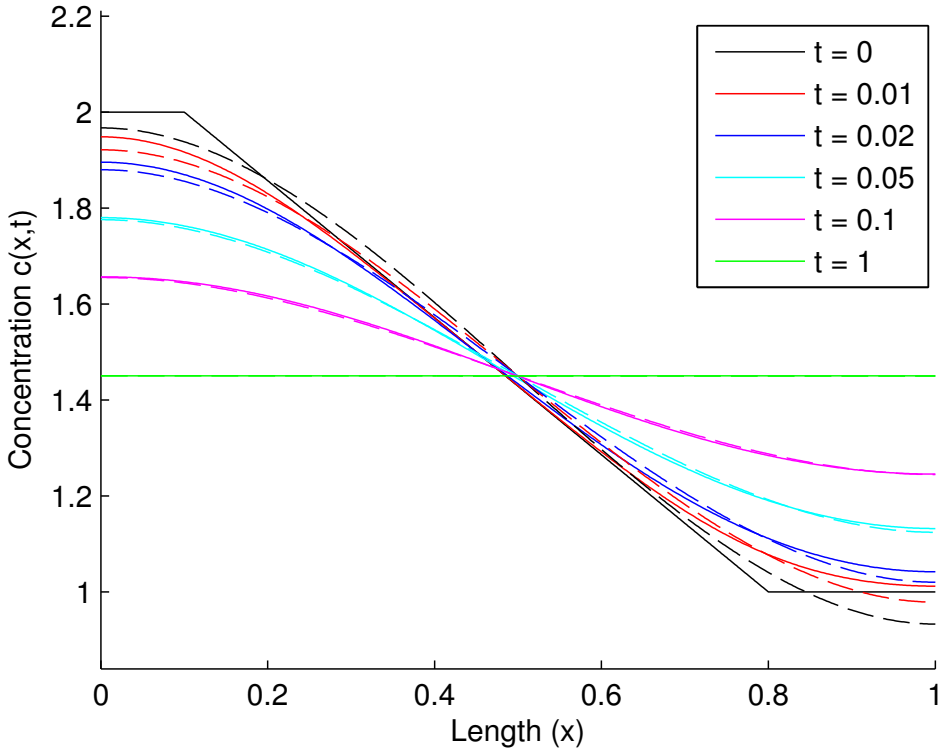

Figure 3.2. Finite Difference [Solid] vs. Fourth Order Orthogonal Projection [Dash]

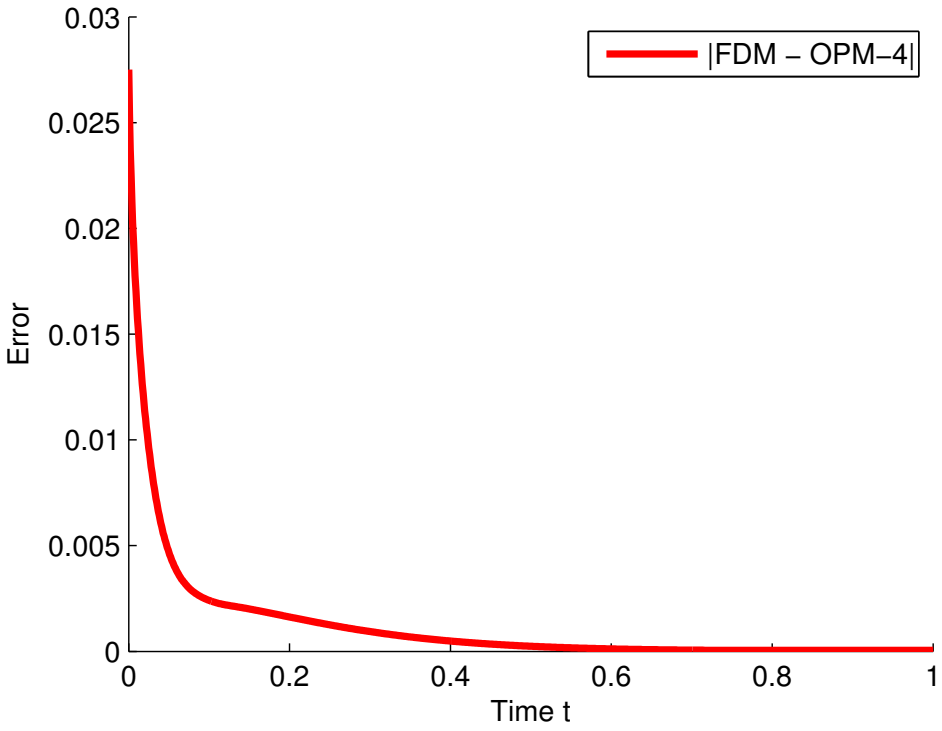

Figure 3.3. Error: FDM - OPM-4 [Red], 10,001 Time Steps, 1001 Space Nodes

3.2 Comparison of fourth order and fifth order Orthogonal Projection Methods

There is a natural tradeoff between simulation accuracy and simulation speed that arises from the number of Legendre polynomials used for each orthogonal projection. To determine this tradeoff, the linear, one domain diffusion problem is solved using both a fourth order and a fifth order Orthogonal Projection method. In Section 3.1.2 the fourth order solution (OPM-4) is discussed. In the next section the fifth order method (OPM-5) is briefly discussed.

3.2.1 Orthogonal Projection Method (Fifth Order)

The OPM-5 model is an extension of the previous fourth order method by adding $\phi_4(x)$ to both the concentration and input terms. This changes the A and B matrices to $5X5$:

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \\ \dot{\beta}_4 \end{bmatrix} = \frac{D}{\epsilon} \begin{bmatrix} 0 & 0 & \frac{12\sqrt{5}}{L^2} & 0 & \frac{40\sqrt{9}}{L^2} \\ 0 & 0 & 0 & \frac{20\sqrt{21}}{L^2} & 0 \\ 0 & 0 & 0 & 0 & \frac{28\sqrt{45}}{L^2} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \frac{1-t^+}{F\epsilon} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} \tag{3.19}$$

Solving the boundary conditions gives two algebraic relationships:

$$2\sqrt{3}\beta_1 + 12\sqrt{7}\beta_3 = 0 \tag{3.20}$$

$$6\sqrt{5}\beta_2 + 20\sqrt{9}\beta_4 = 0 \tag{3.21}$$

Subsituting 3.20 and 3.21, the OPM-5 becomes a system of five equations:

$$\begin{aligned} \dot{\beta_0} &= +\frac{1-t^+}{F\epsilon}\gamma_0 \\ \dot{\beta_1} &= -\frac{D}{\epsilon}\frac{10}{L^2}\beta_1 + \frac{1-t^+}{F\epsilon}\gamma_1 \\ \dot{\beta_2} &= -\frac{D}{\epsilon}\frac{42}{L^2}\beta_2 + \frac{1-t^+}{F\epsilon}\gamma_2 \\ \beta_3 &= \frac{-1}{2\sqrt{21}}\beta_1 \\ \beta_4 &= \frac{-30}{20\sqrt{45}}\beta_2 \end{aligned} \tag{3.22}$$

3.2.2 Comparison

To compare the accuracy of the simulation, two different initial condition scenarios are chosen, and both OPM-4 and OPM-5 are compared to the FDM from Section 3.1.1. The first initial concentration is the same c^0 continuous non symmetrical function as Figure 3.2, while the second initial condition is selected such that it is symmetrical around $x = \frac{1}{2}$, the midpoint of the domain. The results for the non symmetrical initial condition is discussed in Section 3.2.2.1, while the symmetrical initial condtion is discussed in Section 3.2.2.2. Tables 3.3 and 3.4 compares the run time of the three methods with $1,001$ and $10,001$ time steps for the first initial condition. There is an increase in computational run time by adding an extra Legendre Polynomial, however, the fifth order orthogonal projection method is nearly as fast as the fourth order, and is still far faster than the finite difference method when using a large number of spatial discretization nodes.

Table 3.3. Comparison of OPM-4 and OPM-5: 1,001 Time Steps

Number of Space Nodes	Run Time: OPM-4 [Seconds]	Run Time: OPM-5 [Seconds]
101	0.31	0.31
201	0.32	0.32
501	0.35	0.36
1001	0.38	0.40

Table 3.4. Comparison of OPM-4 and OPM-5: 10,001 Time Steps

Number of Space Nodes	Run Time: OPM-4 [Seconds]	Run Time: OPM-5 [Seconds]
101	2.38	2.92
201	2.52	3.13
501	2.77	3.33
1001	3.14	3.88

3.2.2.1 Non-Symmetrical Initial Condition

Figures 3.4 and 3.5 illustrates the improvement from adding an extra Legendre Polynomial. At $t = 0$ the error between FDM and OPM-5 is smaller than the error between FDM and OPM-4. This trend continues until approximately 0.1 seconds until the errors converge.

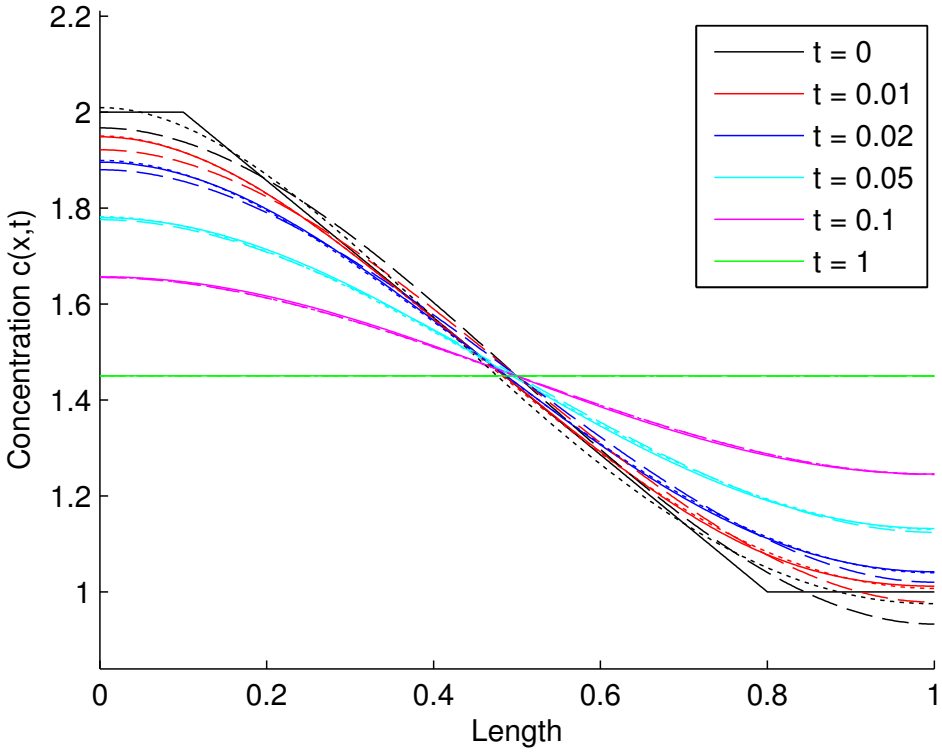

Figure 3.4. FDM [Solid] vs. OPM-4 [Dash] vs. OPM-5 [Dotted]: Initial Condition 1

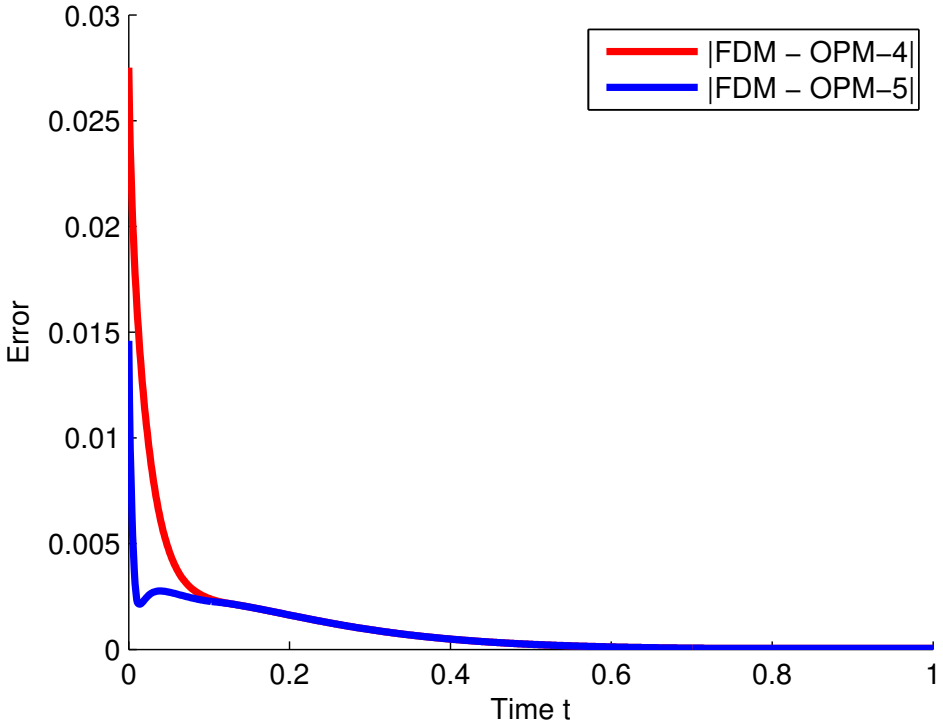

Figure 3.5. Error: FDM - OPM-4 [Red] and FDM - OPM-5 [Blue], 10,001 Time Steps, 1001 Space Nodes, Initial Condition 1

3.2.2.2 Symmetrical Initial Condition

Figures 3.6, 3.7, and 3.8 shows how for symmetrical initial conditions, the fourth order model breaks down due to $\beta_2 = 0 \ \forall t$. β_2 corresponds with the quadratic Legendre Polynomial. The only Legendre Polynomial that has dynamics is the linear, non-symmetric component, whereas in the fifth order model, both the linear and parabolic Legendre Polynomials ($\phi_1(x)$ and $\phi_2(x)$) are dynamic.

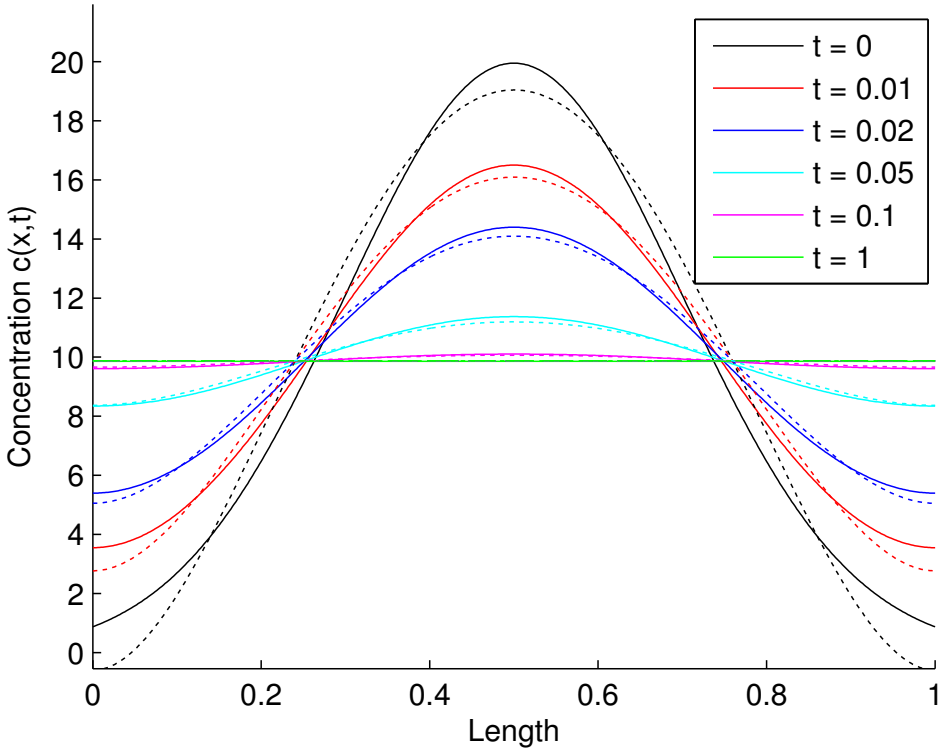

Figure 3.6. FDM [Solid] vs. OPM-4 [Dash] vs. OPM-5 [Dotted]: Initial Condition 2

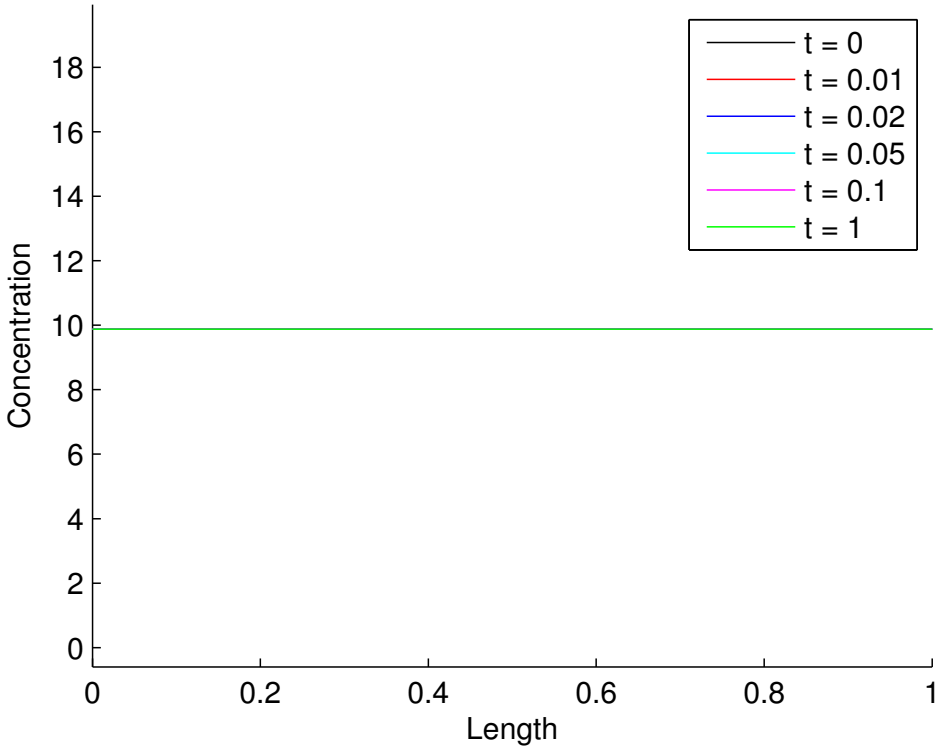

Figure 3.7. OPM-4: Initial Condition 2

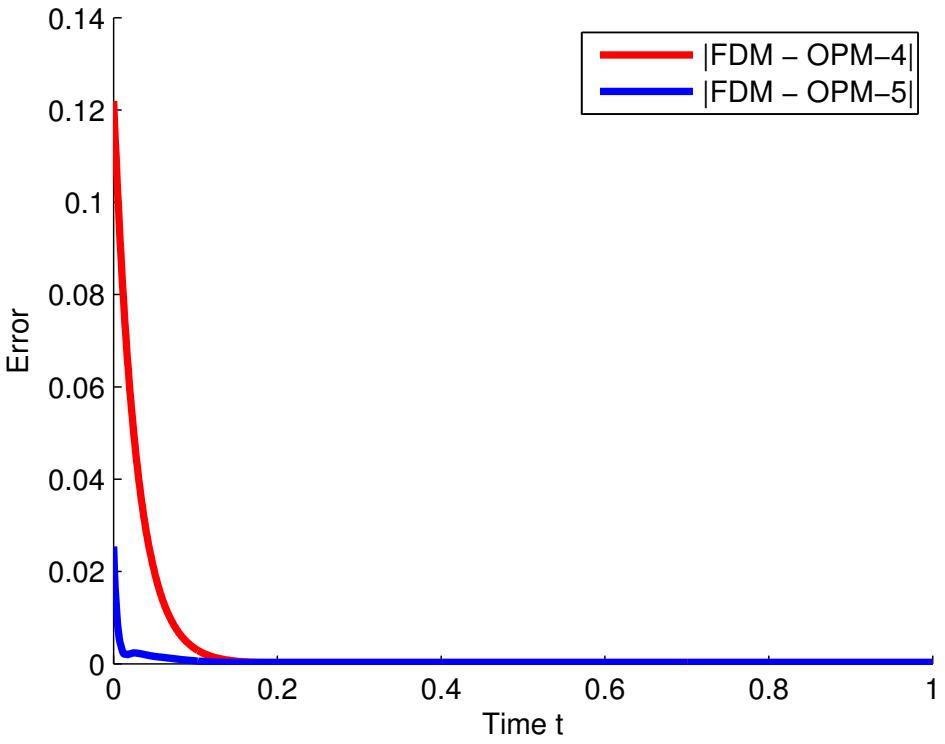

Figure 3.8. Error: FDM - OPM-4 [Red] and FDM - OPM-5 [Blue], 10,001 Time Steps, 1001 Space Nodes, Initial Condition 2

3.3 Comparison of Backward Solving Descriptor form to MATLAB lsim command

In Sections 3.1.1, 3.1.2 and 3.2.1, the MATLAB command lsim was used to solve the ordinary differential equations with respect to time. To incorporate the boundary conditions for the orthogonal projection methods, the left and right side boundary conditions were solved to give relationships between the $\beta(t)$ terms. Rather than explicitly solving the boundary conditions as done in previous work, the fifth order orthogonal projection method (OPM-5) was cast in descriptor form and solved using backward differencing with respect to time. The fifth order orthogonal projection solved in descriptor form (OPM-5-DF) is described in the next subsection.

3.3.1 Orthogonal Projection Method (Fifth Order - Descriptor Form)

The descriptor form of the orthogonal projection method changes the typical state space representation of the linear, one domain diffusion to the following form:

$$[E][\dot{\vec{\beta}}] = \frac{D}{\epsilon}[A][\vec{\beta}] + \frac{1-t^{+}}{F\epsilon}[B][\vec{\gamma}] \tag{3.23}$$

Instead of finding algebraic relationships using the boundary conditions as was done in equation 3.22, the descriptor form replaces the last two rows of equation 3.19 with the two boundary conditions, equations 3.20 and 3.21. The descriptor form of the fifth order Orthogonal projection method then becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \\ \dot{\beta}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{D}{\epsilon}\frac{12\sqrt{5}}{L^2} & 0 & \frac{D}{\epsilon}\frac{40\sqrt{9}}{L^2} \\ 0 & 0 & 0 & \frac{D}{\epsilon}\frac{20\sqrt{21}}{L^2} & 0 \\ 0 & 0 & 0 & 0 & \frac{D}{\epsilon}\frac{28\sqrt{45}}{L^2} \\ 0 & 2\sqrt{3} & 0 & 12\sqrt{7} & 0 \\ 0 & 0 & 6\sqrt{5} & 0 & 20\sqrt{9} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$
$$+\frac{1-t^+}{F\epsilon} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} \tag{3.24}$$

In the FDM, OPM-4 and OPM-5 methods, the lsim command was used in MATLAB. For the descriptor form, the solution is discretized temporally, and solved using a backward difference scheme, shown below in equation 3.25. For OPM-5-DF, k is the discrete time step, and δt is the time step size.

$$\begin{aligned} [E]\dot{\vec{\beta}} &\approx [E](\frac{\vec{\beta}_{k+1}-\vec{\beta}_k}{\delta t}) \\ [E](\frac{\vec{\beta}_{k+1}-\vec{\beta}_k}{\delta t}) &= [A]\vec{\beta}_{k+1} + [B]\vec{\gamma}_k \\ \vec{\beta}_{k+1} &= (\frac{[E]}{\delta t} - [A])^{-1}(\frac{[E]}{\delta t}\vec{\beta}_k + [B]\vec{\gamma}_k) \end{aligned} \tag{3.25}$$

Appendix B.4 contains the MATLAB code used to simulate the OPM-5-DF.

3.3.2 Comparison

To compare the descriptor form method to the MATLAB lsim command, the nonsymmetrical initial condition was used again to test the accuracy and the speed of the simulation. The result of this test is plotted in Figures 3.9 and 3.10. Figure 3.9 shows the full linear one domain system, while Figure 3.10 magnifies the results around the midpoint. Figure 3.11 plots the error between the FDM method and the OPM-5 and OPM-5-DF methods with respect to time.

The plots illustrates that both methods produce almost identical results, even at the initial time step. There are two benefits by using the descriptor form. First, the boundary condition

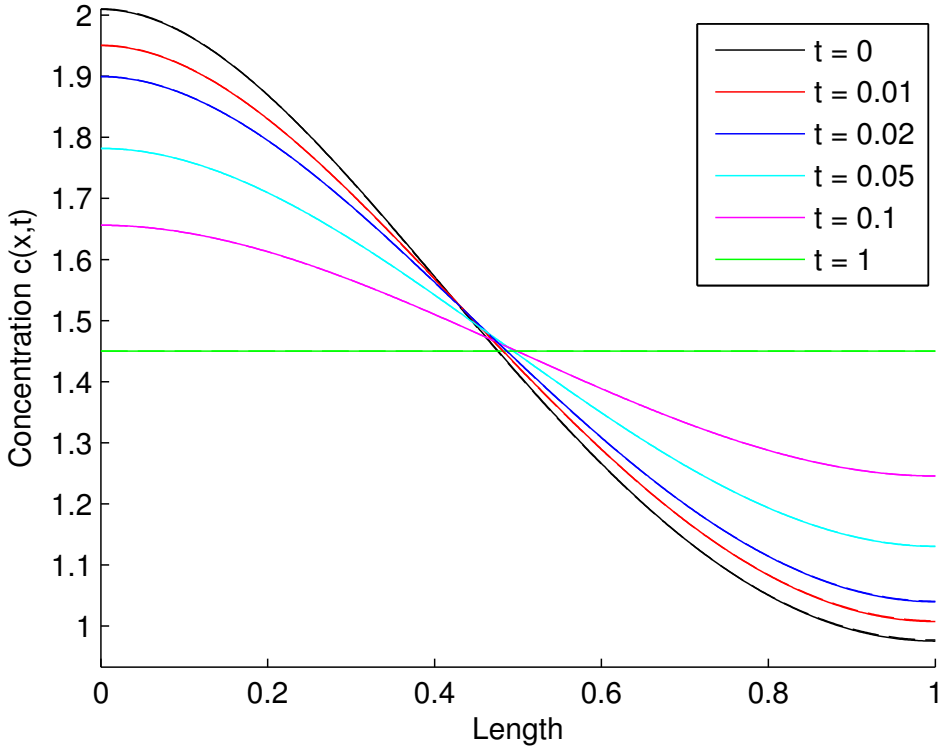

Figure 3.9. OPM-5 [Solid] vs. OPM-5-DF [Dash]: Initial Condition 1

Table 3.5. Comparison of OPM-5 and OPM-5-DF: 1,001 Time Steps

Number of Space Nodes	Run Time: OPM-5 [Seconds]	Run Time: OPM-5-DF [Seconds]
101	0.31	0.18
201	0.32	0.20
501	0.35	0.22
1001	0.40	0.25

Table 3.6. Comparison of OPM-5 and OPM-5-DF: 10,001 Time Steps

Number of Space Nodes	Run Time: OPM-5 [Seconds]	Run Time: OPM-5-DF [Seconds]
101	2.92	1.81
201	3.12	1.98
501	3.41	2.15
1001	3.85	2.56

substitution is simplified compared to the OPM-5 method. Second, there is a computational benefit, shown in Tables 3.5 and 3.6. The descriptor form is faster than the MATLAB lsim command for any number of spatial and time discretizations.

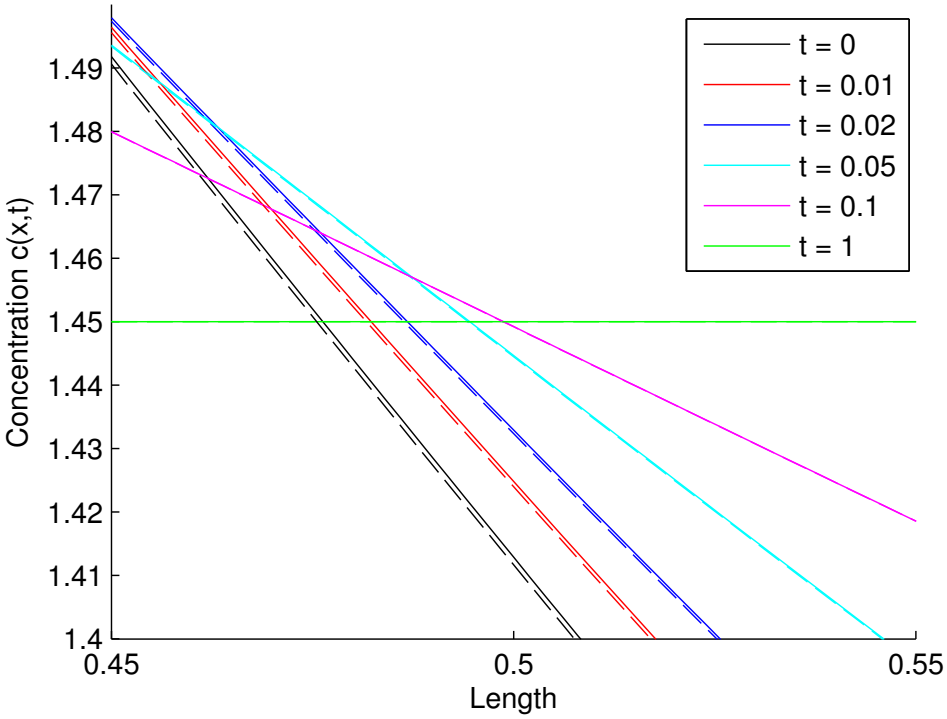

Figure 3.10. OPM-5 [Solid] vs. OPM-5-DF [Dash]: Initial Condition 1, Magnified

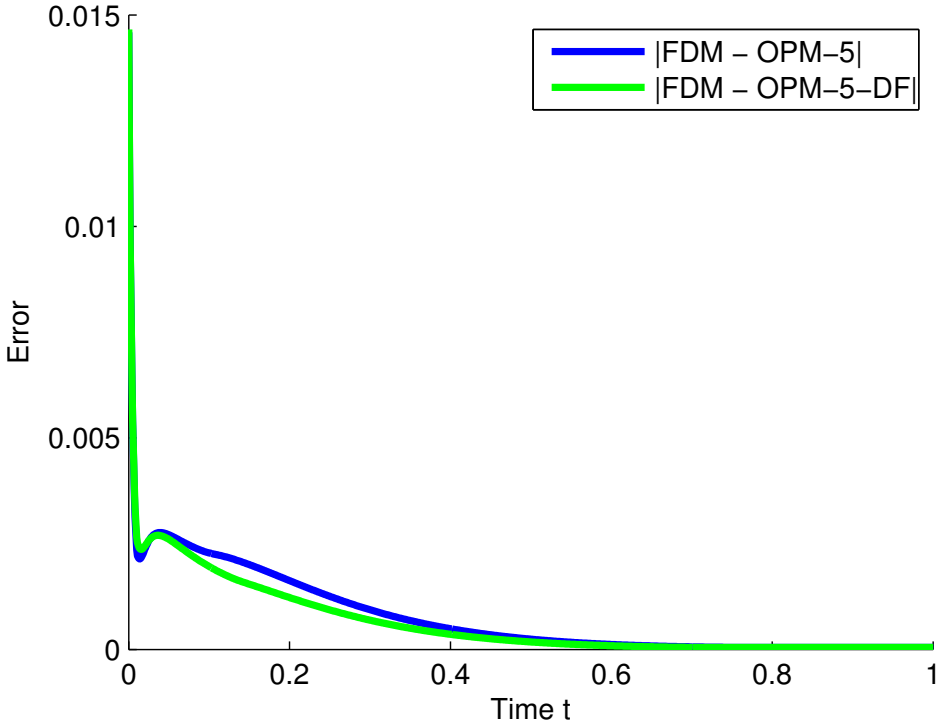

Figure 3.11. Error: FDM - OPM-5 [Blue] and FDM - OPM-5-DF [Green], 10,001 Time Steps, 1001 Space Nodes

Chapter 4

Doyle Fuller Newman Model

As briefly discussed in Chapter 1, the Doyle Fuller Newman model consists of five parts, with some acting only in the positive and negative electrode, while others act in the electrodes along with the separator. Table 4.1 lists the five components of the DFN model and identifies in which domain the component is necessary. Figure 4.1 shows the parts of the battery.

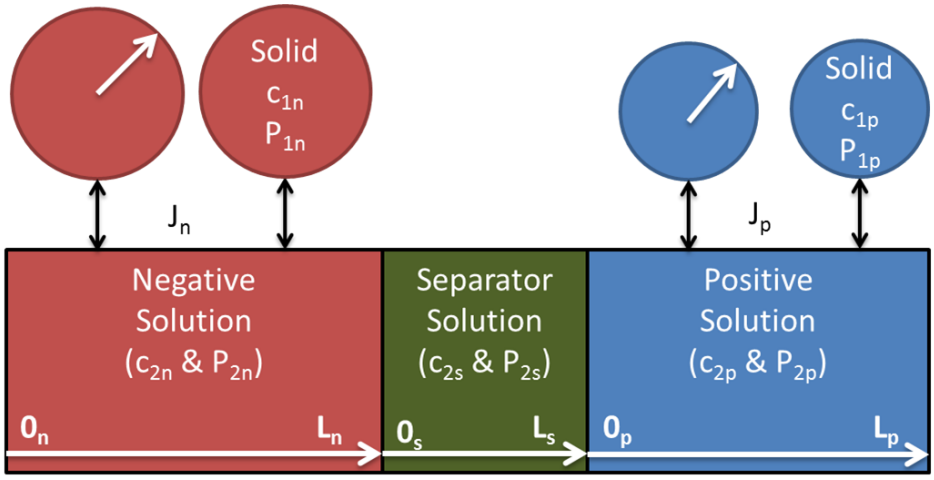

Figure 4.1. DFN Battery Model Schematic

Table 4.1. Components of the DFN model

Component	Domain: Negative (n), Separator (s), Positive (p)
Solid Phase Concentration (c_1)	n,p
Solution Phase Concentration (c_2)	n,s,p
Solid Phase Potential (P_1)	n,p
Solution Phase Potential (P_2)	n,s,p
Butler-Volmer Equation (J)	n,p

There are twelve orthogonal projections necessary to model the time and spatial varying components of the DFN model. The solid phase concentration has two orthogonal projections, one for the negative electrode and one for the positive electrode. The orthogonal projections are shown below in Equations 4.1 and 4.2.

$$\begin{aligned} c_{1,n}(r,x,t) &= \textstyle\sum_{i=0}^{4}\sum_{j=0}^{3}\tau_{(4i+j+1),n}(t)\phi_{i,n}(x)\omega_{2j,n}(r) \\ c_{1,n}(r,x,t) &= (\tau_{1,n}(t)\omega_0(r)+\cdots+\tau_{4,n}(t)\omega_6(r))\phi_0(x)+\cdots+ \\ &\quad (\tau_{17,n}(t)\omega_0(r)+\cdots+\tau_{20,n}(t)\omega_6(r))\phi_4(x) \end{aligned} \tag{4.1}$$

$$\begin{aligned} c_{1,p}(r,x,t) &= \textstyle\sum_{i=0}^{4}\sum_{j=0}^{3}\tau_{(4i+j+1),p}(t)\phi_{i,p}(x)\omega_{2j,p}(r) \\ c_{1,p}(r,x,t) &= (\tau_{1,p}(t)\omega_0(r)+\cdots+\tau_{4,p}(t)\omega_6(r))\phi_0(x)+\cdots+ \\ &\quad (\tau_{17,p}(t)\omega_0(r)+\cdots+\tau_{20,p}(t)\omega_6(r))\phi_4(x) \end{aligned} \tag{4.2}$$

It is important to note that the solid phase concentration is a function of the radial direction r, the Cartesian direction x, and time t. In the x direction the first five Cartesian Legendre Polynomials are used from Table 2.2. In the radial direction the first four even Radial Legendre Polynomials are used, shown in Table 2.3. The reason for this is discussed in Section 4.1.

For the solution phase concentration, three orthogonal projections are required for the three domains the solution concentration varies. These orthogonal projections are only functions of the x direction and time, and are shown in Equations 4.3 to 4.5. The governing equations and coupled boundary conditions associated with the c_2 dynamics are discussed in Section 4.2.

$$c_{2,n} = \beta_{1,n}(t)\phi_{0,n}(x)+\beta_{2,n}(t)\phi_{1,n}(x)+\beta_{3,n}(t)\phi_{2,n}(x)+\beta_{4,n}(t)\phi_{3,n}(x)+\beta_{5,n}(t)\phi_{4,n}(x) \tag{4.3}$$

$$c_{2,s} = \beta_{1,s}(t)\phi_{0,s}(x)+\beta_{2,s}(t)\phi_{1,s}(x)+\beta_{3,s}(t)\phi_{2,s}(x)+\beta_{4,s}(t)\phi_{3,s}(x)+\beta_{5,s}(t)\phi_{4,s}(x) \tag{4.4}$$

$$c_{2,p} = \beta_{1,p}(t)\phi_{0,p}(x)+\beta_{2,p}(t)\phi_{1,p}(x)+\beta_{3,p}(t)\phi_{2,p}(x)+\beta_{4,p}(t)\phi_{3,p}(x)+\beta_{5,p}(t)\phi_{4,p}(x) \tag{4.5}$$

The solid phase potential is represented only in the negative and positive electrodes, thus only two orthogonal projections are necessary. Unlike the solid phase concentration, the solid phase potential does not have any dependency on the radial direction. The orthogonal projections for $P_{1,n}$ and $P_{1,p}$ are shown in Equations 4.6 and 4.7 respectively.

$$P_{1,n} = \zeta_{1,n}(t)\phi_{0,n}(x)+\zeta_{2,n}(t)\phi_{1,n}(x)+\zeta_{3,n}(t)\phi_{2,n}(x)+\zeta_{4,n}(t)\phi_{3,n}(x)+\zeta_{5,n}(t)\phi_{4,n}(x) \tag{4.6}$$

$$P_{1,p} = \zeta_{1,p}(t)\phi_{0,p}(x)+\zeta_{2,p}(t)\phi_{1,p}(x)+\zeta_{3,p}(t)\phi_{2,p}(x)+\zeta_{4,p}(t)\phi_{3,p}(x)+\zeta_{5,p}(t)\phi_{4,p}(x) \tag{4.7}$$

Similar to the solution phase concentration, the solution phase potential acts in all three

domains, thus having three orthogonal projections, one for each domain. These orthogonal projections are shown below in Equations 4.8, 4.9, and 4.10 for the negative, separator, and positive solution potentials.

$$P_{2,n} = \delta_{1,n}(t)\phi_{0,n}(x)+\delta_{2,n}(t)\phi_{1,n}(x)+\delta_{3,n}(t)\phi_{2,n}(x)+\delta_{4,n}(t)\phi_{3,n}(x)+\delta_{5,n}(t)\phi_{4,n}(x) \quad (4.8)$$

$$P_{2,s} = \delta_{1,s}(t)\phi_{0,s}(x) + \delta_{2,s}(t)\phi_{1,s}(x) + \delta_{3,s}(t)\phi_{2,s}(x) + \delta_{4,s}(t)\phi_{3,s}(x) + \delta_{5,s}(t)\phi_{4,s}(x) \quad (4.9)$$

$$P_{2,p} = \delta_{1,p}(t)\phi_{0,p}(x)+\delta_{2,p}(t)\phi_{1,p}(x)+\delta_{3,p}(t)\phi_{2,p}(x)+\delta_{4,p}(t)\phi_{3,p}(x)+\delta_{5,p}(t)\phi_{4,p}(x) \quad (4.10)$$

The last component of the DFN model, the intercalation current governed by the nonlinear Butler-Volmer equation, requires two orthogonal projections, one for the negative electrode and one for the positive electrode. The orthogonal projections representing J_n and J_p are shown in Equations 4.11 and 4.12.

$$J_n = \gamma_{1,n}(t)\phi_{0,n}(x)+\gamma_{2,n}(t)\phi_{1,n}(x)+\gamma_{3,n}(t)\phi_{2,n}(x)+\gamma_{4,n}(t)\phi_{3,n}(x)+\gamma_{5,n}(t)\phi_{4,n}(x) \quad (4.11)$$

$$J_p = \gamma_{1,p}(t)\phi_{0,p}(x) + \gamma_{2,p}(t)\phi_{1,p}(x) + \gamma_{3,p}(t)\phi_{2,p}(x) + \gamma_{4,p}(t)\phi_{3,p}(x) + \gamma_{5,p}(t)\phi_{4,p}(x) \quad (4.12)$$

Equations 4.1 to 4.12 give an overall total of 90 state variables. The number of state variables for each component of the DFN model is shown in Table 4.2.

Table 4.2. State Variables of the DFN model

Component	Domain	Number of States
Solid Phase Concentration	Negative	20
	Positive	20
Solution Phase Concentration	Negative	5
	Separator	5
	Positive	5
Solid Phase Potential	Negative	5
	Positive	5
Solution Phase Potential	Negative	5
	Separator	5
	Positive	5
Butler-Volmer Equation	Negative	5
	Positive	5
Total	**12**	**90**

The following five subsections discuss the methods used to solve each of the five components. It is important to note that all parts of the DFN model are solved using descriptor form, which

allows for the assembly of the parts into a complete model, discussed in Section 4.6.

4.1 Solid Phase Concentration

The solid phase concentration acts only in the negative and positive electrodes. However, the governing partial differential equation the concentration is a function of the radial direction (r), the Cartesian direction (x), and time (t). Equations 4.13 - 4.18 shows the solid phase concentration PDEs and associated boundary conditions for both electrodes.

Negative Electrode:

$$\frac{\partial c_{1,n}(r,x,t)}{\partial t} = \frac{d_{1,n}}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial c_{1,n}(r,x,t)}{\partial r}) \tag{4.13}$$

$$\frac{\partial c_{1,n}(r,x,t)}{\partial r} = 0, \quad r = 0 \tag{4.14}$$

$$\frac{\partial c_{1,n}(r,x,t)}{\partial r} = \frac{-J_n}{d_{1,n}a_n F}, \quad r = R_n \tag{4.15}$$

Positive Electrode:

$$\frac{\partial c_{1,p}(r,x,t)}{\partial t} = \frac{d_{1,p}}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial c_{1,p}(r,x,t)}{\partial r}) \tag{4.16}$$

$$\frac{\partial c_{1,p}(r,x,t)}{\partial r} = 0, \quad r = 0 \tag{4.17}$$

$$\frac{\partial c_{1,p}(r,x,t)}{\partial r} = \frac{-J_p}{d_{1,p}a_p F}, \quad r = R_p \tag{4.18}$$

It is important to note that for the radial component only the first four even Legendre Polynomials are used. The reason for this is due to the fact the center boundary conditions require zero slope at $r = 0$. Figures 2.4 and 2.5 shows that at $r = 0$ the slope is automatically satisfied for the even Legendre Polynomials. In the x direction, the first five Cartesian Legendre Polynomials are used, with no restriction on being even or odd. Appendix C.1 shows the steps required to solve the solid phase concentration using orthogonal projections.

4.2 Solution Phase Concentration

The solution phase concentration is a three domain coupled system, consisting of $c_{2,n}$, $c_{2,s}$, and $c_{2,p}$. Equations 4.19, 4.20, and 4.21 show the governing equations for negative, separator and positive electrode domains respectively. Equations 4.22 to 4.27 show the six coupled boundary conditions.

$$\epsilon_n \frac{\partial c_{2,n}(x,t)}{\partial t} = d_{2,n} \frac{\partial^2 c_{2,n}(x,t)}{\partial x^2} + \frac{1-{t_n}^+}{F} J_n(x,t) \quad 0_n \leq x \leq L_n \tag{4.19}$$

$$\epsilon_s \frac{\partial c_{2,s}(x,t)}{\partial t} = d_{2,s} \frac{\partial^2 c_{2,s}(x,t)}{\partial x^2} + 0 \quad 0_s \leq x \leq L_s \tag{4.20}$$

$$\epsilon_p \frac{\partial c_{2,p}(x,t)}{\partial t} = d_{2,p} \frac{\partial^2 c_{2,p}(x,t)}{\partial x^2} + \frac{1-{t_p}^+}{F} J_p(x,t) \quad 0_p \leq x \leq L_p \tag{4.21}$$

$$\frac{\partial c_{2,n}(x,t)}{\partial x} = 0 \quad x = 0_n \tag{4.22}$$

$$c_{2,n}(L_n,t) - c_{2,s}(0_s,t) = 0 \quad x = L_n = 0_s \tag{4.23}$$

$$d_{2,n} \frac{\partial c_{2,n}(x,t)}{\partial x} - d_{2,s} \frac{\partial c_{2,s}(x,t)}{\partial x} = 0 \quad x = L_n = 0_s \tag{4.24}$$

$$d_{2,s} \frac{\partial c_{2,s}(x,t)}{\partial x} - d_{2,p} \frac{\partial c_{2,p}(x,t)}{\partial x} = 0 \quad x = L_s = 0_p \tag{4.25}$$

$$c_{2,s}(L_s,t) - c_{2,p}(0_p,t) = 0 \quad x = L_s = 0_p \tag{4.26}$$

$$\frac{\partial c_{2,p}(x,t)}{\partial x} = 0 \quad x = L_p \tag{4.27}$$

The coupled boundary conditions makes explicitly solving the boundary conditions to determine algebraic constraints difficult, thus the descriptor form was important to incorporate

the boundary conditions into the state space matrices. The orthogonal projections used for the solution phase concentration for the negative electrode, separator, and positive electrode are shown in Equations 4.3 to 4.5. The method to solve the solution phase concentration is shown in Appendix C.2.

4.3 Solid Phase Potential

The solid phase potential (P_1) has only derivatives with respect to x, and acts in the negative electrode and the positive electrode. The governing equations and associated boundary conditions are shown in Equations 4.28 to 4.33.

Negative Electrode:

$$\sigma_n^{eff} \frac{\partial^2 P_{1,n}(x,t)}{\partial x} - J_n(x,t) = 0 \tag{4.28}$$

$$P_{1,n}(x,t) = 0 \quad x = 0_n \tag{4.29}$$

$$\frac{\partial P_{1,n}(x,t)}{\partial x} = 0 \quad x = L_n = 0_s \tag{4.30}$$

Positive Electrode:

$$\sigma_p^{eff} \frac{\partial^2 P_{1,p}(x,t)}{\partial x} - J_p(x,t) = 0 \tag{4.31}$$

$$\frac{\partial P_{1,p}(x,t)}{\partial x} = 0 \quad x = L_s = 0_p \tag{4.32}$$

$$\frac{\partial P_{1,p}(x,t)}{\partial x} = \frac{i_{app}}{Area * \sigma_p^{eff}} \quad x = L_p \tag{4.33}$$

The first boundary condition in the negative electrode, Equation 4.29, sets the far left boundary to be the ground. The far right boundary condition in the positive electrode, Equation 4.33, gives the current input to the battery cell, i_{app}. Appendix C.3 contains the calculations used to solve the solid phase potential using orthogonal projections.

4.4 Solution Phase Potential

The solution phase potential is a nonlinear, coupled, PDE that acts in the negative electrode, separator and positive electrode. The governing equations for each domain are shown below in Equations 4.34 to 4.36. The nonlinear, coupled, boundary conditions are shown in Equations 4.37 to 4.42. The last term in the three governing equations and in the third and fourth boundary conditions comes from the concentration polarization effect.

$$\kappa_n^{eff}(\frac{\partial^2 P_{2,n}(x,t)}{\partial x^2}) + J_n(x,t) + \kappa_{D,n}(\frac{\partial^2}{\partial x^2}\ln(c_{2,n}(x,t))) = 0 \tag{4.34}$$

$$\kappa_s^{eff}(\frac{\partial^2 P_{2,s}(x,t)}{\partial x^2}) + 0 + \kappa_{D,s}(\frac{\partial^2}{\partial x^2}\ln(c_{2,s}(x,t))) = 0 \tag{4.35}$$

$$\kappa_p^{eff}(\frac{\partial^2 P_{2,p}(x,t)}{\partial x^2}) + J_p(x,t) + \kappa_{D,p}(\frac{\partial^2}{\partial x^2}\ln(c_{2,p}(x,t))) = 0 \tag{4.36}$$

$$\frac{\partial P_{2,n}(x,t)}{\partial x} = 0 \quad x = 0_n \tag{4.37}$$

$$P_{2,n}(L_n,t) - P_{2,s}(0_s,t) = 0 \quad x = L_n = 0_s \tag{4.38}$$

$$\kappa_n^{eff}\frac{\partial P_{2,n}(L_n,t)}{\partial x} - \kappa_s^{eff}\frac{\partial P_{2,s}(0_s,t)}{\partial x} + \kappa_{D,n}\frac{\partial}{\partial x}\ln(c_{2,n}(L_n,t)) - \kappa_{D,s}\frac{\partial}{\partial x}\ln(c_{2,s}(0_s,t)) = 0 \quad x = L_n = 0_s \tag{4.39}$$

$$\kappa_s^{eff}\frac{\partial P_{2,s}(L_s,t)}{\partial x} - \kappa_p^{eff}\frac{\partial P_{2,p}(0_p,t)}{\partial x} + \kappa_{D,s}\frac{\partial}{\partial x}\ln(c_{2,s}(L_s,t)) - \kappa_{D,p}\frac{\partial}{\partial x}\ln(c_{2,p}(0_p,t)) = 0 \quad x = L_s = 0_p \tag{4.40}$$

$$P_{2,s}(L_s,t) - P_{2,p}(0_p,t) = 0 \quad x = L_s = 0_p \tag{4.41}$$

$$\frac{\partial P_{2,p}(x,t)}{\partial x} = 0 \quad x = L_p \tag{4.42}$$

The nonlinearity in the solution potential is due to the dependence on the solution phase concentration. To cast the state equations in the descriptor form, the nonlinear components of the model are quasi-linearized, following the work of Forman et al. [6]. The calculations for the solution phase potential orthogonal projection method and the quasi-linearization of the nonlinear equations are shown in Appendix C.4.

4.5 Butler-Volmer Equation

The Butler-Volmer equation is a nonlinear algebraic constraint that relates the intercalation current density ($J(x,t)$) to the potentials in the solid and solution phase. The governing equations in the negative and positive electrodes are:

$$\begin{gathered} a_n i_{0,n}[\exp(\frac{\alpha_{a,n}F}{RT}\eta_n) - \exp(-\frac{\alpha_{c,n}F}{RT}\eta_n)] - J_n = 0 \\ i_{0,n} = k_n(c_{1,n}^{max} - c_{1,n}^s)^{\alpha_{a,n}}(c_{1,n}^s)^{\alpha_{c,n}}(c_{2,n})^{\alpha_{a,n}} \\ \eta_n = P_{1,n} - P_{2,n} - u_{nref} \end{gathered} \tag{4.43}$$

$$\begin{gathered} a_p i_{0,p}[\exp(\frac{\alpha_{a,p}F}{RT}\eta_p) - \exp(-\frac{\alpha_{c,p}F}{RT}\eta_p)] - J_p = 0 \\ i_{0,p} = k_p(c_{1,p}^{max} - c_{1,p}^s)^{\alpha_{a,p}}(c_{1,p}^s)^{\alpha_{c,p}}(c_{2,p})^{\alpha_{a,p}} \\ \eta_p = P_{1,p} - P_{2,p} - u_{pref} \end{gathered} \tag{4.44}$$

Where $c_{1,n/p}^s$ is equal to $c_{1,n/p}(R_{n/p},t)$, the surface concentration on the spherical particle. There are no boundary conditions, because the Butler-Volmer equation is not a differential equation. The results of the Butler-Volmer quasi-linearization and calculations are shown in Appendix C.5.

4.6 DFN Model Assembly

Sections 4.1 to 4.5 discuss the components of the DFN model, however for the model to run, the components need to be assembled into one final descriptor form, shown in Equation 4.45.

$$[E]_{90X90}[\dot{\vec{z}}] = [A]_{90X90}[\vec{z}] + [B]_{90X1}[u] + [G]_{90X1} \tag{4.45}$$

The state vector, $\vec{z}$ contains the unknown functions of time from the orthogonal projections discussed in the previous sections. They are arranged in the following order:

$$[\vec{z}]_{90X1} = \begin{bmatrix} [\vec{\tau}n]_{20X1} \\ [\vec{\tau}p]_{20X1} \\ [\vec{\beta}n]_{5X1} \\ [\vec{\beta}s]_{5X1} \\ [\vec{\beta}p]_{5X1} \\ [\vec{\zeta}n]_{5X1} \\ [\vec{\zeta}p]_{5X1} \\ [\vec{\delta}n]_{5X1} \\ [\vec{\delta}s]_{5X1} \\ [\vec{\delta}p]_{5X1} \\ [\vec{\gamma}n]_{5X1} \\ [\vec{\gamma}p]_{5X1} \end{bmatrix} \tag{4.46}$$

There is only one input into the model, u which is the applied current in and out of the battery. The E matrix comprised of either 0 or 1 along the diagonal, and zero on the off-diagonal terms. The A matrix is made up of the calculated values from Appendix C. The B matrix is a vector with only one nonzero value, which comes from the solid phase potential boundary conditions. The G vector comes from the linearization of the nonlinear components of the DFN model.

The MATLAB code for the Doyle Fuller Newman model can be found in Appendix D.

Chapter 5

Results

To test the computational speed and validity of the model, different simulation tests are performed for different current trajectories and different amounts of discretization steps in space and time. The first goal of these simulation studies is to test how quickly the simulation studies run. Ideally, a control oriented battery model should be able to run faster than real time on a personal computer. To test the speed, the same current trajectory is applied, while the sampling time and the number of discretization steps are varied in the Cartesian and radial directions. The results of the computational speed tests are discussed in Section 5.1.

The second goal of these simulation studies is to determine the validity of the battery model. Different current trajectory inputs are created to verify that the battery model predicts reasonable voltage outputs. These simulation tests are discussed in Section 5.2.

It is important to note that the parameters used for the DFN model discussed in this thesis are from Forman et al [7]. The battery cells used in the parameter identification tests are A123 26650 cells with a nominal capacity of 2.5 Ampere-Hours (Ah) [1]. For the simulation studies discussed in this chapter, a Toshiba Satellite P750 laptop is used, which has 8.00 GB of RAM and a 2.20 GHz Intel i7 processor.

5.1 DFN Model Computational Speed

A constant current trajectory of -0.5 Amperes (where a negative value implies discharge of the battery cell) is simulated for various discretization values in time and space. Table 5.1 shows the results for 1000 seconds of battery data simulated at 1 Hertz (1001 time steps). Table 5.2 shows the results for 1000 seconds of battery data simulated at 10 Hertz (10001 time steps). To simplify the tests, the number of negative, separator, and positive x discretization points are kept equal, and the number of negative and positive r discretization points are kept equal as well.

The results of the computational speed tests shows that the simulation speed is not heavily dependent on the number of discretization steps in space, however it is highly dependent on the number of temporal discretization steps. Tables 5.1 and 5.2 show that increasing the sampling frequency by an order of magnitude results in an increase in the run time by approximately an

order of magnitude. Both tables also show that there is a reduction in run time when the number of x discretization points and r discretization points are equal.

A comparison can be made to the Cartesian one domain diffusion test discussed in Section 3.1.1. For 1,001 time steps and 1,001 discretization points in the one Cartesian domain, the run time is 25.01 seconds (Table 3.1). For the complete Doyle Fuller Newman model that has 1,001 time steps and 501 discretization points in five domains, the run time is 38.27 seconds. The DFN model solved using orthogonal projections in descriptor form is only slightly slower than the highly simplified one domain diffusion problem solved using finite differences.

Table 5.1. DFN Model Run Time: 1,001 Time Steps

x Discretization Points	r Discretization Points	Run Time [Seconds]
101	101	41.59
501	101	42.47
101	501	37.25
501	501	38.27

Table 5.2. DFN Model Run Time: 10,001 Time Steps

x Discretization Points	r Discretization Points	Run Time [Seconds]
101	101	396.40
501	101	425.30
101	501	436.72
501	501	405.94

5.2 DFN Model Computational Validity

To determine the validity of the battery model three types of tests are performed. The first test, described in Section 5.2.1, initializes the battery simulation to approximately 90% State of Charge (SoC) and rests the battery for one hour. In Section 5.2.2 the simulated battery starts at approximately 90% State of Charge (SoC) and then is discharge to nearly 0% SoC. These tests are repeated for .25 A, 2.5 A, and 6.25 A. These current values correspond to 0.1 C, 1.0 C and 2.5 C, where C represents the C rate of the battery, a normalized value of current based on the capacity of the battery. The third and final type of simulations performed, the pulse tests, are discussed in Section 5.2.3. The battery is discharged from an initial value of 90% SoC to approximately 50% SoC, and then is pulsed charged and discharged at 0.5 C, 1.5 C, 3.0 C and 5.0 C for either 60 seconds or 300 seconds.

For all of the validity tests, the battery model has 101 discretization points in the negative, separator and positive Cartesian directions, and 101 discretization points in the radial negative and positive directions. The sampling frequency is 1 Hz for all tests unless otherwise noted.

5.2.1 Steady State Test

The first test is to verify that the battery is initialized to a steady state condition. The battery is initialized to 3.3 V, however it quickly relaxes to approximately 3.42 V. The concentration of lithium ions in the negative domain (Anode) decays quickly, however the positive domain (Cathode) requires approximately 300 seconds to decay to steady state. Figures 5.1 and 5.2 show that the battery is nearly initialized to steady state conditions, and after 300 seconds the battery relaxes to steady state.

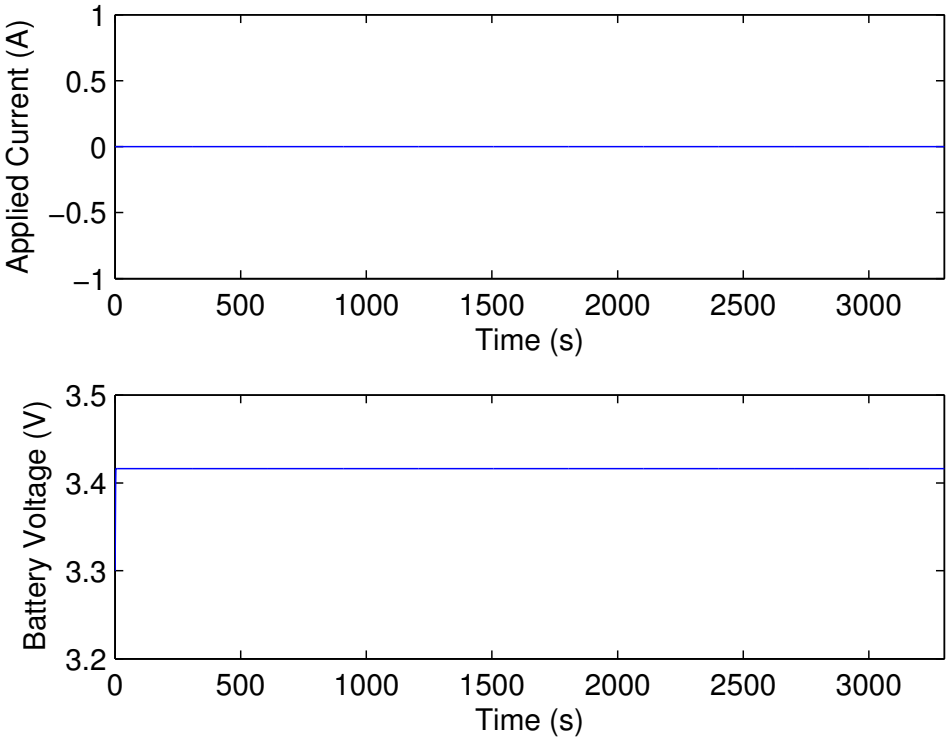

Figure 5.1. Steady State: Current and Voltage

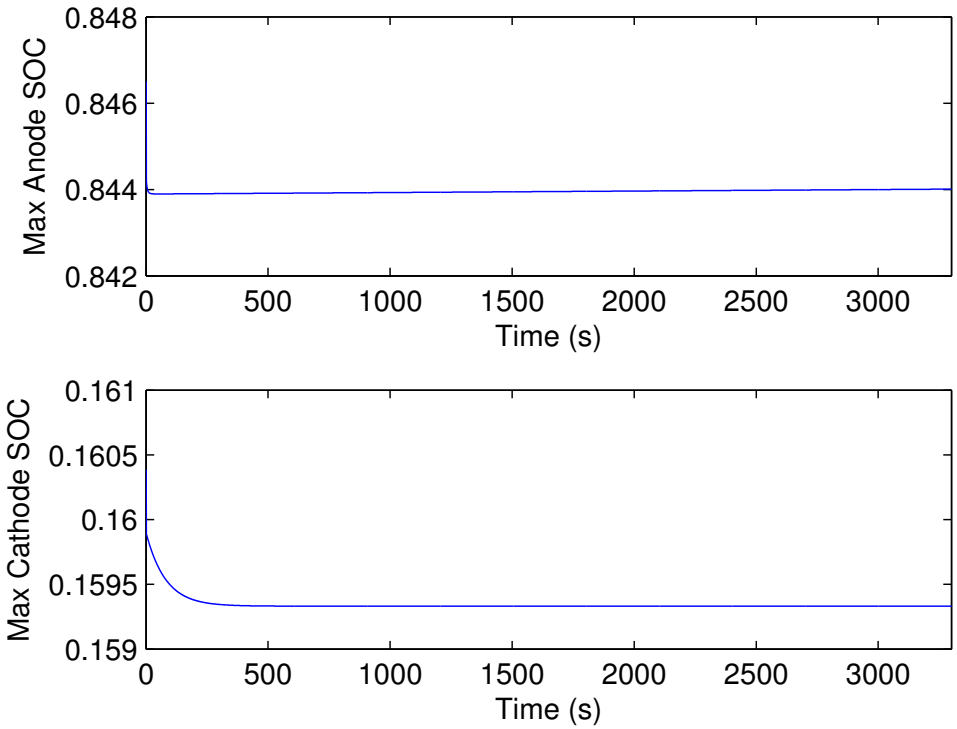

Figure 5.2. Steady State: State of Charge

5.2.2 Continuous Discharge Tests

5.2.2.1 0.1C Discharge

A 0.1C discharge current is applied for 30,000 seconds (8 hours and 20 minutes) at a sampling frequency of 0.1 Hz. The simulation results indicate that nearly all of the lithium ions diffuse from the anode to the cathode (Figure 5.4) during the course of the simulation. The voltage drops significantly at approximately 25,000 seconds, and drops below 2 V around 29,000 seconds (Figure 5.3). The shape of the voltage curve is similar to the A123 data sheet [1], where at the end of a discharge there is a sharp decrease in battery voltage, however in these simulation results the battery is discharged below 2 V while the manufacturer has a lower limit of 2 V.

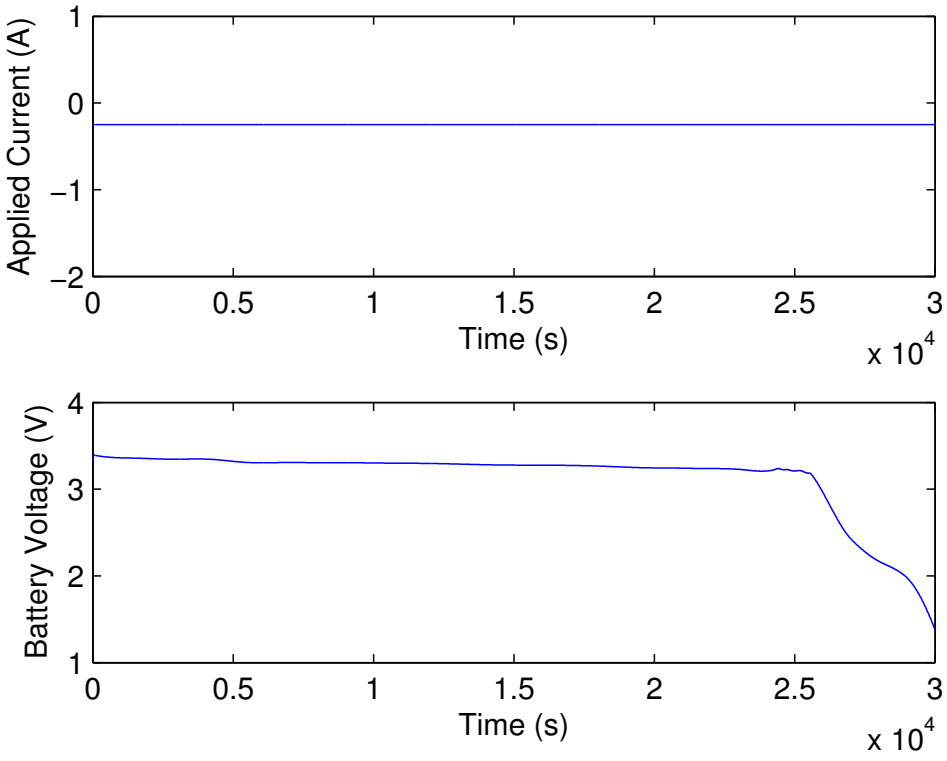

Figure 5.3. 0.1 C Discharge: Current and Voltage

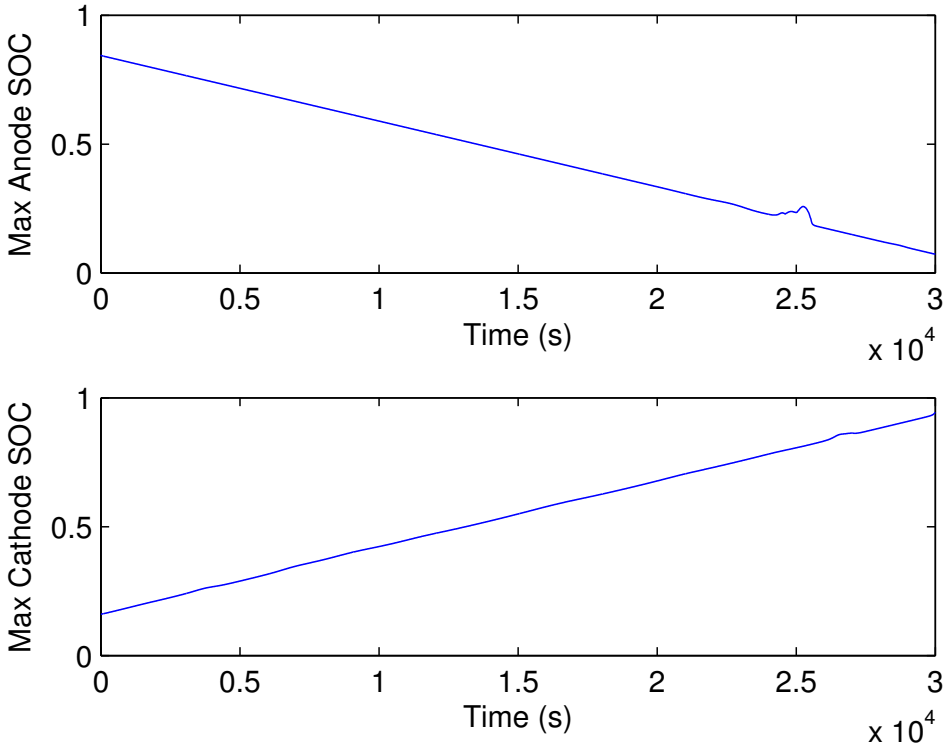

Figure 5.4. 0.1 C Discharge: State of Charge

5.2.2.2 1.0C Discharge

Similar to the previous section, a 1.0 C (2.5 A) discharge is applied for 3000 seconds (50 minutes) at a sampling frequency of 1 Hz. Figure 5.5 shows that at approximately 2500 seconds (42 minutes) the battery voltage drops drastically, and at 2825 seconds (47 minutes) the battery reaches 2 V. The SoC for the anode and cathode are 19.6% and 82.8% at 2500 seconds, and 11.5% and 88.9% at 2825 seconds. It is important to note that the percentages will not add to 100% due to the fact the maximum SoC is found across the anode (negative domain) and cathode (positive domain). Comparing Figures 5.4 and 5.6, the kink in the anode SoC plot is present in both, however the kink is smaller in Figure 5.6.

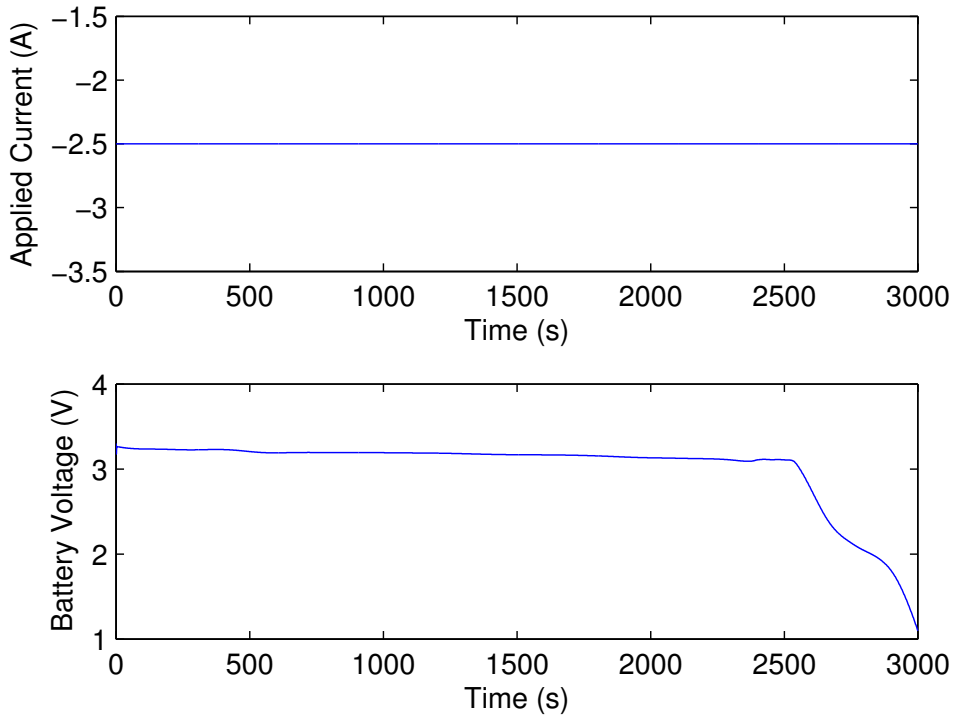

Figure 5.5. 1.0 C Discharge: Current and Voltage

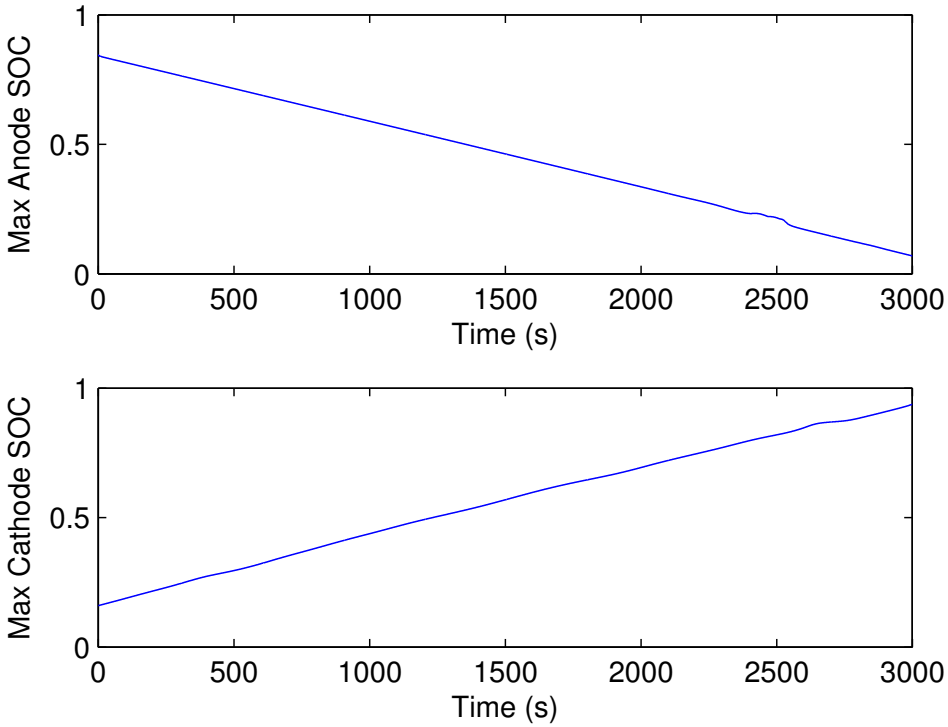

Figure 5.6. 1.0 C Discharge: State of Charge

5.2.2.3 2.5C Discharge

The 2.5 C discharge plots are very similar to the 0.1 C (Section 5.2.2.1) and 1.0 C (Section 5.2.2.2) discharge plots previously discussed. The simulation is 1200 seconds long, sampled at 5 Hz. The knee of the voltage curve occurs at approximately 1000 seconds, and the 2 V limit is reached at 1100 seconds. The SoC plots (Figure 5.8) are similar to the previous two sections as well, with the anode SoC having a kink at the same time as the knee of the voltage curve.

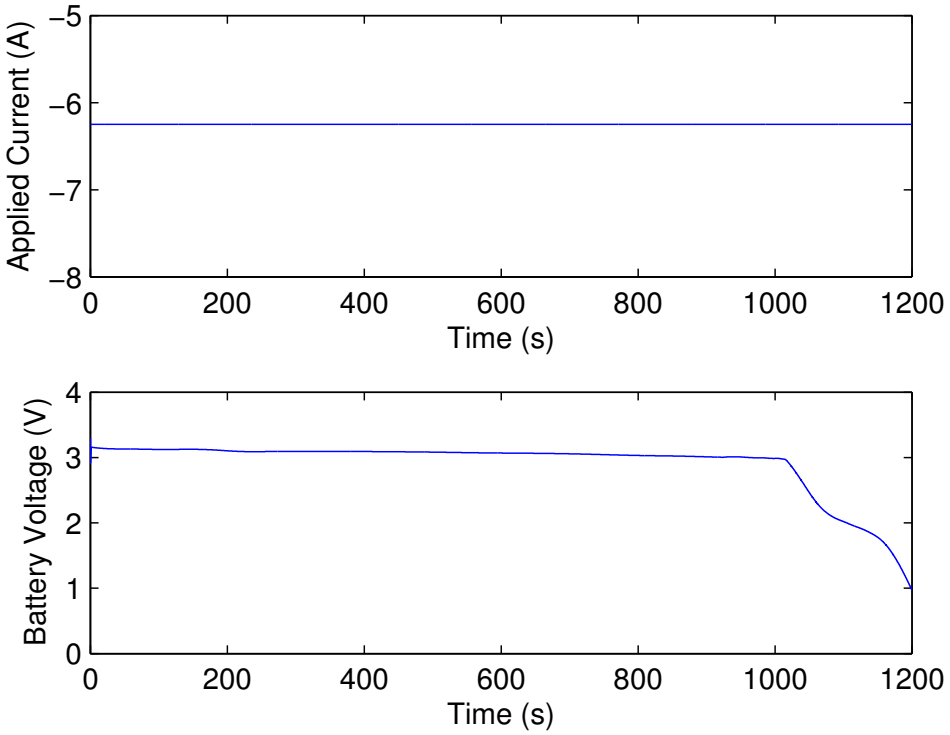

Figure 5.7. 2.5 C Discharge: Current and Voltage

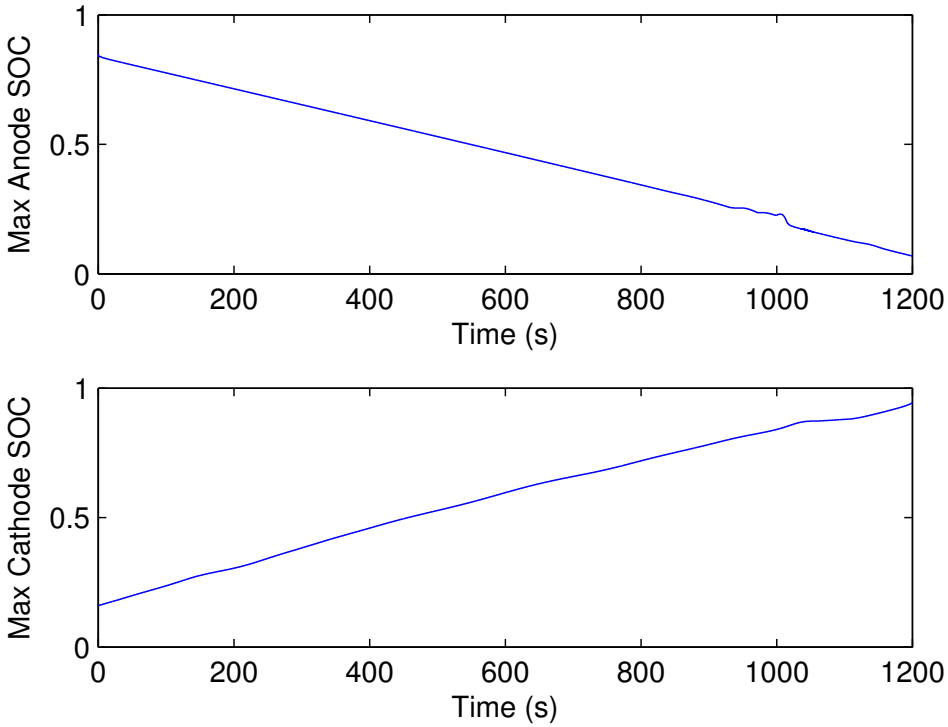

Figure 5.8. 2.5 C Discharge: State of Charge

5.2.3 Pulse Tests

5.2.3.1 60 Second Pulse

The current trajectory for this problem first discharges the battery at 2.5 C for 10 minutes and then rests the battery before pulse charging for 60 seconds followed immediately by pulse discharging for 60 seconds. The pulses have a magnitude of 0.5 C, 1.5 C, 3.0 C, and 5.0 C.

The large voltage spikes that can be seen in Figure 5.9 for the 1.5 C, 3.0 C, and 5.0 C pulses is a result of the internal resistance of the battery. Figure 5.11 shows the voltage spikes for the 5.0 C pulse charge and discharge. The voltage spike from the charging pulse is larger than the pulse that comes from the subsequent discharge. Relaxation dynamics can also be seen in Figure 5.11 at the end of the pulse discharge (3600 seconds).

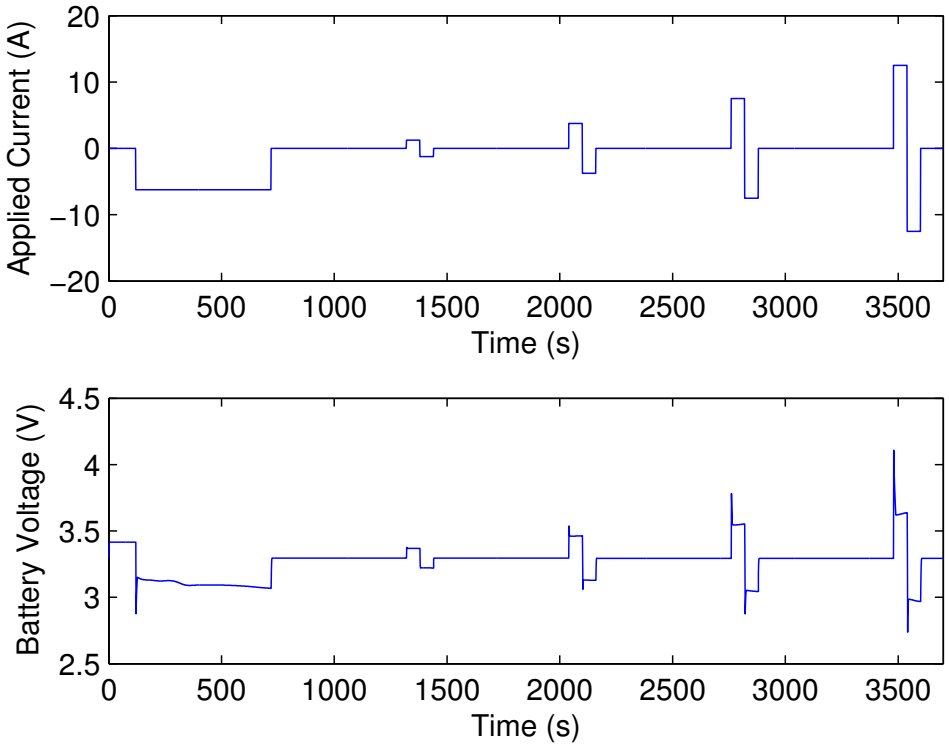

Figure 5.9. 60 Second Pulses: Current and Voltage

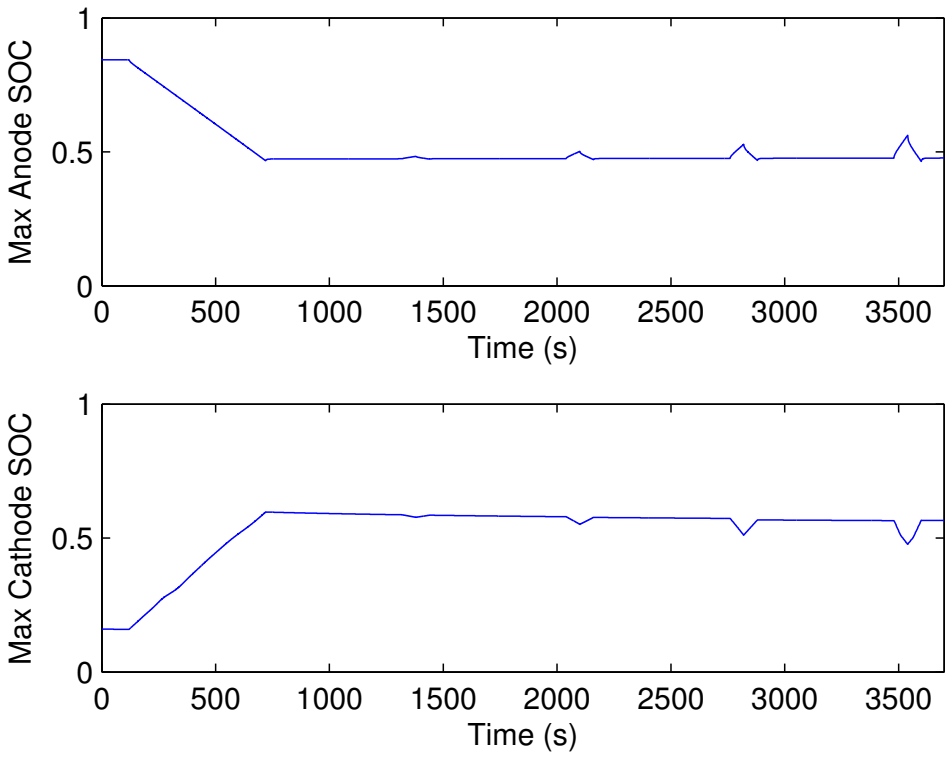

Figure 5.10. 60 Second Pulses: State of Charge

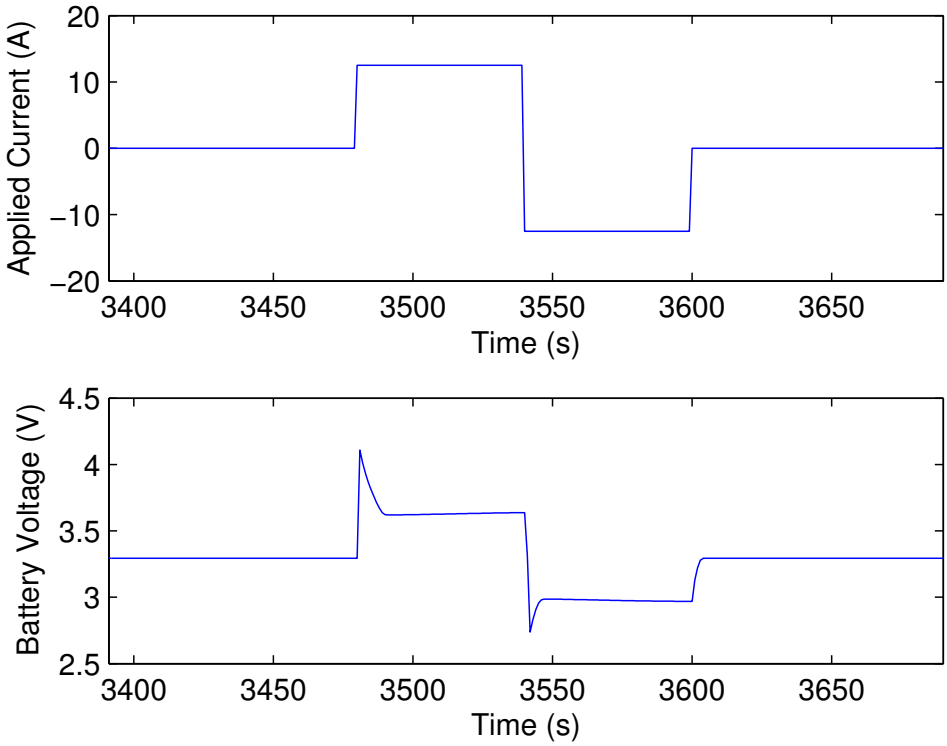

Figure 5.11. 60 Second Pulses: Current and Voltage (Zoom)

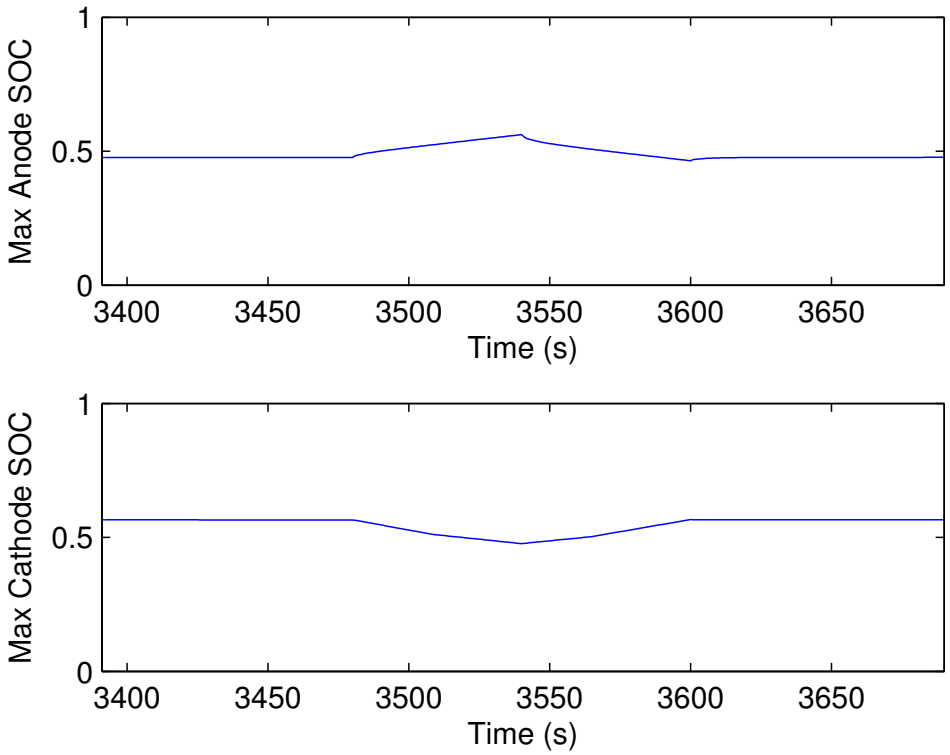

Figure 5.12. 60 Second Pulses: State of Charge (Zoom)

5.2.3.2 300 Second Pulse

Similar to Section 5.2.3.1, the battery is discharged from 90% to approximately 50% initially at a rate of 2.5 C for 10 minutes. After this initial discharge, four charge-discharge pulses are applied to the battery, each lasting for 300 seconds. Voltage spikes can be seen in Figure 5.13 for the 1.5 C, 3.0 C, and 5.0 C pulses, with the charge pulse voltage spikes being larger than the discharge voltage spikes. Focusing on Figure 5.16, the cathode SoC decreases at the end of the discharge pulse. This is caused by the lithium ion diffusing in the cathode, which lowers the maximum SoC in the cathode as time increases.

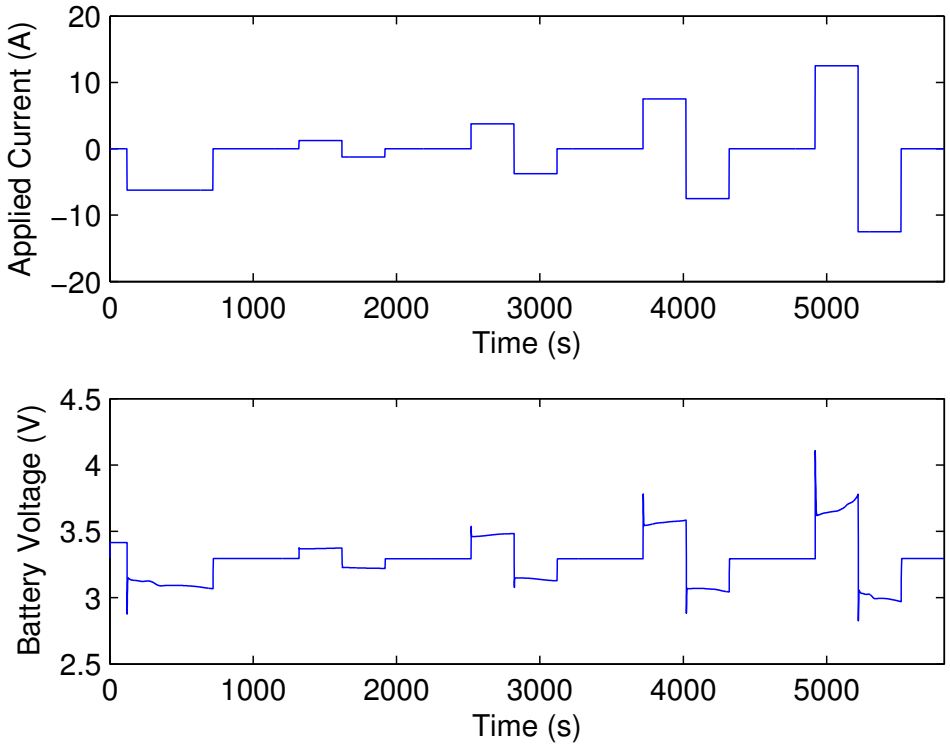

Figure 5.13. 300 Second Pulses: Current and Voltage

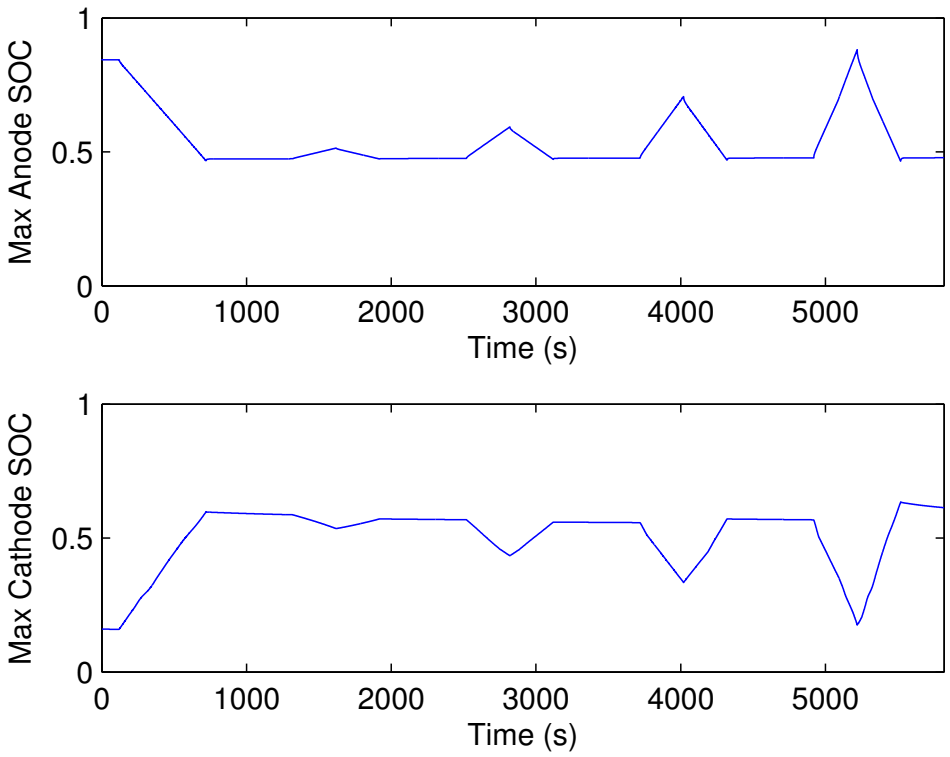

Figure 5.14. 300 Second Pulses: State of Charge

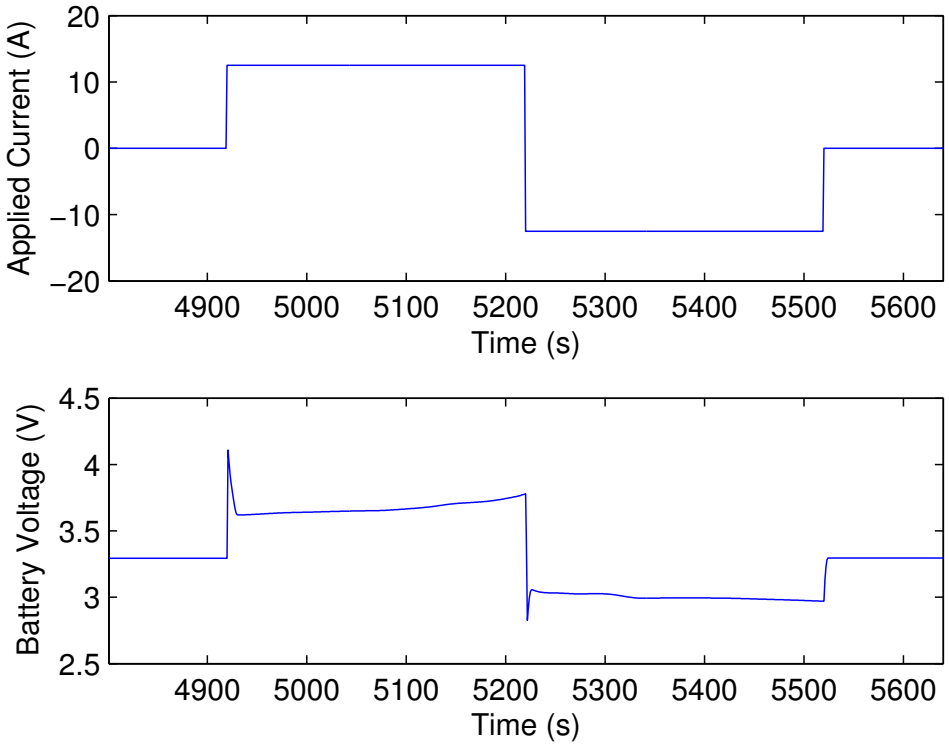

Figure 5.15. 300 Second Pulses: Current and Voltage (Zoom)

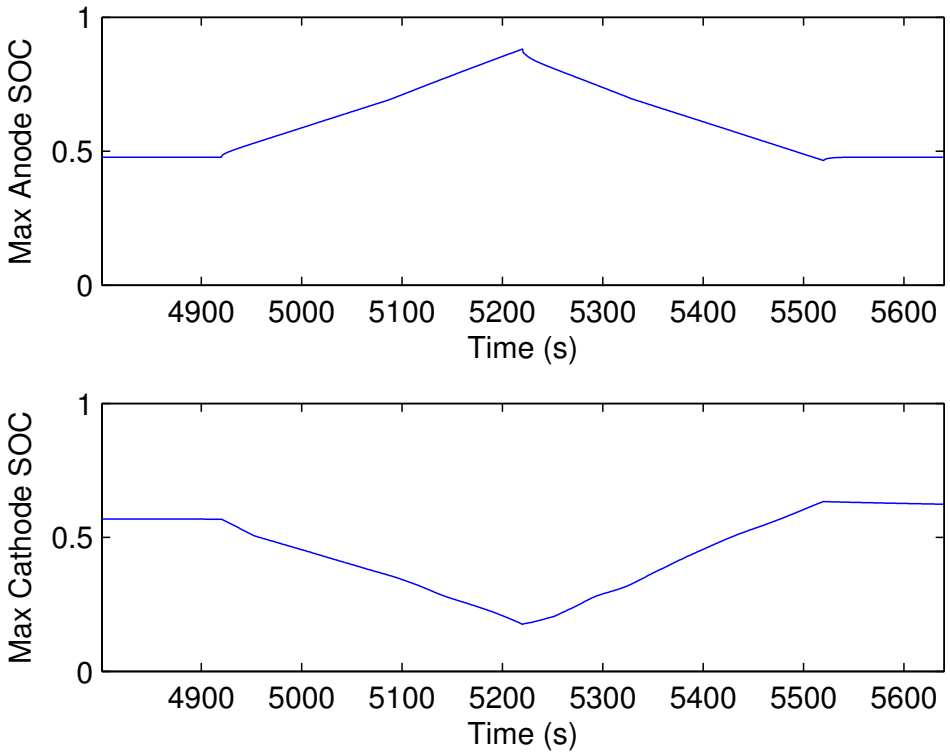

Figure 5.16. 300 Second Pulses: State of Charge (Zoom)

Chapter 6

Conclusions

The primary focus of this thesis is to create an electrochemical battery model using orthogonal projections in descriptor form. The first chapter reviews the equations of the Doyle Fuller Newman model and the previous work done by other researchers in the field of battery modeling and model reduction. The previous work is excellent; however none of the previous authors discuss the benefits and drawbacks of using orthogonal functions, or the tradeoffs between computation speed and accuracy in the number of projections used for modeling.

An orthonormal set of polynomials is used in the orthogonal projection method. The Legendre Polynomials are typically an orthogonal set over the interval $[-1, 1]$, however in the DFN model, two different sets of Legendre Polynomials are required, one over the interval $[0, L]$ and another over the interval $[-R, R]$. Chapter two discusses the process of normalizing and shifting the original set of Legendre Polynomials over the new intervals.

The orthogonal projection method is benchmarked against the finite difference method for a Cartesian one domain diffusion system. It can be seen that the fifth order orthogonal projection method solved in descriptor form is faster than the finite difference method for a large number of spatial discretization points. For $10,001$ time steps and $1,001$ spatial discretization points, the fifth order orthogonal projection method is over 15 times faster than the finite difference method.

Chapter three shows that there is a clear computational speed advantage in using orthogonal projections over the finite difference method, thus all five components of the Doyle Fuller Newman model are solved using orthogonal projections in descriptor form. By solving the coupled domain components of the DFN model in descriptor form, the boundary conditions can easily be added into the solution. The nonlinear components of the DFN model are linearized at every time step. The battery model is able to run faster than real time while giving similar results compared to the manufacturer's data.

There are two key results from this thesis. The first key result is that the fifth order orthogonal projection method solved in descriptor form is computationally faster than the finite difference method, with little loss in accuracy. The second key result is that the Doyle Fuller Newman solved wholly using orthogonal projections in descriptor form is capable of running faster than

real time while giving realistic results consistent with the manufacturer's data.

Bibliography

[1] A123. Nanophosphate High Power Lithium Ion Cell ANR26650M1-B, 2011.

[2] Aviva Brecher. Assessment of Needs and Research Roadmaps for Rechargeable Energy Storage System. Technical Report December 2010, U.S. Department of Transportation, 2011.

[3] Long Cai and Ralph E. White. Reduction of Model Order Based on Proper Orthogonal Decomposition for Lithium-Ion Battery Simulations. *Journal of The Electrochemical Society*, 156(3):A154, 2009.

[4] Long Cai and Ralph E. White. Lithium ion cell modeling using orthogonal collocation on finite elements. *Journal of Power Sources*, 217:248–255, November 2012.

[5] Marc Doyle, TF Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society*, 140(6):1526–1533, 1993.

[6] Joel C. Forman, Saeid Bashash, Jeffrey L. Stein, and Hosam K. Fathy. Reduction of an Electrochemistry-Based Li-Ion Battery Model via Quasi-Linearization and Pade Approximation. *Journal of The Electrochemical Society*, 158(2):A93, 2011.

[7] Joel C. Forman, Scott J. Moura, Jeffrey L. Stein, and Hosam K. Fathy. Genetic identification and fisher identifiability analysis of the DoyleFullerNewman model from experimental cycling of a LiFePO4 cell. *Journal of Power Sources*, 210:263–275, July 2012.

[8] TF Fuller, Marc Doyle, and John Newman. Simulation and optimization of the dual lithium ion insertion cell. *Journal of the Electrochemical Society*, 141(1), 1994.

[9] Meng Guo, Godfrey Sikha, and Ralph E. White. Single-Particle Model for a Lithium-Ion Cell: Thermal Behavior. *Journal of The Electrochemical Society*, 158(2):A122, 2011.

[10] S.I. Hayek. *Advanced Mathematical Methods in Science and Engineering*. CRC Press, second edition, 2011.

[11] Xueyan Li and Song-yul Choe. State-of-charge (SOC) estimation based on a reduced order electrochemical thermal model and extended Kalman filter. In *American Control Conference*, 2013.

[12] Paul W. C. Northrop, Venkatasailanathan Ramadesigan, Sumitava De, and Venkat R. Subramanian. Coordinate Transformation, Orthogonal Collocation, Model Reformulation and Simulation of Electrochemical-Thermal Behavior of Lithium-Ion Battery Stacks. *Journal of The Electrochemical Society*, 158(12):A1461, 2011.

[13] Christopher D. Rahn and Chao-Yang Wang. *Battery System Engineering*. Wiley, 1st edition, 2013.

[14] V. Ramadesigan, P. W. C. Northrop, S. De, S. Santhanagopalan, R. D. Braatz, and V. R. Subramanian. Modeling and Simulation of Lithium-Ion Batteries from a Systems Engineering Perspective. *Journal of the Electrochemical Society*, 159(3):R31–R45, January 2012.

[15] Kandler a. Smith, Christopher D. Rahn, and Chao-Yang Wang. Model Order Reduction of 1D Diffusion Systems Via Residue Grouping. *Journal of Dynamic Systems, Measurement, and Control*, 130(1):011012, 2008.

[16] Sony Corporation. Lithium Ion Rechargeable Batteries Technical Handbook. Technical report, Sony Corporation.

[17] Venkat R. Subramanian, Vinten D. Diwakar, and Deepak Tapriyal. Efficient Macro-Micro Scale Coupled Modeling of Batteries. *Journal of The Electrochemical Society*, 152(10):A2002, 2005.

Appendix A

Legendre Polynomial Calcuations

A.1 MATLAB Code to plot Legendre Polynomials from $[-1, 1]$

```
%% NonShift_NonNorm_LP_Plot.m

% Created by Michael D. Beeney

%% Cleaning the Workspace
close all
clear all
clc

%% Discretizing in space

num_of_x_steps = 501;

del_x_vector = linspace(-1,1,num_of_x_steps);

phi = zeros(6,length(del_x_vector));

phi(1,:)   = 1*ones(1,length(del_x_vector));
phi(2,:)   = del_x_vector;
phi(3,:)   = (1/2)*(3*del_x_vector.^2 - 1);
phi(4,:)   = (1/2)*(5*del_x_vector.^3 - 3*del_x_vector);
phi(5,:)   = (1/8)*(35*del_x_vector.^4 - 30*del_x_vector.^2 + 3);
phi(6,:)   = (1/8)*(63*del_x_vector.^5 - 70*del_x_vector.^3 + 15*del_x_vector);

%% Plotting
figure
plot(del_x_vector,phi)
legend( '\phi_0',...
        '\phi_1',...
        '\phi_2',...
        '\phi_3',...
        '\phi_4',...
        '\phi_5','Location','Best')
xlabel('x')
ylabel('\phi_n(x)')
ylim([-1.1,1.1])
xlim([-1.1,1.1])
```

A.2 Shifted, Normalized, Legendre Polynomials from $[0, L]$ Calcuations

Solving $\phi_0(x)$:

$$\begin{gathered}\int_0^L(\phi_0(x))^2\mathrm{d}x=1\\ \int_0^L(a_0)^2\mathrm{d}x=1\\ a_0=\sqrt{\frac{1}{L}}\end{gathered}\tag{A.1}$$

$$\phi_0(x)=\sqrt{\frac{1}{L}}$$

Solving $\phi_1(x)$:

$$\begin{gathered}\int_0^L\phi_0(x)\phi_1(x)\mathrm{d}x=0\\ \int_0^L(\sqrt{\frac{1}{L}})(a_1x+a_2)\mathrm{d}x=0\\ a_1\frac{L^2}{2}+a_2L=0\\ a_2=\frac{-L}{2}a_1\end{gathered}$$

$$\begin{gathered}\int_0^L(\phi_1(x))^2\mathrm{d}x=1\\ \int_0^L(a_1(x-\frac{L}{2})^2\mathrm{d}x=1\\ a_1=\sqrt{\frac{3}{L}}\frac{2}{L}\end{gathered}\tag{A.2}$$

$$\phi_1(x)=\sqrt{\frac{3}{L}}[\frac{2x}{L}-1]$$

Solving $\phi_2(x)$:

$$\begin{gathered}\int_0^L \phi_0(x)\phi_2(x)\mathrm{d}x = 0 \\ \int_0^L (\sqrt{\frac{1}{L}})(a_3x^2 + a_4x + a_5)\mathrm{d}x = 0 \\ a_5 = \frac{-a_3L^2}{3} + \frac{-a_4L}{2}\end{gathered}$$

$$\begin{gathered}\int_0^L \phi_1(x)\phi_2(x)\mathrm{d}x = 0 \\ \int_0^L (\sqrt{\frac{3}{L}}(\frac{2x}{L} - 1))(a_3x^2 + a_4x + a_5)\mathrm{d}x = 0 \\ a_4 = -La_3\end{gathered} \tag{A.3}$$

$$\begin{gathered}\int_0^L \phi_2(x)^2\mathrm{d}x = 0 \\ \int_0^L (a_3(x^2 - Lx + \frac{L^2}{6}))^2\mathrm{d}x = 0 \\ a_3 = \sqrt{\frac{5}{L}}\frac{6}{L^2}\end{gathered}$$

$$\phi_2(x) = \frac{1}{2}\sqrt{\frac{5}{L}}[3(\frac{2x}{L} - 1)^2 - 1]$$

It can be seen that two transformations are needed to create the normalized, shifted Legendre Polynomials from the non-shifted, non-normalized polynomials. The normalization is accomplished by multiplying the original polynomials by $\sqrt{\frac{2n+1}{L}}$, where n is the order of the polynomial. The shifting is done by substituting the expression $\frac{2x}{L} - 1$ for x in the original Legendre Polynomials.

A.3 MATLAB Code to plot Legendre Polynomials from $[0, L]$

```
%% Shift_Norm_LP_Plot.m

% Created by Michael D. Beeney

%% Cleaning the Workspace
close all
clear all
clc

%% Discretizing in space
L = 1;
num_of_x_steps = 501;

del_x_vector = linspace(0,L,num_of_x_steps);
```

```
phi = zeros(6,length(del_x_vector));

phi(1,:)    = sqrt(1/L)*1*ones(1,length(del_x_vector));
phi(2,:)    = sqrt(3/L)*(2*del_x_vector/L - 1);
phi(3,:)    = (1/2)*sqrt(5/L)*(3*(2*del_x_vector/L - 1).^2 - 1);
phi(4,:)    = (1/2)*sqrt(7/L)*(5*(2*del_x_vector/L - 1).^3 - 3*(2*del_x_vector/L - 1));
phi(5,:)    = (1/8)*sqrt(9/L)*(35*(2*del_x_vector/L - 1).^4 - 30*(2*del_x_vector/L - 1).^2 + 3);
phi(6,:)    = (1/8)*sqrt(11/L)*(63*(2*del_x_vector/L - 1).^5 - 70*(2*del_x_vector/L - 1).^3 + 15*(2*del_x
1));

%% Plotting
llimit = min(min(phi));
ulimit = max(max(phi));

figure
plot(del_x_vector,phi)
legend( '\phi_0',...
        '\phi_1',...
        '\phi_2',...
        '\phi_3',...
        '\phi_4',...
        '\phi_5','Location','Best')
xlabel('x')
ylabel('\phi_n(x)')
ylim([1.1*llimit,1.1*ulimit])
xlim([-0.1,L+0.1])
```

A.4 Normalized Legendre Polynomials from $[-R, R]$ Calcuations

Solving $\phi_0(r)$:

$$\begin{aligned} \int_0^R (\phi_0(r))^2 \mathrm{d}x &= 1 \\ \int_0^R (b_0)^2 \mathrm{d}x &= 1 \\ b_0 &= \sqrt{\frac{1}{R}} \end{aligned} \tag{A.4}$$

$$\phi_0(r) = \sqrt{\frac{1}{R}}$$

Solving $\phi_1(r)$:

$$\begin{gathered}\int_{-R}^{R}(\phi_0(r)\phi_1(r))\mathrm{d}r = 0 \\ \int_{-R}^{R}(\sqrt{\frac{1}{R}})(b_1 r + b_2)\mathrm{d}r = 0 \\ b_1(\frac{R^2}{2}) + b_2(R) - b_1(\frac{(-R)^2}{2}) - b_2(-R) = 0 \\ b_2 = 0\end{gathered}$$

$$\begin{gathered}\int_0^R(\phi_1(r))^2\mathrm{d}r = 1 \\ \int_0^R(b_1 r + b_2)^2\mathrm{d}r = 1 \\ b_1 = \frac{1}{R}\sqrt{\frac{3}{R}}\end{gathered} \tag{A.5}$$

$$\phi_1(r) = \sqrt{\frac{3}{R}}\frac{r}{R}$$

Solving $\phi_2(r)$:

$$\begin{gathered}\int_{-R}^{R}(\phi_0(r)\phi_2(r))\mathrm{d}r = 0 \\ \int_{-R}^{R}(\sqrt{\frac{1}{R}})(b_3 r^2 + b_4 r + b_5)\mathrm{d}r = 0 \\ b_3\frac{(R)^3}{3} + b_4\frac{(R)^2}{2} + b_5(R) - b_3\frac{(-R)^3}{3} - b_4\frac{(-R)^2}{2} - b_5(-R) = 0 \\ b_5 = -\frac{R^2}{3}b_3\end{gathered}$$

$$\begin{gathered}\int_{-R}^{R}(\phi_1(r)\phi_2(r))\mathrm{d}r = 0 \\ \int_{-R}^{R}(\frac{1}{R}\sqrt{\frac{3}{R}}r)(b_3 r^2 + b_4 r + -\frac{R^2}{3}b_3)\mathrm{d}r = 0 \\ b_3\frac{(R)^4}{4} + b_4\frac{(R)^3}{3} - b_5\frac{(R)^4}{6} - b_3\frac{(-R)^4}{4} - b_4\frac{(-R)^3}{3} + b_5\frac{(-R)^4}{6} = 0 \\ b_4 = 0\end{gathered} \tag{A.6}$$

$$\begin{gathered}\int_0^R(\phi_2(r)\phi_2(r))\mathrm{d}r = 1 \\ \int_0^R(b_3)^2(r^2 - R^2)\mathrm{d}r = 1 \\ (b_3)^2(\frac{1}{5}R^5 - \frac{2}{9}R^5 + \frac{1}{9}R^5) = 1 \\ b_3 = \frac{1}{2}\sqrt{\frac{5}{R}}\frac{3}{R^2}\end{gathered}$$

$$\phi_2(r) = \frac{1}{2}\sqrt{\frac{5}{R}}[3(\frac{r}{R})^2 - 1]$$

Analogous to the transformations that normalize and shift the interval of the original Legendre Polynomials, there are similar transformations that convert the original polynomials to

orthonormal Legendre Polynomials that are valid over the domain $[-R, R]$. The first transformation normalizes the polynomials by multiplying the original polynomials by $\sqrt{\frac{2n+1}{R}}$, where n is the order of the polynomial. The second transformation shifts the interval from [-1,1] to [-R,R] by replacing x in the original polynomials with $\frac{r}{R}$.

A.5 MATLAB Code to plot radial Legendre Polynomials

```
%% Radial_LP_Plot.m

% Created by Michael D. Beeney

%% Cleaning the workspace

close all
clear all
clc

%% Space

num_of_r_steps = 501;

R = 3;

del_r_vector = linspace(-R,R,num_of_r_steps);

r_nd = del_r_vector/R;

phi = zeros(7,length(del_r_vector));

phi(1,:) = sqrt( 1/R)*ones(1,length(del_r_vector));
phi(2,:) = sqrt( 3/R)*(r_nd);
phi(3,:) = ( 1/2)*sqrt( 5/R)*( 3*(r_nd).^2 -1 );
phi(4,:) = ( 1/2)*sqrt( 7/R)*( 5*(r_nd).^3 - 3*(r_nd) );
phi(5,:) = ( 1/8)*sqrt( 9/R)*( 35*(r_nd).^4 - 30*(r_nd).^2 + 3 );
phi(6,:) = ( 1/8)*sqrt(11/R)*( 63*(r_nd).^5 - 70*(r_nd).^3 + 15*(r_nd) );
phi(7,:) = (1/16)*sqrt(13/R)*( 231*(r_nd).^6 - 315*(r_nd).^4 + 105*(r_nd).^2 - 5 );

%% Plotting
llimit = min(min(phi));
ulimit = max(max(phi));

figure
plot(del_r_vector,phi)

legend( '\phi_0',...
        '\phi_1',...
        '\phi_2',...
        '\phi_3',...
        '\phi_4',...
        '\phi_5',...
        '\phi_6','Location','Best')
xlabel('r')
ylabel('\phi_n(r)')
xlim([-1.1*R, 1.1*R])
ylim([1.1*llimit, 1.1*ulimit])
```

Appendix B

Orthogonal Projection MATLAB Code

B.1 Linear, One Domain Diffusion: Finite Difference Method

```
function [time_to_run,concentration_profile ] = One_Domain_Finite_Diff(F,t_plus,Diff,eps,del_x_vector,de
% function version of finite differences

%% need to give it

% 1) Physical Parameters
% F
% t_plus
% Diff
% eps
% Length

% 2) Time
% start_time
% stop_time
% step_time

% 3) Space
% num_of_x_steps


% 4) Inputs
% J

% 5) Initial Conditions
% concentration_initial


%% Sim Param

% time
% del_t_vector = start_time:step_time:stop_time; % [sec]

% space

% del_x_vector = linspace(0,Length,num_of_x_steps);
del_x = del_x_vector(2);

%% Calculations

% timing the solution
tic

% A matrix
A = zeros(num_of_x_steps); % pre-allocate space

A(1,1) = -1; % row 1
A(1,2) = 1;  % row 1
for i = 2:1:(num_of_x_steps-1)
    A(i,i-1) = 1; % row 2 to N-1
    A(i,i) = -2;  % row 2 to N-1
    A(i,i+1) = 1; % row 2 to N-1
```

```
end
A(num_of_x_steps,num_of_x_steps-1)=1; % row N
A(num_of_x_steps,num_of_x_steps)=-1;  % row N

A_norm = (Diff)/(eps*(del_x^2))*A; % Out front of the matrix is a factor.

% B_matrix

B = ((1-t_plus)/(eps*F))*eye(num_of_x_steps); % Because each current density only goes into 1 node, the


% C Matrix

C = eye(num_of_x_steps); % Again, because I want my output to be each node at

% D Matrix
D = zeros(num_of_x_steps); % no feed forward component

SYS = ss(A_norm,B,C,D); % Trnasforms the A-D matrices into a state space variable in MATLAB

concentration_profile = lsim(SYS,J,del_t_vector,concentration_initial)'; % simulates the system with the


% average_concentration = mean(concentration_profile); % this averages the concentration across the leng

time_to_run = toc; % Saves the amount of time the code took to run to compare each method in terms of co

end
```

B.2 Linear, One Domain Diffusion: Fourth Order Orthogonal Projection Method

```
function [ time_to_run,concentration_profile,concentration_profile_initial,concentration_profile_initial
] = One_Domain_Four_Legendre( F,t_plus,Diff,eps,del_x_vector,del_t_vector,num_of_x_steps,J,concentratio
% One_Domain_Four_Legendre

% This function simulates one domain diffusion using 4 Shifted, Normalized Legendre Polynomials
% (from 0 to 3)



%% space

del_x = del_x_vector(2);

%% Generating Normalized, Shifted, Legendre Polynomials

tic % Timing the Solution

phi_0 = zeros(1,num_of_x_steps);
phi_1 = zeros(1,num_of_x_steps);
phi_2 = zeros(1,num_of_x_steps);
phi_3 = zeros(1,num_of_x_steps);

for i = 1:num_of_x_steps
    phi_0(i) = sqrt(1)*1;
    phi_1(i) = sqrt(3)*(2*(del_x_vector(i)) -1);
    phi_2(i) = sqrt(5)*(6*(del_x_vector(i))^2 - 6*(del_x_vector(i)) + 1);
    phi_3(i) = sqrt(7)*(20*(del_x_vector(i))^3 - 30*(del_x_vector(i))^2 + 12*(del_x_vector(i)) -1);
end
```

```
%% Generating Gammas from the Input, J

gamma_0 = zeros(1,length(del_t_vector));
gamma_1 = zeros(1,length(del_t_vector));
gamma_2 = zeros(1,length(del_t_vector));
gamma_3 = zeros(1,length(del_t_vector));

for i = 1:length(del_t_vector)
    gamma_0(i) = trapz((J(:,i)').*phi_0)/(num_of_x_steps-1);
    gamma_1(i) = trapz((J(:,i)').*phi_1)/(num_of_x_steps-1);
    gamma_2(i) = trapz((J(:,i)').*phi_2)/(num_of_x_steps-1);
    gamma_3(i) = trapz((J(:,i)').*phi_3)/(num_of_x_steps-1);
end


%% Initial Condtions

% Free Betas

beta_0_init_free = trapz(concentration_profile_initial_raw.*phi_0)/(num_of_x_steps-1);
beta_1_init_free = trapz(concentration_profile_initial_raw.*phi_1)/(num_of_x_steps-1);
beta_2_init_free = trapz(concentration_profile_initial_raw.*phi_2)/(num_of_x_steps-1);
beta_3_init_free = trapz(concentration_profile_initial_raw.*phi_3)/(num_of_x_steps-1);

concentration_profile_initial_free = beta_0_init_free*phi_0 + beta_1_init_free*phi_1 + beta_2_init_free*

% Constrained Betas

alpha_1 = sqrt(84/85);
alpha_3 = - alpha_1/(2*sqrt(21));

L = alpha_1*beta_1_init_free + alpha_3*beta_3_init_free;

beta_0_initial = beta_0_init_free; % no constraint
beta_1_initial = L/sqrt(85/84);
beta_2_initial = 0; % from the BC, beta_2 must be zero for all time
beta_3_initial = (-beta_1_initial)/(2*sqrt(21));

concentration_profile_initial = beta_0_initial*phi_0 + beta_1_initial*phi_1 + beta_2_initial*phi_2 + bet

%% Calculations

k1 = Diff/eps;
k2 = (1-t_plus)/(F*eps);

% beta 0
A_beta_0 = k1*0;
B_beta_0 = k2*1;
C_beta_0 = 1;
D_beta_0 = 0;

sys_beta_0 = ss(A_beta_0,B_beta_0,C_beta_0,D_beta_0);

beta_0_sol = lsim(sys_beta_0,gamma_0,del_t_vector,beta_0_initial);


% beta 1
A_beta_1 = k1*(-10);
B_beta_1 = k2*1;
C_beta_1 = 1;
D_beta_1 = 0;

sys_beta_1 = ss(A_beta_1,B_beta_1,C_beta_1,D_beta_1);
```

```
beta_1_sol = lsim(sys_beta_1,gamma_1,del_t_vector,beta_1_initial);

% beta 2

A_beta_2 = k1*(0);
B_beta_2 = k2*1;
C_beta_2 = 1;
D_beta_2 = 0;

sys_beta_2 = ss(A_beta_2,B_beta_2,C_beta_2,D_beta_2);

beta_2_sol = lsim(sys_beta_2,gamma_2,del_t_vector,beta_2_initial);

beta_3_sol = - beta_1_sol/(2*sqrt(21));

% Adding them all together

zero_part = (beta_0_sol*phi_0)';
one_part = (beta_1_sol*phi_1)';
two_part = (beta_2_sol*phi_2)';
three_part = (beta_3_sol*phi_3)';

concentration_profile = (zero_part + one_part + two_part + three_part);

average_concentration = mean(concentration_profile);

time_to_run = toc;

end
```

B.3 Linear, One Domain Diffusion: Fifth Order Orthogonal Projection Method

```
function [ time_to_run,concentration_profile,concentration_profile_initial,concentration_profile_initial
] = One_Domain_Five_Legendre(F,t_plus,Diff,eps,del_x_vector,del_t_vector,num_of_x_steps,J,concentration_
% One_Domain_Five_Legendre

% This function finds the solution to the one domain diffusion problem by
% using five, shifted, normalized Legendre Polynomials


%% space

del_x = del_x_vector(2);


%% Generating Normalized, Shifted, Legendre Polynomials

tic

phi_0 = zeros(1,num_of_x_steps);
phi_1 = zeros(1,num_of_x_steps);
phi_2 = zeros(1,num_of_x_steps);
phi_3 = zeros(1,num_of_x_steps);
phi_4 = zeros(1,num_of_x_steps);

for i = 1:num_of_x_steps
    phi_0(i) = sqrt(1)*1;
    phi_1(i) = sqrt(3)*(2*(del_x_vector(i)) -1);
    phi_2(i) = sqrt(5)*(6*(del_x_vector(i))^2 - 6*(del_x_vector(i)) + 1);
    phi_3(i) = sqrt(7)*(20*(del_x_vector(i))^3 - 30*(del_x_vector(i))^2 + 12*(del_x_vector(i)) -1);
```

```
    phi_4(i) = sqrt(9)*(70*(del_x_vector(i))^4 - 140*(del_x_vector(i))^3 + 90*(del_x_vector(i))^2 -
20*(del_x_vector(i)) +1);

end

%% Generating Gammas from the Input, J

gamma_0 = zeros(1,length(del_t_vector));
gamma_1 = zeros(1,length(del_t_vector));
gamma_2 = zeros(1,length(del_t_vector));
gamma_3 = zeros(1,length(del_t_vector));
gamma_4 = zeros(1,length(del_t_vector));


for i = 1:length(del_t_vector)
    gamma_0(i) = trapz((J(:,i)').*phi_0)/(num_of_x_steps-1);
    gamma_1(i) = trapz((J(:,i)').*phi_1)/(num_of_x_steps-1);
    gamma_2(i) = trapz((J(:,i)').*phi_2)/(num_of_x_steps-1);
    gamma_3(i) = trapz((J(:,i)').*phi_3)/(num_of_x_steps-1);
    gamma_4(i) = trapz((J(:,i)').*phi_4)/(num_of_x_steps-1);

end

%% Initial Condtions

% Free Betas

beta_0_init_free = trapz(concentration_profile_initial_raw.*phi_0)/(num_of_x_steps-1);
beta_1_init_free = trapz(concentration_profile_initial_raw.*phi_1)/(num_of_x_steps-1);
beta_2_init_free = trapz(concentration_profile_initial_raw.*phi_2)/(num_of_x_steps-1);
beta_3_init_free = trapz(concentration_profile_initial_raw.*phi_3)/(num_of_x_steps-1);
beta_4_init_free = trapz(concentration_profile_initial_raw.*phi_4)/(num_of_x_steps-1);

concentration_profile_initial_free = beta_0_init_free*phi_0 + beta_1_init_free*phi_1 + beta_2_init_free*

% NEW The boundary conditions constrain the inital condtions!
% NEW X2 beta1 and beta3 constrain each other, beta2 and beta4 constrain
% each other

beta_0_initial = beta_0_init_free; % no constraint


% beta 1 and beta 3
alpha_1 = sqrt(84/85);
alpha_3 = - alpha_1/(2*sqrt(21));

L_1 = alpha_1*beta_1_init_free + alpha_3*beta_3_init_free;
beta_1_initial = L_1/sqrt(85/84);
beta_3_initial = (-beta_1_initial)/(2*sqrt(21));

% beta 2 and beta 4
alpha_2 = sqrt(20/21);
alpha_4 = -alpha_2/(2*sqrt(5));

L_2 = alpha_2*beta_2_init_free + alpha_4*beta_4_init_free;

beta_2_initial = L_2/sqrt(20/21); % from the BC, beta_2 must be zero for all time
beta_4_initial = (-beta_2_initial)/(2*sqrt(5));

concentration_profile_initial = beta_0_initial*phi_0 + beta_1_initial*phi_1 + beta_2_initial*phi_2 + bet

%% Calculations

k1 = Diff/eps;
```

```
k2 = (1-t_plus)/(F*eps);

% beta 0
A_beta_0 = k1*0;
B_beta_0 = k2*1;
C_beta_0 = 1;
D_beta_0 = 0;

sys_beta_0 = ss(A_beta_0,B_beta_0,C_beta_0,D_beta_0);

beta_0_sol = lsim(sys_beta_0,gamma_0,del_t_vector,beta_0_initial);

% beta 1
A_beta_1 = k1*(-10);
B_beta_1 = k2*1;
C_beta_1 = 1;
D_beta_1 = 0;

sys_beta_1 = ss(A_beta_1,B_beta_1,C_beta_1,D_beta_1);

beta_1_sol = lsim(sys_beta_1,gamma_1,del_t_vector,beta_1_initial);

% beta 2
A_beta_2 = k1*(-42);
B_beta_2 = k2*1;
C_beta_2 = 1;
D_beta_2 = 0;

sys_beta_2 = ss(A_beta_2,B_beta_2,C_beta_2,D_beta_2);

beta_2_sol = lsim(sys_beta_2,gamma_2,del_t_vector,beta_2_initial);

% beta 3
beta_3_sol = - beta_1_sol/(2*sqrt(21));

% beta 4
beta_4_sol = - beta_2_sol/(2*sqrt(5));

% Adding them all together

zero_part = (beta_0_sol*phi_0)';
one_part = (beta_1_sol*phi_1)';
two_part = (beta_2_sol*phi_2)';
three_part = (beta_3_sol*phi_3)';
four_part = (beta_4_sol*phi_4)';

% concentration_profile = (zero_part + one_part + two_part + three_part);

concentration_profile = (zero_part + one_part + two_part + three_part + four_part);

average_concentration = mean(concentration_profile);

time_to_run = toc;

end
```

B.4 Linear, One Domain Diffusion: Fifth Order Orthogonal Projection Method: Descriptor Form

```
function [ time_to_run,concentration_profile, concentration_profile_init_free] = One_Domain_Five_Legendr
% One_Domain_Five_Legendre_Descriptor
```

```
% This method solves the one domain diffusion problem using five,
% normalized shifted Legendre Polynomials.

% The big difference in this one is this method uses backwards differencing
% and descriptor form

%% Space

del_x = del_x_vector(2) - del_x_vector(1);

%% Time

del_t = del_t_vector(2) - del_t_vector(1);


%% Generating Normalized, Shifted, Legendre Polynomials
tic

phi_0 = zeros(1,num_of_x_steps);
phi_1 = zeros(1,num_of_x_steps);
phi_2 = zeros(1,num_of_x_steps);
phi_3 = zeros(1,num_of_x_steps);
phi_4 = zeros(1,num_of_x_steps);


for i = 1:num_of_x_steps
    phi_0(i) = sqrt(1)*1;
    phi_1(i) = sqrt(3)*(2*(del_x_vector(i)) -1);
    phi_2(i) = sqrt(5)*(6*(del_x_vector(i))^2 - 6*(del_x_vector(i)) + 1);
    phi_3(i) = sqrt(7)*(20*(del_x_vector(i))^3 - 30*(del_x_vector(i))^2 + 12*(del_x_vector(i)) -1);
    phi_4(i) = sqrt(9)*(70*(del_x_vector(i))^4 - 140*(del_x_vector(i))^3 + 90*(del_x_vector(i))^2 -
20*(del_x_vector(i)) +1);

end

%% Pre allocating space to speed up the computation

beta_vector = zeros(5,length(del_t_vector));

%% Generating Gammas from the input, J

gamma_0 = zeros(1,length(del_t_vector));
gamma_1 = zeros(1,length(del_t_vector));
gamma_2 = zeros(1,length(del_t_vector));
% gamma_3 = zeros(1,length(del_t_vector));
% gamma_4 = zeros(1,length(del_t_vector));


for i = 1:length(del_t_vector)
    gamma_0(i) = trapz((J(:,i)').*phi_0)/(num_of_x_steps-1);
    gamma_1(i) = trapz((J(:,i)').*phi_1)/(num_of_x_steps-1);
    gamma_2(i) = trapz((J(:,i)').*phi_2)/(num_of_x_steps-1);
%     gamma_3(i) = trapz((J(:,i)').*phi_3)/(num_of_x_steps-1);
%     gamma_4(i) = trapz((J(:,i)').*phi_4)/(num_of_x_steps-1);

end

gamma_vector(1,:) = gamma_0;
gamma_vector(2,:) = gamma_1;
gamma_vector(3,:) = gamma_2;


%% Initial Conditions
```

```
beta_0_init_free = trapz(concentration_profile_initial_raw.*phi_0)/(num_of_x_steps-1);
beta_1_init_free = trapz(concentration_profile_initial_raw.*phi_1)/(num_of_x_steps-1);
beta_2_init_free = trapz(concentration_profile_initial_raw.*phi_2)/(num_of_x_steps-1);
% beta_3_init_free = trapz(concentration_profile_initial_raw.*phi_3)/(num_of_x_steps-1);
% beta_4_init_free = trapz(concentration_profile_initial_raw.*phi_4)/(num_of_x_steps-1);

beta_3_init_free = -(2*sqrt(3))/(12*sqrt(7))*beta_1_init_free;
beta_4_init_free = -(6*sqrt(5))/(20*sqrt(9))*beta_2_init_free;

concentration_profile_init_free = beta_0_init_free*phi_0 + beta_1_init_free*phi_1 + beta_2_init_free*phi


% beta_vector = zeros(5,length(del_t_vector+1));

beta_vector(1,1) = beta_0_init_free;
beta_vector(2,1) = beta_1_init_free;
beta_vector(3,1) = beta_2_init_free;
beta_vector(4,1) = beta_3_init_free;
beta_vector(5,1) = beta_4_init_free;

%% Calcuations

% the _un means the A and B matrices are unscaled, i.e. they have not been
% multiplied by the k1 and k2 constants

k1 = Diff/eps;
k2 = (1-t_plus)/(F*eps);


E = zeros(5,5);
E(1,1) = 1;
E(2,2) = 1;
E(3,3) = 1;

A_un = zeros(5,5);

A_un(1,3) = sqrt(5)*12;
A_un(1,5) = sqrt(9)*40;

A_un(2,4) = sqrt(21)*20;

A_un(3,5) = sqrt(45)*28;

A_un(4,2) = 2*sqrt(3)*2;
A_un(4,4) = 2*sqrt(7)*12;

A_un(5,3) = 2*sqrt(5)*6;
A_un(5,5) = 2*sqrt(9)*20;


A = k1*A_un;

B_un = zeros(5,3);
B_un(1,1) = 1;
B_un(2,2) = 1;
B_un(3,3) = 1;

B = k2*B_un;

inverted_matrix = inv(E/del_t - A);


for i = 1:length(del_t_vector)
```

```
    beta_vector(:,i+1) = inverted_matrix*(E/del_t*beta_vector(:,i) + B*gamma_vector(:,i));
end

beta_0_sol = beta_vector(1,:)';
beta_1_sol = beta_vector(2,:)';
beta_2_sol = beta_vector(3,:)';
beta_3_sol = beta_vector(4,:)';
beta_4_sol = beta_vector(5,:)';

zero_part = (beta_0_sol*phi_0)';
one_part = (beta_1_sol*phi_1)';
two_part = (beta_2_sol*phi_2)';
three_part = (beta_3_sol*phi_3)';
four_part = (beta_4_sol*phi_4)';

concentration_profile = (zero_part + one_part + two_part + three_part + four_part);

time_to_run = toc;
end
```

B.5 Linear, One Domain Diffusion: Run Script

```
%% Script to Run All

% Commented and Documented 2013 - 03 - 29

% This script is design to give the four functions
%
% 1) One_Domain_Finite_Diff (This solves the one domain diffusion equation
%    using Finite Difference)
% 2) One_Domain_Four_Legendre (This solves the one domain diffusion
% equation using 4 Legendre Polynomials. I had to substitute in the the BC
% manually into the orignal A Matrix)
% 3) One_Domain_Five_Legendre (This solves the one domain diffusion
% equation using 5 Legendre Polynomials. I had to substitute in the the BC
% manually into the orignal A Matrix)
% 4) One_Domain_Five_Legendre_Descriptor (This solves the one domain
% diffusion problem using 5 Legendre Polynomials, however it requires no
% substitution like the previous method)
%
% The same number of space discrtetizations, time discretizations, physical
% parameters (such as diffusion)

% It also can choose between 7 individual initial conditions and 4 inputs
% to the system
%
% And also this script will then plot each case individually, and then plot
% Functions 1-3 in the same figure, and similarly plot Functions 1, 3,4 on
% the same figure to compare which method is better

%% Cleaning the Workspace

close all
clear all
clc

%% Physical Parameters

F = 96; % [units?]
t_plus = 0.25; % [units?]
Diff = 0.85; % [units?]
```

```
eps = 0.92; % [units?]
Length = 1; % [m]

%% Time

start_time = 0; % [sec]
stop_time = 1; % [sec]
step_time = 0.001; % [sec]
del_t_vector = start_time:step_time:stop_time; % [sec]
length_of_time = length(del_t_vector);
% Timesteps to plot
time_to_plot = [(1) (101) (201) (501) (1001) (length(del_t_vector))]; % Picks the time steps I want to p
% time_to_plot = [(1) (11) (21) (101) (201) (length(del_t_vector))]; % Picks the time steps I want to pl

%% Space

num_of_x_steps = 1001; % [unitless]
del_x_vector = linspace(0,Length,num_of_x_steps);
del_x = del_x_vector(2);

%% Input

% Pre allocate

J = zeros(num_of_x_steps,length(del_t_vector));

choice = menu('Select an input','Input 0','Input 1','Input 2','Input 3');

switch (choice)
    case {1}
        J = zeros(num_of_x_steps,length(del_t_vector));
    case {2}

        a = 0.1*(num_of_x_steps - 1);
        b = 0.8*(num_of_x_steps - 1);

        input_shape(1:a) = 2;
        for i = (a+1):b
            input_shape(i) = -10/7*(del_x_vector(i)) + 15/7;
        end
        input_shape((b+1):num_of_x_steps) = 1;

        J = ones(num_of_x_steps,length(del_t_vector));

        for i = 1:(length(del_t_vector))
            J(:,i) = input_shape;
        end

    case {3}
        a = 0.1*(num_of_x_steps - 1);
        b = 0.8*(num_of_x_steps - 1);

        input_shape(1:a) = 2;
        for i = (a+1):b
            input_shape(i) = -10/7*(del_x_vector(i)) + 15/7;
        end
        input_shape((b+1):num_of_x_steps) = 1;

        J = ones(num_of_x_steps,length(del_t_vector));

        for i = 1:(length(del_t_vector))
            time_increase = (i-1)/length(del_t_vector);
            J(:,i) = time_increase*input_shape;
        end
```

```
    case {4}

        mu = 0.5;

        sigma = 0.5;

        input_shape = 10*normpdf(del_x_vector,mu,sigma);

        for i = 1:(length(del_t_vector))
            time_increase = (i-1)/length(del_t_vector);
            J(:,i) = time_increase*input_shape;
            J(:,i) = input_shape;
        end
end



%% Init Conditions

choice = menu('Select an initial condition','IC 0','IC 1','IC 2','IC 3', 'IC 4', 'IC 5', 'IC 6');

switch (choice)
    case{1}
        concentration_initial = zeros(1,length(del_x_vector));
    case{2}
        a = 0.1*(num_of_x_steps - 1);
        b = 0.8*(num_of_x_steps - 1);

        concentration_initial(1:a) = 2;
        for i = (a+1):b
            concentration_initial(i) = -10/7*(del_x_vector(i)) + 15/7;
        end
        concentration_initial((b+1):num_of_x_steps) = 1;
    case{3}
        mu = 0.5;

        sigma = 1;

        concentration_initial = 10*normpdf(del_x_vector,mu,sigma);
    case{4}
        mu = 0.5;

        sigma = 0.1;

        concentration_initial = 10*normpdf(del_x_vector,mu,sigma);

    case{5}
        mu = 0.3;

        sigma = 1;

        concentration_initial = 10*normpdf(del_x_vector,mu,sigma);

    case{6}
        mu = 0.3;

        sigma = 0.1;

        concentration_initial = 10*normpdf(del_x_vector,mu,sigma);

    case{7}
        mu = 0.45;
```

```
        sigma = 0.15;

        concentration_initial = 10*normpdf(del_x_vector,mu,sigma);
end

%% Finite Difference
tic
[time_to_run_FD,concentration_profile_FD ] = One_Domain_Finite_Diff(F,t_plus,Diff,eps,del_x_vector,del_t
time_to_run_FD_out = toc
%% Orthogonal Projections using 4 Normalized, Shifted Legendre Polynomials
tic
[time_to_run_OP4,concentration_profile_OP4,concentration_profile_initial_OP4,concentration_profile_initi
time_to_run_OP4_out = toc
%% Orthogonal Projections using 5 Normalized, Shifted Legendre Polynomials
tic
[ time_to_run_OP5,concentration_profile_OP5,concentration_profile_initial_OP5,concentration_profile_init
time_to_run_OP5_out = toc
%% Descriptor Form
tic
[ time_to_run_DF,concentration_profile_DF, concentration_profile_init_free_DF] = One_Domain_Five_Legendr
time_to_run_DF_out = toc

%% Calculating error
error_FD_OP4 = zeros(1,length(del_t_vector));
error_FD_OP5 = zeros(1,length(del_t_vector));
error_FD_DF = zeros(1,length(del_t_vector));

for i = 1:length(del_t_vector)
    error_FD_OP4(i) = mean(abs(concentration_profile_FD(1:num_of_x_steps,i)-concentration_profile_OP4(1:
    error_FD_OP5(i) = mean(abs(concentration_profile_FD(1:num_of_x_steps,i)-concentration_profile_OP5(1:
    error_FD_DF(i)  = mean(abs(concentration_profile_FD(1:num_of_x_steps,i)-concentration_profile_DF(1:n
end

%% Plotting

% Plotting_Script
% Extra script to plot functions

%% Animation

% Animation_Script
```

Appendix C

Doyle Fuller Newman Model Calculations and Code

C.1 Solid Phase Concentration Calculations

The solid phase potential has a dependency on both the linear and radial direction. It is easiest to consider the radial component first, then solve for the $\phi(x)$. The negative and positive electrode can be solved in a similar manner, so only the negative electrode calculations are done.

First, simplify the governing equation, Equation 4.13

$$\frac{\partial c_{1,n}(r,x,t)}{\partial t} = d_{1,n}(\frac{2}{r}\frac{\partial c_{1,n}(r,x,t)}{\partial r} + \frac{\partial^2 c_{1,n}(r,x,t)}{\partial r^2}) \tag{C.1}$$

After this, simplifty the orthogonal projection expression, Equation 4.1.

$$\begin{aligned} c_{1,n} = &\begin{bmatrix} \omega_{0,n} & \omega_{2,n} & \omega_{4,n} & \omega_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0 + \cdots \\ &+ \begin{bmatrix} \omega_{0,n} & \omega_{2,n} & \omega_{4,n} & \omega_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{17,n} \\ \tau_{18,n} \\ \tau_{19,n} \\ \tau_{20,n} \end{bmatrix} \phi_4 \end{aligned} \tag{C.2}$$

Next, substitute in the orthogonal projection expression in for $c_{1,n}$.

$$\begin{aligned}
&\begin{bmatrix} \omega_{0,n} & \cdots & \omega_{6,n} \end{bmatrix} \begin{bmatrix} \dot{\tau_{1,n}} \\ \dot{\tau_{2,n}} \\ \dot{\tau_{3,n}} \\ \dot{\tau_{4,n}} \end{bmatrix} \phi_0(x) + \cdots + \begin{bmatrix} \omega_{0,n} & \cdots & \omega_{6,n} \end{bmatrix} \begin{bmatrix} \dot{\tau_{17,n}} \\ \dot{\tau_{18,n}} \\ \dot{\tau_{19,n}} \\ \dot{\tau_{20,n}} \end{bmatrix} \phi_4(x) \\
&= d_{1,n}(\tfrac{2}{r})(\begin{bmatrix} \omega'_{0,n} & \cdots & \omega'_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0(x) + \cdots + \begin{bmatrix} \omega'_{0,n} & \cdots & \omega'_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{17,n} \\ \tau_{18,n} \\ \tau_{19,n} \\ \tau_{20,n} \end{bmatrix} \phi_4(x)) \\
&d_{1,n}(\begin{bmatrix} \omega''_{0,n} & \cdots & \omega''_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0(x) + \cdots + \begin{bmatrix} \omega''_{0,n} & \cdots & \omega''_{6,n} \end{bmatrix} \begin{bmatrix} \tau_{17,n} \\ \tau_{18,n} \\ \tau_{19,n} \\ \tau_{20,n} \end{bmatrix} \phi_4(x))
\end{aligned} \tag{C.3}$$

Applying Galerkin projections ($\int_0^{R_n} \omega_{i,n}(r)\mathrm{d}r$, $i = 0, 2, 4, 6$):

$$\begin{aligned}
&\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\tau_{1,n}} \\ \dot{\tau_{2,n}} \\ \dot{\tau_{3,n}} \\ \dot{\tau_{4,n}} \end{bmatrix} \phi_0(x) + \cdots \\
&= \begin{bmatrix} 0 & \frac{9\sqrt{5}d_{1,n}}{R_n^2} & \frac{20d_{1,n}}{R_n^2} & \frac{147\sqrt{13}d_{1,n}}{5R_n^2} \\ 0 & 0 & \frac{35\sqrt{5}d_{1,n}}{R_n^2} & \frac{84\sqrt{65}d_{1,n}}{5R_n^2} \\ 0 & 0 & 0 & \frac{231\sqrt{13}d_{1,n}}{5R_n^2} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0(x) + \cdots
\end{aligned} \tag{C.4}$$

Only the boundary at $r = R_n$ (Equation 4.15) needs to be applied, the boundary condition at $r = 0$ is automatically satisfied by selection of the even Radial Legendre Polynomials.

$$\begin{bmatrix} 0 & \frac{3\sqrt{5}}{R_n^{3/2}} & \frac{30}{R_n^{3/2}} & \frac{21\sqrt{13}}{R_n^{3/2}} \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0(x) \cdots + \\ \frac{1}{d_{1,n} a_n F} \begin{bmatrix} \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix} = 0 \tag{C.5}$$

The second boundary condition is substituted into the last row of Equation C.4 to give:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\tau_{1,n}} \\ \dot{\tau_{2,n}} \\ \dot{\tau_{3,n}} \\ \dot{\tau_{4,n}} \end{bmatrix} \phi_0(x) \cdots \\ = \begin{bmatrix} 0 & \frac{9\sqrt{5}d_{1,n}}{R_n^2} & \frac{20d_{1,n}}{R_n^2} & \frac{147\sqrt{13}d_{1,n}}{5R_n^2} \\ 0 & 0 & \frac{35\sqrt{5}d_{1,n}}{R_n^2} & \frac{84\sqrt{65}d_{1,n}}{5R_n^2} \\ 0 & 0 & 0 & \frac{231\sqrt{13}d_{1,n}}{5R_n^2} \\ 0 & \frac{3\sqrt{5}}{R_n^{3/2}} & \frac{30}{R_n^{3/2}} & \frac{21\sqrt{13}}{R_n^{3/2}} \end{bmatrix} \begin{bmatrix} \tau_{1,n} \\ \tau_{2,n} \\ \tau_{3,n} \\ \tau_{4,n} \end{bmatrix} \phi_0(x) \cdots \\ + \frac{1}{d_{1,n} a_n F} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix} \tag{C.6}$$

It is easier now to write Equation C.6 in a more compact form, including all the $\phi(x)$ terms.

$$[E]_{4X4}[\dot{\vec{\tau}}_{1\cdots4,n}]\phi_0(x) + \cdots [E][\dot{\vec{\tau}}_{17\cdots20,n}]\phi_4(x) = \\ 1[A]_{4X4}[\vec{\tau}_{1\cdots4,n}]\phi_0(x) + \cdots [A][\vec{\tau}_{17\cdots20,n}]\phi_4(x) + \\ \frac{1}{d_{1,n} a_n F}[A]_{4X5}[\vec{\gamma}n] \tag{C.7}$$

Applying Galerkin projections using $\phi_{i,n}(x)$ yields state equations for $\tau_{1\cdots20,n}$.

$$\begin{bmatrix} [E]_{4X4} & 0 & 0 & 0 & 0 \\ 0 & [E] & 0 & 0 & 0 \\ 0 & 0 & [E] & 0 & 0 \\ 0 & 0 & 0 & [E] & 0 \\ 0 & 0 & 0 & 0 & [E] \end{bmatrix} \begin{bmatrix} \dot{\vec{\tau}}_{1\cdots4,n} \\ \dot{\vec{\tau}}_{5\cdots8,n} \\ \dot{\vec{\tau}}_{9\cdots12,n} \\ \dot{\vec{\tau}}_{13\cdots16,n} \\ \dot{\vec{\tau}}_{17\cdots20,n} \end{bmatrix} = \begin{bmatrix} [A]_{4X4} & 0 & 0 & 0 & 0 \\ 0 & [A] & 0 & 0 & 0 \\ 0 & 0 & [A] & 0 & 0 \\ 0 & 0 & 0 & [A] & 0 \\ 0 & 0 & 0 & 0 & [A] \end{bmatrix} \begin{bmatrix} \vec{\tau}_{1\cdots4,n} \\ \vec{\tau}_{5\cdots8,n} \\ \vec{\tau}_{9\cdots12,n} \\ \vec{\tau}_{13\cdots16,n} \\ \vec{\tau}_{17\cdots20,n} \end{bmatrix} + \begin{bmatrix} [A]_{1X4} & 0 & 0 & 0 & 0 \\ 0 & [A] & 0 & 0 & 0 \\ 0 & 0 & [A] & 0 & 0 \\ 0 & 0 & 0 & [A] & 0 \\ 0 & 0 & 0 & 0 & [A] \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix} \tag{C.8}$$

C.2 Solution Phase Concentration Calculations

The negative, positive, and separator governing equations are very similar, thus only the calculations for $c_{2,n}$ are shown. After the calculations are completed for $c_{2,n}$, the full 15X15 descriptor form state space model is shown, including $c_{2,n}$, $c_{2,s}$, and $c_{2,p}$, but without boundary conditions imposed. After this, the boundary conditions will be solved and substituted back into the descriptor form state space equations to give the final form of the coupled three domain solution phase concentration.

Equations 4.3 and 4.11 are first substituted into Equation 4.19 to give:

$$\begin{bmatrix} \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix} \begin{bmatrix} \dot{\beta_{1,n}} \\ \dot{\beta_{2,n}} \\ \dot{\beta_{3,n}} \\ \dot{\beta_{4,n}} \\ \dot{\beta_{5,n}} \end{bmatrix} =$$

$$\frac{d_{2,n}}{\epsilon_n} \begin{bmatrix} \phi''_{0,n} & \phi''_{1,n} & \phi''_{2,n} & \phi''_{3,n} & \phi''_{4,n} \end{bmatrix} \begin{bmatrix} \beta_{1,n} \\ \beta_{2,n} \\ \beta_{3,n} \\ \beta_{4,n} \\ \beta_{5,n} \end{bmatrix} + \tag{C.9}$$

$$\frac{1-t_n{}^+}{\epsilon_n F} \begin{bmatrix} \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix}$$

After this, the integrals $\int_0^{L_n} \phi_{0\ldots4,n} \mathrm{d}x$ are applied to give the following 5X5 descriptor form of the negative electrode domain:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\beta_{1,n}} \\ \dot{\beta_{2,n}} \\ \dot{\beta_{3,n}} \\ \dot{\beta_{4,n}} \\ \dot{\beta_{5,n}} \end{bmatrix} =$$

$$\frac{d_{2,n}}{\epsilon_n} \begin{bmatrix} 0 & 0 & \frac{12\sqrt{5}}{{L_n}^2} & 0 & \frac{120}{{L_n}^2} \\ 0 & 0 & 0 & \frac{20\sqrt{21}}{{L_n}^2} & 0 \\ 0 & 0 & 0 & 0 & \frac{84\sqrt{5}}{{L_n}^2} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_{1,n} \\ \beta_{2,n} \\ \beta_{3,n} \\ \beta_{4,n} \\ \beta_{5,n} \end{bmatrix} + \tag{C.10}$$

$$\frac{1-{t_n}^+}{\epsilon_n F} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix}$$

The 15X15 three domain system will have the form:

$$\begin{bmatrix} [E]n & 0 & 0 \\ 0 & [E]s & 0 \\ 0 & 0 & [E]p \end{bmatrix} \begin{bmatrix} \dot{\vec{\beta_n}} \\ \dot{\vec{\beta_s}} \\ \dot{\vec{\beta_p}} \end{bmatrix} = \begin{bmatrix} [A]n & 0 & 0 \\ 0 & [A]s & 0 \\ 0 & 0 & [A]p \end{bmatrix} \begin{bmatrix} \vec{\beta_n} \\ \vec{\beta_s} \\ \vec{\beta_p} \end{bmatrix} + \begin{bmatrix} [B]n & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & [B]p \end{bmatrix} \begin{bmatrix} \vec{\gamma_n} \\ \vec{0} \\ \vec{\gamma_p} \end{bmatrix} \tag{C.11}$$

Boundary conditions 1 (Equation 4.22) and 6 (Equation 4.27) replace the fourth and fifteenth rows in the 15X15 descriptor form model of the solution phase concentration. Substituting the orthogonal projection expression into BC 1 and BC 6 yields:

$$\beta_{1,n}\phi'_{0,n} + \beta_{2,n}\phi'_{1,n} + \beta_{3,n}\phi'_{2,n} + \beta_{4,n}\phi'_{3,n} + \beta_{5,n}\phi'_{4,n} = 0$$

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{{L_n}^{3/2}} & \frac{-6\sqrt{5}}{{L_n}^{3/2}} & \frac{12\sqrt{7}}{{L_n}^{3/2}} & \frac{-60}{{L_n}^{3/2}} \end{bmatrix} \begin{bmatrix} \beta_{1,n} \\ \beta_{2,n} \\ \beta_{3,n} \\ \beta_{4,n} \\ \beta_{5,n} \end{bmatrix} = 0 \tag{C.12}$$

$$\beta_{1,p}\phi'_{0,p} + \beta_{2,p}\phi'_{1,p} + \beta_{3,p}\phi'_{2,p} + \beta_{4,p}\phi'_{3,p} + \beta_{5,p}\phi'_{4,p} = 0$$

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{{L_p}^{3/2}} & \frac{6\sqrt{5}}{{L_p}^{3/2}} & \frac{12\sqrt{7}}{{L_p}^{3/2}} & \frac{60}{{L_p}^{3/2}} \end{bmatrix} \begin{bmatrix} \beta_{1,p} \\ \beta_{2,p} \\ \beta_{3,p} \\ \beta_{4,p} \\ \beta_{5,p} \end{bmatrix} = 0 \tag{C.13}$$

Boundary conditions 2 (Equation 4.23) and 5 (Equation 4.26) replace the fifth and fourteenth rows in the 15X15 descriptor form model. Substituting the orthogonal projection expression into BC 2 and BC 5 gives:

$$(\beta_{1,n}\phi_{0,n} + \beta_{2,n}\phi_{1,n} + \beta_{3,n}\phi_{2,n} + \beta_{4,n}\phi_{3,n} + \beta_{5,n}\phi_{4,n}) -$$
$$(\beta_{1,s}\phi_{0,s} + \beta_{2,s}\phi_{1,s} + \beta_{3,s}\phi_{2,s} + \beta_{4,s}\phi_{3,s} + \beta_{5,s}\phi_{4,s}) = 0$$
$$\begin{bmatrix} \sqrt{\frac{1}{L_n}} & \sqrt{\frac{3}{L_n}} & \sqrt{\frac{5}{L_n}} & \sqrt{\frac{7}{L_n}} & \sqrt{\frac{9}{L_n}} \end{bmatrix} \vec{\beta_n} - \begin{bmatrix} \sqrt{\frac{1}{L_s}} & -\sqrt{\frac{3}{L_s}} & \sqrt{\frac{5}{L_s}} & -\sqrt{\frac{7}{L_s}} & \sqrt{\frac{9}{L_s}} \end{bmatrix} \vec{\beta_s} = 0 \tag{C.14}$$

$$(\beta_{1,s}\phi_{0,s} + \beta_{2,s}\phi_{1,s} + \beta_{3,s}\phi_{2,s} + \beta_{4,s}\phi_{3,s} + \beta_{5,s}\phi_{4,s}) -$$
$$(\beta_{1,p}\phi_{0,p} + \beta_{2,p}\phi_{1,p} + \beta_{3,p}\phi_{2,p} + \beta_{4,p}\phi_{3,p} + \beta_{5,p}\phi_{4,p}) = 0$$
$$\begin{bmatrix} \sqrt{\frac{1}{L_s}} & \sqrt{\frac{3}{L_s}} & \sqrt{\frac{5}{L_s}} & \sqrt{\frac{7}{L_s}} & \sqrt{\frac{9}{L_s}} \end{bmatrix} \vec{\beta_s} - \begin{bmatrix} \sqrt{\frac{1}{L_p}} & -\sqrt{\frac{3}{L_p}} & \sqrt{\frac{5}{L_p}} & -\sqrt{\frac{7}{L_p}} & \sqrt{\frac{9}{L_p}} \end{bmatrix} \vec{\beta_p} = 0 \tag{C.15}$$

The final two boundary conditions, 3 and 4 (Equations 4.24 and 4.25) replace the ninth and tenth rows of the solution phase concentration model. Substituting the orthogonal projection expressions into the boundary conditions simplifies to:

$$d_{2,n}\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{{L_n}^{3/2}} & \frac{6\sqrt{5}}{{L_n}^{3/2}} & \frac{12\sqrt{7}}{{L_n}^{3/2}} & \frac{60}{{L_n}^{3/2}} \end{array}\right]\vec{\beta_n} - d_{2,s}\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{{L_s}^{3/2}} & \frac{-6\sqrt{5}}{{L_s}^{3/2}} & \frac{12\sqrt{7}}{{L_s}^{3/2}} & \frac{-60}{{L_s}^{3/2}} \end{array}\right]\vec{\beta_s} = 0 \tag{C.16}$$

$$d_{2,s}\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{{L_s}^{3/2}} & \frac{6\sqrt{5}}{{L_s}^{3/2}} & \frac{12\sqrt{7}}{{L_s}^{3/2}} & \frac{60}{{L_s}^{3/2}} \end{array}\right]\vec{\beta_s} - d_{2,p}\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{{L_p}^{3/2}} & \frac{-6\sqrt{5}}{{L_p}^{3/2}} & \frac{12\sqrt{7}}{{L_p}^{3/2}} & \frac{-60}{{L_p}^{3/2}} \end{array}\right]\vec{\beta_p} = 0 \tag{C.17}$$

C.3 Solid Phase Potential Calculations

The solid phase potential governing equations have no time derivative, however to be consistent with the solid and solution phase concentration, the solid phase potential will also be solved in descriptor form. The negative and positive electrode have similar governing equations, thus only the negative electrode is solved. However, the boundary conditions are different, and will be solved separately.

Negative Electrode:

First, it is easier to re-write the governing equation, Equation 4.28.

$$0\dot{P}_{1,n} = \sigma_n^{eff} P''_{1,n} - J \tag{C.18}$$

Substituting in Equations 4.6 and 4.11 into the governing equation:

$$
\begin{aligned}
0\begin{bmatrix} \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix}
\begin{bmatrix} \dot{\zeta}_{1,n} \\ \dot{\zeta}_{2,n} \\ \dot{\zeta}_{3,n} \\ \dot{\zeta}_{4,n} \\ \dot{\zeta}_{5,n} \end{bmatrix} = \\
\sigma_n^{eff}\begin{bmatrix} \phi''_{0,n} & \phi''_{1,n} & \phi''_{2,n} & \phi''_{3,n} & \phi''_{4,n} \end{bmatrix}
\begin{bmatrix} \zeta_{1,n} \\ \zeta_{2,n} \\ \zeta_{3,n} \\ \zeta_{4,n} \\ \zeta_{5,n} \end{bmatrix} - \\
\begin{bmatrix} \phi_{0,n} & \phi_{1,n} & \phi_{2,n} & \phi_{3,n} & \phi_{4,n} \end{bmatrix}
\begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix}
\end{aligned}
\tag{C.19}
$$

Applying Galerkin Projections to Equation C.19 yields the following descriptor form:

$$
\begin{aligned}
[0]_{5X5}\begin{bmatrix} \dot{\zeta}_{1,n} \\ \dot{\zeta}_{2,n} \\ \dot{\zeta}_{3,n} \\ \dot{\zeta}_{4,n} \\ \dot{\zeta}_{5,n} \end{bmatrix} =& \sigma_n^{eff}\begin{bmatrix} 0 & 0 & \frac{12\sqrt{5}}{L_n^2} & 0 & \frac{120}{L_n^2} \\ 0 & 0 & 0 & \frac{20\sqrt{21}}{L_n^2} & 0 \\ 0 & 0 & 0 & 0 & \frac{84\sqrt{5}}{L_n^2} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} \zeta_{1,n} \\ \zeta_{2,n} \\ \zeta_{3,n} \\ \zeta_{4,n} \\ \zeta_{5,n} \end{bmatrix} - \\
& \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix}
\end{aligned}
\tag{C.20}
$$

Equation C.20 is similar for the positive electrode. The last two rows will be replaced by either the negative or positive electrode boundary conditions depending upon the domain.

For the negative electrode, Equation 4.29 becomes:

$$\left[\begin{array}{ccccc} \sqrt{\frac{1}{L_n}} & -\sqrt{\frac{3}{L_n}} & \sqrt{\frac{5}{L_n}} & -\sqrt{\frac{7}{L_n}} & \sqrt{\frac{9}{L_n}} \end{array} \right] \left[\begin{array}{c} \zeta_{1,n} \\ \zeta_{2,n} \\ \zeta_{3,n} \\ \zeta_{4,n} \\ \zeta_{5,n} \end{array} \right] = 0 \tag{C.21}$$

And the second boundary condition, Equation 4.30 becomes:

$$\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{L_n^{3/2}} & \frac{6\sqrt{5}}{L_n^{3/2}} & \frac{12\sqrt{7}}{L_n^{3/2}} & \frac{20\sqrt{9}}{L_n^{3/2}} \end{array} \right] \left[\begin{array}{c} \zeta_{1,n} \\ \zeta_{2,n} \\ \zeta_{3,n} \\ \zeta_{4,n} \\ \zeta_{5,n} \end{array} \right] = 0 \tag{C.22}$$

Equations C.21 and C.22 replace the last two rows in Equation C.20 for the negative electrode solid phase potential.

For the positive electrode, the boundary conditions are Equations 4.32 and 4.33, which replaces the last two rows of the positive electrode solid phase potential descriptor form. The two boundary conditions become:

$$\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{L_p^{3/2}} & -\frac{6\sqrt{5}}{L_p^{3/2}} & \frac{12\sqrt{7}}{L_p^{3/2}} & -\frac{20\sqrt{9}}{L_p^{3/2}} \end{array} \right] \left[\begin{array}{c} \zeta_{1,p} \\ \zeta_{2,p} \\ \zeta_{3,p} \\ \zeta_{4,p} \\ \zeta_{5,p} \end{array} \right] = 0 \tag{C.23}$$

$$\left[\begin{array}{ccccc} 0 & \frac{2\sqrt{3}}{L_p^{3/2}} & \frac{6\sqrt{5}}{L_p^{3/2}} & \frac{12\sqrt{7}}{L_p^{3/2}} & \frac{20\sqrt{9}}{L_p^{3/2}} \end{array} \right] \left[\begin{array}{c} \zeta_{1,p} \\ \zeta_{2,p} \\ \zeta_{3,p} \\ \zeta_{4,p} \\ \zeta_{5,p} \end{array} \right] = \frac{iapp}{Area * \sigma_p^{eff}} \tag{C.24}$$

The last boundary equation, Equation C.24 contains the only input to the battery model, the applied current.

C.4 Solution Phase Potential Calculations

The negative, positive and separator governing equations are similar, thus only the negative electrode solution phase potential governing equation is solved and quasi-linearized. The first step is to simplify Equation 4.34.

$$0\dot{\vec{P}}_{2,n} = \kappa_n^{eff} P''_{2,n} + J_n + \kappa_{D,n}[\frac{-1}{c_{2,n}^2}(c'_{2,n})^2 + \frac{1}{c_{2,n}}c''_{2,n}] \tag{C.25}$$

The first two terms are linear and do not need to be linearized, however the last term in Equation C.25 is nonlinear and is quasi-linearized. The linear terms are solved first (f_l), followed by the quasi-linearization of the nonlinear term (f_{nl}).

Linear:

The first step is to substitute in the orthogonal expressions for $P_{2,n}$ and J_n, and ignore the nonlinear component. This gives the following expression:

$$f_l = \begin{bmatrix} \phi''_{0,n} & \cdots & \phi''_{4,n} \end{bmatrix}\vec{\delta}n + \begin{bmatrix} \phi_{0,n} & \cdots & \phi_{4,n} \end{bmatrix}\vec{\gamma}n \tag{C.26}$$

Applying Galerkin Projections then gives 5 equations. The linear component then has the form:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \kappa_n^{eff} \begin{bmatrix} 0 & 0 & \frac{12\sqrt{5}}{L_n^2} & 0 & \frac{120}{L_n^2} \\ 0 & 0 & 0 & \frac{20\sqrt{21}}{L_n^2} & 0 \\ 0 & 0 & 0 & 0 & \frac{84\sqrt{5}}{L_n^2} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_{1,n} \\ \delta_{2,n} \\ \delta_{3,n} \\ \delta_{4,n} \\ \delta_{5,n} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_{1,n} \\ \gamma_{2,n} \\ \gamma_{3,n} \\ \gamma_{4,n} \\ \gamma_{5,n} \end{bmatrix} \tag{C.27}$$

The same can be applied to the separator and positive electrode, to give a total of 15 equations.

Nonlinear:

First, substituting in the orthogonal projection expression for $c_{2,n}$:

$$f_{nl}(\vec{\beta}_n) = -\kappa_{D,n}\frac{(\beta_{1.n}\phi'_{0,n} + \cdots + \beta_{5.n}\phi'_{4,n})^2}{(\beta_{1.n}\phi_{0,n} + \cdots + \beta_{5.n}\phi_{4,n})^2} + \kappa_{D,n}\frac{(\beta_{1.n}\phi''_{0,n} + \cdots + \beta_{5.n}\phi''_{4,n})}{(\beta_{1.n}\phi_{0,n} + \cdots + \beta_{5.n}\phi_{4,n})} \quad \text{(C.28)}$$

Quasi-linearizing yields:

$$f(\vec{\beta}_n(k)) = f(\vec{\beta}_n(k-1)) + \frac{\partial f_{nl}}{\partial \beta_{1,n}}(\beta_{1,n}(k) - \beta_{1,n}(k-1)) + \cdots + \frac{\partial f_{nl}}{\partial \beta_5}(\beta_{5,n}(k) - \beta_{5,n}(k-1)) \quad \text{(C.29)}$$

Where $f(\vec{\beta}_n(k-1))$ is the nonlinear component evaluated at the last time step, $\frac{\partial f_{nl}}{\partial \beta_{i,n}}$ is the derivate evaluated at the previous time step, and $\beta_{i,n}(k)$ and $\beta_{i,n}(k-1)$ are the state variables at the present and previous time step respectively.

Rearranging Equation C.29 based on the time step:

$$\begin{aligned} f(\vec{\beta}_n(k)) = &\begin{bmatrix} \frac{\partial f_{nl}}{\partial \beta_{1,n}} & \cdots & \frac{\partial f_{nl}}{\partial \beta_{5,n}} \end{bmatrix}\vec{\beta}_n(k) \\ +f(\vec{\beta}_n(k-1)) - &\begin{bmatrix} \frac{\partial f_{nl}}{\partial \beta_{1,n}} & \cdots & \frac{\partial f_{nl}}{\partial \beta_{5,n}} \end{bmatrix}\vec{\beta}_n(k-1) \end{aligned} \quad \text{(C.30)}$$

Evaluating $\frac{\partial f_{nl}}{\partial \beta_{1,n}}$ and $\frac{\partial f_{nl}}{\partial \beta_{5,n}}$ gives the following:

$$\begin{aligned} \frac{\partial f_{nl}}{\partial \beta_{1,n}} = &-2\frac{\kappa_{D,n}(\sum \beta_{i,n}\phi'_{i,n})\phi'_{1,n}}{(\sum \beta_{i,n}\phi_{i,n})^2} + 2\frac{\kappa_{D,n}(\sum \beta_{i,n}\phi'_{i,n})^2\phi_{1,n}}{(\sum \beta_{i,n}\phi_{i,n})^3} \\ &+\frac{\kappa_{D,n}\phi''_{1,n}}{(\sum \beta_{i,n}\phi_{i,n})} - \frac{\kappa_{D,n}(\sum \beta_{i,n}\phi_{i,n}{}'')\phi_{1,n}}{(\sum \beta_{i,n}\phi_{i,n})^2} \end{aligned} \quad \text{(C.31)}$$

$$\begin{aligned} \frac{\partial f_{nl}}{\partial \beta_{5,n}} = &-2\frac{\kappa_{D,n}(\sum \beta_{i,n}\phi'_{i,n})\phi'_{5,n}}{(\sum \beta_{i,n}\phi_{i,n})^2} + 2\frac{\kappa_{D,n}(\sum \beta_{i,n}\phi'_{i,n})^2\phi_{5,n}}{(\sum \beta_{i,n}\phi_{i,n})^3} \\ &+\frac{\kappa_{D,n}\phi''_{5,n}}{(\sum \beta_{i,n}\phi_{i,n})} - \frac{\kappa_{D,n}(\sum \beta_{i,n}\phi_{i,n}{}'')\phi_{5,n}}{(\sum \beta_{i,n}\phi_{i,n})^2} \end{aligned} \quad \text{(C.32)}$$

Equations C.31 and C.32 are then substituted back into Equation C.30. Galerkin Projections are then applied to give 5 equations.

The first term, multiplied by $\vec{\beta}_n(k)$ will be a $5X5$ matrix that will be included in the A matrix, shown in Equation 4.45.

$$[A]_{5X5} = \begin{bmatrix} \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{1,n}} \phi_{0,n} \mathrm{d}x & \cdots & \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{5,n}} \phi_{0,n} \mathrm{d}x \\ \vdots & \ddots & \vdots \\ \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{1,n}} \phi_{4,n} \mathrm{d}x & \cdots & \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{5,n}} \phi_{4,n} \mathrm{d}x \end{bmatrix} \tag{C.33}$$

The last two terms, associated with $\vec{\beta}_n(k-1)$, is a $5X1$ vector that is a part of the G vector in Equation 4.45.

$$[G]_{5X1} = \begin{bmatrix} \int_0^{L_n} f(\vec{\beta}_n(k-1)) \phi_{0,n} \mathrm{d}x \\ \vdots \\ \int_0^{L_n} f(\vec{\beta}_n(k-1)) \phi_{4,n} \mathrm{d}x \end{bmatrix} - \begin{bmatrix} \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{1,n}} \phi_{0,n} \mathrm{d}x & \cdots & \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{5,n}} \phi_{0,n} \mathrm{d}x \\ \vdots & \ddots & \vdots \\ \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{1,n}} \phi_{4,n} \mathrm{d}x & \cdots & \int_0^{L_n} \frac{\partial f_{nl}}{\partial \beta_{5,n}} \phi_{4,n} \mathrm{d}x \end{bmatrix} [\vec{\beta}_n(k-1)] \tag{C.34}$$

The linear and nonlinear parts are added back together to give 5 state equations. A similar procedure is done for the separator and positive electrode to give a further 10 state equations. The boundary conditions couple the negative electrode, separator, and positive electrode solution phase potential. Boundary conditions 1, 2, 5, 6, are linear and are solved first.

Boundary conditions 1 (Equation 4.37) and 6 (Equation 4.42) replace rows 4 and 15 in the solution phase potential equations. The equations for boundary conditions 1 and 6 are:

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L_n^{3/2}} & \frac{-6\sqrt{5}}{L_n^{3/2}} & \frac{12\sqrt{7}}{L_n^{3/2}} & \frac{-20\sqrt{9}}{L_n^{3/2}} \end{bmatrix} \begin{bmatrix} \delta_{1,n} \\ \delta_{2,n} \\ \delta_{3,n} \\ \delta_{4,n} \\ \delta_{5,n} \end{bmatrix} = 0 \tag{C.35}$$

$$\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L_p^{3/2}} & \frac{+6\sqrt{5}}{L_p^{3/2}} & \frac{12\sqrt{7}}{L_p^{3/2}} & \frac{+20\sqrt{9}}{L_p^{3/2}} \end{bmatrix} \begin{bmatrix} \delta_{1,p} \\ \delta_{2,p} \\ \delta_{3,p} \\ \delta_{4,p} \\ \delta_{5,p} \end{bmatrix} = 0 \tag{C.36}$$

Rows 5 and 14 are replaced with the second boundary condition (Equation 4.38) and fifth

boundary condition (Equation 4.41).

$$\begin{bmatrix} \sqrt{\frac{1}{L_n}} & \sqrt{\frac{3}{L_n}} & \sqrt{\frac{5}{L_n}} & \sqrt{\frac{7}{L_n}} & \sqrt{\frac{9}{L_n}} \end{bmatrix} \begin{bmatrix} \delta_{1,n} \\ \delta_{2,n} \\ \delta_{3,n} \\ \delta_{4,n} \\ \delta_{5,n} \end{bmatrix} - \begin{bmatrix} \sqrt{\frac{1}{L_s}} & -\sqrt{\frac{3}{L_s}} & \sqrt{\frac{5}{L_s}} & -\sqrt{\frac{7}{L_s}} & \sqrt{\frac{9}{L_s}} \end{bmatrix} \begin{bmatrix} \delta_{1,s} \\ \delta_{2,s} \\ \delta_{3,s} \\ \delta_{4,s} \\ \delta_{5,s} \end{bmatrix} = 0 \tag{C.37}$$

$$\begin{bmatrix} \sqrt{\frac{1}{L_s}} & \sqrt{\frac{3}{L_s}} & \sqrt{\frac{5}{L_s}} & \sqrt{\frac{7}{L_s}} & \sqrt{\frac{9}{L_s}} \end{bmatrix} \begin{bmatrix} \delta_{1,s} \\ \delta_{2,s} \\ \delta_{3,s} \\ \delta_{4,s} \\ \delta_{5,s} \end{bmatrix} - \begin{bmatrix} \sqrt{\frac{1}{L_p}} & -\sqrt{\frac{3}{L_p}} & \sqrt{\frac{5}{L_p}} & -\sqrt{\frac{7}{L_p}} & \sqrt{\frac{9}{L_p}} \end{bmatrix} \begin{bmatrix} \delta_{1,p} \\ \delta_{2,p} \\ \delta_{3,p} \\ \delta_{4,p} \\ \delta_{5,p} \end{bmatrix} = 0 \tag{C.38}$$

The final two boundary conditions, 3 and 4, replace rows 9 and 10. These two boundary conditions are nonlinear. The procedure is the same for both, thus only the calculations are shown for the third boundary condition (Equation 4.39).

The first step is to simplify BC3:

$$\kappa_n^{eff} P'_{2,n} - \kappa_s^{eff} P'_{2,s} + \kappa_{D,n}(\frac{1}{c_{2,n}} c'_{2,n}) - \kappa_{D,s}(\frac{1}{c_{2,s}} c'_{2,s}) = 0 \tag{C.39}$$

The next step is to solve for the two linear componenets. They have the form:

$$f_{l,BC3} = \kappa_n^{eff}\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L_n^{3/2}} & \frac{-6\sqrt{5}}{L_n^{3/2}} & \frac{12\sqrt{7}}{L_n^{3/2}} & \frac{-20\sqrt{9}}{L_n^{3/2}} \end{bmatrix}\begin{bmatrix} \delta_{1,n} \\ \delta_{2,n} \\ \delta_{3,n} \\ \delta_{4,n} \\ \delta_{5,n} \end{bmatrix} - \kappa_s^{eff}\begin{bmatrix} 0 & \frac{2\sqrt{3}}{L_s^{3/2}} & \frac{-6\sqrt{5}}{L_s^{3/2}} & \frac{12\sqrt{7}}{L_s^{3/2}} & \frac{-20\sqrt{9}}{L_s^{3/2}} \end{bmatrix}\begin{bmatrix} \delta_{1,s} \\ \delta_{2,s} \\ \delta_{3,s} \\ \delta_{4,s} \\ \delta_{5,s} \end{bmatrix} \tag{C.40}$$

The nonlinear components are linearized with respect to the $\vec{\beta}_n$ and $\vec{\beta}_s$ terms.

$$f_{nl,BC3}(\vec{\beta}_n, \vec{\beta}_s) = \kappa_{D,n}(\frac{1}{c_{2,n}} c'_{2,n}) - \kappa_{D,s}(\frac{1}{c_{2,s}} c'_{2,s}) \tag{C.41}$$

The quasi-linearization process is similar to the one shown in Equations C.28 to C.34.

C.5 Butler-Volmer Calculations

The calculations for the positive and negative electrodes are similar, so only the negative electrode calculations are shown here. The first step is to rearrange the Butler-Volmer equation, and assume that $\alpha_{a,n} = \alpha_{c,n} = \alpha_n$.

$$f_{BV} = -J_n + a_n(k_n(c_{1,n}^{max} - c_{1,n}^s)^{\alpha_n}(c_{1,n}^s)^{\alpha_n}(c_n)^{\alpha_n}) \sinh\left(\frac{\alpha_n F}{RT}(P_{1,n} - P_{2,n} - u_{nref})\right) = 0 \tag{C.42}$$

The linearized Butler-Volmer will have the form:

$$\begin{bmatrix} \frac{\partial f_{BV}}{\partial \gamma_{1,n}} & \cdots & \frac{\partial f_{BV}}{\partial \gamma_{5,n}} \end{bmatrix}(\vec{\gamma}_n(k) - \vec{\gamma}_n(k-1)) + \begin{bmatrix} \frac{\partial f_{BV}}{\partial \tau_{1,n}} & \cdots & \frac{\partial f_{BV}}{\partial \tau_{20,n}} \end{bmatrix}(\vec{\tau}_n(k) - \vec{\tau}_n(k-1)) +$$
$$\begin{bmatrix} \frac{\partial f_{BV}}{\partial \beta_{1,n}} & \cdots & \frac{\partial f_{BV}}{\partial \beta_{5,n}} \end{bmatrix}(\vec{\beta}_n(k) - \vec{\beta}_n(k-1)) + \begin{bmatrix} \frac{\partial f_{BV}}{\partial \zeta_{1,n}} & \cdots & \frac{\partial f_{BV}}{\partial \zeta_{5,n}} \end{bmatrix}(\vec{\zeta}_n(k) - \vec{\zeta}_n(k-1)) +$$
$$\begin{bmatrix} \frac{\partial f_{BV}}{\partial \delta_{1,n}} & \cdots & \frac{\partial f_{BV}}{\partial \delta_{5,n}} \end{bmatrix}(\vec{\delta}_n(k) - \vec{\delta}_n(k-1)) = 0 \tag{C.43}$$

Before substituting the expression for J_n, $c_{1,n}$, $c_{2,n}$, $P_{1,n}$, and $P_{2,n}$ then quasi-linearizing, it is easier to use the chain rule. For example, to linearize Equation C.42 with respect to $\gamma_{1,n}$ from Equation 4.11:

$$\frac{\partial f_{BV}}{\partial \gamma_{1,n}} = \frac{\partial f_{BV}}{\partial J_n}\frac{\partial J_n}{\partial \gamma_{1,n}} \tag{C.44}$$

The first and last components of each orthogonal projection expression are solved below.

$$\frac{\partial f_{BV}}{\partial \gamma_{1,n}} = [-1][\phi_{0,n}(x)] \tag{C.45}$$

$$\frac{\partial f_{BV}}{\partial \gamma_{5,n}} = [-1][\phi_{4,n}(x)] \tag{C.46}$$

$$\begin{gathered}\frac{\partial f_{BV}}{\partial \tau_{1,n}} = [a_n\alpha_n(c_{2,n})^{\alpha_n}k_n\sinh((\frac{\alpha_n F}{RT})(P_{1,n}-P_{2,n}-u_{nref}))]* \\ 1[(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n}(c_{1,n}^{s})^{\alpha_n-1}-(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n-1}(c_{1,n}^{s})^{\alpha_n}][\phi_{0,n}(x)\omega_{0,n}(R_n)]\end{gathered} \tag{C.47}$$

$$\begin{gathered}\frac{\partial f_{BV}}{\partial \tau_{20,n}} = [a_n\alpha_n(c_{2,n})^{\alpha_n}k_n\sinh((\frac{\alpha_n F}{RT})(P_{1,n}-P_{2,n}-u_{nref}))]* \\ 1[(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n}(c_{1,n}^{s})^{\alpha_n-1}-(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n-1}(c_{1,n}^{s})^{\alpha_n}][\phi_{4,n}(x)\omega_{6,n}(R_n)]\end{gathered} \tag{C.48}$$

$$\begin{gathered}\frac{\partial f_{BV}}{\partial \beta_{1,n}} = [a_n k_n(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n}(c_{1,n}^{s})^{\alpha_n}]\sinh((\frac{\alpha_n F}{RT})(P_{1,n}-P_{2,n}-u_{nref}) \\ \alpha(c_{2,n})^{(\alpha_n-1)}[\phi_{0,n}(x)]\end{gathered} \tag{C.49}$$

$$\begin{gathered}\frac{\partial f_{BV}}{\partial \beta_{5,n}} = [a_n k_n(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n}(c_{1,n}^{s})^{\alpha_n}]\sinh((\frac{\alpha_n F}{RT})(P_{1,n}-P_{2,n}-u_{nref}) \\ \alpha(c_{2,n})^{(\alpha_n-1)}[\phi_{4,n}(x)]\end{gathered} \tag{C.50}$$

$$\begin{gathered}\frac{\partial f_{BV}}{\partial \zeta_{1,n}} = [a_n k_n(c_{1,n}^{max}-c_{1,n}^{s})^{\alpha_n}(c_{1,n}^{s})^{\alpha_n}(c_{2,n})^{(\alpha_n)}] \\ 1[\cosh((\frac{\alpha_n F}{RT})(P_{1,n}-P_{2,n}-u_{nref}))[\frac{\alpha_n F}{RT}][\phi_{0,n}(x)]\end{gathered} \tag{C.51}$$

$$\frac{\partial f_{BV}}{\partial \zeta_{5,n}} = [a_n k_n (c_{1,n}^{max} - c_{1,n}^{s})^{\alpha_n} (c_{1,n}^{s})^{\alpha_n} (c_{2,n})^{(\alpha_n)}] \\ 1[\cosh((\frac{\alpha_n F}{RT})(P_{1,n} - P_{2,n} - u_{nref}))[\frac{\alpha_n F}{RT}][\phi_{4,n}(x)] \tag{C.52}$$

$$\frac{\partial f_{BV}}{\partial \delta_{1,n}} = [a_n k_n (c_{1,n}^{max} - c_{1,n}^{s})^{\alpha_n} (c_{1,n}^{s})^{\alpha_n} (c_{2,n})^{(\alpha_n)}] \\ 1[\cosh((\frac{\alpha_n F}{RT})(P_{1,n} - P_{2,n} - u_{nref}))[-\frac{\alpha_n F}{RT}][\phi_{0,n}(x)] \tag{C.53}$$

$$\frac{\partial f_{BV}}{\partial \delta_{5,n}} = [a_n k_n (c_{1,n}^{max} - c_{1,n}^{s})^{\alpha_n} (c_{1,n}^{s})^{\alpha_n} (c_{2,n})^{(\alpha_n)}] \\ 1[\cosh((\frac{\alpha_n F}{RT})(P_{1,n} - P_{2,n} - u_{nref}))[-\frac{\alpha_n F}{RT}][\phi_{4,n}(x)] \tag{C.54}$$

After all of the linearizations are completed, Galerkin projections are applied to Equation C.43 to give five algebraic equations. The process is repeated for the positive electrode.

Similar to Section C.4, there are components that enter into the A matrix in Equation 4.45 and components that enter into the G vector that accounts for the previous time step linearization.

Appendix D

Doyle Fuller Newman Model MATLAB Code

The DFN_Parameter_List_Script.m, Legendre_Polynomial_Script.m, BV_Linearization_v2.m, Ureflookupn.m, Ureflookupp.m, P2_Linearization_v2.m, and Plotting_Script_Ver_1.m MATLAB files are not included below. The parameter list along with the reference lookup tables can be found in [7].

D.1 DFN Model - Run File

```
%% Main DFN Model Code

% This code runs the model itself. It calls other scripts for the parameter
% list and plotting

%% Clearning the workspace

close all
clear all
clc

warning('off','MATLAB:nearlySingularMatrix')

tic
%% Parameter List

DFN_Parameter_List_Script

%% x discretizations

num_of_x_n_steps = 501;
num_of_x_s_steps = 501;
num_of_x_p_steps = 501;

x_n = linspace(0,L_n,num_of_x_n_steps);
x_s = linspace(0,L_s,num_of_x_s_steps);
x_p = linspace(0,L_p,num_of_x_p_steps);

%% r discretizations

num_of_r_n_steps = 501;
num_of_r_p_steps = 501;

r_n = linspace(0,R_n,num_of_r_n_steps);
r_p = linspace(0,R_p,num_of_r_p_steps);

%% t discretizations

start_time = 0;
stop_time = 1000;
% sample_time = 0.1;
num_of_time_steps = 10*1000 + 1;
```

```
%t = start_time:sample_time:stop_time;
t = linspace(start_time,stop_time,num_of_time_steps);
dt = t(2)-t(1);

%% Current Input to the System

% Speed Test

i_app = -0.5*ones(1,length(t));

% Steady State
% i_app = zeros(1,length(t));

% Discharge
% i_app = (0.1)*(-2.5*ones(1,length(t)));
% i_app = (1.0)*(-2.5*ones(1,length(t)));
% i_app = (2.5)*(-2.5*ones(1,length(t)));


% Pulse Tests

% Test 1
% i_app(1:120) = 0;
% i_app(121:720) = -2.5*(2.5);
% i_app(721:1320) = 0;
% i_app(1321:1380) = +0.5*(2.5);
% i_app(1381:1440) = -0.5*(2.5);
% i_app(1441:2040) = 0;
% i_app(2041:2100) = +1.5*(2.5);
% i_app(2101:2160) = -1.5*(2.5);
% i_app(2161:2760) = 0;
% i_app(2761:2820) = +3.0*(2.5);
% i_app(2821:2880) = -3.0*(2.5);
% i_app(2881:3480) = 0;
% i_app(3481:3540) = +5.0*(2.5);
% i_app(3541:3600) = -5.0*(2.5);

% Test 2
i_app(1:120) = 0;
i_app(121:720) = -2.5*(2.5);
i_app(721:1320) = 0;

i_app(1321:1620) = +0.5*(2.5);
i_app(1621:1920) = -0.5*(2.5);
i_app(1921:2520) = 0;

i_app(2521:2820) = +1.5*(2.5);
i_app(2821:3120) = -1.5*(2.5);
i_app(3121:3720) = 0;

i_app(3721:4020) = +3.0*(2.5);
i_app(4021:4320) = -3.0*(2.5);
i_app(4321:4920) = 0;

i_app(4921:5220) = +5.0*(2.5);
i_app(5221:5520) = -5.0*(2.5);

% %i_app(1:20) = 0;
% i_app(1:300) = - 1*1.1;
% i_app(301:601) = + 1*1.1;

%% Legendre Polynomials
```

```
Legendre_Polynomial_Script

%% Initial Conditions
z(:,1) = zeros(90,1);

% Solid Concentration (Anode): z1-z20
    for i = 1:5
        for j = 1:4
            z(4*(i-1)+j,1) = trapz(x_n,trapz(r_n,c1_int_n*omega_n(j,:))*phi_n(i,:));
        end
    end

% Solid Concentration (Cathode): z21-z40
    for i = 1:5
        for j = 1:4
            z(20+4*(i-1)+j,1) = trapz(x_p,trapz(r_p,c1_int_p*omega_p(j,:))*phi_p(i,:));
        end
    end

% Solution Concentration (Anode): z41-z45
    for i = 1:5
        z(40+i,1) = trapz(x_n,c2_int_n*phi_n(i,:));
    end

% Solution Concentration (Separator): z46-z50
    for i = 1:5
        z(45+i,1) = trapz(x_s,c2_int_s*phi_s(i,:));
    end

% Solution Concentration (Cathode): z51-z55
    for i = 1:5
        z(50+i,1) = trapz(x_p,c2_int_p*phi_p(i,:));
    end

z(61,1) = 0.0266218;

%% Governing Equations

% Pre Allocate

E = zeros(90,90);
A = zeros(90,90);
B = zeros(90,1);
G = zeros(90,1);


% Solid Concentration
    %(Negative)
    % Tau_n 1 - 20, Rows 1 - 20 in State Space

    for i = 0:4:16
        E(i+1:i+3,i+1:i+3) = eye(3);
    A(i+1:i+4,i+1:i+4) = ...
     [0 9*sqrt(5)*diff1_n/R_n^2 20*diff1_n/R_n^2         29.4*sqrt(13)*diff1_n/R_n^2
      0 0                       35*sqrt(5)*diff1_n/R_n^2 16.8*sqrt(65)*diff1_n/R_n^2
      0 0                       0                        46.2*sqrt(13)*diff1_n/R_n^2
      0 3/R_n*sqrt(5/R_n)       10/R_n*sqrt(9/R_n)       21/R_n*sqrt(13/R_n)];
    end
    for i = 1:5
        A(4*i+0,80+i) = 1/(diff1_n*a_n*F);
    end

    % (Positive)
    % Tau_p 1 - 20, Rows 21 - 40 in State Space
```

```
    for i = 20:4:36
        E(i+1:i+3,i+1:i+3) = eye(3);
    A(i+1:i+4,i+1:i+4) = ...
     [0 9*sqrt(5)*diff1_p/R_p^2 20*diff1_p/R_p^2         29.4*sqrt(13)*diff1_p/R_p^2
      0 0                       35*sqrt(5)*diff1_p/R_p^2 16.8*sqrt(65)*diff1_p/R_p^2
      0 0                       0                        46.2*sqrt(13)*diff1_p/R_p^2
      0 3/R_p*sqrt(5/R_p)       10/R_p*sqrt(9/R_p)       21/R_p*sqrt(13/R_p)];
    end
    for i = 1:5
        A(4*i+20,85+i) = 1/(diff1_p*a_p*F);
    end

% Solution Concentration
    %(Negative)
    % Beta_n 1 - 5, Rows 41-45 in State Space
    E(41:43,41:43) = eye(3);
    A(41:43,41:45) = diff2_n/eps2_n*[0 0 12*sqrt(5)/L_n^2 0                 120/L_n^2;
                                     0 0 0                20*sqrt(21)/L_n^2 0;
                                     0 0 0                0                 84*sqrt(5)/L_n^2];

    A(41:43,81:83) = (1-t_plus_n)/F/eps2_n*eye(3);

    %(Separator)
    % Beta_s 1 - 5, Rows 46 - 50 in State Space
    E(46:48,46:48) = eye(3);
    A(46:48,46:50) = diff2_s/eps2_s*[0 0 12*sqrt(5)/L_s^2 0                 120/L_s^2;
                                     0 0 0                20*sqrt(21)/L_s^2 0;
                                     0 0 0                0                 84*sqrt(5)/L_s^2];

    %(Positive)
    % Beta_p 1 - 5, Rows 51 - 55 in State Space
    E(51:53,51:53) = eye(3);
    A(51:53,51:55) = diff2_p/eps2_p*[0 0 12*sqrt(5)/L_p^2 0                 120/L_p^2;
                                     0 0 0                20*sqrt(21)/L_p^2 0;
                                     0 0 0                0                 84*sqrt(5)/L_p^2];

    A(51:53,86:88) = (1-t_plus_p)/F/eps2_p*eye(3);

    % Coupled Boundary Conditions
    % BC 1
    A(44,41:45) = [0 2*sqrt(3)/L_n^(3/2) -6*sqrt(5)/L_n^(3/2) 12*sqrt(7)/L_n^(3/2) -60/L_n^(3/2)];

    % BC 2
    A(45,41:50) = [[sqrt(1/L_n)  sqrt(3/L_n) sqrt(5/L_n)  sqrt(7/L_n) sqrt(9/L_n)],...
                  -[sqrt(1/L_s) -sqrt(3/L_s) sqrt(5/L_s) -sqrt(7/L_s) sqrt(9/L_s)]];

    % BC 3
    A(49,41:50) = [diff2_n*[0 2*sqrt(3)/L_n^(3/2)  6*sqrt(5)/L_n^(3/2) 12*sqrt(7)/L_n^(3/2)
60/L_n^(3/2)], ...
                  -diff2_s*[0 2*sqrt(3)/L_s^(3/2) -6*sqrt(5)/L_s^(3/2) 12*sqrt(7)/L_s^(3/2) -60/L_s^(3/2

    % BC 4
    A(50,46:55) = [diff2_s*[0 2*sqrt(3)/L_s^(3/2)  6*sqrt(5)/L_s^(3/2) 12*sqrt(7)/L_s^(3/2)
60/L_s^(3/2)],...
                  -diff2_p*[0 2*sqrt(3)/L_p^(3/2) -6*sqrt(5)/L_p^(3/2) 12*sqrt(7)/L_p^(3/2) -60/L_p^(3/2

    % BC 5
    A(54,46:55) = [[sqrt(1/L_s)  sqrt(3/L_s) sqrt(5/L_s)  sqrt(7/L_s) sqrt(9/L_s)],...
                  -[sqrt(1/L_p) -sqrt(3/L_p) sqrt(5/L_p) -sqrt(7/L_p) sqrt(9/L_p)]];

    % BC 6
    A(55,51:55) = [0 2*sqrt(3)/L_p^(3/2) 6*sqrt(5)/L_p^(3/2) 12*sqrt(7)/L_p^(3/2) 60/L_p^(3/2)];
```

```
% Solid Potential

    %(Negative)
    % Zeta_n 1 - 5, Rows 56 - 60 in State Space
    A(56:58,56:60) = sigma_eff_n*...
        [0 0 12*sqrt(5)/L_n^2 0                 120/L_n^2;
         0 0 0                20*sqrt(21)/L_n^2 0;
         0 0 0                0                 84*sqrt(5)/L_n^2];

    A(56:58,81:83) = -eye(3);

    %(Positive)
    % Zeta_p 1 - 5, Rows 61 - 65 in State Space
    A(61:63,61:65) = sigma_eff_p*...
        [0 0 12*sqrt(5)/L_p^2 0                 120/L_p^2;
         0 0 0                20*sqrt(21)/L_p^2 0;
         0 0 0                0                 84*sqrt(5)/L_p^2];

    A(61:63,86:88) = -eye(3);

   % Decoupled Boundary Conditions

   % BC 1
   A(59,56:60) = [sqrt(1/L_n) -sqrt(3/L_n) sqrt(5/L_n) -sqrt(7/L_n) sqrt(9/L_n)];
   % BC 2
   A(60,56:60) = [0 2/L_n*sqrt(3/L_n) 6/L_n*sqrt(5/L_n) 12/L_n*sqrt(7/L_n) 20/L_n*sqrt(9/L_n)];
   % BC 3
   A(64,61:65) = [0 2/L_p*sqrt(3/L_p) -6/L_p*sqrt(5/L_p) 12/L_p*sqrt(7/L_p) -20/L_p*sqrt(9/L_p)];
   % BC 4
   A(65,61:65) = [0 2/L_p*sqrt(3/L_p)  6/L_p*sqrt(5/L_p) 12/L_p*sqrt(7/L_p)  20/L_p*sqrt(9/L_p)];

   % Input to the Battery Model
   B(65,1) = -1/Area/sigma_eff_p;

% Solution Potential (Linear Parts ONLY)
    %(Negative)
    % Delta_n 1 -5 Rows 66 - 70 in State Space
    A(66:68,66:70) = kappa_eff_n*...
        [0 0 12*sqrt(5)/L_n^2 0                 120/L_n^2;
         0 0 0                20*sqrt(21)/L_n^2 0;
         0 0 0                0                 84*sqrt(5)/L_n^2];

    A(66:68,81:83) = eye(3);

    %(Separator)
    % Delta_s 1 -5 Rows 71 - 75 in State Space

    A(71:73,71:75) = kappa_eff_s*...
        [0 0 12*sqrt(5)/L_s^2 0                 120/L_s^2;
         0 0 0                20*sqrt(21)/L_s^2 0;
         0 0 0                0                 84*sqrt(5)/L_s^2];

    %(Separator)
    % Delta_p 1 -5 Rows 76 - 80 in State Space

    A(76:78,76:80) = kappa_eff_p*...
        [0 0 12*sqrt(5)/L_p^2 0                 120/L_p^2;
         0 0 0                20*sqrt(21)/L_p^2 0;
         0 0 0                0                 84*sqrt(5)/L_p^2];

     A(76:78,86:88) = eye(3);

    % Coupled Boundary Conditions
```

```
    % BC 1
    A(69,66:70) = [0 2/L_n*sqrt(3/L_n) -6/L_n*sqrt(5/L_n) 12/L_n*sqrt(7/L_n) -20/L_n*sqrt(9/L_n)];
    % BC 2
    A(70,66:75) = [[sqrt(1/L_n)  sqrt(3/L_n) sqrt(5/L_n)  sqrt(7/L_n) sqrt(9/L_n)],...
                  -[sqrt(1/L_s) -sqrt(3/L_s) sqrt(5/L_s) -sqrt(7/L_s) sqrt(9/L_s)]];
    % BC 3
    A(74,66:75) = [kappa_eff_n*...
        [0 2*sqrt(3)/L_n^(3/2)  6*sqrt(5)/L_n^(3/2) 12*sqrt(7)/L_n^(3/2)  60/L_n^(3/2)],...
        -kappa_eff_s*...
        [0 2*sqrt(3)/L_s^(3/2) -6*sqrt(5)/L_s^(3/2) 12*sqrt(7)/L_s^(3/2) -60/L_s^(3/2)]];
    % BC 4
    A(75,71:80) = [kappa_eff_s*...
        [0 2*sqrt(3)/L_s^(3/2)  6*sqrt(5)/L_s^(3/2) 12*sqrt(7)/L_s^(3/2)  60/L_s^(3/2)],...
        -kappa_eff_p*...
        [0 2*sqrt(3)/L_p^(3/2) -6*sqrt(5)/L_p^(3/2) 12*sqrt(7)/L_p^(3/2) -60/L_p^(3/2)]];
    % BC 5
    A(79,71:80) = [[sqrt(1/L_s)  sqrt(3/L_s) sqrt(5/L_s)  sqrt(7/L_s) sqrt(9/L_s)],...
                  -[sqrt(1/L_p) -sqrt(3/L_p) sqrt(5/L_p) -sqrt(7/L_p) sqrt(9/L_p)]];
    % BC 6
    A(80,76:80) = [0 2*sqrt(3)/L_p^(3/2) 6*sqrt(5)/L_p^(3/2) 12*sqrt(7)/L_p^(3/2) 60/L_p^(3/2)];

% Butler Volmer Equation (Linear Parts Only)

%% Loop to Simulate

% BV_RUN_TIME = zeros(1,(length(t)-1));

for k = 1:(length(t)-1)
    % Quasi-Linearize BV about previous time step
    %tic
    [A(81:85,:),A(86:90,:),G(81:85,1),G(86:90,1)] = BV_Linearization_v2(z(:,k));
    %[A(81:85,:),A(86:90,:),G(81:85,1),G(86:90,1)] = BV_Linearization_v3(z(:,k),x_n,x_s,x_p,r_n,r_p,phi_
    %BV_RUN_TIME(k) = toc;
    % Quasi-Linearize P2 about previous time step
    [A_P2,G_P2] = P2_Linearization_v2(z(:,k));
        A(66:80,:) = A(66:80,:) + A_P2;
        G(66:80,1) = G(66:80,1) + G_P2;
    % Solve next time step
    z(:,k+1) = (E/dt-A)^-1*(E/dt*z(:,k) + G + B*i_app(k));

    if k==1; A_P20 = A_P2(:,41:55); G_P20 = G_P2; c20 = z(41:55); end
end


%time_for_calcs = toc

%% Post Processing

Post_Processing_Script

time_to_run = toc

%% Plotting Script

% Plotting_Script_Ver_1

% DFN_Plotting_Script_Ver_1
```

D.2 DFN Model - Post-Processing

```
%% Post Processing Script
```

```
%% Code
% Pre Allocating

% c1_n = zeros(length(x_n),length(r_n),length(t));
% c1_p = zeros(length(x_p),length(r_p),length(t));

% negative
phi1_n_omega1_n = ( phi_n(1,:)'*omega_n(1,:) );
phi1_n_omega2_n = ( phi_n(1,:)'*omega_n(2,:) );
phi1_n_omega3_n = ( phi_n(1,:)'*omega_n(3,:) );
phi1_n_omega4_n = ( phi_n(1,:)'*omega_n(4,:) );

phi2_n_omega1_n = ( phi_n(2,:)'*omega_n(1,:) );
phi2_n_omega2_n = ( phi_n(2,:)'*omega_n(2,:) );
phi2_n_omega3_n = ( phi_n(2,:)'*omega_n(3,:) );
phi2_n_omega4_n = ( phi_n(2,:)'*omega_n(4,:) );

phi3_n_omega1_n = ( phi_n(3,:)'*omega_n(1,:) );
phi3_n_omega2_n = ( phi_n(3,:)'*omega_n(2,:) );
phi3_n_omega3_n = ( phi_n(3,:)'*omega_n(3,:) );
phi3_n_omega4_n = ( phi_n(3,:)'*omega_n(4,:) );

phi4_n_omega1_n = ( phi_n(4,:)'*omega_n(1,:) );
phi4_n_omega2_n = ( phi_n(4,:)'*omega_n(2,:) );
phi4_n_omega3_n = ( phi_n(4,:)'*omega_n(3,:) );
phi4_n_omega4_n = ( phi_n(4,:)'*omega_n(4,:) );

phi5_n_omega1_n = ( phi_n(5,:)'*omega_n(1,:) );
phi5_n_omega2_n = ( phi_n(5,:)'*omega_n(2,:) );
phi5_n_omega3_n = ( phi_n(5,:)'*omega_n(3,:) );
phi5_n_omega4_n = ( phi_n(5,:)'*omega_n(4,:) );

% pos
phi1_p_omega1_p = ( phi_p(1,:)'*omega_p(1,:) );
phi1_p_omega2_p = ( phi_p(1,:)'*omega_p(2,:) );
phi1_p_omega3_p = ( phi_p(1,:)'*omega_p(3,:) );
phi1_p_omega4_p = ( phi_p(1,:)'*omega_p(4,:) );

phi2_p_omega1_p = ( phi_p(2,:)'*omega_p(1,:) );
phi2_p_omega2_p = ( phi_p(2,:)'*omega_p(2,:) );
phi2_p_omega3_p = ( phi_p(2,:)'*omega_p(3,:) );
phi2_p_omega4_p = ( phi_p(2,:)'*omega_p(4,:) );

phi3_p_omega1_p = ( phi_p(3,:)'*omega_p(1,:) );
phi3_p_omega2_p = ( phi_p(3,:)'*omega_p(2,:) );
phi3_p_omega3_p = ( phi_p(3,:)'*omega_p(3,:) );
phi3_p_omega4_p = ( phi_p(3,:)'*omega_p(4,:) );

phi4_p_omega1_p = ( phi_p(4,:)'*omega_p(1,:) );
phi4_p_omega2_p = ( phi_p(4,:)'*omega_p(2,:) );
phi4_p_omega3_p = ( phi_p(4,:)'*omega_p(3,:) );
phi4_p_omega4_p = ( phi_p(4,:)'*omega_p(4,:) );

phi5_p_omega1_p = ( phi_p(5,:)'*omega_p(1,:) );
phi5_p_omega2_p = ( phi_p(5,:)'*omega_p(2,:) );
phi5_p_omega3_p = ( phi_p(5,:)'*omega_p(3,:) );
phi5_p_omega4_p = ( phi_p(5,:)'*omega_p(4,:) );


% for t_index = 1:length(t)
%     % neg
%     c1_n(:,:,t_index) = (...
```

```
%         (phi1_n_omega1_n).*z( 1,t_index) + (phi1_n_omega2_n).*z( 2,t_index) + (phi1_n_omega3_n).*z( 3,
%        +(phi2_n_omega1_n).*z( 5,t_index) + (phi2_n_omega2_n).*z( 6,t_index) + (phi2_n_omega3_n).*z( 7,
%        +(phi3_n_omega1_n).*z( 9,t_index) + (phi3_n_omega2_n).*z(10,t_index) + (phi3_n_omega3_n).*z(11,
%        +(phi4_n_omega1_n).*z(13,t_index) + (phi4_n_omega2_n).*z(14,t_index) + (phi4_n_omega3_n).*z(15,
%        +(phi5_n_omega1_n).*z(17,t_index) + (phi5_n_omega2_n).*z(18,t_index) + (phi5_n_omega3_n).*z(19,
%     );
%     % pos
%     c1_p(:,:,t_index) = (...
%         (phi1_p_omega1_p).*z(21,t_index) + (phi1_p_omega2_p).*z(22,t_index) + (phi1_p_omega3_p).*z(23,
%        +(phi2_p_omega1_p).*z(25,t_index) + (phi2_p_omega2_p).*z(26,t_index) + (phi2_p_omega3_p).*z(27,
%        +(phi3_p_omega1_p).*z(29,t_index) + (phi3_p_omega2_p).*z(30,t_index) + (phi3_p_omega3_p).*z(31,
%        +(phi4_p_omega1_p).*z(33,t_index) + (phi4_p_omega2_p).*z(34,t_index) + (phi4_p_omega3_p).*z(35,
%        +(phi5_p_omega1_p).*z(37,t_index) + (phi5_p_omega2_p).*z(38,t_index) + (phi5_p_omega3_p).*z(39,
%     );
% end

c1_s_n = (...
    ( z( 1,:)*omega_n(1,length(omega_n)) + z( 2,:)*omega_n(2,length(omega_n)) + z( 3,:)*omega_n(3,length
   +( z( 5,:)*omega_n(1,length(omega_n)) + z( 6,:)*omega_n(2,length(omega_n)) + z( 7,:)*omega_n(3,length
   +( z( 9,:)*omega_n(1,length(omega_n)) + z(10,:)*omega_n(2,length(omega_n)) + z(11,:)*omega_n(3,length
   +( z(13,:)*omega_n(1,length(omega_n)) + z(14,:)*omega_n(2,length(omega_n)) + z(15,:)*omega_n(3,length
   +( z(17,:)*omega_n(1,length(omega_n)) + z(18,:)*omega_n(2,length(omega_n)) + z(19,:)*omega_n(3,length
    )';

c1_s_p = (...
    ( z(21,:)*omega_p(1,length(omega_p)) + z(22,:)*omega_p(2,length(omega_p)) + z(23,:)*omega_p(3,length
   +( z(25,:)*omega_p(1,length(omega_p)) + z(26,:)*omega_p(2,length(omega_p)) + z(27,:)*omega_p(3,length
   +( z(29,:)*omega_p(1,length(omega_p)) + z(30,:)*omega_p(2,length(omega_p)) + z(31,:)*omega_p(3,length
   +( z(33,:)*omega_p(1,length(omega_p)) + z(34,:)*omega_p(2,length(omega_p)) + z(35,:)*omega_p(3,length
   +( z(37,:)*omega_p(1,length(omega_p)) + z(38,:)*omega_p(2,length(omega_p)) + z(39,:)*omega_p(3,length
    )';

c2_n = ((z(41,:)')*(phi_n(1,:)) + (z(42,:)')*(phi_n(2,:)) + (z(43,:)')*(phi_n(3,:)) + (z(44,:)')*(phi_n(
c2_s = ((z(46,:)')*(phi_s(1,:)) + (z(47,:)')*(phi_s(2,:)) + (z(48,:)')*(phi_s(3,:)) + (z(49,:)')*(phi_s(
c2_p = ((z(51,:)')*(phi_p(1,:)) + (z(52,:)')*(phi_p(2,:)) + (z(53,:)')*(phi_p(3,:)) + (z(54,:)')*(phi_p(

P1_n = ((z(56,:)')*(phi_n(1,:)) + (z(57,:)')*(phi_n(2,:)) + (z(58,:)')*(phi_n(3,:)) + (z(59,:)')*(phi_n(
P1_p = ((z(61,:)')*(phi_p(1,:)) + (z(62,:)')*(phi_p(2,:)) + (z(63,:)')*(phi_p(3,:)) + (z(64,:)')*(phi_p(

P2_n = ((z(66,:)')*(phi_n(1,:)) + (z(67,:)')*(phi_n(2,:)) + (z(68,:)')*(phi_n(3,:)) + (z(69,:)')*(phi_n(
P2_s = ((z(71,:)')*(phi_s(1,:)) + (z(72,:)')*(phi_s(2,:)) + (z(73,:)')*(phi_s(3,:)) + (z(74,:)')*(phi_s(
P2_p = ((z(76,:)')*(phi_p(1,:)) + (z(77,:)')*(phi_p(2,:)) + (z(78,:)')*(phi_p(3,:)) + (z(79,:)')*(phi_p(

J_n =  ((z(81,:)')*(phi_n(1,:)) + (z(82,:)')*(phi_n(2,:)) + (z(83,:)')*(phi_n(3,:)) + (z(84,:)')*(phi_n(
J_p =  ((z(86,:)')*(phi_p(1,:)) + (z(87,:)')*(phi_p(2,:)) + (z(88,:)')*(phi_p(3,:)) + (z(89,:)')*(phi_p(
```